Optimal Sampling-Based Trajectory Planning For Autonomous Systems in Urban Environments

Mitchell Lichocki

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Applied Science

(Electrical and Computer Engineering) at

Concordia University

Montreal, Quebec, Canada

October 2020

CONCORDIA UNIVERSITY

School of Graduate Studies

This is to certify that the thesis prepared

By: Mitchell Lichocki

Entitled: Optimal Sampling-Based Trajectory Planning For Autonomous Systems in Urban Environments

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science (Electrical and Computer Engineering)

complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final Examining Committee:

_____ Chair

Dr. Y. R. Shayan

_____ External Examiner

Dr. W. Lucia

_____ Examiner

Dr. R. Selmic

_____ Supervisor

Dr. L. Rodrigues

Approved by _____

Dr. Y. R. Shayan, Chair

Department of Electrical and Computer Engineering

_____ 2020 _____

Dr. Amir Asif, Dean

Faculty of Engineering and Computer Science

ABSTRACT

Optimal Sampling-Based Trajectory Planning For
Autonomous Systems in Urban Environments

Mitchell Lichocki

Motivated by autonomous aerial vehicles, this thesis provides a methodology for optimal trajectory planning of affine systems in non-convex environments. The resulting approximation of the optimal trajectory can then be provided to a flight controller as a reference trajectory, which compares the actual state of the system with the reference trajectory and performs the necessary control input corrections. More specifically, a modified trajectory planner inspired by Kinodynamic RRT* is presented to solve optimal control problems for input constrained affine systems with non-convex state spaces. As a result, if a solution is obtained then the solution is guaranteed to verify the state and control input constraints of the problem. Additionally, a randomized sampler function is proposed for Kinodynamic RRT* using a Gaussian distribution across the system's state space. When the distribution is adequately sized lower cost approximate solutions of the optimal trajectory problem is obtained in less computation time when compared with other methods in the literature. The results are successfully applied to optimal control problems for an affine double integrator with drift that is subject to a maximum control input magnitude in non-convex environments.

ACKNOWLEDGEMENTS

Firstly, I would like to thank my family for the unending and limitless love and support they have provided me throughout my studies. To my parents, who continue to encourage and inspire me to be the best I can every day. To my girlfriend, Marieve you have supported me beyond what I could have ever imagined. I could never thank you all enough nor express just how valuable you are to all that I have accomplished.

I also owe an abundance of gratitude and acknowledgement to the never-ending hard work of my supervisor Dr. Rodrigues. His guidance, advice, and professionalism has inspired me to push the boundaries of my research and to elevate the quality of my work to the highest levels. I would also like to thank Marinvent Corporation and MITACS for their generosity and willingness to fund my research.

To my good friend Paul, thank you for the fascinating coffee break talks and for always teaching me something new about the aerospace industry. You have provided me with valuable insights for my research and I thank you for that. Last but not least, I would like to thank Bruno, Weihong, and Steven who have been nothing short of the greatest colleagues I could have asked for. The three of you were always willing to give my research a fresh perspective and always motivated me in my work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Figure 1: Technologies of self-driving cars [1].

Advancements in computer capabilities, manufacturing, and materials have resulted in powerful technologies becoming increasingly used in the civilian market throughout the past few decades. For example, about 130 years ago, when the primary mode of transportation was horseback, the car was invented - a technology that few could afford. Today, many of the largest car manufacturers and technology companies around the globe are testing and producing semi-autonomous and fully autonomous cars [4]. These vehicles use complex systems to navigate urban roadways and avoid accidents by predicting the motion of nearby vehicles and pedestrians. However, despite the convenience and benefits that many of these technologies present, our growing dependence on them has also led to the emergence of other problems.

Today, many urban areas suffer from road congestion due to the increasing presence of ground vehicles. One study found that, of the 25 most congested cities across the globe, the average driver spends about 100 hours annually in congestion with some cities exceeding 200 hours [5] annually. The growth of e-commerce in particular, which has seen a year-over-year growth of 15% between 2018 and 2019 [6], has had significant impact on road congestion due to its dependence on large ground-based distribution networks.

Figure 2: Future aerial cargo delivery vehicles [2].

One effort to mitigate the issue of urban road congestion is the use of urban aerial vehicles. For example, ride-sharing mogul Uber is collaborating with some of the world's leading aerospace manufacturers to achieve their Uber Elevate objective of developing urban air taxis [7]. Additionally, e-commerce giant Amazon has been developing a drone delivery system called Prime Air [8] for fast on-time delivery of goods. However, prior to the COVID-19 pandemic, the aerospace sector was already suffering from pilot shortages, which Airbus estimated as requiring as many as 560,000 new pilots by 2035 [9]. The unprecedented reduction of global air travel brought on by COVID-19 has likely alleviated this demand temporarily, but it is difficult to estimate by how much and for how long. For example, IATA initially estimated that global air passenger traffic would return to pre-COVID-19 numbers by 2023, but within months of announcing this estimate it was extended to 2024 [10]. Contrary to this prediction, some speculate that, given the extent of which airlines have placed pilots on furlough and have encouraged early pilot retirements, and the closure and reduced capacity of pilot training centers, the forecast pilot shortages may occur sooner rather than later [11]. Although the predictions of how quickly the demand for pilots will return to pre-COVID-19 estimates may vary, the problem of pilot shortages remains a significant concern of the future. In addition to the impending pilot shortage, autonomous aerial vehicles may significantly reduce aircraft accidents as well as operating costs. For these reasons, significant efforts are being applied toward the development of fully autonomous aircraft.

Autonomy is not entirely new in aerospace. For example, the first autopilot system was demonstrated for nearly the full duration of a flight in 1930 [12]. This system is designed as follows. A flight management system (FMS) stores all of the route information. This information originates from a sequence of waypoints entered by the pilot, which are pre-defined points in the airspace. Given a specific cost index (a ratio between the unit cost of time and the unit cost of fuel), the route is then calculated by the FMS such that the flight

Figure 3: Future urban air mobility networks [3].

path minimizes the direct operating costs and passes through the waypoints in the order that they were entered. This information is then made accessible to the autopilot system, which controls the engines and flight control surfaces to keep the aircraft flying along the route of the FMS. Although this implementation has had significant success in long-haul flights at altitudes of up to 45,000 feet, it is highly impractical for autonomous urban mobility. The first reason for this is that a planner (typically a pilot) is required to plan the sequence of waypoints that the vehicle will fly through. Second, the route is limited by the network of waypoints. Third, the FMS assumes the absence of obstacles in the airspace.

When considering the development of large-scale autonomous urban air mobility networks, manual route planning for each vehicle is a daunting task - especially when considering that the future could see hundreds of vehicles in a concentrated urban airspace at any given instant. Furthermore, although the density of commercial airports for the current civil aviation sector is low enough to support the "roadway" structure created by the waypoint system, the density of these airports in an urban environment may be significantly higher. For this reason, the waypoint flight planning structure common in commercial aviation is impractical for urban air mobility. Finally, urban obstacles such as topography and buildings must be considered by the flight planner, since it would be impractical for vehicles flying short ground distances to ascend to altitudes high enough to avoid such obstacles. Therefore, a modernized FMS system that is capable of fully autonomous trajectory planning is a critical technology that is required to ensure the safe operation of urban aerial vehicles.

1.2 Literature Survey

Path planning is described as the search for a geometric path through a space such that the path begins at an initial point in the space and ends at a final point, and avoids all obstacles. If the resulting path is required to be a function of time, then the problem is described as trajectory planning. The problem is known as kinodynamic trajectory planning when the trajectory is required to satisfy the kinematic or the dynamic constraints of the system. If the resulting kinodynamic trajectory is required to minimize a given cost function, then the problem is described as optimal kinodynamic trajectory planning. Optimal kinodynamic trajectory planning is an important domain of motion planning that meets the needs of aerospace route planning for a number of reasons. First, the time parametrization of the trajectory provides insight as to where each vehicle will be in space at any given time instant. Also, the consideration of the kinematic or the dynamic constraints of the vehicle results in trajectories that are feasible. Finally, optimization is a natural component of vehicle operation. For example, operators of emergency vehicles may prefer to minimize flight time whereas operators of cargo delivery vehicles may prefer to minimize fuel consumption and operators of passenger vehicles may prefer to minimize a cost trading off fuel consumption and time.

Traditionally, optimal control theory [13] has demonstrated enormous success for solving optimal trajectory planning problems. However, it becomes increasingly difficult to obtain a solution when the system is subject to state and/or control input constraints. This is particularly true when the constraints cause the problem to be non-convex. As a consequence of this many alternative approaches have been developed to solve or to find approximate solutions to these problems. An in-depth survey of some of the most successful motion planning algorithms is presented in [14]. Gradient-based approaches such as artificial potential fields have been applied to optimal trajectory planning problems [15–17]. However, these planners may fail when local minima are present, which are points in the space where the gradient is zero but is not the desired final point. Genetic algorithms have also been used [18–21], which is an evolutionary algorithm inspired by natural selection. However, these algorithms generally require long computation times. Another approach involves first representing the path generated by a discretized path-planner such as A* [22] or D* Lite [23] by a series of waypoints. Then, Bezier curves are used to generate trajectories through these waypoints [24–27]. However, the Bezier curves generally do not depend explicitly on the differential equations of the system and therefore the resulting trajectory cannot always be proven to be feasible. Additionally, the result obtained is sub-optimal due to the discretization of the space. Particle swarm optimization, an iterative technique of solving optimization problems, has also been used in [28–32]. However, these solutions often attempt to optimize the entire trajectory at each iteration and therefore also demonstrate slow convergence. Neural networks have also been used to solve optimal trajectory planning problems in [33–37]. However, these systems

require lengthy training and it is difficult to prove that the resulting trajectory is feasible and optimal.

Another class of motion planners that have demonstrated success are sampling-based planners. These planners iteratively sample the system's state space and connect the samples to build a graph. The vertices of the graph describe states in the system's state space, and the edges describe trajectories between states. Edges between two states are generated by a *steering* function. These trajectories are then evaluated for collisions with obstacles in the system's state space by a *collision-checker* function, and any trajectories found to result in a collision are discarded. Following the completion of a pre-determined number of iterations, an attempt is made to connect the final state with the graph. If the final state can be connected, then the sequence of edges from the initial state to the final state through the graph describes the trajectory. If it cannot be connected, then the trajectory planner is considered to have failed.

The success of sampling-based trajectory planners is largely attributed to their ability to converge to solutions quickly, to accommodate systems with large numbers of degrees of freedom, and to perform well for environments with many obstacles. One of the first and most highly regarded sampling-based trajectory planners is probabilistic roadmaps (PRM) [38]. PRM is a multiple-query trajectory planner, which means that it can be used to obtain multiple trajectories from various initial states to various final states. This is accomplished by the development of a graph through the system's state space without the consideration of an initial and a final state. As a consequence, multiple trajectories can be obtained from the graph by selecting the initial and final states as vertices of the graph and applying Dijkstra's shortest path algorithm [39] to find the minimum-cost trajectory. PRM has demonstrated success in applications such as warehouse inventory management that use autonomous ground robots to transport goods. Despite the success of PRM, there was a need for single-query trajectory planners that could address more complex systems and quickly plan optimal trajectories between two distinct states. This led to the development of another highly regarded sampling-based trajectory planner called rapidly-exploring random tree (RRT) [40]. In its original design, RRT is a sub-optimal trajectory planner [41]. However, in [42] it is extended to the asymptotically optimal RRT*, which guarantees that, if an optimal solution exists to the motion planning problem, then the probability of finding such a solution approaches one as the number of iterations approaches infinity. This is accomplished via an optimal steering function and additional optimization steps. In [43] Kinodynamic RRT* is presented to approximate the solution of optimal kinodynamic trajectory planning problems. This is accomplished by the development of a steering function that produces trajectories that satisfy the system's kinematic or dynamic constraints, and minimizes a given objective function. Despite the success of Kinodynamic RRT*, there are two main issues in the open literature. First, a steering function is required to generate optimal trajectories between two arbitrary states that satisfy the system's differential equations. Second, although the computation time typically required to obtain an approximate solution of the optimal control problem

is generally less than many other methods, it is still insufficient for many real-time systems.

One of the benefits of RRT* is that it does not require that the steering function consider obstacles in the environment. Rather, trajectories generated by this function are evaluated by a collision-checker function. This allows to significantly simplify the design of the steering function, for which the solution can often be obtained directly using optimal control theory. Furthermore, a steering function that applies optimal control theory not only produces a reference state trajectory, but a reference control input trajectory as well. Despite this convenience, only a small number of contributions in the open literature have applied optimal control theory to the RRT* steering function. The minimization of a finite-horizon linear-quadratic regulator (LQR) problem is addressed for affine systems in [44]. The minimization of an infinite-horizon LQR problem for nonlinear systems is presented in [45]. The minimization of a cost trading off control effort and time has also been addressed. In [46,47] the steering function compares the zero-input trajectory with the desired terminal state, and in [48] the steering function obtains the trajectory directly from optimal control theory. These solutions assume that the final time is fixed, and for problems where the final time is free a numerical solver is used to evaluate the trajectory that minimizes the cost function. In most of the open literature the control input constraints of the system are seldom addressed. In [44, 45] the system's control input constraints are discussed, but not addressed. In [46,48] the system is assumed to have unconstrained control inputs. In [47] the control input weighting matrix of the objective function is selected for particular problems such that the control input constraints are rarely violated. Therefore, there does not appear to exist a Kinodynamic RRT* trajectory planner that guarantees that the approximate solution satisfies the control input constraints of the system.

Many contributions have also been made toward reducing the convergence time of RRT and RRT*. Goal-directed sampling is proposed in [49] to encourage random state space sampling closer to the final state. Artificial potential fields have been applied to bias the random sampling further from obstacles and closer the final state [50, 51]. A measure of quality is applied to Voronoi regions in the system's state space in [52] to favour random sampling in regions of higher quality. Similarly, visibility regions have been used to bias the random sampling in [53]. Pre-planed discretized paths obtained from planners such as A* have also been used to bias the sampling around the discretized path in [54, 55]. Bounding regions have been applied to restrict the search space in [56]. Additionally, multi-tree planners have been proposed in [57], which execute multiple RRT algorithms in parallel and connect them when possible.

A notable trajectory planner worth mentioning is inspired by sampling-based motion planners and is called LQR-Trees [58]. This planner obtains a nonlinear feedback control policy for smooth nonlinear systems. The edges of the tree are described by optimal trajectory segments that minimize a finite-horizon LQR cost function. A Lyapunov function based on an infinite-horizon LQR cost function and a sum-of-squares

method are used to generate planning "funnels", which describe a broad set of initial conditions that are capable of reaching a smaller set of goal states. As a consequence of this, a basin of attraction can be approximated around the tree using semi-definite programming. Rather than planning optimal trajectories from a given initial state to a given final state, the LQR-Trees planner simply attempts to probabilistically cover the entire state space with the basin of attraction, where the term "probabilitically" implies that a "high enough" percentage of the state space can reach the goal state. Despite the success of this planner, there are a number of drawbacks when compared with single-query trajectory planners such as RRT*. The first, and arguably the most significant, is that LQR-Trees cannot guarantee optimality of a trajectory from a given initial state to a given final state. Second, no discussion is made with respect to obstacles in the system's state space. Finally, the system is assumed to have unconstrained control inputs.

Although mathematical models can sometimes closely represent the true kinematics or dynamics of a system, they are never perfect. For example, many aircraft models are simplified with the assumption of level flight and drag is often simplified to be a linear function of velocity. However, even if the perfect model were to be used, there are often various disturbances that cannot be accounted for. For example, some of these disturbances may include wind gusts, changes in temperature, and changes in air pressure. For this reason, trajectory planners are often responsible for computing a reference trajectory. This reference trajectory can then be provided to a flight controller, which compares the current state of the system with the reference state and performs a control input correction. An example of the implementation of such a system is provided in Figure 4.

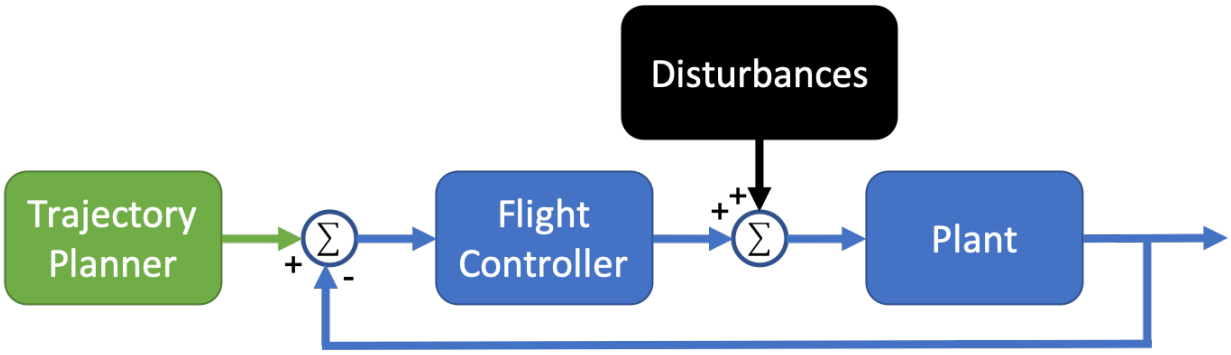


Figure 4: Trajectory planning block diagram.

1.3 Contributions

The work presented in this thesis provides a methodology to plan optimal trajectories for affine systems with non-convex state spaces and convex control input spaces. This thesis has the following contributions:

- In Chapter 2 Kinodyamic RRT* is used to approximate the solutions of optimal trajectory planning problems for a double integrator with drift such that the optimal trajectory minimizes a cost trading off control effort and time. Similar work has been presented in [47, 48]. However, these results assume either a fixed final time or require the use of numerical optimization to obtain the optimal final time for problems with free final time. By contrast, the work of [59] presents an approach to solve optimal control problems for general affine systems with unconstrained state and control input spaces where the final time of the trajectory is assumed to be free. This is an important result since the final time of trajectories in sampling-based kinodynamic trajectory planners is often a free variable. Furthermore, the task of computing trajectories is typically the most time-consuming step of each iteration and reducing the use of numerical solvers can improve the convergence speed of the algorithm. The work of Chapter 2 uses the technique of [59] together with Kinodynamic RRT* to solve a similar problem to that discussed in [47, 48] and a comparison of the computation time required by both methods is made. The comparison is made using MATLAB where the roots in [59] are obtained using the *roots* function and the numerical minimization of the cost function is obtained using *fmincon* with a bounded time variable between 0 and infinity. The results from $10,000$ randomized simulations for a double integrator with drift show that for this particular example the approach in [59] requires on average only 2.2% the computation time of the numerical minimization approach.

- A modified Kinodynamic RRT* trajectory planner is proposed to approximate the solutions of optimal trajectory planning problems with non-convex state constraints and convex control input constraints in Chapter 3. Traditionally, Kinodynamic RRT* assumes unconstrained control inputs, which is not representative of many systems. The proposed planner produces trajectories such that the input constraints are guaranteed to be satisfied. This is accomplished by using unconstrained optimal control theory to obtain analytic solutions of the optimal control input in terms of the optimal final time. Two RRT* functions are then augmented. The augmented collision-checker function rejects optimal control input trajectories that do not satisfy the control input constraints of the system. The local trajectory function is augmented such that a new vertex of the tree is established along the optimal state trajectory to a given state and the optimal control input to the new vertex satisfies the control input constraints of the system. As a result, the methodology of Kindodynamic RRT* is maintained, but the modifications

allow to find approximate solutions to input constrained optimal control problems. The proposed planner is demonstrated for an optimal control problem of a double integrator with drift where the magnitude of the control input is upper bounded, which is a problem that cannot be solved using the traditional Kinodynamic RRT* algorithm.

- A directed sampling technique for Kinodynamic RRT* is proposed in Chapter 4 using a Gaussian distribution across the system's state space to generate random states rather than a uniform distribution. As a result of this, the randomly generated states can be concentrated in a region where the optimal solution is likely to exist. The proposed sampling method is demonstrated for an optimal trajectory planning problem of an input constrained double integrator with drift. The results show that when the proposed Gaussian distributed sampling method is used, the Kinodynamic RRT* algorithm obtains lower cost approximate solutions of the optimal trajectory planning problem in less iterations and less computation time than other methods.

1.4 Thesis Structure

A review of optimal control theory and kinodynamic RRT* is presented in Chapter 2. Optimal control theory is then used with Kinodynamic RRT* to approximate the solution of optimal trajectory planning problems of a double integrator with drift such that the optimal trajectory minimizes a cost trading off control effort and time. Various references in the literature either use numerical optimization to compute the optimal final time of a trajectory, or use a linear change of coordinates to transform the system into a linear system. By contrast, in [59] the optimal final time of a trajectory is obtained by evaluating the roots of a polynomial. This approach is used with Kinodynamic RRT* and compared with similar work in the literature. Simulation results show that, for the double integrator with drift, the technique of [59] calculates the optimal final time of trajectories in just 2.2% the computation time when compared with similar work in the literature. Furthermore, using the results of [59] it is shown that a linear change of coordinates is not always effective to solve the optimal trajectory planning problem. Chapter 3 presents a modified Kinodynamic RRT* trajectory planner for control input constrained systems. Two RRT* functions are modified such that all edges of the RRT* tree are described by trajectories that are guaranteed to satisfy the control input constraints of the system. Simulation results are presented for the minimization of a cost trading off control effort and time for a double integrator with drift that is constrained by a maximum control input magnitude. Optimal control theory is used to obtain an analytical expression for the unconstrained optimal control input of the system. The analytical expression of the optimal control input allows one to evaluate an initial time interval of the optimal trajectory that satisfies the control input constraints. The results show that the traditional

Kinodynamic RRT* delivers trajectories that violate the control input constraints of the system and that the modified Kinodynamic RRT* delivers trajectories that satisfy these constraints. Chapter 4 presents a randomized state space sampling approach that uses a Gaussian distribution to iteratively generate random states in a region where an optimal solution is likely to exist. As a consequence, a solution may be obtained in less iterations, and therefore in less computation time when compared with other sampling methods in the literature. Simulation results and a comparison with previous work in the open literature are also provided. Conclusions and a brief overview of future work are presented in Chapter 5.

1.5 Publications

The work of Chapter 4 was published for two-dimensional problems in [60]:

M. Lichocki and L. Rodrigues, "A Gaussian-biased heuristic for stochastic sampling-based 2D trajectory planning algorithms," in *2020 European Control Conference (ECC)*. Saint Petersburg, Russia: IEEE, 2020, pp. 1949-1954, DOI 10.23919/ECC51009.2020.9143947.

Chapter 2

Optimal Control and Rapidly-Exploring Random Trees

This chapter presents a review of optimal control theory followed by a review of Kinodynamic RRT*. Optimal control theory is then used with Kinodynamic RRT* to approximate the solution of a trajectory planning problem of a double integrator with drift and non-convex state space such that the optimal trajectory minimizes a cost trading off control effort and time. Similar work has already been presented in references [47, 48]. Optimal control theory is used to design an RRT* steering function, which solves the unconstrained optimal control problem to steer the system between two states. In [47, 48] the steering function is designed with the assumption that the final time of the trajectory is fixed. The optimal final time is then computed using numerical optimization to search for the final time that minimizes the cost. The steering function designed in this section is taken from the result of [59], which solves unconstrained optimal control problems of general affine systems with the assumption that the final time is free. Rather than searching for the optimal final time by minimizing the expression of the cost using numerical optimization, a set of candidate optimal final times are obtained as the positive real roots of the Hamiltonian. Then, using an analytical expression for the cost, the optimal final time is obtained as the candidate that minimizes the cost. Simulation results show that, for a double integrator with drift, the method to obtain the optimal final time presented in [59] requires only 2.2% the computation time when compared with the method used in [47, 48]. As a consequence of this, when the steering function designed in this chapter is used, the Kinodynamic RRT* trajectory planner requires less computation time to complete a given number of iterations when compared with the steering function designed in [47, 48].

2.1 Review of Optimal Control Theory

The system's state is described by the vector $\boldsymbol{x} \in \mathbb{R}^n$ and its control input by the vector $\boldsymbol{u} \in \mathbb{R}^m$, where $n, m \in \mathbb{N}_{>0}$ and where $\mathbb{N}_{>0}$ is the set of positive natural numbers. Also, the system's state space is denoted by $\boldsymbol{\mathcal{X}}$ and its control input space is denoted by $\boldsymbol{\mathcal{U}}$. The system's *state transition equation* is described by

$$\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}). \tag{1}$$

A control input trajectory is denoted by $\boldsymbol{u}(t)$, which produces a state trajectory $\boldsymbol{x}(t)$. Without loss of generality, the trajectory is assumed to occur for $t \in [0, t_f]$, where t_f is known as the final time of the trajectory. Given $\boldsymbol{x}(t)$, $\boldsymbol{u}(t)$, and t_f, the *cost* of the trajectory is

$$C\left(\boldsymbol{x}\left(t\right), \boldsymbol{u}\left(t\right), t_f\right) = \phi\left(t_f, \boldsymbol{x}\left(t_f\right)\right) + \int_0^{t_f} L\left(\tau, \boldsymbol{x}\left(\tau\right), \boldsymbol{u}\left(\tau\right)\right) d\tau, \tag{2}$$

where $\phi(t_f, \boldsymbol{x}(t_f))$ is the terminal state cost. Given an initial state $\boldsymbol{x}_0$ and a final state $\boldsymbol{x}_f$, the objective of optimal control theory is to obtain the control input trajectory $\boldsymbol{u}^*(t),\ \forall\ t \in [0, t_f]$ such that $\boldsymbol{x}(0) = \boldsymbol{x}_0$, $\boldsymbol{x}(t_f) = \boldsymbol{x}_f$, $\boldsymbol{x}(t)$ satisfies (1) for all $t \in [0, t_f]$, and the cost in (2) is minimized. The optimal control problem for $\boldsymbol{\mathcal{X}} = \mathbb{R}^n$, $\boldsymbol{\mathcal{U}} = \mathbb{R}^m$, and free t_f is defined as

$$\begin{aligned} \min_{\boldsymbol{u}(t), t_f} \quad & \phi\left(t_f, \boldsymbol{x}\left(t_f\right)\right) + \int_0^{t_f} L\left(\tau, \boldsymbol{x}\left(\tau\right), \boldsymbol{u}\left(\tau\right)\right) d\tau \\ \texttt{s.t.} \quad & \dot{\boldsymbol{x}}\left(t\right) = f\left(\boldsymbol{x}\left(t\right), \boldsymbol{u}\left(t\right)\right) \\ & \boldsymbol{\psi}(\boldsymbol{x}_0, \boldsymbol{x}_f) = \boldsymbol{0} \end{aligned}, \tag{3}$$

where the boundary conditions are described by

$$\boldsymbol{\psi}\left(\boldsymbol{x}_0, \boldsymbol{x}_f\right) = \begin{bmatrix} \boldsymbol{x}(0) - \boldsymbol{x}_0 \\ \boldsymbol{x}(t_f) - \boldsymbol{x}_f \end{bmatrix}. \tag{4}$$

The Hamiltonian of the optimal control problem in (3) is described by

$$H\left(t, \boldsymbol{x}\left(t\right), \boldsymbol{u}\left(t\right), \boldsymbol{\lambda}\left(t\right)\right) = L\left(t, \boldsymbol{x}\left(t\right), \boldsymbol{u}\left(t\right)\right) + \boldsymbol{\lambda}\left(t\right)^T f\left(\boldsymbol{x}\left(t\right), \boldsymbol{u}\left(t\right)\right), \tag{5}$$

where $\boldsymbol{\lambda}(t)$ is known as the *Lagrange multiplier* at time t. The components of $\boldsymbol{\lambda}$ are known as the *costates* of the system, and $\boldsymbol{\lambda}(t)$ describes the costate trajectories. Additionally, let

$$\Phi\left(\boldsymbol{x}_0, \boldsymbol{x}_f, t_f, \boldsymbol{\nu}\right) = \phi\left(t_f, \boldsymbol{x}_f\right) + \boldsymbol{\nu}^T \boldsymbol{\psi}(\boldsymbol{x}_0, \boldsymbol{x}_f), \tag{6}$$

where $\boldsymbol{\nu}$ is a boundary condition multiplier.

Theorem 1. *If $f(\boldsymbol{x}(t), \boldsymbol{u}(t))$, $\phi(t_f, \boldsymbol{x}_f)$, $L(t, \boldsymbol{x}(t), \boldsymbol{u}(t))$ and $\boldsymbol{\psi}(\boldsymbol{x}_0, \boldsymbol{x}_f)$ are all class $\mathcal{C}^1$ functions, $\boldsymbol{\mathcal{X}} = \mathbb{R}^n$ and $\boldsymbol{\mathcal{U}} = \mathbb{R}^m$, and $\boldsymbol{u}^*(t),\ \forall\ t \in [0, t_f^*]$ is the optimal control input trajectory with optimal state and costate trajectories $\boldsymbol{x}^*(t)$ and $\boldsymbol{\lambda}^*(t),\ \forall\ t \in [0, t_f^*]$, where t_f^* is the optimal final time, then*

1. *The Hamiltonian is minimized by the optimal control input trajectory, which implies that*

$$H\left(t, \boldsymbol{x}^*\left(t\right), \boldsymbol{u}^*\left(t\right), \boldsymbol{\lambda}^*\left(t\right)\right) \leq H\left(t, \boldsymbol{x}^*\left(t\right), \boldsymbol{u}, \boldsymbol{\lambda}^*\left(t\right)\right), \ \forall\ \boldsymbol{u} \in \boldsymbol{\mathcal{U}}, \ \forall\ t \in [0, t_f^*]. \tag{7}$$

2. *If the final time t_f is a free variable, then*

$$H\left(t_f^*, \boldsymbol{x}^*\left(t_f^*\right), \boldsymbol{u}^*\left(t_f^*\right), \boldsymbol{\lambda}^*\left(t_f^*\right)\right) = -\frac{\partial \Phi\left(\boldsymbol{x}_0, \boldsymbol{x}_f, t_f, \boldsymbol{\nu}\right)}{\partial t_f}\bigg|_{t_f = t_f^*}, \tag{8}$$

which is known as the transversality condition.

3. *The optimal Lagrange multiplier satisfies Hamilton's equation*

$$\dot{\boldsymbol{\lambda}}^*\left(t\right) = -\left.\frac{\partial H\left(t, \boldsymbol{x}\left(t\right), \boldsymbol{u}\left(t\right), \boldsymbol{\lambda}\left(t\right)\right)}{\partial \boldsymbol{x}\left(t\right)}\right|_{\boldsymbol{x}(t)=\boldsymbol{x}^*(t),\ \boldsymbol{u}(t)=\boldsymbol{u}^*(t),\ \boldsymbol{\lambda}(t)=\boldsymbol{\lambda}^*(t)}. \tag{9}$$

Proof. This theorem is the result of Pontryagin's principle of optimality and a proof is provided in [61]. □

From (7) it can be seen that, if the Hamiltonian in (5) is a class $\mathcal{C}^1$ function and $\boldsymbol{\mathcal{U}} = \mathbb{R}^m$, then the first order necessary condition for $\boldsymbol{u}\left(t\right)$ to minimize (5) is

$$\frac{\partial H\left(t, \boldsymbol{x}\left(t\right), \boldsymbol{u}\left(t\right), \boldsymbol{\lambda}\left(t\right)\right)}{\partial \boldsymbol{u}\left(t\right)} = 0, \ \forall\ t \in \left[0, t_f^*\right]. \tag{10}$$

Also, if (5) is a class $\mathcal{C}^2$ function and $\boldsymbol{\mathcal{U}} = \mathbb{R}^m$, then the second order necessary condition is

$$\frac{\partial^2 H\left(t, \boldsymbol{x}\left(t\right), \boldsymbol{u}\left(t\right), \boldsymbol{\lambda}\left(t\right)\right)}{\partial \boldsymbol{u}^2\left(t\right)} \geq 0, \ \forall\ t \in \left[0, t_f^*\right]. \tag{11}$$

Theorem 2. *If the boundary conditions in* (4) *and the Hamiltonian in* (5) *are class $\mathcal{C}^1$ functions that do not depend explicitly on time, $\boldsymbol{\mathcal{U}} = \mathbb{R}^m$, there is no terminal state cost, and t_f is free, then*

$$H\left(t, \boldsymbol{x}^*\left(t\right), \boldsymbol{u}^*\left(t\right), \boldsymbol{\lambda}^*\left(t\right)\right) = 0, \ \forall\ t \in \left[0, t_f^*\right], \tag{12}$$

where $\boldsymbol{u}^*(t)$, $\forall\ t \in [0, t_f^*]$ is the optimal control input trajectory that minimizes the Hamiltonian, $\boldsymbol{x}^*(t)$ and $\boldsymbol{\lambda}^*(t)$, $\forall\ t \in [0, t_f^*]$ are the optimal state and costate trajectories, and t_f^* is the optimal final time.

Proof. Given the assumption that $\boldsymbol{\psi}(\boldsymbol{x}_0, \boldsymbol{x}_f)$ is a class $\mathcal{C}^1$ function that does not depend explicitly on time, if there is no terminal penalty cost, which leads to $\Phi\left(\boldsymbol{x}_0, \boldsymbol{x}_f, \boldsymbol{\nu}\right) = \boldsymbol{\nu}^T \boldsymbol{\psi}(\boldsymbol{x}_0, \boldsymbol{x}_f)$, then it can be seen from

the transversality condition (8) that

$$H\left(t_f^*, \boldsymbol{x}^*\left(t_f^*\right), \boldsymbol{u}^*\left(t_f^*\right), \boldsymbol{\lambda}^*\left(t_f^*\right)\right) = 0. \tag{13}$$

The time-derivative of the Hamiltonian can be evaluated as

$$\frac{dH^*\left(\cdot\right)}{dt} = \frac{\partial H^*\left(\cdot\right)}{\partial t} + \frac{\partial H^*\left(\cdot\right)}{\partial \boldsymbol{x}^*\left(t\right)}\frac{d\boldsymbol{x}^*\left(t\right)}{dt} + \frac{\partial H^*\left(\cdot\right)}{\partial \boldsymbol{u}^*\left(t\right)}\frac{d\boldsymbol{u}^*\left(t\right)}{dt} + \frac{\partial H^*\left(\cdot\right)}{\partial \boldsymbol{\lambda}^*\left(t\right)}\frac{d\boldsymbol{\lambda}^*\left(t\right)}{dt}, \tag{14}$$

where $H^*(\cdot) = H(t, \boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \boldsymbol{\lambda}^*(t))$. Given the assumption that the Hamiltonian does not explicitly depend on time it can be seen that $\frac{\partial H^*(\cdot)}{\partial t} = 0$. Given the assumption that $\boldsymbol{\mathcal{U}} = \mathbb{R}^m$, if $\boldsymbol{u}^*(t)$ minimizes the Hamiltonian, then it can be seen from (10) that $\frac{\partial H^*(\cdot)}{\partial \boldsymbol{u}^*(t)} = 0$. If $\boldsymbol{x}^*(t)$ and $\boldsymbol{\lambda}^*(t)$ are the optimal state and costate trajectories then it can be seen from (9) that $\frac{\partial H^*(\cdot)}{\partial \boldsymbol{x}^*(t)} = -\dot{\boldsymbol{\lambda}}^*(t)$. Finally, it can be seen from (5) that $\frac{\partial H^*(\cdot)}{\partial \boldsymbol{\lambda}^*(t)} = f(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t))^T = \dot{\boldsymbol{x}}^*(t)^T$. Applying these observations to (14) yields

$$\frac{dH\left(\boldsymbol{x}^*\left(t\right), \boldsymbol{u}^*\left(t\right), \boldsymbol{\lambda}^*\left(t\right)\right)}{dt} = -\dot{\boldsymbol{\lambda}}^*\left(t\right)\dot{\boldsymbol{x}}^*\left(t\right)^T + \dot{\boldsymbol{x}}^*\left(t\right)^T\dot{\boldsymbol{\lambda}}^*\left(t\right) = 0, \tag{15}$$

because of the relationship

$$\dot{\boldsymbol{x}}^*\left(t\right)^T\dot{\boldsymbol{\lambda}}^*\left(t\right) = \dot{\boldsymbol{\lambda}}^*\left(t\right)\dot{\boldsymbol{x}}^*\left(t\right)^T. \tag{16}$$

Given that the Hamiltonian is zero at t_f^*, as shown in (13), and since the Hamiltonian is constant $\forall\ t \in [0, t_f^*]$, as shown in (15), then

$$H\left(\boldsymbol{x}^*\left(t\right), \boldsymbol{u}^*\left(t\right), \boldsymbol{\lambda}^*\left(t\right)\right) = 0,\ \forall\ t \in \left[0, t_f^*\right]. \tag{17}$$

□

Theorem 3. *Given an optimal control input trajectory $\boldsymbol{u}^*(t)$, optimal state trajectory $\boldsymbol{x}^*(t)$, and final time t_f^*, the optimal control input and state trajectories from $\boldsymbol{x}^*(0)$ to $\boldsymbol{x}^*(\tilde{t}_f^*)\ \forall\ \tilde{t}_f^* \in [0, t_f^*]$ are $\tilde{\boldsymbol{u}}^*(t) = \boldsymbol{u}^*(t)$ and $\tilde{\boldsymbol{x}}^*(t) = \boldsymbol{x}^*(t), \forall\ t \in [0, \tilde{t}_f^*]$, respectively.*

Proof. This theorem is a consequence of Bellman's principle of optimality [62]. □

Theorem 4. *Given an optimal control input trajectory $\boldsymbol{u}^*(t)$, optimal state trajectory $\boldsymbol{x}^*(t)$, and optimal final time t_f^*, the cost of the optimal trajectory from $\boldsymbol{x}^*(0)$ to $\boldsymbol{x}^*(\tilde{t}_f^*)\ \forall\ \tilde{t}_f^* \in [0, t_f^*]$ is $C(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \tilde{t}_f^*)$, where the cost function is defined in* (2).

Proof. It can be seen from Theorem 3 that the optimal control input and state trajectories to steer the system from $\boldsymbol{x}^*(0)$ to $\boldsymbol{x}^*(\tilde{t}_f^*)$, $\forall\ \tilde{t}_f^* \in [0, t_f^*]$ are $\tilde{\boldsymbol{u}}^*(t) = \boldsymbol{u}^*(t)$ and $\tilde{\boldsymbol{x}}^*(t) = \boldsymbol{x}^*(t)$, $\forall\ t \in [0, \tilde{t}_f^*]$, respectively. The cost of the optimal trajectory from $\boldsymbol{x}^*(0)$ to $\boldsymbol{x}^*(\tilde{t}_f^*)$ is obtained according to (2) as

$$C\left(\tilde{\boldsymbol{x}}^*\left(t\right), \tilde{\boldsymbol{u}}^*\left(t\right), \tilde{t}_f^*\right) = \phi\left(\tilde{t}_f^*, \tilde{\boldsymbol{x}}^*\left(\tilde{t}_f^*\right)\right) + \int_0^{\tilde{t}_f^*} L\left(\tau, \tilde{\boldsymbol{x}}^*\left(\tau\right), \tilde{\boldsymbol{u}}^*\left(\tau\right)\right) d\tau. \tag{18}$$

Given that $\tilde{\boldsymbol{x}}^*(t) = \boldsymbol{x}^*(t)$ and $\tilde{\boldsymbol{u}}^*(t) = \boldsymbol{u}^*(t)$, $\forall\ t \in [0, \tilde{t}_f^*]$, equation (18) may be expressed as

$$C\left(\tilde{\boldsymbol{x}}^*\left(t\right), \tilde{\boldsymbol{u}}^*\left(t\right), \tilde{t}_f^*\right) = \phi\left(\tilde{t}_f^*, \boldsymbol{x}^*\left(\tilde{t}_f^*\right)\right) + \int_0^{\tilde{t}_f^*} L\left(\tau, \boldsymbol{x}^*\left(\tau\right), \boldsymbol{u}^*\left(\tau\right)\right) d\tau = C(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \tilde{t}_f^*). \tag{19}$$

□

2.1.1 Optimal Control of Affine Systems and the Minimization of a Cost Trading Off Control Effort and Time

This section presents a review of optimal control theory of affine systems and the minimization of a cost trading off control effort and time and is based on references [47, 48, 59]. Let the system be a point-mass that verifies Newton's second law of motion. This implies that

$$\boldsymbol{F} + m\boldsymbol{g} = m\boldsymbol{a}, \tag{20}$$

where $\boldsymbol{F}$ is the vector of input forces acting on the system, m is the mass of the system, and $\boldsymbol{a}$ is the acceleration of the system. From (20) it can be seen that

$$\boldsymbol{a} = \frac{1}{m}\boldsymbol{F} + \boldsymbol{g}. \tag{21}$$

Next, assume that the system is subject to a constant wind velocity vector $\boldsymbol{c}_v$. The time-derivative of the relative position of the system may therefore before described by

$$\dot{\boldsymbol{p}} = \boldsymbol{v} + \boldsymbol{c}_v, \tag{22}$$

where $\boldsymbol{p}$ is the system's position, $\boldsymbol{v}$ is its velocity, and $\boldsymbol{c}_v$ is the constant wind velocity vector. Similarly, the time-derivative of the velocity of the system may be described by

$$\dot{\boldsymbol{v}} = \boldsymbol{a} = \frac{1}{m}\boldsymbol{F} + \boldsymbol{g}. \tag{23}$$

If the control input is $\boldsymbol{u} = \boldsymbol{F}$ it can be seen that

$$\begin{bmatrix} \dot{\boldsymbol{p}} \\ \dot{\boldsymbol{v}} \end{bmatrix} = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{v} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} I \end{bmatrix} \boldsymbol{u} + \begin{bmatrix} \boldsymbol{c}_v \\ \boldsymbol{g} \end{bmatrix}, \tag{24}$$

which is an affine system. Let the system's state transition equation be described by the more general form

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} + \boldsymbol{c}, \tag{25}$$

where $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$, $\boldsymbol{c} \in \mathbb{R}^n$, and the pair (A, B) is assumed controllable. The system's state space is assumed to be $\boldsymbol{\mathcal{X}} = \mathbb{R}^n$ and its control input space is assumed to be $\boldsymbol{\mathcal{U}} = \mathbb{R}^m$. Given a control input trajectory $\boldsymbol{u}(t)$, a state trajectory $\boldsymbol{x}(t)$, and a final time t_f, the cost of the trajectory is

$$C\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t_f\right) = \int_0^{t_f} \left(\frac{1}{2} \boldsymbol{u}\left(\tau\right)^T R \boldsymbol{u}\left(\tau\right) + C_I \right) d\tau, \tag{26}$$

where $R \in \mathbb{R}^{m \times m}$ is a symmetric and positive definite matrix known as the control input weighting matrix and $C_I \in \mathbb{R}_{>0}$ is a trade-off parameter between control effort and time. Given an initial state $\boldsymbol{x}_0$ and a final state $\boldsymbol{x}_f$, the optimal control problem to steer the system from $\boldsymbol{x}_0$ to $\boldsymbol{x}_f$ for a free t_f such that (26) is minimized is defined as [59]

$$\begin{aligned} & \min_{\boldsymbol{u}(t), t_f} \int_0^{t_f} \left(\tfrac{1}{2} \boldsymbol{u}\left(\tau\right)^T R \boldsymbol{u}\left(\tau\right) + C_I \right) d\tau \\ \texttt{s.t.} \quad & \dot{\boldsymbol{x}}\left(t\right) = A\boldsymbol{x}\left(t\right) + B\boldsymbol{u}\left(t\right) + \boldsymbol{c} \\ & \boldsymbol{\psi}\left(\boldsymbol{x}_0, \boldsymbol{x}_f\right) = \boldsymbol{0} \end{aligned}, \tag{27}$$

where $\boldsymbol{\psi}\left(\boldsymbol{x}_0, \boldsymbol{x}_f\right)$ is given by (4).

Theorem 5. *If $R \in \mathbb{R}^{n \times n}$ is symmetric and positive definite, $C_I \in \mathbb{R}_{>0}$, and the pair (A, B) is controllable, then the solution for the optimal control input trajectory of (27) is*

$$\boldsymbol{u}^*\left(t\right) = -R^{-1} B^T e^{A^T \left(t_f^* - t\right)} \boldsymbol{\lambda}^*(t_f^*), \tag{28}$$

where t_f^* is the optimal final time,

$$\boldsymbol{\lambda}^*\left(t_f^*\right) = -G_c\left(0, t_f^*\right)^{-1}\left(\boldsymbol{x}_f - e^{At_f^*}\boldsymbol{x}_0 - \int_0^{t_f^*} e^{A\left(t_f^*-\tau\right)}\boldsymbol{c}d\tau\right), \tag{29}$$

and where the controllability Grammian is

$$G_c\left(0, t_f^*\right) = \int_0^{t_f^*} e^{A\left(t_f^*-\tau\right)}BR^{-1}B^T e^{A^T\left(t_f^*-\tau\right)}d\tau. \tag{30}$$

Additionally, the solution for the optimal Hamiltonian at t_f^* is

$$H\left(t_f^*, \boldsymbol{x}^*\left(t_f^*\right), \boldsymbol{u}^*\left(t_f^*\right), \boldsymbol{\lambda}^*\left(t_f^*\right)\right) = -\frac{1}{2}\boldsymbol{\lambda}^*\left(t_f^*\right)^T BR^{-1}B^T\boldsymbol{\lambda}^*\left(t_f^*\right) + \boldsymbol{\lambda}^*\left(t_f^*\right)^T\left(A\boldsymbol{x}_f + \boldsymbol{c}\right) + C_I, \tag{31}$$

where the solution for the optimal final time of (27) *satisfies*

$$H\left(t_f^*, \boldsymbol{x}^*\left(t_f^*\right), \boldsymbol{u}^*\left(t_f^*\right), \boldsymbol{\lambda}^*\left(t_f^*\right)\right) = 0. \tag{32}$$

Proof. A proof of this result is presented in [59]. □

Theorem 6. *Let $\boldsymbol{u}^*(t)$ and $\boldsymbol{x}^*(t)$, $\forall\ t \in [0, t_f^*]$ denote the optimal control input and state trajectories, respectively, to steer the system from $\boldsymbol{x}_0$ to $\boldsymbol{x}_f$, where t_f^* is the optimal final time. For a system of the form* (25) *and the cost function in* (26)*, if the pair (A, B) is controllable, $R \in \mathbb{R}^{n\times n}$ is symmetric and positive definite, and $C_I \in \mathbb{R}_{>0}$, then the cost-to-go from $\boldsymbol{x}_0$ to $\boldsymbol{x}^*(\tilde{t}_f^*)$ is a strictly increasing function of $\tilde{t}_f^*$ that satisfies*

$$\begin{array}{rcll} C(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \tilde{t}_f^*) & = & 0 & , \quad \tilde{t}_f^* = 0 \\ C(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \tilde{t}_f^*) & > & 0 & , \quad \forall\ \tilde{t}_f^* \in (0, t_f^*] \end{array}. \tag{33}$$

Proof. From Theorem 4 it can be seen that the cost-to-go from $\boldsymbol{x}_0$ to $\boldsymbol{x}^*(\tilde{t}_f^*)$ is $C(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \tilde{t}_f^*)$. Given the definition of the optimal control input trajectory in (28), the cost-to-go from $\boldsymbol{x}_0$ to $\boldsymbol{x}^*(\tilde{t}_f^*)$ is

$$C(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \tilde{t}_f^*) = \frac{1}{2}\boldsymbol{\lambda}^{*^T}\left(t_f^*\right)e^{At_f^*}\left(\int_0^{\tilde{t}_f^*} e^{-A\tau}BR^{-1}B^T e^{-A^T\tau}d\tau\right)e^{A^T t_f^*}\boldsymbol{\lambda}^*\left(t_f^*\right) + \tilde{t}_f^* C_I. \tag{34}$$

It can be seen that (34) is equivalent to

$$C(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \tilde{t}_f^*) = \frac{1}{2}\boldsymbol{X}_c^T D(\tilde{t}_f^*)\boldsymbol{X}_c + \tilde{t}_f^* C_I, \tag{35}$$

where

$$\boldsymbol{X}_c = e^{A^T t_f^*}\boldsymbol{\lambda}^*\left(t_f^*\right) \quad , \quad D(\tilde{t}_f^*) = \int_0^{\tilde{t}_f^*} e^{-A\tau} B R^{-1} B^T e^{-A^T\tau} d\tau. \tag{36}$$

Given the assumption that R is symmetric and positive definite it can be seen that $D(\tilde{t}_f^*)$ is positive definite $\forall\ \tilde{t}_f^* \in (0, t_f^*]$ and is equal to 0_n when $\tilde{t}_f^* = 0$. Furthermore, given the assumption that $C_I \in \mathbb{R}_{>0}$ it can be seen that $\tilde{t}_f^* C_I > 0$, $\forall\ \tilde{t}_f^* > 0$ and is equal to 0 when $\tilde{t}_f^* = 0$. Therefore,

$$\begin{array}{rcl} C(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \tilde{t}_f^*) & = & 0 \quad , \quad \tilde{t}_f^* = 0 \\ C(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \tilde{t}_f^*) & > & 0 \quad , \quad \forall\ \tilde{t}_f^* \in (0, t_f^*] \end{array} . \tag{37}$$

The derivative of (35) with respect to $\tilde{t}_f^*$ is

$$\frac{dC(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), \tilde{t}_f^*)}{d\tilde{t}_f^*} = \frac{1}{2}\boldsymbol{X}_c^T D(\tilde{t}_f^*)' \boldsymbol{X}_c + C_I, \tag{38}$$

where

$$D(\tilde{t}_f^*)' = e^{-A\tilde{t}_f^*} B R^{-1} B^T e^{-A^T \tilde{t}_f^*}. \tag{39}$$

Given the assumption that R is symmetric and positive definite it can be seen that $D(\tilde{t}_f^*)'$ is positive definite. Therefore, given the assumption that $C_I \in \mathbb{R}_{>0}$, it can be seen that (38) is positive $\forall\ \tilde{t}_f^* \in [0, t_f^*]$, which implies that the cost-to-go from $\boldsymbol{x}_0$ to $\boldsymbol{x}^*(\tilde{t}_f^*)$ is a strictly increasing function of $\tilde{t}_f^*$. □

2.2 Optimal Trajectory Planning and Non-Convex State Constraints

Many optimal trajectory planning problems can be solved using optimal control theory. However, the difficulty of finding an optimal control input trajectory $\boldsymbol{u}^*(t)$ that minimizes the Hamiltonian increases significantly when the system is subject to state constraints. This is especially true when these constraints cause the problem to become non-convex. Obtaining a solution directly from optimal control theory is impossible for many of these problems. Because of this, various techniques have been proposed to solve or to approximate the solutions of non-convex optimal control problems.

The system's state is described by the vector $\boldsymbol{x} \in \mathbb{R}^n$ and its control input by the vector $\boldsymbol{u} \in \mathbb{R}^m$, where $n, m \in \mathbb{N}_{>0}$. The system's state transition equation is given by (1), where the system's state space is assumed to be $\boldsymbol{\mathcal{X}} \subseteq \mathbb{R}^n$ and it's control input space is assumed to be $\boldsymbol{\mathcal{U}} = \mathbb{R}^m$. A control input trajectory is denoted by $\boldsymbol{u}(t)$, which produces a state trajectory $\boldsymbol{x}(t)$. Without loss of generality, the trajectory is assumed to occur for $t \in [0, t_f]$, where t_f is known as the final time. Given $\boldsymbol{x}(t)$, $\boldsymbol{u}(t)$, and t_f, the *cost* of the trajectory is given by (2). It is assumed that a set of obstacles must be avoided, which is described by the open set

$\boldsymbol{\mathcal{X}}_{\texttt{obs}} \subset \boldsymbol{\mathcal{X}}$. The *obstacle-free* space is therefore defined as

$$\boldsymbol{\mathcal{X}}_{\texttt{free}} = \boldsymbol{\mathcal{X}} \setminus \boldsymbol{\mathcal{X}}_{\texttt{obs}}, \tag{40}$$

which is the set of states that do not result in a collision with any obstacles. Planning optimal collision-free trajectories in $\boldsymbol{\mathcal{X}}$ is therefore equivalent to planning optimal trajectories in $\boldsymbol{\mathcal{X}}_{\texttt{free}}$, which is often a non-convex space. The optimal collision-free trajectory planning problem can therefore be defined as an extension of (3) according to

$$\begin{aligned} \min_{\boldsymbol{u}(t), t_f} \quad & \phi\left(t_f, \boldsymbol{x}_f\right) + \int_0^{t_f} L\left(\tau, \boldsymbol{x}\left(\tau\right), \boldsymbol{u}\left(\tau\right)\right) d\tau \\ \texttt{s.t.} \quad & \dot{\boldsymbol{x}}\left(t\right) = f\left(\boldsymbol{x}\left(t\right), \boldsymbol{u}\left(t\right)\right) \\ & \boldsymbol{x} : [0, t_f] \rightarrow \boldsymbol{\mathcal{X}}_{\texttt{free}} \\ & \boldsymbol{\psi}\left(\boldsymbol{x}_0, \boldsymbol{x}_f\right) = \boldsymbol{0} \end{aligned}, \tag{41}$$

where $\boldsymbol{\psi}\left(\boldsymbol{x}_0, \boldsymbol{x}_f\right)$ is given by (4). Evidently, no guarantee can be made that the solution obtained from the theory in Section 2.1 will satisfy the collision-avoidance constraint $\boldsymbol{x} : [0, t_f] \rightarrow \boldsymbol{\mathcal{X}}_{\texttt{free}}$. Iterative sampling-based solvers such as Kinodynamic RRT* attempt to approximate the solution of (41) by applying the theory of Section 2.1 to a graph. The remainder of this section is as follows. First, a review of graph theory is presented with emphasis on the material applicable to Kinodynamic RRT*. Then, the Kinodynamic RRT* trajectory planning algorithm is presented and an example is provided for a double integrator with drift.

2.2.1 Review of Graph Theory

The definitions presented in this section are taken from [63]. A graph $\mathcal{G}$ consists of a set of vertices $\boldsymbol{V}$ and edges $\boldsymbol{E}$, where each edge connects two (not necessarily distinct) vertices in $\boldsymbol{V}$. If the edges in $\boldsymbol{E}$ have no direction associated with them then $\mathcal{G}$ is called an *undirected graph*. A path in an undirected graph is defined as a sequence of edges in $\boldsymbol{E}$ connecting a sequence of (not necessarily distinct) vertices in $\boldsymbol{V}$. If there exists exactly one path between all pairs of distinct vertices in $\boldsymbol{V}$ then $\mathcal{G}$ is known as a *tree*. If the edges in $\boldsymbol{E}$ have a direction associated with them then $\mathcal{G}$ is called a *directed graph*. The underlying undirected graph of a directed graph is obtained when the directed edges are replaced with undirected edges. Let $e_{ij} \in \boldsymbol{E}$ be the edge that is directed from $\boldsymbol{v}_i \in \boldsymbol{V}$ to $\boldsymbol{v}_j \in \boldsymbol{V}$. The vertex $\boldsymbol{v}_i$ is called the *tail* of e_{ij} and $\boldsymbol{v}_j$ is called the *head* of e_{ij}. A directed path is similar to a path in an undirected graph, except that the head of each edge is the tail of the succeeding edge in the sequence. If, for all $\boldsymbol{v} \in \boldsymbol{V}$, there does not exist a directed path that begins and ends at $\boldsymbol{v} \in \boldsymbol{V}$, then $\mathcal{G}$ is called a *directed acyclic graph*. If $\mathcal{G}$ is a directed acyclic graph and its underlying undirected graph is a tree then $\mathcal{G}$ is called a *directed tree*. If $\mathcal{G}$ is a directed tree then,

for the edge e_{ij}, the vertex $\boldsymbol{v}_i$ is called the *parent* vertex of $\boldsymbol{v}_j$, and $\boldsymbol{v}_j$ is called the *child* vertex of $\boldsymbol{v}_i$. If $\mathcal{G}$ is a directed tree, and if all of the edges of $\boldsymbol{E}$ are directed either away or toward a vertex $\boldsymbol{v}_0 \in \boldsymbol{V}$, then $\mathcal{G}$ is called a *directed rooted tree*, where $\boldsymbol{v}_0$ is the *root*. Finally, if $\mathcal{G}$ is a directed rooted tree and all edges are directed away from the root, then $\mathcal{G}$ is called an *arborescence*.

Theorem 7. *If $\mathcal{G}$ is an arborescence, which is comprised of a set of vertices $\boldsymbol{V}$ and directed edges $\boldsymbol{E}$, and is rooted at $\boldsymbol{v}_0 \in \boldsymbol{V}$, then any $\boldsymbol{v} \in \boldsymbol{V} \setminus \{\boldsymbol{v}_0\}$ is the child of exactly one vertex in $\boldsymbol{V} \setminus \{\boldsymbol{v}\}$.*

Proof. A proof of this result is presented in [64]. □

2.2.2 Review of Kinodynamic RRT*

RRT* is a powerful sampling-based motion planner developed to plan trajectories through non-convex spaces. Kinodynamic RRT* searches for a solution of (41) by incrementally growing an arborescence in $\boldsymbol{\mathcal{X}}_{\texttt{free}}$ [42]. The tree is rooted at $\boldsymbol{x}_0 \in \boldsymbol{\mathcal{X}}_{\texttt{free}}$ and is incrementally expanded in search of $\boldsymbol{x}_f \in \boldsymbol{\mathcal{X}}_{\texttt{free}}$. Following the completion of a given number of iterations, $N_{\texttt{iter}} \in \mathbb{R}_{>0}$, an attempt is made to optimally connect $\boldsymbol{x}_f$ with the tree. If this attempt is successful then the approximate solution of (41) is obtained as the concatenation of the sequence of trajectories described by the edges along the directed path from $\boldsymbol{x}_0$ to $\boldsymbol{x}_f$ through the tree. If this attempt is unsuccessful then no solution is obtained.

Definition 1. *The arborescence created by RRT* is denoted by $\mathcal{T}$, which is comprised of a set of vertices $\boldsymbol{V} \subset \boldsymbol{\mathcal{X}}_{\texttt{free}}$ and edges $\boldsymbol{E}$. The i-th edge of $\mathcal{T}$ is denoted by $e_i \in \boldsymbol{E}$. Let $\boldsymbol{v}_{i_0} \in \boldsymbol{V}$ denote the tail of the edge and $\boldsymbol{v}_{i_f} \in \boldsymbol{V}$ the head. The edge e_i is characterized by the triple $(\boldsymbol{x}_i^*(t), \boldsymbol{u}_i^*(t), t_{f_i}^*)$, where $\boldsymbol{u}_i^*(t)$, $\forall\ t \in [0, t_{f_i}^*]$ is the optimal control input trajectory to steer the system from $\boldsymbol{v}_{i_0}$ to $\boldsymbol{v}_{i_f}$, $\boldsymbol{x}_i^*(t)$, $\forall\ t \in [0, t_{f_i}^*]$ is the optimal state trajectory such that $\boldsymbol{x}_i^*(0) = \boldsymbol{v}_{i_0}$, $\boldsymbol{x}_i^*(t_{f_i}^*) = \boldsymbol{v}_{i_f}$ and $\boldsymbol{x}_i^* : [0, t_{f_i}^*] \to \boldsymbol{\mathcal{X}}_{\texttt{free}}$, and $t_{f_i}^*$ is the optimal final time of the trajectory.* □

Definition 2. *The following list taken from [43] defines the functions required by RRT*.*

- *Edge Components Function: Given an edge $e_i \in \boldsymbol{E}$, where $e_i = (\boldsymbol{x}_i^*(t), \boldsymbol{u}_i^*(t), t_{f_i}^*)$, the components of the edge are retrieved by the function*

$$\boldsymbol{x}_i^*(t), \boldsymbol{u}_i^*(t), t_{f_i}^* \leftarrow \texttt{GetComponents}\,(e_i)\,.$$

- *Edge Cost Function: Given an edge $e_i \in \boldsymbol{E}$, the cost of e_i is obtained from the function*

$$c_{e_i} \leftarrow \texttt{Cost}\left(e_i\right),$$

which obtains $\boldsymbol{x}_i^*(t)$, $\boldsymbol{u}_i^*(t)$, and $t_{f_i}^*$ from the edge components function and returns the cost

$$c_{e_i} = C\left(\boldsymbol{x}_i^*\left(t\right), \boldsymbol{u}_i^*\left(t\right), t_{f_i}^*\right). \tag{42}$$

- *"In" Edge Function: Given a vertex $\boldsymbol{v}_i \in \boldsymbol{V} \setminus \{\boldsymbol{v}_0\}$, the edge that is directed to $\boldsymbol{v}_i$ is obtained from the function*

$$e_i \leftarrow \mathcal{T}.\texttt{GetInEdge}\left(\boldsymbol{v}_i\right),$$

which returns the edge $e_i \in \boldsymbol{E}$, where $\boldsymbol{v}_i$ is the head of e_i. It can be seen from Theorem 7 that, since the tree generated by RRT is an arborescence, each $\boldsymbol{v} \in \boldsymbol{V} \setminus \{\boldsymbol{v}_0\}$ is the child of exactly one vertex in $\boldsymbol{V} \setminus \{\boldsymbol{v}\}$. This implies that there exists exactly one $e_i \in \boldsymbol{E}$ directed to $\boldsymbol{v}_i$.*

- *Sampling Function: Random state space samples are obtained from the function*

$$\boldsymbol{x} \leftarrow \texttt{Sample}\left(\right),$$

which returns a randomly generated state $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}_{\texttt{free}}$. Unless otherwise stated, the sampling function is assumed to generate independent identically distributed (IID) states according to a uniform distribution across $\boldsymbol{\mathcal{X}}_{\texttt{free}}$.

- *Steering Function: Let $\boldsymbol{u}_{1,2}^*(t)$, $\forall\ t \in [0, t_{f_{1,2}}^*]$ be the unconstrained optimal control input trajectory to steer the system from $\boldsymbol{x}_1$ to $\boldsymbol{x}_2$, where $t_{f_{1,2}}^*$ is the unconstrained optimal final time. The trajectory $\boldsymbol{u}_{1,2}^*(t)$ and the time $t_{f_{1,2}}^*$ are obtained as the solution of*

$$\begin{aligned} \min_{\boldsymbol{u}_{1,2}(t), t_{f_{1,2}}} \quad & \phi\left(t_{f_{1,2}}, \boldsymbol{x}_2\right) + \int_0^{t_{f_{1,2}}} L\left(\tau, \boldsymbol{x}_{1,2}\left(\tau\right), \boldsymbol{u}_{1,2}\left(\tau\right)\right) d\tau \\ \texttt{s.t.} \quad & \dot{\boldsymbol{x}}_{1,2}\left(t\right) = f\left(\boldsymbol{x}_{1,2}\left(t\right), \boldsymbol{u}_{1,2}\left(t\right)\right) \\ & \boldsymbol{\psi}\left(\boldsymbol{x}_1, \boldsymbol{x}_2\right) = \boldsymbol{0} \end{aligned}, \tag{43}$$

and $\boldsymbol{x}_{1,2}^*(t)$, $\forall\ t \in [0, t_{f_{1,2}}^*]$ is the unconstrained optimal state trajectory obtained by integrating (1) for the control input $\boldsymbol{u}_{1,2}^*(t)$ from $t = 0$ to $t = t_{f_{1,2}}^*$. Given an initial state $\boldsymbol{x}_1 \in \boldsymbol{\mathcal{X}}$ and a final state $\boldsymbol{x}_2 \in \boldsymbol{\mathcal{X}}$, the unconstrained optimal state trajectory $\boldsymbol{x}_{1,2}^*(t)$, $\forall\ t \in [0, t_{f_{1,2}}^*]$, the unconstrained optimal

control input trajectory $\boldsymbol{u}^*_{1,2}(t)$, $\forall\ t \in [0, t^*_{f_{1,2}}]$, *and the unconstrained optimal final time* $t^*_{f_{1,2}}$ *to steer the system from* $\boldsymbol{x}_1$ *to* $\boldsymbol{x}_2$ *are obtained from the function*

$$\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), t^*_{f_{1,2}} \leftarrow \texttt{Steer}(\boldsymbol{x}_1, \boldsymbol{x}_2).$$

- <u>Unconstrained Cost-to-Go Function</u>: *Given an initial state* $\boldsymbol{x}_1 \in \boldsymbol{\mathcal{X}}$ *and a final state* $\boldsymbol{x}_2 \in \boldsymbol{\mathcal{X}}$, *the unconstrained cost-to-go from* $\boldsymbol{x}_1$ *to* $\boldsymbol{x}_2$ *is obtained from the function*

$$c_{TG_{1,2}} \leftarrow \texttt{CTG}(\boldsymbol{x}_1, \boldsymbol{x}_2),$$

which returns the cost

$$c_{TG_{1,2}} = C\left(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), t^*_{f_{1,2}}\right), \tag{44}$$

where the cost function is defined in (2) *and where* $\boldsymbol{x}^*_{1,2}(t)$ *and* $\boldsymbol{u}^*_{1,2}(t)$, $\forall t \in [0, t^*_{f_{1,2}}]$ *are the unconstrained optimal state and control input trajectories, and* $t^*_{f_{1,2}}$ *is the unconstrained optimal final time to steer the system from* $\boldsymbol{x}_1$ *to* $\boldsymbol{x}_2$.

- <u>Vertex Cost Function</u>: *Let* $\boldsymbol{E}_i \subseteq \boldsymbol{E}$ *be the set of edges that describe the directed path from* $\boldsymbol{v}_0$ *to* $\boldsymbol{v}_i \in \boldsymbol{V}$ *through* $\mathcal{T}$. *Note that, since the tree generated by RRT* is an arborescence, it can be seen from the definition of an arborescence in Section 2.2.1 that there exists exactly one path from* $\boldsymbol{x}_0$ *to every* $\boldsymbol{v} \in \boldsymbol{V} \setminus \{\boldsymbol{x}_0\}$. *The cost of* $\boldsymbol{v}_i$ *is obtained from the function*

$$c_{v_i} \leftarrow \mathcal{T}.\texttt{Cost}(\boldsymbol{v}_i),$$

which first obtains the set $\boldsymbol{E}_i$ *and returns the cost*

$$c_{v_i} = \sum_{e \in \boldsymbol{E}_i} \texttt{Cost}(e). \tag{45}$$

Note that the cost of $\boldsymbol{v}_i$ *is not necessarily equal to that returned by* $\texttt{CTG}(\boldsymbol{x}_0, \boldsymbol{v}_i)$. *The reason for this is that the unconstrained cost-to-go function evaluates state trajectories in* $\mathbb{R}^n$, *rather than in* $\boldsymbol{\mathcal{X}}_{free}$. *Because of this, the trajectory considered by the unconstrained cost-to-go function may not be feasible, whereas the trajectories described by the edges in* $\boldsymbol{E}_i$ *are. Additionally, the trajectories described by the edges in* $\boldsymbol{E}_i$ *may not necessarily describe the optimal trajectory from* $\boldsymbol{v}_0$ *to* $\boldsymbol{v}_i$ *through* $\boldsymbol{\mathcal{X}}_{free}$. *However, when a set of conditions are met (which are discussed later), the trajectories described by the edges in* $\boldsymbol{E}_i$ *converge to the optimal trajectory from* $\boldsymbol{x}_0$ *to* $\boldsymbol{v}_i$ *through* $\boldsymbol{\mathcal{X}}_{free}$ *as* $N_{iter} \to \infty$.

- *Local Trajectory Function: Given an optimal control input trajectory* $\boldsymbol{u}^*_{1,2}(t)$, $\forall\, t \in [0, t^*_{f_{1,2}}]$, *an optimal state trajectory* $\boldsymbol{x}^*_{1,2}(t)$, $\forall\, t \in [0, t^*_{f_{1,2}}]$, *and an optimal final time* $t^*_{f_{1,2}}$ *to steer the system from* $\boldsymbol{x}_1 \in \boldsymbol{\mathcal{X}}$ *to* $\boldsymbol{x}_2 \in \boldsymbol{\mathcal{X}}$, *and an upper bound on the cost-to-go* $\eta \in \mathbb{R}_{>0}$, *the local control input and state trajectories are described by* $\tilde{\boldsymbol{u}}^*_{1,2}(t) = \boldsymbol{u}^*_{1,2}(t)$ *and* $\tilde{\boldsymbol{x}}^*_{1,2}(t) = \boldsymbol{x}^*_{1,2}(t)$, $\forall\, t \in [0, \tilde{t}^*_{f_{1,2}}]$, *where the local trajectory final time is*

$$\begin{aligned} \tilde{t}^*_{f_{1,2}} \quad &= \quad \underset{\tau \in \left[0, t^*_{f_{1,2}}\right]}{\operatorname{argmax}} \; C\left(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), \tau\right) \\ &\texttt{s.t.} \quad C\left(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), \tau\right) \leq \eta \end{aligned} \tag{46}$$

and where the cost function is given by (2). *In other words,* $[0, \tilde{t}^*_{f_{1,2}}]$ *is the maximum time interval such that the cost of the trajectory described by* $\tilde{\boldsymbol{u}}^*_{1,2}(t)$ *and* $\tilde{\boldsymbol{x}}^*_{1,2}(t)$, $\forall\, t \in [0, \tilde{t}^*_{f_{1,2}}]$ *is less than, or equal to,* η. *The local trajectories and their final time are obtained from the function*

$$\tilde{\boldsymbol{x}}^*_{1,2}(t), \tilde{\boldsymbol{u}}^*_{1,2}(t), \tilde{t}^*_{f_{1,2}} \leftarrow \texttt{LocalTraj}\left(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), t^*_{f_{1,2}}, \eta\right).$$

Note that it can be seen from Theorem 3 that $\tilde{\boldsymbol{u}}^*_{1,2}(t)$ *and* $\tilde{\boldsymbol{x}}^*_{1,2}(t)$, $\forall\, t \in [0, \tilde{t}^*_{f_{1,2}}]$ *are the optimal control input and state trajectories to steer the system from* $\boldsymbol{x}^*_{1,2}(0)$ *to* $\boldsymbol{x}^*_{1,2}(\tilde{t}^*_{f_{1,2}})$. *Additionally, given the cost function in* (2), *it can be seen from Theorem 4 that* $C(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), \tilde{t}^*_{f_{1,2}}) = C(\tilde{\boldsymbol{x}}^*_{1,2}(t), \tilde{\boldsymbol{u}}^*_{1,2}(t), \tilde{t}^*_{f_{1,2}})$.

- *Nearest Vertex Function: Given a state* $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$, *the vertex* $\boldsymbol{v}_{min} \in \boldsymbol{V}$ *is defined as that for which the unconstrained cost-to-go from* $\boldsymbol{v}_{min}$ *to* $\boldsymbol{x}$ *is the minimum of all vertices in* $\boldsymbol{V}$. *This vertex is obtained from the function*

$$\boldsymbol{v}_{min} \leftarrow \texttt{Nearest}\left(\boldsymbol{V}, \boldsymbol{x}\right),$$

which returns the vertex

$$\boldsymbol{v}_{min} = \underset{\boldsymbol{v} \in \boldsymbol{V}}{\operatorname{argmin}} \, \texttt{CTG}(\boldsymbol{v}, \boldsymbol{x}). \tag{47}$$

- *Nearby Vertices Function: Given a state* $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$ *and an upper bound on the unconstrained cost-to-go* $c_{max} \in \mathbb{R}_{\geq 0}$, *the set of vertices* $\boldsymbol{V}_{max} \subseteq \boldsymbol{V}$ *is defined as that for which the unconstrained cost-to-go from each* $\boldsymbol{v} \in \boldsymbol{V}_{max}$ *to* $\boldsymbol{x}$ *is less than, or equal to,* c_{max}. *This set of vertices is obtained from the function*

$$\boldsymbol{V}_{max} \leftarrow \texttt{Nearby}\left(\boldsymbol{V}, \boldsymbol{x}, c_{max}\right)$$

which returns the set

$$\boldsymbol{V}_{max} = \left\{\boldsymbol{v} \in \boldsymbol{V} : \texttt{CTG}(\boldsymbol{v}, \boldsymbol{x}) \leq c_{max}\right\}. \tag{48}$$

- Collision-Checker Function: *Given an unconstrained optimal state trajectory* $\boldsymbol{x}^*_{1,2}(t)$, $\forall\ t \in [0, t^*_{f_{1,2}}]$ *to steer the system from* $\boldsymbol{x}_1$ *to* $\boldsymbol{x}_2$, *where* $t^*_{f_{1,2}}$ *is the unconstrained optimal final time,* $\boldsymbol{x}^*_{1,2}(t)$ *is evaluated for collisions with* $\boldsymbol{\mathcal{X}}_{obs}$ *according to the function*

$$\beta \leftarrow \texttt{CollisionFree}\left(\boldsymbol{x}^*_{1,2}\left(t\right), t^*_{f_{1,2}}\right),$$

which returns the boolean

$$\beta = \begin{cases} \textit{true}, & \boldsymbol{x}^*_{1,2} : \left[0, t^*_{f_{1,2}}\right] \rightarrow \boldsymbol{\mathcal{X}}_{free} \\ \textit{false}, & \texttt{otherwise} \end{cases}. \tag{49}$$

- Get Solution Function: *Given an arborescence* $\mathcal{T}$, *initial state* $\boldsymbol{x}_0 \in \boldsymbol{V}$, *and final state* $\boldsymbol{x}_f \in \boldsymbol{V}$, *the suboptimal state and control input trajectories,* $\boldsymbol{x}^*(t)$ *and* $\boldsymbol{u}^*(t)$, $\forall\ t \in [0, t^*_f]$, *and the suboptimal final time* t^*_f *obtained by RRT* to steer the system from* $\boldsymbol{x}_0$ *to* $\boldsymbol{x}_f$ *through* $\boldsymbol{\mathcal{X}}_{free}$ *are obtained from the function*

$$\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t^*_f \leftarrow \texttt{GetSol}\left(\mathcal{T}, \boldsymbol{x}_0, \boldsymbol{x}_f\right),$$

where the function $\texttt{GetSol}(\cdot)$ *is presented in Algorithm 1 on page 29.* □

2.2.2.1 Efficient Techniques for Expanding the Tree of RRT*

1. Cost-to-go upper bound η: Let $\boldsymbol{x}_{\texttt{rand}} \in \boldsymbol{\mathcal{X}}_{\texttt{free}}$ denote a randomly generated state space sample and let $\boldsymbol{v}_{\texttt{min}} \in \boldsymbol{V}$ denote the vertex obtained according to (47), where $\boldsymbol{x} = \boldsymbol{x}_{\texttt{rand}}$. The vertex $\boldsymbol{x}_{\texttt{rand}}$ can be added to $\boldsymbol{V}$ if and only if the state trajectory from $\boldsymbol{v}_{\texttt{min}}$ to $\boldsymbol{x}_{\texttt{rand}}$ is collision-free. However, in environments with many obstacles, it may be difficult to generate random samples such that the state trajectory from $\boldsymbol{v}_{\texttt{min}}$ to $\boldsymbol{x}_{\texttt{rand}}$ is collision-free. RRT* therefore defines a reachability region in $\boldsymbol{\mathcal{X}}$ around each $\boldsymbol{v} \in \boldsymbol{V}$ such that the parameter η is an upper bound on the unconstrained cost-to-go from each $\boldsymbol{v}$. The tree is then limited in growth by only adding vertices that lie within this region. RRT* uses the local trajectory function in Definition 2 to steer the system from $\boldsymbol{v}_{\texttt{min}}$ toward $\boldsymbol{x}_{\texttt{rand}}$ up to a maximum cost η. As a consequence of this, a new candidate sample $\boldsymbol{x}_{\texttt{new}}$ is obtained along this trajectory within the reachability region of $\boldsymbol{v}_{\texttt{min}}$.

2. Cost-to-go upper bound $c_{\texttt{max}}$: Let $\boldsymbol{x}_{\texttt{new}} \in \boldsymbol{\mathcal{X}}_{\texttt{free}}$ denote a new candidate state space sample and let $\boldsymbol{v}_{\texttt{min}} \in \boldsymbol{V}$ denote the vertex obtained according to (47), where $\boldsymbol{x} = \boldsymbol{x}_{\texttt{new}}$. If the trajectory from $\boldsymbol{v}_{\texttt{min}}$ to $\boldsymbol{x}_{\texttt{new}}$ is collision-free, then $\boldsymbol{x}_{\texttt{new}}$ is added to $\boldsymbol{V}$ and two optimization procedures are performed. The first procedure tries to find a $\boldsymbol{v} \in \boldsymbol{V} \setminus \{\boldsymbol{v}_{\texttt{min}}\}$ that minimizes the cost of $\boldsymbol{x}_{\texttt{new}}$, where the cost of a vertex is described by the vertex cost function in Definition 2. The second procedure attempts to

minimize the cost of other vertices due to the addition of $\boldsymbol{x}_{\texttt{new}}$ to $\boldsymbol{V}$. However, if all $\boldsymbol{v} \in \boldsymbol{V} \setminus \{\boldsymbol{x}_{\texttt{new}}\}$ are considered, then the steering function is required to produce trajectories from $\boldsymbol{x}_{\texttt{new}}$ to all $\boldsymbol{v} \in \boldsymbol{V}$ and from each $\boldsymbol{v} \in \boldsymbol{V}$ to $\boldsymbol{x}_{\texttt{new}}$, which can be very time-consuming. RRT* uses the parameter $c_{\texttt{max}}$ to define a reachability region in $\boldsymbol{\mathcal{X}}$ around $\boldsymbol{x}_{\texttt{new}}$ such that $c_{\texttt{max}}$ is an upper bound on the unconstrained cost-to-go to $\boldsymbol{x}_{\texttt{new}}$. Let $\boldsymbol{\mathcal{X}}_{\texttt{new}}$ denote this region. RRT* therefore reduces the computation time of these two procedures by only considering the vertices in the set $\boldsymbol{V} \bigcap \boldsymbol{\mathcal{X}}_{\texttt{new}}$, which is obtained from the nearby vertices function in Definition 2. However, observe that

$$\lim_{N_{\texttt{iter}} \to \infty} \boldsymbol{V} = \boldsymbol{\mathcal{X}}_{\texttt{free}}. \tag{50}$$

Therefore, since $\boldsymbol{\mathcal{X}}_{\texttt{free}} \bigcap \boldsymbol{\mathcal{X}}_{\texttt{new}}$ has an infinite number of points for a constant non-zero $c_{\texttt{max}}$, in the limit of the number of iterations converging to infinity it can be seen that the number of vertices in $\boldsymbol{V} \bigcap \boldsymbol{\mathcal{X}}_{\texttt{new}}$ approaches infinity. To circumvent this problem it is proposed in [42] that $c_{\texttt{max}}$ be a decreasing function of $|\boldsymbol{V}|$ described by

$$c_{\texttt{max}} = \min \left\{ \eta, \gamma \left(\frac{\log \left(|\boldsymbol{V}| \right)}{|\boldsymbol{V}|} \right)^{\frac{1}{n}} \right\}, \tag{51}$$

where γ is a tuning parameter selected to scale the decrease of $c_{\texttt{max}}$. As a result, the reachability region considered around $\boldsymbol{x}_{\texttt{new}}$ decreases as $|\boldsymbol{V}|$ increases, and consequently

$$\lim_{|\boldsymbol{V}| \to \infty} c_{\texttt{max}} = 0. \tag{52}$$

Theorem 8. *Let $\mu\left(\boldsymbol{\mathcal{X}}_{free}\right)$ be the Lebesgue measure of $\boldsymbol{\mathcal{X}}_{free}$ and let ζ_n be the volume of a n-dimensional unit ball obtained as*

$$\zeta_n = \frac{\pi^{\frac{n}{2}}}{\Gamma\left(\frac{n}{2} + 1\right)}, \tag{53}$$

where $\Gamma(\cdot)$ is the Euler Gamma function. If $\boldsymbol{\mathcal{X}} \subset \mathbb{R}^n$ is a compact set and

$$\gamma > \left(2 \left(1 + \frac{1}{n} \right) \frac{\mu\left(\boldsymbol{\mathcal{X}}_{free}\right)}{\zeta_n} \right)^{\frac{1}{n}}, \tag{54}$$

then the probability that the RRT trajectory planner returns the optimal solution of* (41)*, provided one exists, approaches one as the number of iterations approaches infinity.*

Proof. A proof of this theorem is provided in references [42] and [43]. □

2.2.2.2 Kinodynamic RRT* Algorithm

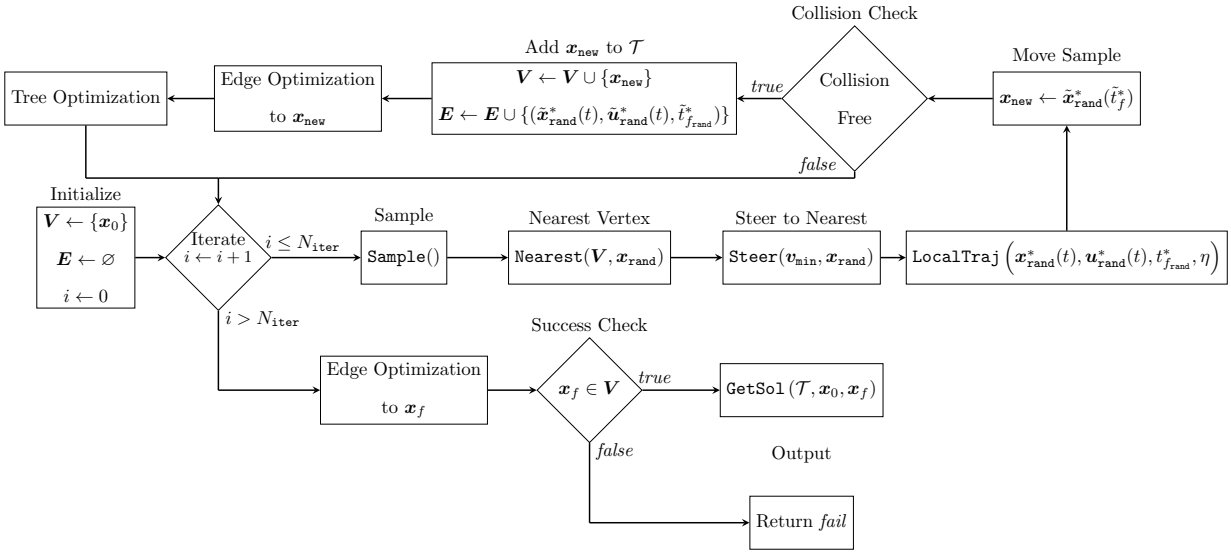


Figure 5: RRT* flow chart.

The RRT* algorithm requires the following inputs:

- The initial and final states: $\boldsymbol{x}_0, \boldsymbol{x}_f \in \boldsymbol{\mathcal{X}}_{\texttt{free}}$.
- The number of iterations to be performed: $N_{\texttt{iter}}$.
- The cost-to-go upper bound: η.
- The reachability region tuning parameter: γ.

A flow chart of the RRT* algorithm is shown in Figure 5. The decision steps are shown as diamond boxes with the result of the decision shown at each output, and the processes are shown as rectangular boxes. The following list describes the steps of each RRT* iteration, where all functions were introduced in Definition 2.

1. A random state space sample $\boldsymbol{x}_{\texttt{rand}} \in \boldsymbol{\mathcal{X}}_{\texttt{free}}$ is obtained from the sampling function.
2. The minimum cost-to-go vertex $\boldsymbol{v}_{\texttt{min}} \in \boldsymbol{V}$ is obtained from the nearest vertex function.
3. The unconstrained optimal state and control input trajectories, $\boldsymbol{x}^*_{\texttt{rand}}(t)$ and $\boldsymbol{u}^*_{\texttt{rand}}(t)$, $\forall\ t \in [0, t^*_{f_{\texttt{rand}}}]$, and the unconstrained optimal final time $t^*_{f_{\texttt{rand}}}$ to steer the system from $\boldsymbol{v}_{\texttt{min}}$ to $\boldsymbol{x}_{\texttt{rand}}$ are obtained from the steering function.
4. The local trajectories $\tilde{\boldsymbol{u}}^*_{\texttt{rand}}(t)$ and $\tilde{\boldsymbol{x}}^*_{\texttt{rand}}(t)$, $\forall\ t \in [0, \tilde{t}^*_{f_{\texttt{rand}}}]$ and their final time $\tilde{t}^*_{f_{\texttt{rand}}}$ are obtained from $\boldsymbol{u}^*_{\texttt{rand}}(t)$, $\boldsymbol{x}^*_{\texttt{rand}}(t)$, and $t^*_{f_{\texttt{rand}}}$ to steer the system from $\boldsymbol{v}_{\texttt{min}}$ toward $\boldsymbol{x}_{\texttt{rand}}$ up to a maximum cost η using the local trajectory function.

5. The new candidate vertex is obtained as $\boldsymbol{x}_{\texttt{new}} = \tilde{\boldsymbol{x}}^*_{\texttt{rand}}(\tilde{t}^*_{f_{\texttt{rand}}})$.

6. The collision-checker function evaluates if $\tilde{\boldsymbol{x}}^*_{\texttt{rand}} : [0, \tilde{t}^*_{f_{\texttt{rand}}}] \rightarrow \boldsymbol{\mathcal{X}}_{\texttt{free}}$. If the collision-checker function returns *false*, then the algorithm proceeds to the next iteration (Step 1). If the collision-checker function returns *true*, then the algorithm continues to Step 7.

7. The state $\boldsymbol{x}_{\texttt{new}}$ is added to $\boldsymbol{V}$, and the edge $e_{\texttt{new}} = (\tilde{\boldsymbol{x}}^*_{\texttt{rand}}(t), \tilde{\boldsymbol{u}}^*_{\texttt{rand}}(t), \tilde{t}^*_{f_{\texttt{rand}}})$ is added to $\boldsymbol{E}$.

8. An edge optimization procedure determines the best connection of $\boldsymbol{x}_{\texttt{new}}$ with $\mathcal{T}$. First, the set $\boldsymbol{V}_{\texttt{max}}$ for $\boldsymbol{x}_{\texttt{new}}$ is obtained from the nearby vertices function using the parameters η and γ. Then, the optimal vertex to connect $\boldsymbol{x}_{\texttt{new}}$ with $\mathcal{T}$ is obtained as

$$\begin{aligned} \boldsymbol{v}_{\texttt{opt}} \quad &= \quad \underset{\boldsymbol{v} \in \boldsymbol{V}_{\texttt{max}}}{\operatorname{argmin}} \left(\mathcal{T}.\texttt{Cost}\left(\boldsymbol{v}\right) + \texttt{CTG}\left(\boldsymbol{v}, \boldsymbol{x}_{\texttt{new}}\right)\right) \\ \texttt{s.t.} \quad & \boldsymbol{x}^*_{\texttt{opt}}(t), \boldsymbol{u}^*_{\texttt{opt}}(t), t^*_{f_{\texttt{opt}}} \leftarrow \texttt{Steer}(\boldsymbol{v}, \boldsymbol{x}_{\texttt{new}}) \\ & \texttt{CollisionFree}\left(\boldsymbol{x}^*_{\texttt{opt}}\left(t\right), t^*_{f_{\texttt{opt}}}\right) = \mathit{true} \end{aligned} \quad . \tag{55}$$

   If a $\boldsymbol{v}_{\texttt{opt}}$ is found, then the edge $e_{\texttt{new}}$ is replaced by $e_{\texttt{opt}} = (\boldsymbol{x}^*_{\texttt{opt}}(t), \boldsymbol{u}^*_{\texttt{opt}}(t), t^*_{f_{\texttt{opt}}})$, where $\boldsymbol{x}^*_{\texttt{opt}}(t)$ and $\boldsymbol{u}^*_{\texttt{opt}}(t)$, $\forall\ t \in [0, t^*_{f_{\texttt{opt}}}]$ are the optimal state and control input trajectories and $t^*_{f_{\texttt{opt}}}$ is the optimal final time to steer the system from $\boldsymbol{v}_{\texttt{opt}}$ to $\boldsymbol{x}_{\texttt{new}}$.

9. A tree optimization procedure attempts to minimize the cost of any $\boldsymbol{v} \in \boldsymbol{V}_{\texttt{max}}$ using $\boldsymbol{x}_{\texttt{new}}$. Let $\boldsymbol{v}_{\texttt{max}_i}$ denote the i-th vertex of $\boldsymbol{V}_{\texttt{max}}$. Additionally, let $\boldsymbol{x}^*_i(t)$, $\forall\ t \in [0, t^*_{f_i}]$ denote the unconstrained optimal state trajectory to move the system from $\boldsymbol{x}_{\texttt{new}}$ to $\boldsymbol{v}_{\texttt{max}_i}$, where $t^*_{f_i}$ is the unconstrained optimal final time. For each $i \in [1, 2, \ldots, |\boldsymbol{V}_{\texttt{max}}|]$, if

$$\mathcal{T}.\texttt{Cost}(\boldsymbol{x}_{\texttt{new}}) + \texttt{CTG}(\boldsymbol{x}_{\texttt{new}}, \boldsymbol{v}_{\texttt{max}_i}) < \mathcal{T}.\texttt{Cost}(\boldsymbol{v}_{\texttt{max}_i}) \tag{56}$$

   and $\boldsymbol{x}^*_i(t)$, $\forall\ t \in [0, t^*_{f_i}]$ is evaluated as collision-free, then the edge directed to $\boldsymbol{v}_{\texttt{max}_i}$ is replaced by one characterized by the unconstrained optimal trajectory from $\boldsymbol{x}_{\texttt{new}}$ to $\boldsymbol{v}_{\texttt{max}_i}$.

After the completion of $N_{\texttt{iter}}$ iterations, the set $\boldsymbol{V}_{\texttt{max}}$ is obtained for $\boldsymbol{x}_f$ from the nearby vertices function. An attempt is then made to find a $\boldsymbol{v}_{\texttt{opt}}$ for $\boldsymbol{x}_f$ according to (55), where $\boldsymbol{x}_{\texttt{new}} = \boldsymbol{x}_f$. If no $\boldsymbol{v}_{\texttt{opt}}$ can be found, then $\boldsymbol{x}_f$ cannot be connected with $\mathcal{T}$ and no solution is obtained. If a $\boldsymbol{v}_{\texttt{opt}}$ is found, then $\boldsymbol{x}_f$ is added to $\boldsymbol{V}$ and an edge characterized by $(\boldsymbol{x}^*_f(t), \boldsymbol{u}^*_f(t), t^*_{f_f})$ is added to $\boldsymbol{E}$, where $\boldsymbol{x}^*_f(t)$ and $\boldsymbol{u}^*_f(t)$, $\forall\ t \in [0, t^*_{f_f}]$ are the optimal control input and state trajectories to steer the system from $\boldsymbol{v}_{\texttt{opt}}$ to $\boldsymbol{x}_f$, and $t^*_{f_f}$ is the optimal final time. The reason that the algorithm does not assign $\boldsymbol{x}_{\texttt{rand}} = \boldsymbol{x}_f$ and repeat Steps 2 – 8 is because of Step 6, which terminates the attempt to connect $\boldsymbol{x}_{\texttt{rand}}$ with $\mathcal{T}$ if the trajectory from $\boldsymbol{v}_{\texttt{min}}$ to $\boldsymbol{x}_{\texttt{rand}}$ is not

collision-free. However, after $N_{\texttt{iter}}$ iterations the task addressed by the planner is to attempt to connect $\boldsymbol{x}_f$ with $\mathcal{T}$, which warrants additional computational efforts since there may exists a $\boldsymbol{v} \in \boldsymbol{V}_{\texttt{max}}$ where $\boldsymbol{v} \neq \boldsymbol{v}_{\texttt{min}}$ and the trajectory from $\boldsymbol{v}$ to $\boldsymbol{x}_f$ is collision free, and therefore an approximate solution of the optimal control problem can be obtained. The approximate solution of (41) can then be obtained as the concatenation of the trajectories described by the sequence of directed edges from $\boldsymbol{x}_0$ to $\boldsymbol{x}_f$ through $\mathcal{T}$, which is accomplished by the *get solution* function in Definition 2.

2.2.2.3 Kinodynamic RRT* Pseudocode

The pseudocode for Kinodynamic RRT* is shown in Algorithm 2 on page 30, which describes the implementation of the flow chart in Figure 5. The initialization of the tree occurs on lines 2 – 3, where $\boldsymbol{x}_0$ is established as the root. The iterative procedure of the algorithm occurs on lines 4 – 31. The random state space sample $\boldsymbol{x}_{\texttt{rand}} \in \boldsymbol{\mathcal{X}}_{\texttt{free}}$ is obtained on line 5. The minimum cost-to-go vertex $\boldsymbol{v}_{\texttt{min}}$ is obtained on line 6. The unconstrained optimal state and control input trajectories and the unconstrained final time to move the system from $\boldsymbol{v}_{\texttt{min}}$ to $\boldsymbol{x}_{\texttt{rand}}$ are obtained on line 7. The unconstrained optimal state and control input trajectories and the unconstrained optimal final time to steer the system from $\boldsymbol{v}_{\texttt{min}}$ toward $\boldsymbol{x}_{\texttt{rand}}$ up to a maximum cost η are obtained on line 8. The new candidate state $\boldsymbol{x}_{\texttt{new}}$ is established on line 9. The unconstrained state trajectory from $\boldsymbol{v}_{\texttt{min}}$ to $\boldsymbol{x}_{\texttt{new}}$ is evaluated for collisions with obstacles on line 10. If this trajectory is evaluated as being collision-free, then it is added to $\mathcal{T}$ on lines 11 – 13. The set $\boldsymbol{V}_{\texttt{max}}$ is obtained on lines 14 – 15. The edge optimization procedure for $\boldsymbol{x}_{\texttt{new}}$ occurs on lines 16 – 23, and the tree optimization procedure occurs on lines 24 – 31. After the completion of $N_{\texttt{iter}}$ iterations, an attempt is made to optimally connect $\boldsymbol{x}_f$ with $\mathcal{T}$, which occurs on lines 33 – 42. If $\boldsymbol{x}_f$ can be connected with $\mathcal{T}$ then the approximate solution of (41) is retrieved and returned by the planner, as shown on lines 43 – 47, where the pseudocode for the function `GetSol`$(\cdot)$ is provided in Algorithm 1. If $\boldsymbol{x}_f$ cannot be connected with $\mathcal{T}$, then a *fail* is returned by the planner, as show on line 48.

Algorithm 1 Get RRT* solution

function GETSOL($\mathcal{T}$, $\boldsymbol{x}_0$, $\boldsymbol{x}_f$)
  $e_{\texttt{in}} \leftarrow \mathcal{T}.\texttt{GetInEdge}(\boldsymbol{x}_f)$;
  $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{GetComponents}(e_{\texttt{in}})$;
  $\boldsymbol{x}^*_{\texttt{end}}(t) \leftarrow \boldsymbol{x}^*(t)$;
  $\boldsymbol{u}^*_{\texttt{end}}(t) \leftarrow \boldsymbol{u}^*(t)$;
  $t^*_{\texttt{end}} \leftarrow t_f^*$;
  $\boldsymbol{x}_{\texttt{parent}} \leftarrow \boldsymbol{x}^*(0)$;
  $e_{\texttt{in}} \leftarrow \mathcal{T}.\texttt{GetInEdge}(\boldsymbol{x}_{\texttt{parent}})$;
  $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{GetComponents}(e_{\texttt{in}})$;
  $\boldsymbol{x}^*_{\texttt{mid}}(t) \leftarrow \boldsymbol{x}^*(t)$;
  $\boldsymbol{u}^*_{\texttt{mid}}(t) \leftarrow \boldsymbol{u}^*(t)$;
  $t^*_{\texttt{mid}} \leftarrow t_f^*$;
  $\boldsymbol{x}_{\texttt{parent}} \leftarrow \boldsymbol{x}^*(0)$;
  while $\boldsymbol{x}_{\texttt{parent}} \neq \boldsymbol{x}_0$ **do**
    $e_{\texttt{in}} \leftarrow \mathcal{T}.\texttt{GetInEdge}(\boldsymbol{x}_{\texttt{parent}})$;
    $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{GetComponents}(e_{\texttt{in}})$;
    $t^*_{\texttt{mid}} \leftarrow t^*_{\texttt{mid}} + t_f^*$;
    $\boldsymbol{x}^*_{\texttt{mid}}(t) \leftarrow \begin{cases} \boldsymbol{x}^*(t) & , \quad t \in [0, t_f^*) \\ \boldsymbol{x}^*_{\texttt{mid}}(t) & , \quad t \in [t_f^*, t^*_{\texttt{mid}}) \end{cases}$;
    $\boldsymbol{u}^*_{\texttt{mid}}(t) \leftarrow \begin{cases} \boldsymbol{u}^*(t) & , \quad t \in [0, t_f^*) \\ \boldsymbol{u}^*_{\texttt{mid}}(t) & , \quad t \in [t_f^*, t^*_{\texttt{mid}}) \end{cases}$;
    $\boldsymbol{x}_{\texttt{parent}} \leftarrow \boldsymbol{x}^*(0)$;
  $e_{\texttt{in}} \leftarrow \mathcal{T}.\texttt{GetInEdge}(\boldsymbol{x}^*(t_f^*))$;
  $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{GetComponents}(e_{\texttt{in}})$;
  $t^*_{\texttt{mid}} \leftarrow t^*_{\texttt{mid}} + t_f^*$;
  $t^*_{\texttt{sol}} \leftarrow t^*_{\texttt{mid}} + t^*_{\texttt{end}}$;
  $\boldsymbol{x}^*_{\texttt{sol}}(t) \leftarrow \begin{cases} \boldsymbol{x}^*(t) & , \quad t \in [0, t_f^*) \\ \boldsymbol{x}^*_{\texttt{mid}}(t) & , \quad t \in [t_f^*, t^*_{\texttt{mid}}) \\ \boldsymbol{x}^*_{\texttt{end}}(t) & , \quad t \in [t^*_{\texttt{mid}}, t^*_{\texttt{sol}}] \end{cases}$;
  $\boldsymbol{u}^*_{\texttt{sol}}(t) \leftarrow \begin{cases} \boldsymbol{u}^*(t) & , \quad t \in [0, t_f^*) \\ \boldsymbol{u}^*_{\texttt{mid}}(t) & , \quad t \in [t_f^*, t^*_{\texttt{mid}}) \\ \boldsymbol{u}^*_{\texttt{end}}(t) & , \quad t \in [t^*_{\texttt{mid}}, t^*_{\texttt{sol}}] \end{cases}$;
  return $\boldsymbol{x}^*_{\texttt{sol}}(t), \boldsymbol{u}^*_{\texttt{sol}}(t), t^*_{\texttt{sol}}$;

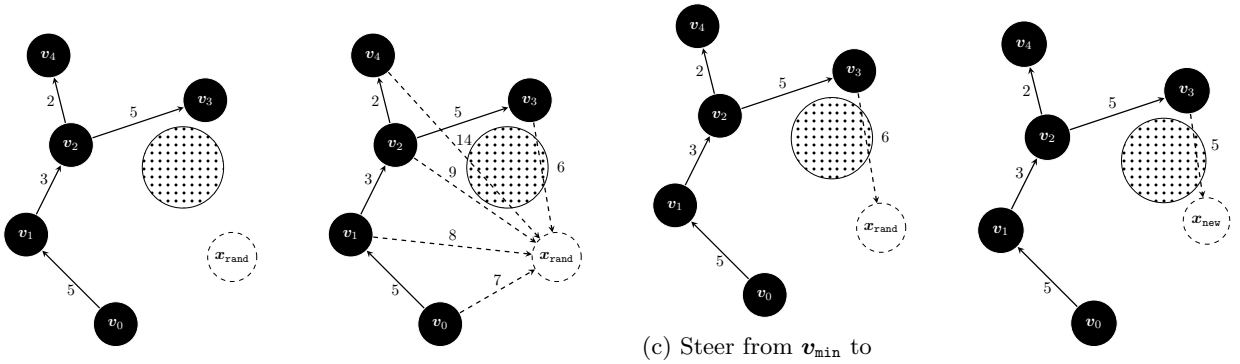
Algorithm 2 Kinodynamic RRT* algorithm

function RRT*($\boldsymbol{x}_0$, $\boldsymbol{x}_f$, $N_{\texttt{iter}}$, η, γ)
  $\boldsymbol{V} \leftarrow \{\boldsymbol{x}_0\}$;
  $\boldsymbol{E} \leftarrow \varnothing$;
  for $i = 1$ to $N_{\texttt{iter}}$ **do**
    $\boldsymbol{x}_{\texttt{rand}} \leftarrow \texttt{Sample}()$;
    $\boldsymbol{v}_{\texttt{min}} \leftarrow \texttt{Nearest}(\boldsymbol{V}, \boldsymbol{x}_{\texttt{rand}})$;
    $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{Steer}(\boldsymbol{v}_{\texttt{min}}, \boldsymbol{x}_{\texttt{rand}})$;
    $\tilde{\boldsymbol{x}}^*(t), \tilde{\boldsymbol{u}}^*(t), \tilde{t}_f^* \leftarrow \texttt{LocalTraj}(\boldsymbol{x}^*(t), t_f^*, \eta)$;
    $\boldsymbol{x}_{\texttt{new}} \leftarrow \tilde{\boldsymbol{x}}^*(\tilde{t}_f^*)$;
    if $\texttt{CollisionFree}(\tilde{\boldsymbol{x}}^*(t), \tilde{t}_f^*)$ **then**
      $e_{\texttt{new}} \leftarrow (\tilde{\boldsymbol{x}}^*(t), \tilde{\boldsymbol{u}}^*(t), \tilde{t}_f^*)$;
      $\boldsymbol{V} \leftarrow \boldsymbol{V} \cup \{\boldsymbol{x}_{\texttt{new}}\}$;
      $\boldsymbol{E} \leftarrow \boldsymbol{E} \cup \{e_{\texttt{new}}\}$;
      $c_{\texttt{max}} = \min\{\eta, \gamma(\frac{\log(|\boldsymbol{V}|)}{|\boldsymbol{V}|})^{\frac{1}{n}}\}$;
      $\boldsymbol{V}_{\texttt{max}} \leftarrow \texttt{Nearby}(\boldsymbol{V} \setminus \{\boldsymbol{x}_{\texttt{new}}\}, \boldsymbol{x}_{\texttt{new}}, c_{\texttt{max}})$;
      for each $\boldsymbol{v} \in \boldsymbol{V}_{\texttt{max}}$ **do**
        $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{Steer}(\boldsymbol{v}, \boldsymbol{x}_{\texttt{new}})$;
        if $\texttt{CollisionFree}(\boldsymbol{x}^*(t), t_f^*)$ **then**
          $e_{\texttt{temp}} \leftarrow (\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*)$;
          $c_e \leftarrow \texttt{Cost}(e_{\texttt{temp}})$;
          if $\mathcal{T}.\texttt{Cost}(\boldsymbol{v}) + c_e < \mathcal{T}.\texttt{Cost}(\boldsymbol{x}_{\texttt{new}})$ **then**
            $\boldsymbol{E} \leftarrow \boldsymbol{E} \cup \{e_{\texttt{temp}}\} \setminus \{e_{\texttt{new}}\}$;
            $e_{\texttt{new}} \leftarrow e_{\texttt{temp}}$;
      for each $\boldsymbol{v} \in \boldsymbol{V}_{\texttt{max}}$ **do**
        $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{Steer}(\boldsymbol{x}_{\texttt{new}}, \boldsymbol{v})$;
        if $\texttt{CollisionFree}(\boldsymbol{x}^*(t), t_f^*)$ **then**
          $e_{\texttt{temp}} \leftarrow (\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*)$;
          $c_e \leftarrow \texttt{Cost}(e_{\texttt{temp}})$;
          if $\mathcal{T}.\texttt{Cost}(\boldsymbol{x}_{\texttt{new}}) + c_e < \mathcal{T}.\texttt{Cost}(\boldsymbol{v})$ **then**
            $\boldsymbol{e}_{\texttt{old}} \leftarrow \mathcal{T}.\texttt{GetInEdge}(\boldsymbol{v})$;
            $\boldsymbol{E} \leftarrow \boldsymbol{E} \cup \{e_{\texttt{temp}}\} \setminus \{e_{\texttt{old}}\}$;
    $c_{\texttt{max}} = \min\{\eta, \gamma(\frac{\log(|\boldsymbol{V}|)}{|\boldsymbol{V}|})^{\frac{1}{n}}\}$;
    $\boldsymbol{V}_{\texttt{max}} \leftarrow \texttt{Nearby}(\boldsymbol{V}, \boldsymbol{x}_f, c_{\texttt{max}})$;
    $c_f \leftarrow \mathcal{T}.\texttt{Cost}(\boldsymbol{x}_f)$;
    for each $\boldsymbol{v} \in \boldsymbol{V}_{\texttt{nearby}}$ **do**
      $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{Steer}(\boldsymbol{v}, \boldsymbol{x}_f)$;
      if $\texttt{CollisionFree}(\boldsymbol{x}^*(t), t_f^*)$ **then**
        $e_{\texttt{temp}} \leftarrow (\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*)$;
        $c_e \leftarrow \texttt{Cost}(e_{\texttt{temp}})$;
        if $\mathcal{T}.\texttt{Cost}(\boldsymbol{v}) + c_e < c_f$ **then**
          $c_f \leftarrow \mathcal{T}.\texttt{Cost}(\boldsymbol{v}) + c_e$;
          $e_{\texttt{new}} \leftarrow e_{\texttt{temp}}$;
    if $c_f \neq \infty$ **then**
      $\boldsymbol{V} \leftarrow \boldsymbol{V} \cup \{\boldsymbol{x}_f\}$;
      $\boldsymbol{E} \leftarrow \boldsymbol{E} \cup \{e_{\texttt{new}}\}$;
      $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{GetSol}(\mathcal{T}, \boldsymbol{x}_0, \boldsymbol{x}_f)$;
      return $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*$
  return *fail*;

2.2.2.4 RRT* Example

Figures 6 and 7 demonstrate two successive iterations of RRT*. These figures show the growth of the tree through $\boldsymbol{\mathcal{X}}_{\texttt{free}}$, where $N_{\texttt{iter}} = 7$ and $n = 2$. The vertices of $\boldsymbol{V}$ are shown as the filled circles with the notation $\boldsymbol{v}_i$, where i denotes their index. $\mathcal{T}$ is rooted at $\boldsymbol{x}_0$, which is shown as $\boldsymbol{v}_0$. Also, the state $\boldsymbol{x}_f$ is assumed to be far from all vertices in $\boldsymbol{V}$ and is omitted for simplicity. The directed edges in $\boldsymbol{E}$ are shown as the solid arrows, where the cost of each edge is shown adjacent to it. The space $\boldsymbol{\mathcal{X}}_{\texttt{obs}}$ describes a single circular obstacle, which is shown as the larger circle filled with dots. The dashed arrows show optimal state trajectories obtained from the steering function, where the cost of each trajectory is shown adjacent to it. States that are not members of $\boldsymbol{V}$, i.e. $\boldsymbol{x}_{\texttt{rand}}$ and $\boldsymbol{x}_{\texttt{new}}$, are shown as the dashed circles. Finally, in this example $\eta = 5$ and $\gamma = 14$ such that $c_{\texttt{max}} = 5$ from (51).

Figure 6 shows a failed iteration, where the optimal state trajectory from $\boldsymbol{v}_{\texttt{min}}$ to $\boldsymbol{x}_{\texttt{new}}$ is rejected by the collision-checker function. In Figure 6a the random state $\boldsymbol{x}_{\texttt{rand}}$ is generated. In Figure 6b the optimal state trajectory from each $\boldsymbol{v} \in \boldsymbol{V}$ is shown, along with their corresponding costs. From this figure it can be seen that the vertex that results in the minimum cost-to-go to $\boldsymbol{x}_{\texttt{rand}}$ is $\boldsymbol{v}_3$, and therefore $\boldsymbol{v}_{\texttt{min}} = \boldsymbol{v}_3$. Figure 6c shows the optimal state trajectory from $\boldsymbol{v}_3$ to $\boldsymbol{x}_{\texttt{rand}}$, which has a cost of 6. Since the cost of this trajectory is greater than $\eta = 5$, $\boldsymbol{x}_{\texttt{new}} \neq \boldsymbol{x}_{\texttt{rand}}$, as shown in Figure 6d. Observe that the optimal state trajectory from $\boldsymbol{v}_3$ to $\boldsymbol{x}_{\texttt{new}}$ intersects with $\boldsymbol{\mathcal{X}}_{\texttt{obs}}$. Therefore, the collision-checker function returns *false* when evaluating this trajectory and the algorithm proceeds to the next iteration.

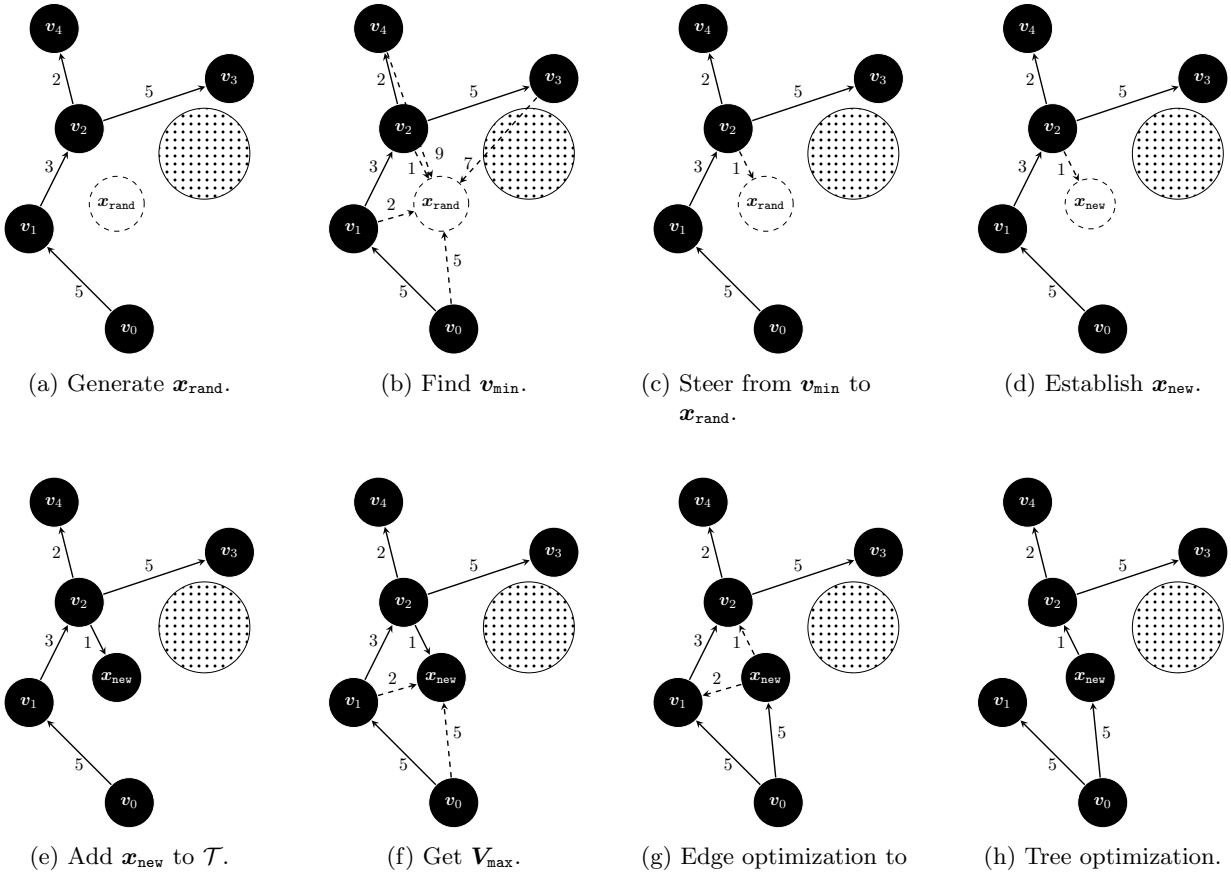


(a) Generate $\boldsymbol{x}_{\texttt{rand}}$. (b) Find $\boldsymbol{v}_{\texttt{min}}$. (c) Steer from $\boldsymbol{v}_{\texttt{min}}$ to $\boldsymbol{x}_{\texttt{rand}}$. (d) Establish $\boldsymbol{x}_{\texttt{new}}$.

Figure 6: RRT* example with failed iteration.

Figure 7 shows a successful iteration where $\boldsymbol{x}_{\texttt{new}}$ is added to $\boldsymbol{V}$ and is used to optimize $\mathcal{T}$. In Figure 7a the random state $\boldsymbol{x}_{\texttt{rand}}$ is generated. In Figure 7b the optimal state trajectory from each $\boldsymbol{v} \in \boldsymbol{V}$ is shown, along with their corresponding costs. From this figure it can be seen that the vertex that results in the minimum cost-to-go to $\boldsymbol{x}_{\texttt{rand}}$ is $\boldsymbol{v}_2$, and therefore $\boldsymbol{v}_{\texttt{min}} = \boldsymbol{v}_2$. Figure 7c shows the optimal state trajectory

from $\boldsymbol{v}_2$ to $\boldsymbol{x}_{\texttt{rand}}$, which has a cost of 1. Since the cost of this trajectory is less than $\eta = 5$, $\boldsymbol{x}_{\texttt{new}} = \boldsymbol{x}_{\texttt{rand}}$, as shown in Figure 7d. Observe that the optimal state trajectory from $\boldsymbol{v}_2$ to $\boldsymbol{x}_{\texttt{new}}$ does not intersect with $\boldsymbol{\mathcal{X}}_{\texttt{obs}}$, and therefore the collision-checker function returns *true* when evaluating this trajectory. In Figure 7e $\boldsymbol{x}_{\texttt{new}}$ is added to $\boldsymbol{V}$ and a directed edge is added from $\boldsymbol{v}_2$ to $\boldsymbol{x}_{\texttt{new}}$. The edge optimization procedure then attempts to minimize the cost of $\boldsymbol{x}_{\texttt{new}}$. Figure 7f shows the optimal trajectories from all $\boldsymbol{v} \in \boldsymbol{V}_{\texttt{max}}$ and their associated costs, where $\boldsymbol{V}_{\texttt{max}} = \{\boldsymbol{v}_0, \boldsymbol{v}_1, \boldsymbol{v}_2\}$. It can be seen that the cost of $\boldsymbol{x}_{\texttt{new}}$ is 9. However if the edge arriving at $\boldsymbol{x}_{\texttt{new}}$ were to originate from $\boldsymbol{v}_1$, then the state trajectory component of this edge would be collision-free and the cost of $\boldsymbol{x}_{\texttt{new}}$ would be reduced to 7. Similarly, if the edge arriving at $\boldsymbol{x}_{\texttt{new}}$ were to originate from $\boldsymbol{v}_0$, then the state trajectory component of this edge would be collision-free and the cost of $\boldsymbol{x}_{\texttt{new}}$ would be reduced to 5. Therefore, the edge optimization procedure replaces the edge from $\boldsymbol{v}_2$ to $\boldsymbol{x}_{\texttt{new}}$ with one from $\boldsymbol{v}_0$ to $\boldsymbol{x}_{\texttt{new}}$, as shown in Figure 7g. Finally, the tree optimization procedure evaluates if $\boldsymbol{x}_{\texttt{new}}$ can be used to reduce the cost of any $\boldsymbol{v} \in \boldsymbol{V}_{\texttt{max}}$. It can be seen from Figure 7g that, if the edge from $\boldsymbol{v}_1$ to $\boldsymbol{v}_2$ is replaced by one from $\boldsymbol{x}_{\texttt{new}}$ to $\boldsymbol{v}_2$ then the cost of $\boldsymbol{v}_2$ is reduced from 8 to 6. Since this is an improvement of the cost of $\boldsymbol{v}_2$, this edge replacement is made, as shown in Figure 7h. This concludes the iteration.



(a) Generate $\boldsymbol{x}_{\texttt{rand}}$. (b) Find $\boldsymbol{v}_{\texttt{min}}$. (c) Steer from $\boldsymbol{v}_{\texttt{min}}$ to $\boldsymbol{x}_{\texttt{rand}}$. (d) Establish $\boldsymbol{x}_{\texttt{new}}$.

(e) Add $\boldsymbol{x}_{\texttt{new}}$ to $\mathcal{T}$. (f) Get $\boldsymbol{V}_{\texttt{max}}$. (g) Edge optimization to $\boldsymbol{x}_{\texttt{new}}$ (h) Tree optimization.

Figure 7: RRT example sketch successful iteration.

2.2.2.5 Kinodynamic RRT* MATLAB Implementation

This section presents an overview of the implementation of Kinodynamic RRT* in MATLAB [65] using object-oriented programming. Four classes are used: (i) a tree class, (ii) a vertex class, (iii) a trajectory class, and (iv) an obstacle class. The reason for selecting this design approach is first and foremost because it facilitates solving different problems since all function implementations regarding specific problems are fully contained in the vertex and obstacle classes. Therefore, when solving specific problems only the vertex and obstacle classes require modifications. The remainder of this section is as follows. First, a generalization of the obstacle and vertex classes are presented, which change according to the problem being solved. Then, the trajectory and the tree classes are presented, which are consistent for all problems. A more specific implementation of the vertex and obstacle classes for the minimization of the cost function in (26) for a double integrator with drift is provided in Appendix C. Also, note that the following functions are native to MATLAB:

- `length` $(\cdot)$
- `vecnorm` $(\cdot)$
- `min` $(\cdot)$
- `log` $(\cdot)$
- `isempty` $(\cdot)$

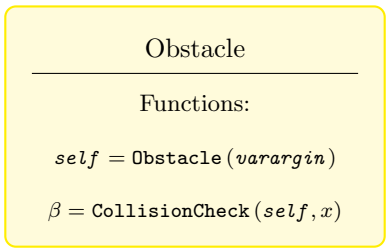


Figure 8: General obstacle class.

A generalization of the obstacle class is shown in Figure 8. Generally speaking, the obstacle class has the following member functions:

- `Obstacle` $(\texttt{varargin})$: This function is the class constructor, which receives one or more inputs described by $\texttt{varargin}$. Each input corresponds to a property of the obstacle class, and is assigned accordingly.

- `CollisionCheck` $(\texttt{self}, x)$: The input of this function is an n row double precision array x, where each column of x describes a state. This function evaluates if any column of x describes a state that intersects with the obstacle. If no states described by the columns of x intersect with the obstacle then the function returns *true*, and it returns *false* otherwise.

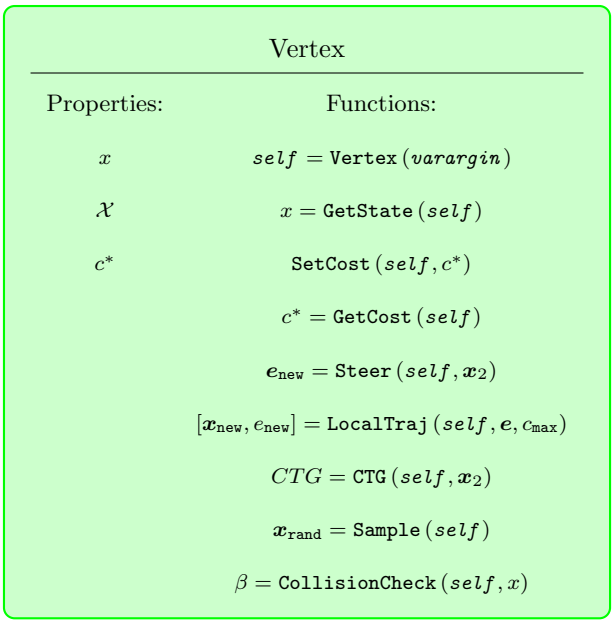


Figure 9: General vertex class.

A generalization of the vertex class is shown in Figure 9. Generally speaking, the vertex class has the following member properties:

- x: An n row single-column double precision array that describes the state of the vertex.
- $\mathcal{X}$: A double precision array that describes the bounds on the state space of the system.
- c^*: A double precision variable that describes the cost of the vertex, where the vertex cost is introduced in Definition 2.

Furthermore, the vertex class has the following member functions:

- $\texttt{Vertex}(varargin)$: This function is the class constructor, which receives all or some of the class properties from the input $varargin$ and assigns them accordingly. Also, the vertex cost is initialized to infinity.
- $\texttt{GetState}(self)$: This function returns the property x.
- $\texttt{SetCost}(self, c^*)$: The input of this function is a double precision variable c^*, which is assigned to the property c^*.
- $\texttt{GetCost}(self)$: This function returns the property c^*.
- $\texttt{Steer}(self, \boldsymbol{x}_2)$: The input of this function is the vertex object $\boldsymbol{x}_2$. This function computes the unconstrained optimal state and control input trajectories, and the unconstrained optimal final time

to steer the system from the state of the vertex object (self) to the state of the vertex object $\boldsymbol{x}_2$. This function returns the trajectory object $\boldsymbol{e}_{\texttt{new}}$ describing the unconstrained optimal trajectory.

- `LocalTraj` $(\textit{self}, \boldsymbol{e}, c_{\texttt{max}})$: The input of this function is the trajectory object $\boldsymbol{e}$ and the double precision variable $c_{\texttt{max}}$. This function computes the unconstrained optimal trajectory from the initial state of $\boldsymbol{e}$ toward the final state of $\boldsymbol{e}$ up to a maximum cost $c_{\texttt{max}}$, and returns the new vertex object $\boldsymbol{x}_{\texttt{new}}$ as the end vertex of this trajectory and the new trajectory $\boldsymbol{e}_{\texttt{new}}$.

- `CTG` $(\textit{self}, \boldsymbol{x}_2)$: The input of this function is a vertex object $\boldsymbol{x}_2$. This function calculates the unconstrained optimal cost-to-go from the the state of the vertex object (self) to the state of the vertex object property $\boldsymbol{x}_2$, and returns this value as the double precision variable CTG.

- `Sample` $(\textit{self})$: This function returns a randomly generated vertex object $\boldsymbol{x}_{\texttt{rand}}$ that has a state that lies in the region described by the bounds in the property $\mathcal{X}$.

- `CollisionCheck` $(\textit{self}, x)$: The input of this function is an n row double precision array x, where each column of x describes a state. This function evaluates if any column of x describes a state that lies outside of the bounds described by the property $\mathcal{X}$. If no state described by the columns of x lies outside of these bounds then the function returns *true*, and it returns *false* otherwise.

Trajectory	
Properties:	Functions:
$\boldsymbol{x}_1$	$\textit{self} = \texttt{Trajectory}\left(\boldsymbol{x}_1, \boldsymbol{x}_2, x^*_{1,2}, u^*_{1,2}, t^*_{1,2}, c^*_{1,2}\right)$
$\boldsymbol{x}_2$	$\boldsymbol{x}_1 = \texttt{GetStartVertex}\,(\textit{self})$
$x^*_{1,2}$	$x_2 = \texttt{GetEndVertex}\,(\textit{self})$
$u^*_{1,2}$	$x^*_{1,2} = \texttt{GetStateTrajectory}\,(\textit{self})$
$t^*_{1,2}$	$u^*_{1,2} = \texttt{GetInputTrajectory}\,(\textit{self})$
$c^*_{1,2}$	$t^*_{1,2} = \texttt{GetTime}\,(\textit{self})$
	$c^*_{1,2} = \texttt{GetCost}\,(\textit{self})$

Figure 10: Trajectory class.

The trajectory class is shown in Figure 10. The trajectory class has the following member properties:

- $\boldsymbol{x}_1$: A vertex object that is associated with the initial state of the trajectory.

- $\boldsymbol{x}_2$: A vertex object that is associated with the final state of the trajectory.

- $x^*_{1,2}$: An n row double precision array describing the discretized state trajectory to steer the system from the state of the vertex object property $\boldsymbol{x}_1$ to the state of the vertex object property $\boldsymbol{x}_2$.
- $u^*_{1,2}$: An m row double precision array describing the discretized control input trajectory to steer the system from the state of the vertex object property $\boldsymbol{x}_1$ to the state of the vertex object property $\boldsymbol{x}_2$.
- $t^*_{1,2}$: A single-row double precision array describing the discretized time span of the state and control input trajectories.
- $c^*_{1,2}$: A double precision variable describing the cost of the trajectory to steer the system from the state of the vertex object property $\boldsymbol{x}_1$ to the state of the vertex object property $\boldsymbol{x}_2$.

Furthermore, the trajectory class has the following member functions:

- `Trajectory` $\left(\boldsymbol{x}_1, \boldsymbol{x}_2, x^*_{1,2}, u^*_{1,2}, t^*_{1,2}, c^*_{1,2}\right)$: This function is the class constructor, which receives all of the class properties as inputs and assigns them accordingly.
- `GetStartVertex` (*self*): This function returns the property $\boldsymbol{x}_1$.
- `GetEndVertex` (*self*): This function returns the property $\boldsymbol{x}_2$.
- `GetStateTrajectory` (*self*): This function returns the array $x^*_{1,2}$.
- `GetInputTrajectory` (*self*): This function returns the array $u^*_{1,2}$.
- `GetTime` (*self*): This function returns the array $t^*_{1,2}$.
- `GetCost` (*self*): This function returns the property $c^*_{1,2}$.

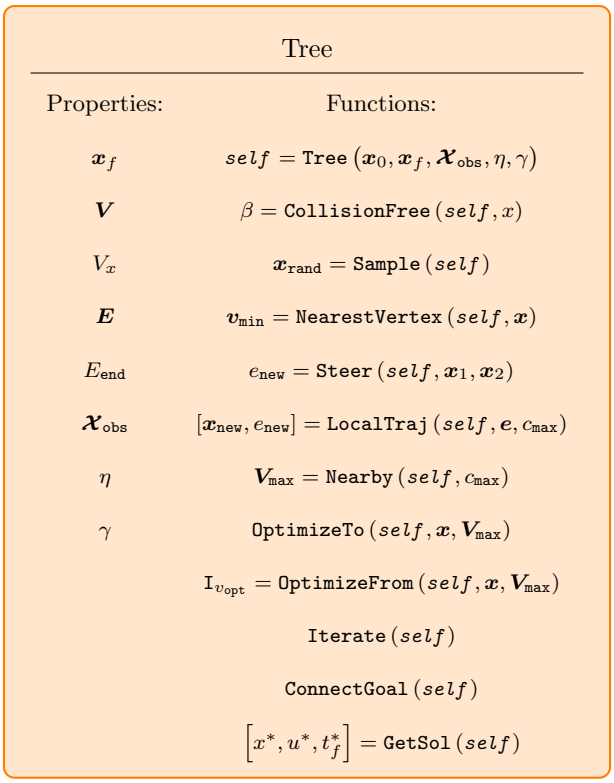


Figure 11: Tree class.

The tree class is shown in Figure 11. The tree class has the following member properties:

- $\boldsymbol{x}_f$: A vertex object that is associated with the final state of the motion planning problem.
- $\boldsymbol{V}$: A single-row multi-column array of vertex objects.
- V_x: An n row double precision array, where each column represents the state of each vertex in $\boldsymbol{V}$. This array is used to reduce the computation time required to find specific vertices.
- $\boldsymbol{E}$: A single-row multi-column array of trajectory objects.
- $E_{\texttt{end}}$: An n row double precision array, where each column represents the final state of each trajectory in $\boldsymbol{E}$. This array is used to reduce the computation time required to find specific trajectories.
- $\boldsymbol{\mathcal{X}}_{\texttt{obs}}$: A single-row multi-column array of obstacle objects.
- η: A double precision variable that describes the cost-to-go upper bound from Section 2.2.2.1.
- γ: A double precision variable that describes the scaling factor of $c_{\texttt{max}}$ from Section 2.2.2.1.

Furthermore, the tree class has the following member functions:

- `Tree` $(\boldsymbol{x}_0, \boldsymbol{x}_f, \boldsymbol{\mathcal{X}}_{\texttt{obs}}, \eta, \gamma)$: This function is the class constructor, which is shown in Algorithm 3.

- `CollisionFree` $(\textit{self}, x)$: This is the collision-checker function that was introduced in Definition 2. This function receives an n row double precision array, where each column of x describes a state. The function output is the boolean variable β. The MATLAB implementation of the collision-checker function is in Algorithm 4.

- `Sample` $(\textit{self})$: This is the sampling function that was introduced in Definition 2. The output of this function is the vertex object $\boldsymbol{x}_{\texttt{rand}}$. The MATLAB implementation of the sampling function is in Algorithm 5.

- `NearestVertex` $(\textit{self}, \boldsymbol{x})$: This is the nearest vertex function that was introduced in Definition 2. The input of this function is the vertex object $\boldsymbol{x}$ and the output is the vertex object $\boldsymbol{v}_{\texttt{min}}$. The MATLAB implementation of the sampling function is in Algorithm 6.

- `Steer` $(\textit{self}, \boldsymbol{x}_1, \boldsymbol{x}_2)$: This is the steering function that was introduced in Definition 2. The inputs of this function are the vertex object $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. This function simply invokes the steering function on $\boldsymbol{x}_1$ to obtain the unconstrained optimal trajectory to steer the system from the state of the vertex objects $\boldsymbol{x}_1$ to the state of the vertex object $\boldsymbol{x}_2$.

- `LocalTraj` $(\textit{self}, \boldsymbol{e}, c_{\texttt{max}})$: This is the local trajectory function that was introduced in Definition 2. The input of this function is the trajectory object $\boldsymbol{e}$ and an upper bound on the cost of the trajectory $c_{\texttt{max}}$. The output of this function is the vertex object $\boldsymbol{x}_{\texttt{new}}$ and the trajectory object $\boldsymbol{e}_{\texttt{new}}$. This function simply invokes the local trajectory function of the vertex class.

- `Nearby` $(\textit{self}, c_{\texttt{max}})$: This is the nearby vertices function that was introduced in Definition 2. The input of this function is the vertex object $\boldsymbol{x}$ and the double precision variable $c_{\texttt{max}}$. The output of this function is the single-row multi-column array of vertex objects $\boldsymbol{V}_{\texttt{max}}$. The MATLAB implementation of this function is in Algorithm 7.

- `OptimizeTo` $(\textit{self}, \boldsymbol{x}, \boldsymbol{V}_{\texttt{max}})$: The inputs of this function are the vertex object $\boldsymbol{x}$ and the single-row multi-column array of vertex objects $\boldsymbol{V}_{\texttt{max}}$. This function determines the optimal vertex in $\boldsymbol{V}_{\texttt{max}}$ to connect with $\boldsymbol{x}$. The MATLAB implementation of this function is in Algorithm 8.

- `OptimizeFrom` $(\textit{self}, \boldsymbol{x}, \boldsymbol{V}_{\texttt{max}})$: The inputs of this function are the vertex objects $\boldsymbol{x}$ and the single-row multi-column array of vertex objects $\boldsymbol{V}_{\texttt{max}}$. This function improves the tree by minimizing the cost of

the vertices in $\boldsymbol{V}_{\texttt{max}}$ by arriving from $\boldsymbol{x}$. The MATLAB implementation of this function is in Algorithm 9.

- `Iterate` (*self*): This function performs an iteration of the Kinodynamic RRT* algorithm. The MATLAB implementation of this function is in Algorithm 10.

- `ConnectGoal` (*self*): This function attempts to connect the vertex object property $\boldsymbol{x}_f$ with the tree. The MATLAB implementation of this function is in Algorithm 11.

- `GetSol` (*self*): This is the function used to retrieve the approximate solution obtained using Kinodynamic RRT*. This function can only be used if the `ConnectGoal` (*self*) returns true, which implies that the final state was successfully connected with the tree. The MATLAB implementation of this function is in Algorithm 12.

Algorithm 3 Tree Class: Constructor

function TREE($\boldsymbol{x}_0, \boldsymbol{x}_f, \boldsymbol{\mathcal{X}}_{\texttt{obs}}, \eta, \gamma$)
  $self.\boldsymbol{V} = \boldsymbol{x}_0$;
  $self.\boldsymbol{V}_x = \boldsymbol{x}_0$.GetState;
  $self.\boldsymbol{E} = []$;
  $self.\boldsymbol{E}_{\texttt{end}} = []$;
  $self.\boldsymbol{x}_f = \boldsymbol{x}_f$;
  $self.\boldsymbol{\mathcal{X}}_{\texttt{obs}} = \boldsymbol{\mathcal{X}}_{\texttt{obs}}$;
  $self.\eta = \eta$;
  $self.\gamma = \gamma$;
  return *self*;

Algorithm 4 Tree Class: Collision Free

function COLLISIONFREE(*self*, x)
  $\beta = self.\boldsymbol{x}_f$.CollisionCheck (x);
  if $\sim \beta$ **then**
    return *false*;
  for $i = 1 :$ length $(self.\boldsymbol{\mathcal{X}}_{\texttt{obs}})$ **do**
    $\boldsymbol{o} = self.\boldsymbol{\mathcal{X}}_{\texttt{obs}}(i)$;
    $\beta = \boldsymbol{o}$.CollisionCheck (x);
    if $\sim \beta$ **then**
      return *false*;
  return *true*

Algorithm 5 Tree Class: Sample Function

function SAMPLE(*self*)
  $\boldsymbol{x}_{\texttt{rand}} = self.\boldsymbol{x}_f$.Sample;
  $x_{\texttt{rand}} = \boldsymbol{x}_{\texttt{rand}}$.GetState;
  while $\sim self$.CollisionFree $(x_{\texttt{rand}})$ **do**
    $\boldsymbol{x}_{\texttt{rand}} = self.\boldsymbol{x}_f$.Sample;
    $x_{\texttt{rand}} = \boldsymbol{x}_{\texttt{rand}}$.GetState;
  return $\boldsymbol{x}_{\texttt{rand}}$;

Algorithm 6 Tree Class: Nearest Vertex Function

function NEARESTVERTEX(*self*, $\boldsymbol{x}$)
  $d_{\texttt{min}} = \infty$;
  for $i = 1 :$ length $(self.\boldsymbol{V})$ **do**
    $\boldsymbol{v} = self.\boldsymbol{V}(i)$;
    $d = \boldsymbol{v}$.CTG $(\boldsymbol{x})$;
    if $d < d_{\texttt{min}}$ **then**
      $d_{\texttt{min}} = d$;
      $I_{\texttt{min}} = i$;
  return $self.\boldsymbol{V}(I_{\texttt{min}})$;

Algorithm 7 Tree Class: Nearby Function

function NEARBY(*self*, $\boldsymbol{x}, c_{\texttt{max}}$)
  $I_{\texttt{max}} = []$;
  for $i = 1 :$ length $(self.\boldsymbol{V})$ **do**
    $\boldsymbol{v} = self.\boldsymbol{V}(i)$;
    $d = \boldsymbol{v}$.CTG $(\boldsymbol{x})$;
    if $d <= c_{\texttt{max}}$ **then**
      $I_{\texttt{max}}(\texttt{end} + 1) = i$;
  return $self.\boldsymbol{V}(I_{\texttt{max}})$;

Algorithm 8 Tree Class: Optimize To Function

```
1: function OptimizeTo(self, x, V_max)
2:     [∼, I_x] = min(vecnorm(self.V_x − x.GetState, 2, 1));
3:     [∼, I_e] = min(vecnorm(self.E_end − x.GetState, 2, 1));
4:     c_x = x.GetCost;
5:     for i = 1 : length(V_max) do
6:         v = V_max(i);
7:         e_new = v.Steer(x);
8:         x* = e_new.GetStateTrajectory;
9:         if self.CollisionFree(x*) then
10:            c_v = v.GetCost;
11:            c_e = e_new.GetCost;
12:            if c_v + c_e < c_x then
13:                c_x = c_v + c_e;
14:                self.V(I_x).SetCost(c_x);
15:                self.E(I_e) = e_new;
16:    return ;
```

Algorithm 9 Tree Class: Optimize From Function

```
1: function OptimizeFrom(self, x, V_max)
2:     c_x = x.GetCost;
3:     I_opt = [];
4:     for i = 1 : length(V_max) do
5:         v = V_max(i);
6:         e_new = x.Steer(v);
7:         x* = e_new.GetStateTrajectory;
8:         if self.CollisionFree(x*) then
9:             c_v = v.GetCost;
10:            c_e = e_new.GetCost;
11:            if c_x + c_e < c_v then
12:                c_v = c_x + c_e;
13:                [∼, I_v] = min(vecnorm(self.V_x − v.GetState, 2, 1));
14:                [∼, I_e] = min(vecnorm(self.E_end − v.GetState, 2, 1));
15:                self.V(I_v).SetCost(c_v);
16:                self.E(I_e) = e_new;
17:                I_opt(end + 1) = I_v;
18:    return I_opt;
```

Algorithm 10 Tree Class: Iterate Function

function ITERATE($self$)
  $\boldsymbol{x}_{\texttt{rand}} = self.\texttt{Sample}$;
  $\boldsymbol{v}_{\texttt{min}} = self.\texttt{NearestVertex}\left(\boldsymbol{x}_{\texttt{rand}}\right)$;
  $\boldsymbol{e}_{\texttt{new}} = self.\texttt{Steer}\left(\boldsymbol{v}_{\texttt{min}}, \boldsymbol{x}_{\texttt{rand}}\right)$;
  $\left[\boldsymbol{x}_{\texttt{new}}, \boldsymbol{e}_{\texttt{new}}\right] = self.\texttt{LocalTraj}\left(\boldsymbol{e}_{\texttt{new}}, self.\eta\right)$;
  $x^*_{\texttt{new}} = \boldsymbol{e}_{\texttt{new}}.\texttt{GetStateTrajectory}$;
  if $\sim self.\texttt{CollisionFree}\left(x^*_{\texttt{new}}\right)$ **then**
    return ;
  $\boldsymbol{V}\left(\texttt{end}+1\right) = \boldsymbol{x}_{\texttt{new}}$;
  $V_x\left(\texttt{end}+1\right) = \boldsymbol{x}_{\texttt{new}}.\texttt{GetState}$;
  $\boldsymbol{E}\left(\texttt{end}+1\right) = \boldsymbol{e}_{\texttt{new}}$;
  $E_{\texttt{end}}\left(\texttt{end}+1\right) = \boldsymbol{e}_{\texttt{new}}.\texttt{GetEndVertex}.\texttt{GetState}$;
  $n = \texttt{length}\left(self.\boldsymbol{x}_f.\texttt{GetState}\right)$;
  $c_{\texttt{max}} = \texttt{min}\left(self.\eta, self.\gamma\left(\frac{\log(\texttt{length}(self.\boldsymbol{V}))}{\texttt{length}(self.\boldsymbol{V})}\right)^{\frac{1}{n}}\right)$;
  $\boldsymbol{V}_{\texttt{max}} = \texttt{Nearby}\left(\boldsymbol{x}_{\texttt{new}}, c_{\texttt{max}}\right)$;
  $self.\texttt{OptimizeTo}\left(\boldsymbol{x}_{\texttt{new}}, \boldsymbol{V}_{\texttt{max}}\right)$;
  $I_{\texttt{opt}} = self.\texttt{OptimizeFrom}\left(\boldsymbol{x}_{\texttt{new}}, \boldsymbol{V}_{\texttt{max}}\right)$;
  while $\sim \texttt{isempty}\left(I_{\texttt{opt}}\right)$ **do**
    $\boldsymbol{v} = self.V\left(I_{\texttt{opt}}\left(1\right)\right)$;
    $I_{\texttt{opt}}\left(1\right) = []$;
    $\boldsymbol{V}_{\texttt{max}} = \texttt{Nearby}\left(\boldsymbol{v}_{\texttt{new}}, c_{\texttt{max}}\right)$;
    $I_{\texttt{new}} = self.\texttt{OptimizeFrom}\left(\boldsymbol{v}, \boldsymbol{V}_{\texttt{max}}\right)$;
    $I_{\texttt{opt}} = \left[I_{\texttt{opt}}; I_{\texttt{new}}\right]$;
  return ;

Algorithm 11 Tree Class: Connect Goal Function

function CONNECTGOAL($self$)
  $n = \texttt{length}\left(self.\boldsymbol{x}_f.\texttt{GetState}\right)$;
  $c_{\texttt{max}} = \texttt{min}\left(self.\eta, self.\gamma\left(\frac{\log(\texttt{length}(self.\boldsymbol{V}))}{\texttt{length}(self.\boldsymbol{V})}\right)^{\frac{1}{n}}\right)$;
  $\boldsymbol{V}_{\texttt{max}} = \texttt{Nearby}\left(self.\boldsymbol{x}_f, c_{\texttt{max}}\right)$;
  $c_f = self.\boldsymbol{x}_f.\texttt{GetCost}$;
  for $i = 1 : \texttt{length}\left(\boldsymbol{V}_{\texttt{max}}\right)$ **do**
    $\boldsymbol{v} = \boldsymbol{V}_{\texttt{max}}\left(i\right)$;
    $\boldsymbol{e}_f = \boldsymbol{v}.\texttt{Steer}\left(self.\boldsymbol{x}_f\right)$;
    $x^* = \boldsymbol{e}_f.\texttt{GetStateTrajectory}$;
    if $self.\texttt{CollisionFree}\left(x^*\right)$ **then**
      $c_v = \boldsymbol{v}.\texttt{GetCost}$;
      $c_e = \boldsymbol{e}_f.\texttt{GetCost}$;
      if $c_v + c_e < c_f$ **then**
        $c_f = c_v + c_e$;
        $self.\boldsymbol{x}_f.\texttt{SetCost}\left(c_f\right)$;
        $\boldsymbol{e}_f = \boldsymbol{e}_{\texttt{new}}$;
  if $c_f == \infty$ **then**
    return *false*;
  else
    $\boldsymbol{V}\left(\texttt{end}+1\right) = self.\boldsymbol{x}_f$;
    $V_x\left(\texttt{end}+1\right) = self.\boldsymbol{x}_f.\texttt{GetState}$;
    $\boldsymbol{E}\left(\texttt{end}+1\right) = \boldsymbol{e}_f$;
    $E_{\texttt{end}}\left(\texttt{end}+1\right) = self.\boldsymbol{x}_f.\texttt{GetState}$;
    return *true*;

Algorithm 12 Tree Class: Get Solution Function

```
function GetSol(self)
    x_0 = self.V(1).GetState;
    x_f = self.x_f.GetState;
    x* = [];
    u* = [];
    t* = [];
    v = x_f;
    while vecnorm(x_0 − v, 2, 1) ∼= 0 do
        [∼, I_e] = min(vecnorm(self.E_end − v, 2, 1));
        e = self.E(I_e);
        x* = [e.GetStateTrajectory, x*];
        u* = [e.GetInputTrajectory, u*];
        t* = [e.GetTime, t*];
        v = e.GetStartVertex.GetState;
    return [x*, u*, t*];
```

2.3 Optimal Trajectory Planning For a Double Integrator With Drift

This section addresses optimal trajectory planning of a double integrator with drift such that the optimal trajectory minimizes a cost trading off control effort and time. The system's state is described by the vector

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{v} \end{bmatrix}, \tag{57}$$

where $\boldsymbol{p} = [p_x, p_y, p_z]^T \in \mathbb{R}^3$ is the system's position in meters, and $\boldsymbol{v} = [v_x, v_y, v_z]^T \in \mathbb{R}^3$ is its velocity in meters per second. The system's control input is denoted by $\boldsymbol{u} = [u_x, u_y, u_z]^T \in \mathbb{R}^3$, which is the system's acceleration in meters per second squared. The state transition equation of the system is

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} + \boldsymbol{c}, \quad A = \begin{bmatrix} 0_3 & I_3 \\ 0_3 & 0_3 \end{bmatrix}, \quad B = \begin{bmatrix} 0_3 \\ I_3 \end{bmatrix}, \tag{58}$$

where $\boldsymbol{c} = [\boldsymbol{c}_v^T, \boldsymbol{c}_a^T]^T$, $\boldsymbol{c}_v, \boldsymbol{c}_a \in \mathbb{R}^3$ are the velocity and acceleration vectors of the drift term, and where 0_k and I_k denote a $k \times k$ zero and identity matrix, respectively, for $k \in \mathbb{N}_{>0}$. Given a control input trajectory $\boldsymbol{u}(t)$, a state trajectory $\boldsymbol{x}(t)$, and a final time t_f, the cost of the trajectory is evaluated according to (26), where $R \in \mathbb{R}^{3\times 3}$ is assumed to be symmetric and positive definite and $C_I \in \mathbb{R}_{>0}$.

Optimal trajectory planning for a double integrator with drift is not a new topic in the open literature. For example, the optimal trajectory planning problem in (41) for a cost function (26) has been addressed in [59] for general affine systems when $\boldsymbol{\mathcal{X}}_{\texttt{free}} = \mathbb{R}^n$, which can be solved directly using optimal control theory. The optimal final time of an optimal trajectory is obtained as the root of a polynomial, which is

a quartic for systems of the form (58). In [66] the problem described by (41) with the cost function (26) where $R = I_3$ is addressed for the particular system described by (58) when $\boldsymbol{c} = [0, 0, 0, 0, 0, -g]^T$ and $\boldsymbol{\mathcal{X}}_{\texttt{free}}$ is convex and constrained by a maximum velocity. The same problem is addressed for a general system of the form (58) in [67] where it is assumed that $R = I_3$. Furthermore, the work in references [47, 48] obtains an approximate solution of (41) for the cost function in (26) and for the system described by (58) when $\boldsymbol{\mathcal{X}}_{\texttt{free}}$ is a non-convex bounded space. This is accomplished by combining optimal control theory with Kinodynamic RRT* to search for the optimal trajectory from a given initial state to a given final state as a sequence of unconstrained optimal trajectory segments through $\boldsymbol{\mathcal{X}}_{\texttt{free}}$. The final time of the unconstrained optimal trajectories found in references [47, 48, 66] is assumed to be fixed. For problems with free final time the optimal final time is obtained using numerical optimization to evaluate the time that minimizes the cost function. What is proposed in this section is to apply the result of [59] to Kinodynamic RRT*, which assumes a free final time in the problem of the unconstrained optimal trajectory planning problem. As a result, the optimal final time of optimal trajectories can be obtained as the root of the Hamiltonian, which avoids the need of numerical optimization to minimize the cost function.

2.3.1 Optimal Trajectory Planning in Unconstrained Convex State Spaces

The problem solved in this section is that of (27) for the system in (58). One approach to solve this problem is to apply a change of coordinates to transform the affine system into a linear system. However, in [59] it is stated and proved that "whenever $\boldsymbol{x}_f$ is chosen such that $A\boldsymbol{x}_f + \boldsymbol{c}$ is not in the range of B the change of coordinates is ineffective". This implies that, for a change of coordinates to be effective, then there must exist a $\boldsymbol{z} \in \mathbb{R}^3$ such that

$$B\boldsymbol{z} = A\boldsymbol{x}_f + \boldsymbol{c}, \tag{59}$$

which is equivalent to

$$\begin{bmatrix} 0_3 \\ I_3 \end{bmatrix} \boldsymbol{z} = \begin{bmatrix} \boldsymbol{v}_f + \boldsymbol{c}_v \\ \boldsymbol{c}_a \end{bmatrix}. \tag{60}$$

This result implies that a change of coordinates is only possible when $\boldsymbol{v}_f = -\boldsymbol{c}_v$. However, this is a very specific case for a small subset of states in $\boldsymbol{\mathcal{X}}_{\texttt{free}}$, for example when there is wind and the final desired velocity compensates the wind velocity. Therefore, the general case for all $\boldsymbol{x}_f \in \boldsymbol{\mathcal{X}}_{\texttt{free}}$ is considered, which includes $\boldsymbol{v}_f \neq -\boldsymbol{c}_v$ for which a change of coordinates is ineffective.

Definition 3. *Controllability [59]: For a system of the form*

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} + \boldsymbol{c}, \tag{61}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $\boldsymbol{c} \in \mathbb{R}^n$, $n, m \in \mathbb{N}_{>0}$, *the controllability matrix is defined as*

$$\mathcal{C} = \begin{bmatrix} B & AB & A^2B & \dots & A^{n-1}B \end{bmatrix}. \tag{62}$$

If the matrix $\mathcal{C}$ *has maximum rank* n *then the system, or equivalently the pair* (A, B), *is controllable.* □

It can be seen from Definition 3 that, for the system in (58), the controllability matrix is

$$\mathcal{C} = \begin{bmatrix} 0_3 & I_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ I_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \end{bmatrix}, \tag{63}$$

which has maximum rank. Since the controllability matrix for the system in (58) is full rank the system is controllable. As a consequence of this, the solution of (27) is presented using Theorem 5, which is detailed in Appendix B. Given $\boldsymbol{x}_0$, $\boldsymbol{x}_f$, and $\boldsymbol{c}$, the optimal control input to steer the system from $\boldsymbol{x}_0$ to $\boldsymbol{x}_f$ through $\boldsymbol{\mathcal{X}}$ is given by equation (28) as

$$\boldsymbol{u}^*(t) = \begin{bmatrix} 6\left(\frac{2t-t_f^*}{t_f^{*3}}\right) I_3 & 2\left(\frac{3t-2t_f^*}{t_f^{*2}}\right) I_3 & -6\left(\frac{2t-t_f^*}{t_f^{*3}}\right) I_3 & 2\left(\frac{3t-t_f^*}{t_f^{*2}}\right) I_3 & 6\left(\frac{2t-t_f^*}{t_f^{*2}}\right) I_3 & -I_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}. \tag{64}$$

Additionally, the cost of the optimal trajectory is given by equation (26) as

$$C\left(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*\right) = \frac{1}{2} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix} \frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & -\frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & \frac{12}{t_f^{*2}}R & 0_3 \\ \frac{6}{t_f^{*2}}R & \frac{4}{t_f^*}R & -\frac{6}{t_f^{*2}}R & \frac{2}{t_f^*}R & \frac{6}{t_f^*}R & R \\ -\frac{12}{t_f^{*3}}R & -\frac{6}{t_f^{*2}}R & \frac{12}{t_f^{*3}}R & -\frac{6}{t_f^{*2}}R & -\frac{12}{t_f^{*2}}R & 0_3 \\ \frac{6}{t_f^{*2}}R & \frac{2}{t_f^*}R & -\frac{6}{t_f^{*2}}R & \frac{4}{t_f^*}R & \frac{6}{t_f^*}R & -R \\ \frac{12}{t_f^{*2}}R & \frac{6}{t_f^*}R & -\frac{12}{t_f^{*2}}R & \frac{6}{t_f^*}R & \frac{12}{t_f^*}R & 0_3 \\ 0_3 & R & 0_3 & -R & 0_3 & t_f^*R \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix} + t_f^* C_I. \tag{65}$$

Furthermore, the Hamiltonian at t_f^* is given by equation (31) as

$$H\left(t_f^*\right) = \frac{1}{t_f^{*^4}}\left(C_I t_f^{*^4} + \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix} -18R & -6t_f^*R & 18R & -6t_f^*R & -12t_f^*R & 0_3 \\ -6t_f^*R & -2t_f^{*^2}R & 6t_f^*R & -t_f^{*^2}R & -3t_f^{*^2}R & 0_3 \\ 18R & 6t_f^*R & -18R & 6t_f^*R & 12t_f^*R & 0_3 \\ -6t_f^*R & -t_f^{*^2}R & 6t_f^*R & -2t_f^{*^2}R & -3t_f^{*^2}R & 0_3 \\ -12t_f^*R & -3t_f^{*^2}R & 12t_f^*R & -3t_f^{*^2}R & -6t_f^{*^2}R & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & \frac{t_f^{*^4}}{2}R \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}\right). \quad (66)$$

The derivation of the results given by (64), (65), and (66) is detailed in Appendix B.

A similar problem is solved in [67] when $R = I_3$ and $c_v = 0$, and where c_v may be time-variant. In [67] the optimal final time is obtained by minimizing the cost function, which is accomplished by finding the positive real roots of the derivative of the cost function with respect to t_f^* that result in the second derivative of the cost function with respect to t_f^* being positive. When $R = I_3$, $c_v = 0$, and constant $\dot{c}_v$ the optimal control input, optimal trajectory cost, and the Hamiltonian given in (64), (65), and (66) are the same as those in [67] for the same conditions. The result of [67] is more general than the result of this section in the sense that it provides solutions for time-variant affine terms when $R = I_3$ and $c_v = 0$. However, the result of this section is more general than that in [67] in the sense that the solution is provided for a general symmetric and positive definite control input weighting matrix R and time-invariant affine terms where $c_v \neq 0$. In references [47, 48, 66] numerical optimization is used to evaluate the optimal final time as the positive final time that minimizes (65). However, in [59] it is shown that the optimal final time of the optimal trajectory is a real positive root of (66). As a consequence of this result, a finite number of candidate final times can be computed as the real positive roots of (66). Then, the optimal final time can be obtained as the candidate that minimizes (65). A discussion on the existence of real roots of (66) is presented in Appendix A, where under certain circumstances these roots may be computed analytically.

The proposed technique to obtain the optimal final time of the optimal trajectory proposed in [59] is compared with the numerical approach used in references [47,48,66]. For this comparison $10,000$ simulations are performed where each component of $\boldsymbol{x}_0$, $\boldsymbol{x}_f$, and $\boldsymbol{c}$, are randomly drawn from $[-1000, 1000]$ for each simulation. Additionally, the cost index is randomly drawn from $(0, 10]$ for each simulation, and the control input weighting matrix is obtained according to

$$R = I_3 + \tilde{R}\tilde{R}^T, \quad (67)$$

where each element of the matrix $\tilde{R}$ is randomly drawn from $[0, 1]$. The simulation is performed in MATLAB R2018a on an Intel Core i5 2.9 GHz processor with 16GB of RAM. The numerical solver used to evaluate the performance of the techniques of [47, 48, 66] is *fmincon*, which uses an interior point algorithm [68] to minimize (65) subject to $0 \leq t_f^* < \infty$. Furthermore, the roots of (66) are obtained using the *roots* function. In Theorem 16 of Appendix A it is shown that under certain circumstances the real roots of (66) can be obtained analytically. For the cases where this is not possible, the roots are obtained using the *roots* function, which finds the roots of a polynomial as the eigenvalues of the *companion matrix* [69]. When using the approach discussed in [59], the optimal final time for the optimal trajectory is obtained in an average of 6.97×10^{-4} seconds. By contrast, the numerical approach used in [47, 48, 66] required on average 0.032 seconds to obtain the optimal final time. Therefore, the method from [59] required on average only 2.2% the computation time to obtain the optimal final time of a trajectory when compared with the methods used in [47, 48, 66].

Figure 12 shows the effect of $\boldsymbol{c}_v$ on the position vector of the optimal state trajectory when gravity is present. For this example the initial state, final state, and affine term are

$$\begin{aligned} \boldsymbol{x}_0 &= [0, 0, 0, 0, 0, 0]^T \\ \boldsymbol{x}_f &= [10, 10, 10, 0, 0, 0]^T \\ \boldsymbol{c} &= [\boldsymbol{c}_v^T, 0, 0, -9.8]^T \end{aligned} , \tag{68}$$

where the constant wind velocity vector $\boldsymbol{c}_v$ for each state trajectory is shown in the legend of Figure 12. Note that the direction of each $\boldsymbol{c}_v$ vector is the same and the magnitude is decreased from $\sqrt{350}$ to 0. The initial position is shown as the green diamond, the final position is shown as the red diamond, the direction of the wind is shown by the black arrows, and the position vector of each optimal trajectory is shown as a coloured curve.

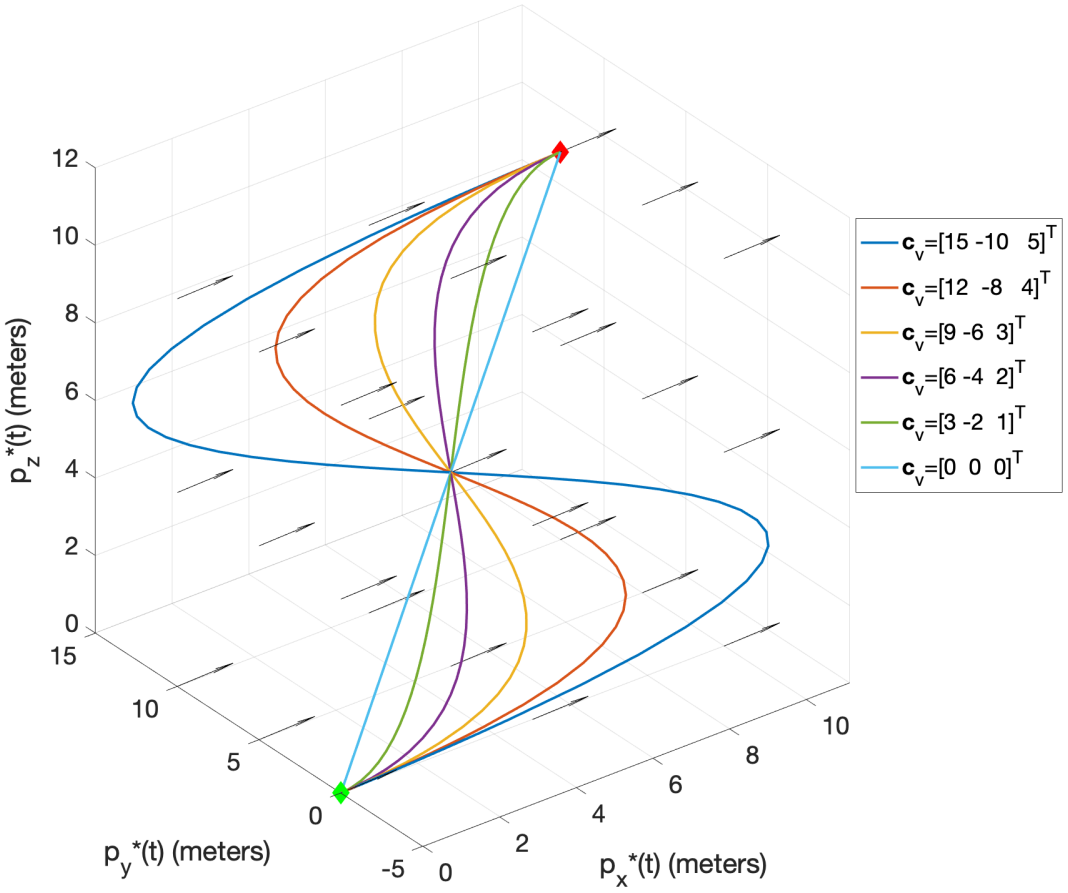


Figure 12: Position vectors of optimal state trajectories of double integrator and constant velocity winds.

2.3.2 Optimal Trajectory Planning in Non-Convex State Spaces

Recall from (41) that, when $\boldsymbol{\mathcal{X}}_{\texttt{free}} \neq \mathbb{R}^n$, the optimal trajectory planning problem addressed in Section 2.3.1 can be described by

$$\begin{aligned} \min_{\boldsymbol{u}(t),t_f} \quad & \int_0^{t_f} \left(\tfrac{1}{2} \boldsymbol{u}(\tau)^T R \boldsymbol{u}(\tau) + C_I \right) d\tau \\ \texttt{s.t.} \quad & \dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} + \boldsymbol{c} \\ & \boldsymbol{x} : [0, t_f] \rightarrow \boldsymbol{\mathcal{X}}_{\texttt{free}} \\ & \boldsymbol{\psi}(\boldsymbol{x}_0, \boldsymbol{x}_f) = \boldsymbol{0} \end{aligned} , \tag{69}$$

where $\boldsymbol{\psi}(\boldsymbol{x}_0, \boldsymbol{x}_f)$ is defined in (4).

Three functions must be defined so that the Kinodynamic RRT* may be used to solve (69): the cost-to-go function, the steering function, and the local trajectory function. The results obtained in Section 2.3.1 may be used to design these functions. Given an initial state $\boldsymbol{x}_1 \in \boldsymbol{\mathcal{X}}$ and a final state $\boldsymbol{x}_2 \in \boldsymbol{\mathcal{X}}$, let $\boldsymbol{u}^*_{1,2}(t)$, $\boldsymbol{x}^*_{1,2}(t)$, and $t^*_{f_{1,2}}$ denote the optimal control input, state trajectories and the optimal final time to steer the system from $\boldsymbol{x}_1$ to $\boldsymbol{x}_2$. For the cost-to-go and steering functions $t^*_{f_{1,2}}$ is obtained from Definition 2 as the

real positive solution of

$$\frac{1}{t_{f_{1,2}}^{*4}}\left(C_I t_{f_{1,2}}^{*4}+\begin{bmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\\ \boldsymbol{c}\end{bmatrix}^T\begin{bmatrix} -18R & -6t_{f_{1,2}}^{*}R & 18R & -6t_{f_{1,2}}^{*}R & -12t_{f_{1,2}}^{*}R & 0_3\\ -6t_{f_{1,2}}^{*}R & -2t_{f_{1,2}}^{*^2}R & 6t_{f_{1,2}}^{*}R & -t_{f_{1,2}}^{*^2}R & -3t_{f_{1,2}}^{*^2}R & 0_3\\ 18R & 6t_{f_{1,2}}^{*}R & -18R & 6t_{f_{1,2}}^{*}R & 12t_{f_{1,2}}^{*}R & 0_3\\ -6t_{f_{1,2}}^{*}R & -t_{f_{1,2}}^{*^2}R & 6t_{f_{1,2}}^{*}R & -2t_{f_{1,2}}^{*^2}R & -3t_{f_{1,2}}^{*^2}R & 0_3\\ -12t_{f_{1,2}}^{*}R & -3t_{f_{1,2}}^{*^2}R & 12t_{f_{1,2}}^{*}R & -3t_{f_{1,2}}^{*^2}R & -6t_{f_{1,2}}^{*^2}R & 0_3\\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & \frac{t_{f_{1,2}}^{*4}}{2}R\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\\ \boldsymbol{c}\end{bmatrix}\right)=0, \quad (70)$$

which minimizes

$$C\left(\boldsymbol{x}_{1,2}^*(t),\boldsymbol{u}_{1,2}^*(t),t_{f_{1,2}}^*\right)= \frac{1}{2}\begin{bmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\\ \boldsymbol{c}\end{bmatrix}^T\begin{bmatrix} \frac{12}{t_{f_{1,2}}^{*3}}R & \frac{6}{t_{f_{1,2}}^{*2}}R & -\frac{12}{t_{f_{1,2}}^{*3}}R & \frac{6}{t_{f_{1,2}}^{*2}}R & \frac{12}{t_{f_{1,2}}^{*2}}R & 0_3\\ \frac{6}{t_{f_{1,2}}^{*2}}R & \frac{4}{t_{f_{1,2}}^{*}}R & -\frac{6}{t_{f_{1,2}}^{*2}}R & \frac{2}{t_{f_{1,2}}^{*}}R & \frac{6}{t_{f_{1,2}}^{*}}R & R\\ -\frac{12}{t_{f_{1,2}}^{*3}}R & -\frac{6}{t_{f_{1,2}}^{*2}}R & \frac{12}{t_{f_{1,2}}^{*3}}R & -\frac{6}{t_{f_{1,2}}^{*2}}R & -\frac{12}{t_{f_{1,2}}^{*2}}R & 0_3\\ \frac{6}{t_{f_{1,2}}^{*2}}R & \frac{2}{t_{f_{1,2}}^{*}}R & -\frac{6}{t_{f_{1,2}}^{*2}}R & \frac{4}{t_{f_{1,2}}^{*}}R & \frac{6}{t_{f_{1,2}}^{*}}R & -R\\ \frac{12}{t_{f_{1,2}}^{*2}}R & \frac{6}{t_{f_{1,2}}^{*}}R & -\frac{12}{t_{f_{1,2}}^{*2}}R & \frac{6}{t_{f_{1,2}}^{*}}R & \frac{12}{t_{f_{1,2}}^{*}}R & 0_3\\ 0_3 & R & 0_3 & -R & 0_3 & t_{f_{1,2}}^{*}R\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\\ \boldsymbol{c}\end{bmatrix}+t_{f_{1,2}}^{*}C_I \quad (71)$$

The derivation of the results given by (70) and (71) is detailed in Appendix B. A discussion on the existence of real solutions of (70) is presented in Appendix A, where under certain circumstances these roots may be computed analytically. Once $t_{f_{1,2}}^*$ is computed, the cost-to-go function returns the cost given by (71). The steering function computes $\boldsymbol{u}_{1,2}^*(t)$ according to (64), which yields

$$\boldsymbol{u}_{1,2}^*(t)=\begin{bmatrix}6\left(\frac{2t-t_{f_{1,2}}^*}{t_{f_{1,2}}^{*3}}\right)I_3 & 2\left(\frac{3t-2t_{f_{1,2}}^*}{t_{f_{1,2}}^{*2}}\right)I_3 & -6\left(\frac{2t-t_{f_{1,2}}^*}{t_{f_{1,2}}^{*3}}\right)I_3 & 2\left(\frac{3t-t_{f_{1,2}}^*}{t_{f_{1,2}}^{*2}}\right)I_3 & 6\left(\frac{2t-t_{f_{1,2}}^*}{t_{f_{1,2}}^{*2}}\right)I_3 & -I_3\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_1\\ \boldsymbol{x}_2\\ \boldsymbol{c}\end{bmatrix}. \quad (72)$$

The optimal state trajectory $\boldsymbol{x}_{1,2}^*(t)$ is then obtained by replacing $\boldsymbol{u}$ by $\boldsymbol{u}_{1,2}^*(t)$ in (58) and integrating from $t=0$ to $t=t_{f_{1,2}}^*$. Given $\boldsymbol{u}_{1,2}^*(t)$, $\boldsymbol{x}_{1,2}^*(t)$, and $t_{f_{1,2}}^*$, it can be seen from equation (19) that the cost of the optimal trajectory from $\boldsymbol{x}_1$ to $\boldsymbol{x}_{1,2}^*(\tilde{t}_{f_{1,2}}^*)$, where $\tilde{t}_{f_{1,2}}^*\in[0,t_{f_{1,2}}^*]$, is evaluated according to

$$C\left(\boldsymbol{x}_{1,2}^*(t),\boldsymbol{u}_{1,2}^*(t),\tilde{t}_{f_{1,2}}^*\right)=\int_0^{\tilde{t}_{f_{1,2}}^*}\left(\frac{1}{2}\boldsymbol{u}_{1,2}^*(\tau)^T R\boldsymbol{u}_{1,2}^*(\tau)+C_I\right)d\tau. \quad (73)$$

From equation (35) it can be seen that

$$C(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), \tilde{t}^*_{f_{1,2}}) = \frac{1}{2}\boldsymbol{X}^T_{c_{1,2}} D(\tilde{t}^*_{f_{1,2}})\boldsymbol{X}_{c_{1,2}} + \tilde{t}^*_{f_{1,2}} C_I, \tag{74}$$

where

$$\boldsymbol{X}_{c_{1,2}} = e^{A^T t^*_{f_{1,2}}} \boldsymbol{\lambda}^*_{1,2}\left(t^*_f\right) \quad , \quad D\left(\tilde{t}^*_{f_{1,2}}\right) = \begin{bmatrix} \frac{\tilde{t}^{*^3}_{f_{1,2}}}{3} R^{-1} & -\frac{\tilde{t}^{*^2}_{f_{1,2}}}{2} R^{-1} \\ -\frac{\tilde{t}^{*^2}_{f_{1,2}}}{2} R^{-1} & \tilde{t}^*_{f_{1,2}} R^{-1} \end{bmatrix}, \tag{75}$$

and where the final costates are obtained using equation (29) such that

$$\boldsymbol{\lambda}^*_{1,2}\left(t^*_{f_{1,2}}\right) = \begin{bmatrix} \frac{12}{t^{*^3}_{f_{1,2}}} R & \frac{6}{t^{*^2}_{f_{1,2}}} R & -\frac{12}{t^{*^3}_{f_{1,2}}} R & \frac{6}{t^{*^2}_{f_{1,2}}} R & \frac{12}{t^{*^2}_{f_{1,2}}} R & 0_3 \\ -\frac{6}{t^{*^2}_{f_{1,2}}} R & -\frac{2}{t^*_{f_{1,2}}} R & \frac{6}{t^{*^2}_{f_{1,2}}} R & -\frac{4}{t^*_{f_{1,2}}} R & -\frac{6}{t^*_{f_{1,2}}} R & R \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{c} \end{bmatrix}. \tag{76}$$

Observe that $D(\tilde{t}^*_{f_{1,2}})$ in (75) can be expressed as

$$D\left(\tilde{t}^*_{f_{1,2}}\right) = \frac{1}{3}\begin{bmatrix} R^{-1} & 0_3 \\ 0_3 & 0_3 \end{bmatrix} \tilde{t}^{*^3}_{f_{1,2}} - \frac{1}{2}\begin{bmatrix} 0_3 & R^{-1} \\ R^{-1} & 0_3 \end{bmatrix} \tilde{t}^{*^2}_{f_{1,2}} + \begin{bmatrix} 0_3 & 0_3 \\ 0_3 & R^{-1} \end{bmatrix} \tilde{t}^*_{f_{1,2}}, \tag{77}$$

which allows to recast the cost in (74) as the polynomial

$$C\left(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), \tilde{t}^*_{f_{1,2}}\right) = C_3 \tilde{t}^{*^3}_{f_{1,2}} + C_2 \tilde{t}^{*^2}_{f_{1,2}} + C_1 \tilde{t}^*_{f_{1,2}}, \tag{78}$$

where

$$C_3 = \tfrac{1}{6}\boldsymbol{X}^T_{c_{1,2}} \begin{bmatrix} R^{-1} & 0_3 \\ 0_3 & 0_3 \end{bmatrix} \boldsymbol{X}_{c_{1,2}}$$

$$C_2 = -\tfrac{1}{4}\boldsymbol{X}^T_{c_{1,2}} \begin{bmatrix} 0_3 & R^{-1} \\ R^{-1} & 0_3 \end{bmatrix} \boldsymbol{X}_{c_{1,2}} \quad . \tag{79}$$

$$C_1 = \tfrac{1}{2}\boldsymbol{X}^T_{c_{1,2}} \begin{bmatrix} 0_3 & 0_3 \\ 0_3 & R^{-1} \end{bmatrix} \boldsymbol{X}_{c_{1,2}} + C_I$$

Lemma 1. *If R is symmetric and positive definite and $C_I > 0$ then the coefficients in* (79) *satisfy*

$$\begin{array}{rcl} C_3 & \geq & 0 \\ C_1 & > & 0 \end{array}. \tag{80}$$

Proof. It can be seen from (79) that C_3 is a quadratic form and C_1 is the sum of a quadratic form and C_I. Given the assumption that R is symmetric and positive definite it can be seen that the matrices

$$\begin{bmatrix} R^{-1} & 0_3 \\ 0_3 & 0_3 \end{bmatrix}, \begin{bmatrix} 0_3 & 0_3 \\ 0_3 & R^{-1} \end{bmatrix} \tag{81}$$

are positive semi-definite. Given this result, and the assumption that $C_I > 0$, it can be seen that $C_3 \geq 0$ and $C_1 > 0$. □

Given $\boldsymbol{u}^*_{1,2}(t)$, $\boldsymbol{x}^*_{1,2}(t)$, $t^*_{f_{1,2}}$, and a maximum cost-to-go $\eta \in \mathbb{R}_{>0}$, the local trajectory function from Definition 2 calculates the time $\tilde{t}^*_{f_{1,2}}$ as the non-negative real solution of

$$C_3 \tilde{t}^{*^3}_{f_{1,2}} + C_2 \tilde{t}^{*^2}_{f_{1,2}} + C_1 \tilde{t}^*_{f_{1,2}} - \tilde{\eta} = 0, \tag{82}$$

where

$$\tilde{\eta} = \min \left\{ C\left(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), t^*_{f_{1,2}}\right), \eta \right\}. \tag{83}$$

The local trajectory function from Definition 2 calculates $\tilde{t}^*_{f_{1,2}}$ as the smallest non-negative real solution of (82) and returns the optimal state and control input trajectories, $\tilde{\boldsymbol{x}}^*_{1,2}(t) = \boldsymbol{x}^*_{1,2}(t)$ and $\tilde{\boldsymbol{u}}^*_{1,2}(t) = \boldsymbol{u}^*_{1,2}(t)$, $\forall\, t \in [0, \tilde{t}^*_{f_{1,2}}]$, and the optimal final time $\tilde{t}^*_{f_{1,2}}$. Evidently, if $\tilde{\eta} = 0$ then it can be seen that $\tilde{t}^*_{f_{1,2}} = 0$ is the smallest non-negative real solution of (82). Furthermore, it can be seen from Theorem 6 that $C\left(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), t^*_{f_{1,2}}\right) \geq 0$. Therefore, given the assumption that $\eta > 0$, it can be seen that $\tilde{\eta} \geq 0$. The existence of non-negative real solutions of (82) is now studied when $\tilde{\eta} > 0$.

Lemma 2. *Descarte's Rule of Signs [70]: Let $f(x)$ be a polynomial described by*

$$f(x) = \sum_{k=0}^{m} a_k x^k,\ a_k \in \mathbb{R}\ \forall\ k \in \{0, 1, \ldots, m\}. \tag{84}$$

The number of positive real zeros of $f(x)$ is either equal to the number of variations of nonzero coefficient signs in the sequence $a_0, a_1, \ldots, a_m$ or less than that by an even number. Additionally, if the number of sign changes is zero, then the number of positive real roots is zero. The number of real negative roots is obtained

similarly for the coefficients of $f(-x)$.

Theorem 9. *If* R *is symmetric and positive definite,* $C_I > 0$*, and* $\tilde{\eta} > 0$ *then there exists at least one positive real solution of* (82).

Proof. It can be seen from Lemma 1 that $C_3 \geq 0$ and $C_1 > 0$. Descarte's rule of signs from Lemma 2 can be used to determine the number of positive real roots in (82), which are shown in Table 1.

Case	$C_3 > 0$ $C_2 > 0$	$C_3 > 0$ $C_2 = 0$	$C_3 > 0$ $C_2 < 0$	$C_3 = 0$ $C_2 > 0$	$C_3 = 0$ $C_2 = 0$	$C_3 = 0$ $C_2 < 0$
Positive Real Roots	1	1	1 or 3	1	1	0 or 2

Table 1: Double integrator with drift: local trajectory function roots.

It can be seen from Table 1 that all cases except for $C_3 = 0$, $C_2 < 0$ have at least one positive real root. This case is now studied.

If $C_3 = 0$ then (82) simplifies to

$$C_2 \tilde{t}_{f_{1,2}}^{*^2} + C_1 \tilde{t}_{f_{1,2}}^{*} - \tilde{\eta} = 0, \tag{85}$$

which has the solutions

$$\tilde{t}_{f_{1,2}}^{*} = \frac{-C_1}{2C_2} \pm \frac{\sqrt{C_1^2 + 4C_2\tilde{\eta}}}{2C_2}. \tag{86}$$

It can be seen that the solutions in (86) are real if $C_1^2 + 4C_2\tilde{\eta} \geq 0$. Observe from Theorem 6 that $C(\boldsymbol{x}_{1,2}^{*}(t), \boldsymbol{u}_{1,2}^{*}(t), t_{f_{1,2}}^{*}) \geq 0$. Given the assumption that $\tilde{\eta} > 0$, it can be seen that, if $C_2 \geq 0$, then $C_1^2 + 4C_2\tilde{\eta} > 0$. Observe that, if $C_3 = 0$ and $C_2 < 0$, then (78) may be expressed as

$$C^* = -\left|C_2\right| \tilde{t}_{f_{1,2}}^{*^2} + C_1 \tilde{t}_{f_{1,2}}^{*}, \tag{87}$$

where $|\cdot|$ denotes the absolute value. Furthermore, if $C_2 < 0$ then

$$C_1^2 + 4C_2\tilde{\eta} = C_1^2 - 4\left|C_2\right|\tilde{\eta}. \tag{88}$$

Observe from (83) and (87) that

$$0 \leq \tilde{\eta} \leq C^* \Rightarrow 0 \leq \tilde{\eta} \leq -\left|C_2\right| \tilde{t}_{f_{1,2}}^{*^2} + C_1 \tilde{t}_{f_{1,2}}^{*}, \tag{89}$$

which can be used with (88) to yield

$$C_1^2 + 4C_2\tilde{\eta} = C_1^2 - 4\left|C_2\right|\tilde{\eta} \geq C_1^2 + 4\left|C_2\right|^2 \tilde{t}_{f_{1,2}}^{*^2} - 4C_1\left|C_2\right|\tilde{t}_{f_{1,2}}^{*}. \tag{90}$$

It can be seen that

$$C_1^2 + 4\left|C_2\right|^2 \tilde{t}_{f_{1,2}}^{*^2} - 4C_1\left|C_2\right|\tilde{t}_{f_{1,2}}^{*} = \left(C_1 - 2\left|C_2\right|\tilde{t}_{f_{1,2}}^{*}\right)^2, \tag{91}$$

which yields the inequality

$$C_1^2 + 4C_2\tilde{\eta} \geq \left(C_1 - 2\left|C_2\right|\tilde{t}_{f_{1,2}}^{*}\right)^2 \geq 0. \tag{92}$$

Descarte's theorem from Lemma 2 can then be used to determine the number of negative real roots in (85) when $C_2 < 0$, which is 0. Since both of the roots of (82) for the case $C_3 = 0$ and $C_2 < 0$ have been proven to be real, and since it was shown that neither of these roots are negative, then both roots must be positive real. Combining this result with those outlined in Table 1 proves that if R is symmetric and positive definite, $C_I > 0$, $\tilde{\eta} > 0$, then the solutions of (82) are real positive numbers. □

2.3.2.1 Validation and Study of Optimality

This section validates and studies the optimality of Kinodynamic RRT* for a simple one dimensional double integrator problem for which an analytical solution can be obtained. The system's state space is described by

$$\boldsymbol{\mathcal{X}} = \left\{ \boldsymbol{x} \in \mathbb{R}^6 : \begin{array}{c} -1 \leq p_x \leq 2,\ p_y = 0,\ p_z = 0, \\ \left|v_x\right| \leq 1,\ v_y = 0,\ v_z = 0 \end{array} \right\} \tag{93}$$

and the obstacle space is described by $\boldsymbol{\mathcal{X}} = \varnothing$, which implies that $\boldsymbol{\mathcal{X}}_{\texttt{free}} = \boldsymbol{\mathcal{X}}$. It can be seen from (172) that this problem is identical to the double integrator with drift, where the state space is constrained to a plane in $\mathbb{R}^6$. The position of the system lies along the x-axis between -1 and 2, and its velocity is constrained to be between -1 and 1. Let the initial and final state be described by

$$\begin{array}{rcl} \boldsymbol{x}_0 & = & \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \\ \boldsymbol{x}_f & = & \begin{bmatrix} p_f & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \end{array}, \tag{94}$$

where $p_f \neq 0$. Also, let the drift term be

$$\boldsymbol{c} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T. \tag{95}$$

Finally, the control input weighting matrix is $R = I_3$.

The optimal control solution is obtained as follows. The Hamiltonian at $t = t_f^*$ is evaluated according to (70), which yields

$$H\left(t_f^*\right) = \frac{1}{t_f^{*4}}\left(C_I t_f^{*4} - 18p_f^2\right) = 0. \tag{96}$$

The optimal final time is therefore

$$t_f^* = \sqrt[4]{\frac{18p_f^2}{C_I}}. \tag{97}$$

The cost of the optimal can then be obtained according to (71), which yields the optimal trajectory cost

$$C\left(\boldsymbol{x}_{1,2}^*(t), \boldsymbol{u}_{1,2}^*(t), \tilde{t}_{f_{1,2}}^*\right) = \frac{C_I t_f^{*4} + 6p_f^2}{t_f^{*3}}\ \frac{m^2}{s^3}. \tag{98}$$

The optimal control input can then be obtained according to (64), which yields

$$\boldsymbol{u}^*\left(t\right) = \begin{bmatrix} \frac{6p_f t_f^* - 12p_f t}{t_f^{*3}} \\ 0 \\ 0 \end{bmatrix}. \tag{99}$$

Therefore, it can be seen that p_y, p_z, v_y, and v_z are equal to zero $\forall\ t \in [0, t_f^*]$, which confirms that the system remains on the plane of p_x and v_x. Furthermore, it can be seen that

$$\dot{p}_x\left(t\right) = v_x, \qquad \dot{v}_x\left(t\right) = \frac{6p_f t_f^* - 12p_f t}{t_f^{*3}}\ . \tag{100}$$

Integrating with respect to time yields the trajectories

$$p_x\left(t\right) = \frac{3p_f t_f^* t^2 - 2p_f t^3}{t_f^{*3}}, \qquad v_x\left(t\right) = \frac{6p_f t_f^* t - 6p_f t^2}{t_f^{*3}}\ . \tag{101}$$

It can be seen that

$$p_x\left(t\right) = \frac{v_x\left(t\right)}{2}t + \frac{p_f}{t_f^{*3}}t^3, \tag{102}$$

which can be used to obtain the expression

$$t^3 = \frac{p_x\left(t\right)t_f^{*3}}{p_f} - \frac{v_x\left(t\right)t_f^{*3}}{2p_f}t. \tag{103}$$

Applying the result of (103) to $p_x(t)$ in (101) yields

$$p_x\left(t\right) = v_x\left(t\right)t - 2p_x\left(t\right) + \frac{3p_f}{t_f^{*^2}}t^2 \tag{104}$$

which can be used to obtain the result

$$t^2 + \frac{v_x\left(t\right)t_f^{*^2}}{3p_f}t - \frac{p_x\left(t\right)t_f^{*^2}}{p_f} = 0. \tag{105}$$

Since (105) is a quadratic equation it has two solutions, which are

$$t = -\frac{v_x\left(t\right)t_f^{*^2}}{6p_f} \pm \frac{\sqrt{\frac{v_x^2(t)t_f^{*^4}}{9p_f^2} + \frac{4p_x(t)t_f^{*^2}}{p_f}}}{2}. \tag{106}$$

It can be seen from (101) that

$$\frac{v_x^2\left(t\right)t_f^{*^4}}{9p_f^2} + \frac{4p_x\left(t\right)t_f^{*^2}}{p_f} = \left(\frac{2t\left(t - 2t_f^*\right)}{t_f^*}\right)^2 \tag{107}$$

which allows to simplify (106) to

$$t = -\frac{v_x\left(t\right)t_f^{*^2}}{6p_f} \pm \frac{t\left(t - 2t_f^*\right)}{t_f^*} \tag{108}$$

Given the boundary condition $v_x(t) = 0$ when $t = t_f^*$ it can be seen that only one solution is consistent, which leads to

$$t = -\frac{v_x\left(t\right)t_f^{*^2}}{6p_f} - \frac{t\left(t - 2t_f^*\right)}{t_f^*} = -\frac{1}{t_f^*}t^2 + 2t - \frac{v_x\left(t\right)t_f^{*^2}}{6p_f}. \tag{109}$$

It can be seen from (105) that

$$t^2 = -\frac{v_x\left(t\right)t_f^{*^2}}{3p_f}t + \frac{p_x\left(t\right)t_f^{*^2}}{p_f} \tag{110}$$

which can be used in (109) to yield

$$t = \frac{6p_x\left(t\right)t_f^* + t_f^{*^2}v_x\left(t\right)}{2\left(3p_f + t_f^*v_x\left(t\right)\right)} \tag{111}$$

The result in (111) can then be applied to (110) to yield

$$t^2 = \frac{t_f^{*^2}\left(18p_x\left(t\right)p_f - t_f^{*^2}v_x^2\left(t\right)\right)}{6p_f\left(3p_f + t_f^*v_x\left(t\right)\right)} \tag{112}$$

Equation (112) can then be equated to the square of (111), which results in the expression

$$-\frac{t_f^{*^2}\left(108p_x^2\left(t\right)p_f-108p_x\left(t\right)p_f^2+9p_ft_f^{*^2}v_x^2\left(t\right)+2t_f^{*^3}v_x^3\left(t\right)\right)}{12p_f\left(3p_f+t_fv_x\left(t\right)\right)^2}=0. \tag{113}$$

Observe that equation (113) is inconsistent if the denominator is equal to zero. It can be seen from (101) that

$$t^2-t_f^*t+\frac{t_f^{*^3}v_x\left(t\right)}{6p_f}=0, \tag{114}$$

which can be used to evaluate the times when $v_x\left(t\right)=\frac{-3p_f}{t_f^*}$. Observe that

$$t^2-t_f^*t+\frac{t_f^{*^3}}{6p_f}\left(-\frac{-3p_f}{t_f^*}\right)=t^2-t_f^*t-\frac{t_f^{*^2}}{2}=0, \tag{115}$$

which has the solutions

$$t=\begin{cases}\left(\frac{1-\sqrt{3}}{2}\right)t_f^* & < & 0\\ \left(\frac{1+\sqrt{3}}{2}\right)t_f^* & > & t_f^*\end{cases}. \tag{116}$$

Therefore, given this result and the assumption that $p_c\neq 0$, it can be seen that (113) is consistent $\forall\, t\in[0,t_f^*]$. Observe from (97) that $t_f^*\neq 0$, which allows to simplify (113) to yield

$$2t_f^{*^3}v_x^3\left(t\right)+9p_ft_f^{*^2}v_x^2\left(t\right)+108p_fp_x\left(t\right)\left(p_x\left(t\right)-p_f\right)=0. \tag{117}$$

Therefore, the velocity trajectory satisfies (117) and an analytical velocity trajectory may be obtained in terms of $p_x(t)$ as a real root of this equation.

Lemma 3. *The solutions of*

$$ax^3+bx^2+cx+d=0, \tag{118}$$

are equal to

$$x=y-\frac{b}{3a}, \tag{119}$$

where y is the solutions of

$$y^3+Py+Q=0 \tag{120}$$

and where

$$\begin{aligned}P &= \tfrac{3ac-b^2}{3a^2}\\ Q &= \tfrac{2b^3-9abc+27a^2d}{27a^3}\end{aligned} \tag{121}$$

Proof. Equation (118) is the general form of a cubic polynomial where (120) is known as the depressed cubic.

A proof of this result is provided in [71]. □

Lemma 4. *Given a cubic equation*

$$y^3 + Py + Q = 0, \tag{122}$$

the discriminant is

$$\Delta = -\left(4P^3 + 27Q^2\right) \tag{123}$$

and if $\Delta > 0$, or equivalently $4P^3 + 27Q^2 < 0$, then there are three real solutions of (122). If $\Delta = 0$ then there is either a double or a triple root.

Proof. A proof of this result is provided in [72]. □

Lemma 3 can be used to obtain the depressed cubic of (117) as

$$\bar{v}_x^3(t) + P\bar{v}_x(t) + Q = 0, \tag{124}$$

where

$$\begin{aligned} v_x(t) &= \bar{v}_x(t) - \tfrac{3p_f}{2t_f} \\ P &= -\tfrac{27p_f^2}{4t_f^{*2}} \\ Q(p_x(t)) &= \tfrac{27p_f\left(8p_x^2(t) - 8p_x(t)p_f + p_f^2\right)}{4t_f^{*3}} \\ Q(t) &= \tfrac{27p_f^3\left(32t^6 - 96t_f^*t^5 + 72t_f^{*2}t^4 + 16t_f^{*3}t^3 - 24t_f^{*4}t^2 + t_f^{*6}\right)}{4t_f^{*9}} \end{aligned} . \tag{125}$$

$$\Delta(t) = -\left(4P^3 + 27Q^2\right) = -\frac{19683p_f^6t^2\left(2t - t_f^*\right)^2\left(2t - 3t_f^*\right)\left(t - t_f^*\right)^2\left(2t + t_f^*\right)\left(2t^2 - 2tt_f^* - t_f^{*2}\right)^2}{t_f^{*18}}. \tag{126}$$

It can be seen that equation (126) is equal to zero when

$$t \in \left\{0, \frac{t_f^*}{2}, \frac{3}{2}t_f^*, t_f^*, -\frac{1}{2}t_f^*, -\frac{t_f^*\left(1-\sqrt{3}\right)}{2}, \frac{t_f^*\left(1-\sqrt{3}\right)}{2}\right\} \tag{127}$$

Given that $p_x(t)$ and $v_x(t)$ are known at the boundary times $t = 0$ and $t = t_f^*$, it is necessary to investigate the sign of the discriminant $\forall\ t \in (0, t_f^*)$ to determine the real root(s) of (124). It can be seen from (127) that, for the interval $t \in (0, t_f^*)$, there is at most one possible sign change of the discriminant and one point when the discriminant is equal to zero. The sign of the discriminant for the time intervals $t \in (0, \frac{t_f^*}{2})$ and $t \in (\frac{t_f^*}{2}, t_f^*)$ is therefore investigated by evaluating $\Delta(t)$ at $t = \frac{t_f^*}{4}$ and $t = \frac{3t_f^*}{4}$, which yields

$$\Delta\left(\frac{t_f^*}{4}\right) = \Delta\left(\frac{3t_f^*}{4}\right) = \frac{321521805p_f^6}{262144t_f^{*6}} > 0. \tag{128}$$

From Lemma 4 it can be seen that there are 3 real solutions of (124).

Lemma 5. *The roots of the depressed cubic*

$$y^3 + Py + Q = 0 \tag{129}$$

are

$$y_k = 2\sqrt{-\frac{P}{3}}\,\mathtt{cos}\left(\frac{\mathtt{accos}\left(\frac{3Q\sqrt{-3}}{2P\sqrt{P}}\right) - 2\pi k}{3}\right). \tag{130}$$

Proof. A proof of this result is provided in [73]. □

It can be seen from Lemma 5 that the solutions of (124) are

$$\begin{aligned}
\bar{v}_{x_0}(p_x(t)) &= \frac{3|p_f|}{t_f^*}\mathtt{cos}\left(\frac{\mathtt{arccos}\left(-\frac{8p_x^2(t)-8p_x(t)p_f+p_f^2}{p_f|p_f|}\right)}{3}\right)\\
\bar{v}_{x_1}(p_x(t)) &= \frac{3|p_f|}{t_f^*}\mathtt{cos}\left(\frac{\mathtt{arccos}\left(-\frac{8p_x^2(t)-8p_x(t)p_f+p_f^2}{p_f|p_f|}\right)-2\pi}{3}\right)\\
\bar{v}_{x_2}(p_x(t)) &= \frac{3|p_f|}{t_f^*}\mathtt{cos}\left(\frac{\mathtt{arccos}\left(-\frac{8p_x^2(t)-8p_x(t)p_f+p_f^2}{p_f|p_f|}\right)-4\pi}{3}\right)
\end{aligned}. \tag{131}$$

Then, using the relationships between $\bar{v}_x(p_x(t))$ and $v_x(p_x(t))$ in (125), the solutions of $v_x(p_x(t))$ in (113) are obtained as

$$\begin{aligned}
v_{x_0}(p_x(t)) &= \frac{3|p_f|}{t_f^*}\mathtt{cos}\left(\frac{\mathtt{arccos}\left(-\frac{8p_x^2(t)-8p_x(t)p_f+p_f^2}{p_f|p_f|}\right)}{3}\right) - \frac{3p_f}{2t_f^*}\\
v_{x_1}(p_x(t)) &= \frac{3|p_f|}{t_f^*}\mathtt{cos}\left(\frac{\mathtt{arccos}\left(-\frac{8p_x^2(t)-8p_x(t)p_f+p_f^2}{p_f|p_f|}\right)-2\pi}{3}\right) - \frac{3p_f}{2t_f^*}\\
v_{x_2}(p_x(t)) &= \frac{3|p_f|}{t_f^*}\mathtt{cos}\left(\frac{\mathtt{arccos}\left(-\frac{8p_x^2(t)-8p_x(t)p_f+p_f^2}{p_f|p_f|}\right)-4\pi}{3}\right) - \frac{3p_f}{2t_f^*}
\end{aligned}. \tag{132}$$

The boundary conditions of $p_x(t)$ and $v_x(t)$ are then used to determine which solution in (132) is the one that describes the velocity trajectory of the system. Applying the boundary conditions $p_x(t) = 0$ and $v_x(t) = 0$

for $t = 0$ yields

$$\begin{aligned}
v_{x_0}\left(p_x\left(t\right)\right)|_{t=0} &= \frac{6|p_f|\cos\left(\frac{\pi-\operatorname{acos}\left(\frac{p_f}{|p_f|}\right)}{3}\right)-3p_f}{t_f^*} = \begin{cases} 0 & \texttt{if} \quad p_f > 0 \\ -\frac{9p_f}{2t_f^*} & \texttt{if} \quad p_f < 0 \end{cases} \\
v_{x_1}\left(p_x\left(t\right)\right)|_{t=0} &= \frac{6|p_f|\cos\left(\frac{\pi+\operatorname{acos}\left(\frac{p_f}{|p_f|}\right)}{3}\right)-3p_f}{t_f^*} = \begin{cases} 0 & \texttt{if} \quad p_f > 0 \\ 0 & \texttt{if} \quad p_f < 0 \end{cases} \\
v_{x_2}\left(p_x\left(t\right)\right)|_{t=0} &= \frac{-6|p_f|\cos\left(\frac{\operatorname{acos}\left(\frac{p_f}{|p_f|}\right)}{3}\right)-3p_f}{t_f^*} = \begin{cases} -\frac{9p_f}{2t_f^*} & \texttt{if} \quad p_f > 0 \\ 0 & \texttt{if} \quad p_f < 0 \end{cases}
\end{aligned} \tag{133}$$

Additionally, analysing the boundary conditions $p_x(t) = p_f$ and $v_x(t) = 0$ for $t = t_f^*$ yields

$$\begin{aligned}
v_{x_0}\left(p_x\left(t\right)\right)|_{t=t_f^*} &= \frac{6|p_f|\cos\left(\frac{\pi-\operatorname{acos}\left(\frac{p_f}{|p_f|}\right)}{3}\right)-3p_f}{t_f^*} = \begin{cases} 0 & \texttt{if} \quad p_f > 0 \\ -\frac{9p_f}{2t_f^*} & \texttt{if} \quad p_f < 0 \end{cases} \\
v_{x_1}\left(p_x\left(t\right)\right)|_{t=t_f^*} &= \frac{6|p_f|\cos\left(\frac{\pi+\operatorname{acos}\left(\frac{p_f}{|p_f|}\right)}{3}\right)-3p_f}{t_f^*} = \begin{cases} 0 & \texttt{if} \quad p_f > 0 \\ 0 & \texttt{if} \quad p_f < 0 \end{cases} \\
v_{x_2}\left(p_x\left(t\right)\right)|_{t=t_f^*} &= \frac{-6|p_f|\cos\left(\frac{\operatorname{acos}\left(\frac{p_f}{|p_f|}\right)}{3}\right)-3p_f}{t_f^*} = \begin{cases} -\frac{9p_f}{2t_f^*} & \texttt{if} \quad p_f > 0 \\ 0 & \texttt{if} \quad p_f < 0 \end{cases}
\end{aligned} \tag{134}$$

Observe that no conclusion can be drawn from these results, since the velocity trajectory may be described by either $v_{x_0}(p_x(t))$ or $v_{x_1}(p_x(t))$ if $p_f > 0$, and it may be described by either $v_{x_1}(p_x(t))$ or $v_{x_2}(p_x(t))$ if $p_f < 0$. Therefore, we look at the point $t = \frac{t_f^*}{2}$, where

$$\begin{aligned}
p_x\left(t\right)|_{t=\frac{t_f^*}{2}} &= \frac{p_f}{2} \\
v_x\left(t\right)|_{t=\frac{t_f^*}{2}} &= \frac{3p_f}{2t_f^*}
\end{aligned} \tag{135}$$

are obtained from (101) and we compare these results with the points of the velocity trajectories in (132).

It can be seen that

$$
\begin{aligned}
v_{x_0}\left(p_x\left(t\right)\right)\big|_{t=\frac{t_f^*}{2}} &= \frac{6|p_f|\cos\left(\frac{\operatorname{acos}\left(\frac{p_f}{|p_f|}\right)}{3}\right)-3p_f}{t_f^*} = \begin{cases} \frac{3p_f}{2t_f^*} & \texttt{if} \quad p_f > 0 \\ -\frac{3p_f}{t_f^*} & \texttt{if} \quad p_f < 0 \end{cases} \\
v_{x_1}\left(p_x\left(t\right)\right)\big|_{t=\frac{t_f^*}{2}} &= \frac{6|p_f|\cos\left(\frac{2\pi+\operatorname{acos}\left(\frac{p_f}{|p_f|}\right)}{3}\right)-3p_f}{t_f^*} = \begin{cases} -\frac{3p_f}{t_f^*} & \texttt{if} \quad p_f > 0 \\ -\frac{3p_f}{t_f^*} & \texttt{if} \quad p_f < 0 \end{cases} \\
v_{x_2}\left(p_x\left(t\right)\right)\big|_{t=\frac{t_f^*}{2}} &= \frac{-6|p_f|\cos\left(\frac{\operatorname{acos}\left(\pi-\frac{p_f}{|p_f|}\right)}{3}\right)-3p_f}{t_f^*} = \begin{cases} -\frac{3p_f}{t_f^*} & \texttt{if} \quad p_f > 0 \\ \frac{3p_f}{2t_f^*} & \texttt{if} \quad p_f < 0 \end{cases}
\end{aligned}
\tag{136}
$$

Therefore, the velocity trajectory can be expressed as a function of $p_x(t)$ according to

$$
v_x\left(p_x\left(t\right)\right) = \begin{cases} \frac{3p_f}{t_f^*}\cos\left(\frac{\operatorname{arccos}\left(-\frac{8p_x^2(t)-8p_x(t)p_f+p_f^2}{p_f^2}\right)}{3}\right)-\frac{3p_f}{2t_f^*} & \texttt{if} \quad p_f > 0 \\ -\frac{3p_f}{t_f^*}\cos\left(\frac{\operatorname{arccos}\left(\frac{8p_x^2(t)-8p_x(t)p_f+p_f^2}{p_f^2}\right)-4\pi}{3}\right)-\frac{3p_f}{2t_f^*} & \texttt{if} \quad p_f < 0 \end{cases}
\tag{137}
$$

Let the final state of the system be described by

$$
\boldsymbol{x}_f = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T. \tag{138}
$$

The number of iterations of the Kinodynamic RRT* solver was selected as $N_{\texttt{iter}} = 1000$, the cost-to-go upper bound was selected as $\eta = 1$, and the reachability tuning parameter as $\gamma = 1000$. The following simulation results were obtained using MATLAB R2018a on an Intel Core i5 2.9 GHz processor with 16GB of RAM. Figure 13 shows the optimal state trajectory and compares it with the approximate solution obtained using Kinodynamic RRT*. The initial state is shown as the green diamond and the final state is shown as the red diamond. The optimal state trajectory is described by the green curve, which has a trajectory cost of 2.7464 $\frac{m^2}{s^3}$. The first approximate solution of the Kinodynamic RRT* algorithm was obtained in 4 iterations. This result is shown in blue, which has a cost of 2.7978 $\frac{m^2}{s^3}$ and represents an error of 1.9%. This approximation was then improved as the solver iterated to the state trajectory shown in red, which has a cost of 2.7896 $\frac{m^2}{s^3}$ and represents an error of 1.6%. The approximation was then improved three more times, where the cost of each approximation is shown in the legend of Figure 13. The curve shown in purple

shows the final approximation obtained by the solver following the completion of 1000 iterations. It can be seen that this approximation closely matches the optimal solution, and consists of an error of only 0.025%. Figure 14 shows the optimal control input trajectory in green that produces the state trajectory shown in green in Figure 13. Additionally, the approximate control input trajectory obtained from the Kinodynamic RRT* algorithm is shown in black, which produces the state trajectory shown in purple in Figure 13. Figure 15 shows $\mathcal{T}$, where the initial state is shown as the green diamond and the final state is shown as the red diamond, the vertices of the graph are represented by the points, and the solid curves describe the state trajectory components of each edge. Furthermore, the approximate solution obtained by the solver is shown as the solid green curve. From these results it can be seen that the Kinodynamic RRT* quickly obtains an approximate solution of the optimal control problem, which is iteratively improved and converges toward the optimal solution.

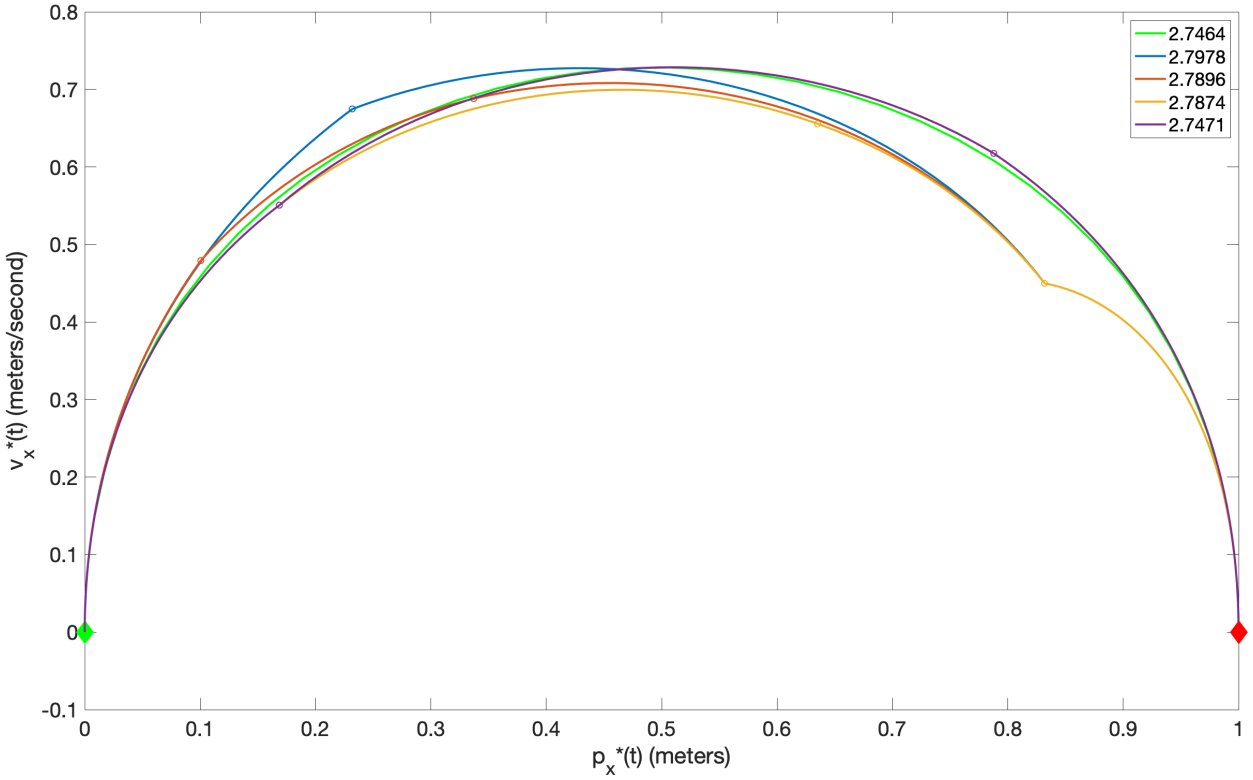


Figure 13: 1D double integrator: optimal state and Kinodynamic RRT* approximate state trajectories.

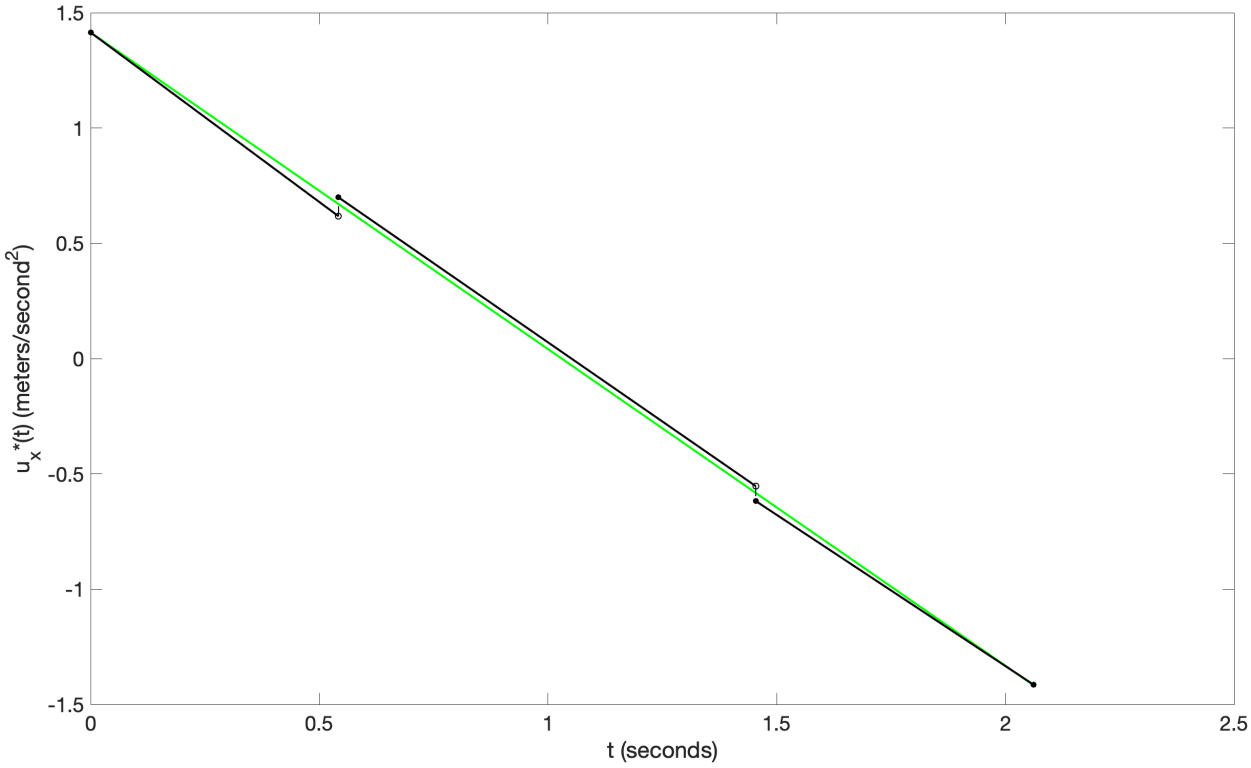


Figure 14: 1D double integrator: optimal state and Kinodynamic RRT* approximate input trajectories.

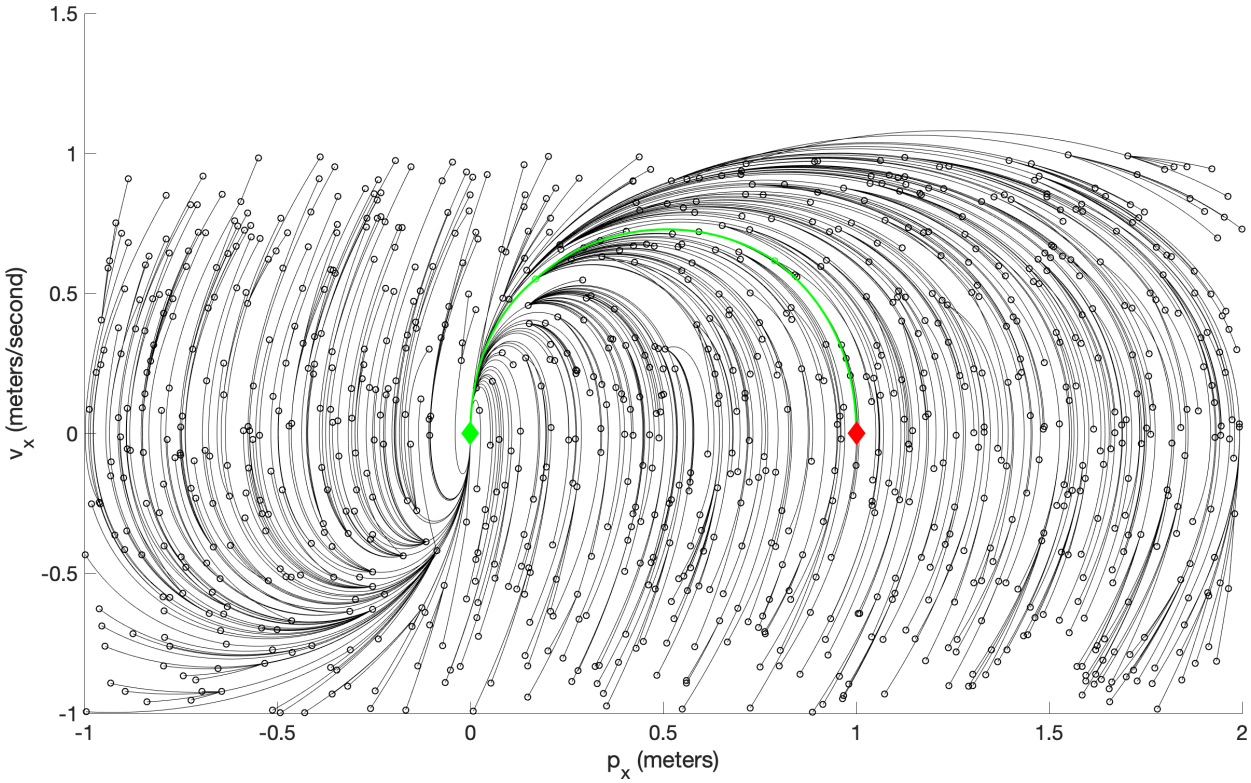


Figure 15: 1D double integrator: Kinodynamic RRT* tree in the state space.

2.3.2.2 Example

An optimal trajectory planning problem in a non-convex bounded state space is now addressed, which is solved using optimal control theory and Kinodynamic RRT*. The system's state space is

$$\boldsymbol{\mathcal{X}} = \left\{ \boldsymbol{x} \in \mathbb{R}^6 : \begin{array}{c} -100 \leq p_x, p_y \leq 100,\ 0 \leq p_z \leq 100, \\ \|\boldsymbol{v}\| \leq 20 \end{array} \right\} \tag{139}$$

and the obstacle space is described by

$$\boldsymbol{\mathcal{X}}_{\texttt{obs}} = \bigcup_{i=1}^{i=4} \bigcup_{j=1}^{j=4} \boldsymbol{\mathcal{X}}_{\texttt{obs}_{ij}}, \tag{140}$$

where

$$\boldsymbol{\mathcal{X}}_{\texttt{obs}_{ij}} = \left\{ \boldsymbol{x} \in \mathbb{R}^6 : \left\| \begin{bmatrix} p_x + 100 - 40i \\ p_y + 100 - 40j \end{bmatrix} \right\| < 10 \right\}. \tag{141}$$

The obstacle free space $\boldsymbol{\mathcal{X}}_{\texttt{free}}$ is obtained according to (40). The initial and final states of the system are

$$\begin{array}{rcl} \boldsymbol{x}_0 & = & \begin{bmatrix} -40 & -40 & 40 & 0 & 0 & 0 \end{bmatrix}^T \\ \boldsymbol{x}_f & = & \begin{bmatrix} 40 & 40 & 80 & 0 & 0 & 0 \end{bmatrix}^T \end{array}, \tag{142}$$

and the affine term is

$$\boldsymbol{c} = \begin{bmatrix} \boldsymbol{c}_v^T & 0 & 0 & -g \end{bmatrix}^T \quad \boldsymbol{c}_v = \begin{bmatrix} 3 & -2 & 0.5 \end{bmatrix}^T, \tag{143}$$

where $g = 9.8\ \frac{m}{s^2}$ is the acceleration due to gravity and the vector $\boldsymbol{c}_v$ describes a constant wind velocity in meters per second. Finally, the control input weighting matrix is $R = I_3$ and the cost index is $C_I = 10$. The maximum number of iterations was selected as $N_{\texttt{iter}} = 3000$, the cost-to-go upper bound as $\eta = 300$, and the reachability region tuning parameter as $\gamma = 1000$. The simulation is performed in MATLAB R2018a on an Intel Core i5 2.9 GHz processor with 16GB of RAM. Figures 16 and 17 show the RRT* tree in the position subspace of $\boldsymbol{\mathcal{X}}$. In these figures the red cylinders represent $\boldsymbol{\mathcal{X}}_{\texttt{obs}}$, the solid black curves show the position vectors of the set of edges $\boldsymbol{E}$, the smaller black circles show the position of the set of vertices $\boldsymbol{V}$, the green diamond indicates $\boldsymbol{p}_0$, the red diamond indicates $\boldsymbol{p}_f$, and the green curve represents the optimal position trajectory. Figure 18 shows the optimal position trajectory, where the blue vectors indicate the constant wind velocity vector $\boldsymbol{c}_v$. Figure 19 shows the velocity vector components of the optimal state trajectory, the optimal velocity magnitude, and the maximum velocity. Figures 20 – 22 present the optimal control input trajectory components, and Figure 23 shows the optimal control input trajectory magnitude. In these figures the solid lines describe the optimal control input of each trajectory segment, and the dashed lines

represent discontinuities between two segments.

These results demonstrate how the Kinodynamic RRT* trajectory planning algorithm is capable of obtaining an approximate solution of optimal control problems with non-convex state constraints where optimal control theory struggles. Obtaining a solution of (69) using optimal control theory directly is very difficult. However, combining optimal control theory with Kinodynamic RRT* provides the ability to efficiently obtain approximate solutions of such problems.

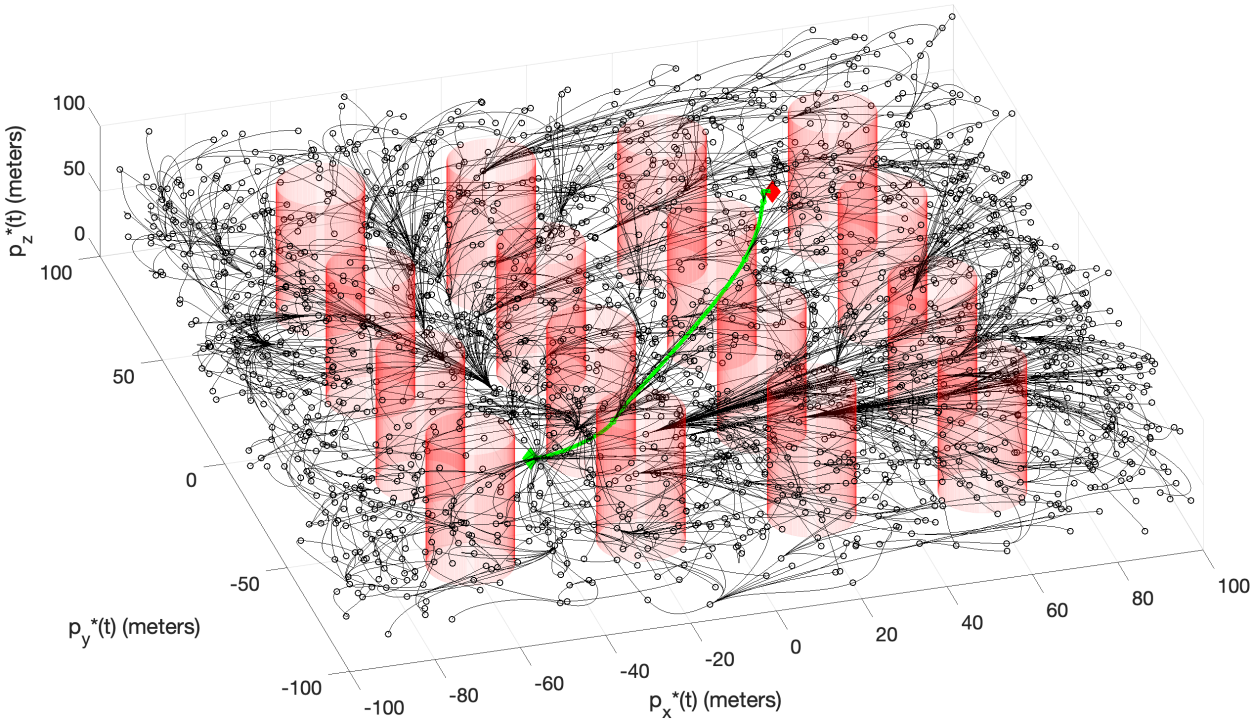


Figure 16: Double integrator with drift: RRT* tree in position subspace.

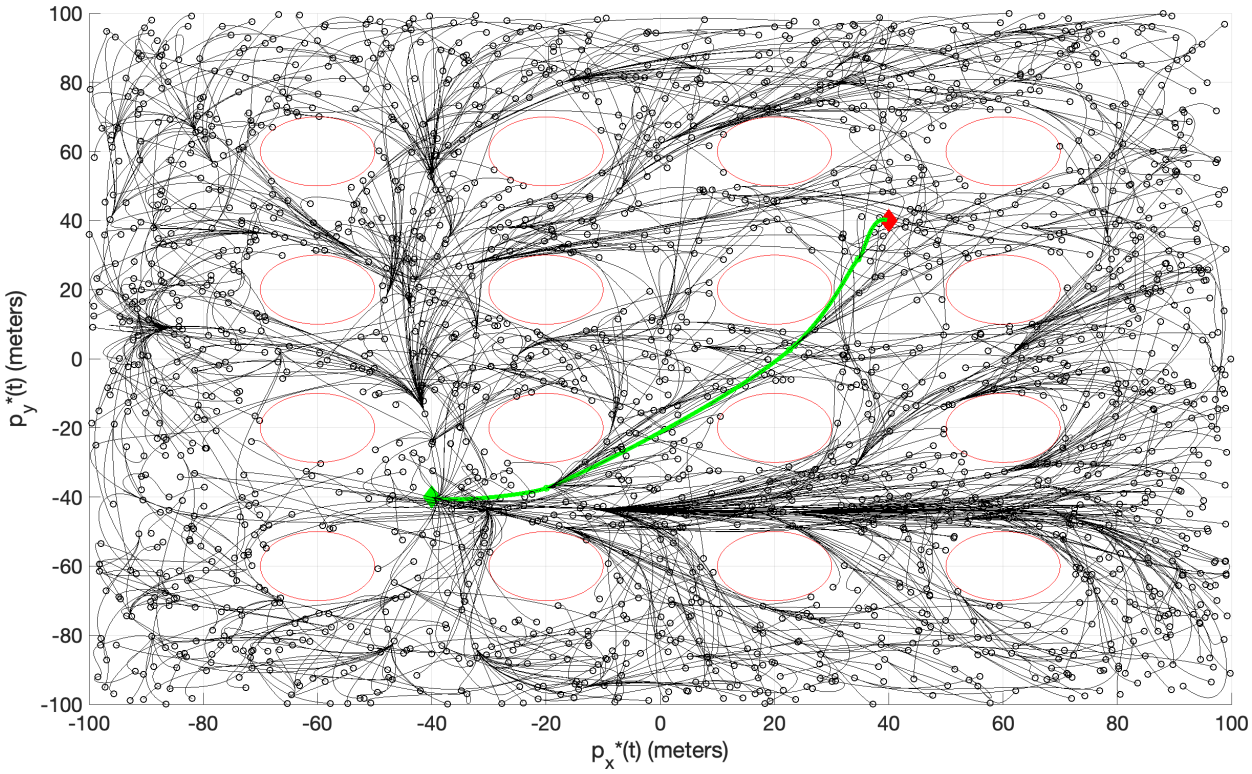


Figure 17: Double integrator with drift: RRT* tree in position subspace top-down view.

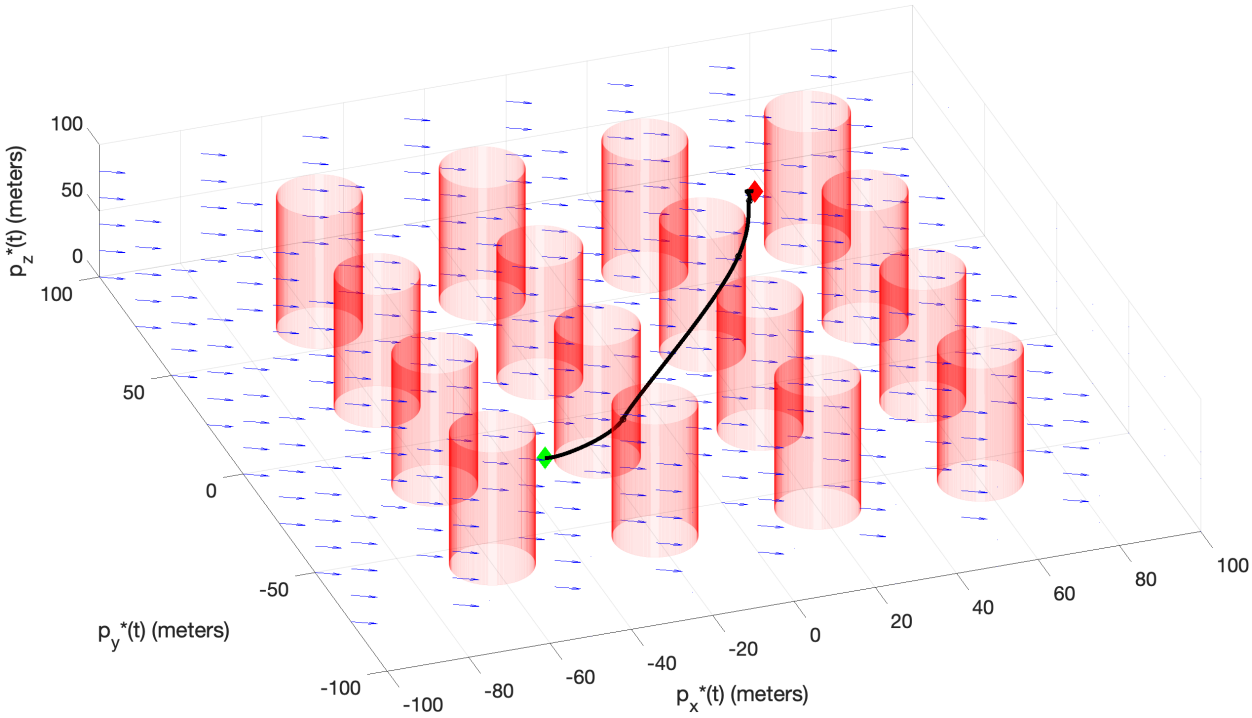


Figure 18: Double integrator with drift: optimal position trajectory.

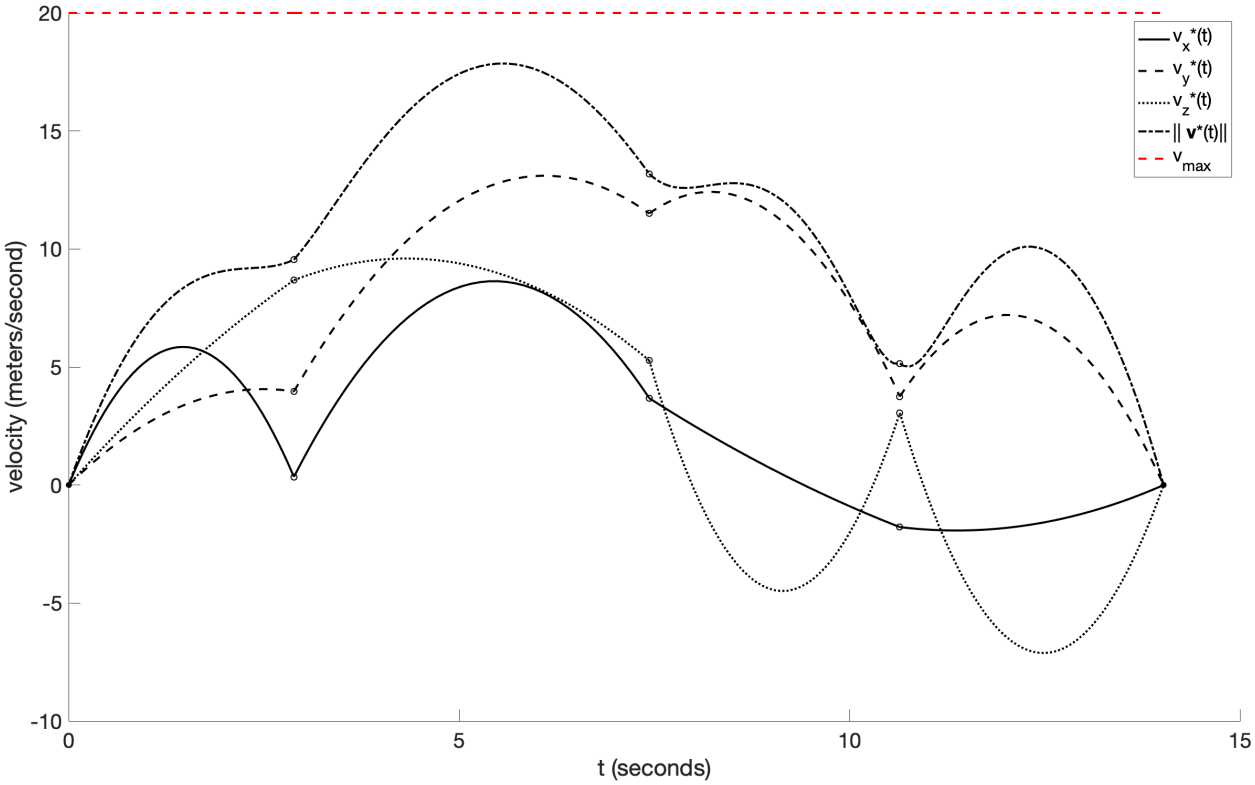


Figure 19: Double integrator with drift: optimal velocity trajectory.

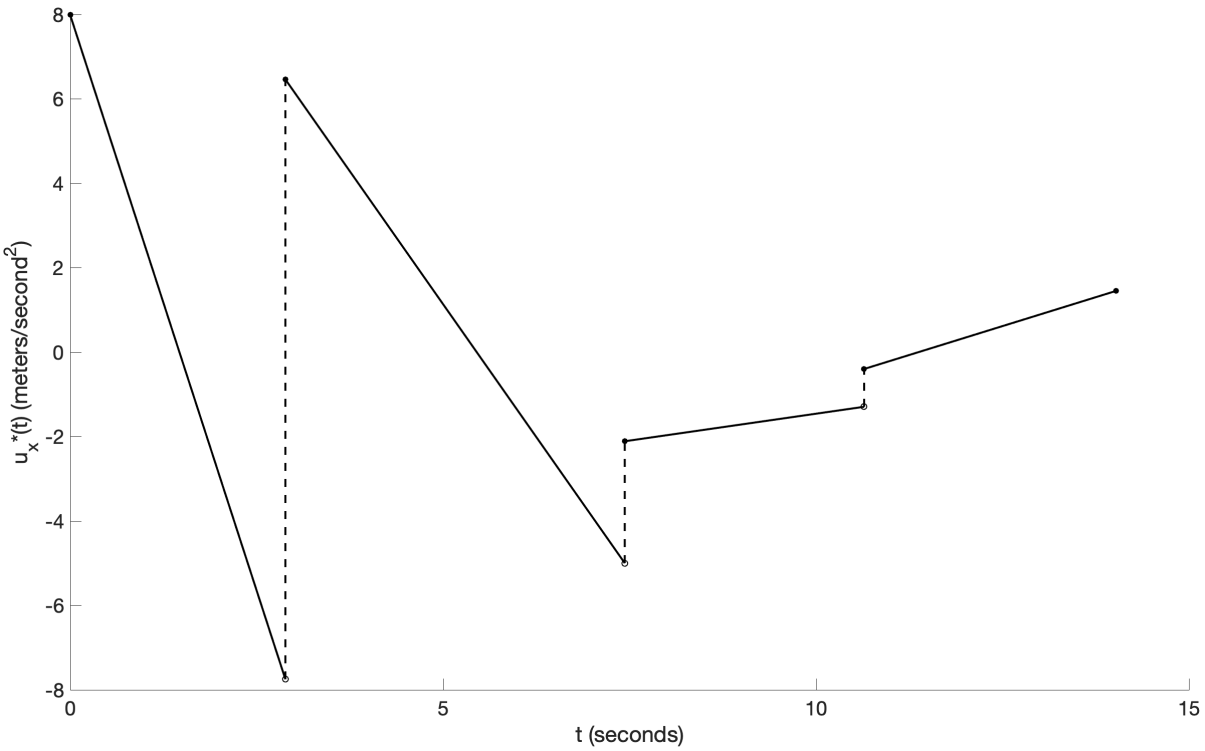


Figure 20: Double integrator with drift: optimal control input trajectory $u_x^*(t)$.

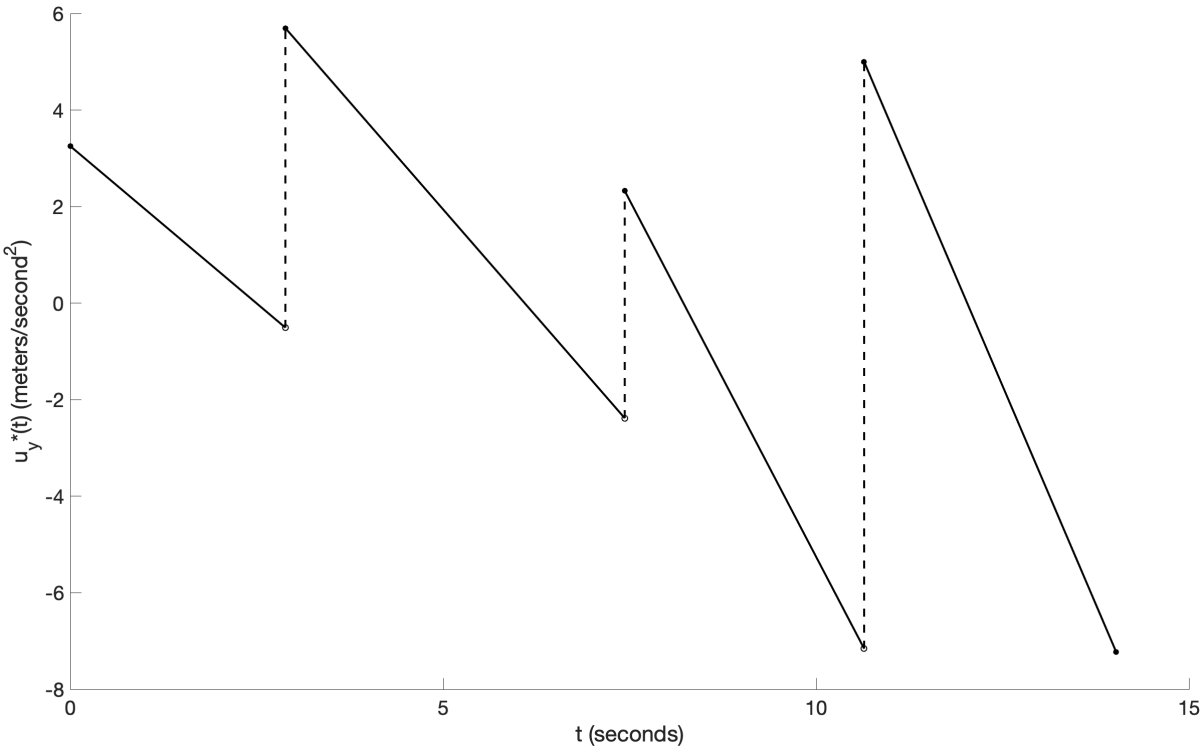


Figure 21: Double integrator with drift: optimal control input trajectory $u_y^*(t)$.

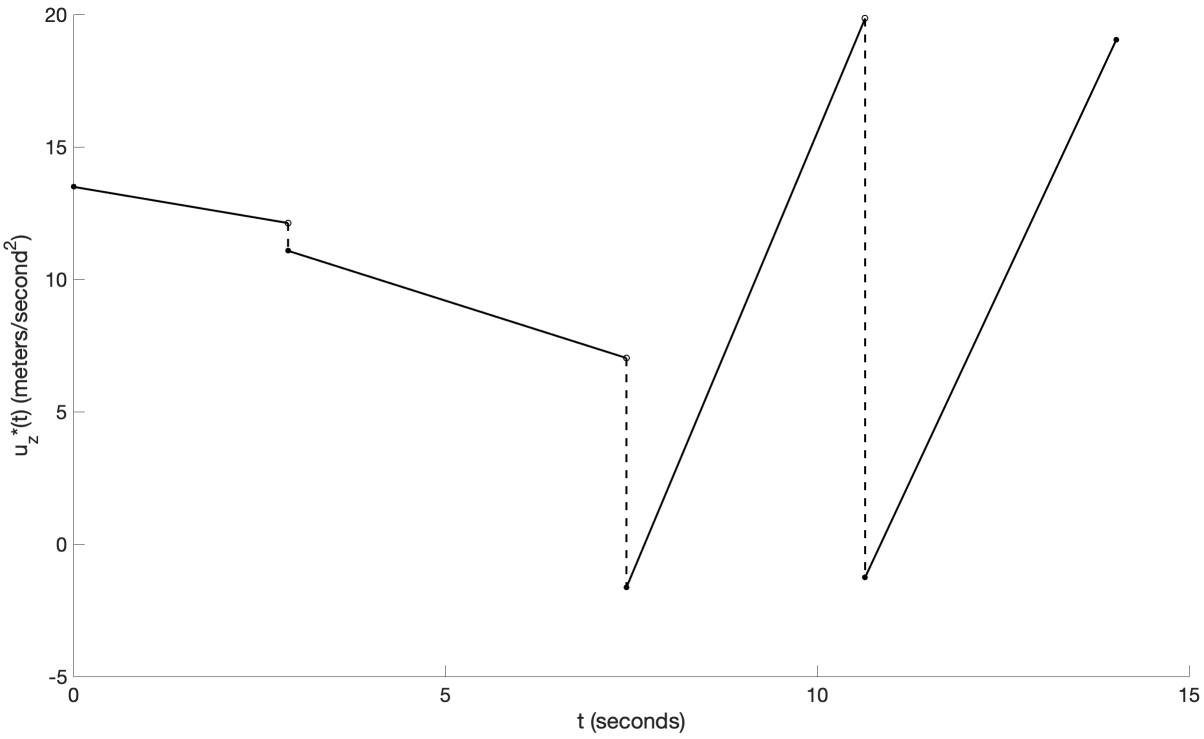


Figure 22: Double integrator with drift: optimal control input trajectory $u_z^*(t)$.

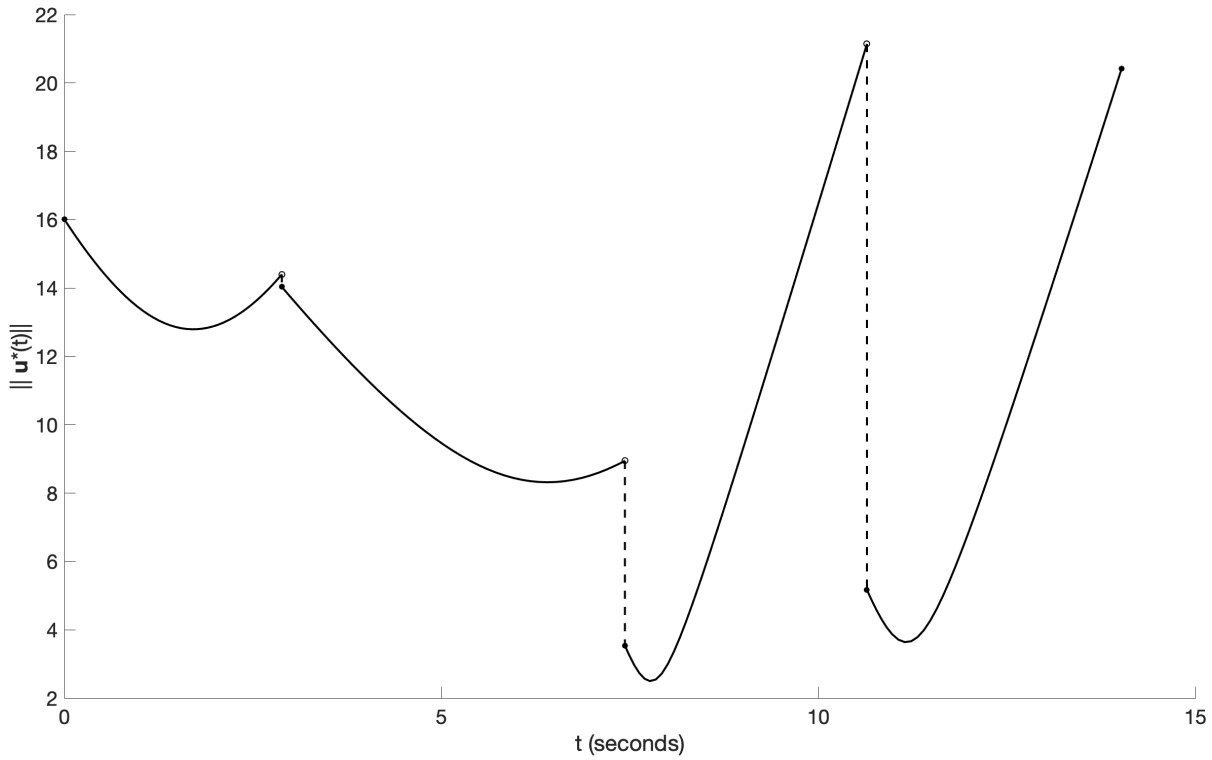


Figure 23: Double integrator with drift: optimal control input trajectory magnitude.

Chapter 3

Optimal Control of Input Constrained Affine Systems

Numerous contributions in the open literature have been made toward improving RRT*, but the majority of these extensions assume systems with unconstrained control inputs. However, in reality, the majority of systems are subject to such constraints. This chapter addresses this gap in the literature by proposing a modified Kinodynamic RRT* that solves optimal control problems for affine systems with non-convex state spaces that are subject to convex control input constraints. Optimal control theory is used to obtain the explicit unconstrained control input trajectory to steer the system between arbitrary states such that the resulting trajectory minimizes a cost. Then, two augmented RRT* functions are proposed to evaluate if and when the control input constraints of the system are violated. As a result of this, all edges of the modified Kinodynamic RRT* trajectory planner are guaranteed to verify the control input constraints of the system. The remainder of this chapter is organized as follows. First, the optimal trajectory planning problem is formulated for affine systems with non-convex state spaces that are subject to convex control input constraints. This problem is formulated as an extension of that in (69). Since Kinodynamic RRT* does not consider the control input constraints of the system the problem is solved by proposing an extension of two RRT* functions. Finally, a modified version of the problem discussed in Section 2.3.2.2 is presented where the system's control input is assumed to be constrained by a maximum control input magnitude, which cannot be solved using the Kinodynamic RRT* that was discussed in Chapter 2.

3.1 Problem Formulation

The system's state is denoted by the vector $\boldsymbol{x} \in \mathbb{R}^n$ and its control input is denoted by the vector $\boldsymbol{u} \in \mathbb{R}^m$, where $n, m \in \mathbb{N}_{>0}$. The system's state space is denoted by $\boldsymbol{\mathcal{X}} \subseteq \mathbb{R}^n$, and its control input space is denoted by $\boldsymbol{\mathcal{U}} \subset \mathbb{R}^m$, where $\boldsymbol{\mathcal{U}}$ is assumed to be a closed convex set. The state transition equation of the system is assumed to be of the form

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} + \boldsymbol{c}, \tag{144}$$

where $A \in \mathbb{R}^{n\times n}$, $B \in \mathbb{R}^{n\times m}$, $\boldsymbol{c} \in \mathbb{R}^n$, and the pair (A, B) is assumed to be controllable. Furthermore, given a control input trajectory $\boldsymbol{u}(t)$, a state trajectory $\boldsymbol{x}(t)$, and a final time t_f, the cost is evaluated as

$$C\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t_f\right) = \int_0^{t_f} \left(\frac{1}{2}\boldsymbol{u}\left(\tau\right)^T R\boldsymbol{u}\left(\tau\right) + C_I\right) d\tau, \tag{145}$$

where $R \in \mathbb{R}^{m\times m}$ is the control input weighting matrix and is assumed to be symmetric and positive definite, and $C_I \in \mathbb{R}_{>0}$ is the cost index. It is assumed that there is a subspace of $\boldsymbol{\mathcal{X}}$ to be avoided, which is described by the open set $\boldsymbol{\mathcal{X}}_{\texttt{obs}} \subset \boldsymbol{\mathcal{X}}$. The obstacle-free subspace of $\boldsymbol{\mathcal{X}}$ can then be described by

$$\boldsymbol{\mathcal{X}}_{\texttt{free}} = \boldsymbol{\mathcal{X}} \setminus \boldsymbol{\mathcal{X}}_{\texttt{obs}}. \tag{146}$$

Given an initial state $\boldsymbol{x}_0 \in \boldsymbol{\mathcal{X}}_{\texttt{free}}$ and final state $\boldsymbol{x}_f \in \boldsymbol{\mathcal{X}}_{\texttt{free}}$ the optimal control problem is defined as

$$\begin{aligned} & \min_{\boldsymbol{u}(t), t_f} \int_0^{t_f} \left(\tfrac{1}{2}\boldsymbol{u}\left(\tau\right)^T R\boldsymbol{u}\left(\tau\right) + C_I\right) d\tau \\ \texttt{s.t.} \quad & \dot{\boldsymbol{x}}\left(t\right) = A\boldsymbol{x}\left(t\right) + B\boldsymbol{u}\left(t\right) + \boldsymbol{c} \\ & \boldsymbol{x} : [0, t_f] \to \boldsymbol{\mathcal{X}}_{\texttt{free}} \\ & \boldsymbol{u} : [0, t_f] \to \boldsymbol{\mathcal{U}} \\ & \boldsymbol{\psi}(\boldsymbol{x}_0, \boldsymbol{x}_f) = \boldsymbol{0} \end{aligned}, \tag{147}$$

where the boundary conditions are described by

$$\boldsymbol{\psi}(\boldsymbol{x}_0, \boldsymbol{x}_f) = \begin{bmatrix} \boldsymbol{x}(0) - \boldsymbol{x}_0 \\ \boldsymbol{x}(t_f) - \boldsymbol{x}_f \end{bmatrix}. \tag{148}$$

3.2 Problem Solution

As discussed in Chapter 2, optimal control theory has had success in solving optimal control problems. However, this theory struggles at obtaining a solution when the system is subject to state or control input constraints - particularly when these constraints cause the problem to be non-convex. As a consequence, many alternative approaches have been developed to solve such problems. Kinodynamic RRT* is one such technique, which is an iterative solver that uses optimal control theory and a tree to obtain approximate solutions of problems of the form (69). A brief overview of the methodology of Kinodynamic RRT* is presented next, while the algorithm is described in detail in Section 2.2. The tree is denoted by $\mathcal{T}$, which is comprised of a set of vertices $\boldsymbol{V} \subset \boldsymbol{\mathcal{X}}_{\texttt{free}}$, and a set of edges $\boldsymbol{E}$. A detailed definition of the tree, vertices, and

edges is presented in Definition 1 of Section 2.2.2. The tree is rooted at $\boldsymbol{x}_0$ and is iteratively grown through $\boldsymbol{\mathcal{X}}_{\texttt{free}}$. At each iteration the sampling function generates a new state sample $\boldsymbol{x}_{\texttt{rand}} \in \boldsymbol{\mathcal{X}}_{\texttt{free}}$. Then, the vertex $\boldsymbol{v}_{\texttt{min}} \in \boldsymbol{V}$ is obtained from the nearest vertex function. The unconstrained optimal control input trajectory, unconstrained optimal state trajectory, and the final time to steer the system from $\boldsymbol{v}_{\texttt{min}}$ to $\boldsymbol{x}_{\texttt{rand}}$ are then obtained from the steering function. Then, the local trajectory function is used to establish a new candidate state $\boldsymbol{x}_{\texttt{new}}$ along the unconstrained optimal state trajectory from $\boldsymbol{v}_{\texttt{min}}$ to $\boldsymbol{x}_{\texttt{rand}}$ such that the cost-to-go from $\boldsymbol{v}_{\texttt{min}}$ to $\boldsymbol{x}_{\texttt{new}}$ is maximized while constrained by an upper bound $\eta \in \mathbb{R}_{>0}$. A description of the parameter η is presented in Section 2.2.2.1. If the trajectory from $\boldsymbol{v}_{\texttt{min}}$ to $\boldsymbol{x}_{\texttt{new}}$ is evaluated as collision-free by the collision-checker function, then $\boldsymbol{x}_{\texttt{new}}$ is added to $\boldsymbol{V}$ and an edge characterised by the unconstrained optimal trajectory components to steer the system from $\boldsymbol{v}_{\texttt{min}}$ to $\boldsymbol{x}_{\texttt{new}}$ is added to $\boldsymbol{E}$. Two optimization steps follow. The first determines the best $\boldsymbol{v} \in \boldsymbol{V}$ to connect with $\boldsymbol{x}_{\texttt{new}}$. The second attempts to minimize the cost of the vertices in $\boldsymbol{V}$ due to the addition of $\boldsymbol{x}_{\texttt{new}}$ to $\boldsymbol{V}$. After a given number of iterations, $N_{\texttt{iter}} \in \mathbb{R}_{>0}$, the planner attempts to connect $\boldsymbol{x}_f$ with $\mathcal{T}$. If this connection fails then no solution is obtained. If the connection is successful, then the solution is obtained as the concatenation of the edges from $\boldsymbol{x}_0$ to $\boldsymbol{x}_f$. A detailed description of all functions is provided in Definition 2 of Section 2.2.2. Despite the success of Kinodynamic RRT*, the algorithm is designed for systems where $\boldsymbol{\mathcal{U}} = \mathbb{R}^m$. The problem (147) will therefore be solved using a modified Kinodynamic RRT*. The modifications will be in two functions from Definition 2: the collision-checker function, and the local trajectory function.

Definition 4. *Steering Function: Given an initial state $\boldsymbol{x}_1 \in \boldsymbol{\mathcal{X}}$ and final state $\boldsymbol{x}_2 \in \boldsymbol{\mathcal{X}}$, the steering function for the optimal control problem described in* (147) *obtains the unconstrained optimal control input trajectory and the final time of the trajectory as the solution of*

$$\begin{aligned} & \min_{\boldsymbol{u}_{1,2}(t), t_{f_{1,2}}} \int_0^{t_{f_{1,2}}} \left(\tfrac{1}{2} \boldsymbol{u}_{1,2}(\tau)^T R \boldsymbol{u}_{1,2}(\tau) + C_I \right) d\tau \\ \texttt{s.t.} \quad & \dot{\boldsymbol{x}}_{1,2}(t) = A\boldsymbol{x}_{1,2}(t) + B\boldsymbol{u}_{1,2}(t) + \boldsymbol{c} \\ & \boldsymbol{\psi}(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{0} \end{aligned} . \tag{149}$$

The unconstrained optimal state trajectory $\boldsymbol{x}^*_{1,2}(t)$ is then obtained by integrating (144) *from $t = 0$ to $t = t^*_{f_{1,2}}$ for the control input $\boldsymbol{u}^*_{1,2}(t)$, where $\boldsymbol{u}^*_{1,2}(t)$ and $t^*_{f_{1,2}}$ are the solutions of* (149). □

Lemma 6. *Using the collision-checker function in Algorithm 2 on page 30, all edges in $\boldsymbol{E}$ are characterized by trajectories that satisfy the constraints for the collision-checker function to return true.*

Proof. Observe in Algorithm 2 that unconstrained optimal trajectories are obtained from the steering function and then used to define edges. However, immediately prior to the generation of these edges, the

trajectories are required to be validated by the collision-checker function. Therefore, if an edge is added to $\boldsymbol{E}$, then its trajectory components satisfy the constraints necessary for the collision-checker function to return *true*. □

The collision-checker function in Definition 2 of Section 2.2.2 considers only unconstrained optimal state trajectories, as shown in (49). However, it can be seen from Lemma 6 that, if the condition for the collision-checker function to return *true* also requires that the control input trajectory satisfy $\boldsymbol{u}^*_{1,2}(t) \in \boldsymbol{\mathcal{U}}$, $\forall\ t \in [0, t^*_{f_{1,2}}]$, where $\boldsymbol{u}^*_{1,2}(t)$ is the unconstrained optimal control input trajectory and $t^*_{f_{1,2}}$ is the final time of the trajectory, then all edges in $\boldsymbol{E}$ would also be characterized by control input trajectories that satisfy the control input constraints of the system. An augmented collision-checker function is therefore proposed in Definition 5 to consider the control input constraints of the system.

Definition 5. *Augmented Collision-Checker Function: Given an unconstrained optimal control input trajectory $\boldsymbol{u}^*_{1,2}(t)$, unconstrained optimal state trajectory $\boldsymbol{x}^*_{1,2}(t)$, and final time $t^*_{f_{1,2}}$ obtained from the steering function in Definition 4 to steer the system from $\boldsymbol{x}_1 \in \boldsymbol{\mathcal{X}}$ to $\boldsymbol{x}_2 \in \boldsymbol{\mathcal{X}}$, the proposed augmented collision-checker function*

$$\beta \leftarrow \texttt{AugCollisionFree}(\boldsymbol{u}^*_{1,2}(t), \boldsymbol{x}^*_{1,2}(t), t^*_{f_{1,2}})$$

returns the boolean

$$\beta = \begin{cases} \textit{true}, & \textit{if } \begin{array}{l} \boldsymbol{x}^*_{1,2} : [0, t^*_{f_{1,2}}] \rightarrow \boldsymbol{\mathcal{X}}_{free}, \\ \boldsymbol{u}^*_{1,2} : [0, t^*_{f_{1,2}}] \rightarrow \boldsymbol{\mathcal{U}} \end{array} \\ \textit{false}, & \texttt{otherwise} \end{cases} . \tag{150}$$

□

It can be seen from Lemma 6 that, if the augmented collision-checker function in Definition 5 is used in place of that from Definition 2 of Section 2.2.2 in Algorithm 2 on page 30, then all edges in $\boldsymbol{E}$ would be characterized by optimal state trajectories that satisfy the obstacle-avoidance state constraints of the optimal control problem, as well as optimal control input trajectories that satisfy the control input constraints of the system. However, it may be difficult to generate a random state space sample $\boldsymbol{x}_{\texttt{rand}}$ such that the optimal control input trajectory from $\boldsymbol{v}_{\texttt{min}} \in \boldsymbol{V}$ to $\boldsymbol{x}_{\texttt{rand}}$ satisfies the constraints for the augmented collision-checker function to return *true*. The parameter $\eta \in \mathbb{R}_{>0}$, which is discussed in Section 2.2.2.1, is used with the local trajectory function in Definition 2 of Section 2.2.2 to address a similar issue related to the obstacle-avoidance state constraints of the problem. This is accomplished by steering the system from $\boldsymbol{v}_{\texttt{min}}$ toward $\boldsymbol{x}_{\texttt{rand}}$ up to a maximum cost η. An augmentation of the local trajectory function is now proposed to also consider the control input constraints of the system. The augmented function steers the system from $\boldsymbol{v}_{\texttt{min}}$ toward $\boldsymbol{x}_{\texttt{rand}}$

up to a maximum cost η or up to the time instant when the control input trajectory reaches the boundary of $\boldsymbol{\mathcal{U}}$, whichever occurs first. If the resulting control input trajectory lies outside of $\boldsymbol{\mathcal{U}}$ then it can be seen from Lemma 6 that this trajectory will be discarded by the planner.

Definition 6. *Augmented Local Trajectory Function: Given an unconstrained optimal control input trajectory $\boldsymbol{u}^*_{1,2}(t)$, unconstrained optimal state trajectory $\boldsymbol{x}^*_{1,2}(t)$, and final time $t^*_{f_{1,2}}$ obtained from the steering function in Definition 4 to steer the system from $\boldsymbol{x}_1 \in \boldsymbol{\mathcal{X}}$ to $\boldsymbol{x}_2 \in \boldsymbol{\mathcal{X}}$, and an upper bound on the cost-to-go $\eta \in \mathbb{R}_{>0}$, the augmented local trajectory function returns the unconstrained optimal control input and state trajectories $\tilde{\boldsymbol{u}}^*_{1,2}(t) = \boldsymbol{u}^*_{1,2}(t)$ and $\tilde{\boldsymbol{x}}^*_{1,2}(t) = \boldsymbol{x}^*_{1,2}(t)$, $\forall\ t \in [0, \tilde{t}^*_{f_{1,2}}]$, and the final time $\tilde{t}^*_{f_{1,2}}$, where*

$$\begin{aligned} \tilde{t}^*_{f_{1,2}} \quad = \quad & \underset{\tau \in \left(0, t^*_{f_{1,2}}\right]}{\operatorname{argmax}} \ C\left(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), \tau\right) \\ \texttt{s.t.} \quad & C\left(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), \tau\right) \leq \eta \\ & \boldsymbol{u}^*_{1,2} : [0, \tau] \rightarrow \boldsymbol{\mathcal{U}} \end{aligned} \tag{151}$$

If no solution of (151) *exists then the augmented local trajectory function assigns $\tilde{t}^*_{f_{1,2}} = 0$. The augmented local trajectory function is denoted by*

$$\tilde{\boldsymbol{x}}^*(t), \tilde{\boldsymbol{u}}^*(t), \tilde{t}^*_f \leftarrow \texttt{AugLocalTraj}(\boldsymbol{x}^*(t), \boldsymbol{u}*(t), t^*_f, \eta)$$

□

The conditions that are required to be satisfied for a solution of (151) to exist are now studied. Let $\boldsymbol{u}^*_{1,2}(t)$, $\boldsymbol{x}^*_{1,2}(t)$, and $t^*_{f_{1,2}}$ denote the unconstrained optimal control input and state trajectories and the final time to steer the system from $\boldsymbol{x}_1 \in \boldsymbol{\mathcal{X}}$ to $\boldsymbol{x}_2 \in \boldsymbol{\mathcal{X}}$. From Theorem 6 it can be seen that, for $\tilde{t}^*_{f_{1,2}} \in [0, t^*_{f_{1,2}}]$, the function $C(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), \tilde{t}^*_{f_{1,2}})$ is a strictly increasing function that satisfies

$$0 \leq C(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), \tilde{t}^*_{f_{1,2}}) \leq C(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), t^*_{f_{1,2}}),\ \forall\ \tilde{t}^*_{f_{1,2}} \in [0, t^*_{f_{1,2}}]. \tag{152}$$

Because of this, and given the assumption that $\eta > 0$, (151) is equivalent to

$$\begin{aligned} \tilde{t}^*_{f_{1,2}} \quad = \quad & \underset{\tau \in \left(0, t^*_{f_{1,2}}\right]}{\operatorname{argmax}} \ \tau \\ \texttt{s.t.} \quad & C\left(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), \tau\right) \leq \eta \\ & \boldsymbol{u}^*_{1,2} : [0, \tau] \rightarrow \boldsymbol{\mathcal{U}} \end{aligned} \tag{153}$$

Let

$$\begin{aligned} t_{\eta_{1,2}} \quad = \quad & \underset{\tau \in \left(0, t^*_{f_{1,2}}\right]}{\text{argmax}} \ \tau \\ \texttt{s.t.} \quad & C\left(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), \tau\right) \leq \eta \end{aligned} \tag{154}$$

and

$$\begin{aligned} t_{u_{1,2}} \quad = \quad & \underset{\tau \in \left(0, t^*_{f_{1,2}}\right]}{\text{argmax}} \ \tau \\ \texttt{s.t.} \quad & \boldsymbol{u}^*_{1,2} : [0, \tau] \to \boldsymbol{\mathcal{U}} \end{aligned} . \tag{155}$$

It can be seen that (153) is equivalent to

$$\begin{aligned} \tilde{t}^*_{f_{1,2}} \quad = \quad & \min\left\{t_{\eta_{1,2}}, t_{u_{1,2}}\right\} \\ \texttt{s.t.} \quad & (154) \\ & (155) \end{aligned} . \tag{156}$$

Since $C(\boldsymbol{x}^*_{1,2}(t), \boldsymbol{u}^*_{1,2}(t), \tilde{t}^*_{f_{1,2}})$ is a strictly increasing function that is bounded according to (152), it can be seen that, if $\eta \in \mathbb{R}_{>0}$ then there is a solution of (154). Therefore, if $\eta \in \mathbb{R}_{>0}$ and a solution of (155) exists, then a solution of (151) is guaranteed to exist.

Theorem 10. *Using the collision-checker function from Definition 5 and the local trajectory function from Definition 6 in place of those from Definition 2 on page 20, all edges in $\boldsymbol{E}$ are characterized by optimal trajectories.*

Proof. Observe from Theorem 3 on page 14 that the control input, state trajectories, and the final time returned by the augmented local trajectory function in Definition 6 are the unconstrained optimal trajectories and final time to steer the system from $\tilde{\boldsymbol{x}}^*_{1,2}(0)$ to $\tilde{\boldsymbol{x}}^*_{1,2}(\tilde{t}^*_{f_{1,2}})$. Furthermore, from the definition of the augmented collision-checker function in Definition 5 and the result of Lemma 6 it can be seen that states and trajectories are only added to $\mathcal{T}$ if the trajectory satisfies both the state and control input constraints of (147). Therefore, it can be seen that any state in $\boldsymbol{V}$ can feasibly be reached from $\boldsymbol{x}_0$ along a sequence of edges in $\boldsymbol{E}$ such that each edge in the sequence is characterised by (i) an unconstrained optimal state trajectory that satisfies the collision-avoidance state constraints of (147) and (ii) an unconstrained optimal control input trajectory that satisfies the control input constraints of (147). If the unconstrained optimal state and control input trajectories satisfy the state and control input constraints of (147), then they are also the optimal state and control input trajectories of the constrained optimal control problem because the cost is the same. As a result of this, all edges in $\boldsymbol{E}$ are characterised by optimal trajectories. □

It can be seen from Algorithm 1 on page 29 that the state and control input trajectories obtained by the

Kinodynamic RRT* to steer the system from $\boldsymbol{x}_0$ to $\boldsymbol{x}_f$ consist of piecewise continuous optimal trajectories. Let the the approximate solution of (147) obtained by Kinodynamic RRT* be described by K piecewise continuous optimal trajectories, where each continuous optimal state trajectory piece is denoted by $\boldsymbol{x}_i^*(t)$ and each continuous optimal control input trajectory piece is denoted by $\boldsymbol{u}_i^*(t)$, where $i \in \{1, 2, \ldots, K\}$. Also, let $t_{f_i}^*$ denote the final time of the i-th continuous trajectory piece. The solution of (147) obtained using Kinodynamic RRT* may therefore be described according to

$$\boldsymbol{x}^*(t) = \begin{cases} \boldsymbol{x}_1^*(t) & , \quad t \in \left[0, t_{f_1}^*\right) \\ \boldsymbol{x}_2^*(t - t_{f_1}^*) & , \quad t \in \left[t_{f_1}^*, t_{f_1}^* + t_{f_2}^*\right) \\ \vdots & \vdots \quad \vdots \\ \boldsymbol{x}_K^*(t - \sum_{i=1}^{K-1} t_{f_i}^*) & , \quad t \in \left[\sum_{i=1}^{K-1} t_{f_i}^*, \sum_{i=1}^{K} t_{f_i}^*\right] \end{cases},$$

$$\boldsymbol{u}^*(t) = \begin{cases} \boldsymbol{u}_1^*(t) & , \quad t \in \left[0, t_{f_1}^*\right) \\ \boldsymbol{u}_2^*(t - t_{f_1}^*) & , \quad t \in \left[t_{f_1}^*, t_{f_1}^* + t_{f_2}^*\right) \\ \vdots & \vdots \quad \vdots \\ \boldsymbol{u}_K^*(t - \sum_{i=1}^{K-1} t_{f_i}^*) & , \quad t \in \left[\sum_{i=1}^{K-1} t_{f_i}^*, \sum_{i=1}^{K} t_{f_i}^*\right] \end{cases}. \tag{157}$$

$$t_f^* = \sum_{i=1}^{K} t_{f_i}^*$$

It can be seen from Theorem 10 that $\boldsymbol{x}_i^*(t)$ and $\boldsymbol{u}_i^*(t)$, $\forall\ t \in [0, t_{f_i}^*]$, $\forall\ i \in \{1, 2, \ldots, K\}$ are the optimal trajectories of the constrained optimal control problem to steer the system from $\boldsymbol{x}_i^*(0)$ to $\boldsymbol{x}_i^*(t_{f_i}^*)$. However, this does not imply that the concatenation of these trajectories is also an optimal trajectory of the constrained optimal control problem. Namely, this does not imply that $\boldsymbol{x}^*(t)$ and $\boldsymbol{u}^*(t)$, $\forall\ t \in [0, t_f^*]$ are optimal trajectories from $\boldsymbol{x}^*(0)$ to $\boldsymbol{x}^*(t_f^*)$. Because of this, the solution described by (157) is an approximate solution. The modified Kinodynamic RRT* is shown in Algorithm 13 on page 75. The augmented collision-checker function from Definition 5 is used in place of that from Definition 2 on lines 11, 19, 27, and 39. The augmented local trajectory function from Definition 6 is used in place of that from Definition 2 on line 9.

Algorithm 13 Modified RRT* algorithm

function RRT*($\boldsymbol{x}_0$, $\boldsymbol{x}_f$, $N_{\texttt{iter}}$, η, γ)
  $\boldsymbol{V} \leftarrow \{\boldsymbol{x}_0\}$;
  $\boldsymbol{E} \leftarrow \varnothing$;
  for $i = 1$ to $N_{\texttt{iter}}$ **do**
    $\boldsymbol{x}_{\texttt{rand}} \leftarrow \texttt{Sample}()$;
    $\boldsymbol{v}_{\texttt{min}} \leftarrow \texttt{Nearest}(\boldsymbol{V}, \boldsymbol{x}_{\texttt{rand}})$;
    $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{Steer}(\boldsymbol{v}_{\texttt{min}}, \boldsymbol{x}_{\texttt{rand}})$;
    $\tilde{\boldsymbol{x}}^*(t), \tilde{\boldsymbol{u}}^*(t), \tilde{t}_f^* \leftarrow \texttt{AugLocalTraj}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*, \eta)$;
    $\boldsymbol{x}_{\texttt{new}} \leftarrow \tilde{\boldsymbol{x}}^*(\tilde{t}_f^*)$;
    if $\texttt{AugCollisionFree}(\tilde{\boldsymbol{u}}^*(t), \tilde{\boldsymbol{x}}^*(t), \tilde{t}_f^*)$ **then**
      $e_{\texttt{new}} \leftarrow (\tilde{\boldsymbol{x}}^*(t), \tilde{\boldsymbol{u}}^*(t), \tilde{t}_f^*)$;
      $\boldsymbol{V} \leftarrow \boldsymbol{V} \cup \{\boldsymbol{x}_{\texttt{new}}\}$;
      $\boldsymbol{E} \leftarrow \boldsymbol{E} \cup \{e_{\texttt{new}}\}$;
      $c_{\texttt{max}} = \min\{\eta, \gamma(\frac{\log(|\boldsymbol{V}|)}{|\boldsymbol{V}|})^{\frac{1}{n}}\}$;
      $\boldsymbol{V}_{\texttt{max}} \leftarrow \texttt{Nearby}(\boldsymbol{V} \setminus \{\boldsymbol{x}_{\texttt{new}}\}, \boldsymbol{x}_{\texttt{new}}, c_{\texttt{max}})$;
      for each $\boldsymbol{v} \in \boldsymbol{V}_{\texttt{max}}$ **do**
        $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{Steer}(\boldsymbol{v}, \boldsymbol{x}_{\texttt{new}})$;
        if $\texttt{AugCollisionFree}(\boldsymbol{u}^*(t), \boldsymbol{x}^*(t), t_f^*)$ **then**
          $e_{\texttt{temp}} \leftarrow (\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*)$;
          $c_e \leftarrow \texttt{Cost}(e_{\texttt{temp}})$;
          if $\mathcal{T}.\texttt{Cost}(\boldsymbol{v}) + c_e < \mathcal{T}.\texttt{Cost}(\boldsymbol{x}_{\texttt{new}})$ **then**
            $\boldsymbol{E} \leftarrow \boldsymbol{E} \cup \{e_{\texttt{temp}}\} \setminus \{e_{\texttt{new}}\}$;
            $e_{\texttt{new}} \leftarrow e_{\texttt{temp}}$;
      for each $\boldsymbol{v} \in \boldsymbol{V}_{\texttt{max}}$ **do**
        $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{Steer}(\boldsymbol{x}_{\texttt{new}}, \boldsymbol{v})$;
        if $\texttt{AugCollisionFree}(\boldsymbol{u}^*(t), \boldsymbol{x}^*(t), t_f^*)$ **then**
          $e_{\texttt{temp}} \leftarrow (\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*)$;
          $c_e \leftarrow \texttt{Cost}(e_{\texttt{temp}})$;
          if $\mathcal{T}.\texttt{Cost}(\boldsymbol{x}_{\texttt{new}}) + c_e < \mathcal{T}.\texttt{Cost}(\boldsymbol{v})$ **then**
            $\boldsymbol{e}_{\texttt{old}} \leftarrow \mathcal{T}.\texttt{GetInEdge}(\boldsymbol{v})$;
            $\boldsymbol{E} \leftarrow \boldsymbol{E} \cup \{e_{\texttt{temp}}\} \setminus \{e_{\texttt{old}}\}$;
    $c_{\texttt{max}} = \min\{\eta, \gamma(\frac{\log(|\boldsymbol{V}|)}{|\boldsymbol{V}|})^{\frac{1}{n}}\}$;
    $\boldsymbol{V}_{\texttt{max}} \leftarrow \texttt{Nearby}(\boldsymbol{V}, \boldsymbol{x}_f, c_{\texttt{max}})$;
    $c_f \leftarrow \mathcal{T}.\texttt{Cost}(\boldsymbol{x}_f)$;
    for each $\boldsymbol{v} \in \boldsymbol{V}_{\texttt{nearby}}$ **do**
      $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{Steer}(\boldsymbol{v}, \boldsymbol{x}_f)$;
      if $\texttt{AugCollisionFree}(\boldsymbol{u}^*(t), \boldsymbol{x}^*(t), t_f^*)$ **then**
        $e_{\texttt{temp}} \leftarrow (\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*)$;
        $c_e \leftarrow \texttt{Cost}(e_{\texttt{temp}})$;
        if $\mathcal{T}.\texttt{Cost}(\boldsymbol{v}) + c_e < c_f$ **then**
          $c_f \leftarrow \mathcal{T}.\texttt{Cost}(\boldsymbol{v}) + c_e$;
          $e_{\texttt{new}} \leftarrow e_{\texttt{temp}}$;
    if $c_f \neq \infty$ **then**
      $\boldsymbol{V} \leftarrow \boldsymbol{V} \cup \{\boldsymbol{x}_f\}$;
      $\boldsymbol{E} \leftarrow \boldsymbol{E} \cup \{e_{\texttt{new}}\}$;
      $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{GetSol}(\mathcal{T}, \boldsymbol{x}_0, \boldsymbol{x}_f)$;
      return $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*$
  return *fail*;

3.3 Optimal Trajectory Planning: Input Constrained Double Integrator with Drift

This section addresses optimal trajectory planning of a double integrator with drift such that the optimal trajectory minimizes a cost trading off control effort and time. The system's state is

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{v} \end{bmatrix}, \tag{158}$$

where $\boldsymbol{p} = [p_x, p_y, p_z]^T \in \mathbb{R}^3$ is the system's position in meters, and $\boldsymbol{v} = [v_x, v_y, v_z]^T \in \mathbb{R}^3$ is its velocity in meters per second. The system's control input is $\boldsymbol{u} = [u_x, u_y, u_z]^T \in \mathbb{R}^3$, which is the system's acceleration in meters per second squared. The control input of the system is assumed to be constrained in magnitude such that

$$\boldsymbol{\mathcal{U}} = \left\{ \boldsymbol{u} \in \mathbb{R}^3 : \|\boldsymbol{u}\| \leq u_{\texttt{max}} \right\}, \tag{159}$$

where $\|\cdot\|$ denotes the 2-norm and $u_{\texttt{max}} \in \mathbb{R}_{>0}$. The state transition equation of the system is described by

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} + \boldsymbol{c}, \quad A = \begin{bmatrix} 0_3 & I_3 \\ 0_3 & 0_3 \end{bmatrix}, \quad B = \begin{bmatrix} 0_3 \\ I_3 \end{bmatrix}, \tag{160}$$

where $\boldsymbol{c} = [\boldsymbol{c}_v^T, \boldsymbol{c}_a^T]^T$, $\boldsymbol{c}_v, \boldsymbol{c}_a \in \mathbb{R}^3$ are the velocity and acceleration components of the drift term, and where 0_k and I_k denote a $k \times k$ zero and identity matrix for $k \in \mathbb{N}_{>0}$, respectively. Given a control input trajectory $\boldsymbol{u}(t)$, a state trajectory $\boldsymbol{x}(t)$, and a final time t_f, the cost of the trajectory is evaluated according to (145), where $R \in \mathbb{R}^{3\times 3}$ is assumed to be symmetric and positive definite and $C_I \in \mathbb{R}_{>0}$. Three functions are required to be defined so that Kinodynamic RRT* may be used to solve (147): the cost-to-go function, the steering function, and the local trajectory function. The cost-to-go function and the steering function have previously been defined in Section 2.3.2. Therefore, only the augmented local trajectory function from Definition 6 is required to be designed in this section.

Given an initial state $\boldsymbol{x}_1 \in \boldsymbol{\mathcal{X}}$ and a final state $\boldsymbol{x}_2 \in \boldsymbol{\mathcal{X}}$, let $\boldsymbol{u}^*_{1,2}(t)$, $\boldsymbol{x}^*_{1,2}(t)$, and $t^*_{f_{1,2}}$ denote the unconstrained optimal control input and state trajectories and the optimal final time to steer the system from $\boldsymbol{x}_1$ to $\boldsymbol{x}_2$. The final time $\tilde{t}^*_{f_{1,2}}$ is obtained from the augmented local trajectory function as the solution of (151). It is shown in Section 3.2 that the solution of (151) can be obtained according to

$$\tilde{t}^*_{f_{1,2}} = \min\left\{ t_{\eta_{1,2}}, t_{u_{1,2}} \right\}, \tag{161}$$

where $t_{\eta_{1,2}}$ is given by (154) and $t_{u_{1,2}}$ is given by (155). The solution of (154) is discussed in Section 2.3.2, where $t_{\eta_{1,2}}$ is obtained as the positive real solution of

$$C_3 t_{\eta_{1,2}}^3 + C_2 t_{\eta_{1,2}}^2 + C_1 t_{\eta_{1,2}} - \tilde{\eta} = 0, \tag{162}$$

where the coefficients C_i, $i \in \{1,2,3\}$, are given by (79) and $\tilde{\eta}$ is given by (83). It can also be seen that, given (159), (155) is equivalent to

$$\begin{array}{rcll} t_{u_{1,2}} & = & \underset{\tau \in \left(0, t^*_{f_{1,2}}\right]}{\operatorname{argmax}} \ \tau & \\ & & \texttt{s.t.} \quad \boldsymbol{u}_{1,2}^{*^T}(t)\, \boldsymbol{u}_{1,2}^*(t) \leq u_{\texttt{max}}^2, \ \forall\, t \in [0,\tau] & . \end{array} \tag{163}$$

Using the expression of $\boldsymbol{u}_{1,2}^*(t)$ from (72) yields

$$\boldsymbol{u}_{1,2}^{*^T}(t)\, \boldsymbol{u}_{1,2}^*(t) = U_2 t^2 + U_1 t + U_0, \tag{164}$$

where

$$\begin{aligned} U_2 &= \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix} \frac{144}{t^{*^6}_{f_{1,2}}} I_3 & \frac{72}{t^{*^5}_{f_{1,2}}} I_3 & -\frac{144}{t^{*^6}_{f_{1,2}}} I_3 & \frac{72}{t^{*^5}_{f_{1,2}}} I_3 & \frac{144}{t^{*^5}_{f_{1,2}}} I_3 & 0_3 \\ \frac{72}{t^{*^5}_{f_{1,2}}} I_3 & \frac{36}{t^{*^4}_{f_{1,2}}} I_3 & -\frac{72}{t^{*^5}_{f_{1,2}}} I_3 & \frac{36}{t^{*^4}_{f_{1,2}}} I_3 & \frac{72}{t^{*^4}_{f_{1,2}}} I_3 & 0_3 \\ -\frac{144}{t^{*^6}_{f_{1,2}}} I_3 & -\frac{72}{t^{*^5}_{f_{1,2}}} I_3 & \frac{144}{t^{*^6}_{f_{1,2}}} I_3 & -\frac{72}{t^{*^5}_{f_{1,2}}} I_3 & -\frac{144}{t^{*^5}_{f_{1,2}}} I_3 & 0_3 \\ \frac{72}{t^{*^5}_{f_{1,2}}} I_3 & \frac{36}{t^{*^4}_{f_{1,2}}} I_3 & -\frac{72}{t^{*^5}_{f_{1,2}}} I_3 & \frac{36}{t^{*^4}_{f_{1,2}}} I_3 & \frac{72}{t^{*^4}_{f_{1,2}}} I_3 & 0_3 \\ \frac{144}{t^{*^5}_{f_{1,2}}} I_3 & \frac{72}{t^{*^4}_{f_{1,2}}} I_3 & -\frac{144}{t^{*^5}_{f_{1,2}}} I_3 & \frac{72}{t^{*^4}_{f_{1,2}}} I_3 & \frac{144}{t^{*^4}_{f_{1,2}}} I_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{c} \end{bmatrix} \\ &= \frac{36}{t^{*^6}_{f_{1,2}}} \left(2\left(\boldsymbol{p}_1 - \boldsymbol{p}_2\right) + t^*_{f_{1,2}} \left(2\boldsymbol{c}_v + \boldsymbol{v}_1 + \boldsymbol{v}_2\right) \right)^2 \end{aligned} \tag{165a}$$

$$U_1 = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix} -\frac{144}{t^{*5}_{f_{1,2}}} I_3 & -\frac{84}{t^{*4}_{f_{1,2}}} I_3 & \frac{144}{t^{*5}_{f_{1,2}}} I_3 & -\frac{60}{t^{*4}_{f_{1,2}}} I_3 & -\frac{144}{t^{*4}_{f_{1,2}}} I_3 & -\frac{12}{t^{*3}_{f_{1,2}}} I_3 \\ -\frac{84}{t^{*4}_{f_{1,2}}} I_3 & -\frac{48}{t^{*3}_{f_{1,2}}} I_3 & \frac{84}{t^{*4}_{f_{1,2}}} I_3 & -\frac{36}{t^{*3}_{f_{1,2}}} I_3 & -\frac{84}{t^{*3}_{f_{1,2}}} I_3 & -\frac{6}{t^{*2}_{f_{1,2}}} I_3 \\ \frac{144}{t^{*5}_{f_{1,2}}} I_3 & \frac{84}{t^{*4}_{f_{1,2}}} I_3 & -\frac{144}{t^{*5}_{f_{1,2}}} I_3 & \frac{60}{t^{*4}_{f_{1,2}}} I_3 & \frac{144}{t^{*4}_{f_{1,2}}} I_3 & \frac{12}{t^{*3}_{f_{1,2}}} I_3 \\ -\frac{60}{t^{*4}_{f_{1,2}}} I_3 & -\frac{36}{t^{*3}_{f_{1,2}}} I_3 & \frac{60}{t^{*4}_{f_{1,2}}} I_3 & -\frac{24}{t^{*3}_{f_{1,2}}} I_3 & -\frac{60}{t^{*3}_{f_{1,2}}} I_3 & -\frac{6}{t^{*2}_{f_{1,2}}} I_3 \\ -\frac{144}{t^{*4}_{f_{1,2}}} I_3 & -\frac{84}{t^{*3}_{f_{1,2}}} I_3 & \frac{144}{t^{*4}_{f_{1,2}}} I_3 & -\frac{60}{t^{*3}_{f_{1,2}}} I_3 & -\frac{144}{t^{*3}_{f_{1,2}}} I_3 & -\frac{12}{t^{*2}_{f_{1,2}}} I_3 \\ -\frac{12}{t^{*3}_{f_{1,2}}} I_3 & -\frac{6}{t^{*2}_{f_{1,2}}} I_3 & \frac{12}{t^{*3}_{f_{1,2}}} I_3 & -\frac{6}{t^{*2}_{f_{1,2}}} I_3 & -\frac{12}{t^{*2}_{f_{1,2}}} I_3 & 0_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{c} \end{bmatrix} \tag{165b}$$

$$= -\frac{12}{t^{*5}_{f_{1,2}}} \left(2\left(\boldsymbol{p}_1 - \boldsymbol{p}_2\right) + t^*_{f_{1,2}}\left(2\boldsymbol{c}_v + \boldsymbol{v}_1 + \boldsymbol{v}_2\right)\right) \left(6\left(\boldsymbol{p}_1 - \boldsymbol{p}_2\right) + t^*_{f_{1,2}}\left(6\boldsymbol{c}_v + 4\boldsymbol{v}_1 + 2\boldsymbol{v}_2\right) + t^{*^2}_{f_{1,2}}\boldsymbol{c}_a\right)$$

$$U_0 = \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix} \frac{36}{t^{*4}_{f_{1,2}}} I_3 & \frac{24}{t^{*3}_{f_{1,2}}} I_3 & -\frac{36}{t^{*4}_{f_{1,2}}} I_3 & \frac{12}{t^{*3}_{f_{1,2}}} I_3 & \frac{36}{t^{*3}_{f_{1,2}}} I_3 & \frac{6}{t^{*2}_{f_{1,2}}} I_3 \\ \frac{24}{t^{*3}_{f_{1,2}}} I_3 & \frac{16}{t^{*2}_{f_{1,2}}} I_3 & -\frac{24}{t^{*3}_{f_{1,2}}} I_3 & \frac{8}{t^{*2}_{f_{1,2}}} I_3 & \frac{24}{t^{*2}_{f_{1,2}}} I_3 & \frac{4}{t^*_{f_{1,2}}} I_3 \\ -\frac{36}{t^{*4}_{f_{1,2}}} I_3 & -\frac{24}{t^{*3}_{f_{1,2}}} I_3 & \frac{36}{t^{*4}_{f_{1,2}}} I_3 & -\frac{12}{t^{*3}_{f_{1,2}}} I_3 & -\frac{36}{t^{*3}_{f_{1,2}}} I_3 & -\frac{6}{t^{*2}_{f_{1,2}}} I_3 \\ \frac{12}{t^{*3}_{f_{1,2}}} I_3 & \frac{8}{t^{*2}_{f_{1,2}}} I_3 & -\frac{12}{t^{*3}_{f_{1,2}}} I_3 & \frac{4}{t^{*2}_{f_{1,2}}} I_3 & \frac{12}{t^{*2}_{f_{1,2}}} I_3 & \frac{2}{t^*_{f_{1,2}}} I_3 \\ \frac{36}{t^{*3}_{f_{1,2}}} I_3 & \frac{24}{t^{*2}_{f_{1,2}}} I_3 & -\frac{36}{t^{*3}_{f_{1,2}}} I_3 & \frac{12}{t^{*2}_{f_{1,2}}} I_3 & \frac{36}{t^{*2}_{f_{1,2}}} I_3 & \frac{6}{t^*_{f_{1,2}}} I_3 \\ \frac{6}{t^{*2}_{f_{1,2}}} I_3 & \frac{4}{t^*_{f_{1,2}}} I_3 & -\frac{6}{t^{*2}_{f_{1,2}}} I_3 & \frac{2}{t^*_{f_{1,2}}} I_3 & \frac{6}{t^*_{f_{1,2}}} I_3 & I_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \\ \boldsymbol{c} \end{bmatrix} \tag{165c}$$

$$= \frac{1}{t^{*4}_{f_{1,2}}} \left(6\left(\boldsymbol{p}_1 - \boldsymbol{p}_2\right) + t^*_{f_{1,2}}\left(6\boldsymbol{c}_v + 4\boldsymbol{v}_1 + 2\boldsymbol{v}_2\right) + t^{*^2}_{f_{1,2}}\boldsymbol{c}_a\right)^2$$

It can be seen from (165) that $U_0 \geq 0$, $U_2 \geq 0$, and $U_1^2 = 4U_2U_0$. The set of time instants when the inputs are at the boundary of $\boldsymbol{\mathcal{U}}$ can be obtained as the real solutions of

$$U_2 t^2 + U_1 t + U_0 - u^2_{\max} = 0. \tag{166}$$

It can be seen from (164) that the control input magnitude at $t = 0$ is equal to U_0. Therefore, if $U_0 > u^2_{\max}$ then $\boldsymbol{u}^*_{1,2}(0) \notin \boldsymbol{\mathcal{U}}$ and no solution of (163) exists. Given that $U_1^2 = 4U_2U_0$, if $U_2 = 0$ then $U_1 = 0$ and (164) is constant and equal to U_0 $\forall\, t \in [0, t^*_{f_{1,2}}]$. For this case the solution of (163) is $t_{u_{1,2}} = t^*_{f_{1,2}}$ if $U_0 \leq u^2_{\max}$ and does not exist otherwise. If $U_2 \neq 0$, which implies that $U_2 > 0$ since $U_2 \geq 0$, then the time instants that the control inputs are at the boundary of $\boldsymbol{\mathcal{U}}$ can be evaluated as

$$t = \frac{-U_1}{2U_2} \pm \frac{\sqrt{U_1^2 - 4U_2\left(U_0 - u^2_{\max}\right)}}{2U_2}. \tag{167}$$

If $U_2 > 0$ and $U_0 < u_{\texttt{max}}^2$ then, given that $U_1^2 = 4U_2U_0$, it can be seen that (167) simplifies to

$$t = \frac{-U_1 \pm 2u_{\texttt{max}}\sqrt{U_2}}{2U_2}. \tag{168}$$

Furthermore, it can be seen from (164) that the control input magnitude at $t = 0$ is an interior point of $\boldsymbol{\mathcal{U}}$ and there exists a positive real solution of (167). Since $U_2 > 0$ it can be seen that the solution of (163) is

$$\begin{array}{rcll} t_{u_{1,2}} & = & \frac{-U_1 - 2u_{\texttt{max}}\sqrt{U_2}}{2U_2} & \texttt{if} \quad -U_1 - 2u_{\texttt{max}}\sqrt{U_2} > 0 \\ t_{u_{1,2}} & = & \frac{-U_1 + 2u_{\texttt{max}}\sqrt{U_2}}{2U_2} & \texttt{if} \quad |U_1| < 2u_{\texttt{max}}\sqrt{U_2} \end{array}, \tag{169}$$

and no solution exists otherwise. If $U_0 = u_{\texttt{max}}^2$ the the control input magnitude is at the boundary of $\boldsymbol{\mathcal{U}}$ at $t = 0$ and (167) simplifies to

$$t = \frac{-U_1}{2U_2} \pm \frac{U_1}{2U_2}. \tag{170}$$

Since $U_2 > 0$ it can be seen that the solution of (163) for this case is $t_{u_{1,2}} = -\frac{U_1}{U_2}$ if $U_1 < 0$, and no solution exists otherwise. In summary, the positive real solution of (163) can therefore be obtained as

$$t_{u_{1,2}} = \begin{cases} t^*_{f_{1,2}} & \texttt{if} \quad U_2 = U_1 = 0, U_0 \leq u_{\texttt{max}}^2 \\ \frac{-U_1 - 2u_{\texttt{max}}\sqrt{U_2}}{2U_2} & \texttt{if} \quad U_2 > 0, U_1 < -2u_{\texttt{max}}\sqrt{U_2}, U_0 < u_{\texttt{max}}^2 \\ \frac{-U_1 + 2u_{\texttt{max}}\sqrt{U_2}}{2U_2} & \texttt{if} \quad U_2 > 0, |U_1| < 2u_{\texttt{max}}\sqrt{U_2}, U_0 < u_{\texttt{max}}^2 \\ -\frac{U_1}{U_2} & \texttt{if} \quad U_2 > 0, U_1 < 0, U_0 = u_{\texttt{max}}^2 \\ \texttt{no solution} & \texttt{if} \quad \texttt{otherwise} \end{cases}. \tag{171}$$

3.3.1 Validation and Study of Optimality

This section shows that the modified Kinodynamic RRT* algorithm consists of edges that are characterized by trajectories that satisfy the state constraints of the system and that the Kinodynamic RRT* trajectory planner from Chapter 2 does not. The system's state space is described by

$$\boldsymbol{\mathcal{X}} = \left\{ \boldsymbol{x} \in \mathbb{R}^6 : \begin{array}{c} -1 \leq p_x \leq 2,\ p_y = 0,\ p_z = 0, \\ |v_x| \leq 1,\ v_y = 0,\ v_z = 0 \end{array} \right\} \tag{172}$$

and the obstacle space is described by $\boldsymbol{\mathcal{X}} = \varnothing$, which implies that $\boldsymbol{\mathcal{X}}_{\texttt{free}} = \boldsymbol{\mathcal{X}}$. Additionally, the control input space is

$$\boldsymbol{\mathcal{U}} = \{\|\boldsymbol{u}\| \leq 1.5\}. \tag{173}$$

The initial state and final state are

$$\begin{aligned} \boldsymbol{x}_0 &= \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \\ \boldsymbol{x}_f &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T \end{aligned} \tag{174}$$

and the drift term is

$$\boldsymbol{c} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T. \tag{175}$$

Finally, the control input weighting matrix is $R = I_3$ and the cost index is $C_I = 1$.

The unconstrained optimal control solution was derived in Section 2.3.2.1, where it can be seen that the resulting state and control input trajectories satisfy the state and control input constraints of the problem proposed in this section. As a consequence of this, the resulting unconstrained optimal control solution of Section 2.3.2.1 is also the optimal control solution of the constrained problem. The number of iterations of the Kinodynamic RRT* solver was selected as $N_{\texttt{iter}} = 1000$, the cost-to-go upper bound was selected as $\eta = 1$, and the reachability tuning parameter as $\gamma = 1000$. It can be seen that the problem addressed in this section is the same as that in Section 2.3.2.1, except that the control input of the system is constrained to belong to $\boldsymbol{\mathcal{U}}$. The simulation is performed in MATLAB R2018a on an Intel Core i5 2.9 GHz processor with 16GB of RAM. Figure 24 shows the optimal state trajectory and compares it with the approximate solution obtained using the modified Kinodynamic RRT*. The initial state is shown as the green diamond and the final state is shown as the red diamond. The optimal state trajectory is described by the green curve, which has a trajectory cost of 2.7464 $\frac{m^2}{s^3}$. The first approximate solution obtained by the modified Kinodynamic RRT* algorithm was obtained in 94 iterations. This result is shown in dark blue, which has a cost of 2.8008 $\frac{m^2}{s^3}$ and represents an error of 2%. This approximation was then improved as the solver continued to iterate and found the state trajectory shown in red, which has a cost of 2.7896 $\frac{m^2}{s^3}$ and represents an error of 0.4%. The approximation was then improved four more times, where the cost of each approximation is shown in the legend of Figure 24. The curve in light blue shows the final approximation found by the solver following the completion of 1000 iterations. It can be seen that this approximation closely matches the optimal solution and consists of an error of less than 0.004%. Figure 25 shows the optimal control input trajectory in green, which produces the state trajectory shown in green in Figure 24. Additionally, the approximate control input trajectory obtained from the modified Kinodynamic RRT* algorithm is shown in black, which produces the state trajectory shown in light blue in Figure 24. Figure 26 shows $\mathcal{T}$, where the initial state is shown as the green diamond and the final state is shown as the red diamond, the vertices of the graph are represented by the points, and the solid curves describe the state trajectory components of each edge. Finally, Figures

27 and 28 compare the edges of the modified Kinodynamic RRT* with those obtained from Section 2.3.2.1. Each line segment in these figures describes an unconstrained optimal control input trajectory of an edge in the tree. Segments of edges that satisfy the control input constraints described by $\boldsymbol{\mathcal{U}}$ are shown in green, whereas those that do not satisfy the constraints are shown in red. It can be seen from Figure 27 that, since the Kinodynamic RRT* algorithm in Chapter 2 assumes unconstrained control inputs, many of the edges of the tree do not satisfy the control input constraints described by $\boldsymbol{\mathcal{U}}$. However, it can be seen from Figure 28 that, due to the modifications that address systems with constrained inputs, all edges of the tree are described by unconstrained optimal control input trajectories that satisfy the constraints described by $\boldsymbol{\mathcal{U}}$.

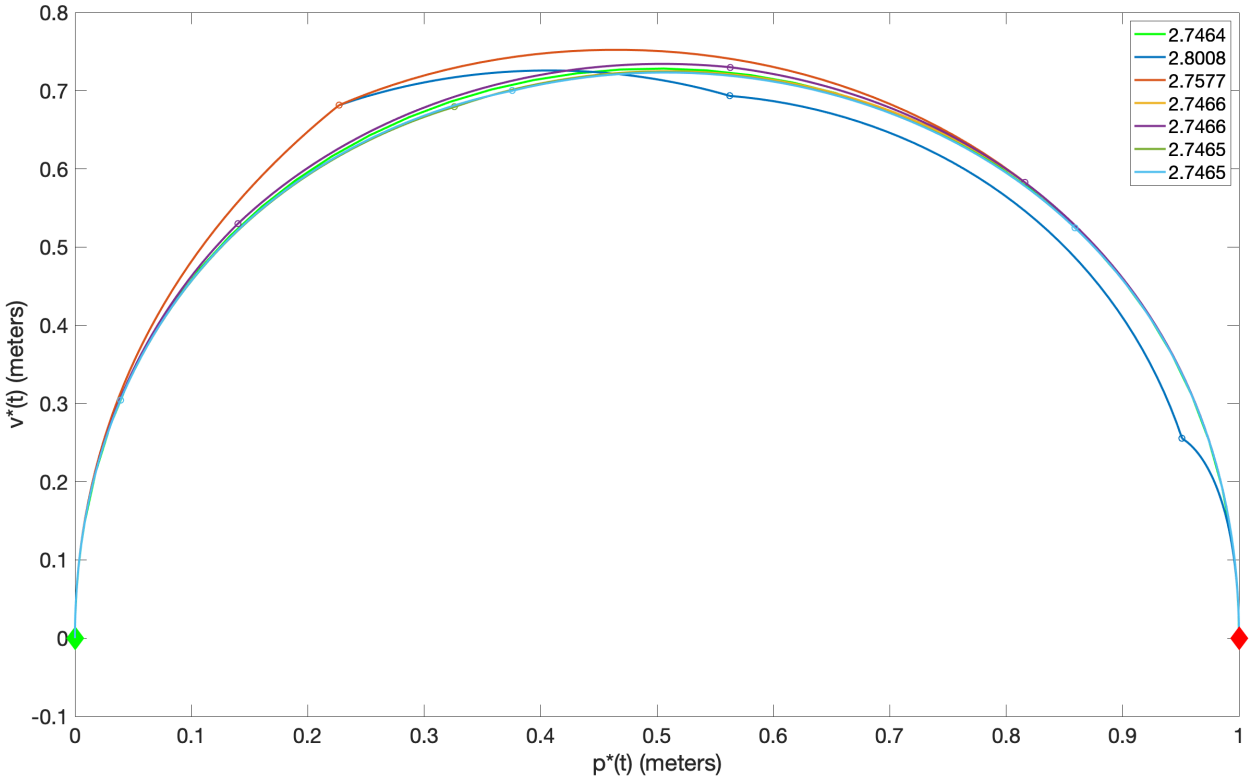


Figure 24: 1D double integrator: optimal state and Kinodynamic RRT* approximate state trajectories.

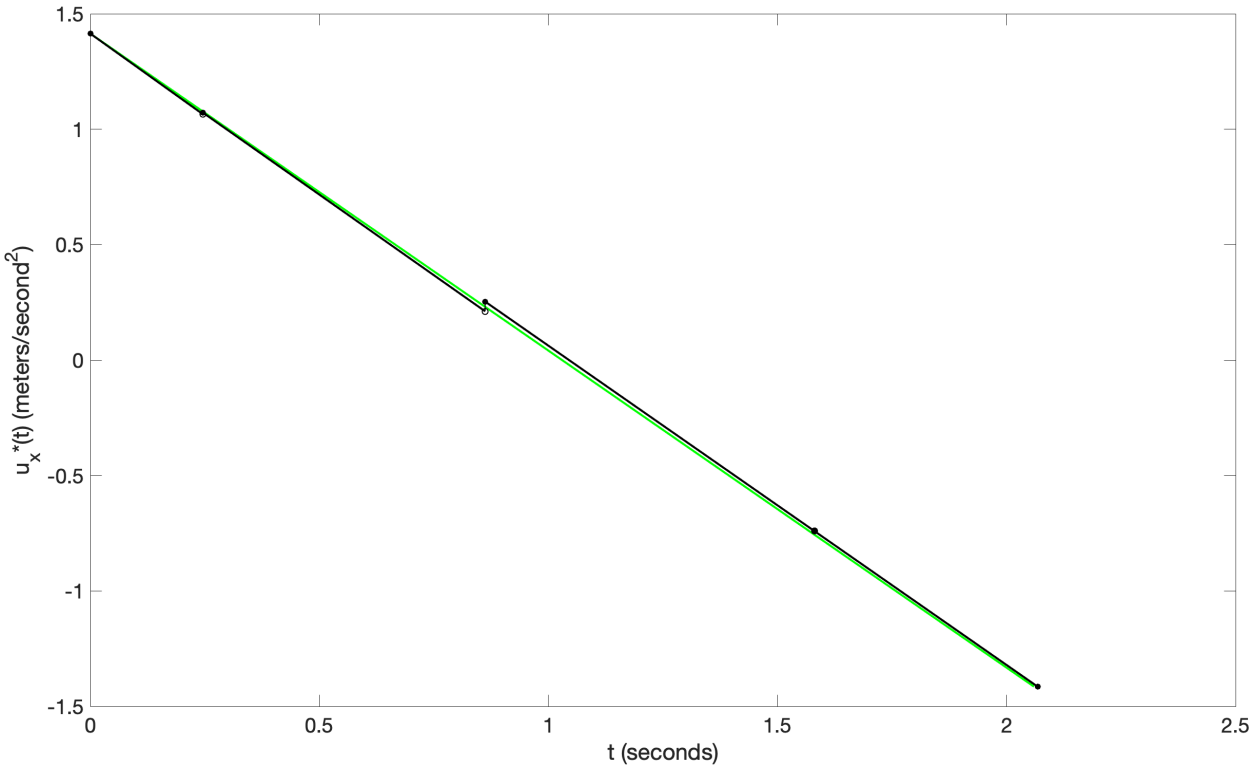


Figure 25: 1D double integrator: optimal state and Kinodynamic RRT* approximate input trajectories.

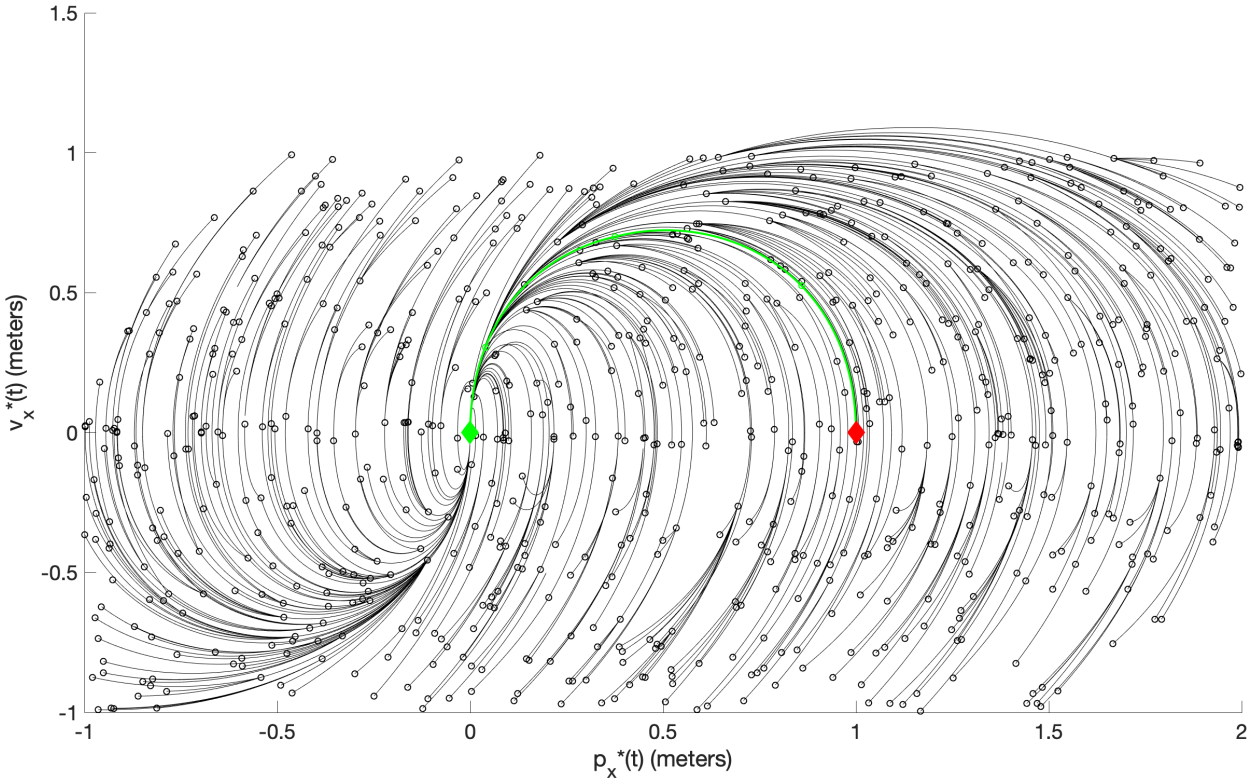


Figure 26: 1D double integrator: Kinodynamic RRT* tree in the state space.

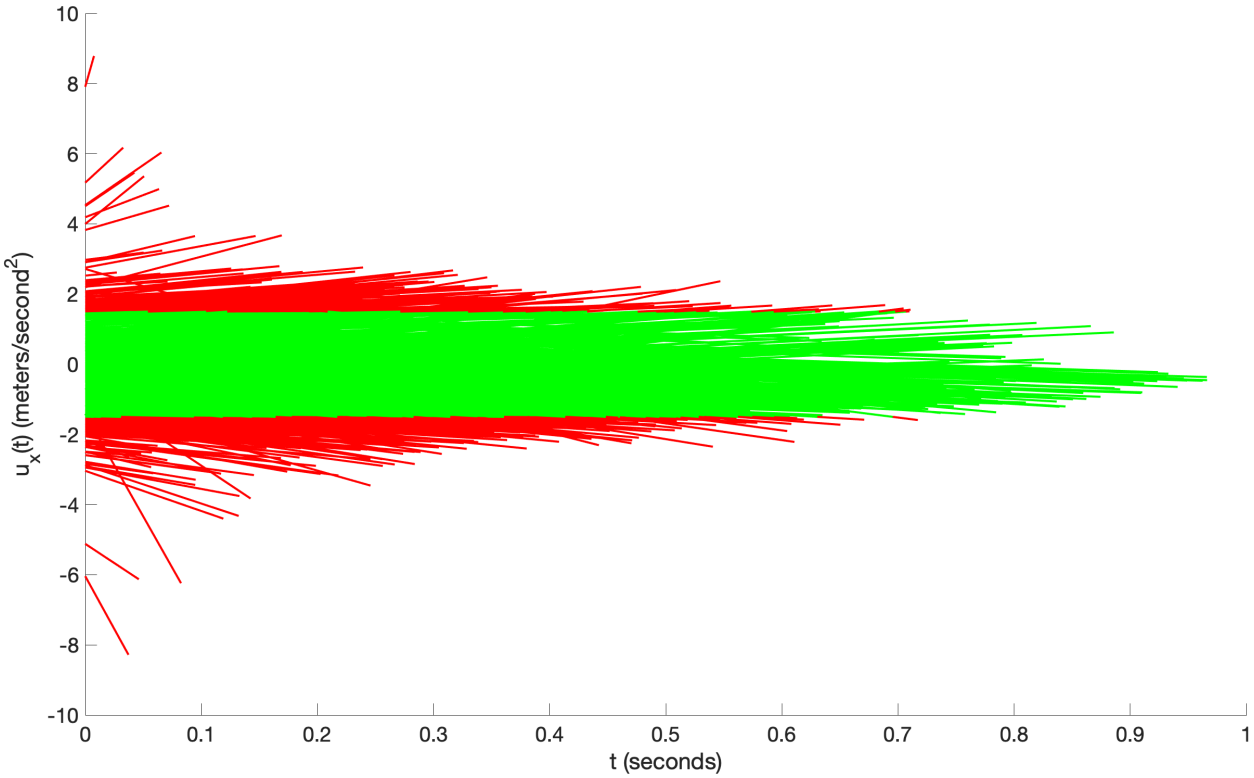


Figure 27: 1D double integrator: Kinodynamic RRT* input trajectories.

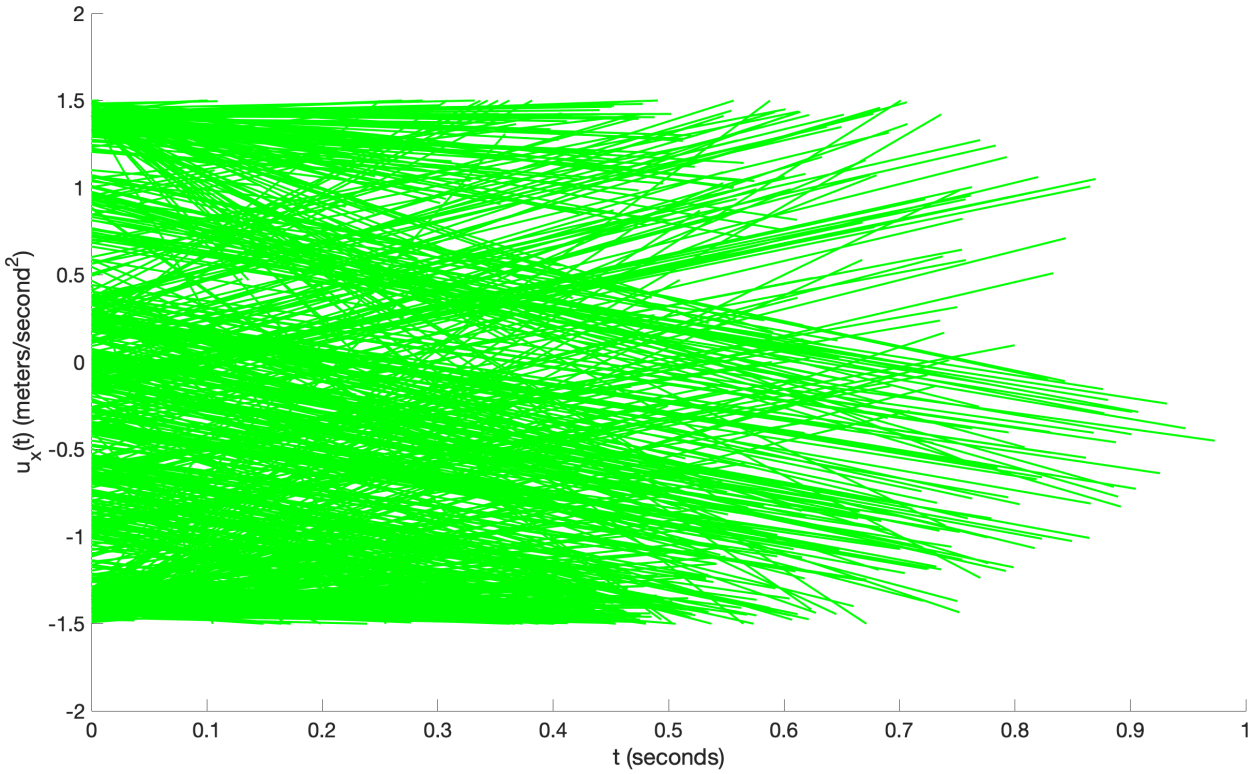


Figure 28: 1D double integrator: modified Kinodynamic RRT* input trajectories.

3.3.2 Example

An optimal trajectory planning problem in a non-convex bounded state space subject to a maximum control input magnitude is now addressed, which is solved using optimal control theory and Kinodynamic RRT*. The system's state space is constrained to

$$\boldsymbol{\mathcal{X}} = \left\{ \boldsymbol{x} \in \mathbb{R}^6 : \begin{array}{c} -100 \leq p_x, p_y \leq 100,\ 0 \leq p_z \leq 100, \\ \|\boldsymbol{v}\| \leq 20 \end{array} \right\} \tag{176}$$

and the obstacle space is described by

$$\boldsymbol{\mathcal{X}}_{\texttt{obs}} = \bigcup_{i=1}^{i=4} \bigcup_{j=1}^{j=4} \boldsymbol{\mathcal{X}}_{\texttt{obs}_{ij}}, \tag{177}$$

where

$$\boldsymbol{\mathcal{X}}_{\texttt{obs}_{ij}} = \left\{ \boldsymbol{x} \in \mathbb{R}^6 : \left\| \begin{bmatrix} p_x + 100 - 40i \\ p_y + 100 - 40j \end{bmatrix} \right\| < 10 \right\}. \tag{178}$$

The obstacle free space $\boldsymbol{\mathcal{X}}_{\texttt{free}}$ is obtained according to (146). Additionally, the control input space is defined according to (159), where $u_{\texttt{max}}$ is 20. The initial and final states of the system are

$$\begin{aligned} \boldsymbol{x}_0 &= \begin{bmatrix} -40 & -40 & 40 & 0 & 0 & 0 \end{bmatrix}^T \\ \boldsymbol{x}_f &= \begin{bmatrix} 40 & 40 & 80 & 0 & 0 & 0 \end{bmatrix}^T \end{aligned}, \tag{179}$$

and the affine term is set to

$$\begin{aligned} \boldsymbol{c} &= \begin{bmatrix} \boldsymbol{w}^T & 0 & 0 & -g \end{bmatrix}^T \\ \boldsymbol{w} &= \begin{bmatrix} 3 & -2 & 0.5 \end{bmatrix}^T \end{aligned}, \tag{180}$$

where $g = 9.8\ \frac{m}{s^2}$ is the acceleration due to gravity and the vector $\boldsymbol{w}$ describes a constant wind velocity in meters per second. Finally, the control input weighting matrix is chosen as $R = I_3$ and the cost index as $C_I = 10$. Note that this is an extension of the problem addressed in Section 2.3.2.2 with the additional constraint on the control input magnitude. The simulation is performed in MATLAB R2018a on an Intel Core i5 2.9 GHz processor with 16GB of RAM.

The maximum number of iterations was selected as $N_{\texttt{iter}} = 3000$, the cost-to-go upper bound as $\eta = 300$, and the reachability region tuning parameter as $\gamma = 1000$. Figure 29 shows the position vector of the optimal state trajectory as the solid black curve, where the points along the curve denote the vertices in $\boldsymbol{V}$ along the optimal trajectory, the green diamond indicates $\boldsymbol{p}_0$, the red diamond indicates $\boldsymbol{p}_f$, the blue vectors represent

the constant wind velocity vector $\boldsymbol{w}$, and the red cylinders represent the obstacles of $\boldsymbol{\mathcal{X}}_{\texttt{obs}}$. Figure 30 shows the velocity components of the optimal state trajectory, the optimal velocity magnitude, and the maximum velocity. Figures 31 – 33 present the optimal control input trajectory components, and Figure 34 shows the optimal control input trajectory magnitude. In these figures the solid lines describe the optimal control input of each trajectory segment, the dashed lines represent discontinuities between two segments, the empty circles show the terminal control input of a segment, and the filled circles show the initial control input of a segment. Furthermore, in Figure 34 the maximum control input magnitude is shown as the dashed red line.

These results show how the modified Kinodynamic RRT* trajectory planner is capable of efficiently planning optimal control trajectories of input constrained affine systems. In particular, the optimal control input magnitude shown in Figure 34 can be compared with that of Figure 23 on page 67. The augmentations made to the Kinodynamic RRT* trajectory planner guarantee that all edges of the tree are described by trajectories that satisfy both the obstacle-avoidance state constraints of the problem and the control input constraints of the system. As a result, any solution obtained using the modified Kinodynamic RRT* algorithm is guaranteed to satisfy the control input constraints of the system.

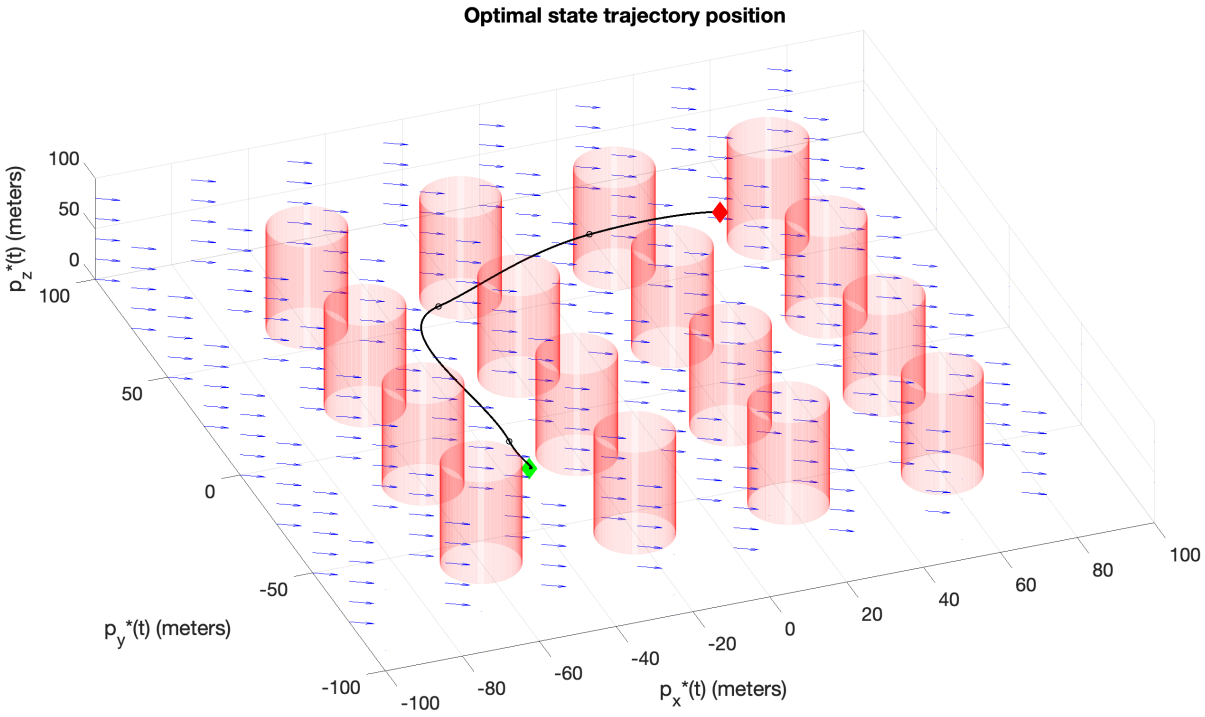


Figure 29: Input constrained double integrator with drift: optimal position trajectory.

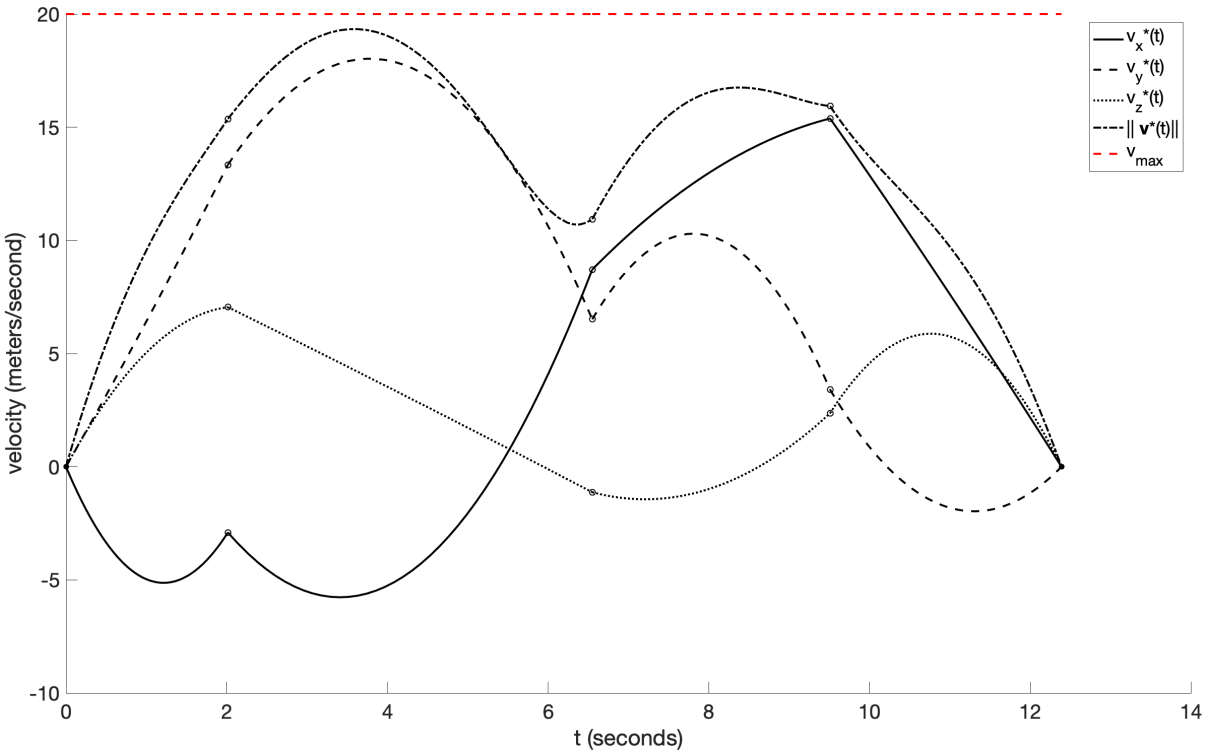


Figure 30: Input constrained double integrator with drift: optimal velocity components.

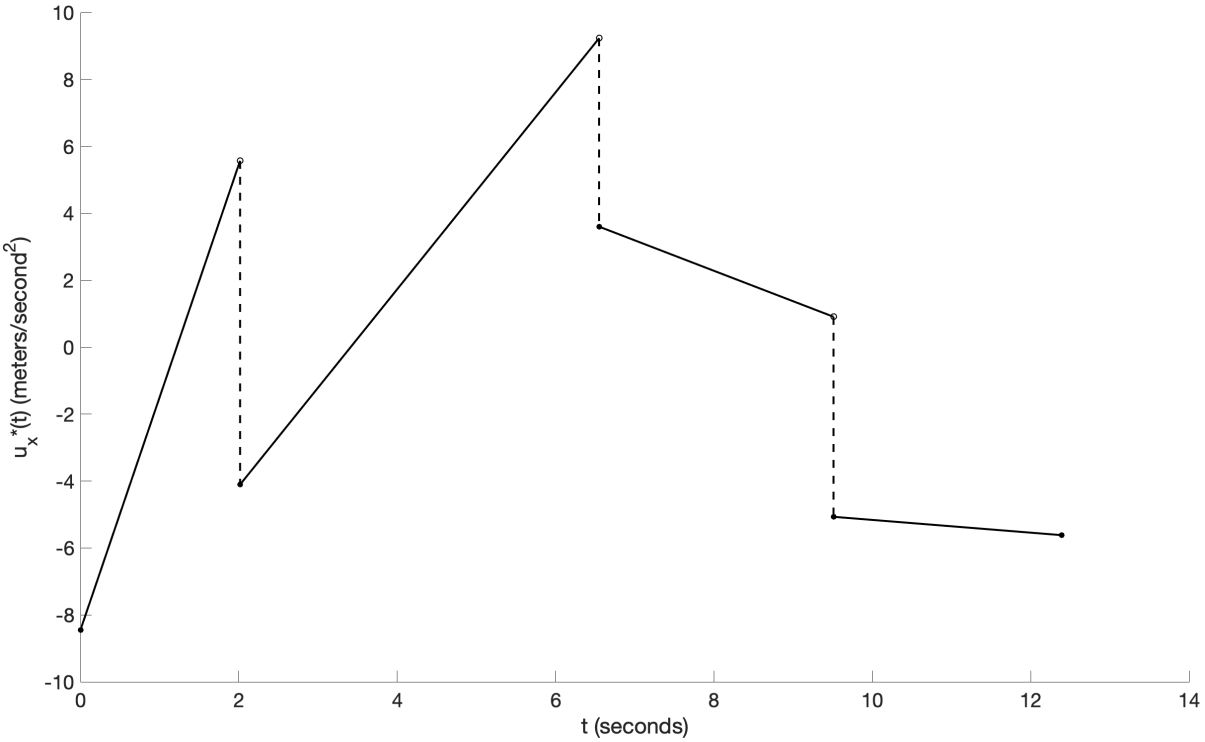


Figure 31: Input constrained double integrator with drift: optimal control input $u_x^*(t)$.

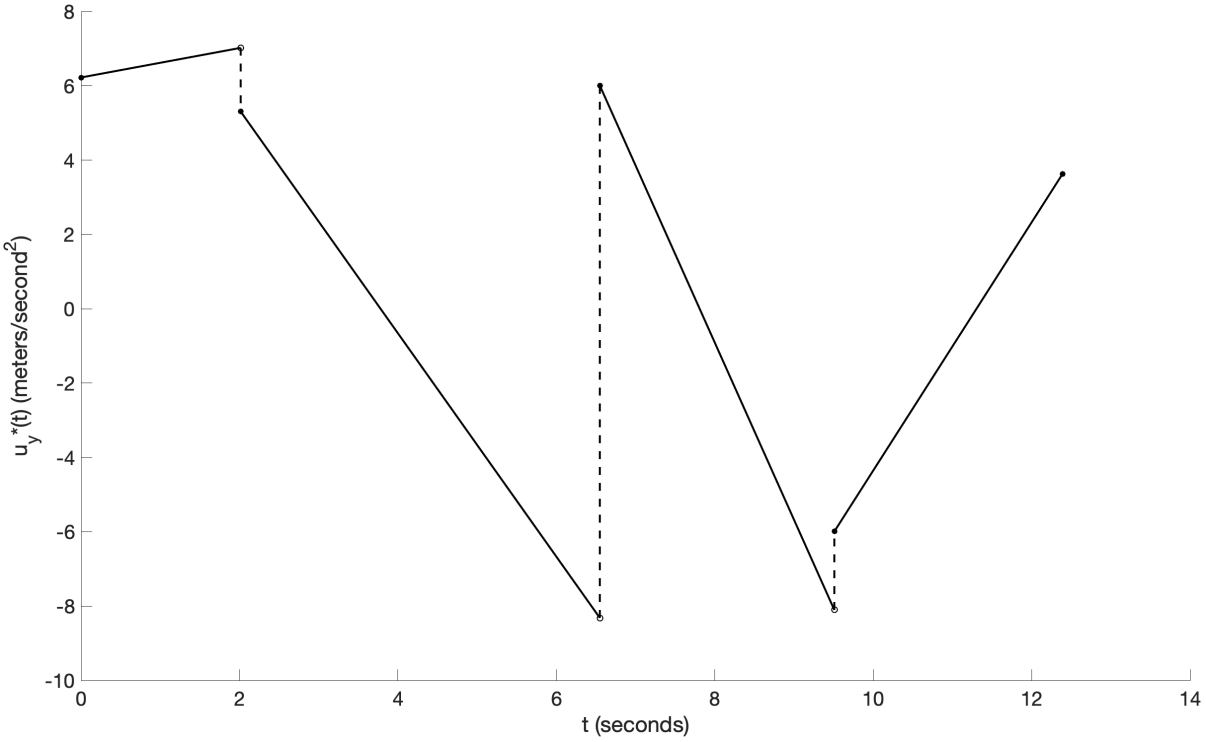


Figure 32: Input constrained double integrator with drift: optimal control input $u_y^*(t)$.

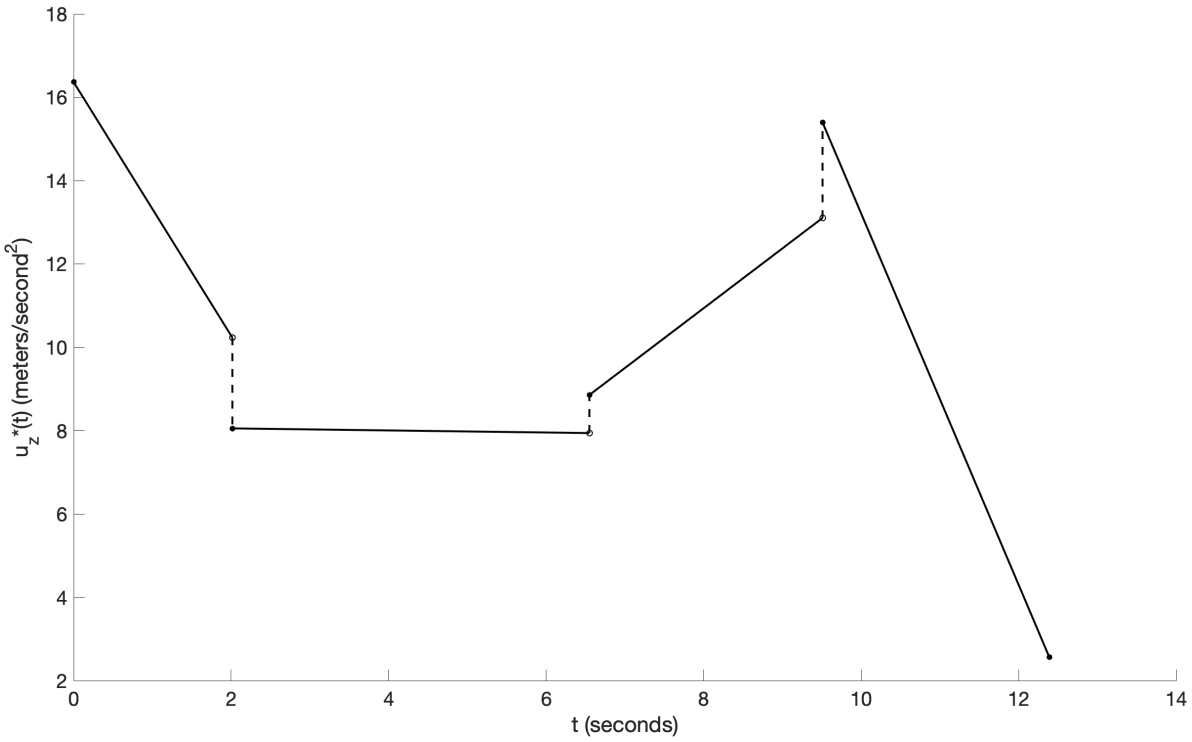


Figure 33: Input constrained double integrator with drift: optimal control input $u_z^*(t)$.

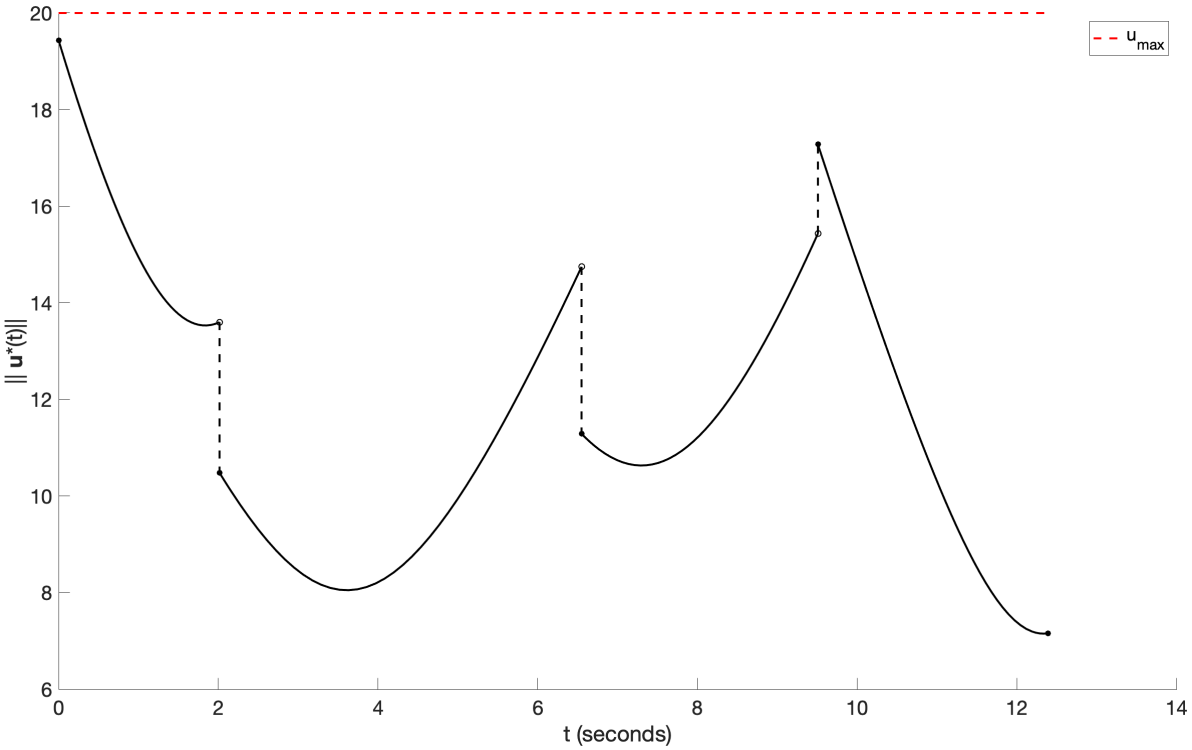


Figure 34: Input constrained double integrator with drift: optimal control input magnitude.

Chapter 4

Directed Sampling for Kinodynamic RRT*

Slow convergence is a common problem for many trajectory planning algorithms. Although sampling-based trajectory planners generally demonstrate faster convergence speeds than many alternative techniques, they still struggle to meet the convergence speeds necessary for real-time trajectory planning of many systems. Typically, random state space samples are generated according to a uniform distribution on $\boldsymbol{\mathcal{X}}_{\texttt{free}}$, where each component of the sample is generated independently of the others. As a consequence of this, a large number of randomly generated state space samples lie in regions where a solution is unlikely to exist. This results in wasted computation time expended on generating, connecting, and optimizing these samples.

This chapter proposes a sampling method that applies a Gaussian distribution to the position subspace of the system's state space, rather than a uniform distribution. This allows to direct the randomly generated position vectors of the state samples to a region where a solution is likely to exist. As a result of this, an approximate solution of the trajectory planning problem is obtained in less iterations and with less computation time when compared with other approaches proposed in the literature. Furthermore, given a maximum number of iterations or a maximum computation time, the approximate solution is on average a lower cost approximation compared to other sampling methods. This chapter is organized as follows. First, some preliminary notions are introduced. Then, the directed sampling problem is formulated and solved. Finally, simulations results and a comparison with other techniques proposed in the literature are discussed.

4.1 Preliminaries

The diagonal matrix of a vector $\boldsymbol{v} \in \mathbb{R}^n$, where $n \in \mathbb{N}_{>0}$ and $\boldsymbol{v} = [v_1, v_2, \ldots, v_n]$, is described by the function

$$\Lambda : \mathbb{R}^n \to \mathbb{R}^{n \times n}, \Lambda\left(\boldsymbol{v}\right) = \begin{bmatrix} v_1 & 0 & \ldots & 0 \\ 0 & v_2 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \ldots & 0 & v_n \end{bmatrix}. \tag{181}$$

Let $\boldsymbol{p} \in \mathbb{R}^3$ denote a position in three-dimensional Euclidean space. The closed convex hull of an ellipsoid in this space is described by

$$\Omega\left(\boldsymbol{p}_c, S\right) = \left\{\boldsymbol{p} \in \mathbb{R}^3 : \left(\boldsymbol{p} - \boldsymbol{p}_c\right)^T S \left(\boldsymbol{p} - \boldsymbol{p}_c\right) \leq 1\right\}, \tag{182}$$

where $\boldsymbol{p}_c \in \mathbb{R}^3$ is the center of the ellipsoid and $S \in \mathbb{R}^{3\times 3}$ is a positive-definite symmetric matrix describing the rotation and the semi-axis lengths of the ellipsoid. The matrix S may be described by

$$S = R_e \Lambda\left(\boldsymbol{s}\right)^{-2} R_e^{-1}, \tag{183}$$

where $R_e \in \mathbb{R}^{3\times 3}$ is an orthogonal matrix that describes the rotation of the ellipsoid, and $\boldsymbol{s} = [s_x, s_y, s_z]$ is the vector of semi-axis lengths of the ellipsoid, where $s_x, s_y, s_z \in \mathbb{R}_{>0}$ are the semi-axis lengths along the ellipsoid's x, y, and z-axes respectively. Let $\boldsymbol{p}_r \in \mathbb{R}^3$ denote a random vector of independent Gaussian random variables with mean $\boldsymbol{\mu} \in \mathbb{R}^3$ and covariance $\Sigma \in \mathbb{R}^{3\times 3}$. The set of random position vectors of a Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance Σ that have a probability of being generated that is less than, or equal to, ρ is described by

$$\boldsymbol{P}_r = \left\{\boldsymbol{p}_r \in \mathbb{R}^3 : \left(\boldsymbol{p}_r - \boldsymbol{\mu}\right)^T \Sigma^{-1} \left(\boldsymbol{p}_r - \boldsymbol{\mu}\right) \leq \chi_3^2\left(1-\rho\right)\right\}, \tag{184}$$

where $\chi_3^2\left(1-\rho\right)$ is the 3 degree of freedom chi-squared statistic for probability $1-\rho$. Therefore, given an ellipsoid with center $\boldsymbol{p}_c$, vector of semi-axes lengths $\boldsymbol{s}$, and rotation matrix R_e, and the sampling probability ρ, the ellipsoid can be transformed into a Gaussian distribution according to

$$\begin{array}{rcl} \boldsymbol{\mu} & = & \boldsymbol{p}_c \\ \Sigma & = & \frac{1}{\chi_3^2(1-\rho)} R_e^{-1} \Lambda\left(\boldsymbol{s}\right)^2 R_e \end{array}. \tag{185}$$

4.2 Problem Formulation

The system's state is denoted by

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{v} \end{bmatrix}, \tag{186}$$

where $\boldsymbol{p} \in \boldsymbol{\mathcal{X}}_p \subset \mathbb{R}^3$ is the position and $\boldsymbol{v} \in \boldsymbol{\mathcal{X}}_v \subset \mathbb{R}^3$ is the velocity, and $\boldsymbol{\mathcal{X}}_p$ and $\boldsymbol{\mathcal{X}}_v$ are bounded closed sets. The system's state space is defined as the Cartesian product of the position and velocity spaces, or

equivalently

$$\boldsymbol{\mathcal{X}} = \boldsymbol{\mathcal{X}}_p \times \boldsymbol{\mathcal{X}}_v. \tag{187}$$

There is a space of obstacles to be avoided, which is denoted by $\boldsymbol{\mathcal{X}}_{\texttt{obs}}$ and is assumed to be a bounded open set. The obstacle-free space can then be defined as

$$\boldsymbol{\mathcal{X}}_{\texttt{free}} = \boldsymbol{\mathcal{X}} \setminus \boldsymbol{\mathcal{X}}_{\texttt{obs}}. \tag{188}$$

The position subspace of $\boldsymbol{\mathcal{X}}_{\texttt{free}}$ is denoted by $\boldsymbol{\mathcal{X}}_{p_{\texttt{free}}}$ and the velocity subspace is denoted by $\boldsymbol{\mathcal{X}}_{v_{\texttt{free}}}$. Given an initial state $\boldsymbol{x}_0 \in \boldsymbol{\mathcal{X}}_{\texttt{free}}$ with position $\boldsymbol{p}_0 \in \boldsymbol{\mathcal{X}}_{p_{\texttt{free}}}$, a final state $\boldsymbol{x}_f \in \boldsymbol{\mathcal{X}}_{\texttt{free}}$ with position $\boldsymbol{p}_f \in \boldsymbol{\mathcal{X}}_{p_{\texttt{free}}}$, the scaling parameters $\zeta_y, \zeta_z \in \mathbb{R}_{>0}$, the ratio $\Upsilon \in (0, 1]$, and the probability $\rho \in (0, 1)$ from (184), the problem is to design a Kinodynamic RRT* sampling function that applies a Gaussian distribution in $\boldsymbol{\mathcal{X}}_p$ such that:

1. the mean of the distribution is at the midpoint between $\boldsymbol{p}_0$ and $\boldsymbol{p}_f$,
2. the x-axis of the ellipsoid that describes the distribution is aligned with the vector $\boldsymbol{p}_f - \boldsymbol{p}_0$,
3. the ratio of the semi-axis length of the ellipsoid that describes the distribution along its y-axis to that along its x-axis is ζ_y, and similarly the ratio of the semi-axis length along its z-axis to that along its x-axis is ζ_z,
4. the ratio of the volume of the ellipsoid to the volume of $\boldsymbol{\mathcal{X}}_p$ is Υ,
5. the ratio of the number of randomly generated samples that are in the closed convex hull of the ellipsoid to those are that not is ρ.

4.3 Problem Solution

The problem presented in Section 4.2 is solved by defining an ellipsoid in $\mathbb{R}^3$ that meets the design criteria 1 – 4, where an example is carried through this section to demonstrate the geometric manipulation of the ellipsoidal region. Then, the ellipsoid is converted to a Gaussian distribution according to (185) to address the criteria 5. Finally, an algorithmic sampling function is presented to use the Gaussian distribution to generate the position vector of the state and a uniform distribution to generate the velocity vector of the state such that the randomly generated state lies in $\boldsymbol{\chi}_{\texttt{free}}$.

For the example carried through this section let

$$\begin{aligned} \boldsymbol{p}_0 &= \begin{bmatrix} -40 & -40 & 40 \end{bmatrix}^T \\ \boldsymbol{p}_f &= \begin{bmatrix} 40 & 40 & 80 \end{bmatrix}^T \\ \zeta_y &= 0.5 \\ \zeta_z &= 0.3 \\ \gamma &= 0.1 \end{aligned} \tag{189}$$

The center of the ellipsoid is taken as the midpoint between $\boldsymbol{p}_0$ and $\boldsymbol{p}_f$, which is obtained according to

$$\boldsymbol{p}_c = \frac{1}{2}\left(\boldsymbol{p}_0 + \boldsymbol{p}_f\right). \tag{190}$$

Next, the rotation matrix must be computed such that the x-axis of the ellipsoid aligns with the x-axis of the inertial frame. From the third design criteria it can be seen that the x-axis of the ellipsoid should be aligned with the vector

$$\boldsymbol{p}_{f0} = \boldsymbol{p}_f - \boldsymbol{p}_0. \tag{191}$$

Figure 35 shows $\boldsymbol{p}_0$ as the green diamond, $\boldsymbol{p}_f$ as the red diamond, $\boldsymbol{p}_c$ as the purple point, $\boldsymbol{p}_{f0}$ as the black vector, and the axes of the inertial frame as the blue vectors for the parameters in (189), where the unit basis vectors of the inertial frame are described by

$$\boldsymbol{e}_x = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{e}_y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \boldsymbol{e}_z = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{192}$$

The reference frame X'Y'Z is then defined as a translation of the XYZ frame to the center of the ellipsoid. This can be seen for the example of this section in Figure 36. Note that, since the transformation from the XYZ frame to the X'Y'Z' frame is simply a translation, the unit basis vectors of the X'Y'Z' frame are the same as those in (192). The next task is to defined the rotation matrix R_e that aligns the vector $\boldsymbol{p}_{f0}$ with the X'-axis. However, the approach taken here is to determine the rotation matrix that aligns the X'-axis with the vector $\boldsymbol{p}_{f0}$, at which point the inverse of this rotation matrix, R_e, can easily be obtained. This can be accomplished in two steps using Euler angles [74]. First, the the X'Y'Z' frame is rotated around the Z'-axis to define a new X"Y"Z" frame. Then the X"Y"Z" frame is rotated around the Y" axis to align the X'-axis with the vector $\boldsymbol{p}_{f0}$.

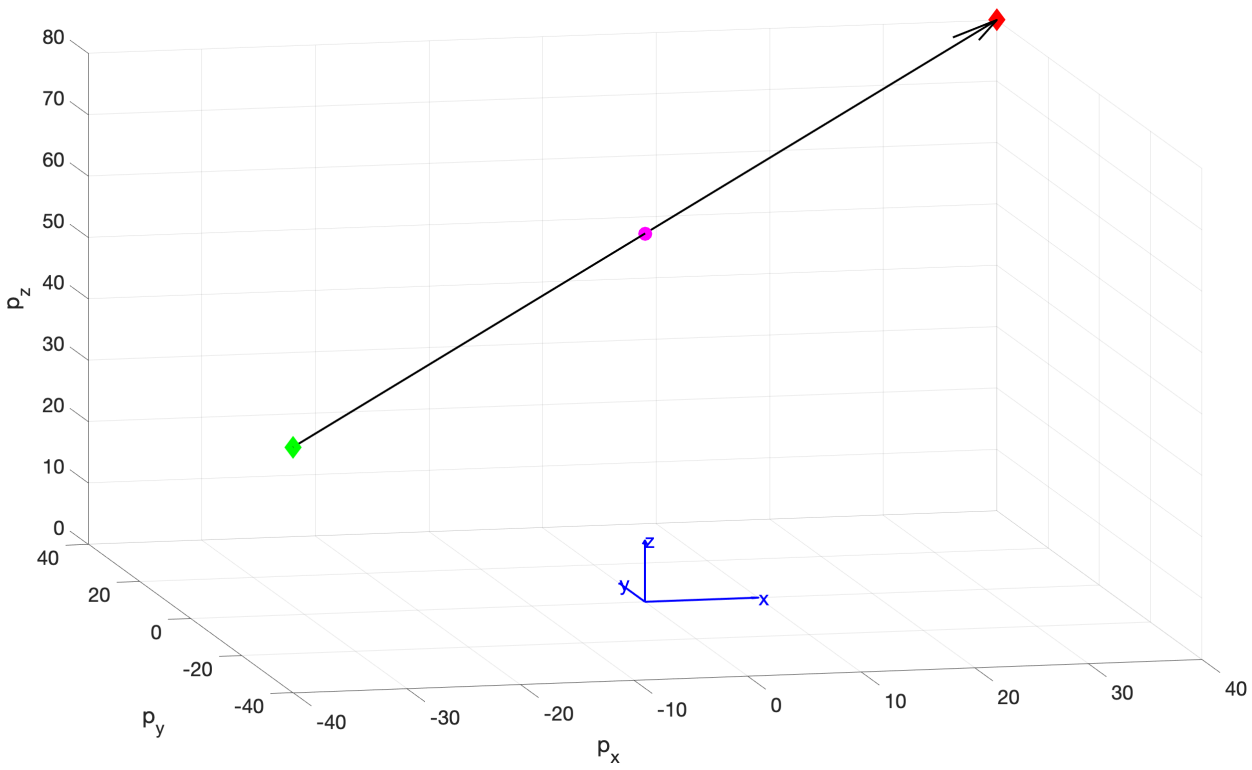


Figure 35: Ellipsoid rotation matrix: XYZ frame.

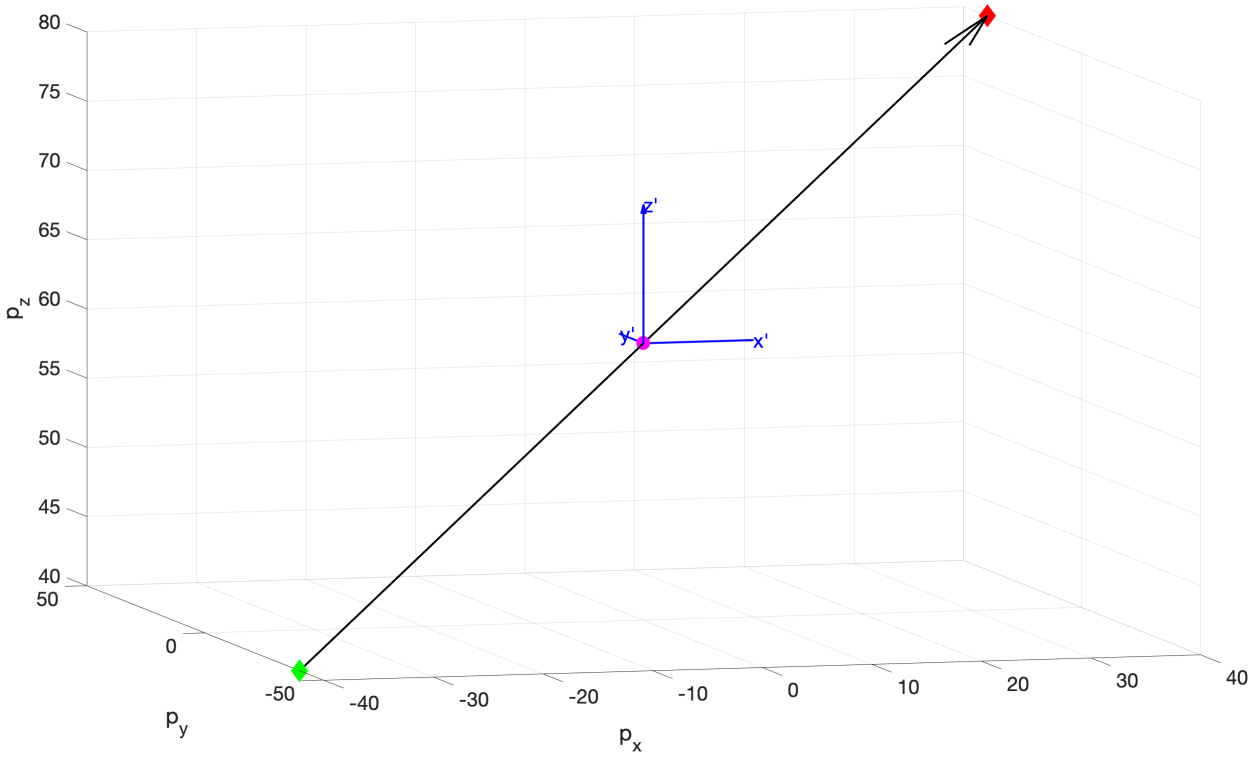


Figure 36: Ellipsoid rotation matrix: X'Y'Z' frame.

The rotation matrix that describes the rotation around the Z'-axis can be obtained by computing the angle from the X'-axis to the projection of $\boldsymbol{p}_{f0}$ onto the X'Y' plane, which has the unit normal vector $\boldsymbol{e}_z$. The projection of $\boldsymbol{p}_{f0}$ onto the X'Y' plane is describe by

$$P_{\boldsymbol{e}_z}(\boldsymbol{p}_{f0}) = \boldsymbol{p}_{f0} - (\boldsymbol{p}_{f0} \cdot \boldsymbol{e}_z)\, \boldsymbol{e}_z. \tag{193}$$

The rotation angle can then be obtained as

$$\psi = \texttt{atan2}\left(P_{\boldsymbol{e}_z}(\boldsymbol{p}_{f0}) \cdot \boldsymbol{e}_y, P_{\boldsymbol{e}_z}(\boldsymbol{p}_{f0}) \cdot \boldsymbol{e}_x\right), \tag{194}$$

where $\texttt{atan2}(y, x)$ is the function found in many programming languages that produces the angle in radians corresponding to the arctangent of $\frac{y}{x}$ while also considering the signs of both x and y. The rotation matrix around the Z'-axis is

$$R_{z'} = \begin{bmatrix} c(\psi) & -s(\psi) & 0 \\ s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{195}$$

where $c(\cdot)$ and $s(\cdot)$ denote the cosine and sine functions, respectively. The X"Y"Z" frame is then described as a rotation of the X'Y'Z' according to the rotation matrix R_z. As a result, the standard basis vectors of the X"Y"Z" frame are

$$\boldsymbol{e}_{x''} = R_{z'} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \quad \boldsymbol{e}_{y''} = R_{z'} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \boldsymbol{e}_{z''} = R_{z'} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{196}$$

The projection of the vector $\boldsymbol{p}_{f0}$ onto the X'Y' plane for the example of this section is

$$P_{\boldsymbol{e}_z}(\boldsymbol{p}_{f0}) = \begin{bmatrix} 80 & 80 & 0 \end{bmatrix}^T \tag{197}$$

which is shown in Figure 37. The rotation angle around the Z'-axis is then evaluated as $\psi = \frac{\pi}{4}$ radians, which yields the rotation matrix

$$R_{z'} = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{198}$$

The resulting X"Y"Z" frame is shown in Figure 38.

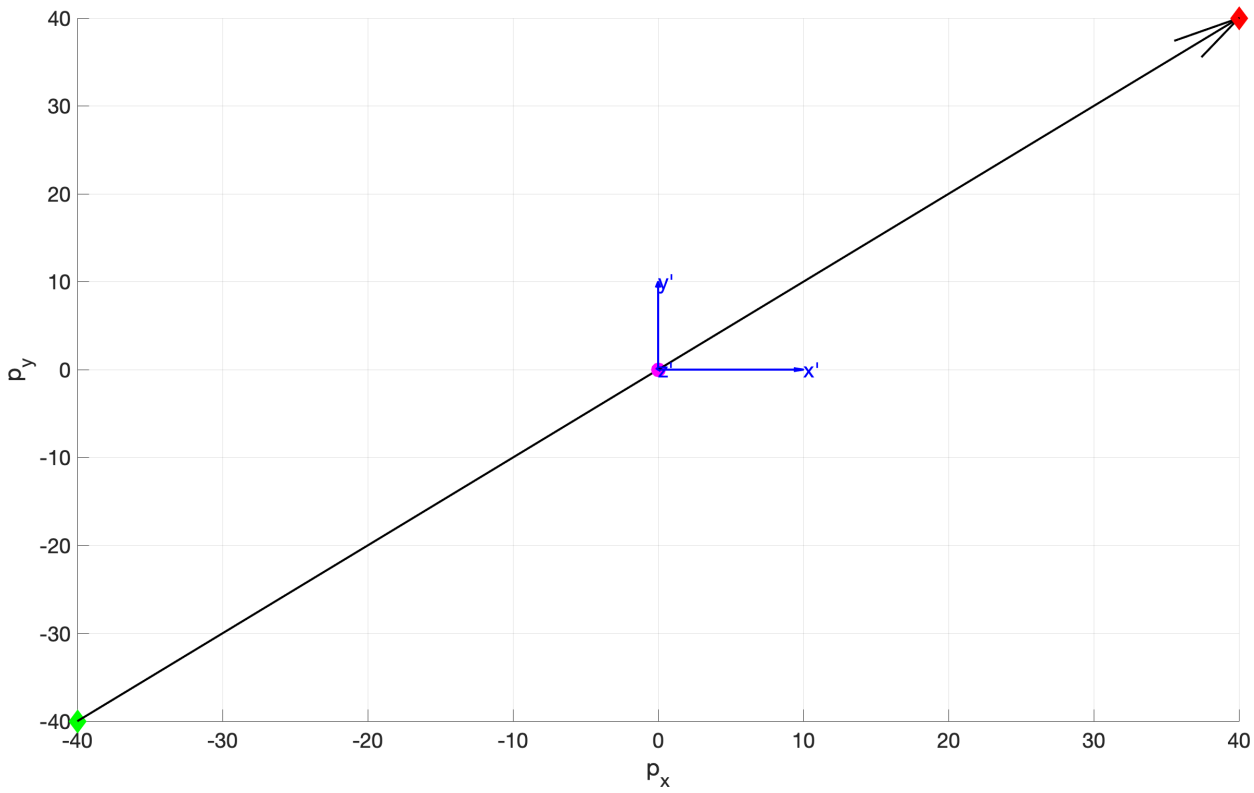


Figure 37: Ellipsoid rotation matrix: projection of $\boldsymbol{p}_{f0}$ onto the X'Y' plane.

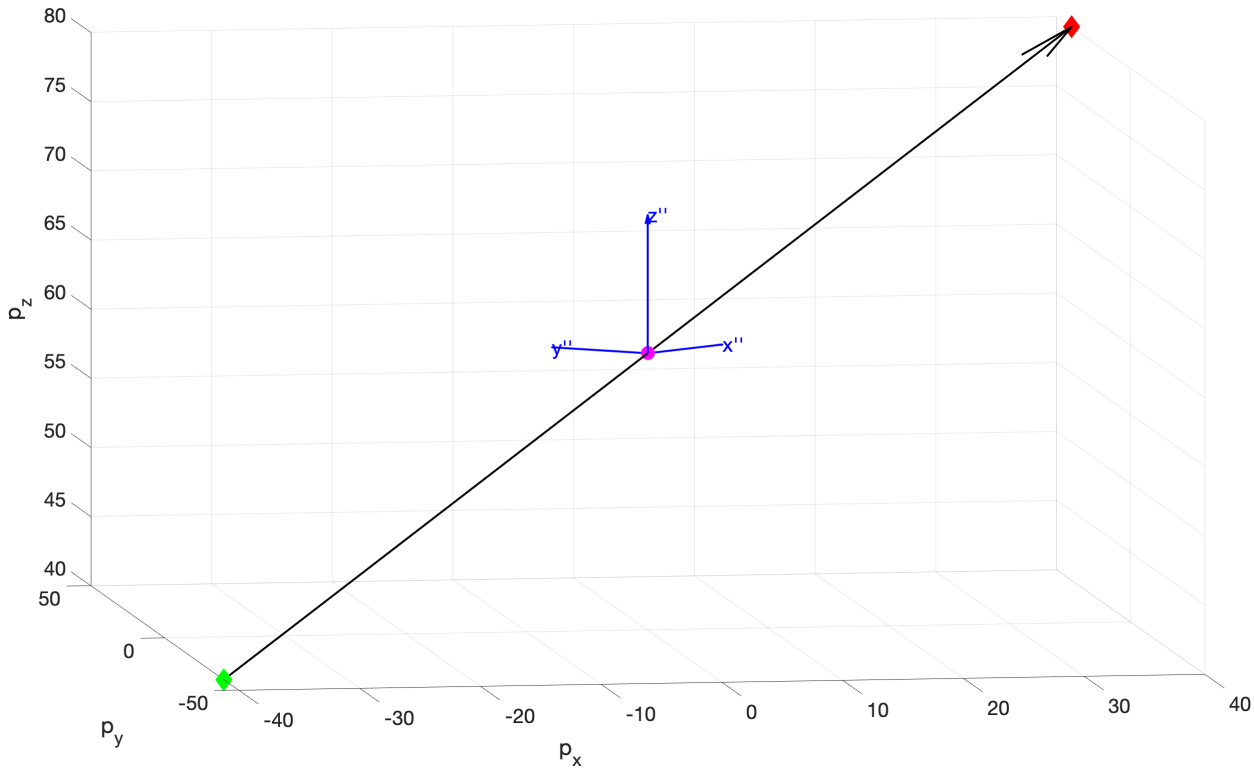


Figure 38: Ellipsoid rotation matrix: X"Y"Z" frame.

The next step is to rotate the X"Y"Z" frame around Y" to obtain the ellipsoidal reference frame such that the rotation of the X"-axis aligns with the vector $\boldsymbol{p_{f0}}$. This is accomplished using a similar approach to that which was used to obtain $R_{z'}$. The projection of $\boldsymbol{p}_{f0}$ onto the X"Z" plane, which has unit normal vector $\boldsymbol{e}_{y''}$ is

$$P_{\boldsymbol{e}_{y''}}\left(\boldsymbol{p}_{f0}\right) = \boldsymbol{p}_{f0} - \left(\boldsymbol{p}_{f0} \cdot \boldsymbol{e}_{y''}\right) \boldsymbol{e}_{y''}. \tag{199}$$

The rotation angle can then be obtained as

$$\phi = \texttt{atan2}\left(P_{\boldsymbol{e}_{y''}}\left(\boldsymbol{p}_{f0}\right) \cdot \boldsymbol{e}_{z''}, P_{\boldsymbol{e}_{y''}}\left(\boldsymbol{p}_{f0}\right) \cdot \boldsymbol{e}_{x''}\right) \tag{200}$$

and the rotation matrix around the Y"-axis is

$$R_{y''} = \begin{bmatrix} c\left(\phi\right) & 0 & -s\left(\phi\right) \\ 0 & 1 & 0 \\ s\left(\phi\right) & 0 & c\left(\phi\right) \end{bmatrix}. \tag{201}$$

The projection of the vector $\boldsymbol{p}_{f0}$ onto the X"Z" plane for the example of this section is

$$P_{\boldsymbol{e}_{y''}}\left(\boldsymbol{p}_{f0}\right) = \boldsymbol{p}_{f0}, \tag{202}$$

which yields the rotation angle $\phi \approx \frac{716\pi}{6619}$ and the rotation matrix

$$R_{y''} \approx \begin{bmatrix} \frac{544}{577} & 0 & -\frac{1}{3} \\ 0 & 1 & 0 \\ \frac{1}{3} & 0 & \frac{544}{577} \end{bmatrix}. \tag{203}$$

As a result, the rotation matrix describing the rotation from the X'Y'Z' frame to the X"'Y"'Z"' frame is described by

$$R_{X'Y'Z'}^{X'''Z'''Y'''} = R_{z'} R_{y''}, \tag{204}$$

where the X"'Y"'Z"' for the example is shown in Figure 39.

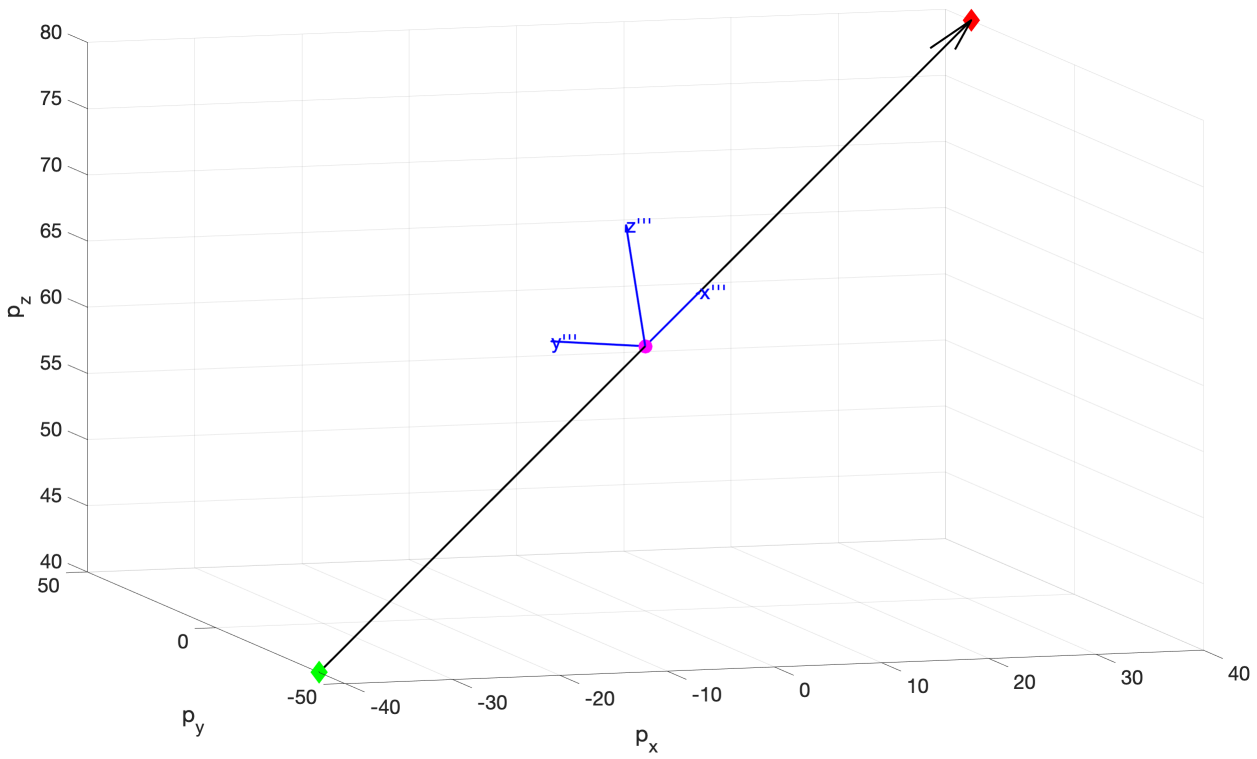


Figure 39: Ellipsoid rotation matrix: X"'Y"'Z"' frame.

Theorem 11. *Given the rotation matrix $R_{z'}$ in* (195) *with the rotation angle given by* (194)*, and given the rotation matrix $R_{y''}$ in* (201) *with the rotation angle given by* (200)*, the rotation matrix described by*

$$R = R_{z'} R_{y''} \tag{205}$$

aligns the standard basis vector $\boldsymbol{e}_x$ given by (192) *with the vector $\boldsymbol{p}_{f0}$.*

Proof. Let the vector $\boldsymbol{p}_{f0}$ be described by

$$\boldsymbol{p}_{f0} = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}^T. \tag{206}$$

If the rotation described by $R_{z'} R_{y''}$ aligns $\boldsymbol{e}_x$ with $\boldsymbol{p}_{f0}$ then

$$R_{z'} R_{y''} \boldsymbol{e}_x \left\| \boldsymbol{p}_{f0} \right\| = \boldsymbol{p}_{f0}. \tag{207}$$

It can be seen from (195) and (201) that

$$R_{z'}R_{y''}\boldsymbol{e}_x \left\|\boldsymbol{p}_{f0}\right\| = \begin{bmatrix} c\left(\psi\right)c\left(\phi\right) \\ s\left(\psi\right)c\left(\phi\right) \\ s\left(\phi\right) \end{bmatrix} \left\|\boldsymbol{p}_{f0}\right\|. \tag{208}$$

Recall the following relationship

$$\begin{array}{rcl} \mathtt{cos}\left(\mathtt{atan}\left(\alpha\right)\right) & = & \frac{1}{\sqrt{1+\alpha^2}} \\ \mathtt{sin}\left(\mathtt{atan}\left(\alpha\right)\right) & = & \frac{\alpha}{\sqrt{1+\alpha^2}} \end{array}. \tag{209}$$

It can be seen from (194) that

$$\psi = \mathtt{atan}\left(\frac{p_y}{p_x}\right), \tag{210}$$

which can be used with the relationships in (209) to yield

$$\begin{array}{rcl} c\left(\psi\right) & = & \frac{p_x}{\sqrt{p_x^2+p_y^2}} \\ s\left(\psi\right) & = & \frac{p_y}{\sqrt{p_x^2+p_y^2}} \end{array}. \tag{211}$$

It can also be seen that (199) reduces to

$$P_{\boldsymbol{e}_{y''}}\left(\boldsymbol{p}_{f0}\right) = \boldsymbol{p}_{f0}0\left(\boldsymbol{p}_{f0}\cdot R_{z'}\boldsymbol{e}_y\right)R_{z'}\boldsymbol{e}_f = \begin{bmatrix} p_x c^2\left(\psi\right) + p_y s\left(\psi\right)c\left(\psi\right) \\ p_x s\left(\psi\right)c\left(\psi\right) + p_y s^2\left(\psi\right) \\ p_z \end{bmatrix}. \tag{212}$$

This result can then be used with (200) to yield

$$\phi = \mathtt{atan}\left(\frac{P_{\boldsymbol{e}_{y''}}\left(\boldsymbol{p}_{f0}\right)\cdot R_{z'}\boldsymbol{e}_z}{P_{\boldsymbol{e}_{y''}}\left(\boldsymbol{p}_{f0}\right)\cdot R_{z'}\boldsymbol{e}_x}\right) = \mathtt{atan}\left(\frac{p_z}{p_x c\left(\psi\right)+p_y s\left(\psi\right)}\right), \tag{213}$$

which can then be used with (209) to obtain the result

$$\begin{array}{rcccl} c\left(\phi\right) & = & \frac{p_x c(\psi)+p_y s(\psi)}{\sqrt{(p_x c(\psi)+p_y s(\psi))^2+p_z^2}} & = & \frac{\sqrt{p_x^2+p_y^2}}{\left\|\boldsymbol{p}_{f0}\right\|} \\ s\left(\phi\right) & = & \frac{p_z}{\sqrt{(p_x c(\psi)+p_y s(\psi))^2+p_z^2}} & = & \frac{p_z}{\left\|\boldsymbol{p}_{f0}\right\|} \end{array}. \tag{214}$$

Applying the results of (214) and (211) to (208) yields the condition in (207) and proves that a rotation described by $R_{z'}R_{y''}$ aligns $\boldsymbol{e}_x$ with $\boldsymbol{p}_{f0}$. □

It can be seen from Theorem 11 that the rotation matrix described by $R_{z'}R_{y''}$ is sufficient to align $\boldsymbol{e}_x$ with $\boldsymbol{p}_{f0}$. However, the rotation matrix that is required is that which aligns $\boldsymbol{p}_{f0}$ with $\boldsymbol{e}_x$. Therefore, the rotation matrix of the ellipsoid is

$$R_e = R_{y''}^T R_{z'}^T. \tag{215}$$

Now that the location and the orientation of the ellipsoidal region has been established to meet the design criteria 1 and 2, the ellipsoidal region needs to be properly sized and then transformed into the Gaussian distribution. The volume of the ellipsoid is described by [75]

$$V_e = \frac{4}{3}\pi \prod_{i=\{x,y,z\}} s_i, \tag{216}$$

where, according to the third design criteria,

$$s_y = \zeta_y s_x, \quad s_z = \zeta_z s_x \ . \tag{217}$$

Note that the Lebesgue measure of a bounded closed three dimensional Euclidean space is equivalent to its volume. Therefore, the volume of $\boldsymbol{\mathcal{X}}_p$ is denoted by $\mathcal{L}(\boldsymbol{\mathcal{X}}_p)$. According to the fourth design criteria, the ratio of the volume of the ellipsoid to the volume of $\boldsymbol{\mathcal{X}}_p$ is Υ. The semi-axis length s_x can therefore be evaluated as

$$s_x = \left(\frac{3\Upsilon\mathcal{L}(\boldsymbol{\mathcal{X}}_p)}{4\pi\zeta_y\zeta_z}\right)^{\frac{1}{3}} \tag{218}$$

Using the result of (218) with the relationships in (217), the rotation matrix in (215), and the ellipsoid center position in (190), the Gaussian distribution may be obtained according to (185) for a given $\rho \in (0,1)$, where

$$\boldsymbol{s} = \begin{bmatrix} s_x & s_y & s_z \end{bmatrix}^T. \tag{219}$$

Figure 41 shows the resulting ellipsoidal region for the example discussed in this section, where

$$\boldsymbol{s} = \begin{bmatrix} \left(\frac{2\times10^6}{\pi}\right)^{\frac{1}{3}} & \left(\frac{2.5\times10^5}{\pi}\right)^{\frac{1}{3}} & \left(\frac{5.4\times10^4}{\pi}\right)^{\frac{1}{3}} \end{bmatrix}^T. \tag{220}$$

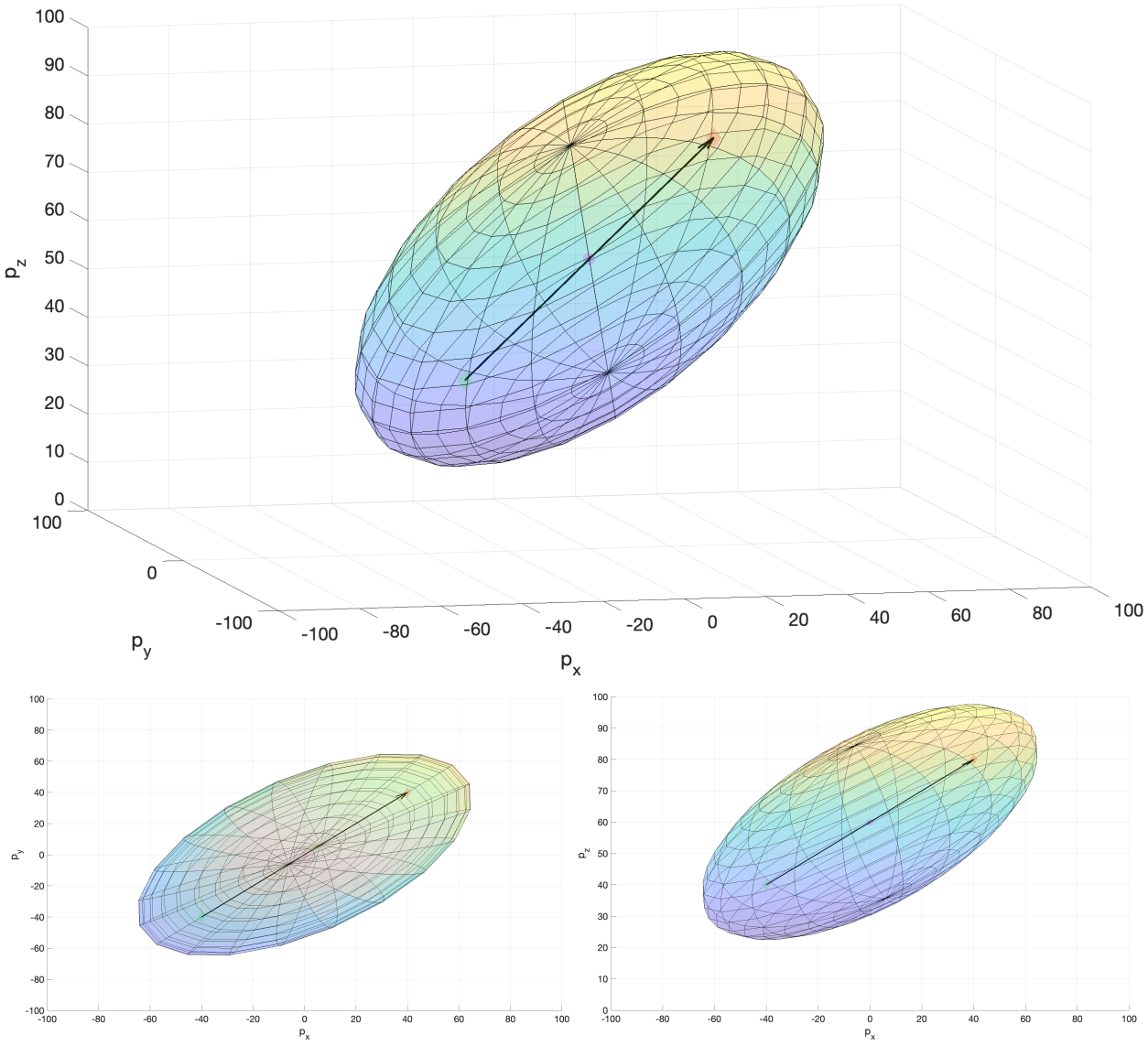


Figure 40: Gaussian distribution directed sampling ellipsoidal region.

Observe that the sampling function in Definition 2 of Section 2.2.2 produces random state space samples $\boldsymbol{x}_{\texttt{rand}} \in \boldsymbol{\mathcal{X}}_{\texttt{free}}$. There is no guarantee that random states obtained according to the Gaussian distribution in (185) will lie in $\boldsymbol{\mathcal{X}}_{p_{\texttt{free}}}$. Therefore, a verification is required to ensure that the position vectors of the randomly generated samples are members of $\boldsymbol{\mathcal{X}}_{p_{\texttt{free}}}$. Additionally, the proposed Gaussian distribution does not generate the velocity vectors of the state. Therefore, a normal distribution is applied to $\boldsymbol{\mathcal{X}}_v$ to generate the velocity vectors of the state. Additionally, a verification is used to validate that the randomly generated states, which consists of a position vector obtained from the Gaussian distribution on $\boldsymbol{\mathcal{X}}_p$ and a velocity vector obtained from the uniform distribution on $.\boldsymbol{\mathcal{X}}_v$, lies in $\boldsymbol{\mathcal{X}}_{p_{\texttt{free}}}$. The proposed sampler function is given in Algorithm 14 on page 101, where the random state component generators are defined in Definition 7.

Definition 7. *The following functions are used to generate random vectors in Algorithm 14.*

- *Uniform State Sampler: Given a bounded closed space $\boldsymbol{\mathcal{X}}$, a random element of $\boldsymbol{\mathcal{X}}$ is obtained according to a uniform distribution across $\boldsymbol{\mathcal{X}}$ from the function*

$$\boldsymbol{x} \leftarrow Uniform\left(\boldsymbol{\mathcal{X}}\right)$$

such that $\boldsymbol{x} \in \boldsymbol{\mathcal{X}}$.

- *Gaussian State Sampler: Given a mean $\boldsymbol{\mu}$ and covariance matrix Σ, a random state is generated according to a Gaussian distribution described by $\boldsymbol{\mu}$ and Σ from the function*

$$\boldsymbol{x} \leftarrow Gaussian\left(\boldsymbol{\mu}, \Sigma\right).$$

Algorithm 14 Directed Sampler Function

```
function DirectedSample(μ, Σ)
    p ← Gaussian(μ, Σ);
    v ← Uniform(X_v);
    x = [p^T, v^T]^T;
    while x ∉ X_free do
        p ← Gaussian(μ, Σ);
        v ← Uniform(X_v);
        x = [p^T, v^T]^T;
    return x;
```

4.4 Directed Random Sampling For Optimal Trajectory Planning of an Input Constrained Double Integrator with Drift

This section studies the directed sampling technique developed in Section 4.3 applied to an optimal trajectory planning problem of an input constrained double integrator with drift. The problem used in this section is equivalent to that addressed in Section 3.3.1. The system's state is described by the vector

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{v} \end{bmatrix}, \tag{221}$$

where $\boldsymbol{p} = [p_x, p_y, p_z]^T \in \mathbb{R}^3$ is the system's position in meters, and $\boldsymbol{v} = [v_x, v_y, v_z]^T \in \mathbb{R}^3$ is its velocity in meters per second. The system's control input is $\boldsymbol{u} = [u_x, u_y, u_z]^T \in \mathbb{R}^3$, which is its acceleration in meters

per second squared. The system's state space is constrained to

$$\boldsymbol{\mathcal{X}} = \left\{ \boldsymbol{x} \in \mathbb{R}^6 : \begin{array}{c} -100 \leq p_x, p_y \leq 100,\ 0 \leq p_z \leq 100, \\ \|\boldsymbol{v}\| \leq 20 \end{array} \right\} \tag{222}$$

and the obstacle space is described by

$$\boldsymbol{\mathcal{X}}_{\texttt{obs}} = \bigcup_{i=1}^{i=4} \bigcup_{j=1}^{j=4} \boldsymbol{\mathcal{X}}_{\texttt{obs}_{ij}}, \tag{223}$$

where

$$\boldsymbol{\mathcal{X}}_{\texttt{obs}_{ij}} = \left\{ \boldsymbol{x} \in \mathbb{R}^6 : \left\| \begin{bmatrix} p_x + 100 - 40i \\ p_y + 100 - 40j \end{bmatrix} \right\| < 10 \right\}. \tag{224}$$

The obstacle free space $\boldsymbol{\mathcal{X}}_{\texttt{free}}$ is obtained according to (188). The control input of the system is assumed to be constrained in magnitude such that

$$\boldsymbol{\mathcal{U}} = \left\{ \boldsymbol{u} \in \mathbb{R}^3 : \|\boldsymbol{u}\| < 20 \right\}, \tag{225}$$

where $\|\cdot\|$ denotes the 2-norm. The state transition equation of the system is described by

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} + \boldsymbol{c}, \quad A = \begin{bmatrix} 0_3 & I_3 \\ 0_3 & 0_3 \end{bmatrix}, \quad B = \begin{bmatrix} 0_3 \\ I_3 \end{bmatrix}, \tag{226}$$

where $\boldsymbol{c} = [\boldsymbol{c}_v^T, \boldsymbol{c}_a^T]^T$, $\boldsymbol{c}_v, \boldsymbol{c}_a \in \mathbb{R}^3$ are the velocity and acceleration components of the drift term, and where 0_k and I_k denote a $k \times k$ zero and identity matrix for $k \in \mathbb{N}_{>0}$, respectively. Given a control input trajectory $\boldsymbol{u}(t)$, a state trajectory $\boldsymbol{x}(t)$, and a final time t_f, the cost of the trajectory is evaluated according to

$$C\left(\boldsymbol{x}(t), \boldsymbol{u}(t), t_f\right) = \int_0^{t_f} \left(\frac{1}{2} \boldsymbol{u}\left(\tau\right)^T R \boldsymbol{u}\left(\tau\right) + C_I \right) d\tau, \tag{227}$$

where $R = I_3$ and $C_I = 10$. The initial and final states of the system are

$$\begin{aligned} \boldsymbol{x}_0 &= \begin{bmatrix} -40 & -40 & 40 & 0 & 0 & 0 \end{bmatrix}^T \\ \boldsymbol{x}_f &= \begin{bmatrix} 40 & 40 & 80 & 0 & 0 & 0 \end{bmatrix}^T \end{aligned}, \tag{228}$$

and the affine term is set to

$$\boldsymbol{c} = \begin{bmatrix} \boldsymbol{c}_v^T & 0 & 0 & -g \end{bmatrix}^T \quad \boldsymbol{c}_v = \begin{bmatrix} 3 & -2 & 0.5 \end{bmatrix}^T \quad , \tag{229}$$

where $g = 9.8\ \frac{m}{s^2}$ is the acceleration due to gravity and the vector $\boldsymbol{c}_v$ describes a constant wind velocity in meters per second. The optimal control problem is defined as

$$\begin{aligned} & \min_{\boldsymbol{u}, t_f} \int_0^{t_f} \left(\tfrac{1}{2} \boldsymbol{u}(\tau)^T R \boldsymbol{u}(\tau) + C_I \right) d\tau \\ \texttt{s.t.} \quad & \dot{\boldsymbol{x}}(t) = A\boldsymbol{x}(t) + B\boldsymbol{u}(t) + \boldsymbol{c} \\ & \boldsymbol{x} : [0, t_f] \rightarrow \boldsymbol{\mathcal{X}}_{\texttt{free}} \\ & \boldsymbol{u} : [0, t_f] \rightarrow \boldsymbol{\mathcal{U}} \\ & \boldsymbol{\psi}(\boldsymbol{x}_0, \boldsymbol{x}_f) = \boldsymbol{0} \end{aligned} \quad , \tag{230}$$

where the boundary conditions are described by

$$\boldsymbol{\psi}(\boldsymbol{x}_0, \boldsymbol{x}_f) = \begin{bmatrix} \boldsymbol{x}(0) - \boldsymbol{x}_0 \\ \boldsymbol{x}(t_f) - \boldsymbol{x}_f \end{bmatrix}. \tag{231}$$

All of the Kinodynamic RRT* functions used to approximate the solution of (230) are discussed in Section 2.3.2 and Section 3.3. The maximum number of iterations was selected as $N_{\texttt{iter}} = 2000$, the cost-to-go upper bound as $\eta = 300$, and the reachability region tuning parameter as $\gamma = 1000$, where the parameter γ is discussed in Section 2.2.2.1 on page 24. Furthermore, for the Gaussian distributed sampling technique proposed in this chapter the parameters ζ_y and ζ_z were selected as 0.5 and 0.3, respectively, and the parameters Υ and ρ were selected as 0.1 and 0.75, respectively.

An example of 3000 position vectors generated according to the proposed state sampler given by Algorithm 14 on page 101 is shown in Figure 41, where different perspectives are provided to show the ellipsoidal region created by the random samples. In this figure the initial position is shown as the green diamond, the final position is shown as the red diamond, the red cylinders describe the obstacles, and the black points describe randomly generated positions obtained from the Gaussian distributed sampling function. Furthermore, the red rectangles represent the side view of the rows of cylindrical obstacles. A comparison of the proposed Gaussian distributed sampling technique is made with a uniformly distributed sampling technique, the goal-directed technique of [49], and Informed-RRT* from [56] through the repetition of 100 simulations for each sampling method using the modified Kinodynamic RRT* trajectory planner in Chapter 3. The reason why the problem in (230) is solved using the modified Kinodynamic RRT* algorithm 100 times for

each sampling method is to minimize the effects of randomness in the algorithm by analyzing the average performance of each sampling method. Also, Algorithm 13 on page 75 was modified to collect information that is important for the comparison of each sampling method. The data collection pseudocode is shown in Algorithm 15 and the data collection version of the modified Kinodynamic RRT* is shown in Algorithm 16. Note that the functions `tic` and `toc`$(\cdot)$ are MATLAB functions that are used to measure time by comparing the CPU time when `tic` was called to that when `toc`$(\cdot)$ is called. Once the code has terminated all of the data relating to iterations, computation time, and cost of $\boldsymbol{x}_f$ can be retrieved from the array `Data` in Algorithm 15.

Algorithm 15 RRT* Data Collection

function $\mathcal{T}$.DataCollect(τ, i)
  $c_{\texttt{max}} = \min\{\eta, \gamma(\frac{\log(|\boldsymbol{V}|)}{|\boldsymbol{V}|})^{\frac{1}{n}}\}$;
  $\boldsymbol{V}_{\texttt{max}} \leftarrow \texttt{Nearby}(\boldsymbol{V}, \boldsymbol{x}_f, c_{\texttt{max}})$;
  $c_f \leftarrow \mathcal{T}.\texttt{Cost}(\boldsymbol{x}_f)$;
  for each $\boldsymbol{v} \in \boldsymbol{V}_{\texttt{max}}$ **do**
    $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{Steer}(\boldsymbol{v}, \boldsymbol{x}_f)$;
    if $\texttt{AugCollisionFree}(\boldsymbol{u}^*(t), \boldsymbol{x}^*(t), t_f^*)$ **then**
      $e \leftarrow (\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*)$;
      $c_e \leftarrow \texttt{Cost}(e)$;
      if $\mathcal{T}.\texttt{Cost}(\boldsymbol{v}) + c_e < c_f$ **then**
        $c_f \leftarrow \mathcal{T}.\texttt{Cost}(\boldsymbol{v}) + c_e$;
  if $c_f < \mathcal{T}.\texttt{Cost}(\boldsymbol{x}_f)$ **then**
    $\texttt{TempData} \leftarrow [i, \texttt{toc}(\tau), c_f]^T$;
    if $i = 1$ **then**
      $\texttt{Data} \leftarrow \texttt{TempData}$;
    else
      $\texttt{Data} \leftarrow [\texttt{Data}, \texttt{TempData}]$;
  return

Algorithm 16 Modified RRT* algorithm with data collection

function RRT*($\boldsymbol{x}_0$, $\boldsymbol{x}_f$, $N_{\texttt{iter}}$, η, γ)
  $\tau \leftarrow \texttt{tic}$;
  $\boldsymbol{V} \leftarrow \{\boldsymbol{x}_0\}$;
  $\boldsymbol{E} \leftarrow \varnothing$;
  for $i = 1$ to $N_{\texttt{iter}}$ **do**
    $\boldsymbol{x}_{\texttt{rand}} \leftarrow \texttt{Sample}()$;
    $\boldsymbol{v}_{\texttt{min}} \leftarrow \texttt{Nearest}(\boldsymbol{V}, \boldsymbol{x}_{\texttt{rand}})$;
    $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{Steer}(\boldsymbol{v}_{\texttt{min}}, \boldsymbol{x}_{\texttt{rand}})$;
    $\tilde{\boldsymbol{x}}^*(t), \tilde{\boldsymbol{u}}^*(t), \tilde{t}_f^* \leftarrow \texttt{AugLocalTraj}(\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*, \eta)$;
    $\boldsymbol{x}_{\texttt{new}} \leftarrow \tilde{\boldsymbol{x}}^*(\tilde{t}_f^*)$;
    if $\texttt{AugCollisionFree}(\tilde{\boldsymbol{u}}^*(t), \tilde{\boldsymbol{x}}^*(t), \tilde{t}_f^*)$ **then**
      $e_{\texttt{new}} \leftarrow (\tilde{\boldsymbol{x}}^*(t), \tilde{\boldsymbol{u}}^*(t), \tilde{t}_f^*)$;
      $\boldsymbol{V} \leftarrow \boldsymbol{V} \cup \{\boldsymbol{x}_{\texttt{new}}\}$;
      $\boldsymbol{E} \leftarrow \boldsymbol{E} \cup \{e_{\texttt{new}}\}$;
      $c_{\texttt{max}} = \min\{\eta, \gamma(\frac{\log(|\boldsymbol{V}|)}{|\boldsymbol{V}|})^{\frac{1}{n}}\}$;
      $\boldsymbol{V}_{\texttt{max}} \leftarrow \texttt{Nearby}(\boldsymbol{V} \setminus \{\boldsymbol{x}_{\texttt{new}}\}, \boldsymbol{x}_{\texttt{new}}, c_{\texttt{max}})$;
      for each $\boldsymbol{v} \in \boldsymbol{V}_{\texttt{max}}$ **do**
        $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{Steer}(\boldsymbol{v}, \boldsymbol{x}_{\texttt{new}})$;
        if $\texttt{AugCollisionFree}(\boldsymbol{u}^*(t), \boldsymbol{x}^*(t), t_f^*)$ **then**
          $e_{\texttt{temp}} \leftarrow (\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*)$;
          $c_e \leftarrow \texttt{Cost}(e_{\texttt{temp}})$;
          if $\mathcal{T}.\texttt{Cost}(\boldsymbol{v}) + c_e < \mathcal{T}.\texttt{Cost}(\boldsymbol{x}_{\texttt{new}})$ **then**
            $\boldsymbol{E} \leftarrow \boldsymbol{E} \cup \{e_{\texttt{temp}}\} \setminus \{e_{\texttt{new}}\}$;
            $e_{\texttt{new}} \leftarrow e_{\texttt{temp}}$;
      for each $\boldsymbol{v} \in \boldsymbol{V}_{\texttt{max}}$ **do**
        $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{Steer}(\boldsymbol{x}_{\texttt{new}}, \boldsymbol{v})$;
        if $\texttt{AugCollisionFree}(\boldsymbol{u}^*(t), \boldsymbol{x}^*(t), t_f^*)$ **then**
          $e_{\texttt{temp}} \leftarrow (\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*)$;
          $c_e \leftarrow \texttt{Cost}(e_{\texttt{temp}})$;
          if $\mathcal{T}.\texttt{Cost}(\boldsymbol{x}_{\texttt{new}}) + c_e < \mathcal{T}.\texttt{Cost}(\boldsymbol{v})$ **then**
            $\boldsymbol{e}_{\texttt{old}} \leftarrow \mathcal{T}.\texttt{GetInEdge}(\boldsymbol{v})$;
            $\boldsymbol{E} \leftarrow \boldsymbol{E} \cup \{e_{\texttt{temp}}\} \setminus \{e_{\texttt{old}}\}$;
      $\mathcal{T}.\texttt{DataCollect}(\tau, i)$;
  $c_{\texttt{max}} = \min\{\eta, \gamma(\frac{\log(|\boldsymbol{V}|)}{|\boldsymbol{V}|})^{\frac{1}{n}}\}$;
  $\boldsymbol{V}_{\texttt{max}} \leftarrow \texttt{Nearby}(\boldsymbol{V}, \boldsymbol{x}_f, c_{\texttt{max}})$;
  $c_f \leftarrow \mathcal{T}.\texttt{Cost}(\boldsymbol{x}_f)$;
  for each $\boldsymbol{v} \in \boldsymbol{V}_{\texttt{nearby}}$ **do**
    $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{Steer}(\boldsymbol{v}, \boldsymbol{x}_f)$;
    if $\texttt{AugCollisionFree}(\boldsymbol{u}^*(t), \boldsymbol{x}^*(t), t_f^*)$ **then**
      $e_{\texttt{temp}} \leftarrow (\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*)$;
      $c_e \leftarrow \texttt{Cost}(e_{\texttt{temp}})$;
      if $\mathcal{T}.\texttt{Cost}(\boldsymbol{v}) + c_e < c_f$ **then**
        $c_f \leftarrow \mathcal{T}.\texttt{Cost}(\boldsymbol{v}) + c_e$;
        $e_{\texttt{new}} \leftarrow e_{\texttt{temp}}$;
  if $c_f \neq \infty$ **then**
    $\boldsymbol{V} \leftarrow \boldsymbol{V} \cup \{\boldsymbol{x}_f\}$;
    $\boldsymbol{E} \leftarrow \boldsymbol{E} \cup \{e_{\texttt{new}}\}$;
    $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^* \leftarrow \texttt{GetSol}(\mathcal{T}, \boldsymbol{x}_0, \boldsymbol{x}_f)$;
    return $\boldsymbol{x}^*(t), \boldsymbol{u}^*(t), t_f^*$
  return *fail*;

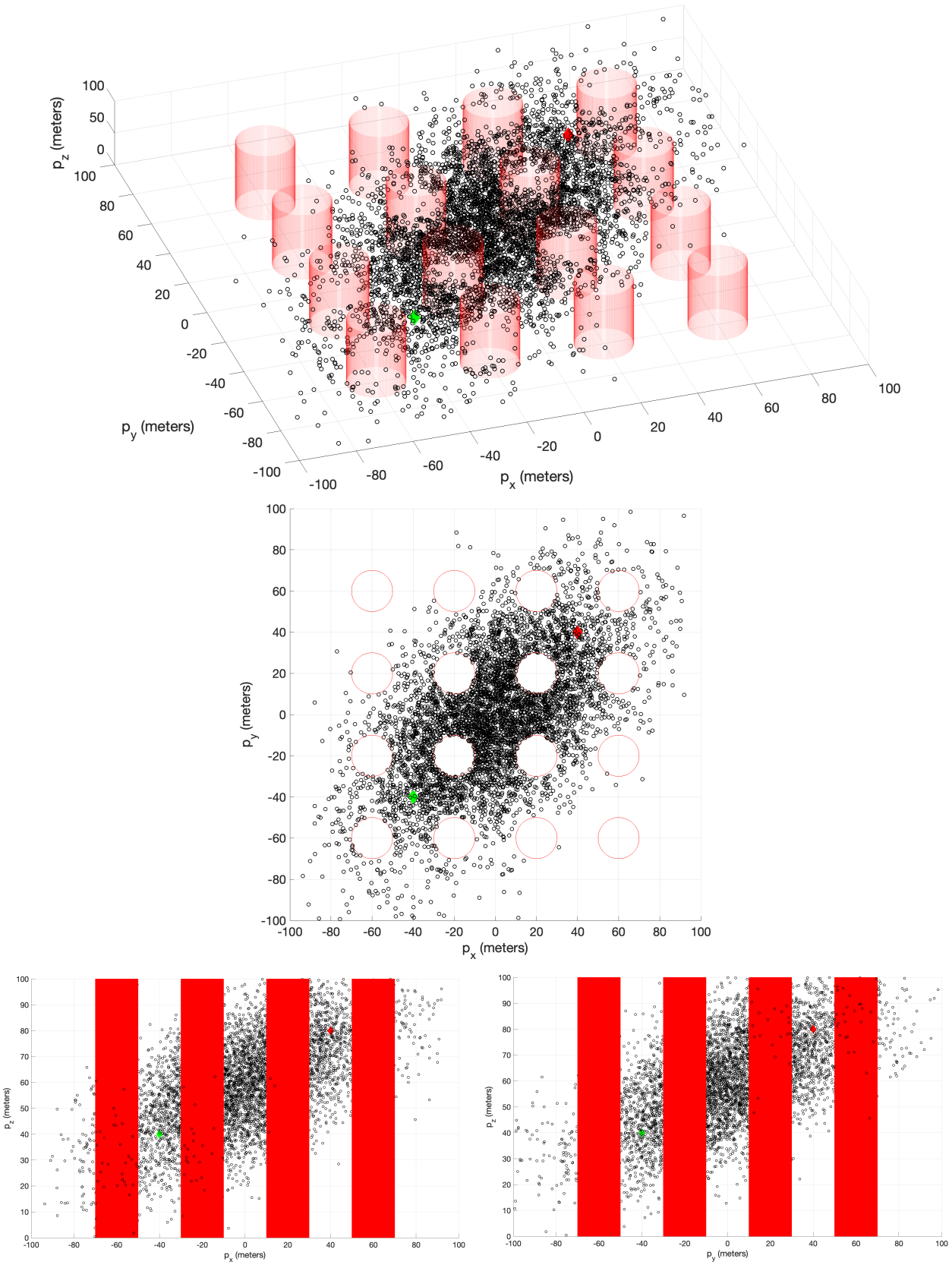
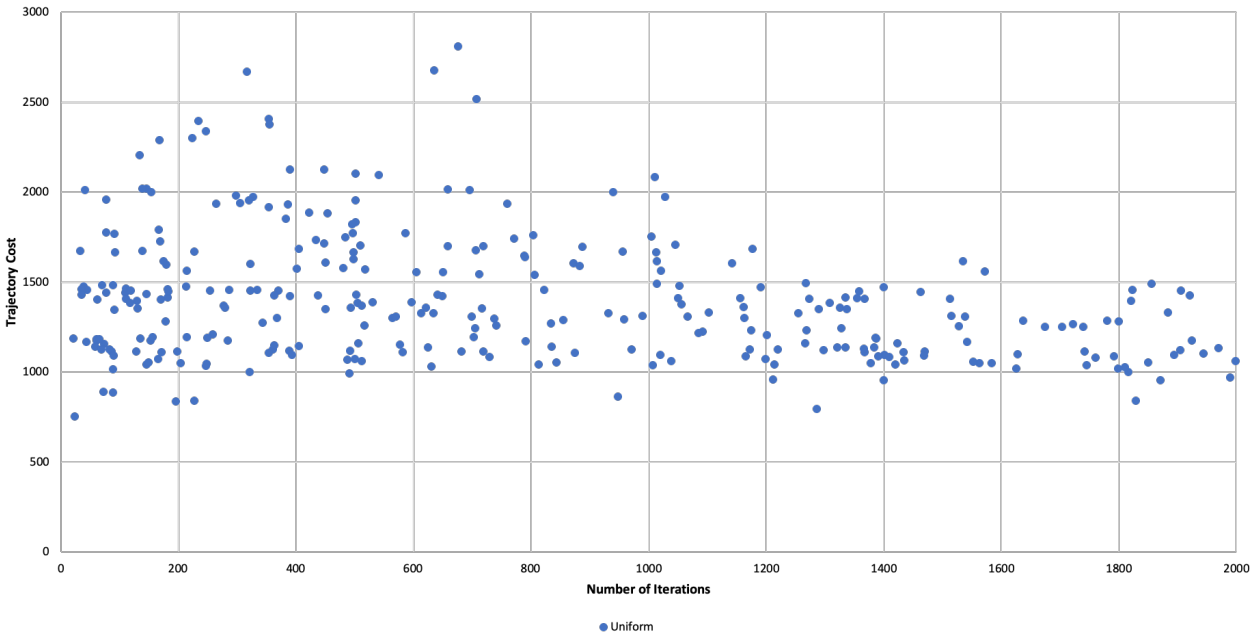


Figure 41: Gaussian distribution directed sampling position vectors.

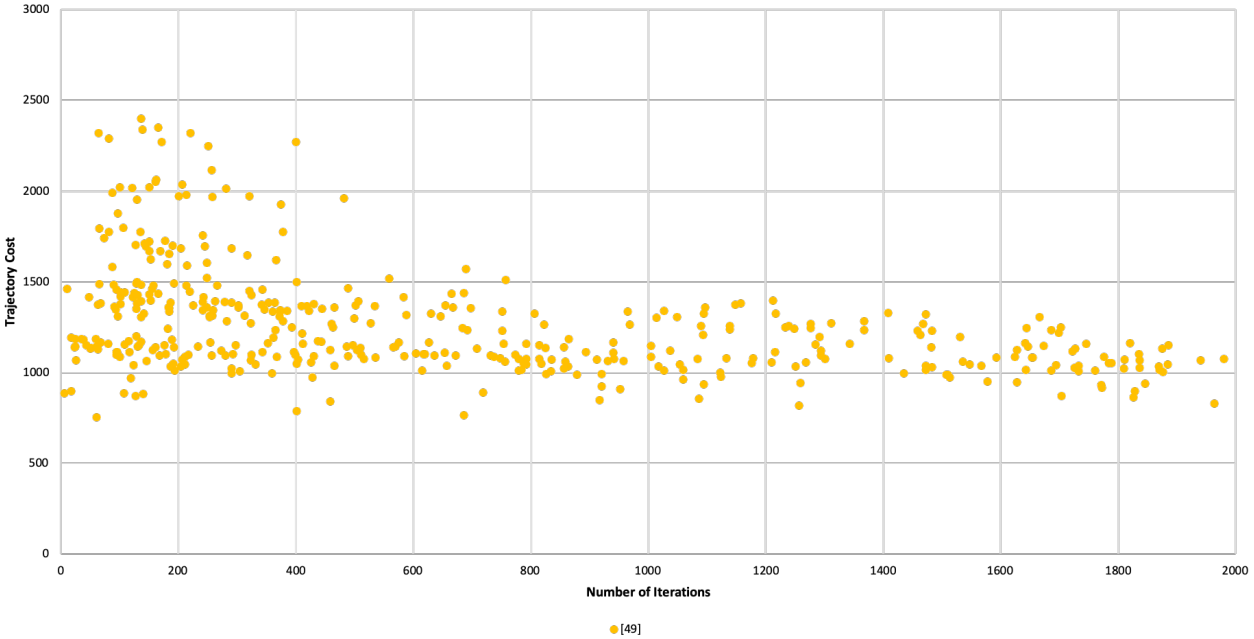
Figure 42 shows the cost of the approximate solution for each simulation versus the number of iterations completed. The results of the uniformly distributed sampling method are shown in Figure 42a, the results of the sampling method from [49] are shown in Figure 42b, the results of Informed-RRT* [56] are shown in Figure 42c, and the results of the proposed Gaussian distributed sampling technique are shown in Figure 42d. These plots allow to study the behaviour of the solver for each sampling technique in terms of the cost of the approximate solution obtained versus the number of iterations completed. Note that, since RRT* is an iterative solver, multiple approximate solutions are typically obtained for each simulation. However, as the number of iterations increases the approximate solution is improved and its cost decreases. Therefore a study is performed on characteristics of the first approximate solution and its trend toward lower cost approximations. Table 4 presents the mean and the standard deviation of the number of iterations for each sampling method to obtain an initial approximate solution of (230). Additionally, the mean and the standard deviation of the cost of the solution is provided, which are in $\frac{m^2}{s^3}$. This is important because it shows the average number of iterations required for each method to obtain an approximate solution and a comparative measure in terms of the cost of the solutions. It can be seen from Table 4 that the uniformly distributed and the Informed-RRT* [56] sampling techniques demonstrated the weakest performance for the first approximate solutions when compared with the other sampling methods. This is because the mean and the standard deviation of the number of iterations required to obtain an initial approximate solution is the largest, as well as the mean and the standard deviation of the cost of the solutions. Note that these two results are similar due to the use of the uniformly distributed sampling approach in Informed-RRT* until an initial approximate solution is obtained. The goal-direct sampling technique of [49] showed a significant improvement in terms of the number of iterations required to obtain an initial approximate solution, and a minor improvement of the cost of the approximate solutions. The proposed Gaussian distributed sampling technique demonstrated the best performance of all methods in terms of the average number of iterations required to obtain an initial approximate solution of (230) as well as in terms of the cost of the first approximate solution.

	First Approximate Solution Result For 100 Simulations							
	Uniformly Distributed		[49]		[56]		Gaussian Distributed	
	Iterations	Cost $\left(\frac{m^2}{s^3}\right)$	Iterations	Cost $\left(\frac{m^2}{s^3}\right)$	Iterations	Cost $\left(\frac{m^2}{s^3}\right)$	Iterations	Cost $\left(\frac{m^2}{s^3}\right)$
Mean	198.5	1571.4	126.8	1461.5	211.5	1573.8	50.0	1258.8
Std. Dev.	157.2	443.3	75.8	403.6	144.4	415.1	37.8	220.2

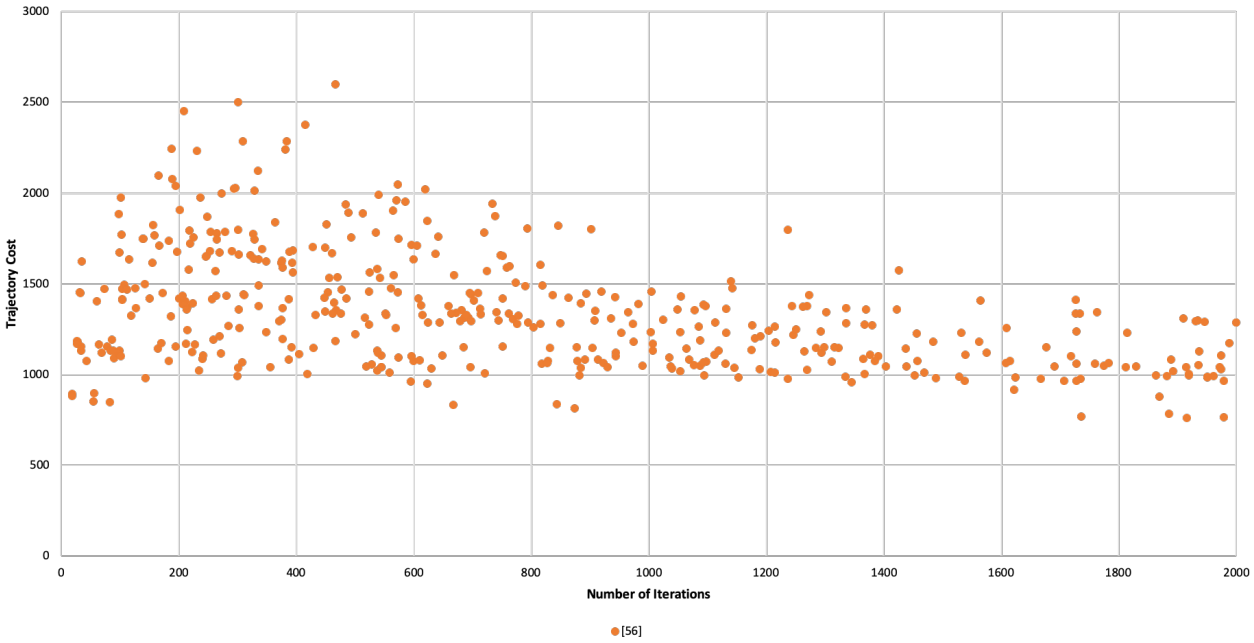
Table 2: Input constrained double integrator with drift: first approximate solution for iterations count.



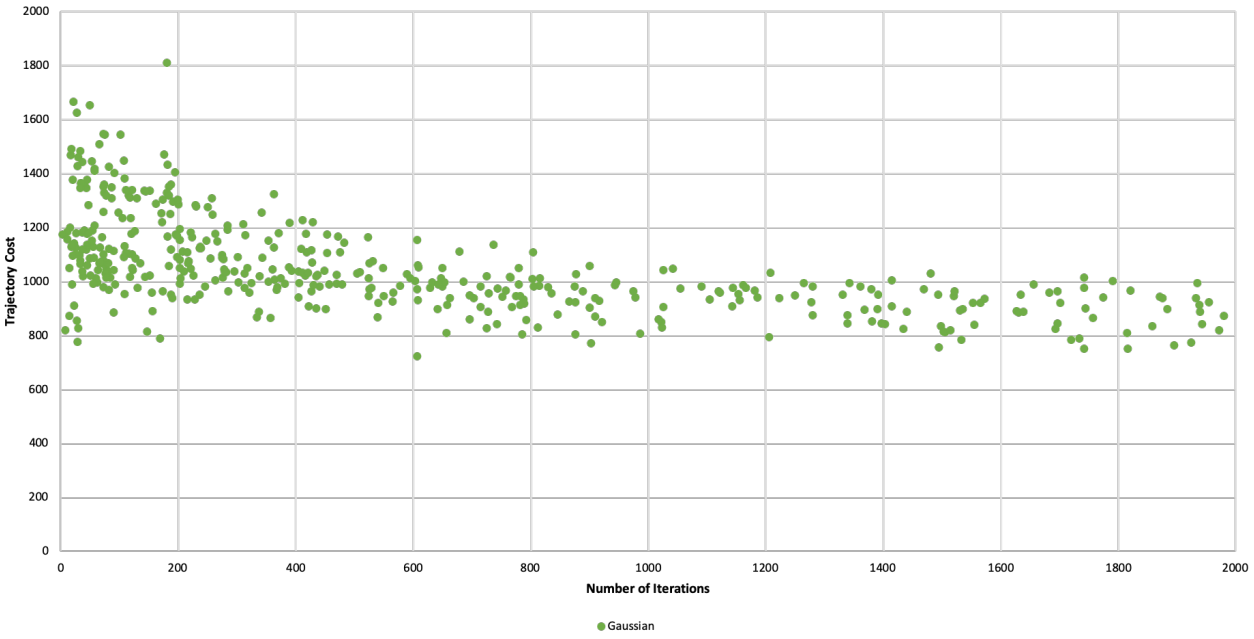
(a) Uniformly distributed sampling trajectory costs v.s. number of iterations.



(b) Goal-directed sampling [49] trajectory costs v.s. number of iterations.



(c) Informed-RRT* [56] trajectory costs v.s. number of iterations.



(d) Gaussian distributed sampling trajectory costs v.s. number of iterations.

Figure 42: Input constrained double integrator with drift: cost versus number of iterations.

Another important characteristic worth discussing is the decrease of the cost of the approximate solution as the number of iterations increases. Additionally, it is important to study the change in the success percentage as the number of iterations increases, where the success percentage for a given number of iterations is defined as the percentage of simulations that have obtained an approximate solution for the given number of iterations. These results are shown in Figure 43 where the results of the uniformly distributed sampling method are shown in blue, the results of the sampling method proposed in [49] are shown in yellow, the results of Informed-RRT* [56] are shown in orange, and the results of the proposed Gaussian distributed sampling technique are shown in green. The large dots show the average trajectory cost for a given number of iterations and are associated with the left axis. The curves show the change in the success percentage as the number of iterations increases and are associated with the right axis. In this figure it can be seen that the initial set of data points are outlier data that are not representative of the overall performance of each method. The reason for this is due to the low success percentage for small numbers of iterations, where a small percentage of the simulations obtained approximate solutions that were very low cost. For this reason, these data are not considered when discussing the overall performance of each sampling technique. It can be seen from Figure 43 that the uniformly distributed sampling technique provides the worst performance in terms of average cost of the approximate solutions and success rate. The majority of successful simulations obtained an initial approximate solution with a cost of $1400\frac{m^2}{s^3}$, and as the number of iterations increased the cost of the approximate solution slowly decreases toward $1100\frac{m^2}{s^3}$. Furthermore, 90% of simulations were successful at around 450 iterations. The sampling approach in [49] demonstrated an improvement when compared with the uniformly distributed sampling technique, where the initial approximate solution had a cost of around $1400\frac{m^2}{s^3}$ and decreases toward $1000\frac{m^2}{s^3}$. Additionally, the success percentage of the goal-directed technique is significantly better than that of the uniformly distributed technique. Informed-RRT* from reference [56] uses a uniformly distributed sampling technique until an initial approximation is obtained, and because of this the results up to about 600 iterations are very similar to those of the uniformly distributed sampling method. However, the technique then restricts the sampling to an ellipsoidal region and the average approximate solution cost versus number of iterations curve decreases more rapidly than that of the uniformly distributed and approaches that of the goal-directed approach. It can be seen that that Gaussian distributed sampling technique outperforms all other methods in terms of cost and success percentage. The initial approximate solution has a cost of around $1200\frac{m^2}{s^3}$ and quickly decreases toward $850\frac{m^2}{s^3}$. The average approximate solution cost following 2000 iterations for each sampling method are shown in Table 5. It can be seen from this table that the proposed Gaussian distributed sampling technique obtained approximate solutions of (230) that had an average cost that was significantly lower than the other sampling methods and also had significantly lower standard deviation than the other methods.

	Approximate Solution Cost $\left(\frac{m^2}{s^3}\right)$ **After 2000 Iterations**			
	Uniformly Distributed	[49]	[56]	Gaussian Distributed
Mean	1117.3	1030.7	1035.4	862.3
Std. Dev.	152.8	112.9	124.4	67.3

Table 3: Input constrained double integrator with drift: final approximate cost.

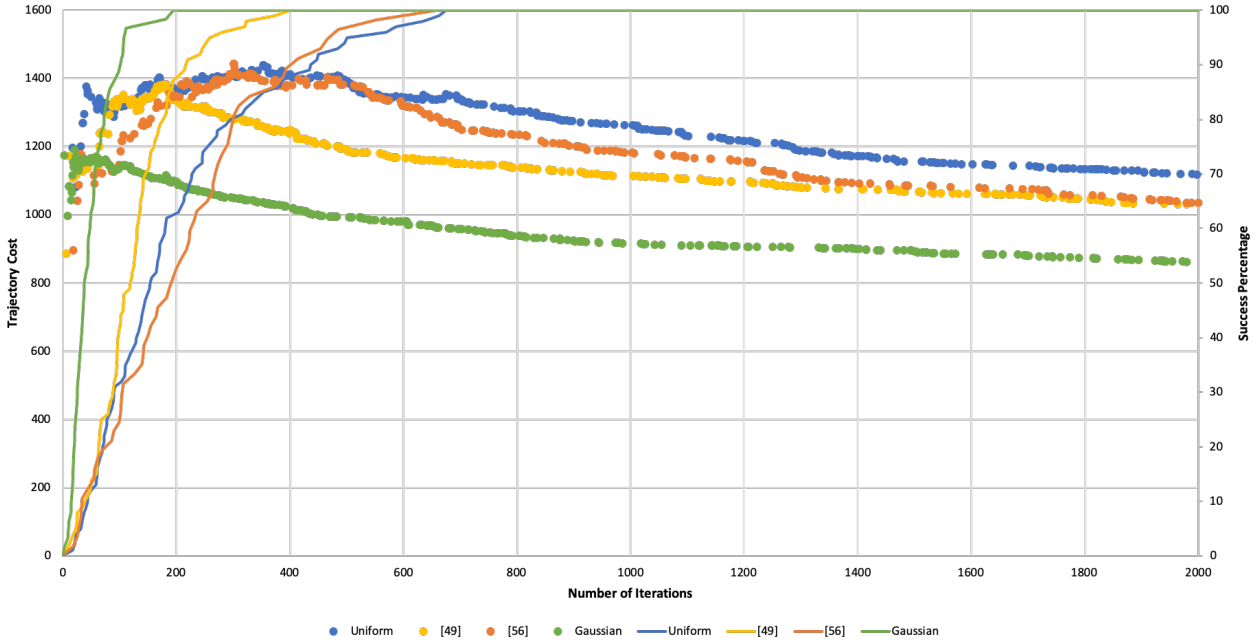
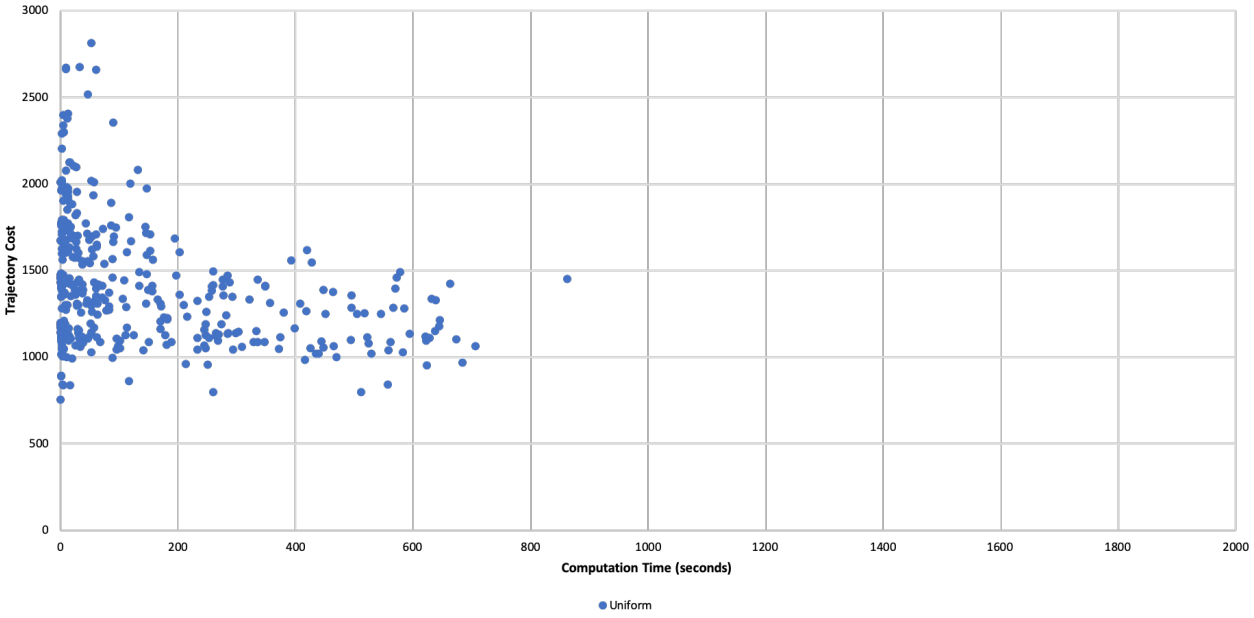


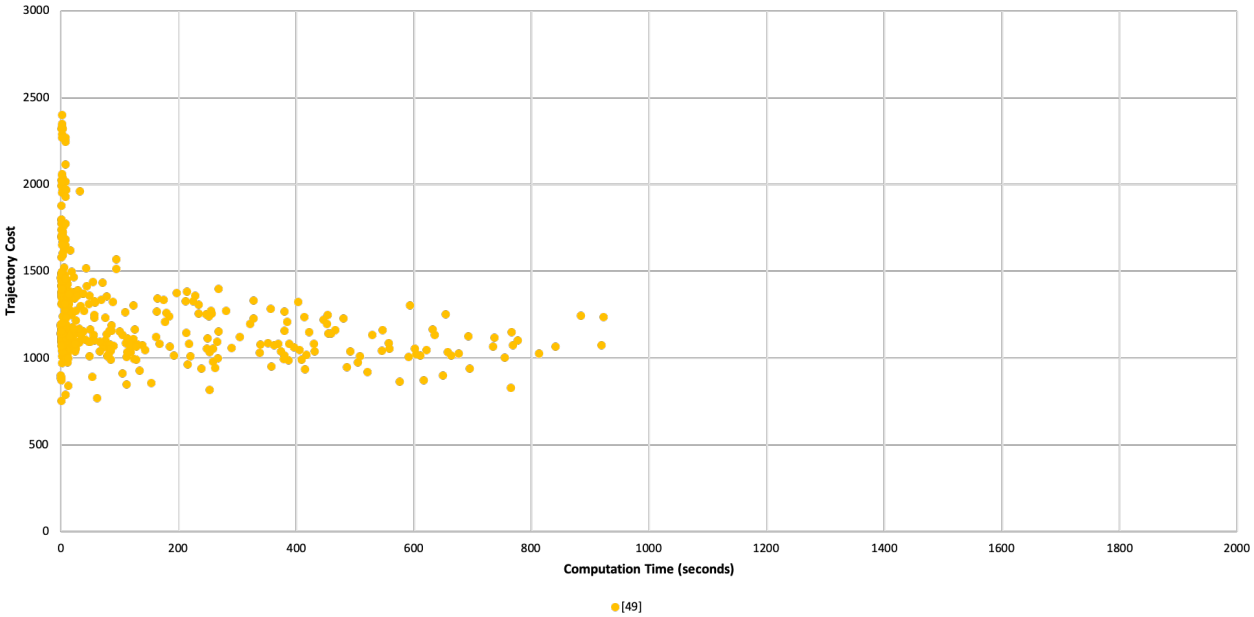
Figure 43: Input constrained double integrator: cost and success percentage versus number of iterations.

A comparison of cost and success percentage versus the number of iterations is common in many research papers that study sampling-based trajectory planners. However, the number of iterations doesn't directly translate into computation time. For example, it can be seen from Figure 41 that the proposed Gaussian distributed sampling technique causes the position vectors of the randomly generated state space samples to be very concentrated around the mean of the distribution and less so as the distance from the mean increases. As a consequence of this, when a new state is added to the tree, there may be more nearby vertices to consider when optimizing the tree, which may result in longer computation times as the number of iterations increases. Figure 44 shows the results of 100 simulations performed for each sampling method, where each point denotes the time instant and that the cost of the approximate solution was improved. Figure 44a shows the results for the uniformly distributed sampling method, Figure 44b shows the results of the sampling method proposed in [49], Figure 44c shows the results of Informed-RRT* [56], and Figure 44d shows the results of the proposed Gaussian distributed sampling method. It can be seen from Figure

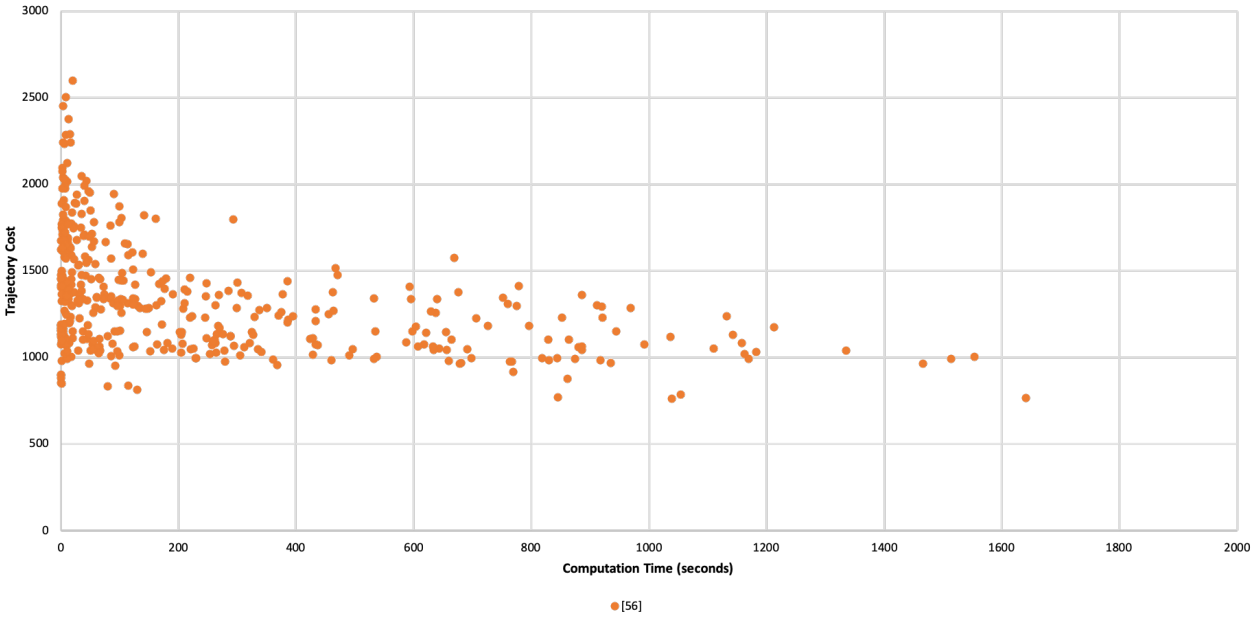
44d that some of the simulations using a Gaussian distribuetd sampling method required more computation time than those of the other methods. Additionally, this is also apparent in the results using the Informed-RRT* sampling method. This is largely due to the high concentration of samples around the mean and the optimization steps of Kinodynamic RRT*. An analysis is therefore performed to study the relationship of the approximate solution cost versus computation time for each sampling method.



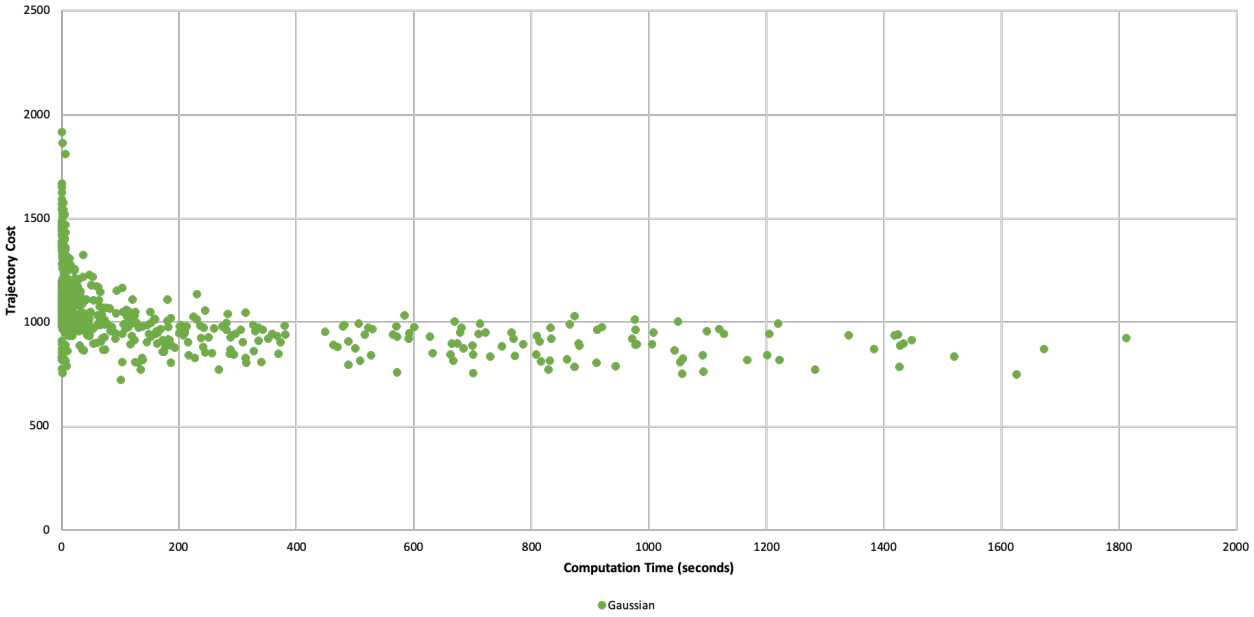
(a) Uniformly distributed sampling trajectory costs v.s. computation time.



(b) Goal-directed random sampling [49] trajectory costs v.s. computation time.



(c) Informed-RRT* [56] trajectory costs v.s. computation time.



(d) Gaussian distributed sampling trajectory costs v.s. computation time.

Figure 44: Input constrained double integrator with drift: cost versus computation time.

Figure 45 shows the change in success percentage of each sampling method versus time. Additionally, Figure 46 shows the change in the average cost of the approximate solutions of (230) obtained using each sampling method versus time. In these figures the results for the uniformly distributed sampling method are shown in blue, the results of the sampling method proposed in [49] are shown in yellow, the results of Informed-RRT* [56] are shown in orange, and the results of the proposed Gaussian distributed sampling technique are shown in green. From Figure 45 it can be seen that the uniformly distributed and the Informed-RRT* state samplers required the longest computation times to achieve 100% success, which was greater than 40 seconds. The goal-directed approach in [49] was an improvement compared to these methods and required about 8 seconds to reach 100% success. The proposed Gaussian distributed sampler outperformed all other methods, where an approximate solution of (230) was obtained by all simulations in just 5.9 seconds. Table 4 shows the mean and standard deviation of the computation time required to obtain an initial approximate solution of (230) for each sampling method as well as the mean and standard deviation of the cost of the initial solution of each method. The units of computation time are in seconds and the units of the cost are in $\frac{m^2}{s^3}$. From this table it can be seen that the uniformly distributed and the Informed-RRT* sampling methods required the longest average computation time to obtain an initial approximate solution and the standard deviation of the computation times are also the largest. Additionally, the mean and standard deviation of the cost of the solution obtained using these methods are also the largest. The goal-directed approach from [49] demonstrated a significant improvement of the computation time when compared with these methods, and a reasonable improvement on the cost of the initial approximate solution. The proposed Gaussian distributed sampling method demonstrated the best performance since it required a significantly lower computation time to find an approximate solution and the average cost of the solution was also significantly lower when compared with all other sampling methods.

	First Approximate Solution For 100 Simulations							
	Uniformly Distributed		[49]		[56]		Gaussian Distributed	
	Time	Cost $\left(\frac{m^2}{s^3}\right)$	Time	Cost $\left(\frac{m^2}{s^3}\right)$	Time	Cost $\left(\frac{m^2}{s^3}\right)$	Time	Cost $\left(\frac{m^2}{s^3}\right)$
Mean	5.8	1571.4	1.9	1461.5	5.8	1573.8	0.9	1258.8
Std. Dev.	9.4	443.3	1.9	403.6	7.5	415.1	1.2	220.2

Table 4: Input constrained double integrator with drift: first approximate solutions time results.

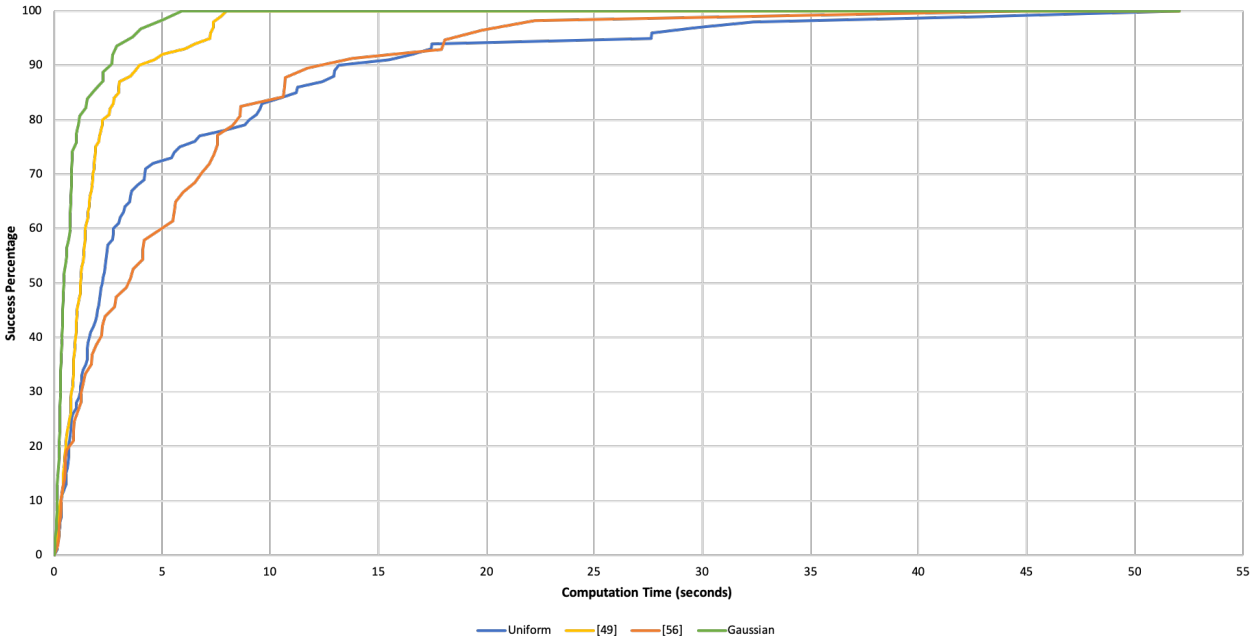


Figure 45: Input constrained double integrator with drift: success percentage versus computation time.

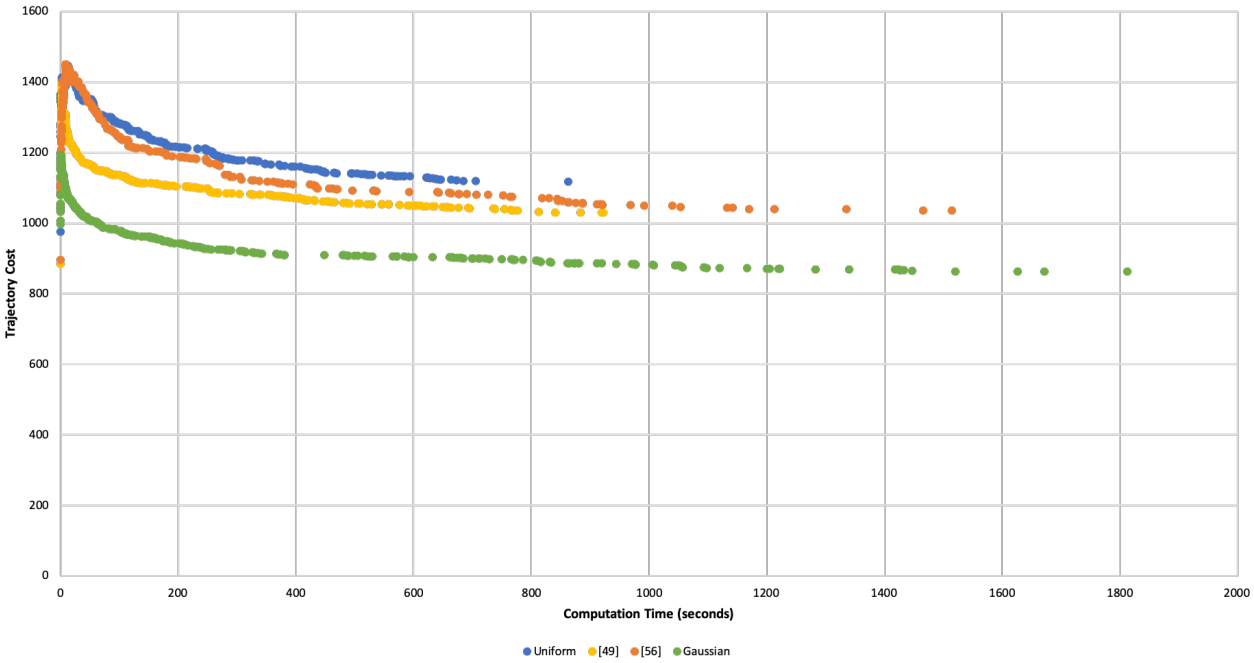


Figure 46: Input constrained double integrator with drift: cost versus computation time.

In Figure 46 it can be seen that for low computation times each sampling method obtains low cost approximate solutions, which then rapidly rise and steadily fall. However, in Figure 45 it can be seen that for small computation times all sampling methods have low success percentages. For this reason the initial set of data in Figure 46 for very small computation times are considered as outlier data and are omitted when discussing the behaviour of each sampling method. It can be seen from Figure 46 that the uniformly distributed sampling technique provides the worst performance of all sampling methods in terms of the average cost of the approximate solutions and the computation time. The goal-directed sampling technique from [49] provides an improvement of this characteristic. The Informed-RRT* method in [56] appears to perform as a hybrid of these two methods, which initially yields results similar to the uniformly distributed method and quickly converges toward those of the goal-directed method. The proposed Gaussian distributed approach however demonstrates the best approximate solution cost versus computation time when compared with all other methods, which is evident since the curve of these results are significantly lower than all other methods. Table 5 shows the mean and standard deviation of the cost of the approximate solution and the computation time after 2000 iterations. The units of the computation time is seconds and the units of the cost is $\frac{m^2}{s^3}$. It can be seen from this table that on average the uniformly distributed sampling method completed the 2000 iterations the fastest, but also had the highest approximate solution cost. The goal-directed sampling technique from [49] took longer, but provided lower cost approximations on average. The Informed-RRT* method took slightly longer than the goal-directed method and obtained approximate solutions that had an average cost that was similar to that of the goal-directed method. The proposed Gaussian distributed sampling technique takes the longest to complete 2000 iterations, however the cost is the smallest. Figures 45 and 43 demonstrate that, given a maximum computation time, the proposed Gaussian distributed sampling method yielded the lowest average approximate solution cost, followed by the goal-directed sampling method in [49], then the Informed-RRT* method in [56], and then the uniformly distributed method.

	Approximate Solution Computation Time For 100 Simulations							
	Uniformly Distributed		[49]		[56]		Gaussian Distributed	
	Time	Cost $\left(\frac{m^2}{s^3}\right)$	Time	Cost $\left(\frac{m^2}{s^3}\right)$	Time	Cost $\left(\frac{m^2}{s^3}\right)$	Time	Cost $\left(\frac{m^2}{s^3}\right)$
Mean	291.6	1117.3	338.1	1030.7	482.9	1035.4	681.8	862.3
Std. Dev.	238.7	152.8	277.5	112.9	421.4	124.4	489.2	67.3

Table 5: Input constrained double integrator with drift: first approximate solution.

In summary, for the optimal control problem described by (230), the proposed Gaussian distributed directed sampling technique found an initial approximate solution in less iterations and in less computation time than the three other sampling methods and the initial approximate solution had an average cost that was lower than the other methods as well. The proposed sampling method also had a higher success rate for a given number of iterations and a given computation time when compared with the other methods. Additionally, the curve of the approximate solution cost versus the number of iterations and versus the computation time was significantly lower for the proposed Gaussian sampling technique, which shows that, given a maximum number of iterations or a maximum computation time, the proposed sampling method produces approximate solutions that are lower cost than the three other methods. This sampling technique significantly reduces the computation time to find approximate solutions of optimal control problems. For trajectory planning of aerial vehicles in particular, this sampling technique brings sampling-based trajectory planning techniques closer to real-time trajectory planning than before. For networks that consist of waypoints, such as the long-haul fixed-wing aviation system in place today, this technique makes quasi real-time trajectory planning attainable. This is because the sequence of waypoints decomposes the optimal trajectory planning problem into sequential optimal trajectory planning subproblems. As a consequence of this, the aircraft is only required to plan the initial segment prior to take-off and can iteratively plan the optimal trajectory of each segment while en-route to the initial waypoint of that segment.

Chapter 5

Conclusions and Future Work

Inspired by real-time trajectory planning for autonomous urban aerial mobility, this thesis addressed problems of optimal trajectory planning of affine systems in non-convex state spaces with convex control input constraints. Traditionally optimal control theory has been successful in obtaining solutions of optimal control problems. However, alternative approaches have been developed for problems where optimal control theory struggles, such as problems with non-convex state spaces. One approach is Kinodynamic RRT*, which is a sampling-based trajectory planner that attempts to approximate the solution of optimal control problems with non-convex state spaces by using optimal control theory and a tree in the state space.

Although many contributions have been made to approximate solutions of optimal control problems for affine systems using Kinodynamic RRT*, the unconstrained problem solved by the steering function is typically formulated for a fixed final time. As a consequence, numerical optimization is required to compute the optimal final time for free final time trajectories. By contrast, similar problems are solved in [59] with the assumption that the final time is free. As a consequence, the optimal final time of the trajectory can be obtained as the root of a polynomial. Chapter 2 used the result of [59] with Kinodynamic RRT* to plan optimal trajectories of affine systems with unconstrained control inputs. It was shown that, for a double integrator with drift, trajectories that are computed using the method presented in [59] require on average just 2.2% the computation time when compared with other methods in the literature.

One of the most significant drawbacks of Kinodynamic RRT* is that it assumes systems with unconstrained control inputs. However, the majority of real systems are subject to input constraints and therefore no guarantee can be made that the approximate solution obtained using Kinodynamic RRT* satisfies these constraints. This was addressed in Chapter 3, where two modified RRT* functions are proposed. As a result, the modified Kinodynamic RRT* trajectory planner is capable of approximating solutions of optimal control problems in non-convex state spaces for affine systems with convex control input constraints. Simulation results are shown for a double integrator with drift that has an upper bound on the control input magnitude. When compared with the traditional Kinodynamic RRT* it is shown that the modified algorithm is capable of finding approximate solutions to the constrained optimal control problem, which could not be found by other methods.

Finally, Chapter 4 addresses the problem of convergence speed of Kinodynamic RRT*. Typically, a

uniform distribution is applied to the non-convex state space to randomly grow the tree. However, this method is inefficient since extensive computation resources are expended on computing trajectories and optimizing the tree in regions where a solution is unlikely to exist. A novel sampling approach is proposed that uses a Gaussian distribution rather than the uniform distribution, and a comparison is made with other work in the literature. The comparison is made for optimal trajectory planning of a double integrator with drift where the state space is non-convex and the control input of the system in constrained in magnitude. The results show that the proposed sampling method is capable of obtaining initial approximate solutions of lower average cost in less iterations and less computation time when compared with various methods. Additionally, given a maximum number of iterations or a maximum computation time, the proposed method obtained lower cost approximate solutions than other methods.

In summary, the main contribution of this thesis is a modified trajectory planner for affine systems with control input constraints in non-convex state spaces. A steering function was designed for a double integrator system with drift using the theory of [59], which calculates the optimal final time of a trajectory in just 2.2% the computation time of similar methods in the literature. A modified Kinodynamic RRT* was then developed to solve optimal trajectory problems of affine systems with convex control input constraints. Finally, a directed sampling approach was designed using a Gaussian distribution in the state space, which showed improvements of the cost of the approximate solution for a given number of iterations and a given computation time when compared with other methods.

5.1 Future Work

Some possible extensions of the work presented in this thesis include:

1. Artificial intelligence (AI) and machine learning has begun to find applications in sampling-based motion planners largely due to the "brute force" approach exhibited by these planners. This is apparent in Chapter 2, where it can be seen that the tree is expanded into regions of the system's state space where intuition says the optimal solution does not lie. In Chapter 4 a Gaussian distribution is obtained from an ellipsoid defined in the state space to reduce this undesirable growth. However, the tuning parameters of the ellipsoid must be input by the operator, which may not necessarily be intuitive. AI and machine learning could add value to this by computing these tuning parameters and determining the optimal sizing according to a pre-defined metric.

2. Multi-agent trajectory planning is another significant obstacle that must be overcome to make complex autonomous UAV networks a reality. Some contributions have been made toward this objective using sampling-based motion planning. However, to the best of the author's knowledge no work

has approached real-time computation. It would be an interesting extension to apply the directed sampling-based approach to multi-agent trajectory planning.

3. Some work has been performed to study the optimality of RRT*, however no proof to the best of the author's knowledge has studied the optimality of general problems solved using Kinodynamic RRT*. It would be an important extension to study the optimality of the modified Kinodynamic RRT* discussed in this thesis to determine if it provably converges to the optimal solution for input constrained affine systems.

4. It would be a valuable contribution to extend the work of the modified Kinodynamic RRT* trajectory planning to general nonlinear systems that are subject to control input constraints.

Appendix A - Roots of Double Integrator Hamiltonian

This appendix studies the existence of non-negative real roots of

$$H\left(t_f^*\right) = \frac{1}{t_f^{*4}}\left(C_I t_f^{*4} + \begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix}^T \begin{bmatrix} -18R & -6t_f^*R & 18R & -6t_f^*R & -12t_f^*R & 0_3\\ -6t_f^*R & -2t_f^{*^2}R & 6t_f^*R & -t_f^{*^2}R & -3t_f^{*^2}R & 0_3\\ 18R & 6t_f^*R & -18R & 6t_f^*R & 12t_f^*R & 0_3\\ -6t_f^*R & -t_f^{*^2}R & 6t_f^*R & -2t_f^{*^2}R & -3t_f^{*^2}R & 0_3\\ -12t_f^*R & -3t_f^{*^2}R & 12t_f^*R & -3t_f^{*^2}R & -6t_f^{*^2}R & 0_3\\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & \frac{t_f^{*4}}{2}R \end{bmatrix} \begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix}\right), \quad (232)$$

where $\boldsymbol{x}_0, \boldsymbol{x}_f, \boldsymbol{c} \in \mathbb{R}^6$, $R \in \mathbb{R}^{3\times 3}$ is assumed to be symmetric and positive definite, $C_I \in \mathbb{R}_{>0}$, and $t_f^* \in \mathbb{R}_{\geq 0}$. Observe that

$$\lim_{t_f^*\to\infty} H\left(t_f^*\right) = C_I + \boldsymbol{c}^T \begin{bmatrix} 0_3 & 0_3\\ 0_3 & \frac{1}{2}R \end{bmatrix} \boldsymbol{c} > 0 \quad (233)$$

since R is positive definite and $C_I > 0$. This result implies that there does not exist a root at $t_f^* \to \infty$. Therefore, all roots of (232) must satisfy

$$H_4 t_f^{*4} + H_2 t_f^{*^2} + H_1 t_f^* + H_0 = 0, \quad (234)$$

where

$$H_4 = C_I + \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix} 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & \frac{1}{2}R \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}$$

$$H_2 = \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix} 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & -2R & 0_3 & -R & -3R & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & -R & 0_3 & -2R & -3R & 0_3 \\ 0_3 & -3R & 0_3 & -3R & -6R & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix} . \tag{235}$$

$$H_1 = \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix} 0_3 & -6R & 0_3 & -6R & -12R & 0_3 \\ -6R & 0_3 & 6R & 0_3 & 0_3 & 0_3 \\ 0_3 & 6R & 0_3 & 6R & 12R & 0_3 \\ -6R & 0_3 & 6R & 0_3 & 0_3 & 0_3 \\ -12R & 0_3 & 12R & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}$$

$$H_0 = \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix} -18R & 0_3 & 18R & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ 18R & 0_3 & -18R & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & 0_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}$$

Definition 8. *Kronecker Product [76]: Given the matrices $A \in \mathbb{R}^{p\times q}$ and $B \in \mathbb{R}^{r\times s}$, where $p, q, r, s \in \mathbb{N}_{>0}$,*

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1q} \\ a_{21} & a_{22} & \dots & a_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ a_{p1} & a_{p2} & \dots & a_{pq} \end{bmatrix}, \tag{236}$$

the Kronecker product of A with B is defined as

$$A \otimes B = \begin{bmatrix} a_{11}B & a_{12}B & \dots & a_{1q}B \\ a_{21}B & a_{22}B & \dots & a_{2q}B \\ \vdots & \vdots & \ddots & \vdots \\ a_{p1}B & a_{p2}B & \dots & a_{pq}B \end{bmatrix}. \tag{237}$$

□

Lemma 7. *Eigenvalues of a Kronecker Product [76]: Let $\boldsymbol{\lambda}_A$ denote the set of eigenvalues of $A \in \mathbb{R}^{p\times p}$ and let $\boldsymbol{\lambda}_B$ denote that of $B \in \mathbb{R}^{r\times r}$, where $p, r \in \mathbb{N}_{>0}$. The set of eigenvalues of the matrix*

$$C = A \otimes B. \tag{238}$$

is described by the set

$$\boldsymbol{\lambda}_C = \{\lambda_A \lambda_B : \forall\ \lambda_A \in \boldsymbol{\lambda}_A, \forall\ \lambda_B \in \boldsymbol{\lambda}_B\}. \tag{239}$$

□

Theorem 12. *For the coefficients defined in* (235)*, if $R \in \mathbb{R}^{3\times 3}$ is symmetric and positive definite and $C_I \in \mathbb{R}_{>0}$, then H_4 is positive, H_2 is non-positive, and H_0 is non-positive $\forall\ \boldsymbol{x}_0, \boldsymbol{x}_f, \boldsymbol{c} \in \mathbb{R}^6$.*

Proof. Let the following substitutions be made

$$\boldsymbol{x}_0 = \begin{bmatrix} \boldsymbol{p}_0 \\ \boldsymbol{v}_0 \end{bmatrix}, \quad \boldsymbol{x}_f = \begin{bmatrix} \boldsymbol{p}_f \\ \boldsymbol{v}_f \end{bmatrix}, \quad \boldsymbol{c} = \begin{bmatrix} \boldsymbol{c}_v \\ \boldsymbol{c}_a \end{bmatrix}, \tag{240}$$

where $\boldsymbol{p}_0, \boldsymbol{p}_f, \boldsymbol{v}_0, \boldsymbol{v}_f, \boldsymbol{c}_v, \boldsymbol{c}_a \in \mathbb{R}^3$. Also, let $\boldsymbol{\lambda}_R$ denote the set of eigenvalues of R. Given the assumption that R is a symmetric and positive definite matrix

$$\lambda_R > 0\ \forall\ \lambda_R \in \boldsymbol{\lambda}_R. \tag{241}$$

Observe that

$$H_4 = C_I + \frac{1}{2}\boldsymbol{c}_a^T R \boldsymbol{c}_a. \tag{242}$$

Since R is assumed to be a symmetric and positive definite matrix,

$$\boldsymbol{c}_a^T R \boldsymbol{c}_a > 0 \ \forall \ \boldsymbol{c}_a \in \mathbb{R}^3 \setminus \mathbf{0}. \tag{243}$$

Therefore, given the assumption that $C_I \in \mathbb{R}_{>0}$, it can be seen that H_4 is positive $\forall \ \boldsymbol{x}_0, \boldsymbol{x}_f, \boldsymbol{c} \in \mathbb{R}^6$.

It can be seen from (240) that

$$H_2 = \boldsymbol{X}_2^T \bar{H}_2 \boldsymbol{X}_2, \tag{244}$$

where

$$\boldsymbol{X}_2 = \begin{bmatrix} \boldsymbol{v}_0 \\ \boldsymbol{v}_f \\ \boldsymbol{c}_v \end{bmatrix}, \quad \bar{H}_2 = \begin{bmatrix} -2R & -R & -3R \\ -R & -2R & -3R \\ -3R & -3R & -6R \end{bmatrix} = \begin{bmatrix} -2 & -1 & -3 \\ -1 & -2 & -3 \\ -3 & -3 & -6 \end{bmatrix} \otimes R \ . \tag{245}$$

The eigenvalues of $\bar{H}_2$ can be obtained according to Lemma 7 as

$$\bar{\boldsymbol{\lambda}}_2 = \{\lambda \lambda_R : \forall \ \lambda \in \{-9, -1, 0\}, \forall \ \lambda_R \in \boldsymbol{\lambda}_R\}, \tag{246}$$

where, given the result of (241), it can be seen that

$$\bar{\lambda}_2 \leq 0 \ \forall \ \bar{\lambda}_2 \in \bar{\boldsymbol{\lambda}}_2. \tag{247}$$

Therefore H_2 is non-positive $\forall \ \boldsymbol{x}_0, \boldsymbol{x}_f, \boldsymbol{c} \in \mathbb{R}^6$. Similarly, it can be seen from (240) that

$$H_0 = \boldsymbol{X}_0^T \bar{H}_0 \boldsymbol{X}_0,$$

$$\boldsymbol{X}_0 = \begin{bmatrix} \boldsymbol{p}_0 \\ \boldsymbol{p}_f \end{bmatrix}, \quad \bar{H}_0 = \begin{bmatrix} -18R & 18R \\ 18R & -18R \end{bmatrix} = \begin{bmatrix} -18 & 18 \\ 18 & -18 \end{bmatrix} \otimes R \tag{248}$$

where it can be seen that the eigenvalues of $\bar{H}_0$ verify

$$\bar{\boldsymbol{\lambda}}_0 = \{\lambda \lambda_R : \forall \ \lambda \in \{-36, 0\}, \forall \ \lambda_R \in \boldsymbol{\lambda}_R\}. \tag{249}$$

Furthermore, it can be seen that

$$\bar{\lambda}_0 \leq 0 \; \forall \; \bar{\lambda}_0 \in \bar{\boldsymbol{\lambda}}_0, \tag{250}$$

which implies that H_0 is non-positive $\forall \; \boldsymbol{x}_0, \boldsymbol{x}_f, \boldsymbol{c} \in \mathbb{R}^6$. □

Theorem 13. *If $H_2 = 0$ and $R \in \mathbb{R}^{3\times 3}$ is symmetric and positive definite, then the following condition is satisfied*

$$\boldsymbol{v}_0 = \boldsymbol{v}_f = -\boldsymbol{c}_v. \tag{251}$$

Proof. Since R is assumed to be symmetric it can be seen that

$$\boldsymbol{y}_1^T R \boldsymbol{y}_2 = \boldsymbol{y}_2^T R \boldsymbol{y}_1 \; \forall \; \boldsymbol{y}_1, \boldsymbol{y}_2 \in \mathbb{R}^3. \tag{252}$$

Using (240) and (252) it can be seen that

$$\begin{aligned} -H_2 &= 2\boldsymbol{v}_0^T R \boldsymbol{v}_0 + 2\boldsymbol{v}_f^T R \boldsymbol{v}_f + 6\boldsymbol{c}_v^T R \boldsymbol{c}_v + 2\boldsymbol{v}_0^T R \boldsymbol{v}_f + 6\boldsymbol{v}_0^T R \boldsymbol{c}_v + 6\boldsymbol{v}_f^T R \boldsymbol{c}_v \\ &= 6\left(\boldsymbol{c}_v + \tfrac{1}{2}\boldsymbol{v}_0 + \tfrac{1}{2}\boldsymbol{v}_f\right)^T R \left(\boldsymbol{c}_v + \tfrac{1}{2}\boldsymbol{v}_0 + \tfrac{1}{2}\boldsymbol{v}_f\right) + \tfrac{1}{2}\left(\boldsymbol{v}_0 - \boldsymbol{v}_f\right)^T R \left(\boldsymbol{v}_0 - \boldsymbol{v}_f\right) \end{aligned}, \tag{253}$$

which shows that, since R is assumed to be symmetric and positive definite, H_2 is the sum of two non-positive terms. Therefore, if $H_2 = 0$ then

$$\begin{aligned} \boldsymbol{c}_v + \tfrac{1}{2}\boldsymbol{v}_0 + \tfrac{1}{2}\boldsymbol{v}_f &= 0 \\ \boldsymbol{v}_0 - \boldsymbol{v}_f &= 0 \end{aligned} \Rightarrow \boldsymbol{v}_0 = \boldsymbol{v}_f = -\boldsymbol{c}_v. \tag{254}$$

□

Theorem 14. *If $H_0 = 0$ and $R \in \mathbb{R}^{3\times 3}$ is symmetric and positive definite, then the following condition is satisfied*

$$\boldsymbol{p}_0 = \boldsymbol{p}_f. \tag{255}$$

Proof. From (240) and (252), expanding (248) yields

$$\begin{aligned} H_0\left(\boldsymbol{x}_0, \boldsymbol{x}_f, \boldsymbol{c}\right) &= -18\boldsymbol{p}_0^T R \boldsymbol{p}_0 + 36\boldsymbol{p}_0^T R \boldsymbol{p}_f - 18\boldsymbol{p}_f^T R \boldsymbol{p}_f \\ &= -18\left(\boldsymbol{p}_0 - \boldsymbol{p}_f\right)^T R \left(\boldsymbol{p}_0 - \boldsymbol{p}_f\right). \end{aligned} \tag{256}$$

Since R is assumed to be symmetric and positive definite, if $H_0 = 0$ then

$$\boldsymbol{p}_0 = \boldsymbol{p}_f. \tag{257}$$

□

Theorem 15. *If $R \in \mathbb{R}^{3\times 3}$ is symmetric and positive definite then $H_1 = 0$ if either $H_0 = 0$ or $H_2 = 0$.*

Proof. Expanding the definition of H_1 in (235) and using (240) yields

$$H_1 = -12\boldsymbol{p}_0^T R\boldsymbol{v}_0 + 12\boldsymbol{v}_0^T R\boldsymbol{p}_f - 12\boldsymbol{p}_0^T R\boldsymbol{v}_f + 12\boldsymbol{p}_f^T R\boldsymbol{v}_f - 24\boldsymbol{p}_0^T R\boldsymbol{c}_v + 24\boldsymbol{p}_f^T R\boldsymbol{c}_v, \tag{258}$$

which is equivalent to

$$H_1 = -24\left(\boldsymbol{p}_0 - \boldsymbol{p}_f\right)^T R\left(\boldsymbol{c}_v + \frac{1}{2}\boldsymbol{v}_0 + \frac{1}{2}\boldsymbol{v}_f\right), \tag{259}$$

given the result in (252). Therefore, since R is assumed to be positive definite, $H_1 = 0$ if either $\boldsymbol{p}_0 = \boldsymbol{p}_f$ or $\boldsymbol{v}_0 = \boldsymbol{v}_f = -\boldsymbol{c}_v$. It can be seen from Theorems 13 and 14 that this happens if $H_0 = 0$ or $H_2 = 0$. □

Theorem 16. *If $R \in \mathbb{R}^{3\times 3}$ is symmetric and positive definite and $C_I \in \mathbb{R}_{>0}$, then there is at least one non-negative real root of* (234) *for all $\boldsymbol{x}_0, \boldsymbol{x}_f, \boldsymbol{c} \in \mathbb{R}^6$.*

Proof. From Theorem 12 it can be seen that

$$\begin{array}{rcl} H_4 > 0 & \forall & \boldsymbol{x}_0, \boldsymbol{x}_f, \boldsymbol{c} \in \mathbb{R}^6 \\ H_2 \leq 0 & \forall & \boldsymbol{x}_0, \boldsymbol{x}_f, \boldsymbol{c} \in \mathbb{R}^6 \\ H_0 \leq 0 & \forall & \boldsymbol{x}_0, \boldsymbol{x}_f, \boldsymbol{c} \in \mathbb{R}^6 \end{array} . \tag{260}$$

Additionally, from Theorems 13 – 15 it can be seen that

$$H_1 = 0 \quad \texttt{if either} \quad H_2 = 0 \quad \texttt{or} \quad H_0 = 0 \ . \tag{261}$$

Given (260) and (261) it can be seen that there exist multiple cases of the solutions of (234). These cases can be evaluated using Descarte's Rule of Signs from Lemma 2 on page 50. Let the function in (234) be denoted by

$$f\left(t_f^*\right) = H_4 t_f^{*^4} + H_2 t_f^{*^2} + H_1 t_f^* + H_0. \tag{262}$$

Case 1: $H_2 = 0$, $H_1 = 0$, $H_0 = 0$.

For this case it can be seen that there are four roots of (262) at $t_f^* = 0$.

Case 2: $H_2 < 0$, $H_1 > 0$, $H_0 < 0$.

For this case it can be seen that there are three sign changes in the sequence of coefficients in $f(t_f^*)$. Therefore, from Lemma 2, (234) either has one or three positive real roots.

Case 3: $H_2 < 0$, $H_1 < 0$, $H_0 < 0$. For this case it can be seen that there is one sign change in the sequence of coefficients in $f(t_f^*)$. Therefore, from Lemma 2, (234) has one positive real root.

Case 4: $H_2(\cdot) < 0$, $H_1(\cdot) = 0$, and $H_0(\cdot) = 0$.

For this case it can be seen that there is one sign change in the sequence of coefficients in $f(t_f^*)$, which implies that there is one positive real root of (262). Furthermore, (262) simplifies to

$$f\left(t_f^*\right) = t_f^{*2}\left(H_4 t_f^{*^2} + H_2\right). \tag{263}$$

Therefore, there exists two roots at $t_f^* = 0$ and the positive real root is

$$t_f^* = \sqrt{\frac{-H_2}{H_4}}. \tag{264}$$

Case 5: $H_2(\cdot) = 0$, $H_1(\cdot) = 0$, and $H_0(\cdot) < 0$.

For this case it can be seen that there is one sign change in the sequence of coefficients in $f(t_f^*)$, which implies that there is one positive real root of (262). Furthermore, (262) simplifies to the biquadratic equation

$$f\left(t_f^*\right) = H_4 t_f^{*^4} + H_0. \tag{265}$$

Therefore, the single positive real root is

$$t_f^* = \sqrt[4]{\frac{-H_0}{H_4}}. \tag{266}$$

Case 6: $H_2(\cdot) < 0$, $H_1(\cdot) = 0$, and $H_0(\cdot) < 0$.

For this case it can be seen that there is one sign change in the sequence of coefficients in $f(t)$, which implies that there is one positive real root of (262). Furthermore, (262) simplifies to

$$f\left(\tau_f^*\right) = H_4 \tau_f^{*^2} + H_2 \tau_f^* + H_0, \tag{267}$$

where $\tau_f^* = t_f^{*^2}$. Therefore, the single positive real root is

$$t_f^* = \sqrt{\frac{-H_2 + \sqrt{H_2^2 - 4H_4H_0}}{2H_4}}. \tag{268}$$

□

Appendix B - Double Integrator With Drift: Optimal Control

The system is described by

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + B\boldsymbol{u} + \boldsymbol{c}, \quad A = \begin{bmatrix} 0_3 & I_3 \\ 0_3 & 0_3 \end{bmatrix}, \quad B = \begin{bmatrix} 0_3 \\ I_3 \end{bmatrix}, \tag{269}$$

where $\boldsymbol{c} \in \mathbb{R}^6$. It can be seen that

$$e^{A\tau} = \begin{bmatrix} I_3 & \tau I_3 \\ 0_3 & I_3 \end{bmatrix}. \tag{270}$$

The controllability Grammian in (30) can therefore be evaluated as

$$\begin{aligned}
G_c\left(0, t_f^*\right) &= \int_0^{t_f^*} e^{A\left(t_f^* - \tau\right)} B R^{-1} B^T e^{A^T\left(t_f^* - \tau\right)} d\tau \\
&= \int_0^{t_f^*} \begin{bmatrix} I_3 & \left(t_f^* - \tau\right) I_3 \\ 0_3 & I_3 \end{bmatrix} \begin{bmatrix} 0_3 \\ I_3 \end{bmatrix} R^{-1} \begin{bmatrix} 0_3 & I_3 \end{bmatrix} \begin{bmatrix} I_3 & 0_3 \\ \left(t_f^* - \tau\right) I_3 & I_3 \end{bmatrix} d\tau \\
&= \int_0^{t_f^*} \begin{bmatrix} I_3 & \left(t_f^* - \tau\right) I_3 \\ 0_3 & I_3 \end{bmatrix} \begin{bmatrix} 0_3 \\ R^{-1} \end{bmatrix} \begin{bmatrix} 0_3 & I_3 \end{bmatrix} \begin{bmatrix} I_3 & 0_3 \\ \left(t_f^* - \tau\right) I_3 & I_3 \end{bmatrix} d\tau \\
&= \int_0^{t_f^*} \begin{bmatrix} I_3 & \left(t_f^* - \tau\right) I_3 \\ 0_3 & I_3 \end{bmatrix} \begin{bmatrix} 0_3 & 0_3 \\ 0_3 & R^{-1} \end{bmatrix} \begin{bmatrix} I_3 & 0_3 \\ \left(t_f^* - \tau\right) I_3 & I_3 \end{bmatrix} d\tau \\
&= \int_0^{t_f^*} \begin{bmatrix} 0_3 & \left(t_f^* - \tau\right) R^{-1} \\ 0_3 & R^{-1} \end{bmatrix} \begin{bmatrix} I_3 & 0_3 \\ \left(t_f^* - \tau\right) I_3 & I_3 \end{bmatrix} d\tau \\
&= \int_0^{t_f^*} \begin{bmatrix} \left(t_f^* - \tau\right)^2 R^{-1} & \left(t_f^* - \tau\right) R^{-1} \\ \left(t_f^* - \tau\right) R^{-1} & R^{-1} \end{bmatrix} d\tau = \begin{bmatrix} \frac{t_f^{*3}}{3} R^{-1} & \frac{t_f^{*2}}{2} R^{-1} \\ \frac{t_f^{*2}}{2} R^{-1} & t_f^* R^{-1} \end{bmatrix}
\end{aligned} \tag{271}$$

and its inverse as

$$G_c\left(0,t_f^*\right)^{-1} = \begin{bmatrix} \frac{12}{t_f^{*3}}R & -\frac{6}{t_f^{*2}}R \\ -\frac{6}{t_f^{*2}}R & \frac{4}{t_f^*}R \end{bmatrix}. \tag{272}$$

It can be seen that

$$\int_0^{t_f^*} e^{A\left(t_f^*-\tau\right)}d\tau = \int_0^{t_f^*} \begin{bmatrix} I_3 & \left(t_f^*-\tau\right)I_3 \\ 0_3 & I_3 \end{bmatrix} d\tau = \begin{bmatrix} t_f^* I_3 & \frac{t_f^{*2}}{2}I_3 \\ 0_3 & t_f^* I_3 \end{bmatrix}. \tag{273}$$

The vector of costates at $t = t_f^*$ in (29) can then be evaluated as

$$\begin{aligned}
\boldsymbol{\lambda}^*\left(t_f^*\right) &= -G_c\left(0,t_f^*\right)^{-1}\left(\boldsymbol{x}_f - e^{At_f^*}\boldsymbol{x}_0 - \int_0^{t_f^*} e^{A\left(t_f^*-\tau\right)}\boldsymbol{c}d\tau\right) \\
&= -\begin{bmatrix} \frac{12}{t_f^{*3}}R & -\frac{6}{t_f^{*2}}R \\ -\frac{6}{t_f^{*2}}R & \frac{4}{t_f^*}R \end{bmatrix}\left(\boldsymbol{x}_f - \begin{bmatrix} I_3 & t_f^* I_3 \\ 0_3 & I_3 \end{bmatrix}\boldsymbol{x}_0 - \begin{bmatrix} t_f^* I_3 & \frac{t_f^{*2}}{2}I_3 \\ 0_3 & t_f^* I_3 \end{bmatrix}\boldsymbol{c}\right) \\
&= \begin{bmatrix} -\frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R \\ \frac{6}{t_f^{*2}}R & -\frac{4}{t_f^*}R \end{bmatrix}\boldsymbol{x}_f + \begin{bmatrix} \frac{12}{t_f^{*3}}R & -\frac{6}{t_f^{*2}}R \\ -\frac{6}{t_f^{*2}}R & \frac{4}{t_f^*}R \end{bmatrix}\begin{bmatrix} I_3 & t_f^* I_3 \\ 0_3 & I_3 \end{bmatrix}\boldsymbol{x}_0 \\
&\quad + \begin{bmatrix} \frac{12}{t_f^{*3}}R & -\frac{6}{t_f^{*2}}R \\ -\frac{6}{t_f^{*2}}R & \frac{4}{t_f^*}R \end{bmatrix}\begin{bmatrix} t_f^* I_3 & \frac{t_f^{*2}}{2}I_3 \\ 0_3 & t_f^* I_3 \end{bmatrix}\boldsymbol{c} \\
&= \begin{bmatrix} -\frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R \\ \frac{6}{t_f^{*2}}R & -\frac{4}{t_f^*}R \end{bmatrix}\boldsymbol{x}_f + \begin{bmatrix} \frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R \\ -\frac{6}{t_f^{*2}}R & -\frac{2}{t_f^*}R \end{bmatrix}\boldsymbol{x}_0 + \begin{bmatrix} \frac{12}{t_f^{*2}}R & 0_3 \\ -\frac{6}{t_f^*}R & R \end{bmatrix}\boldsymbol{c} \\
&= \begin{bmatrix} \frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & -\frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & \frac{12}{t_f^{*2}}R & 0_3 \\ -\frac{6}{t_f^{*2}}R & -\frac{2}{t_f^*}R & \frac{6}{t_f^{*2}}R & -\frac{4}{t_f^*}R & -\frac{6}{t_f^*}R & R \end{bmatrix}\begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}
\end{aligned} \tag{274}$$

The optimal control input trajectory in (28) can then be obtained as

$$
\begin{aligned}
\boldsymbol{u}^*(t) &= -R^{-1}B^T e^{A^T\left(t_f^*-t\right)}\boldsymbol{\lambda}^*(t_f^*) \\
&= -R^{-1}\begin{bmatrix}0_3 & I_3\end{bmatrix}\begin{bmatrix} I_3 & 0_3 \\ \left(t_f^*-t\right)I_3 & I_3\end{bmatrix}\begin{bmatrix} \frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & -\frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & \frac{12}{t_f^{*2}}R & 0_3 \\ -\frac{6}{t_f^{*2}}R & -\frac{2}{t_f^*}R & \frac{6}{t_f^{*2}}R & -\frac{4}{t_f^*}R & -\frac{6}{t_f^*}R & R\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c}\end{bmatrix} \\
&= \begin{bmatrix}0_3 & -R^{-1}\end{bmatrix}\begin{bmatrix} I_3 & 0_3 \\ \left(t_f^*-t\right)I_3 & I_3\end{bmatrix}\begin{bmatrix} \frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & -\frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & \frac{12}{t_f^{*2}}R & 0_3 \\ -\frac{6}{t_f^{*2}}R & -\frac{2}{t_f^*}R & \frac{6}{t_f^{*2}}R & -\frac{4}{t_f^*}R & -\frac{6}{t_f^*}R & R\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c}\end{bmatrix} \qquad (275) \\
&= \begin{bmatrix}\left(t-t_f^*\right)R^{-1} & -R^{-1}\end{bmatrix}\begin{bmatrix} \frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & -\frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & \frac{12}{t_f^{*2}}R & 0_3 \\ -\frac{6}{t_f^{*2}}R & -\frac{2}{t_f^*}R & \frac{6}{t_f^{*2}}R & -\frac{4}{t_f^*}R & -\frac{6}{t_f^*}R & R\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c}\end{bmatrix} \\
&= \begin{bmatrix}\frac{6\left(2t-t_f^*\right)}{t_f^{*3}}I_3 & \frac{2\left(3t-2t_f^*\right)}{t_f^{*2}}I_3 & -\frac{6\left(2t-t_f^*\right)}{t_f^{*3}}I_3 & \frac{2\left(3t-t_f^*\right)}{t_f^{*2}}I_3 & \frac{6\left(2t-t_f^*\right)}{t_f^{*2}}I_3 & -I_3\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c}\end{bmatrix}.
\end{aligned}
$$

It can be seen that

$$\boldsymbol{u}^*(t)^T R\boldsymbol{u}^*(t)$$

$$= \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix} \frac{6\left(2t-t_f^*\right)}{t_f^{*3}} I_3 \\ \frac{2\left(3t-2t_f^*\right)}{t_f^{*2}} I_3 \\ -\frac{6\left(2t-t_f^*\right)}{t_f^{*3}} I_3 \\ \frac{2\left(3t-t_f^*\right)}{t_f^{*2}} I_3 \\ \frac{6\left(2t-t_f^*\right)}{t_f^{*2}} I_3 \\ -I_3 \end{bmatrix} R \begin{bmatrix} \frac{6\left(2t-t_f^*\right)}{t_f^{*3}} I_3 & \frac{2\left(3t-2t_f^*\right)}{t_f^{*2}} I_3 & -\frac{6\left(2t-t_f^*\right)}{t_f^{*3}} I_3 & \frac{2\left(3t-t_f^*\right)}{t_f^{*2}} I_3 & \frac{6\left(2t-t_f^*\right)}{t_f^{*2}} I_3 & -I_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix} \tag{276}$$

$$= \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix} \frac{36\left(2t-t_f^*\right)^2}{t_f^{*6}} R & \frac{12\left(2t-t_f^*\right)\left(3t-2t_f^*\right)}{t_f^{*5}} R & -\frac{36\left(2t-t_f^*\right)^2}{t_f^{*6}} R & \frac{12\left(2t-t_f^*\right)\left(3t-t_f^*\right)}{t_f^{*5}} R & \frac{36\left(2t-t_f^*\right)^2}{t_f^{*5}} R & -\frac{6\left(2t-t_f^*\right)}{t_f^{*3}} R \\ \frac{12\left(3t-2t_f^*\right)\left(2t-t_f^*\right)}{t_f^{*5}} R & \frac{4\left(3t-2t_f^*\right)^2}{t_f^{*4}} R & -\frac{12\left(3t-2t_f^*\right)\left(2t-t_f^*\right)}{t_f^{*5}} R & \frac{4\left(3t-2t_f^*\right)\left(3t-t_f^*\right)}{t_f^{*4}} R & \frac{12\left(3t-2t_f^*\right)\left(2t-t_f^*\right)}{t_f^{*4}} R & -\frac{2\left(3t-2t_f^*\right)}{t_f^{*2}} R \\ -\frac{36\left(2t-t_f^*\right)^2}{t_f^{*6}} R & -\frac{12\left(2t-t_f^*\right)\left(3t-2t_f^*\right)}{t_f^{*5}} R & \frac{36\left(2t-t_f^*\right)^2}{t_f^{*6}} R & -\frac{12\left(2t-t_f^*\right)\left(3t-t_f^*\right)}{t_f^{*5}} R & -\frac{36\left(2t-t_f^*\right)^2}{t_f^{*5}} R & \frac{6\left(2t-t_f^*\right)}{t_f^{*3}} R \\ \frac{12\left(3t-t_f^*\right)\left(2t-t_f^*\right)}{t_f^{*5}} R & \frac{4\left(3t-t_f^*\right)\left(3t-2t_f^*\right)}{t_f^{*4}} R & -\frac{12\left(3t-t_f^*\right)\left(2t-t_f^*\right)}{t_f^{*5}} R & \frac{4\left(3t-t_f^*\right)^2}{t_f^{*4}} R & \frac{12\left(3t-t_f^*\right)\left(2t-t_f^*\right)}{t_f^{*4}} R & -\frac{2\left(3t-t_f^*\right)}{t_f^{*2}} R \\ \frac{36\left(2t-t_f^*\right)^2}{t_f^{*5}} R & \frac{12\left(2t-t_f^*\right)\left(3t-2t_f^*\right)}{t_f^{*4}} R & -\frac{36\left(2t-t_f^*\right)^2}{t_f^{*5}} R & \frac{12\left(2t-t_f^*\right)\left(3t-t_f^*\right)}{t_f^{*4}} R & \frac{36\left(2t-t_f^*\right)^2}{t_f^{*4}} R & -\frac{6\left(2t-t_f^*\right)}{t_f^{*2}} R \\ -\frac{6\left(2t-t_f^*\right)}{t_f^{*3}} R & -\frac{2\left(3t-2t_f^*\right)}{t_f^{*2}} R & \frac{6\left(2t-t_f^*\right)}{t_f^{*3}} R & -\frac{2\left(3t-t_f^*\right)}{t_f^{*2}} R & -\frac{6\left(2t-t_f^*\right)}{t_f^{*2}} R & R \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}$$

and

$$\int_0^{t_f^*} \boldsymbol{u}^*(\tau)^T R\boldsymbol{u}^*(\tau)\, d\tau = \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix} \frac{12}{t_f^{*3}} R & \frac{6}{t_f^{*2}} R & -\frac{12}{t_f^{*3}} R & \frac{6}{t_f^{*2}} R & \frac{12}{t_f^{*2}} R & 0_3 \\ \frac{6}{t_f^{*2}} R & \frac{4}{t_f^*} R & -\frac{6}{t_f^{*2}} R & \frac{2}{t_f^*} R & \frac{6}{t_f^*} R & R \\ -\frac{12}{t_f^{*3}} R & -\frac{6}{t_f^{*2}} R & \frac{12}{t_f^{*3}} R & -\frac{6}{t_f^{*2}} R & -\frac{12}{t_f^{*2}} R & 0_3 \\ \frac{6}{t_f^{*2}} R & \frac{2}{t_f^*} R & -\frac{6}{t_f^{*2}} R & \frac{4}{t_f^*} R & \frac{6}{t_f^*} R & -R \\ \frac{12}{t_f^{*2}} R & \frac{6}{t_f^*} R & -\frac{12}{t_f^{*2}} R & \frac{6}{t_f^*} R & \frac{12}{t_f^*} R & 0_3 \\ 0_3 & R & 0_3 & -R & 0_3 & t_f^* R \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}. \tag{277}$$

This allows to evaluate the cost as

$$
\begin{aligned}
C\left(\boldsymbol{x}^*(t),\boldsymbol{u}^*(t),t_f^*\right) &= \int_0^{t_f}\left(\tfrac{1}{2}\boldsymbol{u}^*(\tau)^T R\boldsymbol{u}^*(\tau)+C_I\right)d\tau \\
&= \tfrac{1}{2}\begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix}^T
\begin{bmatrix}
\frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & -\frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & \frac{12}{t_f^{*2}}R & 0_3\\
\frac{6}{t_f^{*2}}R & \frac{4}{t_f^{*}}R & -\frac{6}{t_f^{*2}}R & \frac{2}{t_f^{*}}R & \frac{6}{t_f^{*}}R & R\\
-\frac{12}{t_f^{*3}}R & -\frac{6}{t_f^{*2}}R & \frac{12}{t_f^{*3}}R & -\frac{6}{t_f^{*2}}R & -\frac{12}{t_f^{*2}}R & 0_3\\
\frac{6}{t_f^{*2}}R & \frac{2}{t_f^{*}}R & -\frac{6}{t_f^{*2}}R & \frac{4}{t_f^{*}}R & \frac{6}{t_f^{*}}R & -R\\
\frac{12}{t_f^{*2}}R & \frac{6}{t_f^{*}}R & -\frac{12}{t_f^{*2}}R & \frac{6}{t_f^{*}}R & \frac{12}{t_f^{*}}R & 0_3\\
0_3 & R & 0_3 & -R & 0_3 & t_f^*R
\end{bmatrix}
\begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix} + t_f^*C_I,.
\end{aligned}
\tag{278}
$$

From (275) it can be seen that

$$
\boldsymbol{u}^*\left(t_f^*\right) = \begin{bmatrix}\frac{6}{t_f^{*2}}I_3 & \frac{2}{t_f^{*}}I_3 & -\frac{6}{t_f^{*2}}I_3 & \frac{4}{t_f^{*}}I_3 & \frac{6}{t_f^{*}}I_3 & -I_3\end{bmatrix}
\begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix}. \tag{279}
$$

Given this result one can write

$$
\begin{aligned}
\tfrac{1}{2}\boldsymbol{u}^*\left(t_f^*\right)^T R\boldsymbol{u}^*\left(t_f^*\right) &= \tfrac{1}{2}\begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix}^T
\begin{bmatrix}\frac{6}{t_f^{*2}}I_3\\ \frac{2}{t_f^{*}}I_3\\ -\frac{6}{t_f^{*2}}I_3\\ \frac{4}{t_f^{*}}I_3\\ \frac{6}{t_f^{*}}I_3\\ -I_3\end{bmatrix}
R\begin{bmatrix}\frac{6}{t_f^{*2}}I_3 & \frac{2}{t_f^{*}}I_3 & -\frac{6}{t_f^{*2}}I_3 & \frac{4}{t_f^{*}}I_3 & \frac{6}{t_f^{*}}I_3 & -I_3\end{bmatrix}
\begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix} \\
&= \begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix}^T
\begin{bmatrix}
\frac{18}{t_f^{*4}}R & \frac{6}{t_f^{*3}}R & -\frac{18}{t_f^{*4}}R & \frac{12}{t_f^{*3}}R & \frac{18}{t_f^{*3}}R & -\frac{3}{t_f^{*2}}R\\
\frac{6}{t_f^{*3}}R & \frac{2}{t_f^{*2}}R & -\frac{6}{t_f^{*3}}R & \frac{4}{t_f^{*2}}R & \frac{6}{t_f^{*2}}R & -\frac{1}{t_f^{*}}R\\
-\frac{18}{t_f^{*4}}R & -\frac{6}{t_f^{*3}}R & \frac{18}{t_f^{*4}}R & -\frac{12}{t_f^{*3}}R & -\frac{18}{t_f^{*3}}R & \frac{3}{t_f^{*2}}R\\
\frac{12}{t_f^{*3}}R & \frac{4}{t_f^{*2}}R & -\frac{12}{t_f^{*3}}R & \frac{8}{t_f^{*2}}R & \frac{12}{t_f^{*2}}R & -\frac{2}{t_f^{*}}R\\
\frac{18}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & -\frac{18}{t_f^{*3}}R & \frac{12}{t_f^{*2}}R & \frac{18}{t_f^{*2}}R & -\frac{3}{t_f^{*}}R\\
-\frac{3}{t_f^{*2}}R & -\frac{1}{t_f^{*}}R & \frac{3}{t_f^{*2}}R & -\frac{2}{t_f^{*}}R & -\frac{3}{t_f^{*}}R & \frac{1}{2}R
\end{bmatrix}
\begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix}
\end{aligned}
\tag{280}
$$

and

$$\boldsymbol{\lambda}^* \left(t_f^*\right)^T B \boldsymbol{u}^* \left(t_f^*\right)$$

$$= \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix} \frac{12}{t_f^{*3}} R & -\frac{6}{t_f^{*2}} R \\ \frac{6}{t_f^{*2}} R & -\frac{2}{t_f^*} R \\ -\frac{12}{t_f^{*3}} R & \frac{6}{t_f^{*2}} R \\ \frac{6}{t_f^{*2}} R & -\frac{4}{t_f^*} R \\ \frac{12}{t_f^{*2}} R & -\frac{6}{t_f^*} R \\ 0_3 & R \end{bmatrix} \begin{bmatrix} 0_3 \\ I_3 \end{bmatrix} \begin{bmatrix} \frac{6}{t_f^{*2}} I_3 & \frac{2}{t_f^*} I_3 & -\frac{6}{t_f^{*2}} I_3 & \frac{4}{t_f^*} I_3 & \frac{6}{t_f^*} I_3 & -I_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix} -\frac{6}{t_f^{*2}} R \\ -\frac{2}{t_f^*} R \\ \frac{6}{t_f^{*2}} R \\ -\frac{4}{t_f^*} R \\ -\frac{6}{t_f^*} R \\ R \end{bmatrix} \begin{bmatrix} \frac{6}{t_f^{*2}} I_3 & \frac{2}{t_f^*} I_3 & -\frac{6}{t_f^{*2}} I_3 & \frac{4}{t_f^*} I_3 & \frac{6}{t_f^*} I_3 & -I_3 \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix} \tag{281}$$

$$= \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix} -\frac{36}{t_f^{*4}} R & -\frac{12}{t_f^{*3}} R & \frac{36}{t_f^{*4}} R & -\frac{24}{t_f^{*3}} R & -\frac{36}{t_f^{*3}} R & \frac{6}{t_f^{*2}} R \\ -\frac{12}{t_f^{*3}} R & -\frac{4}{t_f^{*2}} R & \frac{12}{t_f^{*3}} R & -\frac{8}{t_f^{*2}} R & -\frac{12}{t_f^{*2}} R & \frac{2}{t_f^*} R \\ \frac{36}{t_f^{*4}} R & \frac{12}{t_f^{*3}} R & -\frac{36}{t_f^{*4}} R & \frac{24}{t_f^{*3}} R & \frac{36}{t_f^{*3}} R & -\frac{6}{t_f^{*2}} R \\ -\frac{24}{t_f^{*3}} R & -\frac{8}{t_f^{*2}} R & \frac{24}{t_f^{*3}} R & -\frac{16}{t_f^{*2}} R & -\frac{24}{t_f^{*2}} R & \frac{4}{t_f^*} R \\ -\frac{36}{t_f^{*3}} R & -\frac{12}{t_f^{*2}} R & \frac{36}{t_f^{*3}} R & -\frac{24}{t_f^{*2}} R & -\frac{36}{t_f^{*2}} R & \frac{6}{t_f^*} R \\ \frac{6}{t_f^{*2}} R & \frac{2}{t_f^*} R & -\frac{6}{t_f^{*2}} R & \frac{4}{t_f^*} R & \frac{6}{t_f^*} R & -R \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}$$

Furthermore, it can be seen that

$$\begin{aligned}
\boldsymbol{\lambda}^*\left(t_f^*\right)^T\left(A\boldsymbol{x}_f+\boldsymbol{c}\right) &= \boldsymbol{\lambda}^*\left(t_f^*\right)^T\begin{bmatrix}0_3 & 0_3 & 0_3 & I_3 & I_3 & 0_3\\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & I_3\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix}\\
&= \begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix}^T\begin{bmatrix}\frac{12}{t_f^{*3}}R & -\frac{6}{t_f^{*2}}R\\ \frac{6}{t_f^{*2}}R & -\frac{2}{t_f^*}R\\ -\frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R\\ \frac{6}{t_f^{*2}}R & -\frac{4}{t_f^*}R\\ \frac{12}{t_f^{*2}}R & -\frac{6}{t_f^*}R\\ 0_3 & R\end{bmatrix}\begin{bmatrix}0_3 & 0_3 & 0_3 & I_3 & I_3 & 0_3\\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & I_3\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix}\\
&= \begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix}^T\begin{bmatrix}0_3 & 0_3 & 0_3 & \frac{12}{t_f^{*3}}R & \frac{12}{t_f^{*3}}R & -\frac{6}{t_f^{*2}}R\\ 0_3 & 0_3 & 0_3 & \frac{6}{t_f^{*2}}R & \frac{6}{t_f^{*2}}R & -\frac{2}{t_f^*}R\\ 0_3 & 0_3 & 0_3 & -\frac{12}{t_f^{*3}}R & -\frac{12}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R\\ 0_3 & 0_3 & 0_3 & \frac{6}{t_f^{*2}}R & \frac{6}{t_f^{*2}}R & -\frac{4}{t_f^*}R\\ 0_3 & 0_3 & 0_3 & \frac{12}{t_f^{*2}}R & \frac{12}{t_f^{*2}}R & -\frac{6}{t_f^*}R\\ 0_3 & 0_3 & 0_3 & 0_3 & 0_3 & R\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix} \qquad (282)\\
&= \begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix}^T\begin{bmatrix}0_3 & 0_3 & 0_3 & \frac{6}{t_f^{*3}}R & \frac{6}{t_f^{*3}}R & -\frac{3}{t_f^{*2}}R\\ 0_3 & 0_3 & 0_3 & \frac{3}{t_f^{*2}}R & \frac{3}{t_f^{*2}}R & -\frac{1}{t_f^*}R\\ 0_3 & 0_3 & 0_3 & -\frac{6}{t_f^{*3}}R & -\frac{6}{t_f^{*3}}R & \frac{3}{t_f^{*2}}R\\ \frac{6}{t_f^{*3}}R & \frac{3}{t_f^{*2}}R & -\frac{6}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & \frac{9}{t_f^{*2}}R & -\frac{2}{t_f^*}R\\ \frac{6}{t_f^{*3}}R & \frac{3}{t_f^{*2}}R & -\frac{6}{t_f^{*3}}R & \frac{9}{t_f^{*2}}R & \frac{12}{t_f^{*2}}R & -\frac{3}{t_f^*}R\\ -\frac{3}{t_f^{*2}}R & -\frac{1}{t_f^*}R & \frac{3}{t_f^{*2}}R & -\frac{2}{t_f^*}R & -\frac{3}{t_f^*}R & R\end{bmatrix}\begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix}.
\end{aligned}$$

The Hamiltonian at $t = t_f^*$ in (5) can be computed as

$$
\begin{aligned}
H\left(t_f^*\right) &= \tfrac{1}{2}\boldsymbol{u}^*\left(t_f^*\right)^T R\boldsymbol{u}^*\left(t_f^*\right) + C_I + \boldsymbol{\lambda}^*\left(t_f^*\right)^T\left(A\boldsymbol{x}_f + B\boldsymbol{u}^*\left(t_f^*\right) + \boldsymbol{c}\right) \\
&= C_I + \tfrac{1}{2}\boldsymbol{u}^*\left(t_f^*\right)^T R\boldsymbol{u}^*\left(t_f^*\right) + \boldsymbol{\lambda}^*\left(t_f^*\right)^T B\boldsymbol{u}^*\left(t_f^*\right) + \boldsymbol{\lambda}^*\left(t_f^*\right)^T(A\boldsymbol{x}_f + \boldsymbol{c}) \\
&= C_I + \begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix}^T \left(\begin{bmatrix}
\frac{18}{t_f^{*4}}R & \frac{6}{t_f^{*3}}R & -\frac{18}{t_f^{*4}}R & \frac{12}{t_f^{*3}}R & \frac{18}{t_f^{*3}}R & -\frac{3}{t_f^{*2}}R \\
\frac{6}{t_f^{*3}}R & \frac{2}{t_f^{*2}}R & -\frac{6}{t_f^{*3}}R & \frac{4}{t_f^{*2}}R & \frac{6}{t_f^{*2}}R & -\frac{1}{t_f^{*}}R \\
-\frac{18}{t_f^{*4}}R & -\frac{6}{t_f^{*3}}R & \frac{18}{t_f^{*4}}R & -\frac{12}{t_f^{*3}}R & -\frac{18}{t_f^{*3}}R & \frac{3}{t_f^{*2}}R \\
\frac{12}{t_f^{*3}}R & \frac{4}{t_f^{*2}}R & -\frac{12}{t_f^{*3}}R & \frac{8}{t_f^{*2}}R & \frac{12}{t_f^{*2}}R & -\frac{2}{t_f^{*}}R \\
\frac{18}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & -\frac{18}{t_f^{*3}}R & \frac{12}{t_f^{*2}}R & \frac{18}{t_f^{*2}}R & -\frac{3}{t_f^{*}}R \\
-\frac{3}{t_f^{*2}}R & -\frac{1}{t_f^{*}}R & \frac{3}{t_f^{*2}}R & -\frac{2}{t_f^{*}}R & -\frac{3}{t_f^{*}}R & \frac{1}{2}R
\end{bmatrix} \right. \\
&\quad + \begin{bmatrix}
-\frac{36}{t_f^{*4}}R & -\frac{12}{t_f^{*3}}R & \frac{36}{t_f^{*4}}R & -\frac{24}{t_f^{*3}}R & -\frac{36}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R \\
-\frac{12}{t_f^{*3}}R & -\frac{4}{t_f^{*2}}R & \frac{12}{t_f^{*3}}R & -\frac{8}{t_f^{*2}}R & -\frac{12}{t_f^{*2}}R & \frac{2}{t_f^{*}}R \\
\frac{36}{t_f^{*4}}R & \frac{12}{t_f^{*3}}R & -\frac{36}{t_f^{*4}}R & \frac{24}{t_f^{*3}}R & \frac{36}{t_f^{*3}}R & -\frac{6}{t_f^{*2}}R \\
-\frac{24}{t_f^{*3}}R & -\frac{8}{t_f^{*2}}R & \frac{24}{t_f^{*3}}R & -\frac{16}{t_f^{*2}}R & -\frac{24}{t_f^{*2}}R & \frac{4}{t_f^{*}}R \\
-\frac{36}{t_f^{*3}}R & -\frac{12}{t_f^{*2}}R & \frac{36}{t_f^{*3}}R & -\frac{24}{t_f^{*2}}R & -\frac{36}{t_f^{*2}}R & \frac{6}{t_f^{*}}R \\
\frac{6}{t_f^{*2}}R & \frac{2}{t_f^{*}}R & -\frac{6}{t_f^{*2}}R & \frac{4}{t_f^{*}}R & \frac{6}{t_f^{*}}R & -R
\end{bmatrix} \\
&\quad \left. + \begin{bmatrix}
0_3 & 0_3 & 0_3 & \frac{6}{t_f^{*3}}R & \frac{6}{t_f^{*3}}R & -\frac{3}{t_f^{*2}}R \\
0_3 & 0_3 & 0_3 & \frac{3}{t_f^{*2}}R & \frac{3}{t_f^{*2}}R & -\frac{1}{t_f^{*}}R \\
0_3 & 0_3 & 0_3 & -\frac{6}{t_f^{*3}}R & -\frac{6}{t_f^{*3}}R & \frac{3}{t_f^{*2}}R \\
\frac{6}{t_f^{*3}}R & \frac{3}{t_f^{*2}}R & -\frac{6}{t_f^{*3}}R & \frac{6}{t_f^{*2}}R & \frac{9}{t_f^{*2}}R & -\frac{2}{t_f^{*}}R \\
\frac{6}{t_f^{*3}}R & \frac{3}{t_f^{*2}}R & -\frac{6}{t_f^{*3}}R & \frac{9}{t_f^{*2}}R & \frac{12}{t_f^{*2}}R & -\frac{3}{t_f^{*}}R \\
-\frac{3}{t_f^{*2}}R & -\frac{1}{t_f^{*}}R & \frac{3}{t_f^{*2}}R & -\frac{2}{t_f^{*}}R & -\frac{3}{t_f^{*}}R & R
\end{bmatrix} \right) \begin{bmatrix}\boldsymbol{x}_0\\ \boldsymbol{x}_f\\ \boldsymbol{c}\end{bmatrix}
\end{aligned}
\tag{283}
$$

$$
\begin{aligned}
&= C_I + \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix}
-\frac{18}{t_f^{*4}}R & -\frac{6}{t_f^{*3}}R & \frac{18}{t_f^{*4}}R & -\frac{6}{t_f^{*3}}R & -\frac{12}{t_f^{*3}}R & 0_3 \\
-\frac{6}{t_f^{*3}}R & -\frac{2}{t_f^{*2}}R & \frac{6}{t_f^{*3}}R & -\frac{1}{t_f^{*2}}R & -\frac{3}{t_f^{*2}}R & 0_3 \\
\frac{18}{t_f^{*4}}R & \frac{6}{t_f^{*3}}R & -\frac{18}{t_f^{*4}}R & \frac{6}{t_f^{*3}}R & \frac{12}{t_f^{*3}}R & 0_3 \\
-\frac{6}{t_f^{*3}}R & -\frac{1}{t_f^{*2}}R & \frac{6}{t_f^{*3}}R & -\frac{2}{t_f^{*2}}R & -\frac{3}{t_f^{*2}}R & 0_3 \\
-\frac{12}{t_f^{*3}}R & -\frac{3}{t_f^{*2}}R & \frac{12}{t_f^{*3}}R & -\frac{3}{t_f^{*2}}R & -\frac{6}{t_f^{*2}}R & 0_3 \\
0_3 & 0_3 & 0_3 & 0_3 & 0_3 & \frac{1}{2}R
\end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix} \\
&= \frac{1}{t_f^{*4}} \left(C_I t_f^{*^4} + \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix}^T \begin{bmatrix}
-18R & -6t_f^*R & 18R & -6t_f^*R & -12t_f^*R & 0_3 \\
-6t_f^*R & -2t_f^{*^2}R & 6t_f^*R & -t_f^{*^2}R & -3t_f^{*^2}R & 0_3 \\
18R & 6t_f^*R & -18R & 6t_f^*R & 12t_f^*R & 0_3 \\
-6t_f^*R & -t_f^{*^2}R & 6t_f^*R & -2t_f^{*^2}R & -3t_f^{*^2}R & 0_3 \\
-12t_f^*R & -3t_f^{*^2}R & 12t_f^*R & -3t_f^{*^2}R & -6t_f^{*^2}R & 0_3 \\
0_3 & 0_3 & 0_3 & 0_3 & 0_3 & \frac{t_f^{*^4}}{2}R
\end{bmatrix} \begin{bmatrix} \boldsymbol{x}_0 \\ \boldsymbol{x}_f \\ \boldsymbol{c} \end{bmatrix} \right).
\end{aligned}
$$

Appendix C - Double Integrator with Drift: Vertex and Obstacle Classes

This section describes the implementation of the vertex and obstacle classes for the problem addressed in Section 2.3.2.2. Note that these classes are designed for specific optimal control problems, and may require minor or significant changes when the problem is different than that of Section 2.3.2.2. Therefore, the specific classes presented in this section are not considered general, but may be applicable to a variety of systems. They may also provide valuable insight for the implementation of these classes for other problems. Also, note that the following functions are native to MATLAB [65]:

- `vecnorm` $(\cdot)$
- `sqrt` $(\cdot)$
- `length` $(\cdot)$
- `rand` $(\cdot)$
- `zeros` $(\cdot)$
- *roots* $(\cdot)$
- `ode45` $(\cdot)$
- `cos` $(\cdot)$
- `eye` $(\cdot)$
- `imag` $(\cdot)$
- `min` $(\cdot)$
- `sin` $(\cdot)$

Obstacle	
Properties:	Methods:
p_c	*self* = `Obstacle` (p_c, r, h)
r	β = `CollisionCheck` $(self, x)$
h	

Figure 47: Double integrator with drift: obstacle class.

The obstacle class for the double integrator with drift of Section 2.3.2.2 is shown in Figure 47. Since the obstacles are described by cylinders in Euclidean space, they are characterized by a center position, a radius, and a height. The obstacle class has the following properties:

- p_c: A double precision three row single-column array that describes the three-dimensional position of the center of the cylindrical obstacle.
- r: A double precision variable that describes the radius of the cylindrical obstacle.
- h: a double precision variable that describes the height of the cylindrical obstacle.

Furthermore, the obstacle class has the following member functions:

- `Obstacle` (p_c, r, h): This is the class constructor, which receives each property as an input and assigns them accordingly.

- `CollisionCheck` (self, x): The input of this function

  is an n row double precision array x, where each column of x describes a state. This function evaluates if any column of x describes a state that intersects with the obstacle. If no states described by the columns of x intersect with the obstacle then the function returns *true*, and it returns *false* otherwise. The pseudocode for this function is shown in Algorithm 17.

Algorithm 17 Obstacle Class: Collision Check

function CollisionCheck(self, x)
  $d = vecnorm\left(\mathit{self}.p_c\left(1:2\right) - x\left(1:2,:\right), 2, 1\right)$;
  $I_c = d < \mathit{self}.r$;
  $I_c = x\left(3, I_c\right) > \mathit{self}.p_c\left(3\right) - \frac{\mathit{self}.h}{2}$;
  $I_c = x\left(3, I_c\right) < \mathit{self}.p_c\left(3\right) + \frac{\mathit{self}.h}{2}$;
  return $\sim$ `any` $\left(I_c\right)$;

Vertex	
Properties:	Methods:
x	$\mathit{self} =$ `Vertex` (x, c, R, C_I)
$\mathcal{X}$	$x =$ `GetState` (self)
c^*	`SetCost` (self, c^*)
c	$c^* =$ `GetCost` (self)
R	$\boldsymbol{e}_{\texttt{new}} =$ `Steer` $(\mathit{self}, \boldsymbol{x}_2)$
C_I	$[\boldsymbol{x}_{\texttt{new}}, e_{\texttt{new}}] =$ `LocalTraj` $(\mathit{self}, \boldsymbol{e}, c_{\max})$
	$CTG =$ `CTG` $(\mathit{self}, \boldsymbol{x}_2)$
	$\boldsymbol{x}_{\texttt{rand}} =$ `Sample` (self)
	$\beta =$ `CollisionCheck` (self, x)
	$c =$ `GetAffine` (self)
	$R =$ `GetR` (self)
	$C_I =$ `GetCi` (self)
	`SetState` (self, x)

Figure 48: Double integrator with drift: vertex class.

The vertex class for the double integrator with drift of Section 2.3.2.2 is shown in Figure 48. The vertex class has the following properties:

- x: An n row single-column double precision array that describes the state of the vertex.
- $\mathcal{X}$: A double precision array that describes the bounds on the state space of the system. For the particular problem addressed in section 2.3.2.2, where the state space is given by (139), this array is formatted as

$$\mathcal{X} = \begin{bmatrix} -100 & 100 \\ -100 & 100 \\ 0 & 100 \\ 0 & 20 \end{bmatrix},$$

  where the first three rows describe the lower (first column) and upper (second column) bounds on the system's position, and the fourth row describes the bounds on the velocity.
- c^*: A double precision variable that describes the cost of the vertex as is introduced in Definition 2.
- c: An n row single-column double precision array that describes the drift term of the system.
- R: An n row n column double precision array describing the symmetric and positive definite control input weighting matrix in the cost function (26).
- C_I: A positive double precision variable describing the cost index in the cost function (26).

Additionally, the vertex class has the following member functions:

- `Vertex` $(x, \mathcal{X}, c, R, C_I)$: This function is the class constructor, which receives all of the class properties except for c^* as inputs and assigns them accordingly. Also, the property c^* is initialized to infinity.
- `GetState` (self): This function returns the property x.
- `SetCost` (self, c^*): The input of this function is a double precision variable c^* that is assigned to the property c^*.
- `GetCost` (self): This function returns the property c^*.
- `Steer` $(\mathit{self}, \boldsymbol{x}_2)$: The input of this function is the vertex object $\boldsymbol{x}_2$. This function computes the unconstrained optimal state and control input trajectories, and the unconstrained optimal final time to steer the system from the state of the vertex object (self) to the state of $\boldsymbol{x}_2$. This function returns the trajectory object $\boldsymbol{e}_{\texttt{new}}$ describing the unconstrained optimal trajectory. The pseudocode for this function is given in Algorithm 19.

- `LocalTraj` ($\mathit{self}, \boldsymbol{e}, c_{\texttt{max}}$): The input of this function is the trajectory object $\boldsymbol{e}$ and the double precision variable $c_{\texttt{max}}$. This function computes the unconstrained optimal trajectory from the initial state of $\boldsymbol{e}$ toward the final state of $\boldsymbol{e}$ up to a maximum cost $c_{\texttt{max}}$, and returns the new vertex object $\boldsymbol{x}_{\texttt{new}}$ as the end vertex of this trajectory and the new trajectory $\boldsymbol{e}_{\texttt{new}}$. The pseudocode for this function is given in Algorithm 20.

- `CTG` ($\mathit{self}, \boldsymbol{x}_2$): The input of this function is a vertex object $\boldsymbol{x}_2$. This function calculates the unconstrained optimal cost-to-go from the the state of vertex object (self) to the state of the vertex object $\boldsymbol{x}_2$, and returns this value as the double precision variable CTG. The pseudocode for this function is given in Algorithm 21.

- `Sample` (self): This function returns a randomly generated vertex object $\boldsymbol{x}_{\texttt{rand}}$ that has a state that lies in the region described by the bounds in the property $\mathcal{X}$. The pseudocode for this function is given in Algorithm 22.

- `CollisionCheck` (self, x): The input of this function is an n row double precision array x, where each column of x describes a state. This function evaluates if any column of x describes a state that lies outside of the bounds described by the property $\mathcal{X}$. If no state described by the columns of x lies outside of these bounds then the function returns *true*, and it returns *false* otherwise. The pseudocode for this function is given in Algorithm 23.

- `GetAffine` (self): This function returns the property c.

- `GetR` (self): This function returns the property R.

- `GetCi` (self): This function returns the property C_I.

- `SetState` (self, x): The input of this function is the n row single-column double precision array x, which is assigned to the property x.

Additionally, a function describing the ordinary differential equations of the system is required, which is detailed in Algorithm 18. The input t is a double precision variable describing the time of the system, x is a n row single-column double precision array describing the state of the system, t_f^* is a double precision variable describing the final time of the trajectory, x_0 is a n row single-column double precision array describing the initial state of the system, x_f is a n row single-column double precision array describing the final state of the system, c is a n row single-column double precision array describing the drift of the system, and R is the

control input weighting matrix in the cost function given by (26). The function is described by

$$[dxdt, u_t^*] = ODEFun\left(t, x, t_f^*, x_0, x_f, c, R\right).$$

Algorithm 18 Double Integrator with Drift: ODE Function

1: **function** ODEFUN$(t, x, t_f^*, x_0, x_f, c, R)$
2: $A = [\texttt{zeros}\,(3)\,, \texttt{eye}\,(3)\,; \texttt{zeros}\,(3)\,, \texttt{zeros}\,(3)]$;
3: $B = [\texttt{zeros}\,(3)\,, \texttt{eye}\,(3)]$;
4: $u_t^* = (64)$;
5: $dxdt = A * x + B * u + c$;
6: **return** $[dxdt, u_t^*]$;

Algorithm 19 Vertex Class: Steering Function

1: **function** STEER($self, \boldsymbol{x}_2$)
2: $x_0 = self.x$;
3: $x_f = \boldsymbol{x}_2$.GetState;
4: $c = self.c$;
5: $C_I = self.C_I$;
6: $R = self.R$;
7: H_4, H_2, H_1, H_0 from (235);
8: **if** $H_2 == 0$ && $H_1 == 0$ && $H_0 == 0$ **then**
9: $t_c = 0$;
10: **else if** $H_2 < 0$ && $H_1 == 0$ && $H_0 == 0$ **then**
11: $t_c = \texttt{sqrt}\left(\frac{-H_2}{H_4}\right)$;
12: $t_c = \left[0; t_f^*\right]$;
13: **else if** $H_2 == 0$ && $H_1 == 0$ && $H_0 < 0$ **then**
14: $t_c = \texttt{sqrt}\left(\texttt{sqrt}\left(\frac{-H_2}{H_4}\right)\right)$;
15: **else if** $H_2 < 0$ && $H_1 == 0$ && $H_0 < 0$ **then**
16: $t_c = \texttt{sqrt}\left(\frac{-H_2+\texttt{sqrt}\left(H_2^2-4*H_4*H_0\right)}{2*H_4}\right)$;
17: **else**
18: $t_c = roots\left([H_4, 0, H_2, H_1, H_0]\right)$;
19: $t_c = t_f^*\left(\texttt{imag}\left(t_f^*\right) == 0\right)$;
20: $t_c = t_f^*\left(t_f^* >= 0\right)$;
21: $c_{\texttt{min}} = \infty$;
22: **for** $i = 1 : \texttt{length}\left(t_c\right)$ **do**
23: $t_f^* = t_c\left(i\right)$;
24: $c_f = (65)$;
25: **if** $c_f < c_{\texttt{min}}$ **then**
26: $c_{\texttt{min}} = c_f$;
27: $t_{\texttt{min}} = t_f^*$
28: $c_{1,2}^* = c_{\texttt{min}}$;
29: $t_f^* = t_{\texttt{min}}$;
30: $\left[t_{1,2}^*, x_{1,2}^*\right] = \texttt{ode45}\left(@\left(t, x\right)\texttt{ODEFun}\left(t, x, t_f^*, x_0, x_f, c, R\right), \left[0, t_f^*\right], x_0\right)$;
31: $x_{1,2}^* = x_{1,2}^{*}$';
32: $u_{1,2}^* = []$;
33: **for** $i = 1 : \texttt{length}\left(t_{1,2}^*\right)$ **do**
34: $t = t_{1,2}^*\left(i\right)$;
35: $\left[\sim, u_{1,2}^*\left(:, \texttt{end}+1\right)\right] = \texttt{ODEFun}\left(t, x_{1,2}^*\left(:, i\right), t_f^*, x_0, x_f, c, R\right)$;
36: **return** $\texttt{Trajectory}\left(self, \boldsymbol{x}_2, x_{1,2}^*, u_{1,2}^*, t_{1,2}^*, c_{1,2}^*\right)$;

Algorithm 20 Vertex Class: Local Trajectory Function

function LocalTraj($self, \boldsymbol{e}, c_{\max}$)
  $\boldsymbol{x}_1 = \boldsymbol{e}$.GetStartVertex;
  $\boldsymbol{x}_2 = \boldsymbol{e}$.GetEndVertex;
  $c_e = \boldsymbol{e}$.GetCost;
  if $c_e <= c_{\max}$ **then**
    return $[\boldsymbol{x}_2, \boldsymbol{e}]$;
  $x_1 = \boldsymbol{x}_1$.GetState;
  $c = \boldsymbol{x}_1$.GetAffine;
  $C_I = \boldsymbol{x}_1$.GetCi;
  $R = \boldsymbol{x}_1$.GetR;
  $x_2 = \boldsymbol{x}_2$.GetState;
  C_3, C_2, C_1 from (79);
  $\tilde{t}_f = roots\left([C_3, C_2, C_1, c_{\max}]\right)$;
  $\tilde{t}_f = \tilde{t}_f\left(\text{imag}\left(\tilde{t}_f\right) == 0\right)$;
  $\tilde{t}_f = \tilde{t}_f\left(\tilde{t}_f >= 0\right)$;
  $\tilde{t}_f = \min\left(\tilde{t}_f\right)$;
  $t_f = \boldsymbol{e}$.GetTime;
  $t_f = t_f\left(\text{end}\right)$;
  $\left[t^*_{1,2}, x^*_{1,2}\right] = \text{ode45}\left(@\left(t, x\right) \text{ODEFun}\left(t, x, t_f, x_1, x_2, c, R\right), \left[0, \tilde{t}_f\right], x_1\right)$;
  $x^*_{1,2} = x^*_{1,2}$';
  $u^*_{1,2} = []$;
  for $i = 1 : \text{length}\left(t^*_{1,2}\right)$ **do**
    $t = t^*_{1,2}\left(i\right)$;
    $\left[\sim, u^*_{1,2}\left(:, \text{end}+1\right)\right] = \text{ODEFun}\left(t, x^*_{1,2}\left(:, i\right), \tilde{t}_f, x_1, x_2, c, R\right)$;
  $x_2 = x^*_{1,2}\left(:, \text{end}\right)$;
  $\boldsymbol{x}_2$.SetState$\left(x_2\right)$;
  $c^*_{1,2} = \boldsymbol{x}_1$.CTG$\left(\boldsymbol{x}_2\right)$;
  $c_1 = \boldsymbol{x}_1$.GetCost;
  $\boldsymbol{x}_2$.SetCost$\left(c_1 + c^*_{1,2}\right)$;
  $\boldsymbol{e}_{\text{new}} = \text{Trajectory}\left(\boldsymbol{x}_1, \boldsymbol{x}_2, x^*_{1,2}, u^*_{1,2}, t^*_{1,2}, c^*_{1,2}\right)$;
  return $[\boldsymbol{x}_2, \boldsymbol{e}_{\text{new}}]$;

Algorithm 21 Vertex Class: Cost-to-Go Function

function CTG($\texttt{self}, \boldsymbol{x}_2$)
  $x_0 = \texttt{self}.x$;
  $x_f = \boldsymbol{x}_2.\texttt{GetState}$;
  $c = \texttt{self}.c$;
  $C_I = \texttt{self}.C_I$;
  $R = \texttt{self}.R$;
  H_4, H_2, H_1, H_0 from (235);
  if $H_2 == 0$ && $H_1 == 0$ && $H_0 == 0$ **then**
    $t_c = 0$;
  else if $H_2 < 0$ && $H_1 == 0$ && $H_0 == 0$ **then**
    $t_c = \texttt{sqrt}\left(\frac{-H_2}{H_4}\right)$;
    $t_c = \left[0; t_f^*\right]$;
  else if $H_2 == 0$ && $H_1 == 0$ && $H_0 < 0$ **then**
    $t_c = \texttt{sqrt}\left(\texttt{sqrt}\left(\frac{-H_2}{H_4}\right)\right)$;
  else if $H_2 < 0$ && $H_1 == 0$ && $H_0 < 0$ **then**
    $t_c = \texttt{sqrt}\left(\frac{-H_2+\texttt{sqrt}\left(H_2^2-4*H_4*H_0\right)}{2*H_4}\right)$;
  else
    $t_c = roots\left([H_4, 0, H_2, H_1, H_0]\right)$;
    $t_c = t_f^*\left(\texttt{imag}\left(t_f^*\right) == 0\right)$;
    $t_c = t_f^*\left(t_f^* >= 0\right)$;
  $CTG = \infty$;
  for $i = 1 : \texttt{length}\left(t_c\right)$ **do**
    $t_f^* = t_c\left(i\right)$;
    $c_f = (65)$;
    if $c_f < CTG$ **then**
      $CTG = c_f$;
  return CTG;

Algorithm 22 Vertex Class: Sampling Function (Uniform Sampling)

function SAMPLE($\texttt{self}$)
  $\boldsymbol{\mathcal{X}}_p = \texttt{self}.\boldsymbol{\mathcal{X}}\left(1 : \texttt{end} - 1, :\right)$;
  $\boldsymbol{\mathcal{X}}_v = \texttt{self}.\boldsymbol{\mathcal{X}}\left(\texttt{end}, :\right)$;
  $p = \boldsymbol{\mathcal{X}}_p\left(:, 1\right) + \texttt{rand}\left(\texttt{size}\left(\boldsymbol{\mathcal{X}}_p, 1\right), 1\right) * \left(\boldsymbol{\mathcal{X}}_p\left(:, 2\right) - \boldsymbol{\mathcal{X}}_p\left(:, 1\right)\right)$;
  $\phi = \texttt{rand}\left(1, 1\right) * 2 * \pi$;
  $\theta = \texttt{rand}\left(1, 1\right) * 2 * \pi$;
  $r_v = \boldsymbol{\mathcal{X}}_v\left(1, 1\right) + \texttt{rand}\left(1, 1\right) * \left(\boldsymbol{\mathcal{X}}_v\left(1, 2\right) - \boldsymbol{\mathcal{X}}_v\left(1, 1\right)\right)$;
  $v = r_v * \left[\cos\left(\phi\right) * \sin\left(\theta\right); \sin\left(\phi\right) * \sin\left(\theta\right); \cos\left(\theta\right)\right]$;
  $x = [p; v]$;
  while $\sim \texttt{self}.\texttt{CollisionFree}\left(x\right)$ **do**
    $p = \boldsymbol{\mathcal{X}}_p\left(:, 1\right) + \texttt{rand}\left(\texttt{size}\left(\boldsymbol{\mathcal{X}}_p, 1\right), 1\right) * \left(\boldsymbol{\mathcal{X}}_p\left(:, 2\right) - \boldsymbol{\mathcal{X}}_p\left(:, 1\right)\right)$;
    $\phi = \texttt{rand}\left(1, 1\right) * 2 * \pi$;
    $\theta = \texttt{rand}\left(1, 1\right) * 2 * \pi$;
    $r_v = \boldsymbol{\mathcal{X}}_v\left(1, 1\right) + \texttt{rand}\left(1, 1\right) * \left(\boldsymbol{\mathcal{X}}_v\left(1, 2\right) - \boldsymbol{\mathcal{X}}_v\left(1, 1\right)\right)$;
    $v = r_v * \left[\cos\left(\phi\right) * \sin\left(\theta\right); \sin\left(\phi\right) * \sin\left(\theta\right); \cos\left(\theta\right)\right]$;
    $x = [p; v]$;
  return x;

Algorithm 23 Vertex Class: Collision Check Function (Uniform Sampling)

```
1: function CollisionCheck(self, x)
2:   for i = 1 : 3 do
3:     if any(x(i,:) < self.𝒳(i,1)) then
4:       return false;
5:     else if any(x(i,:) > self.𝒳(i,2)) then
6:       return false;
7:   v = vecnorm(x(4:6,:), 2, 1);
8:   if any(v < self.𝒳(4,1)) then
9:     return false;
10:  else if any(v > self.𝒳(4,2)) then
11:    return false;
12:  return true;
```

References

[1] J. Paulsen, "Only data at the edge will make driverless cars safe," *Seagate Blog*. [Online]. Available: https://blog.seagate.com/human/only-data-at-the-edge-will-make-driverless-cars-safe

[2] S. Baur, M. Hader, and T. Schönberg, "Cargo drones: the urban parcel delivery network of tomorrow," *Roland Berger*. [Online]. Available: https://www.rolandberger.com/sv/Point-of-View/Cargo-drones-The-urban-parcel-delivery-network-of-tomorrow.html

[3] A. Tabor, "NASA and Uber test system for future urban air transport," *NASA*. [Online]. Available: https://www.nasa.gov/feature/ames/nasa-and-uber-test-system-for-future-urban-air-transport

[4] CB Insights, "40 corporations working on autonomous vehicles," Mar 2020. [Online]. Available: https://www.cbinsights.com/research/autonomous-driverless-vehicles-corporations-list

[5] T. Reed and J. Kidd, "INRIX global traffic scorecard," 2019.

[6] J. Young, "US ecommerce sales grow 14.9% in 2019," 2020. [Online]. Available: https://www.digitalcommerce360.com/article/us-ecommerce-sales/#:~:text=Consumers%20spent%20%24601.75%20billion%20online,quarterly%20ecommerce%20figures%20released%20Wednesday

[7] M. Moore, "Uber elevate: eVTOL urban mobility," *Rotorcraft Business & Technology Summit*, 2017.

[8] Day One Staff, "Another new frontier for Prime Air," 2019. [Online]. Available: https://blog.aboutamazon.com/transportation/another-new-frontier-for-prime-air

[9] Airbus, "Airbus forecasts $3 trillion commercial aviation aftermarket services over the next 20 years," 2016. [Online]. Available: https://www.airbus.com/newsroom/press-releases/en/2016/07/airbus-forecasts-3-trillion-commercial-aviation-aftermarket-services-over-the-next-20-years.html

[10] IATA, "Recovery delayed as international travel remains locked down," Jul 2020. [Online]. Available: https://www.iata.org/en/pressroom/pr/2020-07-28-02/

[11] S. Barden, "AINsight: is the pilot shortage over?" Sep 2020. [Online]. Available: https://www.ainonline.com/aviation-news/blogs/ainsight-pilot-shortage-over

[12] Popular Science Monthly, "Now - the automatic pilot," 1930.

[13] A. E. Bryson, *Applied optimal control: optimization, estimation and control*. CRC Press, 2018.

[14] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[15] Y. Chen, G. Luo, Y. Mei, J. Yu, and X. Su, "UAV path planning using artificial potential field method updated by optimal control theory," *International Journal of Systems Science*, vol. 47, no. 6, pp. 1407–1420, 2016, DOI 10.1080/00207721.2014.929191.

[16] J. Tisdale, Z. Kim, and J. K. Hedrick, "Autonomous UAV path planning and estimation," *IEEE Robotics Automation Magazine*, vol. 16, no. 2, pp. 35–42, 2009, DOI 10.1109/MRA.2009.932529.

[17] M. Iwamura, M. Yamamoto, and A. Mohri, "A gradient-based approach to collision-free quasi-optimal trajectory planning of nonholonomic systems," in *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No.00CH37113)*, vol. 3, 2000, pp. 1734–1740 vol.3, DOI 10.1109/IROS.2000.895222.

[18] M. Bagherian and A. Alos, "3D UAV trajectory planning using evolutionary algorithms: a comparison study," *The Aeronautical Journal (1968)*, vol. 119, no. 1220, p. 1271–1285, 2015, DOI 10.1017/S0001924000011246.

[19] E. Shintaku, "Minimum energy trajectory for an underwater manipulator and its simple planning method by using a genetic algorithm," *Advanced robotics*, vol. 13, no. 2, pp. 115–138, 1998.

[20] T. Chettibi, "Synthesis of dynamic motions for robotic manipulators with geometric path constraints," *Mechatronics*, vol. 16, no. 9, pp. 547 – 563, 2006. [Online]. Available: https://doi.org/10.1016/j.mechatronics.2006.03.012

[21] A. Ghanbari and S. Noorani, "Optimal trajectory planning for design of a crawling gait in a robot using genetic algorithm," *International Journal of Advanced Robotic Systems*, vol. 8, no. 1, p. 6, 2011, DOI 10.5772/10526.

[22] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968, DOI 10.1109/TSSC.1968.300136.

[23] S. Koenig and M. Likhachev, "Dˆ* lite," *AAAI/IAAI*, vol. 15, 2002. [Online]. Available: https://aaai.org/Papers/AAAI/2002/AAAI02-072.pdf

[24] B. Lau, C. Sprunk, and W. Burgard, "Kinodynamic motion planning for mobile robots using splines," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 2427–2433, DOI 10.1109/IROS.2009.5354805.

[25] S. Liu and D. Sun, "Minimizing energy consumption of wheeled mobile robots via optimal motion planning," *IEEE/ASME Transactions on Mechatronics*, vol. 19, no. 2, pp. 401–411, 2014, DOI 10.1109/TMECH.2013.2241777.

[26] D. Jung and P. Tsiotras, "On-line path generation for unmanned aerial vehicles using B-spline path templates," *Journal of Guidance, Control, and Dynamics*, vol. 36, no. 6, pp. 1642–1653, 2013, DOI 10.2514/1.60780.

[27] Z. Wang, X. Xiang, J. Yang, and S. Yang, "Composite Astar and B-spline algorithm for path planning of autonomous underwater vehicle," in *2017 IEEE 7th International Conference on Underwater System Technology: Theory and Applications (USYS)*, 2017, pp. 1–6, DOI 10.1109/USYS.2017.8309463.

[28] M. Wang, J. Luo, and U. Walter, "Trajectory planning of free-floating space robot using particle swarm optimization (PSO)," *Acta Astronautica*, vol. 112, pp. 77 – 88, 2015. [Online]. Available: https://doi.org/10.1016/j.actaastro.2015.03.008

[29] B. Song, Z. Wang, L. Zou, L. Xu, and F. E. Alsaadi, "A new approach to smooth global path planning of mobile robots with kinematic constraints," *International Journal of Machine Learning and Cybernetics*, vol. 10, no. 1, pp. 107–119, 2019. [Online]. Available: https://doi.org/10.1007/s13042-017-0703-7

[30] M. Pontani and B. A. Conway, "Particle swarm optimization applied to space trajectories," *Journal of Guidance, Control, and Dynamics*, vol. 33, no. 5, pp. 1429–1441, 2010, DOI 10.2514/1.48475.

[31] M. Saska, M. Macas, L. Preucil, and L. Lhotska, "Robot path planning using particle swarm optimization of Ferguson splines," in *2006 IEEE Conference on Emerging Technologies and Factory Automation*, 2006, pp. 833–839, DOI 10.1109/ETFA.2006.355416.

[32] A. Rahimi, K. Dev Kumar, and H. Alighanbari, "Particle swarm optimization applied to spacecraft reentry trajectory," *Journal of Guidance, Control, and Dynamics*, vol. 36, no. 1, pp. 307–310, 2013, DOI 10.2514/1.56387.

[33] B. Geiger and J. Horn, "Neural network based trajectory optimization for unmanned aerial vehicles," in *47th AIAA Aerospace Sciences Meeting including The New Horizons Forum and Aerospace Exposition*, DOI 10.2514/6.2009-54.

[34] J. F. Horn, E. M. Schmidt, B. R. Geiger, and M. P. DeAngelo, "Neural network-based trajectory optimization for unmanned aerial vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 35, no. 2, pp. 548–562, 2012, DOI 10.2514/1.53889.

[35] S. A. Gautam and N. Verma, "Path planning for unmanned aerial vehicle based on genetic algorithm & artificial neural network in 3D," in *2014 International Conference on Data Mining and Intelligent Computing (ICDMIC)*, 2014, pp. 1–5, DOI 10.1109/ICDMIC.2014.6954257.

[36] H. Eslamiat, Y. Li, N. Wang, A. K. Sanyal, and Q. Qiu, "Autonomous waypoint planning, optimal trajectory generation and nonlinear tracking control for multi-rotor UAVs," in *2019 18th European Control Conference (ECC)*, 2019, pp. 2695–2700, DOI 10.23919/ECC.2019.8795855.

[37] R. Wai and A. S. Prasetia, "Adaptive neural network control and optimal path planning of UAV surveillance system with energy consumption prediction," *IEEE Access*, vol. 7, pp. 126 137–126 153, 2019, DOI 10.1109/ACCESS.2019.2938273.

[38] L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996, DOI 10.1109/70.508439.

[39] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959. [Online]. Available: https://doi.org/10.1007/BF01386390

[40] S. M. LaValle and J. James J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001, DOI 10.1177/02783640122067453.

[41] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Proceedings of Robotics: Science and Systems*, Zaragoza, Spain, June 2010, DOI 10.15607/RSS.2010.VI.034.

[42] ——, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011, DOI 10.1177/0278364911406761.

[43] ——, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *49th IEEE Conference on Decision and Control (CDC)*, 2010, pp. 7681–7687, DOI 10.1109/CDC.2010.5717430.

[44] G. Goretkin, A. Perez, R. Platt, and G. Konidaris, "Optimal sampling-based planning for linear-quadratic kinodynamic systems," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 2429–2436, DOI 10.1109/ICRA.2013.6630907.

[45] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez, "LQR-RRT*: optimal sampling-based motion planning with automatically derived extension heuristics," in *2012 IEEE International Conference on Robotics and Automation*, 2012, pp. 2537–2542, DOI 10.1109/ICRA.2012.6225177.

[46] E. Schmerling, L. Janson, and M. Pavone, "Optimal sampling-based motion planning under differential constraints: the drift case with linear affine dynamics," in *2015 54th IEEE Conference on Decision and Control (CDC)*, 2015, pp. 2574–2581, DOI 10.1109/CDC.2015.7402604.

[47] D. J. Webb and J. v. d. Berg, "Kinodynamic RRT*: optimal motion planning for systems with linear differential constraints," *arXiv preprint arXiv:1205.5088*, 2012.

[48] E. Glassman and R. Tedrake, "A quadratic regulator-based heuristic for rapidly exploring state space," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 5021–5028, DOI 10.1109/ROBOT.2010.5509718.

[49] D. B. Moses and G. Anitha, "Goal directed approach to autonomous motion planning for unmanned vehicles," *Defence Science Journal*, vol. 67, no. 1, pp. 45–49, Dec. 2016, DOI 10.14429/dsj.67.10295.

[50] I. Garcia and J. P. How, "Improving the efficiency of rapidly-exploring random trees using a potential function planner," in *Proceedings of the 44th IEEE Conference on Decision and Control*, 2005, pp. 7965–7970, DOI 10.1109/CDC.2005.1583450.

[51] A. H. Qureshi and Y. Ayaz, "Potential functions based sampling heuristic for optimal path planning," *Autonomous Robots*, vol. 40, no. 6, pp. 1079–1093, 2016. [Online]. Available: https://doi.org/10.1007/s10514-015-9518-0

[52] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, vol. 2, 2003, pp. 1178–1183 vol.2, DOI 10.1109/IROS.2003.1248805.

[53] J. Denny, M. Morales, S. Rodriguez, and N. M. Amato, "Adapting RRT growth for heterogeneous environments," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2013, pp. 1772–1778, DOI 10.1109/IROS.2013.6696589.

[54] V. Vonásek, J. Faigl, T. Krajník, and L. Přeučil, "RRT-Path – a guided rapidly exploring random tree," in *Robot Motion and Control 2009*. Springer, 2009, pp. 307–316.

[55] L. Palmieri, S. Koenig, and K. O. Arras, "RRT-based nonholonomic motion planning using any-angle path biasing," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, 2016, pp. 2775–2781, DOI 10.1109/ICRA.2016.7487439.

[56] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2014, pp. 2997–3004, DOI 10.1109/IROS.2014.6942976.

[57] J. J. Kuffner and S. M. LaValle, "RRT-Connect: an efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 995–1001, DOI 10.1109/ROBOT.2000.844730.

[58] R. Tedrake, "LQR-Trees: feedback motion planning on sparse randomized trees," 2009, DOI 10.15607/RSS.2009.V.003.

[59] L. Rodrigues, "On affine quadratic optimal control and aerospace applications," *IEEE Transactions on Aerospace & Electronic Systems*, in press for publication.

[60] M. Lichocki and L. Rodrigues, "A Gaussian-biased heuristic for stochastic sampling-based 2D trajectory planning algorithms," in *2020 European Control Conference (ECC)*, 2020, pp. 1949–1954, DOI 10.23919/ECC51009.2020.9143947.

[61] L. S. Pontryagin, *Mathematical theory of optimal processes*. Routledge, 2018.

[62] D. E. Kirk, *Optimal control theory: an introduction*. Courier Corporation, 2004.

[63] M. Mesbahi and M. Egerstedt, *Graph theoretic methods in multiagent networks*. Princeton University Press, 2010, vol. 33.

[64] L. Liu and M. T. Özsu, *Encyclopedia of database systems*. Springer New York, NY, USA:, 2009, vol. 6.

[65] MATLAB, *version 9.4.0 (R2018a)*. Natick, Massachusetts: The MathWorks Inc., 2018.

[66] W. Yuan and L. Rodrigues, "Onboard generation of optimal flight trajectory for delivery of fragile packages," in *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2019, pp. 1–8, DOI 10.1109/ICUAS.2019.8798130.

[67] C. de Paiva, B. Carvalho, and L. Rodrigues, "UAV optimal guidance in wind fields using ZEM/ZEV with generalized performance index," *IEEE Transactions on Aerospace and Electronic Systems*, 2020, DOI 10.1109/TAES.2020.3005302.

[68] R. H. Byrd, J. C. Gilbert, and J. Nocedal, "A trust region method based on interior point techniques for nonlinear programming," *Mathematical programming*, vol. 89, no. 1, pp. 149–185, 2000. [Online]. Available: https://doi.org/10.1007/PL00011391

[69] A. Edelman and H. Murakami, "Polynomial roots from companion matrix eigenvalues," *Mathematics of Computation*, vol. 64, no. 210, pp. 763–776, 1995. [Online]. Available: https://doi.org/10.1090/S0025-5718-1995-1262279-2

[70] X. Wang, "A simple proof of Descartes's rule of signs," *The American Mathematical Monthly*, vol. 111, no. 6, pp. 525–526, 2004, DOI 10.2307/4145072.

[71] A. G. Kurosh, *Higher algebra*. Mir Publishers, 1972.

[72] R. S. Irving, *Integers, polynomials, and rings: a course in algebra*. Springer Science & Business Media, 2003.

[73] R. W. Nickalls, "A new approach to solving the cubic: Cardan's solution revealed," *The Mathematical Gazette*, vol. 77, no. 480, pp. 354–359, 1993.

[74] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors," *Matrix*, vol. 58, no. 15-16, pp. 1–35, 2006.

[75] J. Wilson, "Volume of n-dimensional ellipsoid," *Sciencia Acta Xaveriana*, vol. 1, no. 1, pp. 101–106, 2010.

[76] K. Schacke, "On the kronecker product," *Master's thesis, University of Waterloo*, 2004.