An Introduction to the Optimal Trajectory Tracking Problem with Feedback Control Designs

by

Ziyuan Sun

A research paper
presented to the University of Waterloo
in fulfillment of the
research paper requirement for the degree of
Master of Mathematics
in
Applied Mathematics

Waterloo, Ontario, Canada, 2022

Abstract

Optimal control plays a significant role in many disciplines, including robotics, systems biology, chemical engineering, and other fields. The core question of this topic is how to produce the optimal controllers and corresponding trajectories for a dynamical system and some conceivable constraints? And, if there is an existing open-loop reference trajectory, how to track it?

This research paper can be partitioned into two parts. The first part will introduce trajectory optimization, especially exact local direct methods, the widely accepted way to compute an open-loop solution to an optimal control problem. Then, the second part will introduce two logically completely different feedback control designs, the classic linear-quadratic regulator control design and the state-of-the-art abstraction-based control design, respectively, to produce closed-loop reference trajectories depending on the open-loop reference trajectory through trajectory optimization. Moreover, some comparative examples will be provided along with each part.

Acknowledgements

First and foremost. I want to express my sincere gratitude to my supervisor, Professor Jun Liu. I appreciate his altruistic spirit toward me. I can say it responsibly; he is the most influential mentor in my life and the best bellwether to take me into academia. I am fascinated by his personality, and it is fortunate to become his mentee. I will never forget his concentration and rigorous attitude toward scholarly studies in my lifelong academic career.

It is a great honor that Professor Roberto Guglielmi would like to be my examiner in his busy schedule; I am thankful for his quick response and valuable pieces of advice. I also like to thank Professor Sander Rhebergen, Professor Xinzhi Liu, Professor Hans De Sterck, Professor Sue Ann Campbell, Professor David Siegel, and all faculty members for teaching and helping me lay down a solid foundation in applied mathematics at the Unversity of Waterloo. In addition, I want to thank everyone in the hybrid system lab for bringing me happiness and profound consideration for the next stage of my life.

Besides, I would like to strongly thank Professor Sibel Alumur Alev for her advice, support, and encouragement at the lowest point during my graduate studies with her optimism. Thanks for helping me to make clear my potential and passion.

Finally, I am very grateful to my friends for their help, particularly Mehrdad Zariean and Hamidreza Arzani, and all my family.

Dedication

This is dedicated to my family.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Robotics has become a phenomenon in recent years; it is innovating and developing at a very fast pace. Robots have an absolute advantage in assisting humans in increasing working efficiency and improving workers' safety who perform those arduous and dangerous tasks. Studies [7] have shown that the market for global robotics will increase by more than \$100 billion in the next decade. Therefore, numerous institutes and researchers worldwide are actively evolving the robotics industry.

The trajectory tracking problem is an indispensable research field in robotic studies. It can be divided into two different categories [1]. The first category is the path following problem; the robot only needs to follow the feasible path from the initial to the goal point without a time limit. For this problem, the reference trajectory can be normally generated through sampling-based path planning, such as probabilistic roadmap (PRM) [17], rapidly-exploring random tree (RRT)[23], and modified rapidly-exploring random tree (RRT*)[16]; the limitation is that the created path does not consider the system dynamic (some recent works, such as [24], have figured out this point). The second category is the reference tracking problem; the robot must track the reference trajectory with the specific velocity and angular velocity in a pre-defined time horizon. Trajectory optimization is a common tool for generating a minimum cost open-loop reference trajectory, satisfying all desired constraints (dynamics, initial values, avoidance, etc.). For both types of problems, feedback control designs (i.e., position proportional-integral-derivative control (PID)[13], Linear-quadratic Regulator control(LQR)[14], model-based predictive control (MPC)[33], etc.) are available to command the robot track the trajectory. In this research paper, the reference tracking problem will be mainly focused on.

1.2 Scope of the Research Paper

The remainder of this research paper is organized as follows, trajectory optimization to generate open-loop reference trajectories with sensitivity analysis will be introduced in chapter 2, especially the exact local direct methods. In chapter 3, the solution methodology for tracking the reference trajectory by the feedback finite and infinite horizon LQR design will be presented. In chapter 4, an innovative solution according to [26] using abstraction-based control design in a tube shape workspace around the reference trajectory will be briefly discussed. Lastly, some personal perspectives and potential research angles will be provided in chapter 5.

Chapter 2

Trajectory Optimization and Applications

Trajectory optimization [4] is a field of designing the optimal open-loop trajectory which satisfies some required constraints or specifications. Figure 2.1 is the architecture diagram of the trajectory optimization family. There are three branches of exact approach trajectory optimization methods. The first branch is direct methods; they directly transfer the problem into a nonlinear program. The second branch is indirect methods; they use Pontryagin's minimum principle or calculus of variations to solve boundary value problems using the relationship between primal and dual states. These methods optimize first, then discretize. Generally, they are more accurate but challenging to solve. The last branch is based on the Hamilton-Jacobi-Bellman approach; the limitation is that the computational cost increases very quickly. This chapter will mainly focus on widely used direct methods.

2.1 Problem Definition

Trajectory optimization is used to find the optimal solution for basic problems with the reach-avoid specification. The reachability specification Σ means for the initial set $I \subseteq X$ of all initial state $x(0) \in I$, the effective trajectory $(x(0), x(t_1), x(t_2), \dots)$ exists a moment t that $x(t)$ touches the required goal set $Z \subseteq X$, where $0 < t_1 < t_2, \dots, < t$. It associates with $I, Z \subseteq X$ and the set of controller U can be expressed in the mathematical form as

$$\Sigma := \{(x, u) \in (X, U)^\infty | x(0) \in I \Rightarrow \exists_{t \in [0;\infty)} : x(t) \in Z\}. \tag{2.1.1}$$

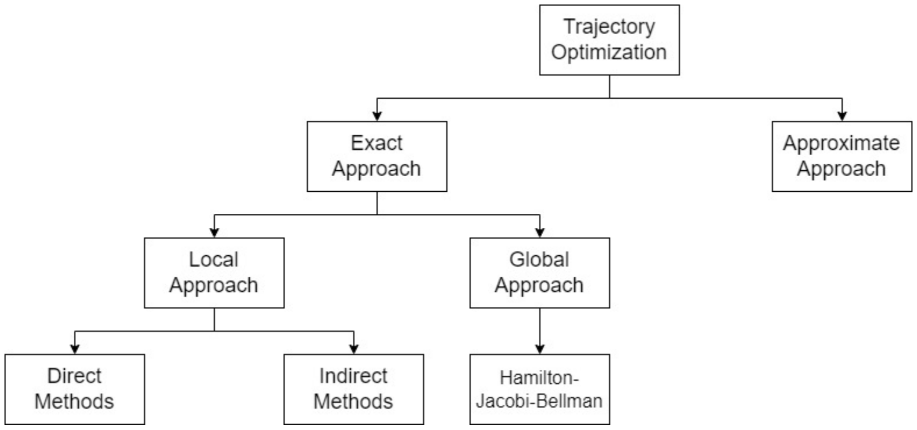

Figure 2.1: The architecture diagram of the trajectory optimization family.

The reach-avoid specification is based on the reachability specification, the states $x(t')$ on the trajectory do not touch any avoidable set $A \subseteq X$, where $t' \in [0, t)$. It associates with $I, Z, A \subseteq X$ and the set of controller U can be expressed in the mathematical form as

$$\Sigma := \{(x, u) \in (X, U)^\infty | x(0) \in I \Rightarrow \exists_{t \in [0;\infty)} : x(t) \in Z \wedge \forall_{t' \in [0;t)} : x(t') \notin A\}. \tag{2.1.2}$$

Problem (2.1.3) is a generic trajectory optimization model

$$\min_{x(t),\, u(t)} \quad m(x(t_f)) + \int_{t_0}^{t_f} \mathcal{I}(x(t), u(t))dt \tag{2.1.3a}$$

$$\text{s.t.} \quad \dot{x}(t) = f(x(t), u(t)), \qquad \forall t \in [t_0, t_f], \tag{2.1.3b}$$

$$g(x(t), u(t)) = 0, \qquad \forall t \in [t_0, t_f], \tag{2.1.3c}$$

$$h(x(t), u(t)) \geq 0, \qquad \forall t \in [t_0, t_f], \tag{2.1.3d}$$

where (2.1.3a) is the objective function; $m(.)$ is the Mayer type penalty function to guarantee that $x(t_f)$ is sufficiently close to the set goal set, and $\mathcal{I}(.,.)$ is the Lagrange type function of the instantaneous cost with respect to the pair x and u at the time $t \in [t_0, t_f]$. (2.1.3b) is the system dynamics have to follow. (2.1.3c) is the equality constraint to ensure some fixed conditions have to comply; for instance, the initial condition and the drone flocking has to keep a specific formulation. (2.1.3d) is the inequality constraint that can be used to define some conditions like the robot does not collide with an obstacle (avoiding the avoidable set), the upper and lower bound of the control input, and so forth.

2.2 Direct Methods for Trajectory Optimization

Problem (2.1.3) is challenging to solve for two reasons; the first is that it requires producing the whole optimal $x(t)$ in the appointed time horizon, and the second is that (2.1.3b) is in the form of a differential equation. The viable solution is to discretize the problem to a nonlinear optimization problem (NLP). Lagrangian relaxation is the general way to solve NLPs through decomposing constraints to obtain a new objective function. Then, using the sub-gradient method updates the Lagrangian multipliers until the upper and lower bound for the new objective function are approximately equal, and the decision variables are desired optimal values.

Three direct methods by the discretization of controls will be introduced in the rest of the section.

2.2.1 Direct Single Shooting Method

For the direct single shooting method ([8], [39]), the time horizon is divided into N pieces of equal length segments. It only discretizes controllers in the NLP, and

$$u(t) = u_k, \, \forall t \in [t_k, t_{k+1}), \tag{2.2.1}$$

is a piecewise constant function. Based on $u(t)$ to produce the corresponding discrete state $x(t)$ by using numerical integration (ode solvers). The direct single shooting NLP is as follows

$$\begin{aligned}
\min_{x_0, u_0, \ldots, u_N} \quad & \tilde{m}(x_{N+1}) + \sum_{k=0}^{N} \tilde{\mathcal{I}}(x_k, u_k) && \text{(2.2.2a)} \\
\text{s.t.} \quad & x_0 = x(t_0), && \text{(2.2.2b)} \\
& \underline{x} \leq x_k \leq \bar{x}, \qquad \forall k \in \{1, \ldots, N\}, && \text{(2.2.2c)} \\
& \underline{u} \leq u_k \leq \bar{u}, \qquad \forall k \in \{0, \ldots, N\}, && \text{(2.2.2d)} \\
& x_{N+1} = x(t_f); && \text{(2.2.2e)}
\end{aligned}$$

where (2.2.2*b*) and (2.2.2*e*) are required initial state and goal state for the problem. (2.2.2*c*) and (2.2.2*d*) are the workspace and controller set boundaries, respectively.

The compact form of the NLP (2.2.2) can be written as

$$\min_{x_0, U} \quad \tilde{J}(x_0, U) \tag{2.2.3a}$$

$$\text{s.t.} \quad \tilde{g}(x_0, U) = 0, \tag{2.2.3b}$$

$$\tilde{h}(x_0, U) \geq 0. \tag{2.2.3c}$$

The following algorithm is steps apply the direct single shooting method to solve the trajectory optimization problem.

Algorithm 1: Direct Single Shooting Method

```
1  Given $\underline{x}$, $\bar{x}$, $\underline{u}$, and $\bar{u}$ are the workspace and controller set boundaries;
2  Guess $x_0, u_0, \ldots, u_N$;
3  while The accuracy is greater than the threshold, or not exceeding the max
     iterations do
4      Initialize $\tilde{J} = 0,\ \tilde{h} = \emptyset,\ \tilde{g} = \emptyset$ ;
5      for $k = 0 : N$ do
6          $x_{k+1} := x_k + \int_{t_k}^{t_{k+1}} f(x(t), u_k) dt$;
7          $\tilde{\mathcal{I}}_k := \int_{t_k}^{t_{k+1}} \tilde{\mathcal{I}}(x(t), u_k) dt$;
8          $\tilde{J} = \tilde{J} + \tilde{I}_k$;
9          $\tilde{h}.append([x_k - \underline{x}; \bar{x} - x_k])$;
10         $\tilde{h}.append([u_k - \underline{u}; \bar{u} - x_u])$;
11     $\tilde{J} = \tilde{J} + \tilde{m}(x_{N+1})$;
12     $\tilde{g} = [x_0 - x(t_0); x_{N+1} - x(t_f)]$;
13     Solve the NLP based on the prepared $\tilde{J}$, $\tilde{g}$, and $\tilde{h}$;
14     Update $x_0, u_0, \ldots, u_N$.
```

2.2.2 Direct Collocation Method

For the direct collocation method ([18], [27]), the time horizon is divided into N pieces of equal length segments. In different with the direct single shooting method, it also discretizes states. States between two discretized time points are further divided into D equidistant discretized sub-points, and each time point can be written as a piecewise polynomial function by sub-points to represent every interval as

$$x_k^D(t) = \sum_{j=0}^{D} p_j(\tau) x_{kj}, \tag{2.2.4}$$

where $\forall k \in \{0, \dots, N\}, j \in \{0, \dots, D\}$. And, initially guessing all states x_{kj}.

The direct collocation NLP is as follow

$$\begin{aligned}
\min_{X_0, \dots, X_{N+1}, u_0, \dots, u_N} \quad & \tilde{m}(x_{N(D+1)+N}) + \sum_{k=0}^{N} \tilde{\mathcal{I}}_k(X_k, u_k) && && \text{(2.2.5a)} \\
\text{s.t.} \quad & x_0 = x(t_0), && && \text{(2.2.5b)} \\
& c_k(X_k, u_k) = 0, && \forall k \in \{0, \dots, N\}, && \text{(2.2.5c)} \\
& \underline{x} \le x_k \le \bar{x}, && \forall k \in \{1, \dots, N\}, && \text{(2.2.5d)} \\
& \underline{u} \le u_k \le \bar{u}, && \forall k \in \{0, \dots, N\}, && \text{(2.2.5e)} \\
& x_{N(D+1)+N} = x(t_f), && && \text{(2.2.5f)}
\end{aligned}$$

where X_k is the set $\{x_{k1}, \dots, x_{kD}\}$, and $\tilde{\mathcal{I}}_k(X_k, u_k)$ is the cost between t_k and t_{k+1}. (2.2.5c) is the constraint of the gap between the actual $\dot{x}$ and the system dynamic, $\dot{x}$ can be estimated by using the secant method, as follows

$$\begin{aligned}
c_k(X_k, u_k) &\approx \dot{x}(t) - f(x(t), u(t)) \\
&\approx \frac{x_{k+1} - x_k}{t_{k+1} - t_k} - f(x_{k+1}, u_k).
\end{aligned} \tag{2.2.6}$$

The compact form of the NLP (2.2.5) can be written as

$$\begin{aligned}
\min_{X, U} \quad & \tilde{J}(x_0, U) && \text{(2.2.7a)} \\
\text{s.t.} \quad & \tilde{g}(x_0, U) = 0, && \text{(2.2.7b)} \\
& \tilde{h}(x_0, U) \ge 0. && \text{(2.2.7c)}
\end{aligned}$$

The following algorithm is steps to apply the direct collocation method to solve the trajectory optimization problem.

Algorithm 2: Direct Collocation Method

```
1  Given $\underline{x}$, $\bar{x}$, $\underline{u}$, and $\bar{u}$ are the workspace and controller set boundaries;
2  Guess $x_0, \dots, x_{N(D+1)+N}, u_0, \dots, u_N$;
3  while The accuracy is greater than the threshold, or not exceeding the max iterations do
4      Initialize $\tilde{J} = 0,\ \tilde{h} = \emptyset,\ \tilde{g} = \emptyset$ ;
5      for $k = 0 : N$ do
6          $\tilde{g}.append(c_k(X_k, u_k))$;
7          $\tilde{J} = \tilde{J} + \tilde{\mathcal{I}}_k(X_k, u_k)$;
8          $\tilde{h}.append([X_k - \underline{x}; \bar{x} - X_k])$;
9          $\tilde{h}.append([u_k - \underline{u}; \bar{u} - x_u])$;
10     $\tilde{J} = \tilde{J} + \tilde{m}(x_{N(D+1)+N})$;
11     $\tilde{g} = [x_0 - x(t_0); x_{N(D+1)+N} - x(t_f)]$;
12     Solve the NLP based on the prepared $\tilde{J}$, $\tilde{g}$, and $\tilde{h}$;
13     Update $x_0, \dots, x_{N(D+1)+N}, u_0, \dots, u_N$.
```

2.2.3 Direct Multiple Shooting Method

For the multiple shooting method [6], the time horizon is divided into N pieces of equal length segments. Similar to the direct collocation method, the direct multiple shooting method discretizes both controllers and states. However, instead of using the piecewise polynomial and guessing all states x_{kj} to represent every interval; it locally discretizes the states x_k by numerical integration like the direct single shooting method. The multiple shooting method can be imagine as solving N times of the direct single shooting method, but the terminal condition for the k shooting point is exact the next shooting method's initial condition.

The direct multiple shooting NLP is as follows:

$$\begin{aligned}
\min_{\substack{x_0,\ldots,x_{N+1},\\ u_0,\ldots,u_N}} \quad & \tilde{m}(x_{N+1}) + \sum_{k=0}^{N} \tilde{\mathcal{I}}_k(x_k, u_k) && && \text{(2.2.8a)}\\
\text{s.t.} \quad & x_0 = x(t_0), && && \text{(2.2.8b)}\\
& x_{k+1} = x_k + \int_{t_k}^{t_{k+1}} f(x(t), u_k)dt, && \forall k \in \{0, \ldots, N\}, && \text{(2.2.8c)}\\
& \underline{x} \leq x_k \leq \bar{x}, && \forall k \in \{1, \ldots, N\}, && \text{(2.2.8d)}\\
& \underline{u} \leq u_k \leq \bar{u}, && \forall k \in \{0, \ldots, N\}, && \text{(2.2.8e)}\\
& x_{N+1} = x(t_f), && && \text{(2.2.8f)}
\end{aligned}$$

where $\tilde{\mathcal{I}}(x_k, u_k)$ is the cost between t_k and t_{k+1}.

The compact form of the NLP (2.2.8) can be written as

$$\begin{aligned}
\min_{X,U} \quad & \tilde{J}(X, U) && \text{(2.2.9a)}\\
\text{s.t.} \quad & \tilde{g}(X, U) = 0, && \text{(2.2.9b)}\\
& \tilde{h}(X, U) \geq 0. && \text{(2.2.9c)}
\end{aligned}$$

The following algorithm is steps to apply the direct multiple shooting method to solve the trajectory optimization problem.

Algorithm 3: Direct Multiple Shooting Method

```
1  Given $\underline{x}$, $\bar{x}$, $\underline{u}$, and $\bar{u}$ are the workspace and controller set boundaries;
2  Guess $x_0, \ldots, x_{N+1}, u_0, \ldots, u_N$;
3  while The accuracy is greater than the threshold, or not exceeding the max iterations do
4      Initialize $\tilde{J} = 0$, $\tilde{h} = \emptyset$, $\tilde{g} = \emptyset$ ;
5      for $k = 0 : N$ do
6          $r_{k+1} := x_{k+1} - x_k - \int_{t_k}^{t_{k+1}} f(x(t), u_k)dt$;
7          $\tilde{g}.append([r_k])$;
8          $\tilde{\mathcal{I}}_k := \int_{t_k}^{t_{k+1}} \tilde{\mathcal{I}}(x(t), u_k)dt$;
9          $\tilde{J} = \tilde{J} + \tilde{\mathcal{I}}_k(x_k, u_k)$;
10         $\tilde{h}.append([x_k - \underline{x}; \bar{x} - x_k])$;
11         $\tilde{h}.append([u_k - \underline{u}; \bar{u} - x_u])$;
12     $\tilde{J} = \tilde{J} + \tilde{m}(x_{N+1})$;
13     $\tilde{g} = [x_0 - x(t_0); x_{N+1} - x(t_f)]$;
14     Solve the NLP based on the prepared $\tilde{J}$, $\tilde{g}$, and $\tilde{h}$;
15     Update $x_0, \ldots, x_{N+1}, u_0, \ldots, u_N$.
```

2.3 Computational Experiments

The direct single shooting method is not decoupled, which means it can only simulate the problem sequentially. It is time consuming; therefore, computational experiments only use and compare the direct collocation method and the direct multiple shooting method in the rest of the section. Both methods are helped with an open-source optimization toolbox in MATLAB call "OptimTraj" ([18], [19]).

All numerical experiments have been run on Intel i7-8565U processor, limited to 16GB of RAM. OptimTraj has access to 4 physical cores, max turbo frequency of 4.60GHz, and using up to 8 threads.

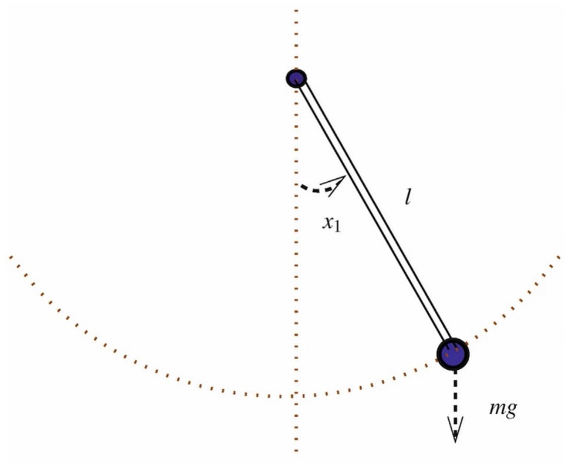

Figure 2.2: The schematic of the simple pendulum system from [40].

2.3.1 Simple Pendulum System

Figure 2.2 is a schematic of a simple pendulum. The system can be expressed as

$$\begin{aligned}\dot{x}_1 &= x_2 \\ \dot{x}_2 &= -\frac{mgl}{I}\sin x_1 - \frac{b}{I}x_2 + \frac{1}{I}u,\end{aligned} \tag{2.3.1}$$

where x_1 is the angle between the rod and the vertical axis through the pivot point, x_2 is the angular velocity, l is the length of the rod without mass, m is the mass of the ball, g is the acceleration of gravity, b is the frictional coefficient of the joint, and $I = ml^2$. Control input u is the force to the pendulum. In the computational experiment, Table 2.1 indicates parameters for the simple pendulum, and $-5 \leq u \leq 5$ is bounds for control inputs.

Parameter	Value (unit)
m	1 (kg)
l	0.5 (m)
b	0.1
g	9.8 (m/s^2)

Table 2.1: Parameters for the simple pendulum.

The first simple pendulum experiment begins with the rod from the vertical down position without the angular velocity ($x_1(0) = 0$ and $x_2(0) = 0$) and ends with the rod

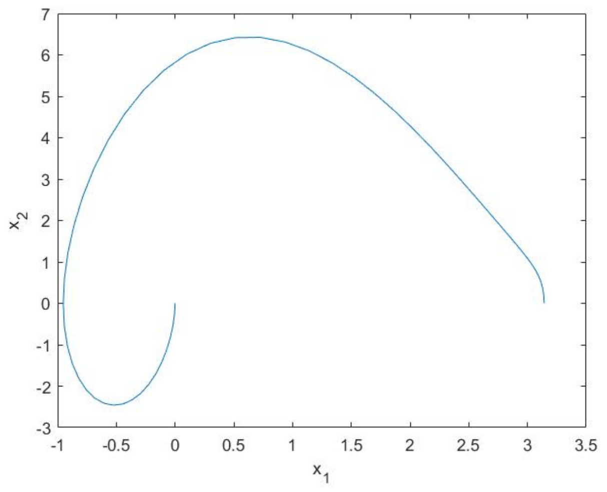

Figure 2.3: Swing upright the simple pendulum from the vertical down position by the direct collocation method.

inverted upright position without the angular velocity ($x_1(t_f) = \pi$ and $x_2(t_f) = 0$). Figure 2.3 and 2.4 show the resulting trajectories of swinging upright the pendulum from the vertical down position by the direct collocation method and the direct multiple shooting method, respectively. Figure 2.5 and 2.6 are corresponding open-loop control inputs versus time. Intuitively, the results are almost identical. Table 2.2 makes a comparison between two methods. As for attaining the same accuracy, the direct collocation method is always quicker than the direct multiple shooting method; the reason behind this is the direct multiple shooting method has to call the time consuming ode solver for every interval.

Method	N	D	t_f	CPU Time (s)
Collocation	20	3	1.9514	2.01
Multiple Shooting	20	-	1.9514	10.13

Table 2.2: The comparison between two methods for the pendulum experiment one.

The second simple pendulum experiment considers the rod in motion, which does not begin from the vertical down position. Assuming that the beginning state is $x_1(0) = -0.5$ with the angular velocity $x_2(0) = 3$, and the ending position is inverted upright without the angular velocity ($x_1(t_f) = \pi$ and $x_2(t_f) = 0$). Figure 2.7 and 2.8 show the resulting trajectories. Figure 2.9 and 2.10 are corresponding open-loop control inputs

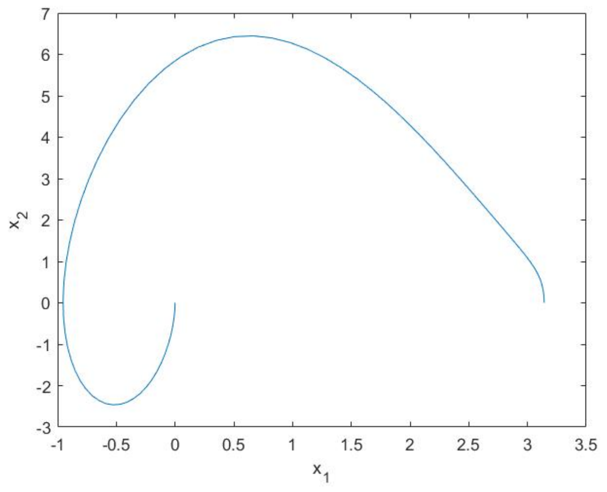

Figure 2.4: Swing upright the simple pendulum from the vertical down position by the direct multiple shooting method.

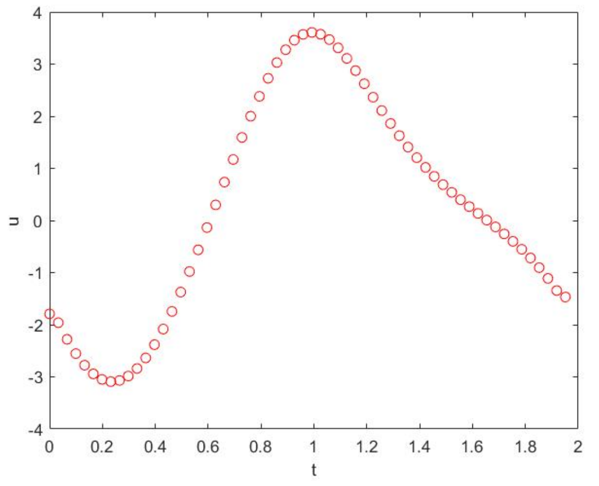

Figure 2.5: Controller versus time for swinging upright the simple pendulum from the vertical down position by the direct collocation method.

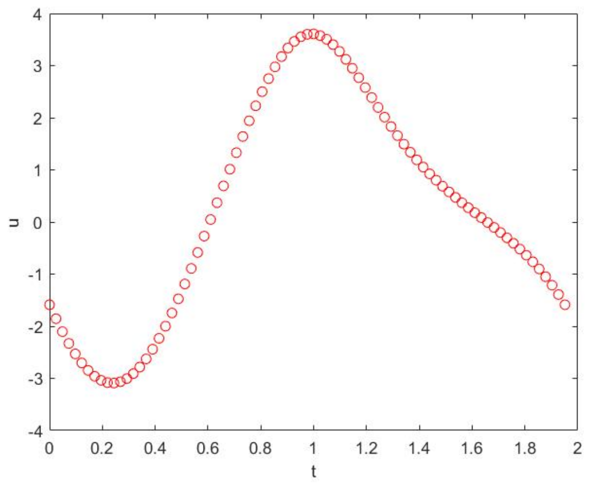

Figure 2.6: Controller versus time for swinging upright the simple pendulum from the vertical down position by the direct multiple shooting method.

versus time. Trajectories and control inputs are also almost identical under two different methods. Table 2.3 compares two methods. In this experiment, the direct collocation method is much quicker than the direct multiple shooting method. It follows that the direct collocation method is a more practical method to deal with local exact trajectory optimization problems.

Method	N	D	t_f	CPU Time (s)
Collocation	20	3	2.5444	3.62
Multiple Shooting	20	-	2.5444	30.54

Table 2.3: The comparison between two methods for the pendulum experiment two.

2.3.2 Three-dimensional Nonlinear System

Consider the three-dimensional nonlinear system, as follows

$$\begin{aligned} \dot{x}_1 &= -\cos(u) + \sin(x_3) + 1 \\ \dot{x}_2 &= \sin(u) + sin(x_1) \\ \dot{x}_3 &= \exp(x_2) + u - 1. \end{aligned} \tag{2.3.2}$$

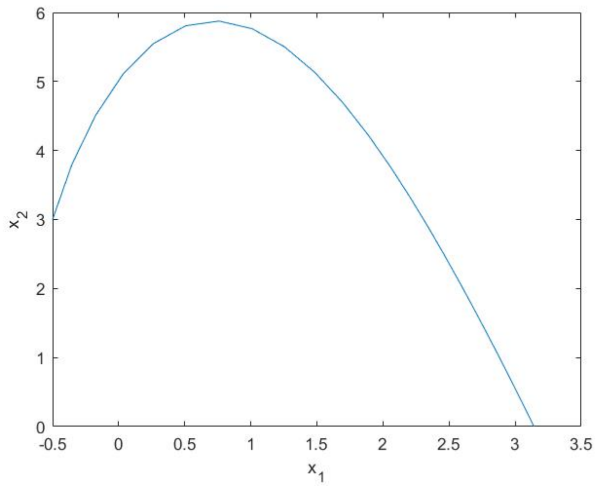

Figure 2.7: Swing upright the simple pendulum from the $x_1 = -0.5$ (radian) position by the direct collocation method.

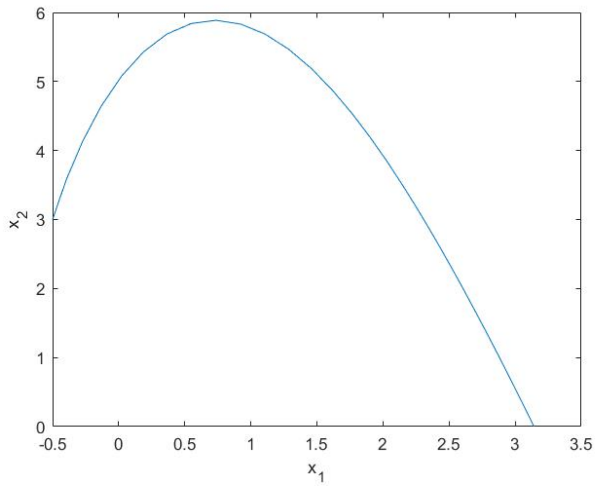

Figure 2.8: Swing upright the simple pendulum from the $x_1 = -0.5$ (radian) position by the direct multiple shooting method.

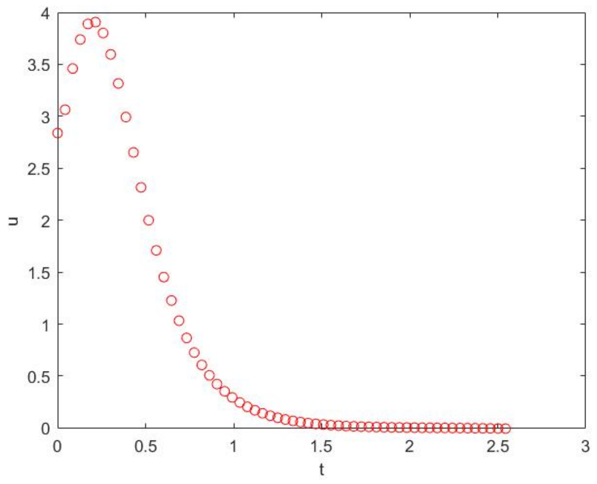

Figure 2.9: Controller versus time for swinging upright the simple pendulum from the $x_1 = -0.5$ (radian) position by the direct collocation method.

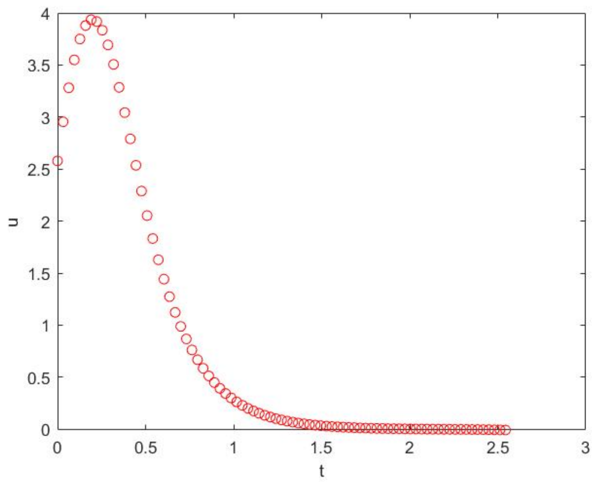

Figure 2.10: Controller versus time for swinging upright the simple pendulum from the $x_1 = -0.5$ (radian) position by the direct multiple shooting method.

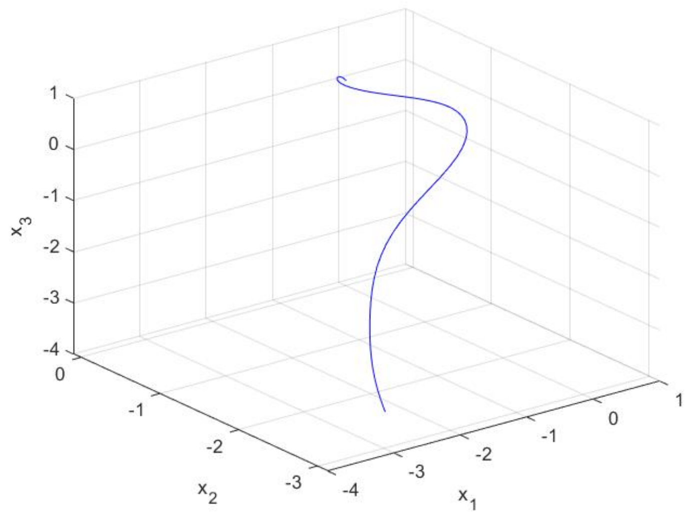

Figure 2.11: Control the three-dimensional nonlinear system from $(-\pi, -\pi, -\pi)$ to $(0, 0, 0)$ by the direct collocation method.

The goal is to find the optimal controller that the initial state is $(-\pi, -\pi, -\pi)$ and the terminal state is $(0, 0, 0)$. Figure 2.11 and 2.12 are trajectories simulated by the direct collocation method and the direct multiple shooting method, respectively. Figure 2.13 and 2.14 are corresponding controllers versus time. Again, both methods produced almost identical trajectories and control inputs. Table 2.4 shows that the direct collocation method is approximate three times faster than the direct multiple shooting method for the same time horizon and accuracy.

Method	N	D	t_f	CPU Time (s)
Collocation	20	3	11.3	4.71
Multiple Shooting	20	-	11.3	12.60

Table 2.4: The comparison between two methods for the three-dimensional nonlinear system.

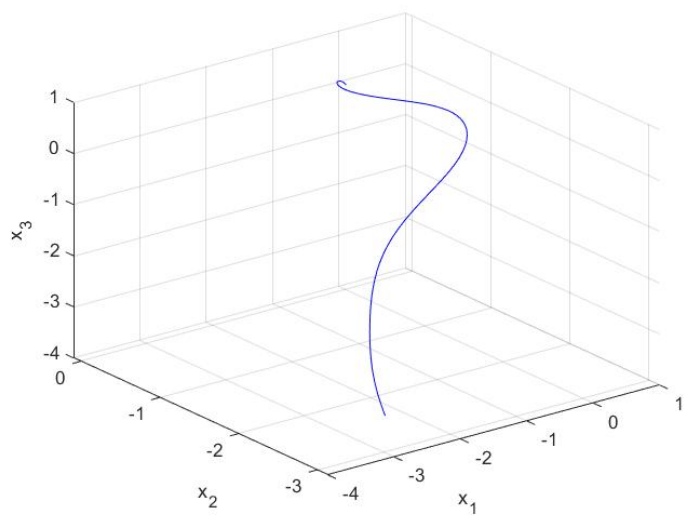

Figure 2.12: Control the three-dimensional nonlinear system from $(-\pi, -\pi, -\pi)$ to $(0, 0, 0)$ by the direct multiple shooting method.

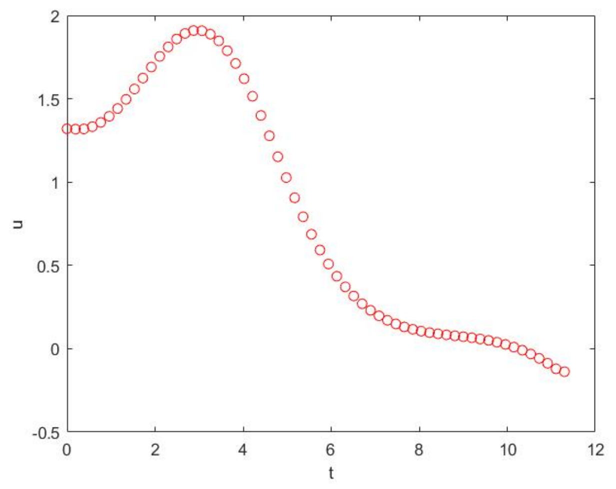

Figure 2.13: Controller versus time for the three-dimensional nonlinear system from $(-\pi, -\pi, -\pi)$ to $(0, 0, 0)$ by the direct collocation method.

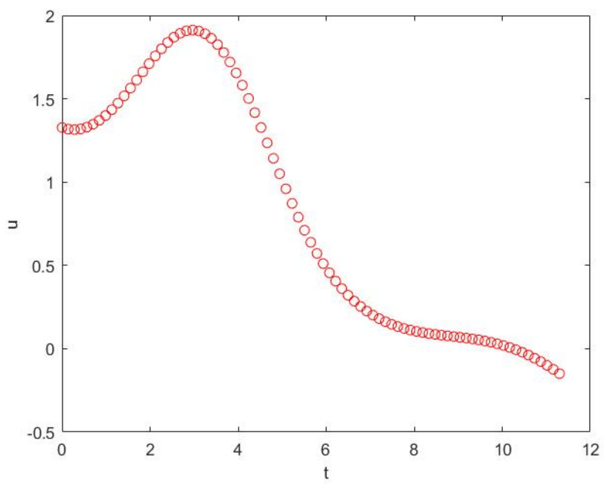

Figure 2.14: Controller versus time for the three-dimensional nonlinear system from $(-\pi, -\pi, -\pi)$ to $(0, 0, 0)$ by the direct multiple shooting method.

Chapter 3

Linear-Quadratic Regulator Control Design

Linear-quadratic Regulator Controller (LQR) [14] was first presented in the 1960s, which is a well-known control method that can provide optimally controlled feedback (closed-loop) gains to actualize the system's stable and high-performance design. Nowadays, it is widely used in various fields.

3.1 Stability and Lyapunov Analysis

Stability analysis is a very important factor in studying the closed-loop system; it can demonstrate that the candidate controller is attainable for specific purposes.

Consider a system

$$\dot{x} = f(x, t). \tag{3.1.1}$$

Let x^* be the equilibrium point, which means $f(x^*, t) = 0$. Suppose in the rest of this section, the origin is one of the equilibrium points. The followings are some important stability concepts based on the book [34].

Definition 3.1.1 (Stable)**.** The origin is stable if for any $\delta > 0$, there exists $\epsilon > 0$, such that if $\|x(0)\| < \epsilon$, then $\|x(t)\| < \delta$ for all $t \geq 0$. Generally speaking, the initial condition starts close to the origin, then it stays close to the origin for all time.

Definition 3.1.2 (Asymptotically Stable)**.** The origin is stable, and if, in addition, there exists some $\epsilon > 0$ such that $\|x(0)\| < \epsilon$ implies that $x(t)$ approaches the origin as t going to infinity. Generally speaking, the initial condition not only has to stay close to the origin, but also approaches to the origin as t going to infinity.

The following Theorem 3.1.1 is the most basic theorem to analyze systems' stability around equilibrium points.

Theorem 3.1.1 (Lyapunov Theorem)**.** *In a region D, let $V(x)$ is a scalar, continuously-first partial derivative function such that*

1.
$$V(0) = 0; \tag{3.1.2}$$

2.
$$V(x) > 0,\ \textit{which is positive definite},\ \forall x \in D\backslash\{0\}; \tag{3.1.3}$$

3.
$$\dot{V}(x) \leq 0, \textit{which is negative semi-definite},\ \forall x \in D; \tag{3.1.4}$$

then the origin is stable. Further on, if the others remain the same, and the derivative of the function
$$\dot{V}(x) < 0,\ \textit{which is negative definite},\ \forall x \in D; \tag{3.1.5}$$
then the origin is asymptotically stable.

Proof. Here is a brief proof. Given $\epsilon > 0$, and $0 < r \leq \epsilon$ such that

$$\mathbf{B}_r = \{\|x\| \leq r\} \subset D. \tag{3.1.6}$$

Let $\alpha = \min_{\|x\|=r} V(x) > 0$. Taking the maximum level surface $\beta \in (0, \alpha)$, there is an interior set of $\mathbf{B}_r$
$$\Omega_\beta = \{x \in \mathbf{B}_r \mid V(x) \leq \beta\}. \tag{3.1.7}$$

Suppose that $\mathbf{B}_\delta$ is the subset in Ω_β, which satisfies

$$\|x\| \leq \delta \Longrightarrow V(x) < \beta. \tag{3.1.8}$$

As the result of Equation (3.1.4), $\dot{V}(x) \leq 0$ in Ω_β; there are

$$x(0) \in \mathbf{B}_\delta \Rightarrow x(0) \in \Omega_\beta \Rightarrow x(t) \in \Omega_\beta \Rightarrow x(t) \in \mathbf{B}_r, \tag{3.1.9}$$

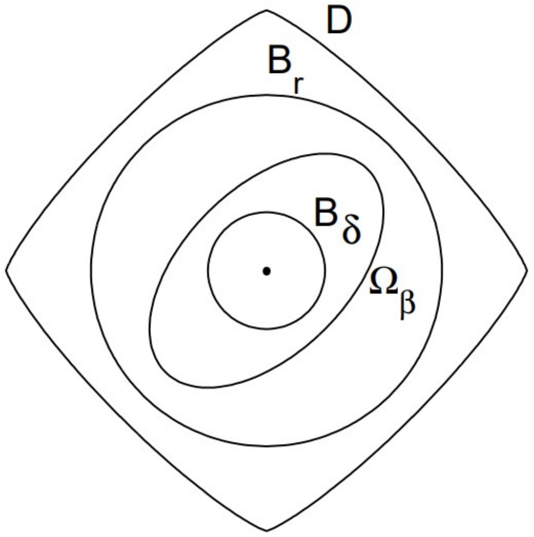

Figure 3.1: The framework in the proof of Theorem 3.1.1
from [21]

and

$$\|x(0)\| < \delta \Rightarrow \|x(t)\| < r < \epsilon, \, \forall t \geq 0. \tag{3.1.10}$$

Therefore, the origin is stable by the definition. Moreover, if $\dot{V}(x) < 0$, $\forall x \in D/\{0\}$. Then $V(x(t))$ is monotonically decreasing, which has

$$\lim_{t\to\infty} V(x(t)) = c \geq 0. \tag{3.1.11}$$

Suppose for the level surface $c > 0$, there exists $\mathbf{B}_d \subset \Omega_c$. $\lim_{t\to\infty} = c > 0$ means there is a T, have $x(t)$ lies outside $\mathbf{B}_d$ for all $t > T$. Let $\gamma = -\max_{d\leq\|x\|\leq r} \dot{V}(x)$, where $\gamma > 0$, the scalar, continuously-first partial derivative function can produce

$$V(x(t)) = V(x(0)) + \int_0^t \dot{V}(x(\tau))d\tau \leq V(x(0)) - \gamma t < 0, \tag{3.1.12}$$

which is contradiction with the condition 2, $V(x) > 0$. Therefore, c can only be 0, and the origin is asymptotically stable by the definition. □

3.2 Hamilton-Jacobi-Bellman Equation with LQR

3.2.1 Hamilton-Jacobi-Bellman Equation

In optimal control, the goal is to search for an optimal control policy $\mathbf{u}^*$ and plug in this control input into the system $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))$ to obtain such trajectory $\mathbf{x}^*$ on a continuous-time interval $t \in [0, T]$ (assuming that the initial time is 0) that reaches to the minimum value J^* of an objective function:

$$J(t, \mathbf{x}) = h(\mathbf{x}(T)) + \int_0^T l(\mathbf{x}(t), \mathbf{u}(t))dt. \tag{3.2.1}$$

In the objective function, $h(\mathbf{x}(T))$ is the terminal cost, and $l(\mathbf{x}(t), \mathbf{u}(t))$ is the instantaneous cost at the moment t. T can be both finite and infinite. When T is finite, the problem is called a finite horizon problem, which means to take an integral of the instantaneous cost from 0 to a finite value of T. On another hand, the problem is called an infinite horizon problem when T is infinite; it is not time-dependent, and there is no terminal cost, $h = 0$,

$$J(\mathbf{x}) = \int_0^\infty l(\mathbf{x}(t), \mathbf{u}(t))dt. \tag{3.2.2}$$

The infinite horizon problem is a sub-class of the finite horizon problem; it only needs some minor changes to transfer between each other. For easy calculation by computer, the system can be written in the difference equation, and Equation (3.2.1) can be estimated in a discrete version, as follow:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \delta\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k), \tag{3.2.3}$$

$$\tilde{J} = h(\mathbf{x}_N) + \sum_{k=0}^{N-1} \delta l(\mathbf{x}_k, \mathbf{u}_k), \tag{3.2.4}$$

where $\delta = \frac{T}{N}$ is the unit length of a piece of time (there are total N pieces) and $k \in \{0, \ldots, N\}$.

To satisfy the soft constraint in the objective function, algorithms to solve this optimization problem start the time from the terminal T backward. At the any discrete time point $t = k\delta$, the optimal value of the objective function is

$$\tilde{J}^*(k\delta, \mathbf{x}) = \min_{\mathbf{u} \in U}\{\delta l(\mathbf{x}, \mathbf{u}) + \tilde{J}^*(\delta k + \delta, \mathbf{x} + \delta\mathbf{f}(\mathbf{x}, \mathbf{u}))\}; \tag{3.2.5}$$

by using the Taylor series expansion to the second term to get

$$\begin{aligned}\tilde{J}^*(k\delta,\mathbf{x}) = \min_{\mathbf{u}\in U}\{&\delta l(\mathbf{x},\mathbf{u}) + \tilde{J}^*(k\delta,\mathbf{x}) + \delta\nabla_t\tilde{J}^*(k\delta,\mathbf{x}) \\ &+\delta\nabla_{\mathbf{x}}\tilde{J}^{*\prime}(k\delta,\mathbf{x})\mathbf{f}(\mathbf{x},\mathbf{u}) + o(\delta)\}.\end{aligned} \tag{3.2.6}$$

After simplifying and ignoring high order terms, becomes

$$0 = \min_{\mathbf{u}\in U}\{l(\mathbf{x},\mathbf{u}) + \nabla_t\tilde{J}^*(k\delta,\mathbf{x}) + \nabla_{\mathbf{x}}\tilde{J}^*(k\delta,\mathbf{x})\mathbf{f}(\mathbf{x},\mathbf{u})\}; \tag{3.2.7}$$

assuming that δ is extremely small, close to 0, and k approaches infinity, the discrete equation can transfer back to the continuous equation, as follows

$$0 = \min_{\mathbf{u}\in U}\{l(\mathbf{x},\mathbf{u}) + \nabla_t J^*(t,\mathbf{x}) + \nabla_{\mathbf{x}} J^*(t,\mathbf{x})\mathbf{f}(\mathbf{x},\mathbf{u})\}, \tag{3.2.8}$$

Equation (3.2.8) is the well-known Hamilton-Jacobi-Bellman (HJB) equation, [9] is a book chapter comprehensively introduce this equation, and

$$\pi^*(t,\mathbf{x}) = \arg\min_{\mathbf{u}\in U}\{l(\mathbf{x},\mathbf{u}) + \nabla_t J^*(t,\mathbf{x}) + \nabla_{\mathbf{x}} J^*(t,\mathbf{x})\mathbf{f}(\mathbf{x},\mathbf{u})\} \tag{3.2.9}$$

is the optimal control policy.

The following Theorem 3.2.1 from the chapter 7 of the book by [3] to shows how the solution to the HJB equation equals the optimal objective function for all t and $\mathbf{x}$.

Theorem 3.2.1 (Hamilton-Jacobi-Bellman Sufficiency Theorem)**.** *Consider the system* $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x},\mathbf{u})$*, suppose* $V(t,\mathbf{x})$ *is continuously differentiable in* t *and* $\mathbf{x}$*, and is such that it is the solution of Equation (3.2.8),*

$$0 = \min_{\mathbf{u}\in U}\{l(\mathbf{x},\mathbf{u}) + \nabla_t V(t,\mathbf{x}) + \nabla_{\mathbf{x}} V(t,\mathbf{x})\mathbf{f}(\mathbf{x},\mathbf{u})\},\ \forall t,\ \mathbf{x}, \tag{3.2.10}$$

$$V(T,\mathbf{x}) = h(\mathbf{x}(T)),\ \forall\mathbf{x}. \tag{3.2.11}$$

Suppose also that $\mathbf{u}^*(t,\mathbf{x})$ *is the optimal control input in Equation (3.2.10). Then* $V(t,\mathbf{x})$ *is exact the optimal objective function for all initial time* $t\in[0,T]$ *and any initial state* $\mathbf{x}$*.*

Proof. Let $\hat{u}(t)\in U$ is any available control input for $t\in[t',T]$, where $t'\in[0,T]$, and $\hat{x}(t)$ is the corresponding state. Then,

$$0 \le l(\hat{\mathbf{x}}(t),\hat{\mathbf{u}}(t)) + \nabla_t V(t,\hat{\mathbf{x}}(t)) + \nabla_{\mathbf{x}} V(t,\hat{\mathbf{x}})\mathbf{f}(\hat{\mathbf{x}}(t),\hat{\mathbf{u}}(t)) \tag{3.2.12}$$

since the instantaneous cost and the objective function are always non-negative.

Given that $\dot{\hat{\mathbf{x}}}(t) = \mathbf{f}(\hat{\mathbf{x}}(t), \hat{\mathbf{u}}(t))$, to obtain

$$\begin{aligned} 0 &\leq l(\hat{\mathbf{x}}(t), \hat{\mathbf{u}}(t)) + \nabla_t V(t, \hat{\mathbf{x}}(t)) + \nabla_{\mathbf{x}} V(t, \hat{\mathbf{x}}(t))\dot{\hat{\mathbf{x}}}(t) \\ &= l(\hat{\mathbf{x}}(t), \hat{\mathbf{u}}(t)) + \frac{d}{dt}(V(t, \hat{\mathbf{x}}(t))). \end{aligned} \tag{3.2.13}$$

Integrating the inequality from $t' = 0$ to T, to obtain

$$\begin{aligned} 0 &\leq \left(\int_0^T l(\hat{\mathbf{x}}(t), \hat{\mathbf{u}}(t))dt\right) + \left(\int_0^T \frac{d}{dt}(V(t, \hat{\mathbf{x}}(t)))dt\right) \\ &= \left(\int_0^T l(\hat{\mathbf{x}}(t), \hat{\mathbf{u}}(t))dt\right) + V(T, \hat{\mathbf{x}}(T)) - V(0, \hat{\mathbf{x}}(0)). \end{aligned} \tag{3.2.14}$$

The initial state is always provided, hence $\hat{\mathbf{x}}(0) = \mathbf{x}(0)$. Also, the terminal condition Equation (3.2.11) should be followed, to yield

$$V(0, \mathbf{x}(0)) = V(0, \hat{\mathbf{x}}(0)) \leq h(\hat{\mathbf{x}}(T)) + \left(\int_0^T l(\hat{\mathbf{x}}(t), \hat{\mathbf{u}}(t))dt\right). \tag{3.2.15}$$

To select $\hat{\mathbf{x}}(t)$ and $\hat{\mathbf{u}}(t)$ are optimal control input $\mathbf{x}^*(t)$ and its corresponding $\mathbf{x}^*(t)$, respectively. The objective function can approach to the minimum bound $J^*(0, \mathbf{x}(0))$

$$V(0, \mathbf{x}(0)) = h(\mathbf{x}^*(T)) + \left(\int_0^T l(\mathbf{x}^*(t), \mathbf{u}^*(t))dt\right) = J^*(0, \mathbf{x}(0)). \tag{3.2.16}$$

Without loss of generality, t' can be any value between 0 and T to take an integral from; thus,

$$V(t, \mathbf{x}) = J^*(t, \mathbf{x}), \ \forall t, x. \tag{3.2.17}$$

□

3.2.2 Hamilton-Jacobi-Bellman Equation with LQR

Consider a linear system at the state equation

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t). \tag{3.2.18}$$

Let the objective function in the quadratic form, which is widely used in engineering, as follows

$$l(\mathbf{x}, \mathbf{u}) = \mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}, \tag{3.2.19}$$

$$h(\mathbf{x}) = \mathbf{x}^T\mathbf{Q}_T\mathbf{x}, \tag{3.2.20}$$

where $\mathbf{Q} = \mathbf{Q}^T$ is positive semi-definite, $\mathbf{R} = \mathbf{R}^T$ is positive definite, and $\mathbf{Q}_T$ is positive definite to ensure $J^* > 0$. Then, Equation (3.2.8) becomes

$$0 = \min_{\mathbf{u}}\{\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + \nabla_t J^*(t, \mathbf{x}) + \nabla_{\mathbf{x}} J^{*\prime}(t, \mathbf{x})(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u})\}; \tag{3.2.21}$$

assuming that the solution of the optimal objective function is also the quadratic form

$$J^*(t, \mathbf{x}) = \mathbf{x}^T\mathbf{S}(t)\mathbf{x}, \tag{3.2.22}$$

where $\mathbf{S}(t)$ is positive definite since the objective function is always positive, to yield

$$0 = \min_{\mathbf{u}}\{\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + \mathbf{x}^T\dot{\mathbf{S}}(t)\mathbf{x} + 2\mathbf{x}^T\mathbf{S}(t)\mathbf{A}\mathbf{x} + 2\mathbf{x}^T\mathbf{S}(t)\mathbf{B}\mathbf{u}\}. \tag{3.2.23}$$

The optimal control input $\mathbf{u}^*$ can be achieved by taking a partial derivative of u since terms inside the minimum is convex, which means any local minimum is also a global minimum

$$\frac{\partial}{\partial \mathbf{u}}\{\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u} + \mathbf{x}^T\dot{\mathbf{S}}(t)\mathbf{x} + 2\mathbf{x}^T\mathbf{S}(t)\mathbf{A}\mathbf{x} + 2\mathbf{x}^T\mathbf{S}(t)\mathbf{B}\mathbf{u}\} = 0 \tag{3.2.24}$$

$$\Longrightarrow \mathbf{u}^* = -\mathbf{K}(t)\mathbf{x} = -\mathbf{R}^{-1}\mathbf{B}\mathbf{S}(t)\mathbf{x}. \tag{3.2.25}$$

Substituting the optimal control input $\mathbf{u}^*$ back to Equation (3.2.23) to obtain

$$0 = \mathbf{x}^T[\mathbf{Q} + \mathbf{S}(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}(t) + \dot{\mathbf{S}}(t) + \mathbf{S}(t)\mathbf{A} + \mathbf{A}^T\mathbf{S}(t) - 2\mathbf{S}(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}(t)]\mathbf{x} \tag{3.2.26}$$

$$\Longrightarrow \dot{\mathbf{S}}(t) = -\mathbf{S}(t)\mathbf{A} - \mathbf{A}^T\mathbf{S}(t) + \mathbf{S}(t)\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}(t) - \mathbf{Q}, \tag{3.2.27}$$

which is the famous Riccati equation. The equation is actually an ordinary differential equation (ODE), and it can be easily solved to get a continuous solution $\mathbf{S}$(t) if given the eligible terminal condition

$$\mathbf{S}(T) = \mathbf{Q}_T; \tag{3.2.28}$$

for procuring the optimal solution of the linear-quadratic problem, the existence of the solution to Equation (3.2.27) is always the necessary and sufficient condition [35].

In addition, if the problem is the infinite time horizon, Equation (3.2.8) does not include the term with respect to t, as follows

$$0 = \min_{\mathbf{u} \in U} \{l(\mathbf{x}, \mathbf{u}) + \nabla_{\mathbf{x}} J^{*\prime}(\mathbf{x}) \mathbf{f}(\mathbf{x}, \mathbf{u})\}. \tag{3.2.29}$$

Assuming that the optimal objective function is the quadratic form only with respect to $\mathbf{x}$

$$J^*(\mathbf{x}) = \mathbf{x}^T \mathbf{S} \mathbf{x}; \tag{3.2.30}$$

it is easy to obtain the optimal control input and an algebraic Riccati equation by following similar steps

$$\mathbf{u}^* = -\mathbf{R}^{-1} \mathbf{B} \mathbf{S} \mathbf{x}, \tag{3.2.31}$$

$$0 = \mathbf{S}\mathbf{A} + \mathbf{A}^T \mathbf{S} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S} + \mathbf{Q}. \tag{3.2.32}$$

Corollary 3.2.1.1. *If Equation (3.2.32) admits a positive definite solution* $\mathbf{S}$*, then the origin for this control system is asymptotically stable.*

Proof. Assuming that $V(\mathbf{x}) = \mathbf{x}^T\mathbf{S}\mathbf{x} > 0$ is a candidate Lyapunov function, and the optimal control policy $\mathbf{u} = -\mathbf{R}^{-1}\mathbf{B}\mathbf{S}\mathbf{x}$ if Equation (3.2.32) has a feasible positive definite solution. Taking the first derivative of this candidate Lyapunov function

$$\begin{aligned} \dot{V}(x) &= \mathbf{x}^T\mathbf{S}\dot{\mathbf{x}} + \dot{\mathbf{x}}^T\mathbf{S}\mathbf{x} \\ &= \mathbf{x}^T\mathbf{S}(\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}) + (\mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u})^T\mathbf{S}\mathbf{x} \\ &= \mathbf{x}^T(\mathbf{S}\mathbf{A} + \mathbf{A}^T\mathbf{S} - 2\mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S})\mathbf{x} \\ &= \mathbf{x}^T(-\mathbf{Q} - \mathbf{S}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{S})\mathbf{x} \\ &< 0; \end{aligned} \tag{3.2.33}$$

therefore, the system is asymptotically stable at the origin by Theorem 3.1.1. □

3.3 Tracking the Reference Trajectory by LQR

When the robot intents to track a reference trajectory, asymptotically stable equilibria (Definition 3.1.2) is necessary to be considered. The robot should exactly converge to the goal point, not even somewhere extremely close to that point. Nevertheless, trajectory optimization can only produce the reference trajectory in the finite time horizon for the real situation, which means the real trajectory's terminal state can only be located at somewhere merely close to the reference trajectory's terminal point. [36] came up with the idea that they could easily figure out this issue using the two-stage LQR design. Locally, the infinite horizon LQR guarantees the approach to the equilibrium point. Globally, the finite horizon LQR is used for narrowing the gaps between the potential trajectories and the reference trajectory.

Here are the detailing steps for this idea. It starts backward in time from local to global since by Corollary 3.2.1.1 that Equation (3.2.32) has to find a feasible solution. The first stage is to stabilize the goal state locally.

Consider a smooth nonlinear system

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{u}). \tag{3.3.1}$$

Let $\mathbf{x_f}$ with $\mathbf{u_f}$ is the goal state that the robot has to approach. $\mathbf{x_f}$ and $\mathbf{u_f}$ is also an equilibrium point of the system $\mathbf{f}(\mathbf{x_f}, \mathbf{u_f}) = 0$; otherwise, the problem will be meaningless since if the controller is no longer in the system, the state will immediately move away from the equilibrium point.

Assuming that

$$\bar{\mathbf{x}} = \mathbf{x} - \mathbf{x_f} \text{ , with } \bar{\mathbf{u}} = \mathbf{u} - \mathbf{u_f} \tag{3.3.2}$$

is the gap between the state and the goal state.

Taking the linearization of the time-invariance system around $(\mathbf{x_f}, \mathbf{u_f})$ to obtain the linear system

$$\dot{\bar{\mathbf{x}}} = \mathbf{A}\bar{\mathbf{x}}(t) + \mathbf{B}\bar{\mathbf{u}}(t). \tag{3.3.3}$$

Define the objective function Equation (3.2.2) in the quadratic form

$$\mathbf{J}(\bar{\mathbf{x}}(0)) = \int_0^\infty [\bar{\mathbf{x}}^T(t)\mathbf{Q}\bar{\mathbf{x}}(t) + \bar{\mathbf{u}}(t)\mathbf{R}\bar{\mathbf{u}}(t)]dt, \tag{3.3.4}$$

where $\mathbf{Q} = \mathbf{Q}^T \geq 0$ and $\mathbf{R} = \mathbf{R}^T > 0$.

The optimal objective function

$$\mathbf{J}^*(\bar{\mathbf{x}}(t)) = \bar{\mathbf{x}}^T(t)\mathbf{S}\bar{\mathbf{x}}(t), \tag{3.3.5}$$

where $\mathbf{S}$ is the solution of Equation (3.2.32) (MATLAB's $lqr(.)$ function can easily solve these equations).

Then, the optimal time-invariant control strategy for stabilizing System (3.3.1) around the goal point is

$$\mathbf{u}^* = \mathbf{u_f} - \mathbf{R}^{-1}\mathbf{B}^T\mathbf{S}(\mathbf{x} - \mathbf{x_f}). \tag{3.3.6}$$

The second stage is used for narrowing the gaps between the potential trajectories and the reference trajectory. Given that $\mathbf{x_{ref}}$ is the reference trajectory with the corresponding reference control input $\mathbf{u_{ref}}$, which is found by the trajectory optimization method.

Assuming that

$$\bar{\mathbf{x}}(t) = \mathbf{x}(t) - \mathbf{x_{ref}}(t) \text{ and } \bar{\mathbf{u}}(t) = \mathbf{u}(t) - \mathbf{u_{ref}}(t) \tag{3.3.7}$$

are gaps between the potential trajectory and the reference trajectory with corresponding control inputs at the moment t.

Taking the linearization of the time-varying system around the reference trajectory to obtain

$$\dot{\bar{\mathbf{x}}}(t) = \mathbf{A}(t)\bar{\mathbf{x}}(t) + \mathbf{B}(t)\bar{\mathbf{u}}(t). \tag{3.3.8}$$

Define the objective function Equation (3.2.1) in the quadratic form

$$\mathbf{J}(\bar{\mathbf{x}}(0), 0) = \bar{\mathbf{x}}^T(t_f)\mathbf{Q}_{t_f}\bar{\mathbf{x}}(t_f) + \int_0^{t_f} [\bar{\mathbf{x}}^T(t)\mathbf{Q}\bar{\mathbf{x}}(t) + \bar{\mathbf{u}}^T(t)\mathbf{R}\bar{\mathbf{u}}(t)]dt, \tag{3.3.9}$$

where parameters $\mathbf{Q}_{t_f} = \mathbf{Q}_{t_f}^T > 0$, $\mathbf{Q} = \mathbf{Q}^T \geq 0$, and $\mathbf{R} = \mathbf{R}^T > 0$. Usually, the presetting parameter $\mathbf{Q}_{t_f}$ should much more greater than $\mathbf{Q}$ and $\mathbf{R}$ for ensuring that $\bar{\mathbf{x}}(t_f)$ is sufficiently small; in another word, it means the potential trajectory converges to the reference trajectory as time approaching to the moment t_f. Moreover, the time horizon is not mandatory from 0 to t_f; it can start with $t' \in [0, t_f]$ to have the objective function $\mathbf{J}(\bar{\mathbf{x}}(t'), t')$. This means the potential trajectory's initial state not only needs be located at somewhere close to the reference trajectory's initial state, but also any part around the reference trajectory.

The optimal objective function

$$\mathbf{J}^*(\bar{\mathbf{x}}, t) = \bar{\mathbf{x}}^T(t)\mathbf{S}(t)\bar{\mathbf{x}}(t), \tag{3.3.10}$$

where $\mathbf{S}(t)$ is the solution to the ODE (3.2.27) with the terminal condition (MATLAB's ode45 function is a high-performance technique to solve these equations)

$$\mathbf{S}(t_f) = \mathbf{S}_{stage1}. \tag{3.3.11}$$

The terminal condition (3.3.11) is for linking up two stages of LQR designs to guarantee that Equation (3.2.32) exists a feasible solution.

Then, the optimal time-varying control strategy for tracking the reference trajectory is

$$\mathbf{u}^*(t) = \mathbf{u_{ref}} - \mathbf{R}^{-1}\mathbf{B}^T(t)\mathbf{S}(t)\left(\mathbf{x}(t) - \mathbf{x_{ref}}(t)\right). \tag{3.3.12}$$

3.4 Computational Experiments

All computational experiments in this section ran under the same system environment as chapter 2 by MATLAB.

3.4.1 Simple Pendulum System

The system and parameters showed in System (2.3.1) and Table 2.1, respectively. To utilize the reference trajectories from chapter 2. The goal is to find feedback control policies that can track the reference trajectory and converge to the reference trajectory's terminal state for all initial states around the initial state of the reference trajectory.

For both simple pendulum computational experiments, the goal state $(\pi, 0)$ is an equilibrium point for the system. Define LQR parameters as

$$\mathbf{Q} = \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix} \text{ and } \mathbf{R} = 20. \tag{3.4.1}$$

Following stage 1's procedures in Section 3.3. After solving Equation (3.2.32), the feasible positive definite solution

$$\mathbf{S}_{stage1} = \begin{bmatrix} 230.8806 & 49.2538 \\ 49.2538 & 230.8806 \end{bmatrix}, \tag{3.4.2}$$

which means the goal state $(\pi, 0)$ is asymptotically stable by Corollary 3.2.1.1.

Then, with respective time spans are $[0, 1.9514]$ and $[0, 2.5444]$ for experiment 1 and experiment 2, respectively. Solving the initial value problem (3.2.27) to obtain $\mathbf{S}(t)$ (i.e., Figure 3.2 is the feedback gain $\mathbf{K}(t)$ in Equation (3.2.25) for the experiment 1). Let the reference trajectory's initial state as the center of a circle with radius is 0.5 to generate

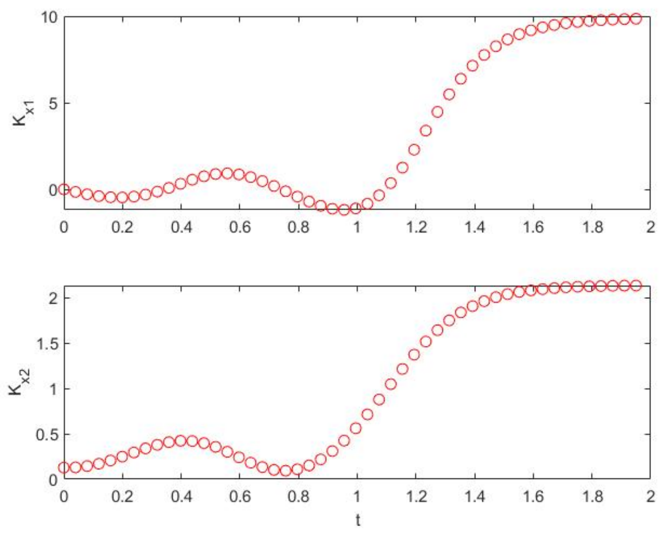

Figure 3.2: Time varying feedback gains for the simple pendulum experiment 1.

random initial states by MATLAB's funtion $rand(.)$. Following stage 2's procedures in Section 3.3, two experiments' desired optimal control policies $\mathbf{u}^*(t)$ and potential trajectories can be achieved.

Figure 3.3, 3.4 are 5 and 1000 potential trajectories' simulations for the experiment 1, and Figure 3.5, 3.6 are 5 and 1000 potential trajectories' simulations for the experiment 2. Four simulations reflect that the initial state around the reference trajectory (the blue trajectory) effectively tracks and converges along the reference trajectory well for the simple pendulum system by the two-stage LQR design with smooth potential trajectories (red trajectories). Moreover, the time consumption for simulating the problem can be ignored.

As mentioned in the explanation of Equation (3.3.9), the potential trajectory's initial can also locate at any part around the reference trajectory and adjust the time horizon with $[t', t_f]$, where $t' \in [0, t_f]$. Figure 3.7 is the example with random initial states (green dots) around any part of the reference trajectory (radius=0.3) with corresponding t'; the figure indicate that feedback controllers are resultful as well.

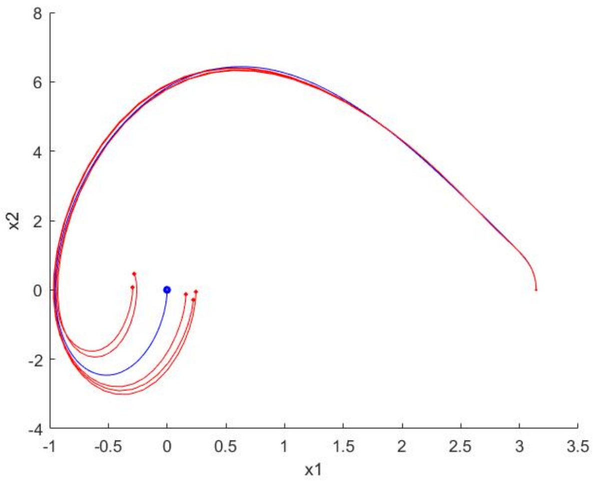

Figure 3.3: Tracking the simple pendulum experiment one's reference trajectory from chapter 2 (5 trajectories).

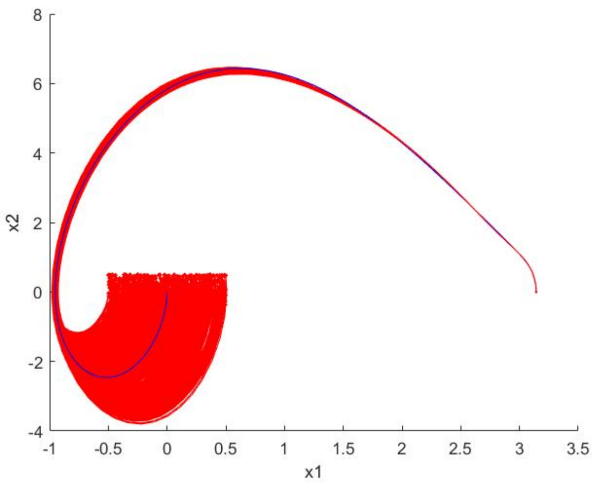

Figure 3.4: Tracking the simple pendulum experiment one's reference trajectory from chapter 2 (1000 trajectories).

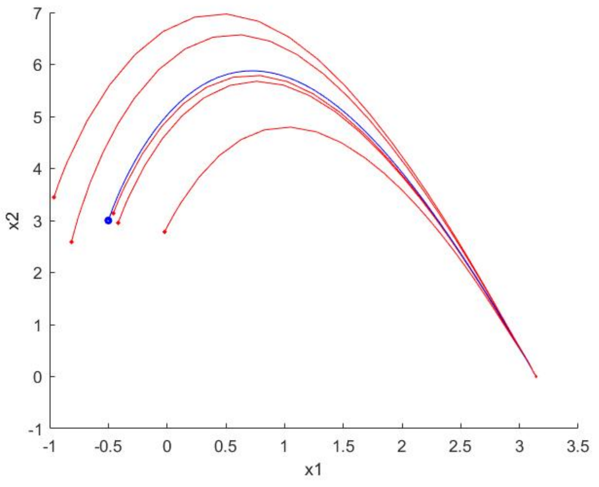

Figure 3.5: Tracking the simple pendulum experiment two's reference trajectory from chapter 2 (5 trajectories).

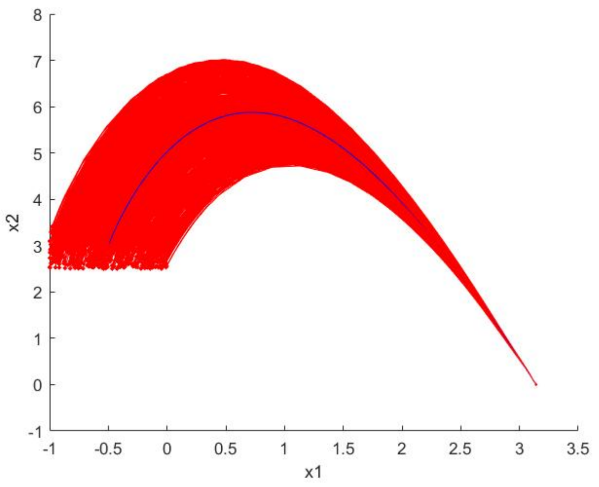

Figure 3.6: Tracking the simple pendulum experiment two's reference trajectory from chapter 2 (1000 trajectories).

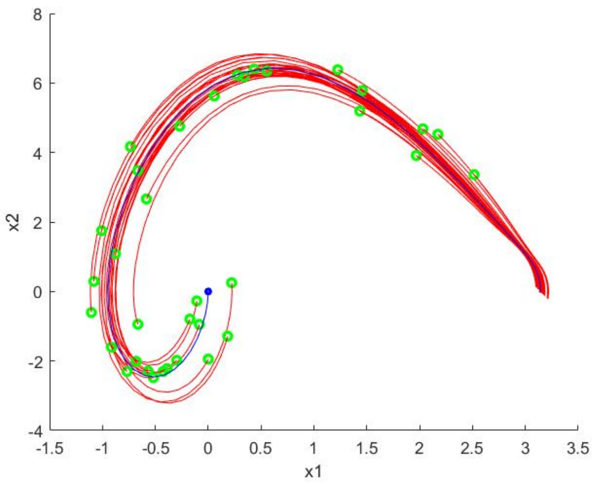

Figure 3.7: Tracking the simple pendulum experiment one's reference trajectory from chapter 2 with random initial states around any part of the reference trajectory.

3.4.2 Three-dimensional Nonlinear System

The system showed in Equation (2.3.2). To utilize the reference trajectory from chapter 2. The goal is to find feedback control policy that can track the reference trajectory and converge to the reference trajectory's terminal state for all initial state of the reference trajectory.

The goal state $(0, 0, 0)$ is an equilibrium point for the system. Define LQR parameters as

$$\mathbf{Q} = \begin{bmatrix} 10 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ and } \mathbf{R} = 20. \tag{3.4.3}$$

Following stage 1's procedures in Section 3.3. After solving Equation (3.2.32), the feasible positive definite solution

$$\mathbf{S}_{stage1} = \begin{bmatrix} 20.3206 & 10.1431 & 14.4684 \\ 10.1431 & 10.4979 & 9.7904 \\ 14.4684 & 9.7904 & 14.6787 \end{bmatrix}. \tag{3.4.4}$$

which means the goal state $(0, 0, 0)$ is asymptotically stable by Corollary 3.2.1.1.

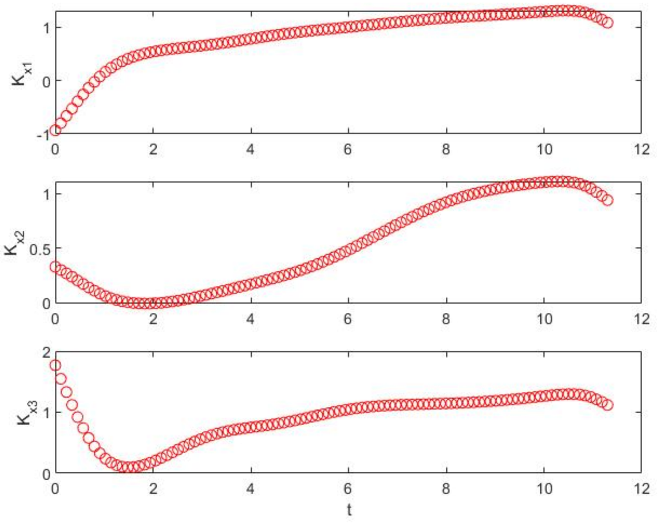

Figure 3.8: Time varying feedback gains for three-dimensional nonlinear system.

Then, with the time span is $[0, 11.3]$, solving the initial value problem (3.2.27) to obtain $\mathbf{S}(t)$; Figure 3.8 is the feedback gain $\mathbf{K}(t)$ in Equation (3.2.25). Let the reference trajectory's initial state as the center of sphere with the radius is 0.3 to generate initial states. Following stage 2's procedures in Section 3.3, the optimal control policies $\mathbf{u}^*(t)$ and potential trajectories can be achieved.

Figure 3.9 and 3.10 are 5 and 200 potential trajectories' simulations. Both simulations reflect that the initial state around the reference trajectory (the blue trajectory) effectively tracks and converges along the reference trajectory well in the three-dimensional space by the two-stage LQR design with smooth potential trajectories (red trajectories).

Figure 3.11 is the example with random initial state (green dots) around any part of the reference trajectory (radius=0.3) with corresponding t', where $t' \in [0, t_f]$. It shows that feedback controllers are resultful for the three-dimensional system as well.

Remark. The code template for implementing the two-stage LQR control design is posted in Appendix A.

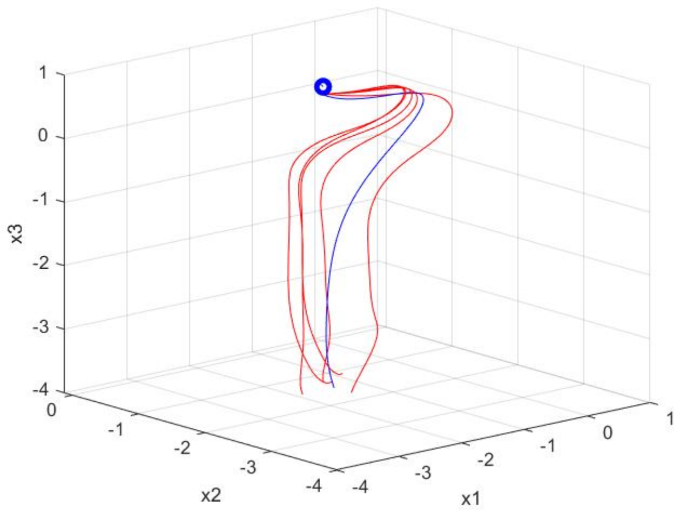

Figure 3.9: Tracking the three-dimensional system nonlinear from chapter 2 (5 trajectories).

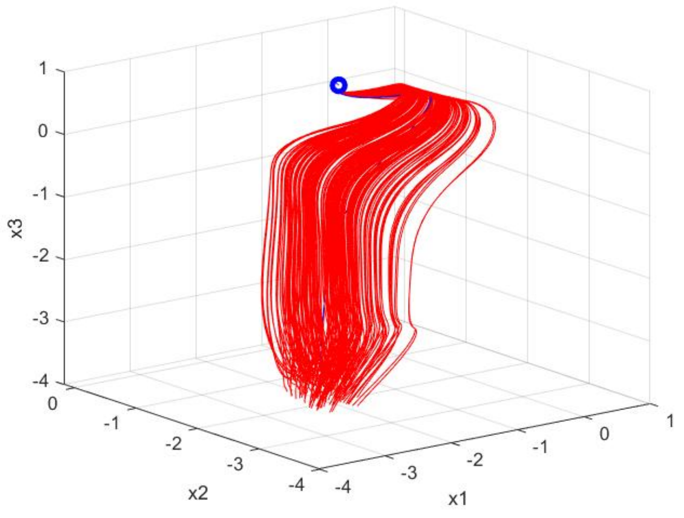

Figure 3.10: Tracking the three-dimensional nonlinear system from chapter 2 (200 trajectories).

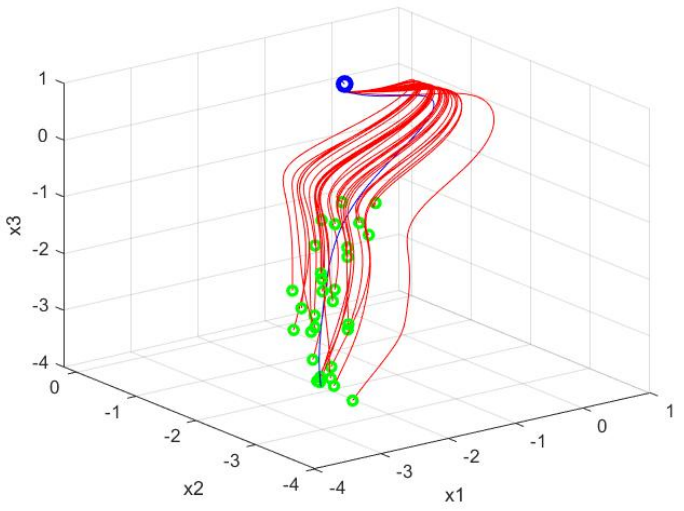

Figure 3.11: Tracking the three-dimensional nonlinear system's reference trajectory from chapter 2 with random initial states around any part of the reference trajectory.

Chapter 4

Abstraction-based Control Design

With the development of science and technology, there are increasing numbers of safety-critical applications [22] in today's society. Thereinto, symbolic control is one of the bright future solutions for building automatically correct-by-construction Safety-Critical Control Software. Algorithmic guaranteed controllers can be provided if given the cyber-physical system and some executable and rigorous requirements.

Abstraction-based Control Design (ABCD) or finite abstraction is another representational term or one of the most cross-sectional method for symbolic control. The deep and particular benefit is that it is one of the state-of-the-art approaches to producing feedback controllers that can provide some formal correctness guarantees and efficiently against the cyber-physical system with boundary uncertainties, i.e.,

$$\dot{\mathcal{E}}(t) \in f(\mathcal{E}(t), u) + W, \tag{4.0.1}$$

where $W := [\![-w, w]\!]$ is a bounded perturbation term, and $\mathcal{E}(t)$ is a continuous solution of the System (4.0.1) under control input u on the time horizon $[0, \tau]$.

4.1 The Framework of Abstraction-based Control Design

4.1.1 Some Preliminaries

In this subsection, some preliminaries about the symbolic approach and what the symbolic controller acts on the system will be introduced.

Considering the system in a triplet

$$S(X, U, F), \tag{4.1.1}$$

where X is the state alphabet has state signal $x : [0; T) \to X$, U is the input alphabet has input signal $u : [0; T) \to U$, and $F : X \times U \rightrightarrows X$ is the transition function or a strict set-valued map correlates to the System (4.0.1) is defined as time discrete dynamics

$$x(t+1) \in F(x(t), u(t)); \tag{4.1.2}$$

the transition function (4.1.2) can be satisfied if and only if it holds as

$$F(x, u) := \{x' | \mathcal{E}(0) = x \wedge \mathcal{E}(\tau) = x' \text{ for the valid solution of System (4.0.1)}\} \tag{4.1.3}$$

A complete control problem in symbolic control is consisted of the system S and specification

$$\Sigma \subseteq (X \times U)^{\infty}, \tag{4.1.4}$$

exempli gratia Expression (2.1.1) and (2.1.2).

Definition 4.1.1 (Behavior). The set $\mathcal{B}(S)$ of the system (4.1.1) is called the behavior of the S

$$\begin{aligned} \mathcal{B}(S) = \{(x, u) | \exists \text{ pairs } (x, u) \text{ are the solution of } S \text{ on } [0; T) \\ \wedge F(x(T-1), u(T-1)) = \emptyset \text{ for } T < \infty\}, \end{aligned} \tag{4.1.5}$$

this means that satisfy both Equation (4.1.2) and exist feasible control input u of the control system S.

Ultimately, the symbolic control problem can be addressed as solvable if and only if there exists a desired solution C of the control system S, which is a composable feedback controller to satisfy that

$$\mathcal{B}(C \times S) \subseteq \Sigma. \tag{4.1.6}$$

4.1.2 The Framework of Abstraction-based Control Design

Figure 4.1 shows the fundamental framework of ABCD. The specific process is built upon the classic approach in [30] and [32]; it can be, on the whole, partitioned into three steps.

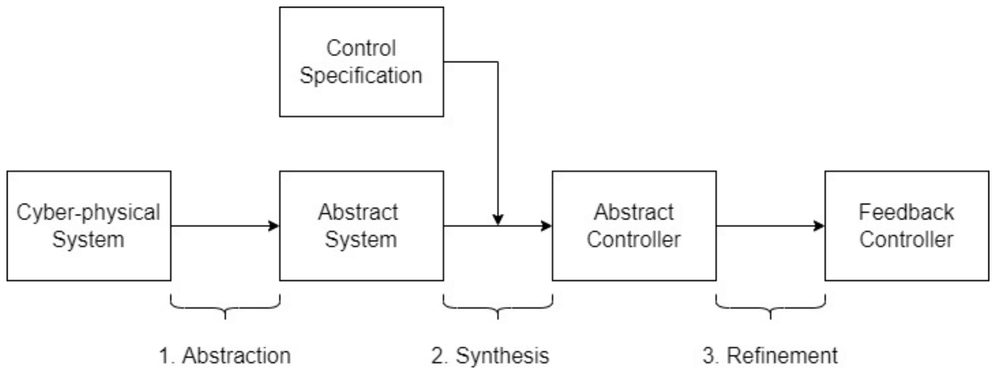

Figure 4.1: The framework of Abstraction-based Control Design

1. Abstraction.

Given a simple system in a triplet $S_1(X_1, U_1, F_1)$ and a specification Σ_1, same as defined in (4.1.1) and (4.1.4), respectively. The first step in Figure 4.1 is called abstraction, which means discretizing the continuous simple system and specification to the discrete simple system and the abstract specification

$$S_2 = (X_2, U_2, F_2) \text{ and } \Sigma_2, \tag{4.1.7}$$

respectively.

The detailing is to construct the state alphabet of X_2 with small equidistant size grids to cover the state X_1 by non-empty, closed hyper-intervals

$$[\![a, b]\!] = \mathbb{R}^n([a_1, b_1] \times \cdots \times [a_n, b_n]) \tag{4.1.8}$$

for some a, $b \in (\mathbb{R} \cup \{\pm\infty\})^n$, where $a < b$.

The sparsity of the grid point set depends on the size of the grid η_i for each dimension, and the set of grid points can be shown as

$$\eta\mathbb{Z}^n = \{c \in \mathbb{R}^n | \exists_{k\in\mathbb{Z}^n} \forall_{i\in\{1,\ldots,n\}} c_i = k_i\eta_i\}; \tag{4.1.9}$$

beyond all doubt, the higher precision (small η) will cause higher time and memory consumption in the computer but a higher success rate in finding a feasible symbolic controller.

However, the range of X_2 is defined as closed but not unbound, so the problem is not in a practical sense and not solvable by the computer in a lot of cases. Therefore, it is doable if only considering the real closed and bounded subset $\bar{X}_2 \subseteq X_2$ as

$$\bar{X}_2 := \{x_2 | \exists_{c\in(\eta\mathbb{Z}^n \cap [\![a,b]\!])} x_2 = c + [\![-\frac{\eta}{2}, \frac{\eta}{2}]\!]\}, \tag{4.1.10}$$

and $x_2 \in X_2 \backslash \bar{X}_2$ is called the overflow symbol, which can be ignored in the subsequent computational process.

To arrow $Z_1 \subseteq X_1$ is goal set, the homologous goal set of the abstract system

$$Z_2 = \{x_2 \in X_2 | x_2 \subseteq Z_1\}. \tag{4.1.11}$$

Likewise, to arrow $I_1 \subseteq X_1$ is the initial set, the homologous initial set of the abstract system

$$I_2 = \{x_2 \in X_2 | x_2 \cap I_2 \neq \emptyset\}. \tag{4.1.12}$$

Assuming that $U_2 \subseteq U_1$ is the finite set, let $\varphi(., p, u)$ is the nominal solution of System (4.0.1) of the initial value problem without considering the perturbation term:

$$\dot{\varphi}(t, p, u) = f(\varphi(t, p, u), u), \tag{4.1.13}$$

where p is the initial condition at the moment $t = 0$ and $t \in [0, \tau]$.

For designing the transition function F_2 of the finite system S_2, the growth bound function $\beta : \mathbb{R}^n_+ \times U_2 \to \mathbb{R}^n_+$ will be applied for setting the upper limit gap between the nominal solution φ of the System (4.1.13) and the solution $\mathcal{E}$ of the System (4.0.1) on the time horizon $[0, \tau]$ for $u \in U_2$, as follows

$$|\mathcal{E}(\tau) - \varphi(\tau, p, u)| \leq \beta(|\mathcal{E}(0) - p|, u). \tag{4.1.14}$$

The growth bound function (containing the disturbance w term from System (4.0.1)) is established on $[0, \tau]$ as

$$\beta(r, u) = \exp(L(u)\tau)r + \int_0^\tau \exp(L(u)s)w \, \mathrm{d}s, \tag{4.1.15}$$

where $L : U_2 \to \mathbb{R}^{n \times n}$ is consisted of the Jacobian matrix of f, as follows

$$L_{i,j}(u) \geq \begin{cases} D_j f_i(x, u) & \text{if } i = j, \\ |D_j f_i(x, u)| & \text{otherwise,} \end{cases} \tag{4.1.16}$$

for letting $\mathcal{E}(0), p \in K \subseteq K' \subseteq \mathbb{R}^n$ with K' being convex, and $\forall x \in K'$ and $\forall u \in U_2$.

The intuition behind it is only considered the worst-case scenario, which means it calculates the maximum radius for the reachable set overall states to skip calculating the reachable set for each cell in the grid. Supposing System (4.0.1) is without the disturbance term; the growth bound function can be simplified as written in

$$\beta(r, u) = exp(L(u)\tau)r. \tag{4.1.17}$$

It is worth mentioning that Equation (4.1.15) is the general solution to the initial value problem

$$\begin{aligned} \dot{r}(t) &= L(u)r(t) + w, \\ r(0) &= r_0; \end{aligned} \tag{4.1.18}$$

since the general solution is consisted of the general homogeneous solution plus a particular solution. In the first place, guessing the particular solution is in the form $r_p(t) = \exp(tL(u))\zeta(t)$, to substitute it into the differential equation (4.1.18) gets

$$L(u)\exp(tL(u))\zeta(t) + \exp(tL(u))\dot{\zeta}(t) = L(u)\exp(tL(u))\zeta(t) + w. \tag{4.1.19}$$

After the simplified,

$$\dot{\zeta}(t) = \exp(-tL(u))w \Rightarrow \zeta(t) = \int_0^t \exp(-sL(u))wds, \tag{4.1.20}$$

and

$$r_p(t) = \exp(tL(u))\zeta(t) = \int_0^t \exp((t-s)L(u))wds. \tag{4.1.21}$$

Thus, to together with the homogeneous solution based on the initial condition, the general solution of the growth bound function is

$$r'(t) = \exp(tL(u))r_0 + \int_0^t \exp((t-s)L(u))wds; \tag{4.1.22}$$

the initial value problem (4.1.18) can be solved by ode solvers.

The following Algorithm is steps to compute the transition function F_2 of the simple finite system.

Algorithm 4: Computing the transition function F_2

```
1 Given $\bar{X}_2$, $U_2$, $\beta$, $\varphi$, $\eta$, $\tau$;
2 for $c + [\![-r, r]\!] \in \bar{X}_2$ and $u \in U_2$ do
3     $r' := \beta(\frac{\eta}{2}, u)$;
4     $c' := \varphi(\tau, c, u)$;
5     $A := \{x_2' \in X_2 | (c' + [\![-r', r']\!]) \cap x_2' \neq \emptyset\}$;
6     if $A \subseteq \bar{X}_2$ then
7         $F_2(x_2, u) := A$;
8     else
9         $F_2(x_2, u) := \emptyset$;
```

Apart from that, the transition function $F_2(x_2, u) = \emptyset$, where $\forall x_2 \in X_2 \backslash \bar{X}_2$ with $u \in U_2$.

Below, there are some requisites for connecting the relationship between the simple system S_1 and the finite system S_2.

Definition 4.1.2 (Admissible Inputs Set)**.** Admissible inputs set $U_S(x)$ is defined as, for $x \in X$,

$$U_S(x) = \{u \in U | F(x, u) \neq \emptyset\}. \tag{4.1.23}$$

Definition 4.1.3 (Feedback Refinement Relation)**.** Assuming that $U_2 \subseteq U_1$, there exists a feedback refinement relation $Q \subseteq X_1 \times X_2$ from the simple system S_1 to the abstract simple system S_2, denotes as $S_1 \preccurlyeq_Q S_2$, if for all $(x_1, x_2) \in Q$ satisfy:

1. $U_{S_2}(x_2) \subseteq U_{S_1}(x_1)$;
2. if $u \in U_{S_2}(x_2)$, which implies to $Q(F_1(x_1, u)) \subseteq F_2(x_2, u)$.

As well as, there exists the feedback refinement relation, if for all Ω, $\Omega' \in X_2$ and all $u \in U_2$ satisfy:

$$\Omega' \cap F_1(\Omega, u) \neq \emptyset \text{ implies to } \Omega' \in F_2(\Omega, u). \tag{4.1.24}$$

Definition 4.1.4 (Abstract Specification)**.** Let S_1 and S_2 are simple systems, $\Sigma_1 \subseteq (X_1 \times U_1)^\infty$ is the specification, and $Q \subseteq X_1 \times X_2$ is the feedback relation. The specification $\Sigma_2 \subseteq X_2 \times U_2$ is called the abstract specification if $(x_2, u) \in \Sigma_2$, where x_2 and u are defined on the time horizon $[0, T)$ for some $T \in \mathbb{N} \cup \{\infty\}$, and if $x_1 : [0, T) \to X_1$ satisfies $(x_1(t), x_2(t)) \in Q$ for all $t \in [0, T)$, then $(x_1, u) \in \Sigma_1$.

Eventually, if there is a feedback refinement relation between the concrete infinite control problem and the abstract control problem $(S_1, \Sigma_1) \preccurlyeq_Q (S_2, \Sigma_2)$, then $S_1 \preccurlyeq_Q S_2$ and Σ_2 is an abstract specification have to be required. SCOTS can automatically compute the feedback refinement relation Q.

Something to remind that [29] is one of the early works talked about using abstractions to solve the nonlinear control problem.

2. Synthesis.

The second step in Figure 4.1 is called synthesis, which aims to solve the abstract control problem (S_2, Σ_2). The control specification in propositional modal logic may be but is not limited to Linear temporal logic (LTL), μ-calculus, etc. Some automate-theoretic schemes can be used to synthesize the problem to obtain the discrete control strategies (abstract controllers). SCOTS [32] gives synthetic algorithms for the invariance specification (or safety specification, which means the trajectory avoids touching a specific set forever) and reachability specification as (2.1.1).

The synthetic algorithm for solving the reachability problem in SCOTS is based on Dijkstra's shortest path algorithm, the pseudo-code is posted in Appendix B. Differently, [25] improves the synthetic algorithm for the reachability specification by the breadth-first search algorithm. All synthetic algorithms start from the abstract goal set Z_2 to solve the problem backward; the linchpin is to take the inverse of the transition function (4.1.2)

$$(x_2, u) \in F_2^{-1}(x_2') \tag{4.1.25}$$

to explore all possible $x_2 \in \bar{X}_2$. Due to the problem being solved backward, ABCD can only solve offline control problems here.

To explain particulars, a map $pre(Y_i)$ is necessary to be defined as

$$pre(Y_i) = \{x_2 \in X_2 | \exists_{u \in U_{S_2}(x_2)} F_2(x_2, u_2) \subseteq Y_i\}, \tag{4.1.26}$$

where, for $i \in \mathbb{N}$, $Y_0 = \emptyset$ and

$$Y_{i+1} = pre(Y_i) \cup Z_2. \tag{4.1.27}$$

The output of synthetic algorithms is set-valued maps $D : X_2 \rightrightarrows U_2$, and the inverse mapping $D^{-1}(U_2)$ is the minimal fixed point of the largest set of initial state Y_∞ of Equation (4.1.27). [2] is a book characterizes about applications of fixed point algorithms in engineering.

The abstract control problem can derive an eligible composable feedback controller C_2 if and only if the abstract initial set belongs to the minimal fixed point set, which is the output

$$I_2 \subseteq D^{-1}(U_2). \tag{4.1.28}$$

The output function H_c to yield state-input pairs, which is the imperative component of the feedback composable C_2; it is delineated as

$$H_c = \begin{cases} D(x_2) \times \{x_2\} & \text{if } x_2 \in D^{-1}(U_2); \\ U_2 \times \{x_2\} & \text{otherwise.} \end{cases} \tag{4.1.29}$$

3. Refinement.

The third step in Figure 4.1 is called refinement. The previous step has already attained a controller for the abstract control problem; this step is used to reach the controller C_1 for the concrete control problem through the abstract controller C_2 and the feedback refinement relation Q.

Figure B.1 in Appendix B is the rough flow path of the closed-loop control design and how the refinement relation (as known as a quantizer) applies to the system. The following Theorem 4.1.1 supports that C_1 can solve the concrete control problem (S_1, Σ_1).

Theorem 4.1.1. *If there is a feedback refinement relation $Q : X_1 \rightrightarrows X_2$ between the concrete infinite control problem and the abstract control problem $(S_1, \Sigma_1) \preccurlyeq_Q (S_2, \Sigma_2)$, and the abstract controller C_2 solves the abstract control problem (S_2, Σ_2), then the refined controller of serial composition $C_1 := C_2 \circ Q$ solves the concrete infinite control problem (S_1, Σ_1). As a consequence, the behavior $\mathcal{B}(C_1 \times S_1)$ of this closed-loop control design is a subset of Σ_1.*

The detailed mathematical proofs for Theorem 4.1.1 and rationale for supporting the feedback refinement relations are shown in Section V.A and Section VI.A in [30].

To sum up, there are many ready-made ABCD toolboxes, such as [32], [20], [12], for solving different sorts of systems and specifications. Beyond that, [25] is a modified toolbox that can handle more mainstream control specifications, such as Büchi and Co-Büchi specification, and optimize run-time and memory consumption at the same time. In addition, [38] is the latest ABCD research used the memory-efficient on-the-fly algorithm and innovative formal correctness methodology during the control synthesis, which has an excellent performance in resource consumption in some experiments; it also compared the above toolboxes in all aspects.

4.2 Abstraction-based Control Design on Account of the Reference Trajectory

In most situations, ABCD has to compute the entire workspace; nevertheless, the bottleneck is that the computation is costly, and it is effortless to exceed the computer's maximum memory if the physical system is in high dimensions and the abstract system with the relative small grid. A considerable time and memory consumption (commonly consumes over several hundred gigabytes and hours) are strangling normal users to work on some simple examples.

A neoteric solution is provided by C++ toolbox GAMARA [26], which is an advanced toolbox based on SCOTS [32]. First of all, they use trajectory optimization to generate an open-loop reference trajectory based on the system without uncertainty. Then, regarding the state $x_{ref}(t_k)$ of the reference trajectory at each time node t_k as the center to create a new workspace with a constant radius δ,

$$Workspace_{new} := \bigcup_{k=0}^{t_f} x_{ref}(t_k) + [-\delta, \delta]; \tag{4.2.1}$$

more specifically, the radius of each dimension δ_i of the system is in the form

$$\delta_i = \text{the number of grids of each dimension} \times \text{the size of the grid } \eta_i. \tag{4.2.2}$$

GAMARA yielded a built-in mapping procedure to lead the tube-sharp workspace in an executable ABCD setting.

Finally, to apply ABCD through SCOTS to find the feedback controller on the new cutdown tube-shape workspace. It differs from LQR control design; the reference trajectory is not used for tracking but simply presets an estimated cutdown workspace, then solves the problem by ABCD. The logic behind it is that the reachable sets for the states inside the winning domain of the controller will be inside the tube to keep trajectories inside the tube.

The advantage of this solution is that trajectory optimization conducts to determine the cut-down workspace to reduce the computational cost obviously and to roughly ensure the reachability set is within the tube shape workspace around the reference trajectory in the later steps globally. And, ABCD can guarantee if there are disturbances in the system locally.

Howbeit, the solution solved by GAMARA in the optimized workspace can be seen as the secondary solution solved by SCOTS in the entire workspace, which means GAMARA can find a solution in the optimized workspace only if SCOTS can find a solution in the entire workspace. Consequently, the potential limitation is that it has very stringent conditions for selecting appropriate parameters, such as the growth bound, the size of each grid and radius, and the time interval, to use GAMARA to find viable solutions that trajectories are fully located within the tube.

4.3 Computational Experiments

The numerical experiment has been run on Inter Gold 6148 Skylake processor at 2.4GHz with a limit of 92GB of RAM on the Béluga cluster with the permission of Compute Canada.

4.3.1 Simple Pendulum System

The system dynamic and parameters showed in System (2.3.1) and Table 2.1, respectively. The experiment and its reference trajectory are based upon the existing simple pendulum experiment one from chapter 2 rather than the trajectory optimization method in GAMARA. The reference trajectory (blue trajectory in Figure 4.2 with time horizon [1,1.95] and the discrete time unit $\tau = 0.01$) exists a little error from the original reference trajectory because the input of GAMARA is control input, and it unavoidably causes an error for calculating states by the ode solver.

Given that bounds of the hyper rectangle of state and input are $X = [-2, 4] \times [-3, 1]$ and $U = [-4, 4]$. The size of single state grid and input grid $\eta_x = [0.001, 0.01]$ and $\eta_u = 1$, and the number of grids of each dimension is [10,10].

To take the manipulation of Function (4.1.16) to find the Jacobian matrix

$$L(u) = \begin{bmatrix} 0 & 1 \\ \frac{mgl}{I} & -\frac{b}{I} \end{bmatrix}; \tag{4.3.1}$$

then, to follow the growth bound function without disturbance (4.1.17) has

$$r' = \beta(r, u) = exp(L(u)\tau)r = \begin{bmatrix} 1.001 & 0.01 \\ 0.1957 & 0.9970 \end{bmatrix} r. \tag{4.3.2}$$

MATLAB's *expm*() function is very convenient for calculating matrix exponential.

The green zigzag trajectory in Figure 4.2 is the experimental result of controlling System (2.3.1) from the downward position to the upright position without disturbance by ABCD. It consumed 0.068GB of RAM and took 0.8127s of CPU time for the abstraction part and 0.01404s of CPU time for the synthesis part to solve the reachability problem. The real trajectory is very approximate to the reference trajectory since the size new tube shape workspace is relatively tiny.

Further on, everything else remains unchanged; the disturbance W adds to the model just like System (4.0.1), where the bound $|W| \leq (0.01; 0.01)$. Following the growth bound function with disturbance (4.1.15) has

$$\begin{aligned} r' = \beta(r,u) &= \exp(L(u)\tau)r + \int_0^\tau \exp(L(u)s)w\,\mathrm{d}s \\ &= \begin{bmatrix} 1.001 & 0.01 \\ 0.1957 & 0.9970 \end{bmatrix} r + \begin{bmatrix} 0.01 & 0 \\ 0.001 & 0.01 \end{bmatrix} W; \end{aligned} \tag{4.3.3}$$

the red zigzag trajectory in Figure 4.2 is the experimental result with disturbance. It consumed 0.0798GB of RAM and took 0.8104s of CPU time for the abstraction part and 0.0325s of CPU time for the synthesis part to solve the reachability problem.

4.3.2 Simple Vehicle Model

Consider the simplest nonlinear vehicle model, and it is also known as the kinematic unicycle model

$$\begin{aligned} \dot{x}_1 &= v\cos(\theta) \\ \dot{x}_2 &= v\sin(\theta) \\ \dot{\theta} &= u, \end{aligned} \tag{4.3.4}$$

where x_1 and x_2 are coordinates for the vehicle in the two-dimensional space, and θ is the angle of the steerable wheel. v is a control input of the velocity for this vehicle, and u is another control input of the steerable wheel's angular velocity; the bounds for control inputs are $-2.4 \leq u,\ v \leq 2.4$. The size of a single input grid $\eta_u = [0.3, 0.3]$. The model shows that $\dot{x}_1$, $\dot{x}_2$, and $\dot{\theta}$ do not rely on this vehicle's location.

Assuming that the reference trajectory has to satisfy with the initial state is $(0, 0, \frac{\pi}{2})$, and the terminal state is $(0, 10, \frac{\pi}{2})$. The minimum cost reference trajectory is a straight route in the two-dimensional space (blue trajectory in Figure 4.3) with constant $\theta = \frac{\pi}{2}$

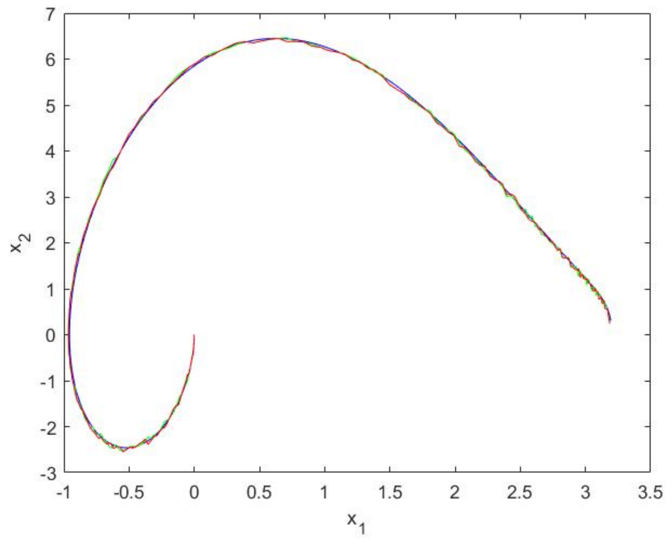

Figure 4.2: Reference trajectory (blue), the real trajectory without disturbance (green), and the real trajectory with disturbance (red) for the simple pendulum system by ABCD.

in the time span [0, 10.3]; this is a weak version experiment from the toolbox. Given the discrete time unit $\tau = 0.1$, the number of grids of each dimension is [7,7,7], and the size of a single state grid $\eta_x = [0.025, 0.025, 0.03]$ to create a new tube shape workspace to solve the reachability problem within it.

On the basis of the original vehicle model (4.3.4) to consider additional disturbance W just like System (4.0.1), where the bounds $|W| \leq (0; 0.025; 0.025)$. GAMARA took a total of 143.013s of CPU time and 8.54GB of RAM to solve this reachability problem; detailedly, the abstraction part accounted for 135.014s, and the synthesis part accounted for 7.99881s. The red zigzag trajectory in Figure 4.3 is a real trajectory found by ABCD to satisfy the reachability specification in the tube shape workspace, in that the initial state is $(0; 0; \frac{\pi}{2})$ and exists disturbance.

For comparison, everything else remains unchanged; the green zigzag trajectory in Figure 4.3 is the same system without the disturbance term W. It took 124.925s of CPU time for the abstraction part and 7.85452s of CPU time for the synthesis part to solve the reachability problem.

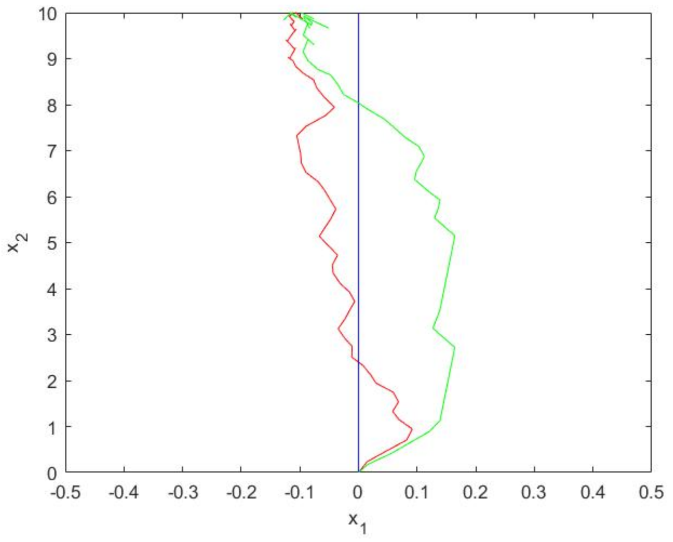

Figure 4.3: Reference trajectory (blue), the real trajectory with disturbance (red), and the real trajectory without disturbance (green) for the vehicle model by ABCD.

Chapter 5

Conclusion and Future Works

In this chapter, we will make a conclusion based on this research paper and list some potential research angles.

5.1 Conclusion

Beyond all doubt, trajectory optimization is a powerful weapon for computing an open-loop nominal trajectory for optimal control problems, especially the direct collocation method. It can rapidly and accurately solve high-dimensional systems with even some complicated constraints. [10], [11] are popular well-optimized trajectory optimization toolboxes for large scale systems.

Sometimes, less is more in the vast majority of scenarios; PID or LQR control designs work decently and simulate in fast time, and experimental results are more close to practical applications in use (For now, ABCD probably can only give results with theoretical guarantees to solve the control problem). For both LQR control design and ABCD, difficulties are choosing appropriate parameters; in some cases, adjusting parameters is time-consuming and interminable. The parameter selection problem is especially prominent on ABCD with an optimized workspace setting in chapter 4; the growth bound, the size of each grid and radius, and the time interval are all influencing factors in achieving satisfactory results. For ABCD, effects are disproportional to causes; this means that a tiny error in any link may bring about the butterfly effect to lead to a mistake or inaccurate result.

However, ABCD as a solution also has its unique advantages. It can find a feedback controller to reduce the impacts of disturbances, handle modeling uncertainties, and solve many high-level safety-critical applications with mathematical guarantees. ABCD is an up-and-coming control method in the near future and will receive more attention from the control field. With the next revolution in computer technology, processing speed and computer hardware can be significantly boosted so that even smaller grids can segment the workspace, and even more complex constraints can be added to the problem to improve the success rate and precision.

5.2 Future Works

- Investigating more complicated nonlinear systems with interesting dynamics and high-dimensional control inputs, such as CartPole, simple Goddard rocket [3], Quadcopter, etc.

- As for solving a complete reachability problem by the two-stages LQR design, the essential part is to find out the reachable set. There are some existing solutions, [36] refers on the definition of Lyapunov function, [31] is a simulation-based solution, and [28] uses the level set methods.

- In reality, the majority of systems are stochastic, which means there is noise or disturbance. The two-stage LQR design can only figure out deterministic systems. Linear-quadratic-Gaussian control (LQG) [15] is a way to deal with the initial condition with Gaussian disturbance by adding the observer gain. Besides, [37] is a mature toolbox to solve stochastic reachability control problems.

- Both LQR control design and ABCD in this research paper solve the problem backward from the goal set; therefore, they can only figure out offline control problems. In the future, we are perhaps attempting more modern optimal control techniques, such as MPC, MHE, etc. So then, more realistic online control problems can be solved.

- In this research paper, only reachability specification was considered. More different control specifications can be considered in the future. One doable idea is to combine [26] with our own toolbox [25] to solve even higher-dimensional systems or multi-agent systems.

- For the toolbox GAMARA [26], they only simulated trajectories with a single initial state coinciding with the reference trajectory's initial state. It is possible to find that all initial states in the tube belong to the research set for states inside the wining domain whose corresponding real trajectories are also inside the tube, similar to LQR design in chapter 2.

References

[1] A Pedro Aguiar, Dragan B Dačić, Joao P Hespanha, and Petar Kokotović. Path-following or reference tracking?: An answer relaxing the limits to performance. *IFAC Proceedings Volumes*, 37(8):167–172, 2004.

[2] Heinz H Bauschke, Regina S Burachik, Patrick L Combettes, Veit Elser, D Russell Luke, and Henry Wolkowicz. *Fixed-point algorithms for inverse problems in science and engineering*, volume 49. Springer Science & Business Media, 2011.

[3] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 1. Athena scientific, 2012.

[4] John T Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207, 1998.

[5] John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

[6] Hans Georg Bock and Karl-Josef Plitt. A multiple shooting algorithm for direct solution of optimal control problems. *IFAC Proceedings Volumes*, 17(2):1603–1608, 1984.

[7] Tilman Buchner. Robotics outlook 2030: How intelligence and mobility will shape the future, Aug 2021.

[8] Moritz Diehl and Sébastien Gros. Numerical optimal control. *Optimization in Engineering Center (OPTEC)*, 2011.

[9] Adriano Festa, Roberto Guglielmi, Christopher Hermosilla, Athena Picarelli, Smita Sahu, Achille Sassi, and Francisco J Silva. Hamilton–jacobi–bellman equations. In *Optimal Control: Novel Directions and Applications*, pages 127–261. Springer, 2017.

[10] Boris Houska, Hans Joachim Ferreau, and Moritz Diehl. Acado toolkit—an open-source framework for automatic control and dynamic optimization. *Optimal Control Applications and Methods*, 32(3):298–312, 2011.

[11] Taylor A Howell, Brian E Jackson, and Zachary Manchester. Altro: A fast solver for constrained trajectory optimization. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7674–7679. IEEE, 2019.

[12] Kyle Hsu, Rupak Majumdar, Kaushik Mallik, and Anne-Kathrin Schmuck. Lazy abstraction-based control for safety specifications. In *2018 IEEE Conference on Decision and Control (CDC)*, pages 4902–4907. IEEE, 2018.

[13] Michael A Johnson and Mohammad H Moradi. *PID control*. Springer, 2005.

[14] Rudolf Emil Kalman et al. Contributions to the theory of optimal control. *Bol. soc. mat. mexicana*, 5(2):102–119, 1960.

[15] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. 1960.

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

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

[18] Matthew Kelly. An introduction to trajectory optimization: How to do your own direct collocation. *SIAM Review*, 59(4):849–904, 2017.

[19] Matthew P Kelly. Optimtraj user's guide, version 1.5, 2019.

[20] Mahmoud Khaled and Majid Zamani. pfaces: An acceleration ecosystem for symbolic control. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 252–257, 2019.

[21] Hassan K Khalil. Nonlinear systems third edition. *Patience Hall*, 115, 2002.

[22] John C Knight. Safety critical systems: challenges and directions. In *Proceedings of the 24th international conference on software engineering*, pages 547–550, 2002.

[23] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. 1998.

[24] Yanbo Li, Zakary Littlefield, and Kostas E Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564, 2016.

[25] Yinan Li, Zhibing Sun, and Jun Liu. Rocs 2.0: An integrated temporal logic control synthesis tool for nonlinear dynamical systems. *IFAC-PapersOnLine*, 54(5):31–36, 2021.

[26] Rupak Majumdar, Kaushik Mallik, Mahmoud Salamati, Sadegh Soudjani, and Mehrdad Zareian. Symbolic reach-avoid control of multi-agent systems. In *Proceedings of the ACM/IEEE 12th International Conference on Cyber-Physical Systems*, pages 209–220, 2021.

[27] Mujeeb R Malik, Thomas A Zang, and M Yousuff Hussaini. A spectral collocation method for the navier-stokes equations. *Journal of Computational Physics*, 61(1):64–88, 1985.

[28] Ian M Mitchell et al. A toolbox of level set methods. *UBC Department of Computer Science Technical Report TR-2007-11*, page 31, 2007.

[29] Gunther Reißig. Computing abstractions of nonlinear systems. *IEEE Transactions on Automatic Control*, 56(11):2583–2598, 2011.

[30] Gunther Reissig, Alexander Weber, and Matthias Rungger. Feedback refinement relations for the synthesis of symbolic controllers. *IEEE Transactions on Automatic Control*, 62(4):1781–1796, 2016.

[31] Philipp Reist and Russ Tedrake. Simulation-based lqr-trees with input and state constraints. In *2010 IEEE International Conference on Robotics and Automation*, pages 5504–5510. IEEE, 2010.

[32] Matthias Rungger and Majid Zamani. Scots: A tool for the synthesis of symbolic controllers. In *Proceedings of the 19th international conference on hybrid systems: Computation and control*, pages 99–104, 2016.

[33] Max Schwenzer, Muzaffer Ay, Thomas Bergs, and Dirk Abel. Review on model predictive control: An engineering perspective. *The International Journal of Advanced Manufacturing Technology*, 117(5):1327–1349, 2021.

[34] Jean-Jacques E Slotine, Weiping Li, et al. *Applied nonlinear control*, volume 199. Prentice hall Englewood Cliffs, NJ, 1991.

[35] Robert F Stengel. *Optimal control and estimation*. Courier Corporation, 1994.

[36] Russ Tedrake, Ian R Manchester, Mark Tobenkin, and John W Roberts. Lqr-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052, 2010.

[37] Abraham P Vinod, Joseph D Gleason, and Meeko MK Oishi. Sreachtools: a matlab stochastic reachability toolbox. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 33–38, 2019.

[38] Alexander Weber, Elisei Macoveiciuc, and Gunther Reissig. Abs: A formally correct software tool for space-efficient symbolic synthesis. In *25th ACM International Conference on Hybrid Systems: Computation and Control*, pages 1–10, 2022.

[39] Teukolsky WH, Vetterling SA, and Flannery WT. Bp (2007),"section 18.1. the shooting method", numerical recipes: The art of scientific computing.

[40] Yunong Zhang, Binbin Qiu, and Xiaodong Li. Zd and zg control of simple pendulum system. In *Zhang-Gradient Control*, pages 123–130. Springer, 2021.

Appendix A

Matlab Code for Tracking the Reference Trajectory by LQR Control Design

The following is MATLAB code for the 3-dimensional nonlinear system experiment using the linear-quadratic regulator control design to track the reference trajectory.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%stage 1_infinite time horizon LQR design
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
mypi = 3.14159265358;
epsilon = 1e-7;
Q=[10 0 0;0 1 0;0 0 1];R=[20];%LQR parameters
x=sym('x',[3 1]);%variables
u=sym('u',[1 1]);
Nx=length(x);
x_bar=sym('x_',[Nx,1]);
x_=x_bar;
%the dynamics of the system
f_original = [ -cos(u)+sin(x(3))+1 ;
               sin(u)+sin(x(1));
               exp(x(2))+u-1];
eq_x=[0.0;0.0;0.0]; eq_u=[0.0];%equalibrium point
[A_l, B_l] = linearization(f_original...
    , x, u, eq_x, eq_u);%linearization around the equalibrium point
N=zeros(3,1);
%solving algebraic Riccati equation
[K, S, ~] = lqr(A_l, B_l, Q, R);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%stage 2_finite time horizon LQR design
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load('importing your reference trajectory here')
t = soln.grid.time;%ref trajectory's simulating t_{f}
%S_{stage1}
S0=[20.3206,   10.1431,   14.4684;
   10.1431,   10.4979,    9.7904;
   14.4684,    9.7904,   14.6787];
nTime=200;%the number of intervals
tSol=linspace(0,t(end),nTime);
ttSol=fliplr(tSol);
%solving the Riccati equation
[T S] = ode45(@(t,S)Riccartieqn(t, S), ttSol, S0);
T=flipud(T);
S=flipud(S);
Ns = sqrt(size(S, 2));
t_temp = T.';
X_data = soln.interp.state(t_temp);%ref trajectory's states
x1=z_data(1, :);x2=z_data(2, :);x3=z_data(3, :);
u_data = soln.interp.control(t_temp);%ref trajectory's control inputs
%linearization around the reference trajectory
f_Bt([xt;ut]) = B_t;
val_B = f_Bt(x1,x2,x3, u_data);
f_At([xt;ut]) = A_t;
val_A = f_At(x1,x2,x3, u_data);
B_mat = zeros(length(t_temp), 3);
for i=1:length(t_temp)
```

```
    B_mat(i, 1) = double(val_B{1}(i));
    B_mat(i, 2) = double(val_B{2}(i));
    B_mat(i, 3) = double(val_B{3}(i));
end
A_mat = zeros(size(val_A, 1), size(val_A, 2), length(t_temp));
for i=1:length(t_temp)
    for j = 1:size(val_A, 1)
        for k = 1:size(val_A, 2)
            A_mat(j, k, i) = double(val_A{j, k}(i));
        end
    end
end
%feedback control gains
K_t = zeros(length(t_temp), 3);
for i=1:length(t_temp)
    K_t(i, :) =  (1/R).* (B_mat(i, :) * reshape(S(i, :), [3 3]));
end
for i=1:5%the number of simulating potential trajectories
csize=0.3;%the random region
plot3(0,0,0,'bo','MarkerSize',7,'LineWidth',3)
hold on
X_init=[x1(1)+csize*(2*rand(1)-1),x2(1)...
    +csize*(2*rand(1)-1),x3(1)+csize*(2*rand(1)-1)];%random initial state
plot3(x1,x2,x3,'b')%reference trajectory
xlabel('x1') ylabel('x2') zlabel('x3')
grid on
syms xx yy zz %variables
%linear interpolating
Ref=soln.interp.state(tSol);
x1_Ref=Ref(1,:);x2_Ref=Ref(2,:);x3_Ref=Ref(2,:);
uRef=soln.interp.control(tSol);
xFit=polyfit(tSol,x1_Ref,10);
yFit=polyfit(tSol,x2_Ref,10);
zFit=polyfit(tSol,x3_Ref,10);
uFit=polyfit(tSol,uRef,10);
Kx=K_t(:,1);Kx=Kx';Ky=K_t(:,2);Ky=Ky';Kz=K_t(:,3);Kz=Kz';
KxFit=polyfit(tSol,Kx,10);
KyFit=polyfit(tSol,Ky,10);
KzFit=polyfit(tSol,Kz,10);
controller = @(t,X) feedbackcontroller(t,X(1,:),X(2,:),X(3,:) ...
    ,xFit,yFit,zFit,uFit,KxFit,KyFit,KzFit);%feedback controller
Func = @(t,X)3dnonlinear(t,z,controller(t,X)); %leading-in the system
[~, x_real] = ode45(Func,[0,t(length(t))],X_init);%potential trajectory
plot3(x_real(:,1),x_real(:,2),x_real(:,3),'r-')
end
```

Appendix B

Some Supplements about Abstraction-based Control Design

The following algorithm is the Pseudo-code of synthesizing the controller for the abstract control system S_2 with reachability specification.

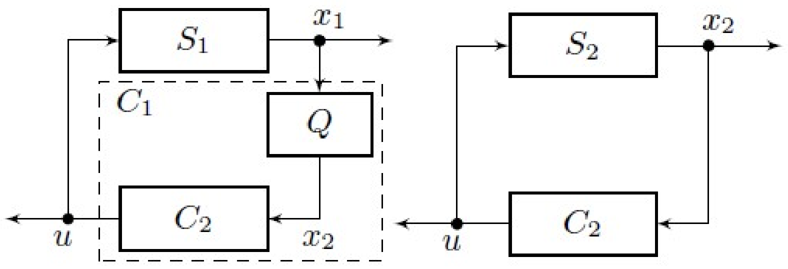

Figure B.1: The flow of the concrete control problem (left), including the refinement step of transferring the abstract controller to the concrete controller for the cyber-physical system; the flow of the abstract control problem (right) from [32].

Algorithm 5: Synthesizing the reachability specification for the finite system

```
Given $S_2 = (X_2, U_2, F_2), Z_2 \subseteq \bar{X}_2, u_0 \in U_2$;
$Q := Z_2$;
$V := \infty$;
$M := 0$;
$E := \emptyset$;
**for** $x_2 \in \bar{X}_2$ **do**
    $D(x_2) := \emptyset$;
    **if** $x_2 \in Z_2$ **then**
        $V(x_2) := 0$;
        $D(x_2) := \{u_0\}$;
**while** $Q \neq \emptyset$ **do**
    $x'_2 :\in Q$;
    $Q := Q \backslash \{x'_2\}$;
    $E := E \cup \{x'_2\}$;
    **for** $(x_2, u) \in F_2^{-1}(x'_2)$ **do**
        $M(x_2, u) := \max\{M(x_2, u), V(x'_2)\}$;
        **if** $F_2(x_2, u) \subseteq E$ *and* $V(x_2) > 1 + M(x_2, u)$ **then**
            $V(x_2) := 1 + M(x_2, u)$;
            $Q := Q \cup \{x_2\}$;
            $D(x_2) := \{u\}$;
Then outputs are $D$ and $V$.
```