People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University M'Hamed BOUGARA – Boumerdès

Institute of Electrical and Electronic Engineering
Department of Electronics

Project Report Presented in Partial Fulfilment of
the Requirements of the Degree of

'MASTER'

In Electrical and Electronic Engineering

Option:

CONTROL

Title:

Feedback Motion Planning with Simulation Based LQR-Trees

Presented by:

- **CHADLI KOUIDER**

Supervisor:

-Dr. Guernane Reda

Registration Number: 161638044507 2020/2021

Abstract

In autonomous and non-autonomous systems, a motion planner generates reference trajectories which are tracked by a low-level controller. In this report we consider the problem of generating a feedback motion planning algorithm for a nonlinear dynamical systems; the algorithm computes the stability regions to build a set of LQR-stabilized trajectories by generating a feedback control law from a set of initial conditions that are goal reachable. Furthermore, we consider the case where these plans must be generated offline, because the LQR trees lack the ability to handle events in which the goal and environments are unknown till run-time. Moreover, the algorithm approximates the funnel [2] of a trajectory using the one step Lyapunov method which is a sampling-based approach, generating a control law that stabilizes the bounded set to the goal is equivalent to adding trajectories to the tree until their funnels cover the design set. We further validate our approach by carefully evaluating the guarantees on invariance provided by funnels on nonlinear systems. We demonstrate and validate our method using simulation experiments of some nonlinear models. These demonstrations constitute examples of provably safe and robust control for robotic systems with complex nonlinear dynamics with Obstacles and dynamic constraints.

Acknowledgements

This document is the result of my research during my graduation project. Having a sincere interest in control techniques and robotics I decided to perform my research under the guidance of Dr. Guernane Reda. He suggested we worked over Feedback motion planning using LQR controller. I was both challenged and fascinated to work on a topic which is relatively new and thus less explored. I initially explored the potential of planning using both cell decomposition and potential field approach, by imposing simultaneous stabilization of multiple models. However, such approaches lead to computationally complex algorithms, and for that I decided to approach the problem with sampling-based planning techniques (PRM, RRT, RRT*…). During the period that I worked in this project the feedback from my supervisor Dr. Guernane Reda kept me motivated to keep up with my research and I am very thankful for that. Finally, I would like to thank my parents and siblings for their unconditional love over the years. Without them, none of this would be possible.

Contents

List of Figures

List of Acronyms

PRM Probabilistic Road Map

RRT Rapid Random-exploring Tree

NLP Non-Linear Programming

DC Direct Collocation

TI Time invariant

TV Time Varying

LQR Linear Quadratic Regulator

SOS Sum of Squares

SS State Space

C-Space Configuration space

HJB Hamilton–Jacobi–Bellman equation

List of Algorithms

Introduction

Overview

Over the recent decades, the field of robotics witnessed a great leap in technological advancement and degree of autonomy; Autonomous robots can be simply defined as intelligent machines that are capable of performing certain tasks in their environment without explicit human interaction, lowering the operation costs and complexity.

Some of the most significant challenges confronting autonomous robotics lie in the area of automatic motion planning. The goal is to be able to specify a task in high-level language and have the robot automatically compile this specification into a set of low-level motion primitives, or feedback controller. The task is accomplished if a trajectory for a robot is found from one configuration to another while avoiding obstacles and potentially in an efficient manner, whether it's a mobile robot, robot arm or even a flying robot.

Since the action in the real world are subjected to physical laws, uncertainties and constraints, the mathematical representation of the system no matter how reliable, can only be regarded as an approximation of a complex underlying behavior. Thus, the inability to take into account uncertainties can have disastrous consequences.

Every robot that operates in an environment requires some ability to plan motions in order to interact with the world. Depending on the system in use, the application at hand, and the computational tools available, there are various ways the motion planning problems can be formulated and solved. In practice they are all equipped with feedback control to help with the deviation from the planned trajectory.

The planning algorithms used in practical motion planning can be classified into: grid-based planning algorithms like cell decomposition algorithms which are suitable for systems with low dimensional state space (5 to 6 dimension at most) or with decoupled systems, however the limitations of these algorithms lies in their computational and time complexities that grow exponentially as the degree of freedom increases since.

On the other hand, sampling-based algorithms like probabilistic roadmap (PRM) and rapidly-exploring randomized tree (RRT)[1] and (RRT*) can handled large state-space dimensions and complex constraints, these latter are applicable in both offline and online motion planning applications. However, their limitation is their inability to detect uncertainties and disturbances which can lead to failure if the system is deviated from its nominal trajectory.

In this report, we used LQR-Trees algorithm that is used to explores the bounded set with random state samples and, where needed, adds new trajectories to the tree using motion planning. Simultaneously, the algorithm approximates the funnel of a trajectory, which is the set of states that can be stabilized to the goal by the trajectory's feedback law. Generating a control signal that stabilizes the bounded set to the goal is equivalent to adding trajectories to the tree until their funnels cover the set. Which means this algorithm create a tree of stabilizing controllers which takes any initial condition in some design set to the goal.

Problem statement

In order to properly define the problem, some notation needs an introduction. Let $X \subseteq \mathbb{R}^n$ be the state space of dimensionality n, where the elements of X are known as states x, and $\mathbb{U} \subseteq \mathbb{R}^m$be the control space of dimensionality m, the state space where the robot is in collision with the obstacles is denominated X_{obs} while the rest is the free space X_{free} . A trajectory$v \in X$ is a continuous function $v(t) = (x, t)$ connecting two states. It is said to be free if and only if $v \in X_{free}$ and feasible if in addition it satisfies the dynamic and actuation constraints.

The motion planning problem is stated as: Given an initial state x_{init} at t_0, an environment with obstacles X_{obs} and a goal state x_{goal}, the motion planning problem deals with finding a feasible collision-free trajectory connecting the initial state to the goal state. The strict goal state condition can be instead relaxed to a goal region, denoted as X_{goal}.

The feasibility of the trajectory$v$ is determined by not only residing in the free space but also governed by the non-linear state space dynamics described by:

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

The above is the usual formulation of the motion planning problem which is concerned only with finding a feasible trajectory. However, for many applications such as the one this report is aimed at, an optimal trajectory is preferred. If the set of trajectories is denoted as P, then a cost function J where $J: \mathrm{P} \to \mathbb{R}^+$ is a function that maps the candidates' free trajectories to a non-negative scalar cost. The optimal motion planning problem deals with finding a trajectory v such that $J(v^*) = \min_{v \in P} J(v)$. The time sequence input u propagating eq (1) forward in time is called the control policy denoted as π. The optimal control policy is the sequence yielding the minimal cost value with respect to the defined cost functional:

$$\pi^* = \underset{\pi}{\operatorname{argmin}} J(x, x_{int}, X_{goal}) \tag{2}$$

Contributions:

In this report, the following contributions are proposed:

1. Incorporate the nonlinear programming (Direct collocation method) to the steering procedure of the RRT.
2. Generalizing the funnels concept to environments with obstacles using a sampling based-method.
3. demonstrate and validate the approach using thorough simulation experiments of a cart pole system with state constraints (4 dimensional system) and a quadrotor model navigating through cluttered environments(6 dimensional system).

Report Outline

This report consists of 3 chapters, organized as follow:

- **Chapter 1:** it gives a brief explanation on motion planners and LQR controller.
- **Chapter 2:** It addresses the main idea behind the Planning in LQR-Trees along with an extension in the RRT and a proposed algorithm in adjusting the funnels in an environment with obstacles.
- **Chapter 3:** presents extensive simulation results on a cart-pole model and also considers a quadrotor model and shows how one can use our approach to guarantee collision-free flight in certain environments.

Chapter 1

Background and Literature Review

This chapter is devoted to classifying the classes of motion planning algorithms as well as the basic theory behind the linear quadratic regulator (LQR). Thus it is convenient to first define them here.

1.1 Basic definitions

Definition 1: The **workspace** is the environment where the robot and obstacle live in. for instance an Autonomous vehicles navigate on a 3-dimensional surface that can be locally approximated as a flat 2-dimensional plane. Therefore, the workspace can be described as W=$\mathbb{R}^2$.Obstacles within this workspace is defined as the obstacle region $O \subset W$ and the region that the vehicle (Agent) occupies as $A \subset W$[1].

Definition 2: The **configuration space** C is the set of all possible configurations of the robot. For instance, a 2-D dimensional pose of a moving vehicle (x, y, θ) is often considered as configuration where (x, y) is the location of the vehicle and θ is its orientation. Thus the number of degree of the robot is the dimension of the C-space. Furthermore, the configuration space (C-space) is the combination of two important subspaces which are the free configuration space (or free space C_{free}) and configuration space obstacle C_{Obs} ,thus the C-space is defined as $C = C_{free} \cup C_{Obs}$ and$C_{free} \cap C_{Obs} = \phi$.[1]

Definition 3: The **state space** is an extension of the C-space, it must be used when the problem is time varying, thus the state space is defined as $X = C \times T$.[11]

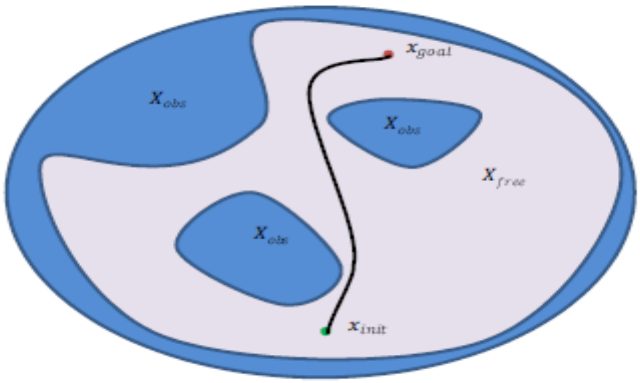


Figure 1-1: Simple motion planning problem

Definition 4: The **complexity** of a Motion Planning algorithm can be described by an upper and lower bound. The upper bound can be established by evaluating the run-time of an implementation. The lower bound is the minimum theoretical complexity that a class of algorithms can have [11]. This gives a good prior estimate of the difficulty of the problem to be solved. Several classes of complexity exist, the most known are:

- **P:** can be solved in polynomial time.
- **NP:** can be solved in polynomial time by a nondeterministic Turing machine.

In this report the problem is NP-hard. Hence, no exact solution exists for the problem that remains to be solved during this report.

Definition 5: Properties of motion planning

- **Optimality:** Return a feasible trajectory that optimizes performance in finite time [9].
- **Completeness:** A motion planning algorithm is complete if for any input it correctly reports whether a solution exists in a finite amount of time. This solution must also be returned within a finite amount of time [3].
- **Asymptotic Optimality**: The algorithm returns a sequence of solutions that converge to the optimal solution[3]

1.2 Transformation from workspace to C-space or SS

The transformation of obstacles from a robot's workspace into a robot state space is, naturally, strongly related to the inverse kinematics of the robotic mechanism. In practice it can be quite challenging to work with a robot that has complex shape, because additional parameters are needed to represent it in the state space. In order to solve this dilemma, we try to convert it to a point where it is easier to work with however by doing so the obstacle regions will get their shapes deformed in the state space or C-space compared to the workspace.

The C-space or state space (SS) construction can be categorized into two different methods: Geometry-based method and topology-based method. Geometry-based methods compute the exact geometric representation of the configuration space, while topology-based methods capture the connectivity of the configuration space.

Geometry-based methods also referred to as combinatorial approach are usually limited to low dimensional configuration spaces, due to the combinatorial complexity involved in computing C-space obstacle for high dimensional configuration spaces.

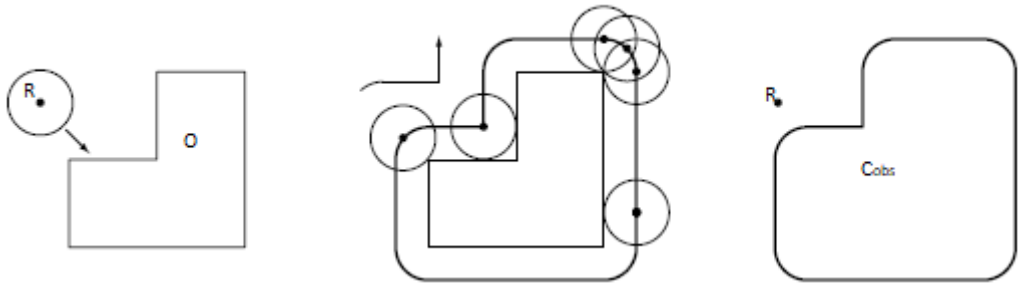
Topology-based methods or sampling-based approach captures the connectivity of the configuration space, meaning the connection between the nodes in the free configuration space. The basic idea is first to generate random samples(called milestones) in C_{free} and then organize these samples using a graph structure or a forest of tree structures .Topology-based methods can compute an approximate C-space representation much faster than geometry-

based methods. However, these methods do not work well with narrow passages and can be slow for high-DOF robots.

A very popular geometry-based method that's also used in this report is the Minkowski difference defined as:

$$X \ominus Y = \{x - y \in \mathbb{R}^n | x \in X \; and \; y \in Y\} \; where \; X, Y \subset \mathbb{R}^n$$

where in our case the configuration space obstacle case we written as $C_{Obs} = \mathcal{O} \ominus R$ where R is dimensions of the robot and $\mathcal{O}$ is the obstacle region.



(a) (b) (c)

Figure 1-2: (a) the workspace representation of the robot(circle) and the obstacle. (b) performing a convolution-like operation between the robot and obstacle region. (c) the spanned C-space.

1.3 Sampling-Based motion planning

The sampling-based motion planning conduct a search that explores the configuration space (or state space) with a sampling scheme in order to avoid the explicit construction of the configuration space obstacle. Both PRM and RRT (including its variants) have proven to be probabilistically complete. Moreover, they utilize the configuration space C (or state space X) and a metric to solve motion planning problems. The metric used is a measure of proximity required to define state neighborhoods and quantify the cost of edges connecting states within the space. The PRM offers several routes to the goal in a single query and therefore deals well with wide open spaces. But it requires a search algorithm to find an optimal trajectory. Whereas the RRTs are a better fit to the motion planning problems where building a roadmap a-priori may be irrelevant or simply infeasible, due to its single query nature. This allows for more options to consider for the metric and the tree edge connections. Although The RRT is efficient in finding an initial solution, but it may contain unnecessary detours which makes it far from optimal.

Sampling-based methods can be divided into two categories:

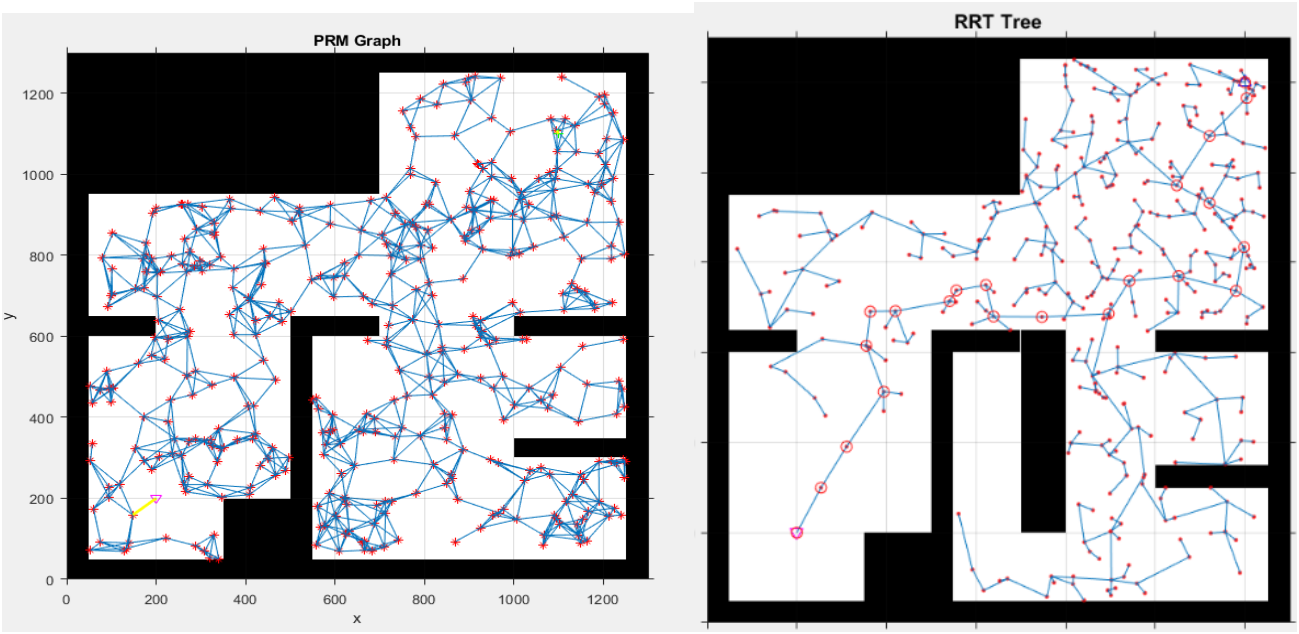
- Multi-query: here multiple start and goal configurations can be connected without reconstructing the graph structure.

- Single-query: where the tree is built from scratch for every set of start and goal configurations.

A summary of the two categories is shown in table 1-1

	MULTI-QUERY	SINGLE-QUERY
PHASE	1-road construction 2-searching	1-Tree construction 2-searching
TYPICAL ALGORITHMS	Probabilistic roadmap (PRM)	Rapid-Exploring Random Tree(RRT and it variants)
ADVANTAGES	Fast searching	No preprocessing
DISADVANTAGES	Cannot deal with dynamic environments	No memory

Table 1-1: The comparison between multi-query and single-query method



(a) (b)

Figure 1-3 :(a) an example of a roadmap for a point robot in a two-dimensional Euclidean space. (bt) an example of a Tree for a point robot in a two-dimensional Euclidean space. The black areas are the obstacle space; the red point corresponds to the sampled states.

1.4 Rapid-Exploring Random Tree (RRT)

Initially presented by Kuffner and LaValle [1], RRT is an incremental single-query method, as well as the most prevalent algorithm. Unlike PRM, RRT can be applied to nonholonomic and kinodynamic planning. Starting from an initial configuration, RRT constructs a tree using random sampling. The tree expands until a predefined time period expires or a fixed number of iterations are executed.

The RRT involves five main components and each can be encapsulated into a function or procedure as follows:

a) **Sampling function** when called, the sampling procedure returns a sampled state $x \in X_{free}$ from a random distribution.
b) **Steering function** when called, the function takes the sampled state and a set of states in the tree, then the function attempts to connect the sampled state with one of the states in the tree such that the constraints are not violated. It returns either a Boolean value indicating the two states are to be connected or a trajectory between the nearest state and not necessarily the sampled state as shown in figure 1-4.
c) **Collision check function** given a trajectory between $\upsilon(t)$ or a state x then this returns a Boolean value where it's true if the trajectory $\upsilon(t)$ or state x is in the free space in its entirety.

d) **Cost function** this procedure evaluates the cost to go from one state to another. If the two states are not connectible, the metric returns an infinite distance**.**
e) **K Nearest Neighbors (KNN) function** given a sampled state x, then the KNN function returns a state that's connected to the tree x_{near} and is the nearest neighbor of x. Meaning x_{near} hold for the following inequality $J(x, x_{near}) \leq J(x, x_i)$ for all $i = 1,2, \ldots \ldots, k$ where k is the number of states in the tree and $x_i \neq x_{near}$

The building of an RRT can be summarized in the following algorithm:

Algorithm 1: Generate Rapid-Exploring Random Tree

DATA:

x: A variable consisting of positions and velocities

$\mathcal{T}$: Tree (Output)

u: robot input

v: the trajectory between the two states

$condition$: Ending or terminating condition Δt: time step

1: $\mathcal{T}.init(x_{init})$
2: **While** $condition$**do**
3: $x_{rand} \leftarrow Random\_State()$
4: $x_{near} \leftarrow KNN(x_{rand}, \mathcal{T})$
5: $u \leftarrow Random\_input()$
6: $[x_{new}, v] \leftarrow New\_State(x_{near}, u, \Delta t)$
7: **If** CollisionFree(v, X_{free})**do**
8: $\mathcal{T}.add\_node(x_{new})$
9: $\mathcal{T}.add\_edge(v)$
10: **End if**
11: **End While**

$\mathcal{T}.init(x_{init})$ creates the tree and adds the start state as a root node to the tree. The next step is to grow the tree. This is done until the termination $condition$ is reached, it might be a time limit or reaching a state close to the goal. The first step in expanding the tree is to choose a random state. In practice the goal state can also be used with some probability as the random state in order to speed up the planning process. This probability is the goal bias; it was proven experimentally that 5-10% is the right choice [5]. Then the closest node to this state is selected for expansion. This closest node is determined using some defined metric. The tree will then expand from this closest node. It will do so by selecting a random viable input and applying this input for duration of time. The state reached through this applied input will then be added to the tree, as well as the motion or edge used to reach this state. At this point the planner either reaches a $Condition$ or a new random state is chosen. Using a random control input is probabilistically complete for extensive time durations. It is, however in practice undesired to let a motion planner run for an extended duration of time. A basic idea to speed up the motion planning process is to use multiple random inputs and then using the cost function to

determine the best newfound state. This, however, makes it so that the planning method is potentially no longer probabilistically complete.

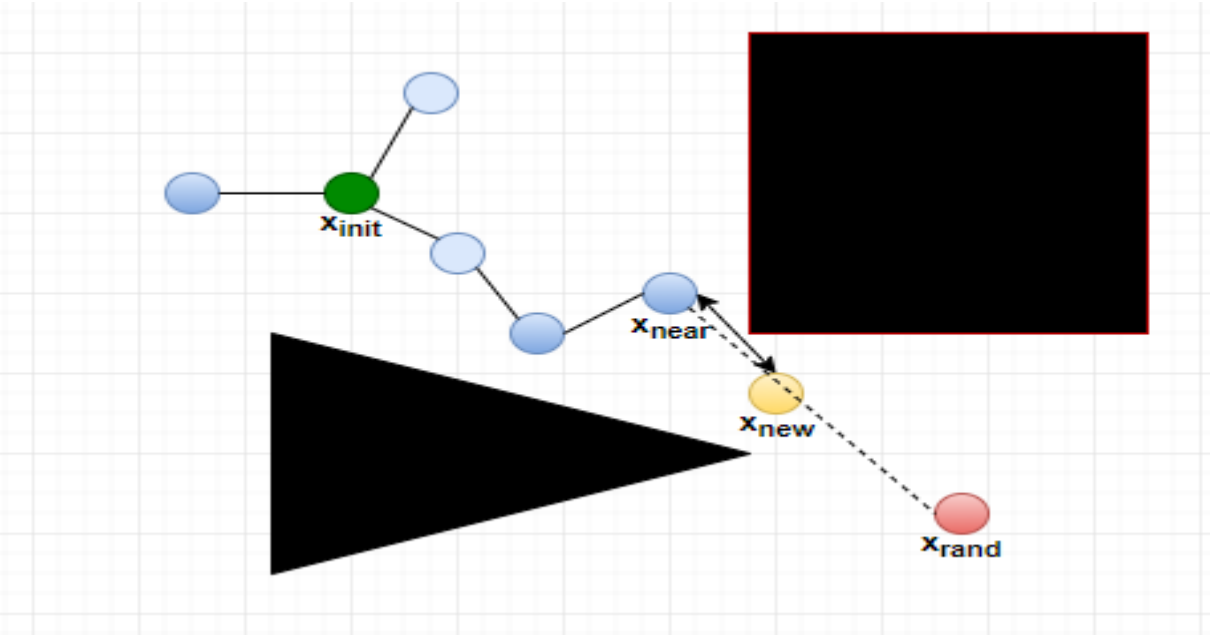


Figure 1-4: Adding a new state to RRT, the state x_{rand} is selected randomly from a uniform distribution in X_{free} . the state x_{near} is the closest state in the tree to it. The state x_{new} is obtained by moving x_{near} with u. Thus only x_{new} is added to the tree.

1.5 Collision Detection Module

Once it has been decided where the samples will be placed, the next problem is to determine whether the configuration is in collision. Thus, collision detection is a critical component of sampling-based planning. Even though it is often treated as a black box, it is important to study its inner workings to understand the information it provides and its associated computational cost. A variety of collision detection algorithms exist, ranging from theoretical algorithms that have excellent computational complexity to heuristic, practical algorithms whose performance is tailored to a particular application. The collision detection algorithm developed in this report relies on sampling a linear segment between two configurations then treat each sample in between as Boolean value {true, false} , if the sample is in the free space return true, otherwise return false.

Henceforth, the whole transition from the initial configuration to the next is removed if at least one of the sampled configurations in between the two is in collision.

1.6 Linear Quadratic Regulator (LQR)

For the majority of systems, open-loop trajectories are almost always unstable, so once given a trajectory from the RRT or any other method; one must chose a feedback solution to ensure that the system will stay near the given path. There are a number of options for this feedback,

including model predictive control (MPC) and time-varying linear quadratic regulators (TVLQR). We further note that in some cases, we can provide stability guarantees on these systems (using funnels), under the assumption of the correctness of our models.

With the Linear Quadratic Regulator (LQR) we enter the class of model-based controllers. The LQR controller is a special case of general optimization based controllers, exploiting the required linearity of the model and the quadratic form of the objective function. Depending on the task, the LQR can be formulated for continuous- or discrete-time and finite or infinite horizons.

Structure:

The controller structure forms around the central optimization problem,

$$\begin{cases} \min\limits_{x(\cdot),u(\cdot)} \sum_{i=0}^{N} x_i Q x_i + u_i R u_i \\ Given\ \ Q = Q^T \geq 0\ , R = R^T > 0 \\ subject\ to\ \boldsymbol{x}_{i+1} = A\boldsymbol{x}_i + B\boldsymbol{u}_i \qquad\qquad i = 0,1, \ldots\ldots, N-1 \end{cases} \tag{1.1}$$

Where > and $\geq$ denote the positive definite (PD) and positive semi-definite (PSD) of the matrices respectively. And A, B are the Jacobians then discretization the of the non-linear dynamic model in (1). Minimizing the quadratic cost subject to the initial value of the state and the model dynamics. The relevant tunable parameters are the weighting matrices Q, R. For the used single input model, R is a scalar representing the cost of deviating the controls from the neutral position. It may be chosen just small enough to represent the real-world cost, in this case the delay of the control surface. The matrix Q incorporates penalties of the state deviating from the zero state (i.e. the chosen reference point of the model).

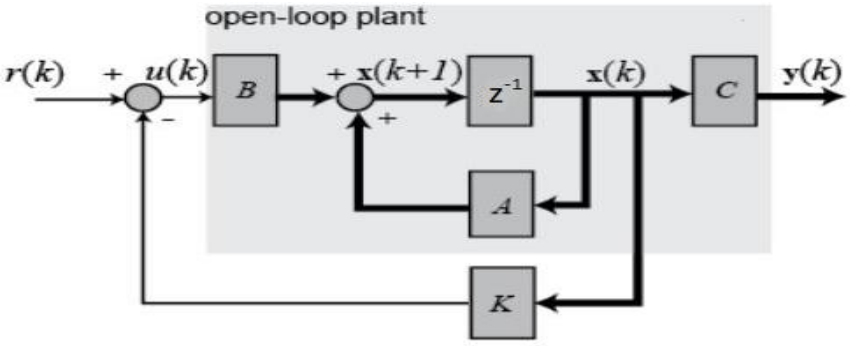


Figure 1-5: The closed loop LQR system

Our goal is to find the optimal cost-to-go function $\boldsymbol{J}^*(x)$ which satisfies the HJB:

$$\forall \boldsymbol{x} \quad 0 = \min_{\boldsymbol{u}} \left[\boldsymbol{x}^T Q \boldsymbol{x} + \boldsymbol{u}^T R \boldsymbol{u} + \frac{\partial J^*}{\partial x} (A\boldsymbol{x} + B\boldsymbol{u}) \right] \tag{1.2}$$

It is well known that for this problem the optimal cost-to-go function is quadratic. This is easy to verify. Let us choose the form

$$\boldsymbol{J^*(x) = x^T S x}\,, \quad \boldsymbol{S = S^T \geq 0} \tag{1.3}$$

The gradient of this function is

$$\frac{\boldsymbol{\partial J^*}}{\boldsymbol{\partial x}} = \boldsymbol{2x^T S} \tag{1.4}$$

Since we have guaranteed, by construction, that the terms inside the $\boldsymbol{min}$ are quadratic and convex (because $\boldsymbol{R} > 0$), we can take the minimum explicitly by finding the solution where the gradient of those terms vanishes:

$$\frac{\partial}{\partial \boldsymbol{u}} = 2\boldsymbol{u^T R} + \boldsymbol{2x^T SB} = 0 \tag{1.5}$$

This yields the optimal policy

$$\boldsymbol{u^* = \pi^*(x) = -R^{-1} B^T S x = -K x} \tag{1.6}$$

Inserting this back into the HJB and simplifying yields

$$0 = \boldsymbol{x^T [Q - SBR^{-1} B^T S + 2SA] x} \tag{1.7}$$

All of the terms here are symmetric except for the $\boldsymbol{2SA}$ **,** but since $\boldsymbol{x^T SAx = x^T A^T S x}$ we can write

$$0 = \boldsymbol{x^T [Q - SBR^{-1} B^T S + SA + A^T S\ ] x} \tag{1.8}$$

And since this condition must hold for all $\boldsymbol{x}$, it is sufficient to consider the matrix equation

$$0 = \ \boldsymbol{SA + A^T S - SBR^{-1} B^T S + Q} \tag{1.9}$$

This extremely important equation is a version of the algebraic Riccati equation. Note that it is quadratic in $\boldsymbol{S}$, making its solution non-trivial, but it is well known that the equation has a single positive-definite solution if and only if the system is controllable and there are good numerical methods for finding that solution, even in high-dimensional problems.

Chapter 2

Planning in LQR -Trees

In this chapter, we presented a brief overview of the LQR-tree along with the proposed extension on the steering function of the RRT, as well as the proposed method the proposed in adjusting the funnels in en environments with obstacles using a sampled-based approach.

2.1 The concept of LQR-Tree algorithm

The key idea in [9] is that the algorithm generates a tree $\mathcal{T} = \{J_1, J_2, \dots \dots .\}$that consists of a set of feedback-stabilized nominal trajectories that cover the set of states in the design set that are to be stabilized to the goal G with the union of their funnels. The tree is generated iteratively with randomized sampling approach.

1) If there is a trajectory in the current tree whose feedback policy can stabilize the sample state to the goal $\mathcal{X}_{goal}$, i.e., the sample is in the funnel of that trajectory, the algorithm proceeds to the next iteration.
2) If there is no such trajectory, the motion-planner module attempts to find a trajectory from the sample state to the goal $\mathcal{X}_{goal}$.
3) If motion-planning is successful, a feedback policy is generated by the feedback-control module, and the new trajectory is added to the tree.
4) If motion-planning fails, then the sample state is not in the stabilizable set, and the algorithm proceeds to the next iteration.

These steps are iterated until the stabilized set is covered by the funnels of the trajectories in the trees. This iterative procedure is outlined in pseudo code in Algorithm 2 along with the illustration from [9].

Algorithm 2: LQR-Trees

DATA:

c: The cost to go function

$\mathcal{T}$: Tree (Output)

$\mathcal{X}_{goal}$: the goal region

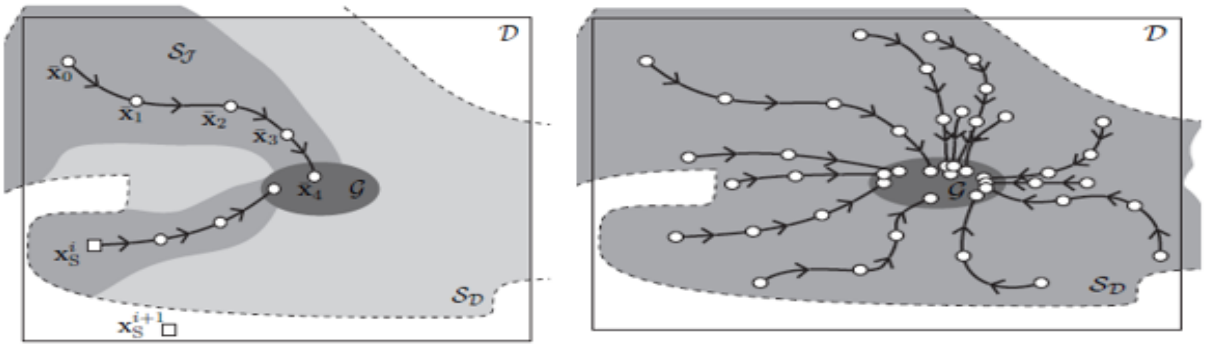
S_D: The stabilizable set

1: $\mathcal{T} \leftarrow \emptyset$
2: **While** $NotCovered(S_D, \mathcal{T})$**do**
3: $x_s \leftarrow getRandomSample()$
4: **If** $IsInFunnel(x_s, \mathcal{T}) or\ InObstacle$**then**
5: Continue
6: **Else**
7: $\{\{u_k\}, \{x_k\}\} \leftarrow RRT\_Planning(x_s, c, \mathcal{I}, \mathcal{X}_{goal})$
8: **If** $PlannerSuccessful$ **then**
9: $\pi \leftarrow addfeedbackControl(\{\{u_k\}, \{x_k\}\})$ // detailed in 2.3
10: $\mathcal{T} \leftarrow addTrajectory(\mathcal{T}, \pi, \{\{u_k\}, \{x_k\}\})$
11: **Else**
12: Continue
13: **End if**
14: **End if**
15: **End while**

2.1.1 LQR-Trees achieve "Probabilistic Feedback Coverage"

While dealing with large dimensional problems, covering the reachable state space may be unnecessary or impractical. Based on the RRTs, the LQR-trees can easily be steered towards a region of state space (e.g., by sampling from that region with slightly higher probability) containing important initial conditions. Termination could then occur when some important subspace is covered by the tree. Hence, the goal of the algorithm is to cover an entire region of interest, the set of points from which the goal state is reachable, or a specified bounded subset of this region, with this stabilizing controller. This property is defined as "probabilistic feedback

coverage", the latter implies that, as the number of algorithm iterations tends to infinity, the tree-policy is able to stabilize all states in the stabilized set S_D to the goal set G. To achieve this, we grow our trees in the fashion of an RRT, where new sub goals are chosen at random from a uniform distribution over the state space. Unlike the RRTs, we have additional information from the estimated regions of stability (funnel) and differential constraints, and we can immediately discard sample points which are sampled inside the previously verified region.



(a) (b)

Figure 2-1: (a) the tree T consists of a single trajectory J with nominal states $\{x_0 \ldots \ldots . x_4\}$.The funnel is shown in medium dark gray. The intersection of the stabilizable set S (dashed outline) with the design set is shown in light gray. The random sample x_s^idrawn at iteration$i$ is not in $S_J$andthe algorithm adds a trajectory to the tree that connects x_s^i to the goal set G (dark gray). The next random sample x_s^{i+1} is not in S_D, causing the motion-planner to fail, and the algorithm proceeds to the next iteration. (b) The algorithm terminates after enough trajectories were added to cover S_D with their funnels [9].

2.2 The Motion Planning Module

An essential component of the LQR-tree algorithm is the method by which the tree is extended. Following the RRT approach, we select a sample at random from some distribution over the state space, and attempt to grow the tree towards that sample. Unfortunately, RRTs typically do not grow very efficiently in differentially constrained (e.g., underactuated) systems, because simple distance metrics like the Euclidean distance are inefficient in determining which node in the tree to extend from. Further embracing LQR as a tool for motion planning, in this section we develop an affine quadratic regulator around the sample point, and then use the resulting cost-to-go function to determine which node to extend from, and use the open-loop optimal policy to extend the tree. In order to grow the RRT in an efficient way, we used the metric derived in [4] as a measure of closeness between the states in the state space.

2.2.1 The distance metric in growing the tree

Choose a random sample (not necessarily a fixed point) in state space, x_{rand} and a default u_0, and use $\bar{x} = x - x_{rand}$, $\bar{u} = u - u_0$.

Now linearize around $\mathrm{x_{rand}}$:

$$\dot{\bar{x}} = \frac{d}{dt}(x(t) - x_{rand}) = \dot{x}$$

$$\dot{\bar{x}} \approx f(x_{rand}, u_0) + \frac{\partial f}{\partial x}(x(t) - x_{rand}) + \frac{\partial f}{\partial u}(u(t) - u_0) \qquad (2.1)$$

$$\dot{\bar{x}} = c + A\bar{x} + B\bar{u}$$

Assume u_0, the input vector at tf , the end of the finite horizon, is zero. Define an affine quadratic regulator problem with a hard constraint on the final state, but with the final time, tf, left as a free variable

$$J(\bar{x}, t_0, t_f) = \int_{t_0}^{t_f} \left(1 + \frac{1}{2}\bar{u}^T R \bar{u}\right) dt \qquad R = R^T > 0 \qquad (2.2)$$

$$s.t\ \bar{x}(t_f) = 0\ , \bar{x}(t_0) = \bar{x}_0,\ \ \dot{\bar{x}} = c + A\bar{x} + B\bar{u}$$

Using Pontryagin's minimum principle, a necessary condition for optimality, it can be derived that the inverse of S(t), referred to here as P(t), is governed by the following differential matrix equation:

$$\dot{P}(t) = AP(t) + P(t)A^T + BR^{-1}B^T, \quad P(t_0) = 0 \qquad (2.3)$$

The resulting cost-to-go is:

$$J^*(\bar{x}, t_f) = t_f + \frac{1}{2} d^T(\bar{x}, t_f) P^{-1}(t_f) d(\bar{x}, t_f) \qquad (2.4)$$

Where:

$$d(t) = e^{At} + \int_0^{tf} e^{A(t_f - \tau)} d\tau$$

The LQR-based proximity heuristic's value for the distance from x to x_{rand} is:

$$LQR\_Based\_Proximity\ (x \to x_{rand}) = min_{tf}\ J^*(\bar{x}, t_f)$$

2.2.2 The proposed steering function used in RRT

In order to build an LQR-Tree, nodes within the tree must be linked together with valid trajectories through state-space with arbitrary endpoints [7] [x_0; x_{rand}] provided by the global planner. For local trajectory generation routine the 4th order Runge-Kutta transcription on the nonlinear equality constraints is used and the Euler transcription on the cost to go function in the direct collocation method with P collocated states to find an optimal open loop trajectory for the system. In order to generate these trajectories we solved the optimization program using MATLAB's fmincon. While direct collocation may introduce small errors from approximating trajectories as cubic splines and the control as linear interpretation, it is often better numerically conditioned than multiple shooting as direct collocation features a sparse

constraint matrix. In addition to penalizing input and state we also chose to make the trajectory time step a decision variable. This allows for a trade-off between time and energy efficiency.

$$\min_{z} \sum_{k=0}^{N-1} \Delta t \left(\left(x_k - x_{goal}\right)^T Q \left(x_k - x_{goal}\right) + \left(u_k - u_{goal}\right)^T R \left(u_k - u_{goal}\right) \right)$$

Subjected to:

$$x_{k+1} = x_k + \frac{\Delta t}{6}(k_{0k} + 2k_{1k} + 2k_{2k} + k_{3k})$$

$$k_{0k} = f(t_k, x_k, u_k)$$

$$k_{1k} = f(t_k + 0.5\Delta t, x_k + 0.5k_{0k}\Delta t, u_k)$$

$$k_{2k} = f(t_k + 0.5\Delta t, x_k + 0.5k_{1k}\Delta t, u_k)$$

$$k_{3k} = f(t_k + 0.5\Delta t, x_k + k_{2k}\Delta t, u_{k+1})$$

$$x_0 = x_{NearestIn\,Tree} \quad And\ x_P = x_{new}$$

$$|u_k| \leq u_{max}^{local} \qquad \forall k$$

$$x_k \in \chi_{free} \qquad \forall k$$

$$\Delta t_{min} < \Delta t < \Delta \mathrm{t}_{\max}$$

$$z = [\Delta t, x_0, x_1, \ldots \ldots x_P, u_0, u_1, \ldots . . u_P]$$

Where x_{new} is closer to x_{rand} than x_0 with respect to the metric (2.4), It is also worth noting that $u_{max}^{local} < u_{max}$ in order to allow some controllability to the LQR controller.

Once x_P reaches the goal region, the RRT planning stops and then we performed a backtracking from the goal state to the initial sampled state. Furthermore we extended this procedure by adding a collision detection function that checks if all the collocated states are collision free. If there is no collision then append the extension to the RRT, otherwise truncate the segment and remove all the states that occur after the first state is not collision free.

The algorithm of the direct collocation based RRT (DC-RRT) planner is as follows:

Algorithm 3: DC-RRT planner

1: $\mathcal{J} \leftarrow \emptyset$
2: $\mathcal{J} \leftarrow InsertNode(x_{init}, \mathcal{J})$
3: **For** $k = 1 \ to \ k = K$
4: $x_{rand} \leftarrow RandomState()$
5: $x_{near} \leftarrow NearestState(x_{rand}, \mathcal{J})$ // using the metric (4)
6: $[t_f, \{x\}_P, \{u\}_p, flag] = dc\_steer(x_{near}, x_{rand})$ // using the proposed method in 2.2.2
7: **If** $flag == 1$ **then** // flag is 1 if the solution from the solver is feasible
8: **If** $CollisionFree$ **then**
9: $\mathcal{J} \leftarrow Insert\_traj(\{x\}_P, \{u\}_P, \mathcal{J})$
10: **Else** // truncate the trajectory
11: $\mathcal{J} \leftarrow Insert\_traj(\{x\}_q, \{u\}_q, \mathcal{J})$ // q<p
12: **End if**
13: **Else**
14: Continue
15: **End if**
16: **End for**

2.3 Stabilizing a Trajectory and Funnel Approximation

The Key to the implementation of the LQR-Tree algorithm are the funnels of the trajectories in the tree. The funnel $S_{\mathcal{J}}$ of the trajectory $\mathcal{J}$ is the set of states that can be stabilized to the goal G by its feedback policy $\pi_k(x)$ without violating state constraints.Funnels are, in general, not straight forward to estimate, they depend on the system dynamics, the feedback policy, the state and input constraints, and the goal set.

2.3.1 Stabilizing the Goal State and Verification of the Basin of Attraction

Goal state controller is derived using the time-invariant linear quadratic regulator (TILQR). Consider the continuous nonlinear system (1), by linearizing it around the goal state and input (x_G, u_G) and discretizing it to obtain the discrete-time, time-invariant linear system dynamics:

$$\bar{x}_{k+1} = A\bar{x}_k + B\bar{u}_k \qquad (2.5)$$

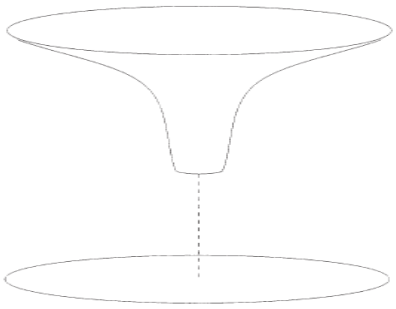


Figure 2-2: The Lyapunov function as a funnel: an idealized graph of a positive-definite function over its state space centered at the goal point [8].

The cost-to-go function to be minimized is:

$$\boldsymbol{J}(\bar{x}_0) = \sum_{n=0}^{\infty} \bar{x}_{\boldsymbol{n}}^{\boldsymbol{T}} Q \bar{x}_{\boldsymbol{n}} + \bar{u}_{\boldsymbol{n}}^{\boldsymbol{T}} R \bar{u}_{\boldsymbol{n}} \qquad (2.6)$$

Where $\bar{x}_0$ is the initial state. The optimal cost-to-go for a linear system is given by:

$$\boldsymbol{J}^*(\bar{x}_k) = \bar{x}_{\boldsymbol{k}}^{\boldsymbol{T}} S_{\boldsymbol{G}} \bar{x}_{\boldsymbol{k}} \qquad (2.7)$$

Where$S_{\boldsymbol{G}} \geq 0$ is the unique stabilizing solution to the discrete algebraic Riccati equation:

$$0 = Q - S_G + A^T (S_G - S_G B (R + B^T S_G B)^{-1} B^T S_G) A \qquad (2.8)$$

The optimal feedback policy is given by:

$$\bar{u}_k^* = -(R + B^T S_G B)^{-1} B^T S_G A \bar{x}_{\boldsymbol{k}} = -K_G \bar{x}_{\boldsymbol{k}} \qquad (2.9)$$

Thus by combining (2.5) and (2.9) we'll get the following closed-loop dynamic formula:

$$\bar{x}_{k+1} = (A - BK_G)\bar{x}_k \qquad (2.10)$$

Next, the approximated the region of attraction of the nonlinear closed-loop system at the goal state which is defined using the optimal cost-to-go function as a metric to describe the goal as a sub level set.

$$\mathcal{B}(\rho_G) = \{\bar{x} : \bar{x}^T S_G \bar{x} \leq \rho_G\} \qquad (2.11)$$

In this report the parameter ρ_G is determined through the following sampling-based procedure:

1. Initialize $\rho_G > 0$ such that $\mathcal{B}(\rho_G)$ belongs to the design set.
2. Draw a random sample $x_S \in \mathcal{B}(\rho_G)$ from a uniform distribution on $\mathcal{B}(\rho_G)$. The used algorithm for the distribution is introduced in [6].
3. Check the state constraints, if $x_S \in \mathcal{X}$ proceed to step 4; else proceed to step 5.

4. Calculate $f(x_S, \bar{u}_s^*)$ using numerical integration (ode45 in MATLAB) , then check if the following Lyapunov test holds :

$$\boldsymbol{J}^*(f(\bar{x}_S, \bar{u}_s^*)) - \boldsymbol{J}^*(\bar{x}_S) < 0 \qquad (2.12)$$

   if yes, then return to step 2; otherwise proceed to step 5
5. Shrink the approximated ellipse with

$$\rho_G \leftarrow \bar{x}_S^T S_G \bar{x}_S$$

   Then return to step 2.

The above procedure is terminated when a consecutive sequence of M samples fulfill the Lyapunov test.

2.3.2 Linear time-varying linear quadratic regulator (TVLQR)

To stabilize the nominal trajectory, we use a time-varying linear quadratic regulator (TVLQR). To apply this control methodology, we linearize then discretized the nonlinear system (1) about the nominal trajectory to obtain a time-varying linear system which can be represented as:

$$\dot{\bar{\boldsymbol{x}}}_{\boldsymbol{k+1}} = A_k \bar{\boldsymbol{x}}_{\boldsymbol{k}} + B_k \bar{\boldsymbol{u}}_{\boldsymbol{k}} \qquad (2.13)$$

Where

$\bar{x}_k = x_k - x_{0k}$, $\bar{u}_k = u_k - u_{0k}$ k=0,1,……..,N

and A_k, B_k are the Jacobian of the nonlinear system around the local stability points.

This method would allow us to divide the motion planning problems into sub problems; therefore, using the TVLQR we construct multiple funnels and sequentially combine them in order to reach the final goal state.

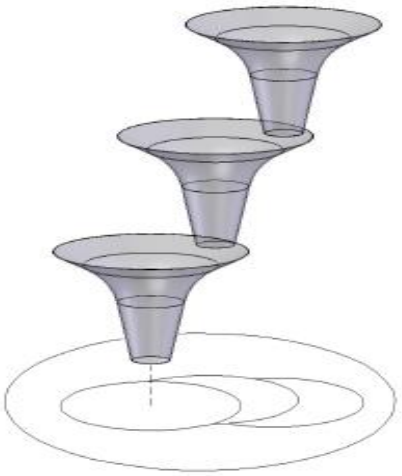


Figure 2-3: The sequential composition of funnels. The goal point of each controller lies within the domain of attraction induced by the next-lower controller. Each controller is only active outside the domains of lower controllers. The lowest controller stabilizes the system at the final destination [8].

It is important to Note that the controller represented by each funnel is only active when the system state is in the appropriate region of the state space (beyond the reach of lower funnels).As each controller drives the system toward its local goal, the state crosses a boundary into another region of state space where another controller is active. This process is repeated until the state reaches the final cell, which is the only one to contain the goal set of its own controller.

Let us define the following cost function for the maneuver for the discrete system that minimizes the control:

$$\boldsymbol{J}(\bar{x}_k) = \bar{x}_{\boldsymbol{G}}^{\boldsymbol{T}} S_{\boldsymbol{G}} \bar{x}_{\boldsymbol{G}} + \sum_{n=k}^{N-1} \bar{x}_{\boldsymbol{k}}^{\boldsymbol{T}} Q \bar{x}_{\boldsymbol{k}} + \bar{u}_{\boldsymbol{k}}^{\boldsymbol{T}} R \bar{u}_{\boldsymbol{k}} \qquad (2.14)$$

Where $S_{\boldsymbol{G}}$, $Q$and R are penalty matrices on the final state deviation and state and input deviation from the nominal trajectory, respectively. The optimal cost-to-go is given by:

$$\boldsymbol{J}_{\boldsymbol{k}}^{*}(\bar{x}_k) = \bar{x}_{\boldsymbol{k}}^{\boldsymbol{T}} S_{\boldsymbol{k}} \bar{x}_{\boldsymbol{k}} \qquad (2.15)$$

Where $S_{\boldsymbol{k}} \geq 0$ is given by backwards-iterating the of the Riccati equation:

$$S_k = Q + A_k^T (S_{k+1} - S_{k+1} B_k (R + B_k^T S_{k+1} B_k)^{-1} B_k^T S_{k+1}) A_k \qquad (2.16)$$

$$K_k = (R + B_k^T S_{k+1} B_k)^{-1} B_k^T S_{k+1} A_k$$

With the boundary condition $S_N = S_{\boldsymbol{G}}$. The optimal inputis given by:

$$\bar{u}_k^* = -(R + B_k^T S_{k+1} B_k)^{-1} B_k^T S_{k+1} A_k \bar{x}_{\boldsymbol{k}} \qquad (2.17)$$

$$\bar{u}_k^* = -K_k \bar{x}_{\boldsymbol{k}} \qquad (2.18)$$

Where $K_k \, \epsilon \, \mathbb{R}^{m \times n}$ is the time-varying compensator matrix.

2.3.3 Simulation-Based Funnel Approximation

2.3.3.1 Funnel Hypothesis test in the free space

The funnel of a trajectory is the set of states around the trajectory which the TVLQR policy can get to the approximated goal basin without violating state and input constraints. After calculating the TVLQR policy for a trajectory, each nominal state x_{0k}. of the trajectory has an associated optimal cost-to-go matrix $S_k$and compensator matrix K_k. The matrix $S_k$together with the funnel parameter$\phi_k \, \epsilon \, \mathbb{R}$ describes a hyper ellipsis around the nominal state x_{0k}. A state x is inside this ellipsis if:

$$(x - x_{0k})^T S_k (x - x_{0k}) \leq \phi_k \qquad (2.19)$$

The union of the ellipses around all nominal states in a trajectory is the approximated funnel of the trajectory. We also used an ellipsis to describe the approximated goal basin $\mathcal{B}(\rho_G)$. But there is an important difference between the two parameters ρ_G and ϕ_k;ρ_G is fixed while

the ϕ_k of a trajectory may change in an iteration of the algorithm. This is due to the fact that We approximated the funnel of a trajectory by falsification (sampled-based method).

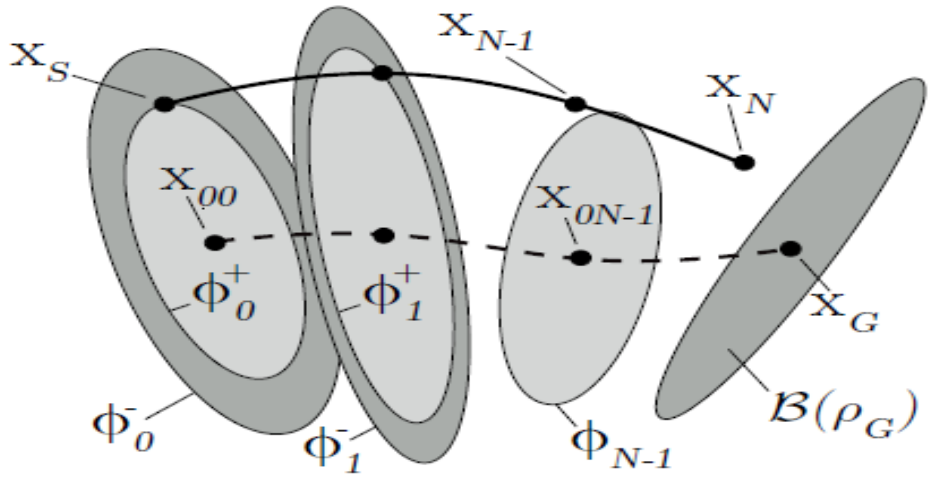


Figure 2-4: Adjusting the funnel after a failed simulation. The simulated trajectory (solid black) failed to reach the goal basin $\mathcal{B}(\rho_G)$ using the policy starting at nod eq: X_{00} , ϕ_0. However, the funnel described by the darker grey ellipses defined by $\phi^-{}_{0,1}$ around the first two nodes of the policy's nominal trajectory (dashed line) predicted a successful simulation. Therefore, we adjust $\phi^-{}_{0,1}$ to $\phi^+{}_{0,1}$ according to (2.19), resulting in the light grey ellipses. The simulated state at time index N - 1 was not inside the ellipsis of node N - 1 and thus ϕ_{N-1}. Remains unchanged.

The funnel approximation mechanism using simulation and falsification is somewhat similar to the one used to compute the goal region.

Consider a random sample x_s which belongs to the design set. And let's assume that x_s happens to be inside an estimated funnel of node x_n in trajectory $\mathcal{J}$ where x_n is the starting node of a nominal trajectory, $x_S \in \mathcal{B}(\bar{x}_n, \phi_n)$. We then test if the hypothesis holds for x_S, i.e. we check if x_S is in the funnel x_n. This test is straightforward when x_S is simulated with the policy of node x_n:

1. Set $x_n = x_s$, simulate the system by applying the control of node x_n (closed loop control), and obtain the state trajectory $\{ x_n, x_{n+1}, \dots\dots\dots, x_N\}$
2. Check if the following condition holds for the obtained trajectory.
$$x_N \in \mathcal{B}(\rho_G) \qquad and \qquad x_k \in \mathcal{X}\ \forall k \in \{n, n+1, \dots\dots, N-1\}$$
3. If the condition is satisfied then the simulation is successful and the funnel doesn't need to be adjusted; otherwise the simulation fails and we need to shrink the size of the funnel, therefore after the failed simulation we set
$$\phi_k \leftarrow \min\{ \boldsymbol{J}_{\boldsymbol{k}}^*(x_k - x_{0k}), \phi_k\} \qquad \forall k \in \{n, n+1, \dots\dots, N-1\}$$

It is important to notice that the only ellipses that are adjusted are only the ones at which they contain the state trajectory but fail to reach the goal. Furthermore, the procedure above only shrinks the ellipses but never expand them.

2.3.3.2 The proposed Funnel Hypothesis test in an environment with obstacles:

This is a proposed extension of the previous approach where the model is in an environment with obstacles. The Algorithm is as follow:

If the procedure presented in 2.3.3.1 if the second step holds, then add then check the following condition:

1. Load the simulated state trajectory $\{ x_n, x_{n+1}, \dots\dots\dots, x_N\}$.
2. Omit first derivatives of each configuration, therefore the state trajectory becomes $\{ q_n, q_{n+1}, \dots\dots\dots, q_N\}$ where $\dim(q_n) = \dim(x_n)/2$.
3. Pass $\{ q_n, q_{n+1}, \dots\dots\dots, q_N\}$ through a collision detection module.
4. If the collision detection module returns false then there is collision and the funnel should be adjusted (need to be shrunk), otherwise the simulation is successful and the funnel doesn't need to be adjusted. Henceforth after the collision is detected we set
$$\phi_k \leftarrow \boldsymbol{J_k^*}(x_k - x_{0k}) \qquad \forall k \in \{n, n+1, \dots\dots, m\}$$
Where m represents the number of applied policies before collision as depicted in figure 2-5.

Moreover, in many cases there are some configurations that are redundant, the redundancy of these latter are high related to the geometric shapes of both the obstacles and the robot. Hence in these cases: $\dim(q_n) \leq \dim(x_n)/2$.

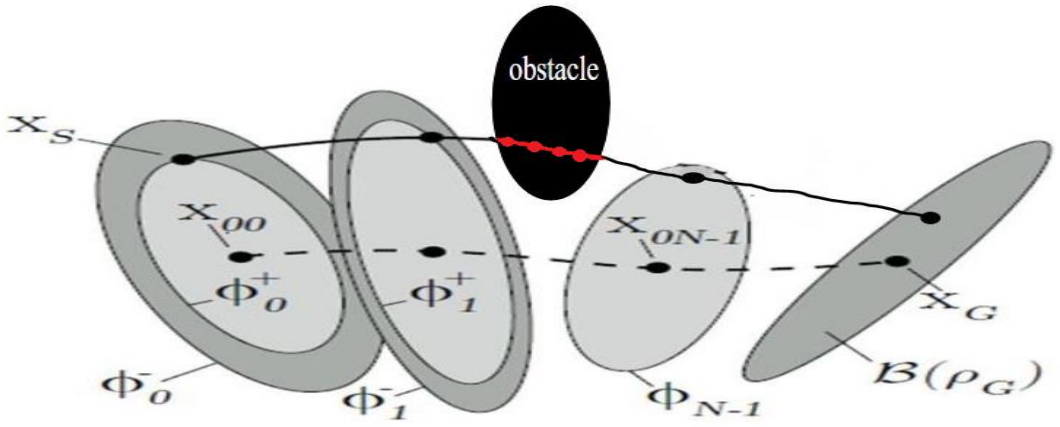


Figure 2-5: Adjusting the funnel after a collision is detected. The simulated trajectory (solid black) reached the goal basin $\mathcal{B}(\rho_G)$ using the policy starting at nod eq: X_{00} , ϕ_0 but a collision was detected. Therefore, we adjust $\phi^-_{0,1}$ to $\phi^+_{0,1}$ according to (2.19), resulting in the light grey ellipses. The simulated state at time index N - 1 was after the collision and thus ϕ_{N-1}. Remains unchanged since the policies were applied after the collision has occurred.

2.4 Iteration of the Algorithm

An iteration of the algorithm can be summarized as follows:

1. Draw a i.i.d random sample
2. Find a node policy that could stabilize the random sample to the goal region by picking nominal states from the LQR-Trees starting from the closest w.r.t to the TV-LQR metric. If no feasible policy is found, then shrink the funnel (through funnel adjustment procedure) such that the random sample is outside any funnels, and then attempt to add a new nominal trajectory to the tree. And proceed to the next iteration.

2.4.1 Interpretation of Funnel Hypotheses

The specific steps taken in an iteration of the algorithm have an implication on the funnel hypotheses that is not obvious, but that is useful to point out for the interpretation of the generated tree-policy: First, note that a sample x_s can be in multiple funnel hypotheses due to possible overlap. Second, note that the algorithm immediately proceeds to the next iteration after the first successful stabilization of the sample x_s (x_s may be simulated with multiple node policies until the first successful stabilization occurs). The consequence of both observations is that some subsets of funnel hypotheses may never be tested, and therefore may contain states that cannot be stabilized by the respective node-policy. This creates an issue in the execution phase where the initial random state is in an overlap of funnel hypotheses then picks the closest node policy, but it cannot be stabilized.

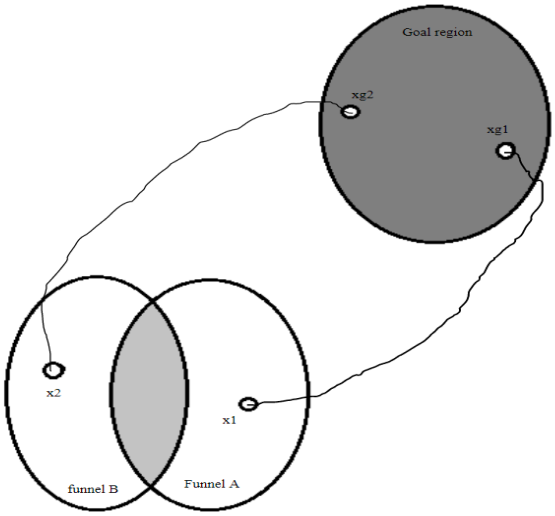


Figure 2-6: Overlap of funnel hypotheses (circles) and policy assignment. There are two trajectories A and B. If the funnel hypothesis of node x1 in Trajectory A never shrinks, the light gray set of states in the hypothesis of x1 in Trajectory b is never tested since the states are closer to x1 than x2. In this case, it may be that some states in the light gray region cannot be stabilized by the node-policy of x2 , but they are never tested because the hypothesis of x1 hides them from the algorithm.

In order to fix this problem or decrease its occurrence we set a heuristic termination condition besides reaching the maximum number of iterations, and that is to set the algorithm terminates after a consecutive sequence of M samples $x_s \in D$ does not cause the tree-policy to change: no funnels are adjusted and no new trajectories are added. Furthermore a sample x_s of the success sequence must be stabilized by the first node policy that is applied, otherwise funnels are adjusted and the sequence for termination (heuristic termination condition) is reset to zero.

2.5 Simulation-Based LQR-Tree Algorithm

We now combine the key concepts and present the LQR-tree generating algorithm. A pseudo-code overview is given in Algorithm 4.

The following algorithm performs the following:

1: Evaluate the goal region using TILQR and LQR parameters ($S_G \; and \; K_G$).
2: Construct a nominal trajectory containing nominal states x_{0k} and nominal inputs u_{0k} leading to the goal region or other nominal trajectories if the sample isn't in a funnel or goal region.
3: Evaluate the funnels using TVLQR and compute LQR parameters ($S_k \; and \; K_k$).

Algorithm 4: Simulation-Based LQR trees

$\mathcal{T} \leftarrow \emptyset$
$[A, B] \leftarrow linearization\ around\ x_G\ and\ u_G\ then\ discretization$
$[S_G, K_G] = dTILQR(A, B, Q, R)$
$\rho_G \leftarrow approximate\ the\ goal\ region\ using\ the\ algorithm\ in\ [6]$
$\mathcal{T}.init(\{x_G, u_G, S_G, K_G, \rho_G, NULL\})$ //{NULL is a pointer that points to the parent state }
For $i = 1\ to\ MaxIteration\ or\ Consecutive\_Success$ **do**
    $x_S \leftarrow random\ sample\ in\ \mathcal{X}$
    While $IsInFunnel(x_S, \mathcal{T})$ **do**
        $N^* \leftarrow GetStartNode(x_S)$
        $xSim \leftarrow SimulateSystem(\mathcal{T}, N^*, x_S)$
        If $IsInGoalRegion\ and\ ConstraintOK$ **then**
            If $Collision\_free$ **then**
                Continue
            Else
                $\mathcal{T} \leftarrow AdjustFunnel(\mathcal{T}, xsim)$
            End If
        Else
            $\mathcal{T} \leftarrow AdjustFunnel(\mathcal{T}, xsim)$
        End if
    End while
    If $NotInFunnel(x_S, \mathcal{T}) or\ NoSuccessfulSim(x_S, \mathcal{T}) or\ NotCollisionfree$ **then**
        $[\{\bar{x}_k\}_N, \{\bar{u}_k\}_N] = RRT\_motionPlaner(x_s)$
        If $MotionPlannerSuccessful$ **then**
            $\pi \leftarrow FeedbackPolicy(\{\bar{x}_k\}_N, \{\bar{u}_k\}_N)$ //using eqs: (2.16) and (2.17)
            $\mathcal{J} \leftarrow initTrajectory(\pi, \{\bar{x}_k\}_N, \{\bar{u}_k\}_N)$
            $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{J}$
            $reset\ counter$
        End If
    End If
End For

2.6 Running the tree policy

Given an initial sample state x_{IC} from the design set, the program tries to find a nominal state in a specific trajectory that x_{IC} is in its funnel with minimum cost-to-go from the initial sampled state to the goal region by after applying the closed loop control.

Let assume that x_{IC} is stabilized by the TV-LQR from the nominal trajectory $\{x_1, x_2, \dots\dots, x_n\}$ and open loop control $\{u_1, u_2, \dots\dots, u_n\}$. The following feedback policy defines the tree-policy that is applied to x_{IC}

$$\pi_k^{Tree}(x_{IC}) = \begin{cases} u_i - K_i(x_{actual} - x_i) \;\; 1 \le i \le n, \qquad for \;\; i = 1 \quad x_{actual} = x_{IC} \\ u_{goal} - K_G\big(x_{actual} - x_{goal}\big) \qquad i > n \end{cases} \tag{2.20}$$

Our approach also allows us to incorporate actuator limits into the verification procedure. Although we examine the single-input case in this section, this framework is easily extended to handle multiple inputs (via clipping procedure).
Let the control input π_k at time k is mapped through the following control saturation function:

$$sat(\pi_k)\begin{cases} u_{max} & if \;\; \pi_k > u_{max} \\ u_{min} & if \;\; \pi_k < u_{min} \\ \pi_k & otherwise \end{cases} \tag{2.21}$$

Finally, if x_{IC} isn't in any funnel hypothesis, one possible approach is to relax the in-funnel constraint, by applying the policy of closest node "w.r.t the quadratic metric" as measured by the TV-LQR cost-to-go.

Algorithm 5: Executing LQR-Tree policy

1: $x_S \leftarrow random\ sample\ in\ \mathcal{X}$
2: $[\{u\}_n, \{x\}_n, \{K\}_n, flag] = \min\{(x_i - x_S)^T S_i (x_i - x_S)\}$ //
$i \in \{1, \dots, all\ nodes\ of\ LQR - trees\}$
3: **If** $flag == -1$ **then**
4: $NotInAnyFunnel$
5: **Else**
6: $simulate\ the\ system\ using\ (2.20) and\ (2.21)\ and\ check\ for$
7: $state\ constraints\ and\ collision\ detection$
8: **End If**

Chapter 3

Simulation models and Results

In this chapter we apply the planning technique presented in this report on two simulation examples. The computations and Implementations in this chapter were performed using MATLAB R2017b on a PC laptop (CPU : Intel(R) Core(TM) i7-9750H with 2.60 GHz and 8 Go RAM) running on Windows 10 x64 bits.

3.1 The cart-pole model

The first example we considered is a cart-pole model with state constraints on its x-axis, a pictorial depiction of the model is provided in Figure 3-1. The cart is constrained to move within a sub range of the design set.

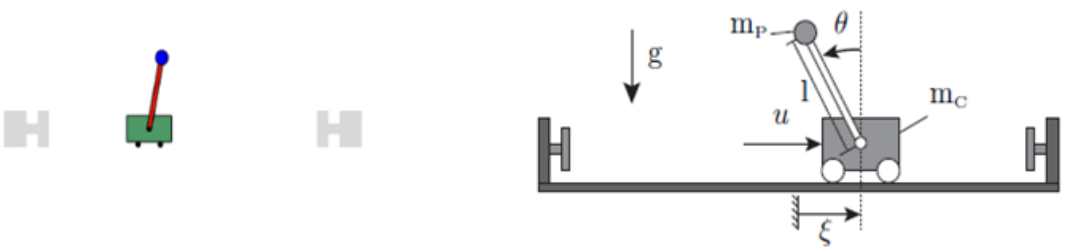


Figure 3-1: A free body diagram depicting a cart-pole system

We show the state constraint capability with the cart-pole, see Figure 3-1. It consists of an actuated cart with an undamped simple pendulum attached. The parameters are:

$m_c = 1.5\ kg, m_p = 0.175 kg, l = 0.28\ m, g = 9.81\ m/s^2$ with input u constrained $\pm\ 60\ N$.The states are defined as cart position $x_1 = \xi$, pendulum angle $x_2 = \theta$, cart velocity $x_3 = \dot{\xi}$, pendulum angular velocity $x_4 = \dot{\theta}$. The goal state is $x_1 = 0\ , x_2 = 0, x_3 = 0\ , x_4 = 0$. Where the cart is at rest and the pendulum is pointing up. The position x is constrained to $\pm\ 0.5\ m$, which is given by a limited rail the cart can move on. we set the following parameters from [9]. The full non-linear dynamics of the system are then given by:

$$\mathbf{x} = \begin{bmatrix} \xi \\ \theta \\ \dot{\xi} \\ \dot{\theta} \end{bmatrix}, \dot{\mathbf{x}} = \begin{bmatrix} \dot{\xi} \\ \dot{\theta} \\ \dfrac{u + m_p \sin\theta\,(g cos\theta - l\dot{\theta}^2)}{m_c + m_p(1 - cos^2\theta)} \\ \dfrac{cos\theta\big(u - l m_p \dot{\theta}^2 sin\theta\big) + g sin\theta (m_c + m_p)}{l(m_c + m_p(1 - \cos^2\theta))} \end{bmatrix}$$

With $\xi \in [-0.5,0.5]\, m, \theta \in [0,2\pi]\, rad, \dot{\xi} \in [-6\,,6]\, m/s\,, \dot{\theta} \in [-20,20]\, rad/s$. The control input is bounded in the range $u \in [-60,60]\, N$

The time invariant LQR control is designed with:

$\Delta t = 0.01s\,, Q_G = diag([5000,50,0.5,5])\ \ and\ \ R_G = 0.1$

The time varying LQR control is designed with:

$Q = diag([1000,300,1000,100])\ \ and\ \ R = 0.1$

3.1.1 Generate the LQR-Trees (Pre-Processing-phase)

The algorithm takes significantly long amount of time in generating a tree policy for the cart-pole model. It converged after 17832 iterations and generating a tree with 28478 nodes corresponding to 75 different nominal trajectories and took about 49.6 hours on my PC in order to generate them. We show the resulting tree in figure 3-2. For convergence, a heuristic was set such that if the tree successfully gets 480 consecutive random samples to the goal basin using the node policy of the nearest node then the program would terminate. The large number of iterations needed is partly because of the dimensionality of the model, but a larger part is due to both its complex dynamics and the increased complexity of adding the state constraints.

It is worth noting that about 97% of the time complexity is spent on the back-end procedure that generates the nominal trajectory with the open loop control using the Algorithm 3, we also set the time horizon of 3 seconds in (2.4). And the tolerance for reaching the goal is $\|\mathbf{x} - \mathbf{x_G}\| \leq 10^{-2}$.

Note that we did not plot the funnels as the projections from 4D to 2D since it can be misleading mainly due to the number of generated trajectories in order to stabilize the whole design set.

The average time for a random sample to be simulated for the whole tree was 10 to 50 milliseconds and stays almost constant over all iterations. Major spikes in time to simulate a sample can be observed after the tree has been grown and many funnels are adjusted and it can reach up to a few seconds.

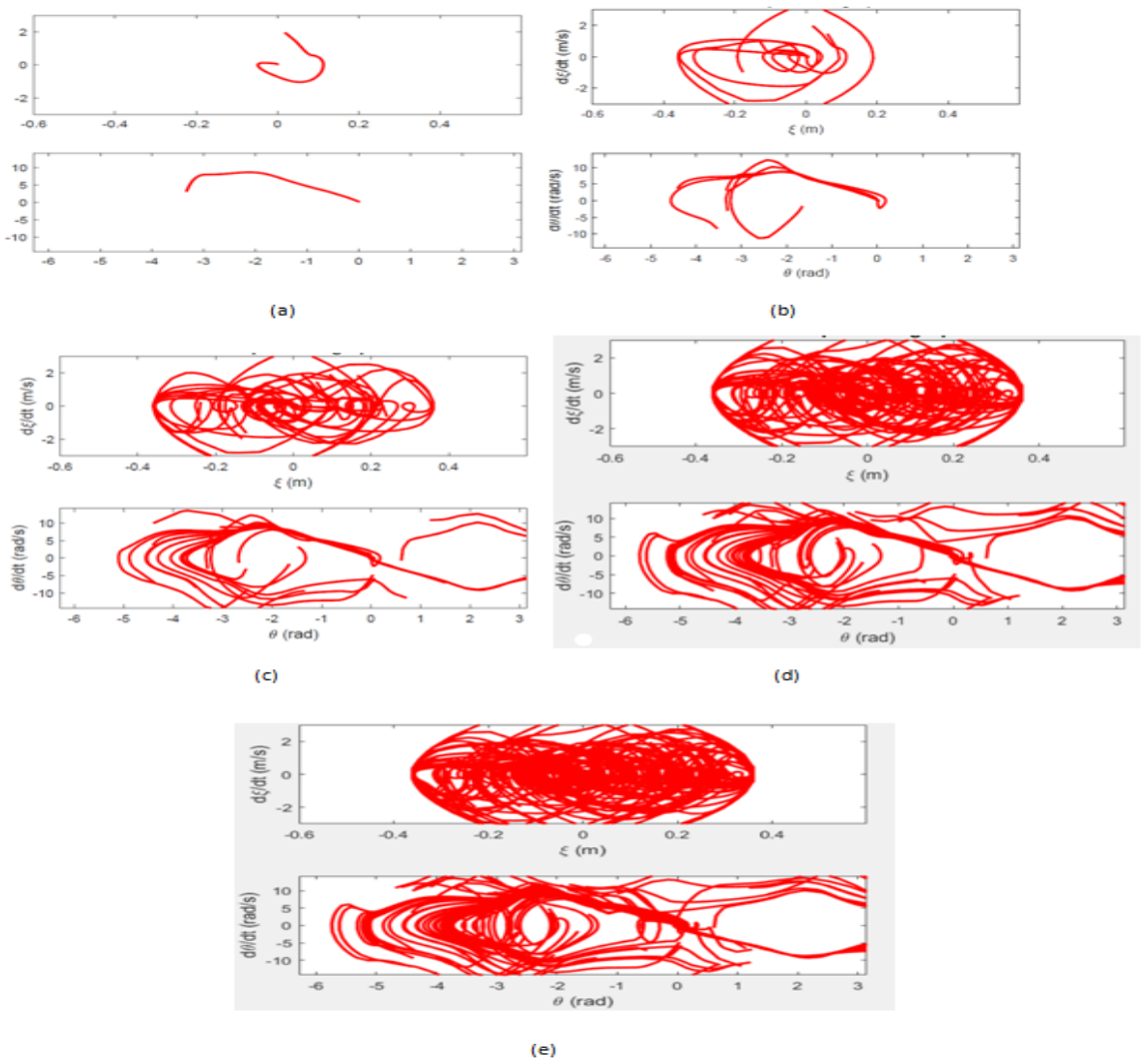
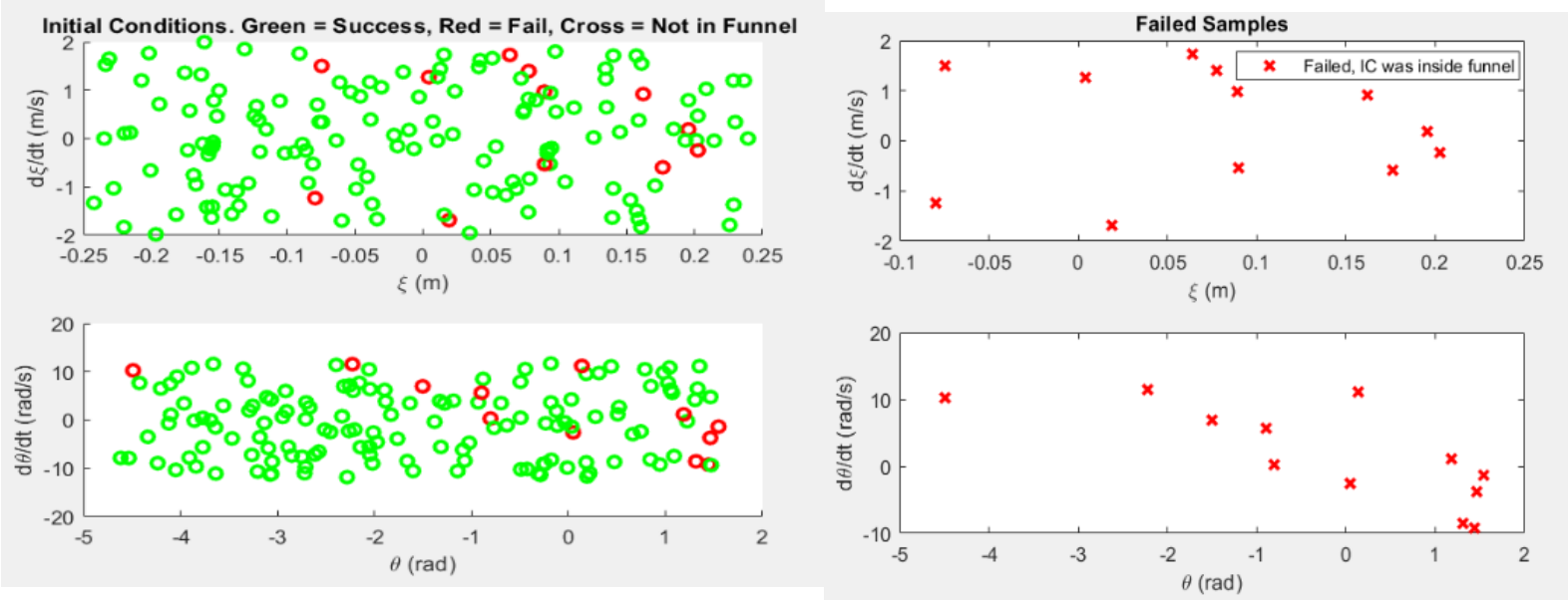


Figure 3-2: The generated tree phase plots for the cart-pole system. (a) 1 nominal trajectory (solid red), (b) 5 nominal trajectories, (c) 19 nominal trajectories , (d) 52 nominal trajectories, (e) 75 nominal trajectories

3.1.2 LQR-Trees policy (Execution-phase)

In this phase no additional nominal trajectories are generated and there are no funnels adjustments. In this step the nominal trajectories with their respective open loop control along with the time varying Riccati matrix S_k and the compensation matrix K_k of each state in the nominal trajectories were loaded, and then we performed 150 experiments, of which 138 were successful. This results in a success rate of 92% with a confidence interval of [84.55%, 96.64%].

The experimental results are shown in the following figures.



(a) (b)

Figure 3-3: (a) initial conditions of all 150 experiments, Green circles indicate success while circles indicate failure. Moreover all the initial conditions are in the design set. (b) The Crosses indicate the failed simulations that are inside a funnel.

The phase plots of some successful experiment shown in Figure 3-4. It is seen that the simulated trajectory is trying to converge to the nominal one all while mimicking its behavior; this phenomena can be observed clearly in the upper half of phase plots. Furthermore the speed at which the simulated trajectory converges to the nominal trajectory depends on the penalty matrices Q and R as well as the saturation input.

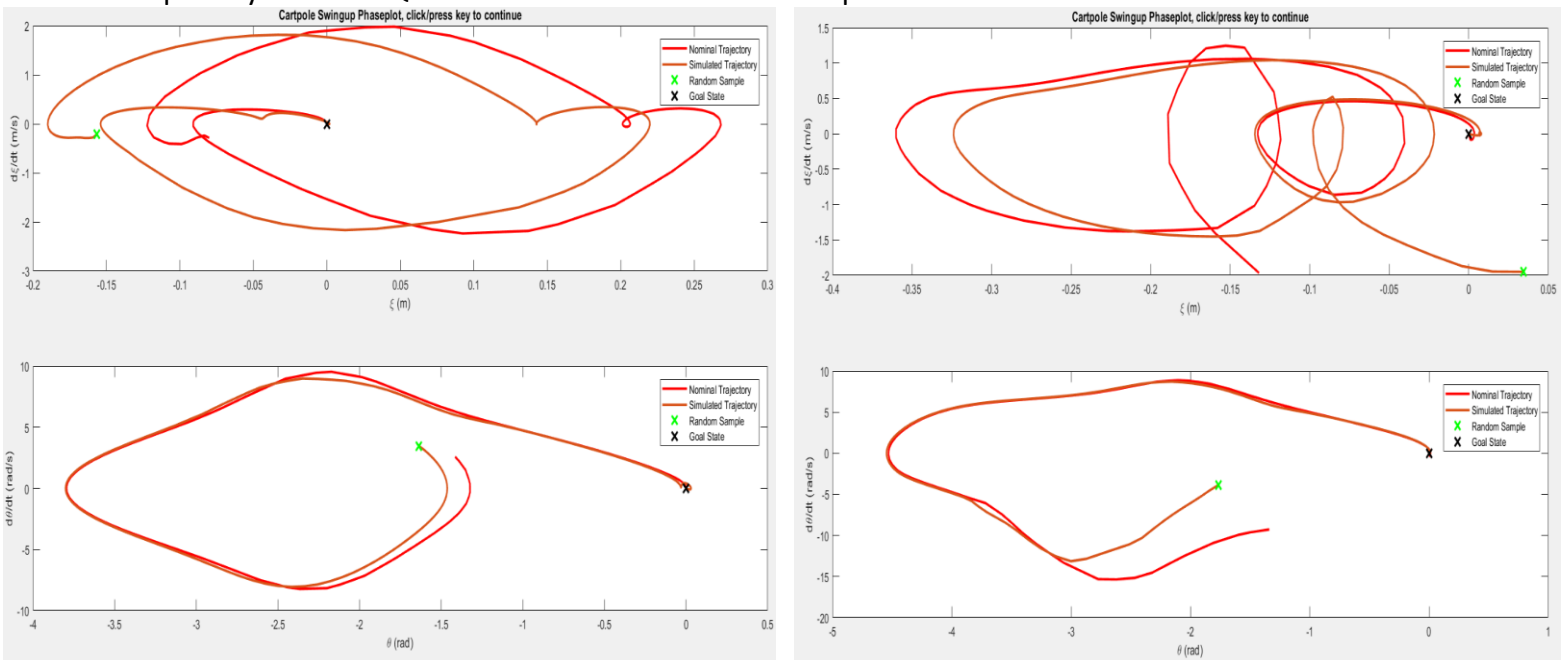


Figure 3-4: Phase plots of a successful experiment (Exp #73 and #122) respectively. The red trajectory represents the nominal one whereas the brown trajectory represents the simulated one. The Green cross represents the initial state and the black cross represents the Goal state.

3.2 Planar Quadrotor model

The next example a model of a quadrotor system navigating through a forest of circular obstacles is considered. A visualization of the system is provided in Figure 3-6. The goal of this example is to demonstrate that we can derive simple geometric conditions on the environment that guarantee collision-free flight. In other words if the environment satisfies these conditions, the Algorithm 4 presented in the previous chapter will always succeed in finding or create a collision-free funnel such that the quadrotor will fly through the environment with no collisions.

The quadrotor model has a 6 dimensional state space consisting of the x-z position of the centre of mass, the roll angle of the body, and the time derivatives of these configuration space variables. Moreover; since it is restricted to live in the plane. Then it only needs two propellers.

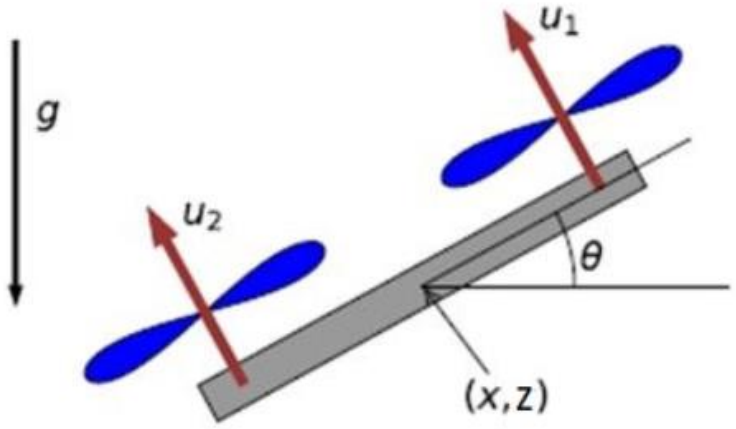


Figure 3-5: The Planar Quadrotor System. The model parameters are mass, m, moment of inertia, I, and the distance from the center to the base of the propeller, r.

The full non-linear dynamics of the system are then given by:

$$\mathbf{x} = \begin{bmatrix} x \\ z \\ \theta \\ \dot{x} \\ \dot{z} \\ \dot{\theta} \end{bmatrix}, \qquad \dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{z} \\ \dot{\theta} \\ -\frac{(u_1+u_2)sin\theta}{m} \\ \frac{(u_1+u_2)cos\theta}{m} - g \\ (u_1 - u_2)r/I \end{bmatrix}$$

Where the value of mass of the quadrotor, m is 0.486 Kg, moment of inertia, I is 0.00383 Kg.m^2, and the distance from the center to the base of the propeller, r is 0.25 m. Furthermore we imposed the state-space bounds and inputs bounds:

$(x, z) \in [-5,5] \times [0,6]\ m$,$(\dot{x}, \dot{z}) \in [-2,2] \times [-1,1]\ m/s$ and $(\theta, \dot{\theta}) \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right] rad \times \left[-\frac{\pi}{3}, \frac{\pi}{3}\right] rad/s$. $(u_1, u_2) \in [-25,25] \times [-25,25]\ N$.

The goal state is set as $x_G = [-3,2,0,0,0,0]^T$ and the goal input as $u_G = [mg/2, mg/2]^T$, it's worth noting that u_G is set such that $f(x_G, u) = \mathbf{0}$.

2.2.1 TI LQR design and Goal set

The Feedback policy that stabilizes the goal state is designed with the sampling period $\Delta t = 0.02\ s$ and the LQR penalty matrices are

$Q = diag(\left[10,10,10,1,1, \frac{r}{2\pi}\right])$ And $R = \begin{bmatrix} 0.1 & 0.05 \\ 0.05 & 0.1 \end{bmatrix}$ The goal set was approximated with the procedure outlines in 2.3.1, and The approximation takes less than 20 s

3.2.2 Motion Planning and TV LQR design

Motion planning and trajectory stabilization are implemented as described in 2.2 and 2.3.2 respectively. First, we used motion planning to find a trajectory as further away from the goal state. And then manually tuned $Q\ and\ R$ achieve acceptable tracking performance. It was found that $Q = diag([200,100,5000,100,100,60])\ and\ R = diag([1,1])$ works quite well. Moreover in order to obtain homogeneous sampling times of the trajectories for the TV LQR design, the trajectories were resampled to $\Delta t = 0.02s$ along with an interpolation of the controls and states. In particular, the sweet spot for this algorithm is taking u to be a first-order polynomial and x to be a cubic polynomial.

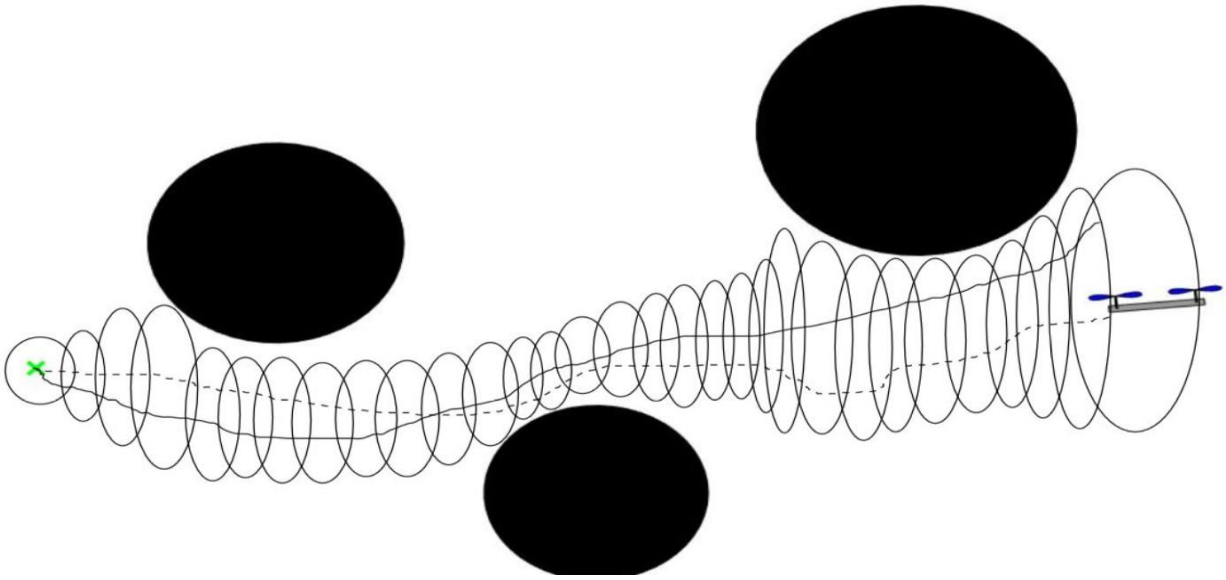


Figure 3-6: The plot shows the quadrotor maneuvering through a forest of obstacles in a collision-free manner. The environment satisfies simple geometric conditions that allow us to guarantee collision-free flight (from Exp #144). The Continuous line is the nominal trajectory, the dash line is the simulated trajectory and the sequence of ellipses represents the funnels.

3.2.3 Generate the LQR-Trees (Pre-Processing-phase)

The algorithm takes significantly long amount of time in generating a tree policy for the planar quadrotor model. It converged after 210587 iterations generating a tree with 561710 nodes corresponding to 320 different nominal trajectories and took about 182.7 hours on my PC in order to generate them. The resulting trees are shown in figure 3-7. For convergence, a heuristic was set such that if the tree successfully gets 193 consecutive random samples to the goal basin then the program would terminate. The large number of iterations needed is partly because of the dimensionality of the model, but a larger part is due to both its complex dynamics and the increased complexity of adding the state constraints as well as the added obstacles.

It is worth noting that almost all of the time complexity is spent on the back-end procedure that generates the nominal trajectory with the open loop control using the Algorithm 2, I also set the time horizon of 7 seconds in (2.4). And the tolerance for reaching the goal is $\|\mathbf{x} - \mathbf{x_G}\| \leq 10^{-3}$.

Note that we did not plot the funnels as the projections from 6D to 2D since it can be misleading mainly due to the number of generated trajectories in order to stabilize the whole design set. And also it makes our program even slower.

The average time for a random sample to be simulated for the whole tree was 100 to 200 milliseconds and stays almost constant over all iterations. Major spikes in time to simulate a sample can be observed after the tree has been grown and many funnels are adjusted and it can reach up to a few seconds or even a few minutes when the number of nodes reaches around thirty thousands. Furthermore by adding obstacles there is also a need to incorporate a collision detection procedure, the latter resulted in a major increase of the average time for a random sample to be simulated for the whole tree which took a few minutes even though the tree has a few hundred nodes, thus It is concluded that the collision detection function increase the time complexity in the procession phase in an exponential manner.

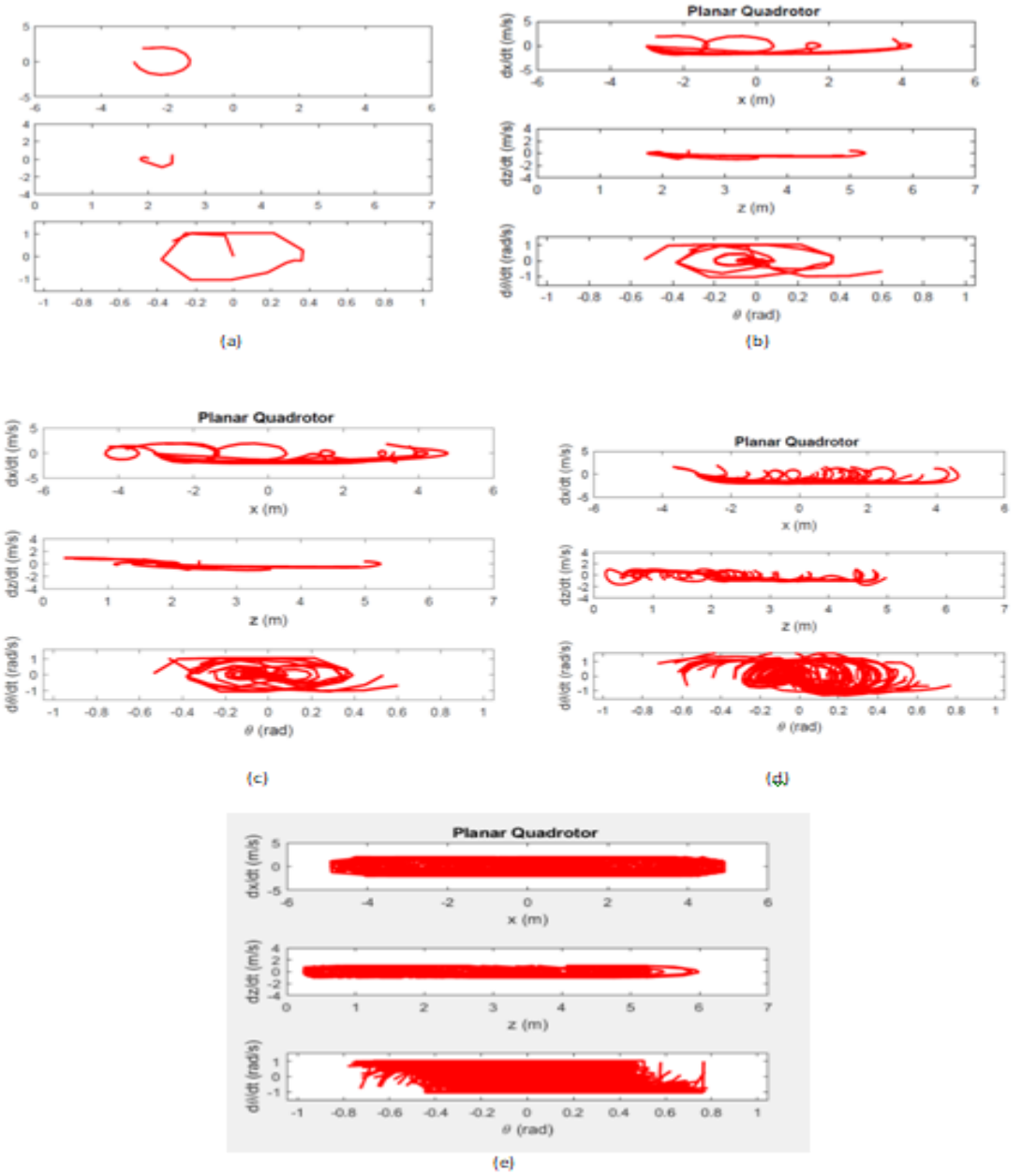


Figure 3-7: The generated tree phase plots for the quadrotor system. (a) 1 nominal trajectory (solid red), (b) 5 nominal trajectories, (c) 10 nominal trajectories , (d) 44 nominal trajectories,(e) 225 nominal trajectories

3.2.4 LQR-Trees policy (Execution-phase)

In this step, the generated LQR-Trees shown in the previous section were uploaded on the run tree program, and then pick a random initial state and look for the closes nominal trajectory w.r.t the LQR metric explained in the last chapter and finally simulate the close loop control in order to reach the goal while avoiding the obstacles.

We performed 200 experiments in each of the following case:

1- State constraints with no obstacles

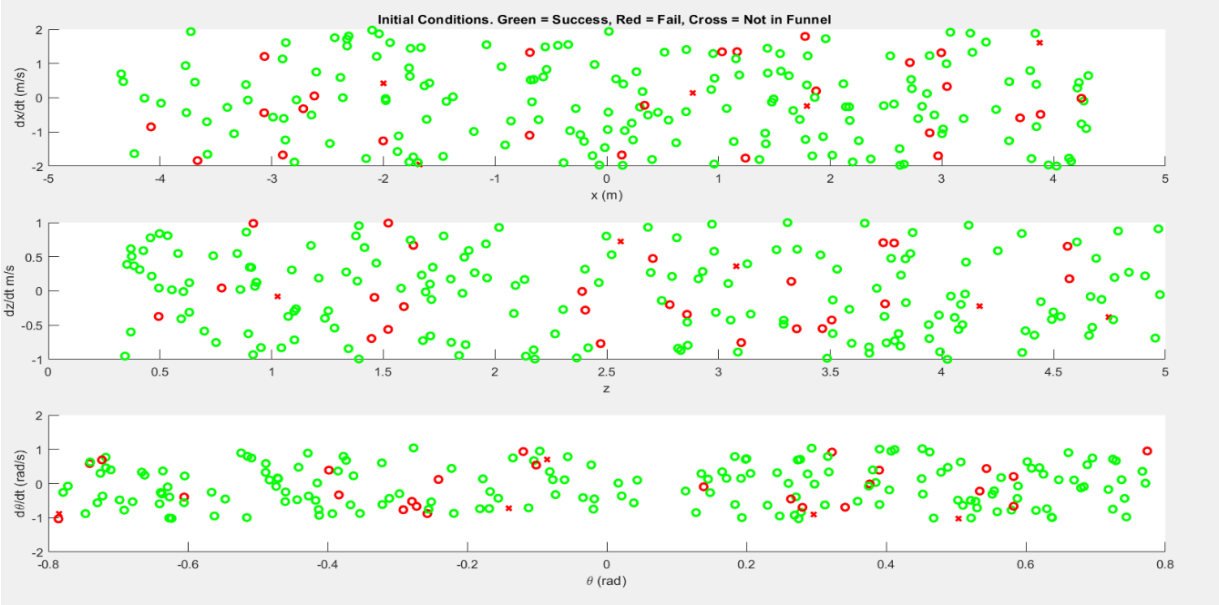


Figure 3-8: initial conditions of all 200 experiments. The green circles indicate success while red indicate failure and the crosses indicates that the samples are outside any funnels. The success rate was 89.6% for the initial conditions that are inside the design set.

Moreover, from the simulation it was concluded that the success rate would be slightly lesser if the initial samples were outside of the design set. And that no matter the number of LQR-Trees generated.

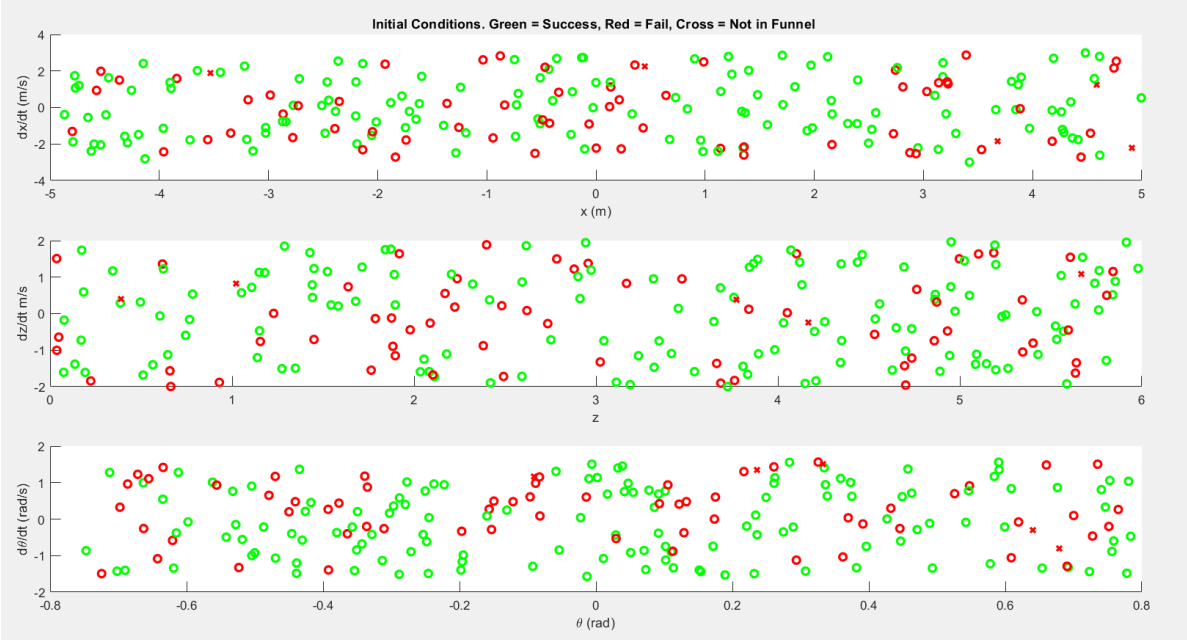


Figure 3-9: initial conditions of all 200 experiments. The green circles indicate success while red indicate failure and the crosses indicates that the samples are outside any funnels. The success rate was 69.5% for the initial conditions are both in and out of the design set.

2- State constraints with obstacles

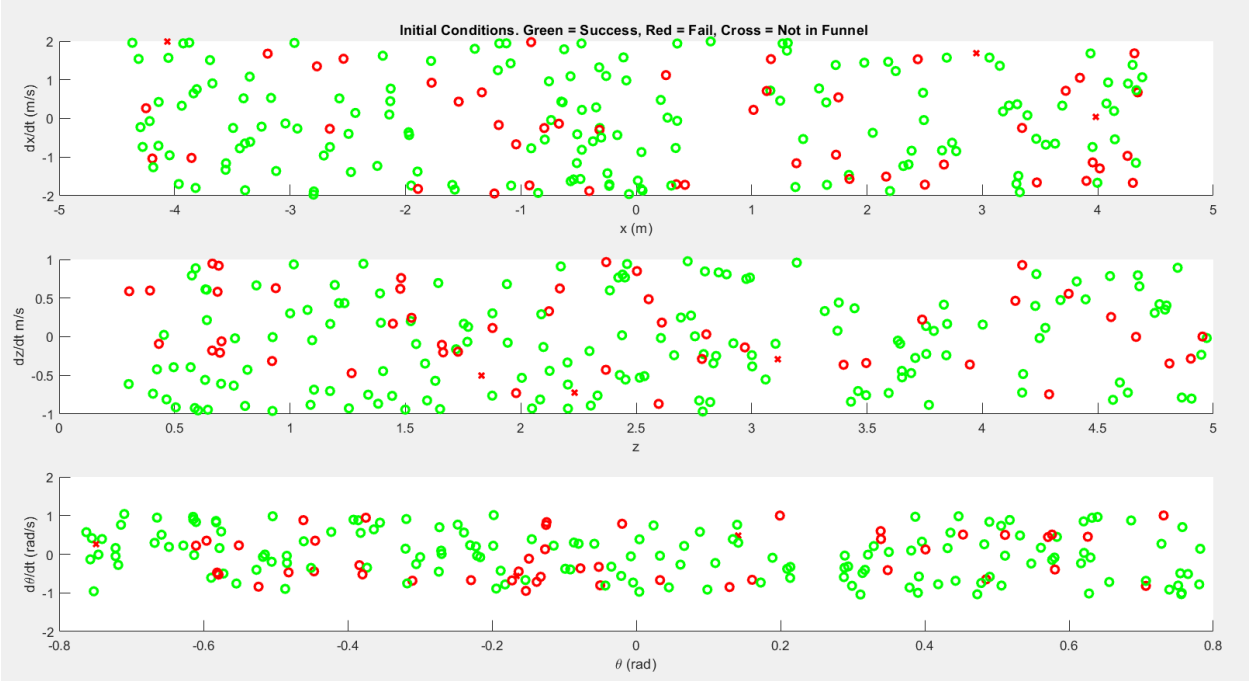


Figure 3-10: initial conditions of all 200 experiments. The green circles indicate success while red indicate failure and the crosses indicates that the samples are outside any funnels. The success rate was 76% for the initial conditions that are inside the design set.

we argue that the main reasons for failures are the tuning of the TVLQR penalty matrices since they are used in picking the appropriate and closest nominal trajectory using the formula in (14).Another reason would be that maneuvers of the quadrotor can lead it to hitting the system bounds and activating the limit switch (saturations). Another contribution may be from the interpolation error that resulted from the re-sampling procedure.

The phase plots of the experiments shown in Figure 3-11. Towards the end, the Quadrotor has to perform a large corrective maneuver to keep the system state close to the nominal trajectory.

Figure 3-11: Phase plots of a two successful simulations. The red trajectory represents the nominal one whereas the brown trajectory represents the simulated one. The Green cross represents the initial state and the black cross represents the Goal state.

Finally, we stress that the purpose of these simulations was to illustrate the feasibility of converting the rich theoretical analysis into a practical tool capable of running on modern hardware, and obtain true validation of the complete planning methodology. Further performance improvements may be obtained through incorporating richer dynamic models and leveraging higher rate dedicated controllers.

General Conclusion

Discussion and Conclusion

In this report we have presented an approach for an offline motion planning in a prior known environment using the LQR-Trees algorithm. Furthermore, in this report we added obstacles. The application of the LQR-Trees algorithm presented here represents one of the most complex (in terms of dimensionality of the state space, and numerical conditioning of the Riccati differential equation due to limited controllability) demonstrations of this nonlinear feedback control approach to date.

The finite time invariant sets or funnels of the trajectories are approximated with a simulation-based falsification method which is a sampling-based technique. Furthermore, it is unlikely that a hyper-ellipsis is the best geometrical primitive to describe the funnel around a trajectory. Simulation-based approximation of the funnel would allow exploring different primitives that could potentially yield tighter fits to the real funnel, further improving the sparsity of the resulting tree. The advantage of the hyper-ellipsis we used is that they are simple to reason about geometrically and are founded on the TVLQR design, thus are dynamically plausible. The main advantage is that the approach allows generating tree-policies for a wider range of dynamic systems, and feedback designs for trajectory stabilization. Theoretical results showed that in the long run, the algorithm tends to improve both the coverage of the initial conditions to be stabilized, and the approximations to the funnels. However While LQR-Trees has broad applicability in the realm of nonlinear systems, it does have limitations with respect to the dimension of the systems that can be solved. Its scaling behavior can best be understood by considering the components that make up the algorithm 4, which consists of a motion planner (RRT planner) , control design (TV-LQR) and a simulation-based falsification method to compute the funnels.

The control design (TV-LQR) scale well with dimension. The motion planner (RRT) has poor asymptotic behavior, but has been shown to work well on systems with 30 or more dimensions, moreover; it tends to perform poorly by adding differential constraints even with low dimensional state spaces; however, the performance can be improved by adding heuristics . The bottleneck in addressing systems of increasing dimensionality is the simulation-based falsification procedure which is a sampling-based approach isn't accurate enough and can be problematic in handling systems with a large dimension of state space, which leads to an increase in finding non-stabilizable sets in the already constructed funnels after the preprocessing. However this problem can be solved by adding more iteration in the preprocessing phase and use fast and more efficient nonlinear solver like SNOPT or IPOPT.

We have demonstrated the approach using extensive simulation experiments on a quadrotor and cart-pole model. These experiments demonstrate that the approach can afford significant

advantages over other offline motion planning techniques. The main pro of this method was the computation of finite trajectories from random initial conditions to cover as much of the desired range initial conditions as possible with funnels. We first initialize the tree with the final goal region. Then, we sample randomly from our set of initial conditions. For each sample point that does not fall in a verified region, a new trajectory to the goal state were constructed. Next, we verify the trajectory by computing a funnel and add this new verified region to the tree. And then repeat this process until all sample points from the set of initial conditions we wish to cover fall in the verified region.

Finally, the two main Limitations of LQR-Trees are the time complexity it requires to generate the nominal trajectories and the open loop control this is mostly related to the dimensionality of the nonlinear system. And the reliance on linearized controller synthesis means that LQR-Trees are not directly applicable to nonlinear systems which are controllable but have uncontrollable linearizations.

Future work

The main contributions of this report project are the Incorporation of the direct collocation technique in the steering procedure of the RRT, Creation of a method in adjusting the funnels in environments with obstacles using a sampling based-method and the demonstration and validate the approach using simulation experiments. However, the models used here have a 6 dimension state space at most and the obstacles are assumed to be simple circles and the geometry of the robots were approximated relatively accurately by circle in order to avoid making the topology of the free configuration space complex; This allowed us to project the funnel onto the x-z plane (in contrast to the full configuration space of the system) and inflate this projection by the radius of the corresponding sphere. These inflated funnels were then used for collision-checking using the proposed method. For robots with complex geometries, inflating the funnel in x-z space in this manner is not an entirely straightforward operation. However, this is potentially a more promising approach than performing collision-checking against C-space obstacles since the inflation can be computed using SoS programming which can be time demanding as it depends not only on the geometry of the robot the obstacles but also the dimension of the system.

References

[1] LaValle, S. M. (2006), *Planning Algorithms*, Cambridge University Press.

[2] M. Mason, "*The mechanics of manipulation*", in Robotics and Automation. Proceedings. 1985 IEEE International Conference on, vol. 2, Mar 1985.

[3] S. M. LaValle and J. J. Kuffner, "*Randomized kinodynamic planning*,"International Journal of Robotics Research, vol. 20, no. 5.

[4] Glassman, E. and Tedrake, R. (2010), "*A Quadratic Regulator-Based Heuristic for Rapidly Exploring State Space*", Proc. IEEE Int. Conf. Robot. Autom. 2010

[5] S. Karaman and E. Frazzoli, "*Sampling-based algorithms for optimal motion planning*," International Journal of Robotics Research, June 2011

[6] Sun, H. and Farooq, M. (2002), "Note on the Generation of Random Points Uniformly Distributed in Hyper- Ellipsoids" in `Proc. Int. Conf. Inf. Fusion', pp. 489,496.

[7] Kuffner, J., Nishiwaki, K., Kagami, S., Inaba, M. and Inoue,H. (2001), "*Motion Planning for Humanoid Robots Under Obstacle and Dynamic Balance Constraints*", Proc. IEEE Conf. Robot. Autom.

[8] Burridge, R. R., Rizzi, A. A. and Koditschek, D. E. (1998),"*Sequential Composition of Dynamically Dexterous Robot Behaviors*", Int. J. Rob.

[9] P. Reist, P. Preiswerk, and R. Tedrake. "*Feedback-motion planning with simulation-based LQR-trees*". In To appear in the Proceedings of the International Conference on Robotics and Automation (ICRA), 2010.

[10] R. Murray and J. Hauser, "A *case study in approximate linearization: The acrobot example*," EECS Department, University of California, Berkeley

[11] Frank C. Park_ Kevin M. Lynch - Modern Robotics: Mechanics, Planning, and Control, Cambridge University Press (2017)