ROBOT MOTION PLANNING VIA CURVE SHORTENING FLOWS

BY

YINAI FAN

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Mechanical Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2021

Urbana, Illinois

Doctoral Committee:

Professor Prashant Mehta, Chair
Associate Professor Mohamed-Ali Belabbas, Director of Research
Professor Geir Dullerud
Professor Daniel Liberzon

ABSTRACT

This work will present a series of developments of geometric heat flow method in robot motion planning and estimation. The key of geometric heat flow is to formulate the motion planning problem into a curve shortening problem. By solving the geometric heat flow, an arbitrary initial curve can be deformed to a curve of minimal length, which corresponds to a feasible motion. Preliminary theories and algorithms for motion planning based on geometric heat flow have been developed for driftless control affine systems. The main contribution of this research is to extend the algorithm to robotic systems, which are dynamic systems with drifts and different types of constraint. Early stages of the research focus on adapting the algorithm to solve motion planning problems for systems with drift. To tackle systems with drift, actuated curve length and affine geometric heat flow is proposed. The method is then enriched to solve robot gymnastics motion planning, in which the effect of state constraints is encoded into curve length. Free boundary conditions are also studied to enforce the conservation of the robot's momentum. The second stage of the research focus on the construction of the geometric heat flow framework for robot locomotion planning, which involves hybrid dynamics due to contact. The activation and deactivation of phase-dependent constraints are controlled by activation functions. Lastly, to solve 3D problems in robotics, planning and estimation in $SO(3)$ space is formulated using the geometric heat flow method.

To my parents, wife and grandmother, for their love and support.

ACKNOWLEDGMENTS

Throughout the writing of this dissertation I have received a great deal of support.

First and foremost I would like to thank my supervisor, Professor Mohamed-Ali Belabbas, for his invaluable advice, insightful feedback, and patience during my PhD study. I would like to thank Professor Seth A. Hutchinson and Professor Hae-Won Park, who guided me in my early years of PhD study and enlightened my path. I also appreciate all the valuable advice from my committee, Professor Prashant Mehta, Professor Geir Dullerud, and Professor Daniel Liberzon.

I would like to acknowledge Department Mechanical Engineering for the great help with the milestones in my PhD study. I would also like to thank Daniel J. Block and Department of Electrical and Computer Engineering for the memorable teaching assistance experience over the years.

My appreciation also goes out to my parents, my wife and friends. I could not have completed this dissertation without their endless encouragement and support.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Motivation

In the past decades, robotics hardware has become more and more reliable, especially for floating based robots and soft robots, making the vision of agile, walking, diving and jumping robots closer to reality. Such robots include, among others, the MIT Cheetah [1], the bipedal robot Cassie robot made by Agility Robotics, Salto robot [2] and BR^2 soft robot [3]. The motion of floating robots are agile and dynamic, which need to be carefully planned and controlled. An badly chosen trajectory of the robot usually results in catastrophic failure, such as a humanoid robot landing on head after performing an somersault. In contrast, a well-chosen trajectory will need minimum effort of the low level tracking controller and can result in safer motions. On the other hand, the pose estimation of soft robot is challenging. This is mainly because the curvilinear nature of soft robot, which excludes the direct use of traditional sensors.

Several challenges arise when planning the motion for robot. Nonlinear and hybrid dynamics, underactuation, varying type of constraints are the main difficulties for motion planning of floating robots. Thanks to the rapid development of computation power, the main stream of robot motion planning in the past decade and now are optimization based methods. Generally, the optimization based methods formulate the motion planning problem into an optimal control problem. Indirect and direct methods are two branches of solving optimal control problem. Indirect methods tend to find necessary and sufficient conditions analytically, and then discretize these conditions, constructing a constrained parameter optimization problem. For example, Hamilton-Jacobi-Bellman (HJB) equation is usually solved in direct method. The indirect method encounters difficulties for complex constraints introduced

by robotics systems. More recent development on optimal control is direct method and numerous algorithms has been proposed. The direct methods discretize the problem first and then convert the optimal control problem to a constrained parameter optimization problem. These methods rely on powerful nonlinear programming solvers that have been developed recently. Among the numerous algorithms, direct collocation method [4] [5] [6] is an efficient way for planning motions and finding the optimal control, where the system dynamics is encoded as equality constraints between adjacent grids. On the computational side of optimization-based method, different nonlinear solvers are used, such as *fmincon* in *Matlab*, *Snopt* and *IPOPT*. To most of the robotics researchers, the bulk of the research efforts are on the formulation from robotics systems to optimization problems, however, the optimization solvers are black boxes, therefore it is hard to analytically make improvement if the motion given by the solver is unsatisfactory.

Motion planning using *geometric heat flow (GHF)* is a new technique in motion planning, which relies on formulating the motion planning problem into a curve shortening problem and solve the motion using partial differential equations. In [7] and [8], fundamental theorems and algorithms are proposed for driftless control affine system with non-holonomic constraints and obstacle avoidance constraints. The method is a homotopy method: given an initial state and a final desired state, x_i and x_f respectively, and an arbitrary curve joining x_i to x_f in state-space, the method deforms the curve into a curve of minimal length, which corresponds to an admissible curve joining x_i to x_f. The key step is to build a customized Riemannian metric that defines the curve length. It is shown that in a driftless control affine system, the convergence to a feasible motion is guaranteed for any initial guess, under some proper assumptions. The evolution of trajectory can be analytically represented by geometric heat flow. The solving process is done using PDE solver whose structure is usually clear to users.

To scale up the scope of application for more complex robotics systems, the geometric heat flow method needs to be reconstructed and specialized in several directions. This work is a collection of developments on these directions, which will be introduced in the following section.

1.2 Contributions

- First, the method need to be adapted to system with drift. In most of the robotics systems, the direct input is usually actuation forces or torques applied on the robot's joints. These inputs steer the robot's position and orientation implicitly by directly steering the acceleration. This "double integrator" nature introduces a drift term to the system dynamics, which can be expressed as

$$\dot{x} = F_d(x) + F(x)u$$

  in general, where $F_d(x)$ is the drift term, and $F(x)u$ is the control affine term with control u. From a theoretical point of view, the Chow-Rashevski theorem provides us with conditions under which a non-holonomic system without drift is controllable [9], but with the presence of a drift term, no equivalent result is known: the general case of controllability of nonlinear systems with drift is still a largely open problem. We extended the geometric heat flow to encompass dynamics with drift in [10] and we term the resulting flow the *affine-geometric heat flow (AGHF)*. The method works by "deforming" an arbitrary path between x_i and x_f into a curve of minimal *actuated curve length*, which corresponds to an almost feasible trajectory for the system with drift. The actuated curve length is the length of the curve $x(t)$ after actuated by the drift term.

- Second, the method should handle conserved quantity and state constraints. This capability is particularly crucial for robot motion planning with no interaction from environment. For dynamics which have conserved quantities, obtaining the boundary conditions (BC) is one of the main issues. Indeed, the boundary conditions assign specific values to these quantities, and these cannot be altered during the motion by the controls. In addition, the states are in general constrained due to hardware limitations. A typical but challenging task in this ca tegory is gymnastics planning, where the robot is in the air, and the momentum is conserved due to the absence of contact forces. Furthermore, the joints have limited range of motion, which prohibit to maintain a desired pose by spinning the limbs as flywheels. In [11] and [12], we made such

extensions and generated motions for robot gymnastics. Transversality conditions are equipped to the AGHF framework to allow free BC, so that the AGHF can automatically find the BC that conserve the momentum and avoid bad choices of predefined BC. New states are added to keep track of the violation of different types of state constraints - equality and inequality constraints. The curve length is augmented such that violation of state constraint results in a large curve length. With the new version of AGHF frame work, a variety of robot gymnastics motions can be generated.

- The third specialization is to plan motions for legged robots, whose dynamics is hybrid. Planning dynamic motions of legged robots has become an increasingly important topic, due in part to improved robot design and hardware, and in part to higher on-board computational capacity. The major difficulty for legged locomotion planning lies in its *hybrid nature*: the dynamics of legged robots is governed by a set of equations and constraints depending on whether there is contact with the ground. Hybrid systems are well known to be difficult to handle; in fact, open questions remain even in the case of linear dynamics [13]. This difficulty stems in part from the fact that most motion planning methods, including the earlier AGHF framework [12], do not admit natural or obvious extensions to handle hybrid dynamics, often resulting in ad-hoc modifications that are difficult to analyze. To carry out this adaption, in [14], we show that the geometric method we proposed for motion planning extends naturally to handle hybrid dynamics. Precisely, with the switching time for different dynamics predefined, we approximated the switching using some customized activation functions. Then we show how the Ansatz developed in [7, 10, 12], contending that motion planning problems can be encoded into Riemannian metrics with the help of these activation functions, can be applied to plan legged locomotion. Moreover, a new mechanism is created to automatically plan the switching time of different dynamics, which is capable of generating more natural motions.

- The last specialization is focusing on soft robot estimation. The pose of a soft robot arm is characterized by the position and orientation of each cross section along its fixed length. Therefore, the pose is a function of a single parameter – the length, which is analog to the time in motion

planning problem. Therefore, the pose estimation of soft robot arm is equivalent as a motion planning problem for a particle with position and orientation, for a fixed terminal time. The path traveled by the particle can be considered the pose of the soft robot arm. The geometric heat flow method can be applied, but with extension to plan for orientation. Depending on the specific representation of orientation, or $SO(3)$ space - Euler angles, quaternion and rotation matrix, the $SO(3)$ state evolves under different dynamics. Earlier works [11] [12] [14] limited the motion in 2D plane and avoided states in $SO(3)$. Extension is needed to tackle 3D orientation states. To this purpose, we choose to represent the orientation using unit quaternions and encode the quaternion dynamics into AGHF framework. To have an accurate estimation of the soft arm's pose, vision based sensor data, gravity and external load is encoded into the curve length, which reflects the minimum potential energy principle. The outcome is that the algorithm now is able to deform an arbitrary initial pose to a feasible pose, which includes valid $SO(3)$ states, and has the effect of gravity, external load. The initial pose can be arbitrary, and does not necessarily include valid $SO(3)$ states. This planning technique is then applied on several scenarios of soft robot arm pose estimation.

CHAPTER 2

PRELIMINARIES

A fundamental problem in robotic motion planning is to find a trajectory which meets the various constraints stemming from the system dynamics, which can be of holonomic or non-holonomic type, and obstacle avoidance constraints, which include constraints on the magnitude of some of the variables describing the system (e.g., a maximal turning radius), or obstacles present in physical space. A new method is proposed in [7] and [8] to find a trajectory which takes into account all the above constraints, we call such a trajectory admissible. The method is a homotopy method: given an initial state and a final desired state, x_i and x_f respectively, and an arbitrary curve joining x_i to x_f in state-space, the method deforms the curve into an admissible curve joining x_i to x_f. Preliminary version of this method that plans the motion for control affine driftless system with only non-holonomic constraints is presented in [7]. In this section, we summarize some preliminary results.

2.1 Geometric Heat Flow (GHF)

2.1.1 Driftless System and Curve Length

We present some background and notation needed to explain the method. We refer to as vehicle/robot/plant whose motion we desire to plan as *the system*. The system is assumed to follow an affine in the control dynamics, which is defined as follows:

Definition 2.1 (Driftless Control affine system)**.** *A controlled differential equation $\dot{x} = f(x, u)$, with $x \in M$, $u \in \mathbb{R}^p$ is called affine in the control if it can be expressed as*

$$\dot{x} = F_f(x)u \tag{2.1}$$

where

$$F_f(x) = \begin{pmatrix} f_1(x) & f_2(x) & \cdots & f_p(x) \end{pmatrix} \tag{2.2}$$

where $x \in M$ with M a (at least locally) differentiable manifold called the configuration space, $f_i(x)$, $1 \leq i \leq p$ the actuation vector fields, and $u := (u_1, \ldots, u_p)^\top \in \mathbb{R}^p$ the controls. All vector fields are assumed to be smooth.

We refer to as *workspace* the physical environment in which the system lives. We denote by $\text{span}_x\{g_i\}$ the (real) vector space spanned by the vectors $g_i(x)$.

We call a *curve* in configuration space a piecewise differentiable function $x(t) : [0, T] \to M$, where $T > 0$, and refer to $x(0)$ and $x(T)$ as start-point and end-point, respectively, of $x(t)$. We refer to them collectively as *end-points*. We call the *image* of a curve a *path*; a path is thus a geometric object (a collection of "contiguous states") and the times at which each point in a path is visited are not specified.

We now introduce a central notion for this work, the notion of homotopy:

Definition 2.2 (Homotopy)**.** *A fixed end-points homotopy between the two curves $x_1(t)$ and $x_2(t)$ with the same end-points (i.e., $x_1(0) = x_2(0)$ and $x_1(T) = x_2(T)$) is a differentiable function $x(t, s) : [0, T] \times [0, \infty) \to M$ with the properties:*

$$\begin{aligned} x(0, s) &= x_1(0) && \text{for all } s \geq 0 \\ x(T, s) &= x_1(T) && \text{for all } s \geq 0 \end{aligned}$$

For each fixed s, $x(t)$ is a curve in M. The *length* of the curve $x(t)$ is defined with respect to an norm on the tangent bundle TM of M. In the following, one can assume that $M = \mathbb{R}^n$ and the tangent space of M at $x \in M$, denoted by T_xM is also $\mathbb{R}^n$. The *length* of a curve can be defined by a *Riemannian inner product*:

Definition 2.3 (Riemannian inner product)**.** *A Riemannian inner product on M is a piecewise differentiable symmetric positive definite bilinear form $G(x) : T_xM \times T_xM \to \mathbb{R}$. With a slight abuse of notation, we also denote by $G(x)$ its matrix representation in coordinates.*

Hence, $G(x)$ is an x-dependent positive definite symmetric matrix. The

length of a curve $p(t)$ is then given by

$$L(x) := \int_0^T \sqrt{\dot{x}^\top(t) G(x(t)) \dot{x}(t)} dt. \tag{2.3}$$

Finally, we introduce the Christoffels' symbols associated to $G(x)$. To this end, denote by g_{ij} the ijth entry of the matrix representation $G(x)$, and by g^{ij} the ijth entry of the matrix $G^{-1}(x)$. The Christoffel's symbols are

$$\Gamma^i_{jk}(x) := \frac{1}{2} \sum_l g^{il} \left(\frac{\partial g_{lj}}{\partial x_k} + \frac{\partial g_{lk}}{\partial x_j} - \frac{\partial g_{jk}}{\partial x_l} \right) \tag{2.4}$$

2.1.2 GHF as Curve Shortening Flow

Our method proceeds with solving the following GHF equation:

$$\frac{\partial}{\partial s} x_i(t,s) = \frac{\partial^2}{\partial t^2} x_i(t,s) + \sum_{j,k} \Gamma^i_{jk} \frac{\partial x_j}{\partial t} \frac{\partial x_k}{\partial t} \quad i = 1, 2, \ldots, n \tag{2.5}$$

where Γ^i_{jk} are the Christoffel symbols introduced in (2.4) for the inner product defined in the previous subsection.

We now elaborate on the origin of Eq. (2.5): it is a type of curve-shortening flow [15], called a *mean-curvature flow* for a 1-dimensional manifold (i.e. a curve) or *geometric heat flow*. For an introduction to mean-curvature flows in arbitrary dimensions, see [16]. For clarity of exposition, we present first the flow in two dimensional plane with the Euclidean inner product. We briefly mention steps that need to be taken for the general flow below.

Consider a curve $p(t) : [0,1] \to \mathbb{R}^2 = (p_1(t), p_2(t))$, as depicted in Fig. 2.1. The scalar curvature [17] of p at $p(t)$ is defined as $\kappa(p(t)) = \|\ddot{p}\|$. Denote by $N_{p(t)}$ the unit normal vector pointing "inward".The curvature of p at $p(t)$ is then $\kappa(p(t))N(p(t))$.

The mean-curvature flow for this curve is defined as follows:

Definition 2.4 (Mean curvature flow)**.** *Consider a family of curves $p(t,s)$, $s \geq 0$, where for each s_0 fixed, $p(t,s_0) : [0,1] \to \mathbb{R}^2$ is a curve joining x_0 to x_1, and $p(t,0)$ is the original curve. Then the mean-curvature flow is the*

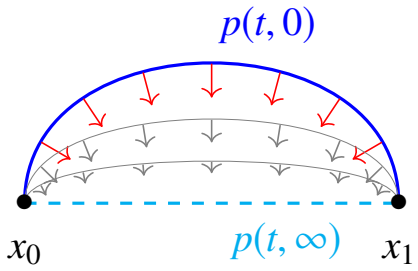


Figure 2.1: In the mean-curvature flow, the curve $p(t,0)$ is continuously deformed in the direction of its normal, depicted by the red arrows. The final curve is a straight line. In general, the final curve is a length minimizing curve. is the corresponding angles

partial differential equation

$$\frac{\partial p}{\partial s} = \kappa(p(t,s))N(p(t,s)).$$

Note that it is in fact a system of two PDEs.

Looking at Fig. 2.1, it is easy to conclude intuitively that $\lim_{s\to\infty} p(t,s)$ converges to a straight line between x_0 and x_1. This is also the *shortest path* between x_0 and x_1 for the usual Euclidean metric. This is no accident, and we can show that in general the solution of this PDE converges to a curve of minimal length. For our purpose, we need to extend this idea in *two* directions: to (i) curves in higher dimensions and (ii) to a general Riemannian metric (or more precisely, inner product). One can show, after some extensive algebraic manipulations which we omit here, that the equivalent of the flow for a general curve in a Riemannian manifold is exactly the geometric heat flow presented in Eq. (2.5).

2.2 Motion Planning with GHF

2.2.1 Riemannian inner product Construction

Holonomic constraints can be formulated as a set of equations

$$q_i(x) = 0, \quad i = 1, 2, \cdots, m_h \tag{2.6}$$

For each i and an infinitesimally small motion δx, we have the approximation $q_i(x_0 + \delta x) \approx q_i(x_0) + \frac{\partial q_i}{\partial x}\delta x$. In order to respect the constraint, δx needs to satisfy $q_i(x_0 + \delta x) = q_i(x_0) = 0$, thus we have $\frac{\partial q_i}{\partial x}\delta x = 0$. This means that for $x(t)$ to be an admissible motion that obeys the constraints, the direction of motion δx needs to be orthogonal to the vectors $\frac{\partial q_i}{\partial x}$ for all i; in other words, it means the *inadmissible* directions of motion are span $\left\{\frac{\partial q_i}{\partial x}\right\}$.

We now turn our attention to non-holonomic constraints. Formally, we define:

Definition 2.5 (Non-holonomic Constraints)**.** *Non-holonomic constraints are constraints that cannot be formulated by 2.6, that is, non-holonomic constraints are not integrable. In this work, we express the non-holonomic constraints as a set of constraints on the allowed velocities $\dot{x}$ when at state x as follows:*

$$\dot{x}^\top f_{c,j}(x) = 0, \quad j = 1, 2, \cdots, m_n. \tag{2.7}$$

The non-holonomic character of the constraints, which is reflected in the fact that they cannot be expressed as $\frac{d}{dt}q_n(x) = 0$ for some function $q_n(x)$.

In our formulation, the fact that a constraint is non-holonomic does *not* play any particular role insofar our local encoding of the constraints is concerned; in fact, the inadmissible directions of motion due to non-holonomic constraints in Def. 2.5 is span $\{f_{c,j}(x)\}$ for all $j = 1, 2, \cdots, m_n$. The inadmissible directions of motion due to holonomic constraints is span $\left\{\frac{\partial q_i}{\partial x}\right\}$ for all $i = 1, 2, \cdots, m_h$.

Non-holonomic constraints can be presented as above, e.g., as non-slippage constraints, but they can also be encoded in the dynamics of the system, which is then called non-holononic. For this latter case, consider the system of Eq. (2.1). We set $f_{f,i} = f_i$ and $f_{c,j}$ to be the m_n vectors orthogonal (for the Euclidean inner product) to $f_{f,i}$ for all $i = 1, \cdots, p$.

The method solves for an **admissible curve**:

Definition 2.6 (Admissible curve)**.** *Given a configuration space M, a set of holonomic, non-holonomic and obstacle avoidance constraints, an initial state x_{init} and a desired final state x_{fin}, provide a curve $x(t) : [0, T] \to M$ which respects these constraints and so that $x(0) = x_{\text{init}}$, and $x(T) = x_{\text{fin}}$, and provide the control u that drive a control system from x_{init} to x_{fin}. We recall that a curve that meets the constraints is an admissible curve, which is an admissible motion.*

Encoding the constraints We set $\bar{p} := n - m_n - m_h$. We define the $n \times (n - \bar{p})$ matrix $\bar{F}_c$ as the matrix with first m_h columns given by $\frac{\partial q_i}{\partial x}$ and the next m_n columns given by the $f_{c,j}$.

Assumption 2.1. *We assume that $\bar{F}_c(x)$ is of constant rank almost everywhere in M, and we denote this rank by l, and set $p := n - l$.*

If $m_h + m_n = l$, it is of *full column rank*, and we set $F_c(x) := \bar{F}_c(x)$. Otherwise $m_h + m_n > l$ and the constraints are not independent, in the sense that satisfying a *subset* of the constraints insures that *all* constraints are met. We set $F_c(x)$ to be a $n \times l$ matrix whose column span equals the column span of $\bar{F}_c(x)$. Such a matrix can be obtained, e.g., via the Gram-Schmidt process. Notice that F_c is of full column rank $l = n - p$ and the *column space of F_c contains all the inadmissible directions of motion.*

Next, find a rank p matrix $F_f(x) \in \mathbb{R}^{n \times p}$ such that

$$F_f(x)^\top F_c(x) = 0,$$

which again can be found using the Gram-Schmidt process. The column space of $F_f(x)$ contains all the directions in which the system can move when at state x, namely, the admissible directions of motion.

Note that in the absence of holonomic constraints, we can start with defining F_f with columns f_i as in Eq. (2.1) and choose F_c to satisfy the above relation. Set

$$F(x) = \begin{pmatrix} | & | \\ F_c(x) & F_f(x) \\ | & | \end{pmatrix} \tag{2.8}$$

Then $F(x) \in \mathbb{R}^{n \times n}$ and we define:

Definition 2.7 (Riemannian inner product for motion planning)**.** *The Riemannian inner product for motion planning is*

$$G(x) = F(x) D F^\top(x) \tag{2.9}$$

where $D = \mathrm{diag}([\underbrace{\lambda \cdots \lambda}_{n-p} \underbrace{1 \cdots 1}_{p}])$ is a constant matrix.

Using the interpretation of the length functional given in the previous section, it is easy to see that if $\dot{x}$ is a direction that respects the constraints, it

is not multiplied by λ in the inner product $\dot{x}^\top G(x)\dot{x}$ with H defined via (2.9), so $\dot{x}^\top G(x)\dot{x}$ will not be scaled by λ. On the other hand, if $\dot{x}$ is a direction that violates a constraint, it has some components lying in span $F_c(x)$, and consequently $\dot{x}^\top G(x)\dot{x}$ is large.

Finally, we record here that the partial derivative of G is given by

$$\frac{\partial G}{\partial x_i}(x) = 2FD\frac{\partial F^\top}{\partial x_i}(x),$$

which is needed for the computation of the Christoffels symbols.

2.2.2 Solving the GHF

With the Riemannian inner product $G(x)$ in (2.9) well defined, the GHF equation (2.5) can be derived. Solving the GHF equation will lead to a admissible curve.

The GHF equation is a PDE, which needs BC and initial condition (IC). We impose the boundary conditions

$$x(0,s) = x_{\text{init}}, x(T,s) = x_{\text{fin}}$$

and a user defined initial condition $v(t)$,

$$x(0,t) = v(t)$$

in order to find the solution. The initial curve $v(t)$ is an arbitrary curves satisfying the boundary conditions: $v(0) = x_{\text{init}}$ and $v(T) = x_{\text{fin}}$.

An important point here is that $v(t)$ does not need to satisfy any holonomic or non-holonomic constraints; it can be simply a curve drawn from x_{init} to x_{fin}.

Notice that for each $s \geq 0$ fixed, the solution $x(\cdot, s)$ represent a curve connecting x_{init} to x_{fin}. As we explain below, as s increases, $x(\cdot, s)$ is a curve that uses "less and less of the constrained directions", said precisely, $F_c^\top \frac{\partial}{\partial t} x(t,s)$ tends to zero. We set $s_{\max}$ to be the simulation time for the PDE and

$$x_{sol}(\cdot) = x(s_{\max}, \cdot).$$

2.2.3 Control Extraction

The control can be directly computed:

$$u(t) = F_f^\dagger(x_{sol}(t))\dot{x}_{sol}(t) \tag{2.10}$$

where $F_f^\dagger = (F_f^\top F_f)^{-1} F_f^\top$ is the pseudo- inverse of F_f. Notice that in the case x_{sol} is admissible, that is, if $\dot{x}_{sol}(t) = F_f v(t)$ for some control v,

$$u = F_f^\dagger \dot{x}_{sol} = (F_f^\top F_f)^{-1} F_f^\top F_f v = v \tag{2.11}$$

Thus we have recovered the control and ideally the system should exactly follow the path x_{sol}. Notice that $F_f F_F^\dagger$ is a minimal square error projection onto the column space of F_f, the control extracted from (2.10) will drive the system along a path that is close to x_{sol}, even if $\dot{x}_{sol}$ has small components in the constrained direction.

A path can be obtained by integrating the system dynamics using the control extracted, we call it the *integrated path*:

Definition 2.8 (Integrated path)**.** *The integrated path $\tilde{x}(t)$ is obtained by integrating the system dynamics* (2.1) *using extracted control* (2.10)*:*

$$\tilde{x}(t) = \int_0^t F_f(\tilde{x}(\tau))u(\tau)d\tau \tag{2.12}$$

with initial value $\tilde{x}(0) = x_{init}$.

With integrated path defined, we have the following proposition,

Proposition 2.1. *If the GHF solution $x_{sol}(t)$ is admissible, namely, if there exists $u \in \mathbb{R}^p$ so that the derivative $\dot{x}_{sol}(t)$ of $x_{sol}(t)$ satisfies* (2.1)*, the integrated path $\tilde{x}(t)$ is equal to $x_{sol}(t)$.*

Proof. The control $u(t)$ of the admissible solution $x_{sol}(t)$ can be extracted according to (2.11). Taking the time derivative for both side of (2.12) and substituting u with (2.10):

$$\dot{\tilde{x}} = F_f(\tilde{x})F_f^\dagger(x_{sol})\dot{x}_{sol} \tag{2.13}$$

substituting $\dot{x}_{sol}$ with $F_f(x_{sol})v(t)$ for some control v since x_{sol} is an admissible

solution:

$$\dot{\tilde{x}} = F_f(\tilde{x})F_f^{\dagger}(x_{sol})F_f(x_{sol})v(t) \tag{2.14}$$

$$= F_f(\tilde{x})(F_f(x_{sol})^{\top}F_f(x_{sol}))^{-1}F_f^{\top}(x_{sol})F_f(x_{sol})v(t) \tag{2.15}$$

$$= F_f(\tilde{x})v(t) \tag{2.16}$$

Therefore, if $x_{sol}(0) = \tilde{x}(0)$, the integrated path is equal to the AGHF solution. □

CHAPTER 3

THE AFFINE GEOMETRIC HEAT FLOW: CONVERGENCE PROPERTIES AND BASIC EXAMPLES

Given a control system

$$\dot{x} = f(x, u) \tag{3.1}$$

evolving on a differentiable manifold M, and two points $x_i, x_f \in M$, the motion planning problem in time $T > 0$ is to find a control $u^*(t)$ that *steers* the system from x_i to x_f in T units of time, i.e. so that the solution $x^*(t)$ of Eq. (3.1) with $u = u^*$ and $x^*(0) = x_i$ yields $x^*(T) = x_f$. Due to its ubiquity in control applications ranging from robotics to autonomous wheeled vehicles, motion planning has been widely studied (see, e.g., [18, 19]) and a host of methods have been developed. One of the early control papers in which the issue of motion planning for non-holonomic systems was clearly addressed is [20], where motion planning is stated as a sub-Riemannian geodesic problem. For a more recent survey of this line of work, we refer to the recent monograph [9]. Another common approach to non-holonomic motion planning is to use sinusoidal control functions to, roughly speaking, generate the "Lie bracket" directions. See for example [21], and [22] for a very recent work on how oscillations can be used for orientation control in $SO(3)$. This idea is also used in derivative-free optimization [23, 24].

The major difficulties that can arise in motion planning problems are: 1. the nonholonomic character of the dynamics, 2. the presence of a drift term, 3. the presence of constraints on the inputs/states. From a theoretical point of view, the Chow-Rashevski theorem provides us with conditions under which a non-holonomic system without drift is controllable (see [9]), but in the latter two cases, no equivalent result is known: the general case of controllability of nonlinear systems with drift is still a largely open problem. Nevertheless, for some specific nonlinear systems with drift, path planning or control algorithms are given in [25, 26, 27].

While most of the methods mentioned above focused on addressing the

first difficulty, we propose here an approach that can address all three. More precisely, we propose a new variational method for motion planning. The novelty of the work lies in an extension of the *geometric heat flow* method of motion planning in Chapter 2, to encompass dynamics with constraints and drift. We term the resulting flow the *affine-geometric heat flow*. The method works by "deforming" an arbitrary path between x_i and x_f into an almost feasible trajectory for the system, from which we can extract the controls u^* that drive the system from x_i to x_f approximately.

3.1 Affine Geometric Heat Flow (AGHF)

Notation: With a slight abuse of notation, we define for $G(x) \in \mathbb{R}^{n\times n}$, $f, g \in \mathbb{R}^n$

$$\left[f\left(\frac{\partial G}{\partial x}\right)g\right] := \begin{pmatrix} f^\top \frac{\partial G}{\partial x_1} g \\ \vdots \\ f^\top \frac{\partial G}{\partial x_n} g \end{pmatrix} \in \mathbb{R}^n;$$

i.e. $\left[f\left(\frac{\partial G}{\partial x}\right)g\right]$ is the vector in $\mathbb{R}^n$ with ith entry $f^\top \frac{\partial G}{\partial x_i} g$, where $\frac{\partial G}{\partial x_i}$ is the $n \times n$ matrix with kl entry $\frac{\partial G_{kl}}{\partial x_i}$. We furthermore use the notation

$$(f \cdot G) := \left(\sum_{l=1}^{n} f_l \frac{\partial G_{ij}}{\partial x_l}\right)_{ij} \in \mathbb{R}^{n\times n}. \tag{3.2}$$

3.1.1 System with Drift

We approach the motion planning problem by first solving the trajectory planning problem defined below:

Trajectory planning problem

Let $M = \mathbb{R}^n$ and refer to M as the configuration space. Note that M can more generally be a C^2-differentiable manifold. We consider the system affine in control:

Definition 3.1 (Control affine system)**.** *The dynamics affine in the control is given by*

$$\dot{x} = F_d(x) + F(x)u \tag{3.3}$$

where, for all $x \in \mathbb{R}^n$, $F_d(x) \in \mathbb{R}^n$ is a vector representing the drift dynamics when in state x and the columns of $F(x) \in \mathbb{R}^{n \times m}$ are the admissible control directions.

Assumption 3.1. *Both $F_d(x), F(x)$ are assumed to be at least C^2, Lipschitz with constants L_1, L_2 respectively, and we assume that $F(x)$ is of constant rank almost everywhere in $\mathbb{R}^n$.*

Note that these assumptions can be weakened at the expense of longer analysis. We focus here on the the case $n \geq m$; that is on potentially *under-actuated* dynamics.

Recall that $x_i, x_f \in M$ are the desired initial and final states respectively, and $T > 0$ is the fixed time allowed to perform the motion. We can define the admissible controls:

Definition 3.2 (Admissible controls)**.** *The set of admissible controls is $\mathcal{U} := L^2([0,T] \to \mathbb{R}^m)$, that is, square integrable functions defined over the interval $[0,T]$.*

We set

$$\mathcal{X} := \{x(\cdot) \in AC([0,T] \to \mathbb{R}^n) : x(0) = x_i, x(T) = x_f\},$$

the space of *absolutely continuous* $\mathbb{R}^n$-valued functions with start- and end-values x_i and x_f respectively.

Then, we can define the admissible solution and feasibility:

Definition 3.3 (Admissible solution)**.** *We call any $x(\cdot) \in \mathcal{X}$ an* **admissible solution** *if there exists $u \in \mathcal{U}$ so that the generalized derivative $\dot{x}(t)$ of $x(\cdot)$ satisfies* (3.3)*. Denote by $\mathcal{X}^* \subseteq \mathcal{X}$ the set of admissible solutions. The* **trajectory planning problem** *(from x_i to x_f with time T) is* **feasible** *if $\mathcal{X}^* \neq \emptyset$. All open sets in $\mathcal{X}$ are with respect to the natural norm $\|x\|_{AC} := \int_0^T (|x(t)| + |\dot{x}(t)|)\, dt$.*

We additionally introduce the space of *continuous controls* $\mathcal{U}' := C^0([0,T] \to \mathbb{R}^m)$, to which correspond differentiable trajectories $\mathcal{X}' := \{x(\cdot) \in C^1([0,T] \to \mathbb{R}^n) : x(0) = x_i, x(T) = x_f\}$. It is well-known that $\mathcal{U}'$ is a dense subspace in $\mathcal{U}$ with respect to the L_2 norm and that $\mathcal{X}'$ is a dense subspace in $\mathcal{X}$ with respect to the $\|\cdot\|_{AC}$ norm. Working over $\mathcal{X}'$ instead of $\mathcal{X}$ allows us to "smoothly deform" a curve, a term which is more rigorously defined later.

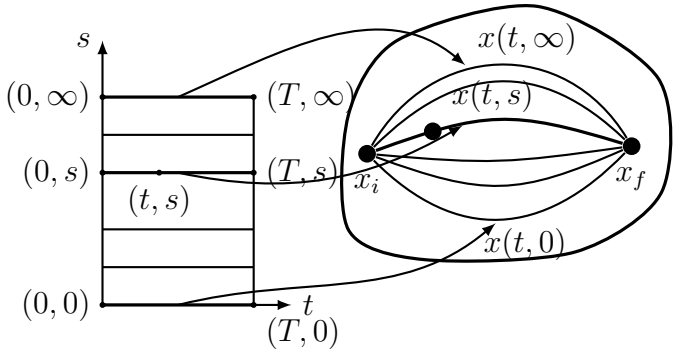


Figure 3.1: Homotopy of trajectories joining x_i to x_f in M.

3.1.2 AGHF as Curve Shortening Flow

We now briefly recap the geometric framework we use to cast and solve the trajectory planning problem in Chapter 2, and then introduce the main object introduced in this work: the *affine geometric heat flow*. We refer to our earlier work [7] for a more detailed presentation and more examples about the general framework. We rely on differentiable **homotopies** of curves (i.e., differentiable "deformations" of a curve), where the variable s is the homotopy parameter; precisely, we use $x(t,s) : [0,T] \times [0, s_{\max})$ where for *each* s *fixed*, $x(\cdot, s) \in \mathcal{X}'$.

Let $G(x)$ be a positive definite matrix defined on M, which we refer to as the Riemannian **metric**. We denote by ∇ the **Levi-Civita** connection of $G(x)$ as in [28], and by $\nabla_f g$ the covariant derivative of the vector field g along the vector field f. Recall that if $a(t) = \sum_{k=1}^{n} a_k(t) e^k$, where $a_i(t)$ are real numbers and e^k basis vectors, is a vector field along a curve $x(t)$, and g is a vector field defined in a neighborhood of $x(t)$, then $\nabla_a g := \frac{da}{dt} + \sum_{i,j,k} \Gamma^k_{ij} a_i g_j e^k$.

The so-called **geometric heat flow (GHF)** is a parabolic partial differential equation, which evolves a curve with fixed end-points toward a curve of minimal length: namely, given a Riemannian metric and an associated Levi-Civita connection ∇, the GHF is the PDE

$$\frac{\partial x(t,s)}{\partial s} = \nabla_{\dot{x}(t,s)} \dot{x}(t,s). \tag{3.4}$$

where $\dot{x}(t,s) := \frac{\partial x(t,s)}{\partial t}$ and $x(0,s) = x_i, x(T,s) = x_f$ are fixed. We refer the reader to [28] for a proof that this PDE yields a curve of minimal length. For applications of this flow to motion planning problems, and some illustrations

of its solutions, we refer [7]

To justify the introduction of the AGHF, we define the notion of *actuated length* [10] of a trajectory: this is, in a sense, the length of the trajectory disregarding the effect of the drift dynamics.

Definition 3.4 (Actuated curve length)**.** *We define the actuated length of a differentiable curve $x : [0, T] \to M$ by*

$$\int_0^T ((\dot{x} - F_d(x))^\top G(x)(\dot{x} - F_d(x)))^{1/2} dt. \tag{3.5}$$

Taking inspiration from the GHF, we introduce here what we term the **affine geometric heat flow (AGHF)**:

Definition 3.5 (Affine geometric heat flow)**.** *The affine geometric heat flow is*

$$\frac{\partial x(t,s)}{\partial s} = \nabla_{\dot{x}(t,s)} \left(\dot{x}(t,s) - F_d\right) + r(t,s) \tag{3.6}$$

where

$$r(t,s) = G^{-1} \left(\left(\frac{\partial F_d}{\partial x} \right)^\top G(\dot{x} - F_d) + \frac{1}{2} \left[(\dot{x} - F_d) \left(\frac{\partial G}{\partial x} \right) F_d \right] \right)$$

and $F_d = F_d(x(t,s))$, $G = G(x(t,s))$, etc.

The AGHF comprises two terms, introduced for distinct reasons: the first term can be thought of as minimizing the length of the curve, and the second term as insuring that the resulting trajectory is feasible for a system with drift vector field $F_d(x)$. In more detail, the first term is the covariant derivative of $\dot{x}(t,s) - F_d(x)$ in the direction $\dot{x}(t,s)$. To understand the origin of this term, recall that $\nabla_{\dot{x}}\dot{x}$ is the acceleration vector of the curve. Because the curve $x(t,s)$ is parametrized by length (i.e. $\|\dot{x}\| = 1$), for all s, updating the curve in the direction of its acceleration decreases its curvature. This can be thought of as the reason why the GHF minimizes length of the curve. In the AGHF, we replaced the term $\dot{x}$ by $\dot{x} - F_d(x(t,s))$; hence we subtracted from the tangent vector to the curve at $x(t,s)$ the drift vector field at that point. This terms thus *"minimizes the length" of the resulting curve* discounting the effect of the drift term. The idea behind this term is that since the drift vector field cannot be altered by the controls, it should not influence the computation of the curvature.

The role of the second term is to align the direction of the curve at x with the drift vector field at that point. To argue for this, we will describe how this term moves a point $x(t,s)$ of the solution curve at iteration s to a point $x(t,s+\delta s)$ for a small increment δs. Denote by $\langle v,w\rangle$ the inner product of $v,w\in T_xM$, and consider the function

$$P:TM\to\mathbb{R}:(x,v)\mapsto\langle v-F_d(x),F_d(x)\rangle \\ =\langle v,F_d(x)\rangle-\langle F_d(x),F_d(x)\rangle\,. \quad (3.7)$$

This function takes an element from the tangent bundle (x,v), with $x\in M$ and $v\in T_xM$ to yield the inner product of v with $F_d(x)$ minus $\langle F_d(x),F_d(x)\rangle$. We will assume that v is fixed, and consider $P_v:M\to\mathbb{R}:P_v(x):=P(x,v)$. The function clearly reaches its maximal value when $F_d(x)$ is aligned with v. Hence, the gradient flow of this function seen as a function from $M\to\mathbb{R}$ will tend to align $F_d(x)$ with v. Now one can show that the term $r(t,s)$ defined above is the *gradient of the function* $P_{\dot{x}}(x):M\to\mathbb{R}$ for the Riemannian metric G: the effect of this term is thus to move the curve (i.e. move $x(t,s+\delta s)$) so that $F_d(x(t,s+\delta s)$ is more aligned with $\dot{x}(t,s)$. Said otherwise, this term deforms the curve to that $F_d(x)$ is more aligned with $\dot{x}(t,s)$.

We highlight that the flow can only update $x(t,s)$, and not $\dot{x}(t,s)$, and thus we move $x(t,s)$ in search of $F_d(x(t,s))$ more aligned with $\dot{x}(t,s)$.

Lemma 3.1 below provides a mathematical justification of the form of the AGHF and additional insights, and we furthermore show in Sec. 3.3 that it does indeed converge to admissible paths on various examples. Note that when $F_d=0$, then $r(t,s)\equiv 0$ and the AGHF reduces to the GHF.

3.1.3 On the convergence of the AGHF

The AGHF is a nonlinear set of PDEs, and thus the existence of a solution is not guaranteed a priori even for short time. We provide here an analysis and convergence guarantees.

To this end, define the Lagrangian L by

$$L(x,\dot{x})=\frac{1}{2}(\dot{x}-F_d(x))^\top G(x)(\dot{x}-F_d(x)) \quad (3.8)$$

Given L, the *action functional* is defined on $\mathcal{X}$ as

$$\mathcal{A}(x(\cdot)) := \int_0^T L(x(t), \dot{x}(t))dt. \tag{3.9}$$

recall that the space of absolutely continuous $\mathbb{R}^n$-valued functions, $\mathcal{X}$, is defined in Definition 3.2.

The Euler-Lagrange equation is given by:

$$\frac{\partial L}{\partial x} - \frac{d}{dt}\frac{\partial L}{\partial \dot{x}} = 0 \tag{3.10}$$

Lemma 3.1 (Nonincreasing energy)**.** *Let $x^*(t)$ be a steady-state solution of the AGHF* (3.6)*. Then $x^*(t)$ is an extremal curve for $\mathcal{A}$ in* (3.9)*. Furthermore, $\mathcal{A}$ decreases along the solutions of the AGHF; i.e. if $x(t,s)$ is such a solution, then $\frac{d}{ds}\mathcal{A}(x(\cdot,s)) \leq 0$, and equality holds only if $x(\cdot,s)$ is an extremal curve for $\mathcal{A}$.*

Proof. Firstly, by first order approximation we have

$$x(t, s+\delta) = x(t,s) + \delta x_s(t,s) + o(\delta)$$

Plug it into the first order variation of L, we have

$$\begin{aligned} V(x(\cdot, s+\delta)) &= \int_0^T L(x(t,s+\delta), x_t(t,s+\delta))dt \\ &= \int_0^T L(x(t,s), x_t(t,s)) + (\delta x_s(t,s) + o(\delta))^\top \frac{\partial L}{\partial x} + \\ &\qquad \left(\frac{d}{dt}(\delta x_s(t,s) + o(\delta))\right)^\top \frac{\partial L}{\partial x_t} + o(\delta)dt \\ &= V(x(\cdot,s)) + \delta \int_0^T x_s(t,s)^\top \frac{\partial L}{\partial x} + x_{ts}(t,s)^\top \frac{\partial L}{\partial x_t}dt + o(\delta), \end{aligned} \tag{3.11}$$

where all $o(\delta)$ terms are collected together. Use integration by parts for the $x_{ts}(t,s)^\top \frac{\partial L}{\partial x_t}$ term, we have

$$\begin{aligned} &V(x(\cdot, s+\delta)) = V(x(\cdot,s)) + \\ &\delta\left(x_s(t,s)^\top \frac{\partial L}{\partial x_t}\bigg|_0^T + \int_0^T x_s(t,s)^\top \frac{\partial L}{\partial x} - x_s(t,s)^\top \frac{d}{dt}\frac{\partial L}{\partial x_t}dt \right) + o(\delta). \end{aligned} \tag{3.12}$$

By our boundary conditions (3.14) we have $x_s(0,s) = x_s(T,s) = 0$ for all $s \geq 0$ and hence the integrated term $x_s(t,s)^\top \frac{\partial L}{\partial x_t}\Big|_0^T$ vanishes. Thus

$$V(x(\cdot, s+\delta)) - V(x(\cdot, s)) = \\ \delta \int_0^T x_s(t,s)^\top \left(\frac{\partial L}{\partial x} - \frac{d}{dt}\frac{\partial L}{\partial x_t} \right) dt + o(\delta) = -\delta \int_0^T |x_s(t,s)|^2 dt + o(\delta), \tag{3.13}$$

which means

$$\frac{\partial V(x(\cdot,s))}{\partial s} = \lim_{\delta \to 0} \frac{V(x(\cdot,s+\delta)) - V(x(\cdot,s))}{\delta} = -\int_0^T |x_s(t,s)|^2 dt \leq 0$$

and equality is achieved if and only if $x_s(t,s) = 0$ almost everywhere for $t \in [0,T]$, that is, Euler-Lagrange equation (3.10) is satisfied almost everywhere on the curve $x(\cdot,s)$ □

From this lemma, one can deduce, with some additional work, the existence of solutions to the affine geometric heat flow as the action functional acts as a 'Lyapunov functional' for the AGHF. The Lemma furthermore states that the solutions converge to extremal points of positive-definite action functional (3.9). For more details and discussion of the proof, we refer to our paper [29].

3.2 Motion Planning with AGHF

3.2.1 Riemannian Metric for Motion Planning

Definition 3.6 (Steady state solution)**.** *A solution $x^*(t)$ is a steady state solution of* (3.6) *if it is a stationary point of the AGHF, namely, if*

$$\nabla_{\dot{x}(t)} \left(\dot{x}(t) - F_d \right) + r(t)|_{x(t)=x^*(t)} = 0$$

for all $t \in [0,T]$.

Some intuition about the form of this PDE can be gathered from Lemma 3.1 above. The AGHF is a system of *parabolic PDEs*, and to solve it we need two

boundary conditions

$$x(0,s) = x_i, x(T,s) = x_f \quad \forall s \geq 0 \tag{3.14}$$

and an **initial** condition

$$x(t,0) = v(t), \quad t \in [0,T] \tag{3.15}$$

for some $v(\cdot) \in \mathcal{X}'$. We refer to the initial path chosen $v(t)$ as the **initial sketch**, since it can easily be obtained (most often, a straight line is used) as it does not need to meet the dynamical or holonomic constraints.

The method. We first describe the steps of the method we propose to perform trajectory planning for systems (3.3) and provide below some convergence guarantees. The method can be summarized through the following four steps. We are given $F_d(x) \in \mathbb{R}^n$, $F(x) \in \mathbb{R}^{n\times m}$, $x_i \in \mathbb{R}^n$ and $x_f \in \mathbb{R}^n$.

Step 1: Find a *bounded* $n \times (n-m)$ x-dependent matrix $F_c(x)$, differentiable in x, such that

$$\bar{F}(x) := \Big(F_c(x)|F(x)\Big) \in \mathbb{R}^{n\times n} \tag{3.16}$$

is invertible for all $x \in \mathbb{R}^n$. The matrix $F_c(x)$ can be obtained using, e.g., the Gram-Schmidt procedure.

Step 2: Evaluate

$$G(x) := (\bar{F}(x)^{-1})^\top D \bar{F}(x)^{-1} \tag{3.17}$$

where $D := \mathrm{diag}(\underbrace{\lambda, \cdots, \lambda}_{n-m}, \underbrace{1, \cdots, 1}_{m})$ for some large $\lambda > 0$ (we discuss below what large means in this context).

Step 3: Solve the AGHF (3.6) with boundary conditions (3.14) and initial condition (3.15). Denote the solution by $x(t,s)$;

Step 4: Evaluate

$$u(t) := \Big(0 \quad I_{m\times m}\Big)\, \bar{F}(x(t,s_{\max}))^{-1}(\dot{x}(t,s_{\max}) - F_d(x(t,s_{\max})) \tag{3.18}$$

Output: The control $u(t)$ obtained in (3.18) yields, when integrating (3.3), a **integrated path** $\tilde{x}(t)$, which is our solution to the trajectory planning problem.

We note that F_c does not depend on the drift F_d and that there is no orthogonality requirement between F and F_c either, which gives much freedom in the construction of F_c and hence in many cases one can choose $\bar{F} = (F_c|F)$, and consequently G, to have relatively simple expressions. We will provide examples of the application of the method in Sec. 3.3.

3.2.2 On convergence guarantees for motion planning

The main new ingredients our method proposes are the definition of the Riemannian inner product in Step 2 above, and the definition of the AGHF. Roughly speaking, for this inner product, short paths are admissible paths, and we refer to our earlier work [8] for a longer justification of the use of the inner product defined.

We now show that the control extracted in Step 4 of the method will drive the system arbitrarily close to the desired final state, *provided* that a solution to the motion planning problem exists (which, as we mentioned earlier, is in general an open problem for systems with drift) and the trajectory $v(t)$ with which we initialize the system is well-chosen. We will see in the examples of Sec. 3.3, that an arbitrary choice of initial condition very often yields a convergent solution.

Theorem 3.1. *Consider the system* (3.3) *and let* $x_i, x_f \in \mathbb{R}^n$. *Assume that the motion planning problem from* x_i *to* x_f *is feasible (i.e.* $\mathcal{X}^*$ *is nonempty) and that Assumption A above is met. Then there exists* $C > 0$ *such that for any* $\lambda > 0$, *there exists an open set* $\Omega_\lambda \subseteq \mathcal{X}'$ *(with respect to* $\|\cdot\|_{AC}$*) so that as long as the initial curve* $v \in \Omega_\lambda$, *the integrated path* $\tilde{x}(t)$ *from our algorithm with sufficiently large* $s_{\max}$ *has the property that*

$$|\tilde{x}(T) - x_f| \leq \sqrt{\frac{3TMC}{\lambda}} \exp\left(\frac{3T}{2}(L_2^2 T + L_1^2 C)\right). \tag{3.19}$$

Proof. Since the path planning problem is solvable, there exists $x^*(\cdot) : [0, T] \to \mathbb{R}^n$ such that $x^*(0) = x_i, x^*(T) = x_f$ and $\dot{x}^* = h(x^*) + F(x^*)u^*$ for some $u^*(\cdot) \in [0, T] \to \mathbb{R}^m$. Plug this x^* into (3.8) we have

$$L(x^*(t), \dot{x}^*(t)) = (\dot{x}^*(t) - h(x^*(t)))^\top G_k(x^*(t))(\dot{x}^*(t) - h(x^*(t))) = |u^*(t)|^2.$$

Define

$$V^* = V(x^*) = \int_0^T L(x^*(t), \dot{x}^*(t))dt = \int_0^T |u^*(t)|dt.$$

Notice that V^* is independent of k. Pick $C > V^*$. From Lemma 3.1 we know that $V(x(\cdot, s))$ decreases as long as (3.10) is not satisfied. Hence, there exists a neighborhood Ω_k around x^* so that the HFE solution $x(t, s)$ derived from (3.5) with any $v \in \Omega_k$ as the initial condition and sufficiently large s will have the property that $V(x(\cdot, s)) < C$. Certainly Ω_k contains the connected set of functions x with $V(x) < C$, and by continuity of the functional $V(\cdot)$ we know the latter set is open and hence Ω_k can also be made open. (If as s increases $V(x(\cdot, s))$ converges to some other $V(y) \geq C$ where y is a solution to (3.10), simply shrink Ω_k by redefining it as $\{x \in \Omega_k : V(x) < V(y)\}$). Define the curve $x(t) := x(t, s)$ and controls $u_h(t), u_F(t), u_G(t)$ by

$$\begin{pmatrix} u_h(t) \\ u_F(t) \\ u_G(t) \end{pmatrix} = \bar{F}(x)^\top \dot{x}(t).$$

Plug it into (3.8) and $V(x(t)) < C$ implies

$$\int_0^T k(u_h(t) - 1)^2 + |u_F(t)|^2 + k|u_G(t)|^2 < C.$$

$$\Rightarrow \int_0^T (u_h(t) - 1)^2 dt \leq \frac{C}{k}, \quad \int_0^T |u_F(t)|^2 dt \leq C, \quad \int_0^T |u_G(t)|^2 dt \leq \frac{C}{k}. \tag{3.20}$$

Compared with (3.18), we see that the extracted control is exactly u_F; in other words, the resultant path is given by

$$\tilde{x}(0) = x_i, \quad \dot{\tilde{x}} = h(\tilde{x}) + F(\tilde{x})u_F.$$

Define the error $e(t) := x(t) - \tilde{x}(t)$. Then

$$\begin{aligned} &e(0) = x(0) - \tilde{x}(0) = 0, \\ &\dot{e} = \dot{x} - \dot{\tilde{x}} = (h(x)u_h - h(\tilde{x})) + (F(x) - F(\tilde{x}))u_F + G(x)u_G. \end{aligned}$$

Therefore we have

$$e(t) = \int_0^t (h(x(\tau))(u_h(\tau) - 1) + (h(x(\tau)) - h(\tilde{x}(\tau))) \\ + (F(x(\tau)) - F(\tilde{x}(\tau)))u(\tau)_F + G(x(\tau))u_G(\tau)d\tau$$

Square the norm of $e(t)$ and apply Cauchy-Schwartz inequality,

$$\begin{aligned} |e(t)|^2 \leq & \int_0^t d\tau \int_0^t |(h(x(\tau))(u_h(\tau) - 1) + (h(x(\tau)) - h(\tilde{x}(\tau))) \\ & + (F(x(\tau)) - F(\tilde{x}(\tau)))u(\tau)_F + G(x(\tau))u_G(\tau)|^2 d\tau \\ \leq & t \int_0^t (|(h(x(\tau))(u_h(\tau) - 1)| + |h(x(\tau)) - h(\tilde{x}(\tau))| \\ & + |(F(x(\tau)) - F(\tilde{x}(\tau)))u(\tau)_F| + |G(x(\tau))u_G(\tau)|)^2 d\tau \end{aligned}$$

Use power mean inequality, the square of the sum of 4 terms inside the integral is no larger than 4 times the sum of the square of each individual,

$$|e(t)|^2 \leq 4t \int_0^t |(h(x(\tau))(u_h(\tau) - 1)|^2 + |h(x(\tau)) - h(\tilde{x}(\tau))|^2 \\ + |(F(x(\tau)) - F(\tilde{x}(\tau)))u_F(\tau)|^2 + |G(x(\tau))u_G(\tau)|^2 d\tau$$

Recall that h, F are globally Lipschitz, h is globally bounded and G is normalized in the way that $\|G(x)\| \equiv 1$ for all $x \in \mathbb{R}^n$, we conclude that

$$\begin{aligned} |e(t)|^2 \leq & 4t \int_0^t M^2(u_h(\tau) - 1)^2 + L_2^2|e(\tau)|^2 + L_1^2|e(\tau)|^2|u_F(\tau)|^2 + |u_G(\tau)|^2 d\tau \\ = & 4t \int_0^t \left(M^2(u_h(\tau) - 1)^2 + |u_G(\tau)|^2 d\right) \tau + \\ & 4t \int_0^t (L_2^2 + L_1^2|u_F(\tau)|^2)|e(\tau)|^2 d\tau \end{aligned}$$

Next, by Grönwall inequality and the fact that $4t \int_0^t M^2(u_h(\tau) - 1)^2 + |u_G(\tau)|^2 d\tau$ is a non-decreasing function of t,

$$|e(t)|^2 \leq \\ \left(4t \int_0^t M^2(u_h(\tau) - 1)^2 + |u_G(\tau)|^2 d\tau\right) \exp\left(4t \int_0^t (L_2^2 + L_1^2|u_F(\tau)|^2) d\tau\right)$$

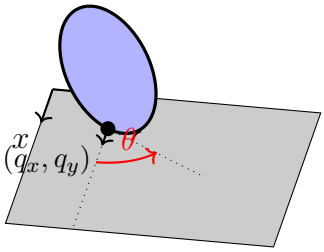


Figure 3.2: The unicycly is a 3 dof system with configuration variables (q_x, q_y), describing the position of the center of the wheel, and θ describing the orientation of the wheel with respect to the x-axis.

Eventually, substituting in the inequalities from (3.20), we conclude that

$$|e(t)|^2 \leq \frac{4t(M^2+1)C}{k} \exp\big(4t(L_2^2 t + L_1^2 C)\big),$$

Thus $|\tilde{x}(T) - x_f| = |e(T)| \leq 2\sqrt{\frac{T(M^2+1)C}{k}} \exp\left(2T(L_2^2 T + L_1^2 C)\right)$. □

This result of Theorem. 3.1 quantifies the trade-off between the size of the parameter λ, and the quality of the control obtained, where the quality is measured according to how close to the desired final state the control drives the system. We see that as $\lambda \to \infty$, the control drives the system to x_f. For more details and discussions for proof of the theorem, we refer to [29].

3.3 Application: Dubin Car and Dynamical Unicycle

We illustrate our method on three canonical motion planning problems, showcasing how it handles the different issues that can arise. The basic system we consider is a unicycle rolling on the plane without slipping, as depicted in Fig. 3.2. The kinematics is given by the differential equations

$$\underbrace{\begin{pmatrix} \dot{q}_x \\ \dot{q}_y \\ \dot{\theta} \end{pmatrix}}_{\dot{x}} = \underbrace{\begin{pmatrix} \cos\theta \\ \sin\theta \\ 0 \end{pmatrix}}_{f_1} u_1 + \underbrace{\begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}}_{f_2} u_2. \tag{3.21}$$

We refer to our earlier work [7] for a description of how to perform basic motion planning tasks on this system.

3.3.1 Unicycle with constant linear velocity or Dubins car

We first consider the model of a planar unicycle with unit *constant linear velocity*. In this case, we have $u_1 \equiv 1$ and hence (3.21) becomes $\dot{x} = f_1 + f_2 u$, where f_1 is now the drift vector field. Hence, even when $u = 0$, $\dot{x} \neq 0$. The control u only allows to steer the unicycle.

Following our method, we pick $F_c = \begin{pmatrix} e_1 & e_2 \end{pmatrix}$ so that $\bar{F}$ is the identity matrix. Hence according to Step 2 we have $G = \mathrm{diag}(\lambda, \lambda, 1)$. The corresponding Lagrangian (3.8) is

$$L(x, \dot{x}) = \lambda(\dot{q}_x - \cos(\theta))^2 + \lambda(\dot{q}_y - \sin(\theta))^2 + \dot{\theta}^2$$

The boundary conditions are set to

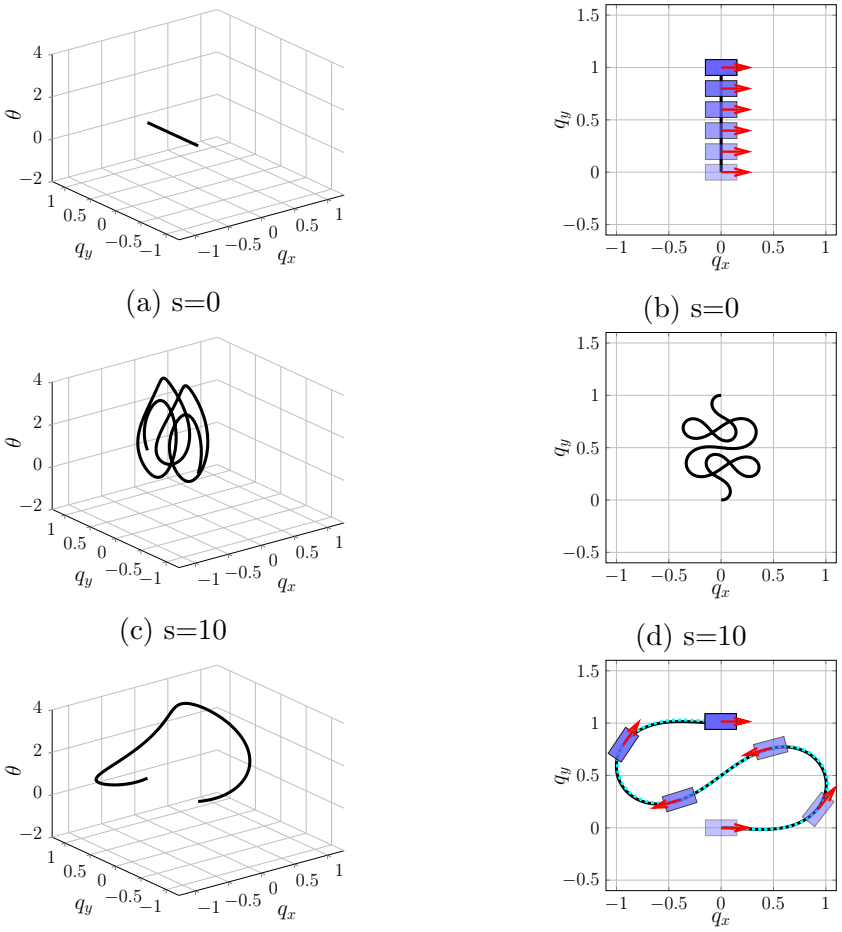
$$x_i = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^\top \text{ and } x_f = \begin{pmatrix} 0 & 1 & 0 \end{pmatrix}^\top,$$

which are the boundary conditions of the so-called "parallel parking" problem. We obtain the results shown in Fig. 3.3.

We see that the initial sketch of straight line $x(t, 0) = v(t) = (0, t, 0)^\top$ in Fig. 3.3a and Fig. 3.3b cannot be followed by the unicycle as such a path violates the non-slip constraints and does not follow the drift dynamics. Fig. 3.3c and Fig. 3.3d show the curve $x(t, s)$ with $s = 10$. Fig. 3.3e and Fig. 3.3f shows $x(t, s_{\max})$ with $s_{\max} = 500$. The integrated trajectory (cyan dotted line) generated by using the extracted control (as in Step 4) is very close to $x(t, s_{\max})$ and drives the unicycle close to x_f with very high precision.

3.3.2 Dynamic unicycle

We now consider the unicycle with inertia; the acceleration of the unicycle is proportional to the applied torque following Newton's second law. To model this system in the form of (3.3), we add two states to the unicycle configuration: u_1 and u_2, representing the linear and angular velocity. Acting on the



(a) s=0 (b) s=0

(c) s=10 (d) s=10

(e) s=500 (f) s=500

Figure 3.3: Unicycle trajectory with $\lambda = 1000, T = 5$. Left column: paths in 3D state space; right column: corresponding (q_x, q_y)-plane projected view. The unicycle follows the black solid curve and moves from the position with the lightest blue color to positions with darker blue colors gradually, with its orientation and magnitude of linear velocity at each snapshot indicated by the red arrow.

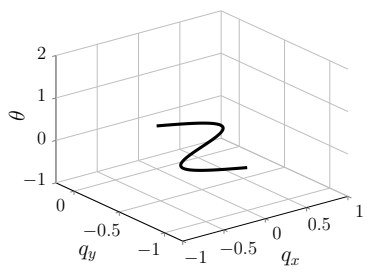
accelerations $\dot{u}_1, \dot{u}_2$, the controls v_1, v_2 are a force and a torque, respectively.

$$\underbrace{\begin{pmatrix} \dot{q}_x \\ \dot{q}_y \\ \dot{\theta} \\ \dot{u}_1 \\ \dot{u}_2 \end{pmatrix}}_{\dot{x}} = \underbrace{\begin{pmatrix} u_1 \cos\theta \\ u_1 \sin\theta \\ u_2 \\ 0 \\ 0 \end{pmatrix}}_{F_d} + \underbrace{\begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}}_{F} \underbrace{\begin{pmatrix} v_1 \\ v_2 \end{pmatrix}}_{v} \tag{3.22}$$

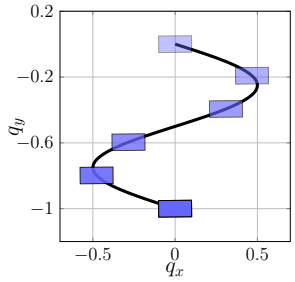
Similar to the previous case, we can take $\bar{F}$ to be the identity matrix and $G = \mathrm{diag}(\lambda, \lambda, \lambda, 1, 1)$. Consequently,

$$L(x, \dot{x}) = \lambda \left((\dot{q}_x - u_1 \cos(\theta))^2 + (\dot{q}_y - u_1 \sin(\theta))^2 + (\dot{\theta} - u_2)^2 \right) + \dot{u}_1^2 + \dot{u}_2^2 \tag{3.23}$$

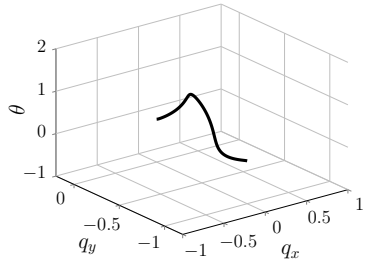
We set the boundary condition to $x_i = (0, 0, 0, 0, 0)^\top$ and $x_f = (0, -1, 0, 0, 0)^\top$. The boundary values for u_1 and u_2 are 0, meaning the unicycle starts and ends with 0 velocities. We use a partial sinusoid $v(t) = (\sin(2\pi t), -t, 0, 0, 0)$ as the initial sketch $x(t, 0)$, shown in Fig. 3.4a and Fig. 3.4b. Following the remaining steps of the algorithm, the results are shown in Fig. 3.4. The unicycle cannot follow the initial curve as seen in Fig. 3.4b. The AGHF yields the curve $x(t, s_{\max})$ shown in Fig. 3.4f (black solid line). Extracting the control, we obtain a trajectory (cyan dotted line) that is almost identical.



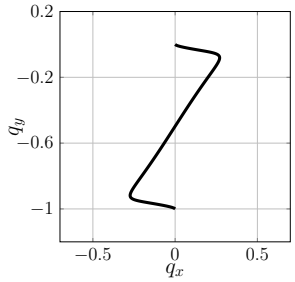
(a) s=0



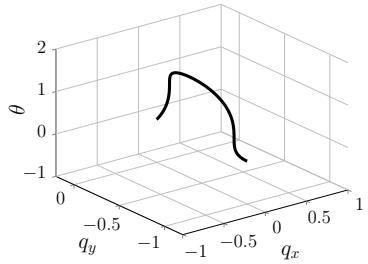
(b) s=0



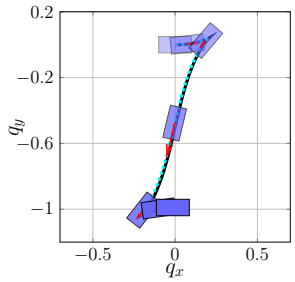
(c) s=0.0005



(d) s=0.0005



(e) s=0.01



(f) s=0.01

Figure 3.4: Dynamic unicycle trajectory with $\lambda = 50000, T = 1$. Only (q_x, q_y, θ)-space (left column) and (q_x, q_y)-projected view (right column) are shown here.

CHAPTER 4

STATE INEQUALITY CONSTRAINTS, INTEGRALS OF MOTION AND BOUNDARY CONDITION: APPLICATIONS TO ROBOT GYMNASTICS

The goal of this chapter is to expand the formulation of the AGHF to allow the inclusion of a broad range of robotic motion planning settings. In particular, we aim to solve robot gymnastics motion planning and extend the method to include state inequality constraints and conserved quantity of motions. State inequality constraints are ubiquitous in robotics systems, and makes it more challenging to find feasible motions. For example, finding mid-air motions for humanoid robots are difficult since the motion of limbs are limited by joint angles. Implementing state equality constraints is, from a geometric perspective, relatively easy. Indeed, such constraints are reflected in the choice of control vector fields which are tangent to the isolevel sets defined by the equality constraints, as demonstrated in Sec. 2.2.1. Equality constraints *reduce* the dimension of the state space of the system. The case of inequality constraints, which do not reduce the dimension of the state space, is mathematically more difficult to handle. On the other hand, conserved quantities such as momentum, exist in robot systems where no external wrench is applied. The conserved quantity is determined by initial or final state of the motion, and acts on the system as an additional constraint. These BC are often an integral part of a motion planning problem and difficult to determine. We will rely on the AGHF algorithm to find the motions with proper BC which satisfy the conserved quantity constraint.

In-air robot motion planning is a growing area in robotics motion planning, as the hardware of robots has become more and more reliable in the past decade, making the vision of agile, diving and jumping robots closer to reality. Such robots include, among others, the MIT Cheetah [1], the bipedal robot Cassie robot made by Agility Robotics and Salto robot [2]. In the range

of dynamic motions that one wants these robots to perform, the motion in mid-air has been a difficult yet crucial part. Indeed, the robot in mid-air can only reorient itself by internal reactions and, in particular, the mid-air dynamics has non-trivial conserved quantities, such as the angular momentum, which constrain the dynamics in rather non-intuitive ways. The joint angle and velocity limitations further constrained the robot's motion in mid-air. On the other hand, the performance of the mid-air motion is the key to a safe take off and safe landing of the robot, and thus fundamental to the overall endeavor.

Due to its importance, there has been a fair amount of work dealing with various aspects of mid-air motion. The falling cat problem is a classical one [30], explaining how a cat reorients itself mid-air while conserving its angular momentum. In the same vein, inspired by free falling of lizards and geckos, Jusufi has shown that a large active tail can function as effective control appendages, therefore enhancing the posture maneuver during falling [31]. Utilizing the active tail, a PD controller [32] and a sliding mode controller [33] is implemented to control the 2D orientation of tailed robot in flight phase. Human somersault and diving is another example of reorienting in mid-air. In early 90's, there was already a successful implementation of controller for biped somersault [34] [35], which regulated the angular velocity by manipulating the length of the tucked leg. An optimization -based mid-air planning method was also proposed in [36] for non-zero initial angular momentum. Space robotics is another domain of application of floating robot motion planning, and conditions under which a control algorithm that can be applied to a fixed base robot can be applied to a free-floating robot were derived in [37, 38].

The robot's motion in mid-air is constrained by state constraints, such as joint angle limit and velocity limit, which are usually determined by the robot's mechanical and electrical structure. In some applications, the states are constrained to have a more stable motion. The torso pitch of MIT Cheetah is bounded to have a more stable bounding motion [39]. In humanoid robot, the centroidal angular momentum is constrained to be small for walking motions. The state constraints can be encoded as obstacle avoidance problem as in [7] and [8], where the state inequality constraints are enforced by obstacle's barrier functions.

As already mentioned earlier, a salient issue in mid-air motion planning is

the conservation of momentum. In [40], the conserved angular momentum is treated as drift term and the nonholonomic constraint derived from angular momentum conservation is encoded in the system dynamics. In some recent works, similar reduced dynamics are used for free-floating robot motion planning relying on using a nonlinear optimizer [36, 41, 42]. With the system reduced by conservation of angular momentum, the ability to directly control joint torques is lost. Instead, joint velocities can be used as inputs. To generate a motion using the joint torques, full dynamics is used in [43] and the motion is solved by direct method of optimization, the hidden conservation of angular momentum is encoded with the system's full dynamics.

Path planning with homotopy classes is utilized in [44], where a graph-search based method is proposed for finding the optimal path with constraints on homotopy classes. We relies on the AGHF motion planning method proposed in Chapter. 3 to solve for mid-air motions. We consider torque control of the joints, gravity is taken into account, and our method is able to naturally find the initial impulse needed to perform a motion by allowing indefinite BC; this initial impulse is the one imparting the robot with an angular momentum that will be constant when airborne. In this chapter, we extend the AGHF motion planning algorithm to include both state equality and inequality constraints, as well as indefinite BC. The state constraints are encoded via *switch functions*, which determines the contribution of the state constraints to the actuated curve length..

The remainder of the chapter is organized as follows. In Section 4.1, the formulation for state equality and inequality constraints are proposed. In Section 4.2, the free boundary conditions are given and discussed. Section 4.3 summarizes the key steps of the extended AGHF motion planning algorithm. Section 4.4 implements the algorithm on a robot gymnastics motion planning, and simulation results are listed.

4.1 State Inequality Constraints

Notation: We use notation $O_{k,p}$ and $O_k$to denote a $k \times p$ matrix and $k \times k$ matrix with zero elements, and I_k to denote $k \times k$ identity matrix.

The existence of state *inequality* constraints, which are difficult to handle in motion planning, is very common in robotic systems. For example, the joint

angles are limited for legged robots which result in a smaller configuration space for motion planning. In our previous work [10], the state constraints are added as obstacles via barrier functions, which are included in the Riemannian metric. The metric tensor grows large when evaluated near obstacles, which results in the actuated length of curves passing in the vicinity of obstacles to grow as well. While this approach allowed us to derive provably correct methods, its numerical implementation can run into issues, since a large metric tensor both slows down the computations (at a fixed precision level).

To address this issue, we propose in this section a new approach to handle both state *equality* and *inequality* constraints. A state constraint can be formulated as a scalar function $h(x) = 0$ for equality constraints and $h(x) \leq 0$ for inequality constraints. This formulation can implement constraints on both q and $\dot{q}$ since $x := [q^\top, \dot{q}^\top]^\top$. When more than one constraint is present, the same method as the one described here is to be applied to each constraint individually.

Following our ansatz, we seek to have the constraints reflected into the actuated length of the curve. To this end, first we add one state, denoted by $x_h \in \mathbb{R}$, for each scalar constraint. The resulting *augmented state* of the system thus becomes: $\hat{x} := [x^\top, x_h]$, recalling that original state is $x \in \mathbb{R}^n$ and the original control is $u \in \mathbb{R}^m$. The new state x_h will keep track of the *accumulated error* between $h(x)$ and zero:

Definition 4.1 (Accumulated error for state constraint)**.** *The accumulated error for the state constraint function $h(x)$ is defined as*

$$x_h(t) := \int_0^t h(x(s))S_h(x(s))ds. \tag{4.1}$$

It is easy to see that the accumulated error thus obeys the following *error dynamics*

$$\dot{x}_h := h(x(t))S_h(x) \tag{4.2}$$

In the definition above, $S_h(x)$ is a scalar *switch function* for the state constraint which can be constructed by the constraint function $h(x)$.

Definition 4.2 (Switch function)**.** *For the inequality constraint $h(x) \leq 0$, the switch function is a binary-valued function, $S_h(x) \in \{0, 1\}$, defined by the constraint function $h(x)$:*

$$S_h(x) = H(h(x)) \tag{4.3}$$

in which $H(h(x))$ is a Heaviside unit step function at $h(x)$. For equality constraint $h(x) = 0$, the switch function is independent of $h(x)$ and has a constant value:

$$S_h(x) \equiv 1 \tag{4.4}$$

The switch functions should be defined for each scalar constraint respectively, and they indicate if the constraints should be enforced at given states. In this chapter, a smooth approximation of Heaviside unit step function is used. More specifically, a logistic approximation is used:

$$H(c) := \frac{1}{1 + e^{-k_s c}} \tag{4.5}$$

which approximates a unit step at $c = 0$, namely, $H(c) \approx 0$ if $c < 0$ and $H(c) \approx 1$ if $c > 0$. The constant k_s controls the accuracy of the approximation. We use a smooth approximation of the step function here to guarantee that the derivative of the step function is well defined, which will be used in the constrained AGHF equation. It is now easy to see that, thanks to the switch function (4.3), x_h is approximately zero if the constraint is satisfied during $t = 0$ to $t = T$.

The augmented system dynamics is now:

$$\dot{\hat{x}} = \hat{F}_d(\hat{x}) + \hat{F}(\hat{x}) \begin{bmatrix} u \\ h(x)S(x) \end{bmatrix} \tag{4.6}$$

where

$$\hat{F}_d(\hat{x}) = \begin{bmatrix} F_d(x) \\ 0 \end{bmatrix}$$
$$\hat{F}(\hat{x}) = \begin{bmatrix} F(x) & O_{n,1} \\ O_{1,n} & 1 \end{bmatrix}$$

in which $\hat{F}_d$ is the augmented drift.

The Riemannian metric should be enriched as well to increase the length of paths violating the constraint. We do so by introducing

Definition 4.3 (Augmented Riemannian metric)**.**

$$\hat{G}(\hat{x}) := \begin{bmatrix} G(x) & O_{n,1} \\ O_{1,n} & \lambda_h \end{bmatrix} \tag{4.7}$$

where $G(x)$ is the Riemannian metric defined for the original states x in (3.17) *and λ_h is a constant.*

We can now derive the so-called *constrained AGHF*, which is a partial differential equation minimizing the length as measured by the above-introduced Riemannian metric.

Lemma 4.1 (Constrained AGHF)**.** *Consider the curve length functional*

$$\begin{aligned}\mathcal{L}_G(x,\dot{x}) &:= \int_0^T \left((\dot{\hat{x}} - \hat{F}_d(\hat{x}))^\top \hat{G}(\hat{x})(\dot{\hat{x}} - \hat{F}_d(\hat{x}))\right)^{1/2} dt \\ &= \int_0^T \left((\dot{x} - F_d(x))^\top G(x)(\dot{x} - F_d(x)) + \lambda_h h(x)^2 S(x)\right)^{1/2} dt \end{aligned} \tag{4.8}$$

associated with the Riemannian metric (4.7)*. Let $x(t,s)$ be a solution of the system of partial differential equations*

$$\frac{\partial x(t,s)}{\partial s} = \nabla_{\dot{x}(t,s)} \left(\dot{x}(t,s) - F_d\right) + r(t,s) - \lambda_h \frac{\partial (h(x)^2 S(x))}{\partial x}. \tag{4.9}$$

Then $x(t,s)$ converges to a stationary point of $\mathcal{L}_G$ as $s \to \infty$.

Proof. The proof follows the lines of the proof of Lemma. 3.1 with $L = L_G(x,\dot{x})$. □

The new actuated length (4.8) has the additional term $\lambda_h h(x)^2 S(x)$ when compared to the original actuated length (3.5). Due to the form of the metric tensor $\hat{G}$, specifically the fact that it is block-diagonal with one block containing λ_h, minimizing the length of a curve will result in minimizing the violation of the state-constraints. For an equality constraint, $S(x) = 1$, the actuated length is penalized by λ_h when $h(x)$ is not close to zero, driving $h(x) \to 0$. For an inequality constraint, the actuated length is penalized by λ_h when $h(x) > 0$ and $S(x) = H(h(x)) \approx 1$, driving $h(x) \to 0$. However, this term has no effect on the actuated length if $h(x) \leq 0$ since $S(x) = H(h(x)) \approx 0$. As a result, the curve with minimum actuated length, solved by (4.9) yields a trajectory that the system can follow while obeying the state constraints.

It should be noted that the constrained AGHF equation (4.9) is derived from the original AGHF (3.6) by replacing x with $\hat{x}$ and substituting $\dot{x}_h := h(x(t))S(x)$. Therefore, the constrained AGHF only has the original state x and has dimension n instead of $n+1$.

4.2 Boundary Conditions for the AGHF

For motion planning problems with conserved quantities, obtaining the *boundary conditions* is one of the main issues. Indeed, the boundary conditions assign specific values to the conserved quantities, and these cannot be altered during the motion by the controls. In the case of robot gymnastics motions, the conserved quantity of interest is the angular momentum, since no external force is applied to the robot in mid-air. A physical interpretation is that when a robot jumps, its angular momentum is constant when in the air, and his initial kick against the ground is thus of high importance, as it sets the value of this momentum. Thus, if one wants to rotate, say three times while in the air, one needs to start with a higher angular momentum than if one needs to rotate only once.

A major upside of our approach is that it can find these important parameters, which are here the boundary conditions necessary for the motion to be feasible, in a natural way. We explain the procedure to achieve this below.

We call a state variable which we do not want to specify at either the beginning or end of the motion *free*. An example of a free variable is the initial vertical velocity of a jumping robot, which is related to the strength of the initial push against the ground, and is not controllable once in the air, or the initial angular velocity as mentioned above. A fixed variable is a variable for which there is a specified value. For example, the starting position of a diver.

We use the following boundary conditions for the AGHF depending on whether the variables are fixed or free:

Definition 4.4 (AGHF boundary condition (BC))**.**

$$x_i(0) \textit{ fixed} : x_i(0) = x_{init,i} \tag{4.10a}$$

$$x_i(0) \textit{ free} : \dot{x}_i(0) = F_{d,i}(x(0)) \tag{4.10b}$$

The same holds for the final conditions (at time T) with x_{fin}. The index i indicates the i-th element of the vector.

Hence,

- If the state x_i has fixed value at boundary, BC (4.10a) is applied

- If state x_i is set to be free at boundary, BC (4.10b) is applied.

The justification for the form of these boundary conditions can be obtained following the Lagrangian approach developed in [10]. More specifically, the free boundary conditions can be obtained by applying the transversality conditions on the Lagrangian.

Use of Boundary Conditions: an example. We now illustrate the use of these boundary conditions on what is perhaps the simple example for which they matter: a point mass moving up and down; in the free motion phase, only gravity acts on the mass.

For a point mass with only gravity and no control—free falling, as shown in Figure. 4.1, the state space can be defined as $x = [y, \dot{y}]$ where y is the height of the point mass. We seek for the motion of the point mass which starts from ground with duration T. This corresponds to the boundary conditions

$$y(0) = 0.$$

We explore the effect of setting the remaining boundary conditions $\dot{y}(0)$, $y(T)$ and $\dot{y}(T)$:

Case 1: $\dot{y}(0)$, $y(T)$ and $\dot{y}(T)$ are fixed. This corresponds to fixing initial vertical velocity, final height and velocity. Because the in-air motion conserves energy, the initial boundary conditions can be thought of as *fixing* the value of the said energy. The final boundary conditions have to meet some constraints insuring that no energy was dissipated during the motion. Thus, they cannot be chosen arbitrarily, but have to satisfy

$$\dot{y}(T) = \dot{y}(0) + gT, \; y(T) = \dot{y}(0)T + \frac{1}{2}gT^2.$$

If the BCs encode different energy for the mass, of course no feasible, hence conservative, motion exists that satisfy said boundary conditions. The AGHF in this case will not converge.

This simple example illustrates that when systems have dynamical drift (here, due to gravity), not all boundary conditions can be fixed in general. This is one of the main reasons behind the fact that motion planning for systems with drift is far more challenging theoretically than that for driftless systems.

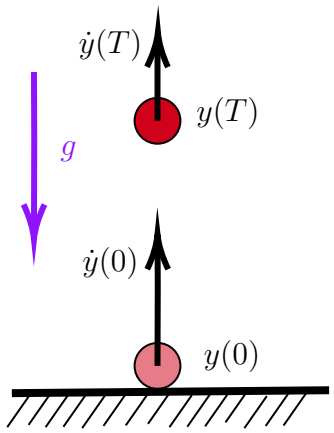


Figure 4.1: 1D Point mass

Case 2: $\dot{y}(0)$, $\dot{y}(T)$ are free and $y(T)$ is fixed. We set a desired final height $y(T)$. The initial and final velocities are free. The planning algorithm finds a motion with proper initial velocity such that the point mass will reach the specified final height $y(T)$ at $t = T$ with a proper final velocity that conserves the total energy.

Case 3: $y(T)$, $\dot{y}(T)$ are free and $\dot{y}(0)$ is fixed. We set an initial velocity and let the final height and velocity free: the planning algorithm is able to find a motion with proper final height $y(T)$ and velocity $\dot{y}(T)$ that the point mass can reach with the specified initial velocity $\dot{y}(0)$.

For these three cases, the motion planning algorithm was reduced to solving the physics of the problem (no controls present, save for the initial kick imparting a nonzero velocity at $t = 0$.) The same type of reasoning of course applies to systems with controls.

4.3 Planning Algorithm Summary

The steps of the proposed motion planning algorithm with state constraints can be summarized as follow.

Step 1: Find the full rank matrix $\bar{F}(x)$ using $F(x)$ based on (3.16).

Step 2: Evaluate the augmented Riemannian metric $\hat{G}(x)$ for a large λ and λ_h.

Step 3: Solve the **constrained AGHF** (3.6) with **boundary conditions** (4.10a) **or** (4.10b) , initial condition (3.15) and large enough

s_{max} to obtain the solution $x^*(t)$. This step is the main extension of the algorithm.

Step 4: Extract control u from $x^*(t)$ using (3.18)

Step 5: Integrate the dynamics (3.3) with extracted control u and initial value x_{init} to obtain the integrated path $\tilde{x}(t)$, which is the planned motion.

4.4 AGHF for Robot Gymnastics

We now focus on planning the motion of kinematic chain robots in 2D space. The defining characteristic of the motion planning tasks we consider here are the fact that the motion takes place 'in-air', and is thus subject to having integrals of motion (angular momentum and linear momentum).

Definition 4.5 (Series open kinematic chain)**.** *A series open kinematic chain is an assembly of rigid bodies connected in series with open endpoints.*

We focus on robots with a series open kinematic chain structure, where the links are connected via revolute joints, as illustrated in Fig. 4.2. These mechanisms can be used to model human and animal bodies. The kinematic chain starts with the (base) link 0, and has k additional links for a total of k joints and $k+1$ links. The configuration space $\bar{Q}$ is of dimension $k+3$, which makes the system under-actuated. In fact, it is easy to see that the configuration is completely described by the position and orientation of *any* one link and the value of the joint angles, and thus $\bar{Q} \in \mathbb{R}^2 \times \mathbb{T}^{k+1}$, where we recall that $\mathbb{T}^k$ is the cross-product of k circles. We choose the following generalized coordinates: $\bar{q} = [x_0, y_0, \theta_0, q_1, q_2, ...q_k]^\top \in \bar{Q}$ where $[x_0, y_0, \theta_0]^\top$ are the position and orientation of the base link represented in ground inertial frame and q_i is the joint angle of the joint associated with link i. Each link i has link length l_i, link mass m_i and a rotational inertia I_i with respect to its own center of mass (CoM).

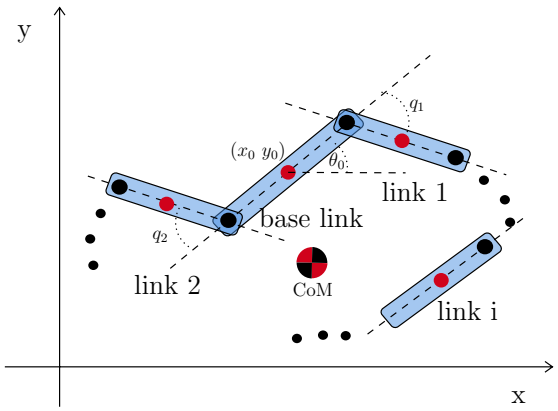


Figure 4.2: Depiction of $k+1$ links floating kinematic chain, with base link defined as link 0.

4.4.1 Full dynamics

It is well known [45] that the equations of motion for the dynamics can be derived using a Lagrangian approach as

$$\bar{D}(\bar{q})\ddot{\bar{q}} + \bar{C}(\bar{q}, \dot{\bar{q}})\dot{\bar{q}} + \bar{G}(\bar{q}) = \bar{U} \tag{4.11}$$

where $\bar{D}$, $\bar{C}$ and $\bar{G}$ are the inertia, Coriolis and gravity matrix, respectively. The vector $\bar{U} := [0, 0, 0, u_1, u_2, ...u_k]^\top$ is the vector of generalized forces corresponding to the generalized coordinates $\bar{q}$. The first three terms are zero, reflecting the fact that the base link's position and orientation are not explicitly actuated, and u_i is the actuating joint torque for the joint associated with link i. These are the inputs of the system.

4.4.2 Reduced dynamics

Since no external wrenches act on the system when airborne, besides gravity in the y direction, both the total angular and translational momentum in the x direction are conserved in this phase of the motion. Furthermore, the trajectory of the CoM for the system is entirely determined by its initial position and velocity (or by the *initial kick* against the ground performed by the robot in order to get airborne). To this end, we introduce the reduced coordinates

$$q = [\theta_0, q_1, q_2, ...q_k]^\top \in \mathbb{T}^{k+1}.$$

and the control

$$U = [u_1, u_2, ...u_k]^\top \in \mathbb{R}^k$$

which are the joint torques.

Compared with the full dynamics in (4.11), the reduced dynamics eliminates the CoM position and its actuation force from the system:

Proposition 4.1. *The reduced dynamics of the floating n-link open chain is given by*

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \begin{bmatrix} 0 \\ U \end{bmatrix} \tag{4.12}$$

where

$$\begin{aligned} D(q) &= D_q(q) - D_2(q)D_{xy}^{-1}(q)D_1(q) \\ C(q,\dot{q}) &= C_q(q,\dot{q}) - D_2(q)D_{xy}^{-1}(q)C_{xy}(q,\dot{q}) \\ G(q) &= G_q(q) - D_2(q)D_{xy}^{-1}(q)G_{xy}(q) = 0. \end{aligned}$$

Proof. To find the matrix D, C and G, we rewrite the LHS of system (4.11) as

$$\begin{bmatrix} D_{xy}(q) & D_1(q) \\ D_2(q) & D_q(q) \end{bmatrix} \begin{bmatrix} \ddot{x}_0 \\ \ddot{y}_0 \\ \ddot{q} \end{bmatrix} + \begin{bmatrix} C_{xy}(q,\dot{q}) \\ C_q(q,\dot{q}) \end{bmatrix} \dot{q} + \begin{bmatrix} G_{xy}(q) \\ G_q(q) \end{bmatrix} \tag{4.13}$$

and RHS as $[0,0,0,U]^\top$, where $U = [u_1, u_2, ...u_k]^\top \in \mathbb{R}^n$ are the inputs for reduced dynamics. The original $\bar{D}$, $\bar{C}$ and $\bar{G}$ matrices are decomposed into smaller matrices: $\bar{D}$ is decomposed into 2×2 and $(k+1) \times (k+1)$ matrices D_{xy} and D_q at diagonal location, matrices D_1 and D_2 at off-diagonal locations. The system dynamics is independent of the base link position and velocity. Consequently, $\dot{x}_0$, $\dot{y}_0$ do not enter the equations of motion and the first two columns of $\bar{C}$ are 0. Removing these columns, we get C_{xy} of dimension $2 \times (k+1)$, and C_q of dimension $(k+1) \times (k+1)$, from $\bar{C}(q)$. In addition, the gravity matrix $\bar{G}$ is a column vector which can be decomposed into two smaller vectors G_{xy} and G_q of lengths 2 and $k+1$ respectively.

Eliminating the first two rows of (4.13) and expressing $[\ddot{x}_0, \ddot{y}_0]^\top$ in terms of the other variables, we obtain

$$\begin{bmatrix} \ddot{x}_0 \\ \ddot{y}_0 \end{bmatrix} = -D_{xy}^{-1}(q)(D_1(q)\ddot{q} + C_{xy}(q,\dot{q})\dot{q} + G_{xy}(q)) \tag{4.14}$$

Similarly, we can take out the last $k+1$ rows of (4.13) and rearrange:

$$D_2(q)\begin{bmatrix}\ddot{x}_0\\ \ddot{y}_0\end{bmatrix} + D_q(q)\ddot{q} + C_q(q,\dot{q})\dot{q} + G_q(q) = \begin{bmatrix}0\\ U\end{bmatrix} \tag{4.15}$$

Substituting (4.14) into (4.15) and reordering, We can have the matrices D, C and G for the desired reduced system (4.12) with reduced generalized coordinate q:

$$\begin{aligned} D(q) &= D_q(q) - D_2(q)D_{xy}^{-1}(q)D_1(q) \\ C(q,\dot{q}) &= C_q(q,\dot{q}) - D_2(q)D_{xy}^{-1}(q)C_{xy}(q,\dot{q}) \\ G(q) &= G_q(q) - D_2(q)D_{xy}^{-1}(q)G_{xy}(q) = 0. \end{aligned}$$

□

Note that the inverse D_{xy}^{-1} exists since the inertia matrix $\bar{D}$ is symmetric and invertible. The new inertia matrix D is still symmetric and invertible. The reduced gravity matrix G is zero since gravity affects the reduced dynamics only through the motion of the CoM, which the above construction isolated from the reduced dynamics. Furthermore, even though the notation $D(q)$ and $C(q)$ suggests that D and C are functions of the reduced states q, they are actually independent of the first state θ_0 of q because the inertia and Coriolis force of the system are independent of the base's orientation.

The remainder of the application example will focus on the trajectory planning of this reduced system (4.12). The CoM trajectory can be easily calculated once the initial condition of CoM is known. The only parameter we need to know from CoM trajectory is the flight duration, which decides the time span of the planning problem.

4.4.3 State Space Model

The equation of motion (4.12) has to be converted to state space representation as (3.3). Define the state by

$$x := [\theta_0, q_1, q_2, ...q_k, \dot{\theta}_0, \dot{q}_1, \dot{q}_2, ...\dot{q}_k]^\top = [q^\top, \dot{q}^\top]^\top.$$

so the dimension of state is $n = 2k + 1$. The system (4.12) can be expressed as a control affine state space model with drift:

$$\dot{x} = F_d(x) + F(x)u \tag{4.16}$$

where

$$u = \begin{bmatrix} 0 \\ U \end{bmatrix}$$
$$F_d(x) = \begin{bmatrix} O_{k+1} & I_{k+1} \\ O_{k+1} & -D^{-1}(q)C(q,\dot{q}) \end{bmatrix} x$$
$$F(x) = \begin{bmatrix} O_{k+1} \\ D^{-1}(q) \end{bmatrix}$$

where the first element of the control u is zero, due to the fact that the orientation of the base link is not actuated. The remaining part of u is U which contains the joint torques. The vector $F_d(x)$ represents the drift dynamics and $F(x)$ represents the admissible control directions. Physically, the first $k + 1$ rows of F_d relate the derivatives of the angular positions with their velocities. The last $k + 1$ rows of F_d is the drift caused by the Coriolis force. The first $k + 1$ rows of F being zero reflects the fact that the control U can only be directly applied to the accelerations, i.e. we perform torque control. The system is *under-actuated*, since there are $2k + 2$ states and k inputs.

4.4.4 Construction of a Riemannian Metric

For our purpose, according to (3.16), we can take $F_c := [I_{k+1}, O_{k+1}]^\top$ so that $span\{F_c\}$ is orthogonal to $span\{F\}$, and $\bar{F} = diag[I_{k+1}, D^{-1}(q)]$ is full rank for all x because the inertia matrix D is invertible.

We then define the Riemannian metric tensor G based on (3.17), with

$$D := \text{diag}(\underbrace{\lambda, \cdots, \lambda}_{k+2}, \underbrace{1, \cdots, 1}_{k})$$

for some large constant $\lambda > 0$.

We recall that the parameter λ can be thought of as a penalty on the

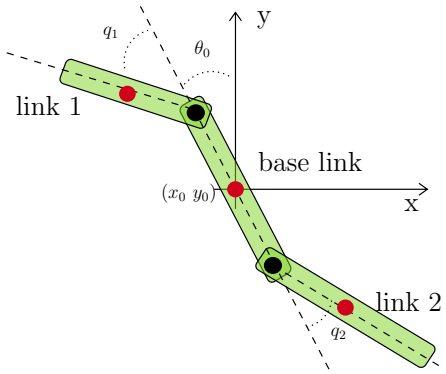


Figure 4.3: Diver robot.

infinitesimal directions $F_c(x)$. Using this metric G, we can measure the actuated length (3.5) of a curve $x(t)$. But note that since λ penalizes $F_c(x)$, a curve of minimal actuated length will use these directions only minimally, and this will yield a motion trajectory that the robot system can follow and the angular momentum is conserved, with very high precision (quantitative relations between λ and precision are derived in Chapter. 3), using some controls u.

With state constraints formulated according to Sec. 4.1, the augmented Riemannian metric (4.7) can be construct with G and a large λ_h. The constraints can be formulated individually with individual values for λ_h. In the next section, we demonstrate examples of different type of constraints.

4.4.5 Implementation and simulation results

Diver Robot We consider a planar diver robot with three links with revolute joints, as illustrated in Fig. 4.3. The middle link, which we consider to be the base link, can be thought of as corresponding to a human torso. The position and orientation of the base link in an inertial frame are denoted by $[x_0, y_0]^\top$ and θ_0. Link 1 and link 2 are connected to the opposite ends of the base link, and can be thought of the arms and legs. The relative angles between link 1, link 2 and the base link are q_1 and q_2. The dynamics follows (4.12), with $q = [\theta_0, q_1, q_2]^\top$ and joint torques as input $U = [u_1, u_2]^\top$.

The system parameters are chosen to be proportional to human's torso, arms and legs, and are displayed in Table 4.1. The CoM of each link is located at the geometric center of the link.

Link i	mass(kg)	inertia(kgm^2)	length(m)
0	1	0.533	0.8
1	0.2	0.0167	1
2	1	0.1875	1.5

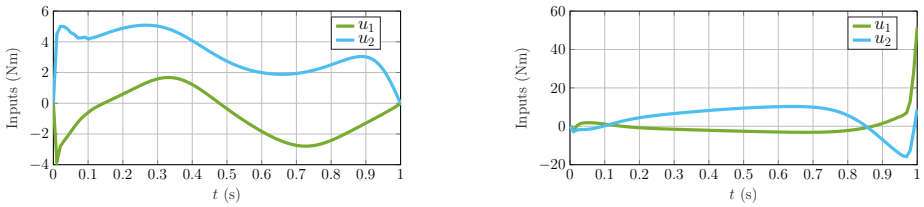
Table 4.1: Robot Parameters

Let the states be $x = [\theta_0, q_1, q_2, \dot{\theta}_0, \dot{q}_1, \dot{q}_2]^\top = [q, \dot{q}]^\top$ and $F_c = [I_3, O_3]^\top$. We will demonstrate different somersault motions generated with and without states constraints.

Free Somersault Planning We used the method introduced in this paper to plan a somersault in two different cases: in the first case, the initial kick is specified (e.g. a gymnast jumps as strongly as possible to increase its angular momentum) and we require the robot to land with the same pose as its initial pose. In the second case, we specify the landing angular velocities of torso and leg to be zero in order to have a stable landing, and solve for both the mid-air motion and the initial kick that will impart the robot with the necessary *momentum* to perform the motion.

It is well known that, when in mid-air, the *center of mass* of the robot will follow a ballistic motion determined by its initial velocity. This initial velocity can be, in turn, determined by the horizontal distance of the jump (we choose 6 here) and time allowed for it (T). Since its motion is unaffected by the controls, it can be precomputed and added to the position of the robot at the end. We did so here, and the position of the CoM is shown in the red curve in Fig. 4.6.

For motions without state constraints, the tensor $G(x)$ can be constructed from (3.17). Solving the AGHF with chosen boundary conditions, a solution curve is obtained and the control can be extracted using (3.18). Single Somersault motions can be realized by requiring $\theta(0) = 0$ and $\theta(T) = 2\pi$. We find the controls that generate them below for different choice of boundary conditions.



(a) Case 1 (b) Case 2

Figure 4.4: Controls for Case 1 and Case 2

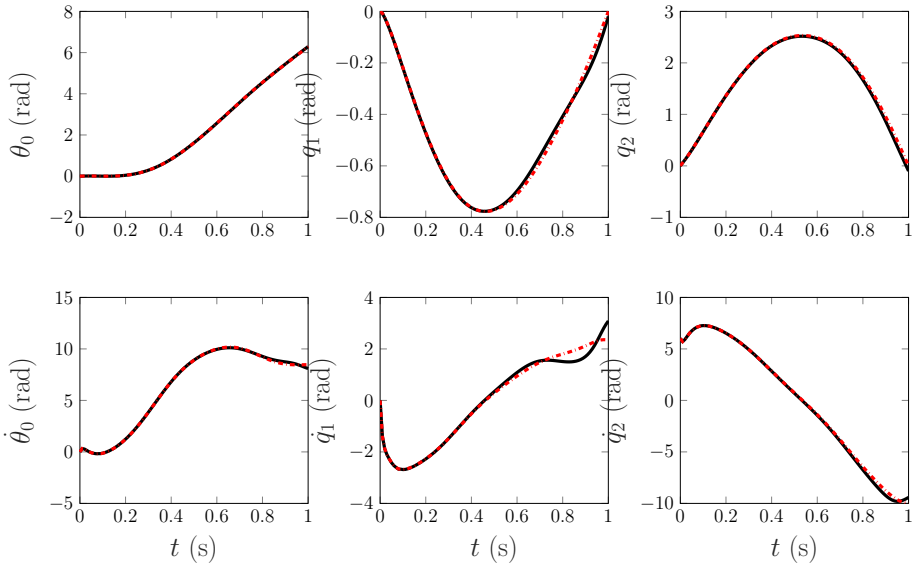


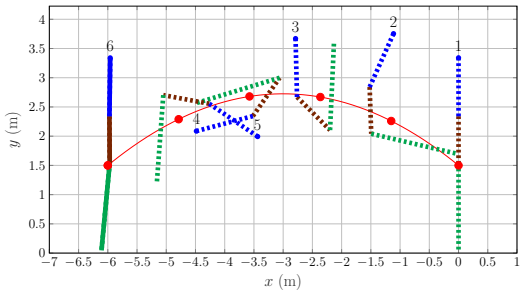
Figure 4.5: Trajectory of the states for Case 1. The red dashed lines is the trajectory solved by heat flow method. The solid black lines are actual trajectories driven by the extracted control.

4.4.5.1 Case 1: Free Final Velocities

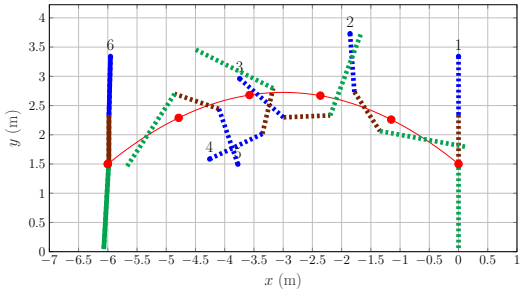
We use $x_{init} = [0, 0, 0, 0, 0, 6]^\top$ as initial state. That is, we assign a nonzero initial condition for $\dot{\theta}_2$, which can be analog to a initial kick with leg. The final state is set to be $x_f = [2\pi, 0, 0, \cdot, \cdot, \cdot]^\top$ which means that we have performed a 360 degrees rotation in the air and then land vertically. The final values for $\dot{\theta}_0$, $\dot{q}_1$ and $\dot{q}_2$ are free.

We set $\lambda = 1000$ and $s_{max} = 0.05$, and obtain $x^*(t) = x(t, s_{max})$ by solving the AGHF. We then extract the controls from (3.18) for $x^*(t)$, see Fig. 4.4a. Using the extracted control above, we solve the dynamic equations, shown in Fig. 4.5 by black lines, and we see that the robot performs the desired motion with these controls. Snapshots of this motion are given by Fig. 4.6a.

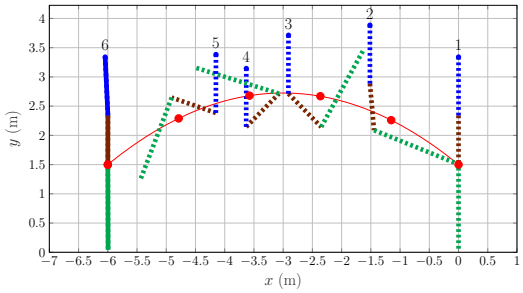
Here we predefined a ballistic CoM trajectory starting from right side which



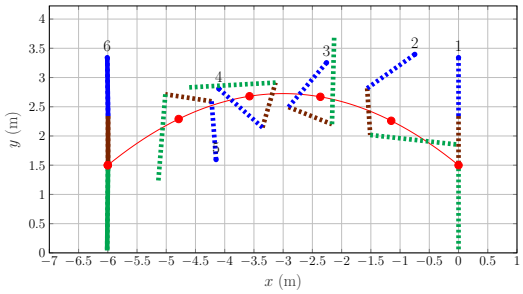
(a) Case 1



(b) Case 2



(c) Case 3



(d) Case 4

Figure 4.6: Snapshots of the somersault motions (right to left) in four cases. The blue links are the arm (link 1), green links are the leg (link 2) and brown links are the body (base link) of the robot. The red curves are the trajectories of CoM motion, with red dots indicating the position of CoM at each snapshots.

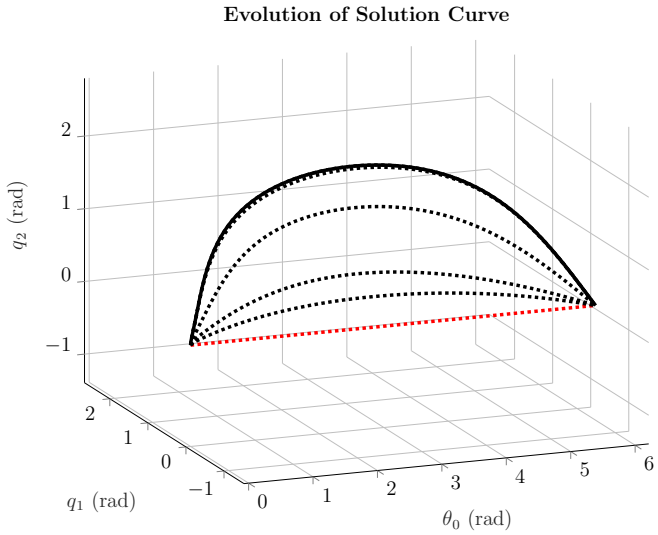


Figure 4.7: Evolution of state trajectory from initial guess (red) to steaty state (solid black).

gives $T = 1s$ for the robot to stay in the air. At time $t = 0$, the robot stands vertically at position $x = 0$. It is assumed to be facing right. The robot then kicks its leg counterclockwise with a speed of 6 rad/s before getting airborne (because $\dot{q}_2 = 6$). During the aerial phase, it bends its arm and leg to decrease the total inertia in order to speed up the overall angular velocity and land with the desired pose after having done one full rotation in the air. Before landing, it extend its limbs, making sure it will stand vertically when it lands. At $t = 1$, the robot lands with the desired pose (same as initial pose) at the left side.

Fig. 4.7 shows the evolution of the trajectory of $q(t)$ from initial guess $q(t, 0)$ (red) to the steady state solution $q(t, s_{max})$ (solid black) in 3D configuration space, namely, $q = [\theta_0, q_1, q_2]^\top$. The initial condition is a straight line connecting $x(0)$ and $x(T)$ in configuration space, which is not a feasible trajectory.

As s increase from 0 to s_{max}, the trajectory evolves from this initial guess to a steady state solution trajectory (black solid line) that minimize the actuated length.

Fig. 4.8 shows the convergence of actuated length to a minimum as s increase from 0 to s_{max}. At $s = 0$, the cost is high since the initial guess trajectory is arbitrarily chosen and the projection of $\dot{x} - F_d(x)$ on inadmissible directions is large, which got amplified by the large constant λ in $G(x)$. As

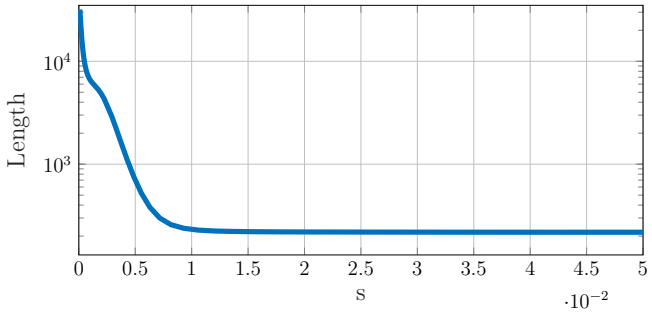


Figure 4.8: Actuated length converging to a local minimum.

s increases, we see that the curve evolves in a direction that decreases the length and eventually converges to a steady state, which corresponds to an admissible solution, i.e., a "short" curve for our Riemannian metric.

4.4.5.2 Case 2: Free initial kick

Another somersault motion is generated with different boundary conditions, see Fig. 4.6b. The initial state is $x_{init} = [0, 0, 0, 0, 0, \cdot]^\top$ and the final state is $x_f = [2\pi, 0, 0, 0, \cdot, 0]^\top$, which sets the initial value of $\dot{q}_2$ and final value of $\dot{q}_1$ to be free. This describe a somersault that starts with a *kick of the leg to be determined*, and ends with leg standing still ($\dot{q}_2(T) = 0$) but we allow the arm to move at landing. That is, we ask the algorithm to find what initial angular momentum is necessary to perform the motion, and this angular momentum is imparted by the leg at the start of the jump and "transferred" to the arm and the end. The algorithm is able to find a trajectory that meets these requirements. Note that in this case, the initial kick chosen by the algorithm is stronger than the one assigned manually in case 1, so that the limbs are not required to be tucked much during flight. An interesting extension would be to constrain this initial kick as well.

From the controls shown in Fig. 4.4 for case 1 and Fig. 4.4b for case 2, we see that the planning algorithm highlights two different strategies of motion. In case 1, since the initial kick is not strong, the controls are needed to tuck the limbs to increase the overall angular velocity. In case 2, the initial kick is stronger, and the controls are used to slow down the leg and accelerate the arm in order to transfer the momentum to the arm at the end.

The final pose in Fig. 4.6a and Fig. 4.6b are not exactly vertical for the value of λ chosen. Theoretically, larger λ gives a closer integrated path to

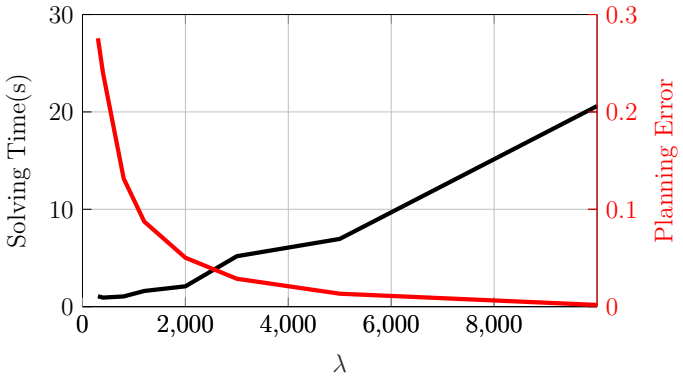


Figure 4.9: Solving time and Planning Error $|\tilde{x}(T) - x(T)|$ vs λ

the solution curve. Fig. 4.9 shows the planning error (red line) between the planned final state and desired final state for the motion in 4.4.5.1. The planning error decreases as λ increases.

4.4.6 Constrained Somersault Planning

In this section, we present motions with state constraints as formulated in Sec. 4.1. For motions with state constraints, the tensor $\hat{G}$ can be constructed from (4.7) and the constrained AGHF (4.9) can be determined analytically with the constraint $h(x)$. Solving the constrained AGHF with chosen boundary conditions, a solution curve is obtained and the control can be extracted using (3.18). Similar somersault motions as Case 1 can be realized with different types of constraints.

4.4.6.1 Case 3: Equality Constraint

To illustrate the motion planning with *equality constraints*, we will regenerate the somersault motion in Sec. 4.4.5.1 but with the arm staying vertical (orthogonal to the ground) during the whole motion. The expression for this constraint is $h(x) = \theta_0 + q_1 = 0$. From (4.3), $S(x) = 1$ for equality constraint. The same boundary conditions and initial condition from Case 1 is used, except that the final value for q_1 is set free so that the solver will find the final value of arm joint that satisfies the equality constraint.

We solved the constrained AGHF (4.9), with $\lambda = 2000$ and $\lambda_h = 20000$, to generate the motion shown in Fig. 4.6c, with the equality constraint, namely,

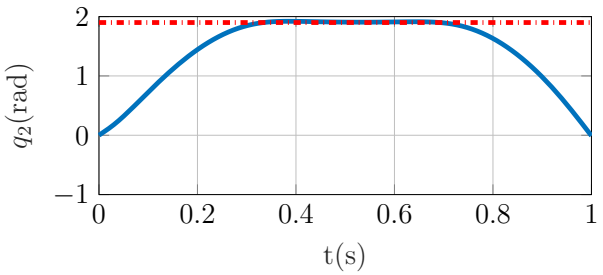


Figure 4.10: Trajectory of constrained state q_2 (black), and the joint limit (red)

arm staying vertical. The solver is able to find a motion in which the links coordinate with each other to keep the arm vertical, while satisfying the desired boundary conditions.

4.4.6.2 Case 4: Inequality Constraint

We now plan a motion where we limit the leg joint angle q_2: $q_2 \leq 1.9$. Therefore the inequality constraint is $h(x) = q_2 - 1.9 \leq 0$. From (4.3), a smooth logistic approximation of the step function is used to construct the switch function $S(x)$.

Solving the constrained AGHF (4.9) with the same boundary conditions and initial condition as Case 1, we obtain an admissible trajectory as shown in Fig. 4.6d. To obtain this motion, we chose $\lambda = 2000$ and $\lambda_h = 20000$. In this motion, we observe that the leg is further away from the torso, when compared with the leg in Case 1 (Fig. 4.6a), because the leg joint angle is limited by 1.9 rad. Hence, the constraint did indeed affect the motion. Because the total angular momentum is conserved, moving the leg further away from torso increases the inertia and decelerates the somersault. As a result, the arm has to move closer (compared to Case 1) to the torso in order to compensate the inertia increment from leg. The planned motion in Fig. 4.6d indeed shows this behavior. Fig. 4.10 shows the trajectory of the constrained leg joint angle q_2.

CHAPTER 5

PLANNING FOR LEGGED LOCOMOTION: AGHF WITH HYBRID DYNAMICS

Planning dynamic motions of legged robots has become an increasingly important topic, due in part to improved robot design and hardware, and in part to higher on-board computational capacity. Typical examples of such robots designs include the MIT Cheetah [1], the bipedal robot Cassie made by Agility Robotics and the Salto robot [2]. The major difficulty for legged locomotion planning lies in its *hybrid nature*: the dynamics of legged robots is governed by a set of equations and constraints depending on whether there is contact with the ground. Hybrid systems are well known to be difficult to handle; in fact, open questions remain even in the case of linear dynamics [13]. In this chapter, we show that the AGHF method we proposed for motion planning extends naturally to handle hybrid dynamics. Precisely, we show how the Ansatz developed in Chapters 3 and 4, contending that motion planning problems can be encoded into Riemannian metrics, can be applied to plan legged locomotion.

A variety of dynamic models have been used for different types of locomotion. Among which, the Linear Inverted Pendulum model is a simplified model that is used in cooperation with Zero Moment Point as the stability criterion for bipedal walking [46]. At the opposite extreme, full dynamics are used to plan the joint trajectories for motions with external contacts in [47, 48]. Centroidal dynamics, i.e. the dynamics of robot projected at its Center of Mass (CoM) [49], has been used for generating whole body motions of a humanoid robot [50] and hydraulic quadruped robot locomotion [51]. By using legs with light weight, or assuming the legs do not significantly deviate from their nominal pose, one can simplify the centroidal dynamics to single rigid body dynamics For example, high speed bouncing motions are achieved for quadruped [39], biped and quadruped locomotion in complex terrains are planned in [52]. In this work, we use the 2 dimensional model with massless legs, which we called a ***Single Rigid Body Model***.

Amongst the existing methods for legged locomotion planning, trajectory optimization (TO) formulates the problem as an optimization problem, e.g. in direct collocation and differential dynamic programming methods. In [47, 50], the hybrid dynamics of contact is modeled as a complementarity problem, with the ability to plan the contact locations. By convex modeling of the dynamics, convex optimization techniques are used in [53, 54] for faster convergence, while the footholds need to be pre-planned. The discrete nature of contact can be also modeled utilizing binary valued decision variables and solved by mix-integer solver, [51]. More recent work [52] plans both gait timings and contact locations automatically by modeling the contact dynamics individually. In this chapter, we extend the AGHF method presented in Chapters 3 and 4 to naturally encode the hybrid dynamics of legged locomotion. The key is to formulate the hybrid legged locomotion problem into AGHF framework with the help of an innovative switching mechanism. The extended algorithm can encode variety of constraints including contact constraints. Meanwhile, the convergence is guaranteed at a given order, as explained in Chapter 3.

5.1 The AGHF for hybrid dynamics

Let $x \in \mathbb{R}^n$ and $u \in \mathcal{U}$, we consider the following control affine time-dependent switched system. Detailed definitions and analysis of switched systems can be found in [13].

Definition 5.1 (Control Affine Time-dependent Switched System)**.** *The dynamics of the control affine time-dependent switched system is given by*

$$\dot{x} = F_{d,\sigma(t)}(x) + F_{\sigma(t)}(x)u \tag{5.1}$$

where $\sigma(t)$ is a piecewise constant function (called switching signal) $\sigma(t) : [0, \infty) \to \mathcal{I}$, with $\mathcal{I} = \{1, 2, \cdots, K\}$ is a finite index set with K indicating the number of subsystems, and the dwell-time of σ is uniformly lower bounded by a positive constant.

The role of σ is to specify, at each time instant t, the index $\sigma(t) \in \mathcal{I}$ of the active system, namely, the system being followed. The system dynamics of the active system at index σ is determined by the function pair $\{F_{d,\sigma}(x), F_\sigma\}$, which are defined in Def. 2.1 and we call such systems the *subsystems* or *modes*

of the system. We call the times at which σ is discontinuous the *switching times*, and the minimum dwell-time is the least difference between pairs of switching times.

A typical example of the switched system (5.1) is bipedal robot system. The robot dynamics depends on how many feet are in contact with ground and the switching of different dynamics can be defined by switching times.

We introduce the following activation functions for each subsystem of the switched system

Definition 5.2 (Activation Function)**.** *The activation function for the i-th subsystem is the piecewise constant function*

$$A_i(t) := \begin{cases} 1, & t \in [t_i, t_{i+1}) \\ 0, & t \in [0, t_i) \cap [t_{i+1}, \infty). \end{cases} \tag{5.2}$$

where $i \in \mathcal{I}$ is the index of the subsystem. The switching times t_i and t_{i+1} are the beginning and ending times of the subsystem. In addition, we have the following constraint

$$\sum_{i=1}^{K} A_i(t) := 1, t \in [0, t_{K+1}) \tag{5.3}$$

Recall that $K < \infty$ is the total number of subsystems and t_{K+1} is the ending time of the K-th subsystem, which equals to the motion duration, $t_{K+1} = T$. The constraint (5.3) ensures that for any $t \in [0, t_{K+1})$, there is a valid and unique subsystem that is active. With the activation function (5.2), the switched system (5.1) can be represented by

$$\dot{x} = \sum_{i=1}^{K} A_i(t)(F_{d,i}(x) + F_i(x)u), i \in \mathcal{I} \tag{5.4}$$

The Riemannian metric also relies on the activation functions to determine the appropriate metric to use, which of course depends on the current dynamical regime of the system:

Definition 5.3 (Switched Riemannian Metric)**.** *The switched Riemannian metric associated with the curve $x(t)$ given by* (5.4) *is*

$$G(x, t) = \sum_{i=1}^{K} A_i(t) G_i(x), i \in \mathcal{I} \tag{5.5}$$

where $G_i(x)$ is the Riemannian metric for the i-th subsystem and can be constructed by (3.16) *and* (3.17) *using $F_i(x)$.*

By this definition, the metric of each subsystem, $G_i(x)$, is activated for $t \in [t_i, t_{i+1})$, therefore the overall metric $G(x,t)$ is dependent with time t. Similar as (3.5), the actuated curve length given by the metric (5.5) is:

Definition 5.4 (Switched Actuated Curve Length)**.**

$$\mathcal{L}_G(x, \dot{x}) := \sum_{i=1}^{K} \int_0^T A_i(t)((\dot{x} - F_{d,i}(x))^\top) G_i(x)(\dot{x} - F_{d,i}(x)))^{1/2} dt. \tag{5.6}$$

The switched actuated curve length is the summation of the active portions of each subsystem's curve length.

Lemma 5.1 (Switched AGHF)**.** *The AGHF of the switched actuated curve length is*

$$\frac{\partial x(t,s)}{\partial s} = \sum_{i=1}^{K} A_i(t)\phi_i(t,s) \tag{5.7}$$

where $\phi(t,s)$ is the AGHF of each subsystem defined by (3.5) *using G_i and $F_{d,i}$ of the i-th subsystem. Let $x^*(t)$ be a steady-state solution of the switched AGHF* (5.7)*. Then $x^*(t)$ is an extremal curve for the switched actuated curve length* (5.6)*. Furthermore, the switched actuated curve length decreases along the solutions of the switched AGHF.*

Proof. The proof follows the same lines as the proof of Theorem. 3.1, where the optimality is obtained piecewise for each 'period' in the switching signal. More precisely, the switched actuated curve length can be rewritten as

$$\mathcal{L}_G(x, \dot{x}) = \sum_{i=1}^{K} \int_{t_i}^{t_{i+1}} ((\dot{x} - F_{d,i}(x))^\top) G_i(x)(\dot{x} - F_{d,i}(x)))^{1/2} dt \tag{5.8}$$

and for each time span $t \in [t_i, t_{i+1})$, the switched AGHF can be expressed as

$$\frac{\partial x(t,s)}{\partial s} = \phi_i(t,s), t \in [t_i, t_{i+1}) \tag{5.9}$$

From Theorem. 3.1, each of the AGHF $\phi_i(t,s)$ is a curve shortening flow for the curve length $\int_{t_i}^{t_{i+1}} ((\dot{x} - F_{d,i}(x))^\top) G_i(x)(\dot{x} - F_{d,i}(x)))^{1/2} dt$, for time span $t \in [t_i, t_{i+1})$. Therefore, the flow (5.7) that is defined by concatenating the

subsystem AGHF for all $i \in \mathcal{I}$, shortens the summation of the actuated curve lengths of all the time intervals, which is the switched actuated curve length (5.6). □

Example: We illustrate the use of switched AGHF via the following unicycle example.

The unicycle dynamics is governed by (3.21) with unicycle position and orientation as states, $x = [q_x, q_y, \theta]^\top$, linear and angular velocity as inputs, $u = [u_1, u_2]^\top$. We now consider a 2-phase motion. The first phase is for $t \in [0, 1)$, where the linear velocity is free and angular velocity is constant, $u_2 = 1$. The second phase is for $t \in [1, 2]$, where the angular velocity is free but linear velocity is constant, $u_1 = 1$. The resulting system is a switched system that can be modeled by (5.4), with $\mathcal{I} = \{1, 2\}$, and

$$\begin{aligned} A_1 &= H(t) - H(t-1) \\ F_1 &= f_1 \\ F_{d,1} &= f_2 \end{aligned}$$

for the subsystem in first phase

$$\begin{aligned} A_2 &= H(t-1) - H(t-2) \\ F_2 &= f_2 \\ F_{d,2} &= f_1 \end{aligned}$$

for the subsystem in the second phase. The column vectors f_1 and f_2 are the unicycle heading direction and turning (angular) direction, as defined in (3.21).

The activation functions $A_1(t)$ and $A_2(t)$ are constructed using heaviside step function $H(\cdot)$. The subsystem dynamics are distinguished by the two different sets of $\{F_i, F_{d,i}\}$. The switched Riemannian metric can be obtained

via (5.5) with

$$
\begin{aligned}
G_1 &= (\bar{F}(x)^{-1})^\top D_1 \bar{F}(x)^{-1} \\
D_1 &= \mathrm{diag}(\lambda, 1, \lambda) \\
G_2 &= (\bar{F}(x)^{-1})^\top D_2 \bar{F}(x)^{-1} \\
D_1 &= \mathrm{diag}(\lambda, \lambda, 1)
\end{aligned}
$$

in which $\bar{F}$ is the invertible matrix that contains both admissible and inadmissible directions. For the unicycle, $\bar{F}$ is

$$
\bar{F}(x) = \begin{bmatrix} f_c & f_1 & f_2 \end{bmatrix} = \begin{bmatrix} \sin\theta & \cos\theta & 0 \\ -\cos\theta & \sin\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5.10}
$$

where the inadmissible direction f_c is the side slip direction. The key difference between the metrics lies in the penalty matrices D_1 and D_2. With D_1, the motion steering on f_c and angular direction f_2 will cause large curve length, however with D_2, the motion steering on f_c and linear direction f_1 will cause large curve length.

As a result, the construction above is able to capture the two distinct unicycle dynamics in two phases. That is, by constructing the switched Riemannian metric, the actuated curve length is able to encode the two dynamics of two different phases. By solving the corresponding AGHF (5.7), a motion that follows different dynamics in different phases can be obtained. □

In the remaining sections, we apply the above-developed framework to motion planning of a legged robot. The legged locomotion dynamics is switched because the number of actuating contact forces depends on what mode of the robot: for example, single leg support mode, double leg support mode, and so on. The constraints are mode dependent as well. A typical constraint is the friction cone constraint of the contact force, and it is only active when the foot is in contact with the ground. We will first introduce the modeling of legged dynamics and its constraints. Then the motion planning framework for switched system is applied, with a more specific variation of the activation functions designed for legged locomotion. At last, an approach for motion planning of switched system with indefinite switching time is proposed

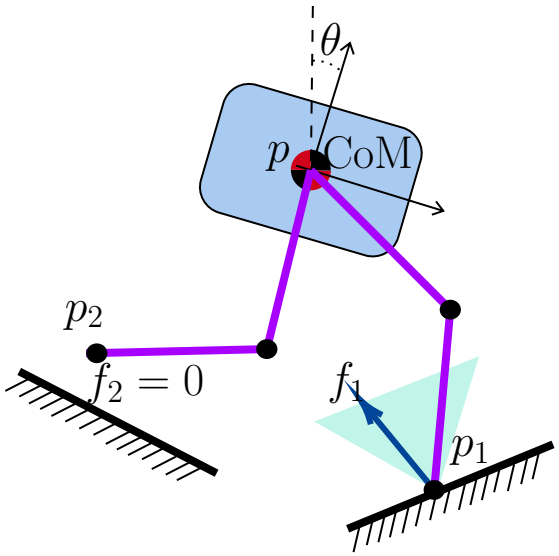


Figure 5.1: 2D Single Rigid Body Model

and applied on legged locomotion planning.

5.2 Legged Robot Dynamics

In this section, the modeling of legged robot is introduced. The model includes the robot system dynamics and legged locomotion constraints.

5.2.1 Single Rigid Body Model

We focus on the problem of planning the motion of a 2D legged robot with massless legs–the Single Rigid Body Model, see Fig. 5.1. The torso of robot is a rigid body with mass M and inertia I around its CoM. The CoM position is $p = [p_x, p_y] \in \mathbb{R}^2$ and the torso's orientation is $\theta \in \mathbb{T}^1$.

The robot has $k > 0$ legs, each leg has point foot at the distal end, with coordinate $p_i = [p_{ix}, p_{iy}]^\top \in \mathbb{R}^2$. The contact force applied on the point foot at p_i is $f_i = [f_{ix}, f_{iy}]^\top \in \mathbb{R}^2$. The number of leg links and their lengths are not predetermined. We ensure the joint angles are feasible by adding constraints on the foot and hip positions; for example, if a leg has two links from hip to foot and one joint at knee, then the joint angle is feasible if the distance between the foot and hip is smaller than the sum of link lengths. This kinematic constraint is discussed in Section 5.2.2. The joint torques and

contact forces can be mapped to each other by the foot Jacobians for the massless leg robot. Therefore we directly use the contact forces as inputs to the system. The equations of motion for the robot thus are:

$$\ddot{p} = \frac{1}{M}\sum_{i=1}^{k} f_i - \begin{bmatrix} 0 & g \end{bmatrix}^\top, \; \ddot{\theta} = \frac{1}{I}\sum_{i=1}^{k} f_i \times (p - p_i) \tag{5.11}$$

where g is the gravitational acceleration. The terrain is **not** assumed to be flat and is given as the zero-set of a C^2 function

$$f_{\text{terr}}(c_x, c_y) = 0 \tag{5.12}$$

where $[c_x, c_y]^\top$ is a point on the 2D terrain.

5.2.2 Constraints for Legged locomotion

For each leg, there are two different modes or phases:

1. Stance phase: the foot is in no-slip contact with a surface

2. Flight phase: the foot is in the air

We now describe the constraints in different phases. In **stance phase** for leg i:

$$\dot{p}_i = 0 \tag{5.13}$$

$$f_{\text{terr}}(p_{ix}, p_{iy}) = 0 \tag{5.14}$$

$$f_i \cdot \overrightarrow{N}(p_i) \geq 0 \tag{5.15}$$

$$|f_i \cdot \overrightarrow{T}(p_i)| \leq \mu f_i \overrightarrow{N}(p_i) \tag{5.16}$$

the foot is in contact with the ground and has zero velocity: (5.13)-(5.14). The force f_i is generated through contact with the ground, thus subject to the following constraints: let $\overrightarrow{N}(p_i)$ and $\overrightarrow{T}(p_i)$ be the unit normal and tangent vectors at the contact point p_i, which can be directly calculated using the gradient of terrain function $f_{\text{terr}}(\cdot, \cdot)$. The foot can only push against the ground: the projection of f_i in normal direction to the surface at the contact point has to be positive (5.15). Furthermore, the contact force is constrained by a friction cone, formed by the normal contact force and the

friction coefficient μ, as shown by the light blue triangle in Fig. 5.1; see (5.16). In **flight phase** for leg i, there is no contact force and the feet are above the ground:

$$|f_i| = 0 \tag{5.17}$$

$$f_{\text{terr}}(p_{ix}, p_{iy}) \geq 0 \tag{5.18}$$

Finally, the following constraints enforce that the joint angles are feasible and hold for **both phases**:

$$|p - p_i| \leq R \tag{5.19}$$

$$f_{\text{terr}}(p_x, p_y) \geq h_c \tag{5.20}$$

Indeed, assuming that the hip for all legs are at CoM and the leg links are connected in series, the feasibility of joint angles can be ensured (5.19), where R is chosen to be less than the sum of the leg link lengths. One can make R smaller to avoid approaching singularity configurations of the leg. If no collision with the torso is desired, this constraint can be replaced by $(|p - p_i| - R)^2 \leq \Delta R^2$ with proper ΔR. Constraint (5.20) ensures that the CoM is higher than the terrain height by some positive constant h_c.

5.2.3 State Space Model

To represent the system in form of (3.3), define the state:

$$x = [p, \theta, \dot{p}, \dot{\theta}, f_1, p_1, f_2, p_2, ...f_k, p_k]^\top \in \mathbb{R}^{6+4k}$$

in which the original controls f_i and p_i are states of the system, and introduce the new controls: $[u_i, v_i]^\top = [\dot{f}_i, \dot{p}_i]^\top \in \mathbb{R}^4$, which are the rate of change for the original controls f_i and p_i. Now, the control to the system is $u = [u_1, v_1, \ldots, u_k, v_k]^\top$. This operation allows us to encode constraints on the original controls as state constraints, as discussed in Sec. 5.3.3. Denoting $m \times n$ zero matrix by $O_{m\times n}$ and $k \times k$ identity matrix by I_k, we can write

the system in form of (3.3) with:

$$F_d(x) = \begin{bmatrix} \begin{bmatrix} x_4 & x_5 & x_6 \end{bmatrix}^\top \\ \frac{1}{M}\sum_{i=1}^{k} f_i - \begin{bmatrix} 0 & g \end{bmatrix}^\top \\ \frac{1}{I}\sum_{i=1}^{k} f_i \times (p - p_i) \\ O_{4k\times 1} \end{bmatrix}, \; F(x) = \begin{bmatrix} O_{6\times 4k} \\ I_{4k} \end{bmatrix} \tag{5.21}$$

where, from the definition of x above, $f_i = [x_{3+4i}, x_{4+4i}]^\top$, $p_i = [x_{5+4i}, x_{6+4i}]^\top$ and $p = [x_1, x_2]$. The 2-D cross product "$\times$" for torque calculation is defined as $[x_1, y_1]^\top \times [x_2, y_2]^\top = x_1 y_2 - x_2 y_1$. The drift term $F_d(x)$ includes all the robot dynamics, and the columns of $F(x)$ are the actuated directions, in other words, the directions that can be directly controlled by u.

The system above is a switched system since the actuation of control u depends on the mode. For example, due to constraint (5.13), the control v_i is disabled during stance phase of the i-th leg. Furthermore, the state constraints will be formulated in a similar fashion in Sec. 4.1, where augmented states are introduced. The augmented states also follow some switched dynamics since the state constraints are switched based on the modes.

5.3 A Riemannian metric for Legged Locomotion

The motion planning problem is to find a trajectory for x obeying (4.16) under constraints (5.13) to (5.20), for given boundary conditions, time span T. The challenge is that the constraints are different during flight and stance phases and the availability of some controls also depends on what phase the foot is in. We solve the motion planning problem by specializing the generic framework designed for switched system in Sec. 5.1 to legged robot system. In this section, instead of defining the activation functions for each subsystem, we define activation functions for the phases of each leg. The underlying idea remains the same, and we first work under the assumption that the schedule of phases are predefined, i.e., the timing of taking off and landing of each leg are set in advance. This results in a switched Riemannian metric. The transition of a constraint from active to inactive is a discrete event, which we formulate with the help of a redefined *Activation Function* in Sec. 5.3.1. Then the input and state constraints are equipped with proper activation

functions, which are discussed in Sec. 5.3.2 and Sec. 5.3.3.

5.3.1 Activation Function

The key step in the formulation of the phase-dependent constraints is the activation function for legs: The activation function $A_i(t, x)$ for the i-th leg is a binary valued function which determines whether the leg is in stance or flight:

$$A_i(t, x) := \begin{cases} 1, & \text{if foot in stance} \\ 0, & \text{if foot in flight.} \end{cases} \tag{5.22}$$

In general, the activation function can be dependent on state and time. For a predefined contact schedule, it is only dependent of time. Throughout the remaining sections of this chapter, the term "activation function" specifically refers to the activation functions for legs, unless otherwise noted.

In the case that the timing of landing and take off is given, the activation can be expressed by Heaviside unit step functions:

Definition 5.5 (Activation function for predefined contacts)**.** *If the time sequences $\{t^j_{i,1}\}_j$ and $\{t^j_{i,2}\}_j$ of landing and take-off time are given, where j indicates the j-th step and i is the leg index. The activation functions $A_i(t)$ are defined as:*

$$A_i(t) := \sum_{j=1}^{k} H(t - t^j_{i,1}) - H(t - t^j_{i,2}). \tag{5.23}$$

where k is the total number of steps of the foot and $H(c)$ is a Heaviside unit step function where the step time is at c.

5.3.2 Fixed Contact Foot Position Formulation

To encode the switching between free and constant foot velocity, i.e. flight and stance phases, we define the Riemannian metric G similarly to (3.17), with a *time varying* penalty matrix $D(t)$ equipped with activation functions of all legs:

Definition 5.6 (Metric for legged locomotion)**.** *The time-varying Riemannian metric $G(x, t)$ that encodes the activation and deactivation of constant foot*

velocity constraint is defined by Eq. (3.17) *with*

$$D = \text{diag}(\underbrace{\lambda, \cdots, \lambda}_{6}, \Lambda_1, \cdots, \Lambda_k), \tag{5.24}$$

where $\Lambda_i = \text{diag}\left(1, 1, 1+\lambda A_i(t), 1+\lambda A_i(t)\right)$ *and* $A_i(t)$ *is the activation function for leg* i*.*

The term $1 + \lambda A_i(t)$ in Λ_i penalizes the length of trajectories that use the control v_i. If the foot is in stance, the value of A_i is 1 and the curve length is increased due to a nonzero v_i multiplied λ; as a result, v_i will be minimized. When the foot is in flight phase, A_i is 0, therefore the value of v_i is free as it will not affect the curve length. Hence, curves of minimal length for this metric are so that the constraint (5.13) is met when the foot is in stance.

With the penalty term for u_i set to 1, we are not constraining the changing *rate* of contact forces, $\dot{f}_i$, in the Riemannian metric. The constraints on f_i (cone (5.16) and positivity (5.15) constraints) will be encoded in Sec. 5.3.3 as state constraints. In the formulation (5.24), the control is either free or constrained to zero depending on the time-dependent penalty matrix $D(t)$. However, one can also have constraints on the magnitude of control by letting u be a state of the system which is directly controlled by a newly introduced unconstrained control. The constraint on u is then a state constraint. This is exactly the intention of formulating contact forces and foot positions as states in Sec. 5.2.3.

Owing to the simple structure of F in (4.16), we take $F_c := [I_6, O_{4k\times 6}]^\top$ so that $\text{span}\{F_c\} \perp \text{span}\{F\}$ and $\bar{F} = I_{4k+6}$ is full rank for all x.

5.3.3 State Constraints Formulation

The state constraints can be formulated in a similar way as Sec. 4.1, but with the ability to activate and deactivate phase dependent constraints. Each of the state constraints from (5.14) to (5.20) can be formulated as scalar function $h(x) = 0$ for equality constraints or $h(x) \leq 0$ for inequality constraints. For example, for constraint (5.20) we have to encode the activation/deactivation of the state constraints in different phases. To this end, denote by $h_j(x)$, $j \in \mathcal{Z}^+ \leq k_c$, the j-th scalar constraint function with k_c the number of such constraints. We apply the following method to *each* constraint individually.

The first step is the same as the state constraint in Sec. 4.1. We add one state per constraint, denoted by ζ_j, resulting in the **augmented state** of the system: $\hat{x} := [x^\top, \zeta_j]^\top$. The new state ζ_j keeps track of the *accumulated signed* error between $h_j(x)$ and zero: $\zeta_j(t) := \int_0^t h_j(x(\tau))S_j(\tau, x(\tau))d\tau \longrightarrow \dot{\zeta}_j := h_j(x(t))S_j(t, x(t))$, where $S_j(t, x)$ is a scalar *switch function* for different type of constraints:

$$S_j(t,x) = \begin{cases} B_j(t), & \text{for equality constraint.} \\ H(h_j(x))B_j(t), & \text{for inequality constraint.} \end{cases} \tag{5.25}$$

in which $H(\cdot)$ is the Heaviside function used in (5.23), $B_j(t)$ is the *constraint activation function* for the j-th scalar state constraint and is defined below. A constraint can be active either for the duration of the motion (e.g., bound on the joint angle), or depending on the phase in the motion. The presence of the switch function in (5.25) is the main difference with the switch function (4.3) in Sec. 4.1.

Definition 5.7 (Constraint activation function)**.** *The constraint activation function for the j-th scalar state constraint is given by:*

$$B_j(t) = \begin{cases} 1 & \textit{if it holds all the time} \\ A_i(t) & \textit{if it holds in stance phase of foot } i \\ 1 - A_i(t) & \textit{if it holds in flight phase of foot } i. \end{cases} \tag{5.26}$$

For example, the activation function for constraint (5.17) of the j-th leg is $1 - A_j(t)$.

The augmented system dynamics is thus given by

$$\dot{\hat{x}} = \hat{F}_d(\hat{x}) + \hat{F}(\hat{x}) \begin{bmatrix} u \\ h_j(x)S_j(t,x) \end{bmatrix} \tag{5.27}$$

where

$$\hat{F}_d(\hat{x}) = \begin{bmatrix} F_d(x) \\ 0 \end{bmatrix}, \; \hat{F}(\hat{x}) = \begin{bmatrix} F(x) & O_{6+4k,1} \\ O_{1,6+4k} & 1 \end{bmatrix}. \tag{5.28}$$

5.3.4 Actuated Curve Length and Geometric Heat Flow

The Riemannian metric $G(t)$ defined by (3.17) and (5.24) is constructed so that the length of paths violating the constraints is large. We do so by introducing the *augmented metric* for legged locomotion, which has the same format as (5.8):

Definition 5.8 (Augmented metric)**.** *The Riemannian metric for the augmented system* (5.27) *is*

$$\hat{G}(t) := \begin{bmatrix} G(t) & O_{6+4k,1} \\ O_{1,6+4k} & \lambda_j \end{bmatrix} \tag{5.29}$$

where $G(t)$ is the Riemannian metric defined for the original states x by (3.17) *and* (5.24)*, and λ_j is a large constant.*

With this metric, we obtain that the actuated curve length of the augmented system is

$$\hat{L} = \int_0^T \left((\dot{\hat{x}} - \hat{F}_d(\hat{x}))^\top \hat{G}(t)(\dot{\hat{x}} - \hat{F}_d(\hat{x}))\right)^{1/2} dt \tag{5.30}$$

For the augmented system, we define the *constrained AGHF*:

$$\frac{\partial x(t,s)}{\partial s} = \nabla_{\dot{x}(t,s)} \left(\dot{x}(t,s) - F_d\right) + r(t,s) - \lambda_j \frac{\partial (h(x)^2 S(t,x))}{\partial x} \tag{5.31}$$

where the first two terms of the right-hand side are the same as (3.6). Similarly as in Section 4.1, the constrained AGHF (5.31) minimize the actuated curve length of the augmented system:

$$\begin{aligned} &\int_0^T \left((\dot{\hat{x}} - \hat{F}_d(\hat{x}))^\top \hat{G}(t,\hat{x})(\dot{\hat{x}} - \hat{F}_d(\hat{x}))\right)^{1/2} dt \\ &= \int_0^T \left((\dot{x} - F_d(x))^\top G(t,x)(\dot{x} - F_d(x)) + \lambda_j h(x)^2 S(t,x)\right)^{1/2} dt \end{aligned} \tag{5.32}$$

and plugging in the defining of $\hat{G}$, we get

$$\hat{L} = \int_0^T \left((\dot{x} - F_d(x))^\top G(t)(\dot{x} - F_d(x)) + \lambda_j h_j^2(x) S_j(t,x)\right)^{1/2} dt.$$

This augmented curve length has the same format as (4.8). However, with the

switch function redefined for phase dependent constraints, the curve length reflected the contribution of these phase dependent constraints. Compared to (3.5), it contains the additional term $\lambda_j h_j(x)^2 S_j(x)$. When the constraint $h_j(x)$ is active, namely, $B_j(t) = 1$: for equality constraint $h_j(x) = 0$, by construction $S_j(x) = 1$, the actuated length is penalized by λ_j when $h_j(x)$ is not close to zero, driving $h_j(x) \to 0$. For an inequality constraint $h_j(x) \leq 0$, the actuated length is penalized by λ_j when $h_j(x) > 0$ and $S_j(x) = H(h_j(x)) \approx 1$, driving $h_j(x) \to 0$. However, this term has no effect on the actuated length if $h_j(x) \leq 0$ since $S_j(x) = H(h_j(x)) \approx 0$. Hence, minimizing the actuated length of an augmented state trajectory results in *minimizing the violation* of the state constraints while it is active. In conclusion, solving the AGHF derived from Lagrangian (5.32) leads to a curve with minimum actuated length, which is a motion admissible for system (5.11) and respects the constraints (5.13)-(5.20).

5.3.5 Step Function Approximation

Since the AGHF requires to take derivatives of a Heaviside step function (whose derivative formally does not exist as a function), and in order to avoid having to implement a solver which works for piecewise continuous systems by solving for each pieces, we approximate the step function $H(c)$ and its derivative $\frac{dH}{dc}(c)$ using a logistic approximation (4.5) (revisited below). It approximates a unit step at $c = 0$, namely, $H(c) \approx 0$ if $c < 0$ and $H(c) \approx 1$ if $c > 0$. The constant α controls the accuracy of the approximation. The derivative is $\frac{\alpha e^{-\alpha c}}{(1+e^{-\alpha c})^2}$, which causes overflow problem when evaluating its value numerically for large α. To circumvent this numerical issue, we use zero-centered normal distribution (5.34) to approximate the derivative of the step function, where β is a large number scaling the value of $\dot{H}(c)$ at $c = 0$.

$$H(c) := \frac{1}{1 + e^{-\alpha c}} \tag{5.33}$$

$$\frac{dH}{dc} := \frac{1}{\beta\sqrt{2\pi}} e^{-(c/\beta)^2} \tag{5.34}$$

5.3.6 Planning Algorithm Summary

The steps of the algorithm can be summarized as follow:

Step 1: Specify the number of legs k, find $F_d(x)$ and $\bar{F}$.

Step 2: Specify timing for stance/flight phases of each leg, construct $A_i(t)$ for all i to build $D(t)$. Then construct metric $G(t)$ with $\bar{F}$ and $D(t)$ from (5.24), using large λ.

Step 3: Specify the terrain $f_{terr}(\cdot,\cdot)$, μ, R, h_c. Construct switch functions $S_j(t)$ for the state constraint using $A_i(t), B_j(t)$. Formulate all state constraints $h_j(x)$ for (5.14)–(5.20) as extra states equipped with $S_j(t)$ to get the augmented states $\hat{x}$.

Step 4: Get the augmented drift $\hat{F}_d$ from F_d, construct the metric $\hat{G}$ (5.29) using G and large λ_j.

Step 5: Find the actuated length (5.30) using $\hat{G}$, $\hat{F}_d$, then find the corresponding AGHF.

Step 6: Solve AGHF with BC (3.14) and IC (3.15) and large enough s_{max} to obtain the solution $x^*(t)$.

Step 7: Extract control $u(t)$ from $x^*(t)$ using (3.18). Integrate the dynamics (5.21) with control $u(t)$ and initial value $x^*(0)$ to obtain the integrated path $\tilde{x}(t)$, which is the planned motion.

5.4 Application Example: Two Leg Hopping

We illustrate the performance of our method in planning motion for a two leg hopping robot on uneven terrain with sinusoidal profile, see Fig. 5.3 for snapshots of the planned motion[1]. The initial condition to solve the AGHF is a straight line connecting x_{init} to x_{fin} which is not a feasible trajectory. Choosing the constant $\lambda = 10^6$ and solving the AGHF with a large $s_{max} = 0.0005$ in pdepe from the Matlab PDE toolbox, we obtain $x^\star(t)$. Using the extracted control (3.18) to integrate the system dynamics (5.21), the

[1]Supplementary animations can be found in playlist: `https://www.youtube.com/playlist?list=PLRi8ecX8Oy0UZzYoRi2DASQfU16sKtRph`

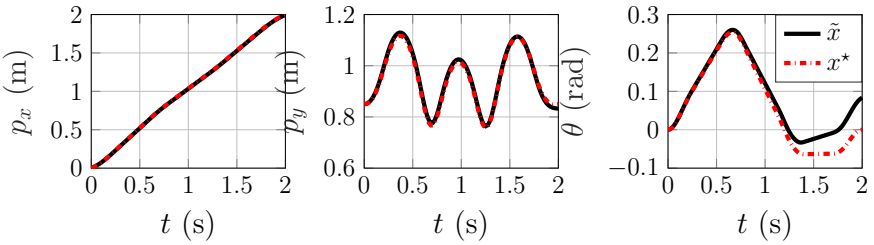


Figure 5.2: Torso position and orientation trajectories for AGHF solution $x^\star$ and integrated path $\tilde{x}$. Error is present between $x^\star$ and $\tilde{x}$.

planned motion $\tilde{x}(t)$ can be obtained. The terrain has a sinusoidal profile, $f_{\text{terr}}(c_x, c_y) = c_y - 0.1\cos(4\pi c_x)$. The torso (cyan box) has a mass $M = 2\,kg$ and inertia $I = 1\ kg \cdot m^2$. The leg is rooted at the CoM of torso and the maximum radius R in constraint (5.19) is 1 m. The CoM is higher than $h_c = 0.3$ in (5.20). Friction coefficient is $\mu = 1$. The goal is to move the CoM from $[0, 0.85]^\top$ to $[2, 0.85]^\top$ with $T = 2$ s and 3 hops. The stance/flight phase timings for leg 1 are predefined to have a $1:1$ ratio while the timing of the second leg is shifted by -0.05 s, so that the two legs are not synchronized. The generated motion shows the following gaits: Leg 1 kicks off at the first hump, then steps on the second and third hump while leg 2 kicks off at the first hump and then steps on the third and fourth hump. Both feet land on the last hump at the end.

Fig. 5.2 shows the trajectories of torso position and orientation. The trajectory is not necessarily periodic for each step since the timing of contact for each foot is not synchronized. The joint angles are guaranteed to be feasible since the distances between feet and the hip are constrained to be less than the maximum distance each foot can reach. The algorithm is able to find a motion with different contact points for each leg while enforcing all the constraints. The contact points being automatically chosen, even in the case of uneven terrain, is quite advantageous compared to methods requiring the user to predefine the contact positions or safe contact regions, as suboptimal choices here can strongly decrease the probability to find feasible solutions. However, this is done at the expense of a terrain function that needs to be C^2.

With the friction cone constraints enforced, we see that the feet step on the inclined contact points where enough friction can be provided. Fig. 5.4 top shows the contact forces for leg 1. The predefined stance/flight phases are

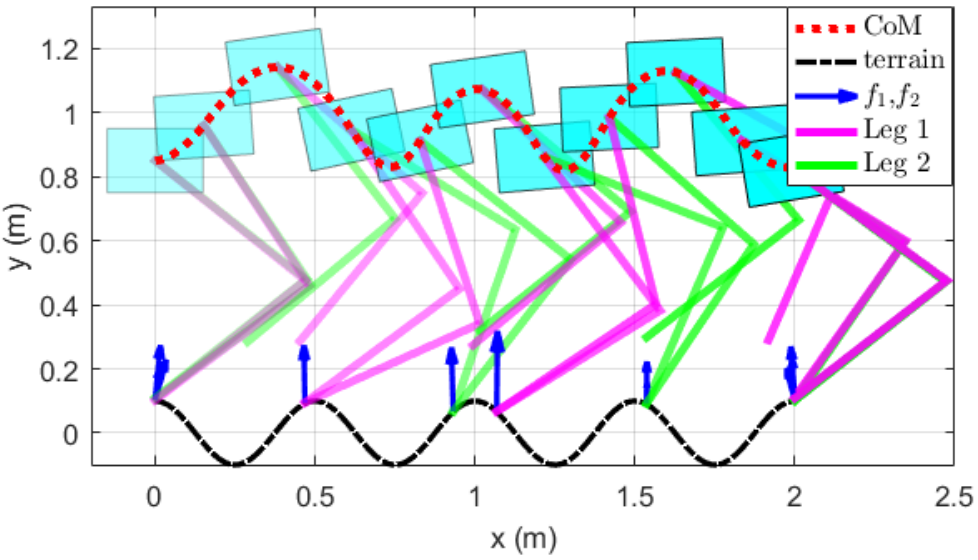


Figure 5.3: Snapshots of two leg hopping.

indicated by light blue/red backgrounds. Both normal and tangential force being zero in flight phases reflects the constraint (5.17). In each stance phase, the normal force is larger than zero, as in (5.15). With $\mu = 1$, The friction cone (5.16) is $|f_i \cdot \overrightarrow{T}(p_i)| \leq f_i \overrightarrow{N}(p_i)$. Namely, the positive normal force and its opposite value (red dotted line) serve as the upper and lower bounds for tangential force. It is clear in Fig. 5.4 top, the value of tangential force is close to but strictly inside the boundary of the friction cone.

The fixed foot position constraint (5.13) during stance phase is also achieved as shown in Fig. 5.4-2, where the foot velocity $\dot{p}_1$ is zero during stance phases. This results in constant foot position, as shown in Fig. 5.4 third and bottom for x and y component of foot position p_1. Furthermore, in Fig. 5.4 bottom, the foot height p_{1y} is constrained to be the same as terrain height to make sure the foot is in contact with the ground during stance. This reflects the constraint (5.14). For flight phases, the algorithm is able to plan the smooth swing motion trajectory of the foot by actuating p_1 with proper $\dot{p}_1$.

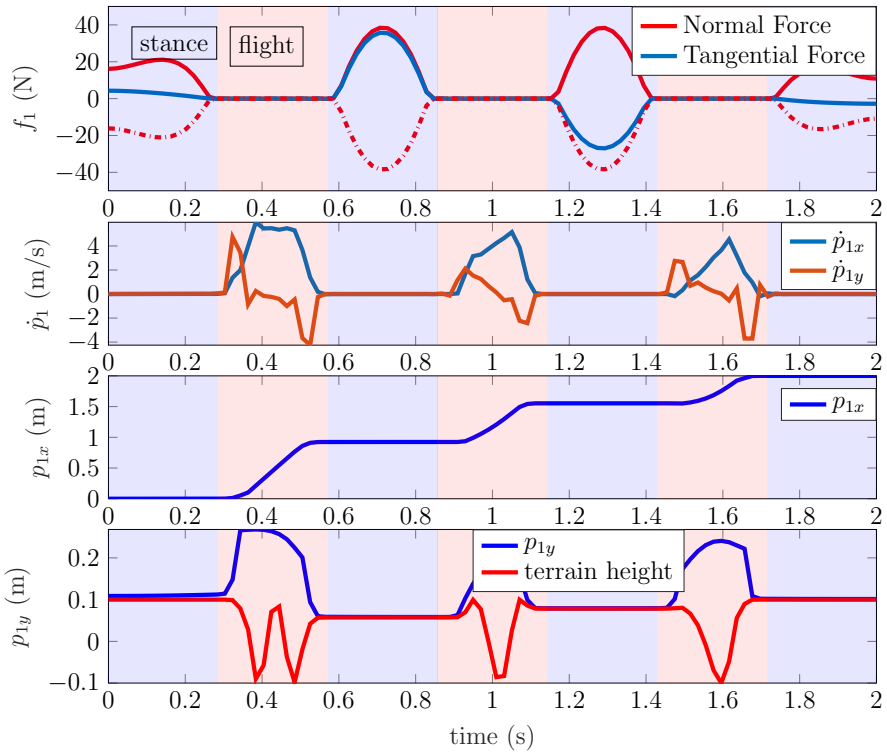


Figure 5.4: Foot contact force (top), velocity (second) and position (third and bottom) of leg 1

5.5 Indefinite Switching Times: Variable Scheduling of Gait

We use the term contact schedule to refer to the contact sequence and timing of the stance/flight transitions. A 'poorly' predefined contact schedule for the legs will result in a motion planning problem without admissible solution. Of course, it is difficult to know beforehand whether a chosen schedule is 'poorly' defined, as this depends on the desired motion and the parameters of the robot. It is thus of great importance to have a method that determines the switching sequence as well as the controls. We show how one can rely on the AGHF to solve this problem here.

Extant methods in the literature tend to rely on the very computationally intensive mixed-integer programming solvers. In short, these approaches are as follows: we start by choosing a *contact model* which describes the state transition at the time of contact, and then associate it with some auxiliary variables or cost term to activate or penalize the contact. The planning algorithm finds the best contact schedule by solving for the auxiliary variables and minimizing the auxiliary cost term. By using integer decision variable to indicate the transition of stance/flight, mixed-integer programming is used to find the contact schedule and footholds [54, 55, 51]. Soft contact model is used in [48] which approximate the hard contact as spring-damper system. The cost is formulated into a quadratic form in terms of the error between the planned trajectory and a reference trajectory. Motions with different contact schedules can be obtained by predefining different reference trajectories. In [47, 50], the contact is modeled as linear complimentary conditions between the foot height and contact force. The resulting motion planner is able to find the contact sequence and timing all at once. In a more recent work [52], the robot's motion is parameterized by the time duration of each foot's stance/flight phase. The problem is then converted to a NLP problem by direct collocation and the optimal time duration can be solved.

In this section, we extend the AGHF motion planning framework discussed in Sec. 5.1 to handle indefinite switching times. The automatic planning of contact schedule is an example of planning indefinite switching times. Similar to the extant methods, auxiliary states are introduced to represent the switching behavior. However, we use a *continuous* state instead of integer, and create virtual dynamics for the auxiliary state such that the continues

trajectory of the auxiliary states can be mapped to a contact sequence. Thus, the contact sequence can be implicitly planned by the AGHF method. The AGHF method is able to deform an arbitrary auxiliary state trajectory which can corresponds to an infeasible contact schedule, to a trajectory that corresponds to a feasible trajectory.

Following Sec. 5.3.1, the contact constraints are activated/deactivated by the activation function (5.22). The contact sequence is predefined. That is, the number of flight/stance phases and the switching times are predefined as in (5.23). A intuitive way to solve for the unknown switching time is to define a new state to represent the switching time. For example, in one leg jumping task, if there is only one jump from stance to flight phase, we can define x_{sw} to represent the only switching time from stance to flight of the leg, so that the stance phase is $t \in [0, x_{sw}]$ and flight phase is $t \in (x_{sw}, T]$.

The dynamics of x_{sw} is

$$\dot{x}_{sw} = 0$$

with constraint $0 \leq x_{sw} \leq T$. The AGHF algorithm will solve for $x_{sw}(t)$ with augmented dynamics $\dot{x}_{sw} = v$ and minimize the virtual control v, using the *free boundary condition* for x_{sw}.

Representing *each* switching time by *one* single constant variable is rather too expensive. The state x_{sw} can only represent the time for a single switch. If the system switches multiple times, each of the switching times is associated with a new state variable similar to x_{sw}. Therefore, the number of new states grows linearly with the number of switching times.

To circumvent this issue, we rely on the "shape" of the activation function signal to determine the switching times, which results in a single variable potentially encoding an infinite number of switching times. For example, the times of discontinuity of a binary-valued function could represent the switching times. We elaborate on such approaches in the following section.

5.5.1 Switching Time as a State variable

We now show how one can use the AGHF to plan the switching times. In the case of the switched system (5.4), one mode being activated recurrently means that the corresponding activation function is a square wave alternating between 0 and 1 with nonuniform frequency. The rising and falling edges

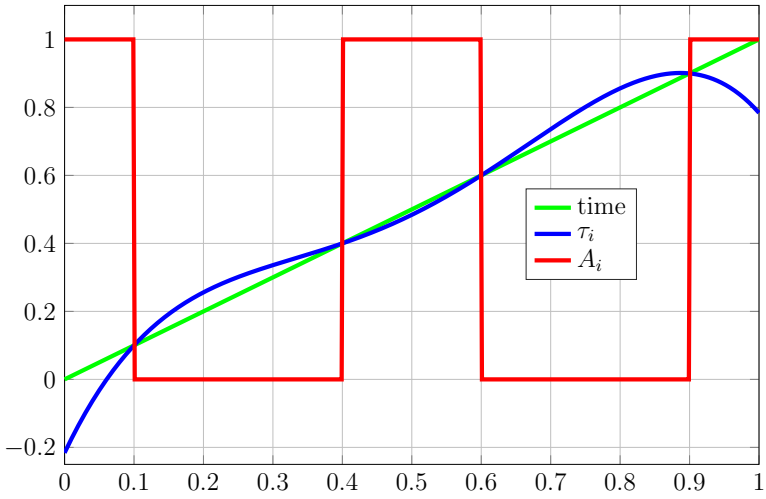


Figure 5.5: Example of τ_i and A_i

of the square wave determines the switching times. To plan the switching times, we can plan the location of the edges. To this end, we let τ_i to be the switching time of the i-th mode. When $\tau_i(t)$ is constant, enforced via the evolution equation $\dot{\tau}_i = 0$, τ_i represent a single switching time. However, if we relax the dynamics and introduce a control a_i to manipulate the trajectory of $\tau_i(t)$, τ_i can be used as multiple switching times, as analyzed below.

The dynamics of τ_i is now:

$$\dot{\tau}_i = a_i$$

The activation function (5.22) is now a function of time and state τ_i and can be defined similarly as (5.23) with Heaviside step function:

$$A_i(t, \tau_i(t)) := H(t - \tau_i(t)) \tag{5.35}$$

A example trajectory of $\tau_i(t)$ is shown in Fig. 5.5. The green line is the actual time, t, which is a straight line connecting $(0, 0)$ and $(1, 1)$, in this example, $T = 1$. The blue line is trajectory $\tau_i(t)$ and the red line is the resulting activation function A_i. When $\tau_i(t) < t$, $A_i = 1$, the switching time is ahead of real time, the mode stays active. When $\tau_i(t) \geq t$, $A_i = 0$, the switching time is behind real time, the mode is inactive. And when $\tau_i(t)$ intersects t, the switching happens. In addition, the boundary value of $\tau_i(t)$ at $t = 0$ and $t = T$ determines if the mode is active at initial and final time.

This formulation is intended to solve for both the timing of the switch and also the number of switches, because both quantities can be determined by

the trajectory of τ_i. However on the implementation side, AGHF method based on this formulation usually fails to find a feasible solution. Because there's no constraint on how many steps the motion has, the algorithm tends to find a $\tau_i(t)$ trajectory oscillating around the true time t with high frequency, which results in a motion with too many switches.

5.5.2 Temporal Scaling of Activation Function

To avoid solutions relying on high frequency switching of the dynamics, we need to constrain the number of switches or directly specify their times. Since for many of the applications we are concerned with, the total number of switches, or at least an upper bound on this number, is known (e.g, the number of steps for legged robot motion), we only deal here with motions with a *finite number* of switches. To this end, we propose the following approach, which goes via the *temporal scaling* of activation functions.

We carry over the idea that $\tau_i(t)$ is a trajectory of a new auxiliary state and it determines the switching times. We now describe a mechanism to apply additional constraints on τ_i so as to control the switching behavior, i.e. have the the ability to move the switching times. The key component of this mechanism is a 2-layer activation function for mode i:

Definition 5.9 (Activation Function with Temporal Scaling)**.** *The 2-layer temporal scaling activation function is*

$$A_i(\tau_i(t)) := H(\varphi(\tau_i(t))) \tag{5.36}$$

where H is the Heaviside unit step function and $\varphi : \mathbb{R} \to \mathbb{R}$ is a periodic function *with image containing 0 in its interior. The dynamics of τ_i follows:*

$$\begin{aligned} \dot{\tau}_i &= \psi(a_i) \\ \dot{a}_i &= w_i \end{aligned} \tag{5.37}$$

where ψ is a non-negative function and w_i a control.

Proposition 5.1. *The activation function $A_i(t)$ in* (5.36) *is a square wave with finite number of rising and falling edges, if the boundary values $\tau_i(0)$ and $\tau_i(T)$ are finite. The number of edges is determined by the period of the function $\varphi(\cdot)$ and the values of $\tau_i(0)$ and $\tau_i(T)$.*

Proof. The proof relies on the **periodicity property** of $\varphi(\cdot)$ and the **non-decreasing property** of $\tau_i(t)$.

The state $\tau_i(t)$ is non-decreasing along t since the time derivative of τ_i is non-negative, by construction (5.37). If the boundary values, $\tau_i(0)$ and $\tau_i(T)$ are finite, and $\tau_i(0) < \tau_i(T)$, and the period of the function $\varphi(x)$ is T_φ, with k intersections across x axis, then the total number of intersection of $\varphi(\tau_i(t))$ with t axis, denoted by K is within range

$$\left\lfloor \frac{\tau_i(T) - \tau_i(0)}{T_\varphi} \right\rfloor k \leq K \leq (\left\lfloor \frac{\tau_i(T) - \tau_i(0)}{T_\varphi} \right\rfloor + 1)k \tag{5.38}$$

where $\lfloor \cdot \rfloor$ is the floor operation (i.e., it returns the largest integer smaller than its argument).

Each of the zero crossing of $\varphi(\tau_i(t))$ generate a 0 to 1 or 1 to 0 transition in the Heaviside unit step function $H(\cdot)$, or equivalently, the edges of the square wave. Therefore, the number of edges of $A_i(t)$ is K given by (5.38). □

Some examples for the parameters are $\varphi(x) = \cos(x)$ and $\psi(x) = x^2$. The two layer together, namely, $H(\varphi(\tau_i))$ generates a square wave activation function as a function of τ_i, which changes values when τ_i crosses zero. The switching times do exist since φ contains zero in the interior of its image. In the particular case $\varphi(\tau_i) = \cos(\tau_i)$, the switching times are $\tau_i = -\pi/2 + k\pi, k \in \mathcal{Z}$.

The state τ_i can be thought as a virtual time that lapses in a *nonuniform* rate. The virtual time τ_i evolves according to Eq. (5.37). From the fact that ψ is non-negative, $\tau_i(t)$ is monotonically non-decreasing as a function of t. The value of $\tau_i(t)$ determines if the value of $\cos(\tau_i)$ is positive or negative, and implicitly determines the binary value of $H(\cos(\tau_i))$. It is important to note that even though τ_i does not decrease, because φ is periodic, one can always reach a switching time moving forward. The evolution of a_i is determined by the *control variable* w_i.

Lemma 5.2. *The number of switches for mode i is determined by the boundary conditions $\tau_i(0)$ and $\tau_i(T)$.*

Proof. The lemma can be directly implied by the proof of Prop. 5.1. The number of square wave edges in activation function (5.36) is determined by $\tau_i(0)$ and $\tau_i(T)$ and each of the edge generates a switch in mode i, therefore the number of switches for mode i is determined by the boundary conditions $\tau_i(0)$ and $\tau_i(T)$. □

With the temporal scaling method proposed above, we can replace the activation function A_i in the original dynamics (5.4) with the newly defined, state-dependent activation functions (5.36), with the dynamics for each pair of new states $[\tau_i, a_i]^\top$, which we terms as *activation states and* activation system:

$$\begin{bmatrix} \dot{\tau}_i \\ \dot{a}_i \end{bmatrix} = \begin{bmatrix} \psi(a_i) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w_i \tag{5.39}$$

which is control affine system with drift as (3.3). Due the the simple structure of the admissible control direction $[0, 1]^\top$, the full rank matrix $\bar{F}$ is simply identity matrix $\bar{F} = I_2$ and the metric for the activation state curve is therefore

$$G = \begin{bmatrix} \lambda & 0 \\ 0 & 1 \end{bmatrix}$$

which penalize the corresponding curve length if τ_i is not evolving at the rate given by $\psi(a_i)$, and the control w_i is free.

Each of the activation system (5.39) is decoupled from the original system (5.4), but the original system is coupled with the activation systems. As a result, the AGHF method is able to plan a motion such that both original and the activation dynamics are satisfied, namely, the method finds a proper switching schedule and the resulting motion of the original system. It is important to note that the formulation (5.36) is not specific to the activation function in the general system (5.4), but a general method to handle switching of binary signal. Therefore it can be also applied to the activation functions for legged locomotion (5.22).

5.5.3 Application to Legged Locomotion

We now apply the formulation for temporal scaling of activation function to legged locomotion planning. As mentioned earlier, we choose $\varphi(x) = \cos(x)$ and $\psi(x) = x^2$ for each of the activation function A_i for legged locomotion, and replace the original one defined in (5.22). The switching times for the virtual time τ_i are $\tau_i = -\pi/2 + k\pi, k \in \mathcal{Z}$.

By this definition of activation dynamics, the initial and final value of τ_i determines the number of stance/flight phases. For example, $\tau_i(0) = 0$ and $\tau_i(T) = 6\pi$ represents a motion with 3 consecutive stance/flight phases,

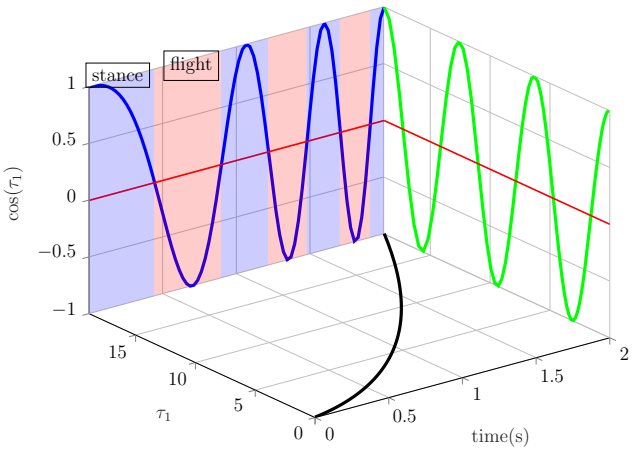


Figure 5.6: Example of $\tau(t)$ and activation function

starting with a stance phase and terminating with an extra stance phase. As time t increases, τ_i will reach $\tau_i(T)$ monotonically. The monotonicity ensures that the motion will have exact number of predefined stance/flight phases. However, the changing rate of τ_i is free, and the algorithm is able to scale the speed of τ_i to move the switching times in the activation function.

An intuitive example is show in Fig. 5.6. The 3D figure shows the relation between the layers of the activation function for leg 1. The 3 axis represent $\cos(\tau_1)$, τ_1 and t respectively. The $\cos(\tau_1)$ vs τ_1 plot is a $\cos(\cdot)$ trajectory that has uniform period on τ_1, as shown on the right plane. For demonstration, $\tau_1(t)$ is chosen to be a quadratic trajectory on t, as shown on the bottom plane. As a result, this $\tau_1(t)$ generate a temporally scaled $\cos(\cdot)$ shaped signal whose frequency is increasing on t, as shown on the left plane. The resulting stance/flight phase schedule is indicated by the blue/red background, where the switching happens more and more frequently.

With the activation dynamics introduced, the AGHF is able to deform an initial guess of $\tau_i(t, 0)$ to a steady state solution $\tau_i(t, \infty)$, which corresponds to a switching schedule with feasible solution of the legged locomotion. It should be noted that the initial guess of $\tau_i(t, 0)$ can be arbitrary and does not need to corresponds to a feasible switching schedule. An example for 2-leg locomotion is shown in Fig. 5.7. The figure directly shows the $\cos(\tau_i(t))$ for both legs. Initial guesses $\tau_i(t, 0)$ are both quadratic but with different slope profiles, the resulting $\cos(\tau_i(t, 0))$ plots are shown by red lines. The AGHF deforms $\tau_i(t, s)$ to a steady state solution $\tau_i(t, \infty)$. The black lines show the

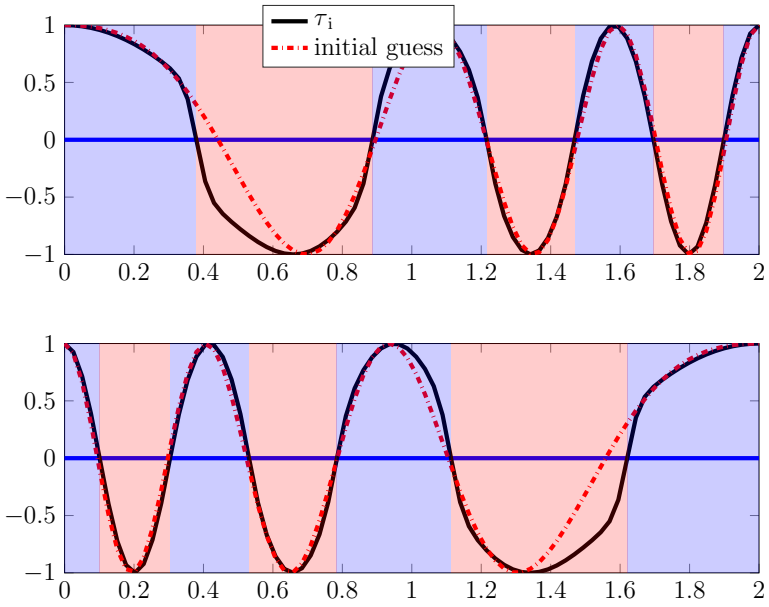


Figure 5.7: Deformation of the activation function $cos(\tau(t))$ layer

plot for $\cos(\tau_i(t, \infty))$. By definition of the activation function, the planned stance/flight timing windows are denoted by the blue/red background.

5.5.4 Temporal Scaling and Determination of Switching Times

Applying directing the approach of Sec. 5.5.2 to legged locomotion requires one extra state per foot. To further decrease the number of states needed for finding the variable switching times, an approach similar as [56] is proposed below. We will solve the motion planning problem for $t \in [0, 1]$.

With slight abuse of notation, we introduce a new scalar state $\tau \in \mathbb{R}$ to the system, which is the **true time** variable that starts from $\tau(0) = 0$ and $\tau(1) = T$ for a given T. τ should be strictly increasing so that the inverse function τ^{-1} exists and we can recover the control as a function of the true time from $u(\cdot)$ by $u^\dagger(t) = u(\tau^{-1}(t))$. For smooth $\tau(\cdot)$, this monotonicity constraint can be enforced by defining $\dot{\tau}(t) = a(t)^2$, $\dot{a}(t) = u_0(t)$ where u_0 is the additional input to the twice-augmented system. Notice that since τ is the true time, $\frac{dx}{d\tau}$ should obey the true system dynamics (3.3) instead of $\frac{dx}{dt}$. Thus using chain rule, we have $\dot{x} := \frac{dx}{dt} = \frac{dx}{d\tau}\frac{d\tau}{dt} = F_d(x)a^2 + F(x)a^2u$. In summary with the augmented states for state constraints in Sec. 5.3.3, denote

the augmented state

$$\hat{x} = \begin{bmatrix} x \\ \tau \\ a \\ \zeta_j \end{bmatrix}, \tag{5.40}$$

where ζ_j is the augmented state for state constraint j as before. we have

$$\dot{\hat{x}} = \begin{bmatrix} \dot{x} \\ \dot{\tau} \\ \dot{a} \\ \dot{\zeta}_j \end{bmatrix} = \underbrace{\begin{bmatrix} h(x)a^2 \\ a^2 \\ 0 \\ 0 \end{bmatrix}}_{\hat{F}_d} + \underbrace{\begin{bmatrix} F(x)a & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\hat{F}} \begin{bmatrix} au \\ u_0 \\ h_j(x)S_j(t,x) \end{bmatrix} \tag{5.41}$$

The corresponding un-admissible directions $\hat{F}_c$ is:

$$\hat{F}_c = \begin{bmatrix} F_c(x) & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{5.42}$$

where $[F_c, F] = I_{4k+6}$ as before. Rearrange and the augmented $\hat{\bar{F}}$ is:

$$\hat{\bar{F}} = \begin{bmatrix} F_c(x) & F(x)a & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.43}$$

The augmented metric penalty matrix $\hat{D}$ is now:

$$\hat{D} = \begin{bmatrix} D & 0 & 0 & 0 \\ 0 & \lambda & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \lambda_j \end{bmatrix} \tag{5.44}$$

where D is defined for original system in (5.24), and the augmented metric $\hat{G} = (\hat{\bar{F}}^{-1})^\top \hat{D} \hat{\bar{F}}^{-1}$.

The key is to predefine the fixed schedule and timing for each legs, in the

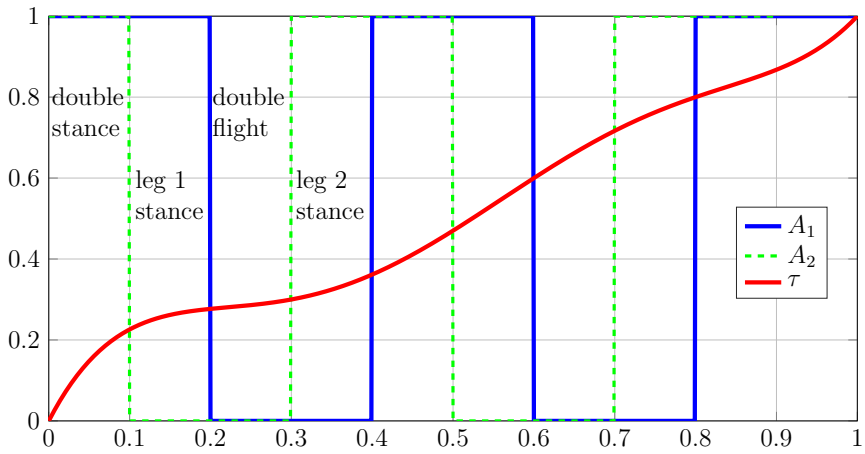


Figure 5.8: Example of $\tau(t)$ and activation function for two legs

"fake" time t, and solve for the motion $x(t)$ and control $u(t)$ then recover the control and state trajectory $u(\tau)$ and $x(\tau)$ in true time. The method relies on the fact that, if a constraint is active in at time t, it's also active at true time $\tau(t)$. The same reasoning holds for the case when the constraint is not active.

Fig. 5.8 shows an example for this method, with 2 leg robot and $T = 1$. First we define the activation function A_1 (blue) and A_2 (green) for the 2 legs. Here the timing of stance/flight phases are equally spaced (increment of 0.1)for both legs, with a delay for the first leg, so that we have 4 phases: double stance, leg 1 stance, double flight and leg 2 stance phase. So for, everything is defined in the coordinate t.

For illustration, we let trajectory $\tau(t)$ (red) to be the AGHF solution. By setup, the switch between the 4 phases happens every 0.1 second in the "fake" time t, therefore the switching times in true time $\tau(t)$ are $\tau(0.1)$, $\tau(0.2)$, $\tau(0.3)$...which are implicitly solved by the AGHF algorithm.

The drawback of the approach is that the timing is scaled at once for all legs, losing the flexibility to adjust the timing for each leg individually. This is not practically efficient, in the case of biped, one has to predefine all the "left foot in stance", "right foot in stance", "both in stance" and "both in flight" sequence. For robot with more than 2 legs, predefining the sequence is much harder due to the lack of intuition.

CHAPTER 6

PLANNING ON $SO(3)$ AND SOFT CONTINUUM ARM ESTIMATION

The motion planning problem we have considered so far involved the Euclidean space for workspace configuration space. We now generalize our approach to include configuration spaces that include the Lie groups $SO(3)$ and $SE(3)$ of rotations and Euclidean motions in 3-space, respectively. Recall that

Definition 6.1. *The Lie group $SO(3)$ is*

$$SO(3) = \left\{ R \in \mathbb{R}^{3\times 3} \middle| \det(R) = 1, R^\top R = RR^\top = I_3 \right\} \tag{6.1}$$

where I_3 is the 3×3 identity matrix.

Definition 6.2. *The Lie group $SE(3)$ is*

$$SE(3) = \left\{ A \middle| A = \begin{bmatrix} R & r \\ O_{1\times 3} & 1 \end{bmatrix}, R \in SO(3), r \in \mathbb{R}^3 \right\} \tag{6.2}$$

where $O_{m\times n}$ is the $m \times n$ zero matrix.

Hence, we aim to address the following problem, which we state using $SO(3)$, but can be stated similarly for $SE(3)$:

Trajectory planning problem in SO(3): Let $T > 0$, and X_0 and X_T be two elements in $SO(3)$. We consider the following planning task: find a potentially constrained curve $X : [0, T] \to SO(3)$ that connects X_0 and X_T in motion duration T, namely, $X(0) = X_0$, $X(T) = X_T$. The task also allows for free boundary conditions, in which case $X(0)$ and $X(T)$ are not specified. The dynamical constraints are

$$\dot{X}(t) = F(X(t), \omega(t)) \tag{6.3}$$

where $F(X, u) \in T_X SO(3)$ and $\omega(t) = [\omega_x(t), \omega_y(t), \omega_z(t)]^\top \in \mathbb{R}^3$ is the angular velocity in the chosen frame and serves as the control.

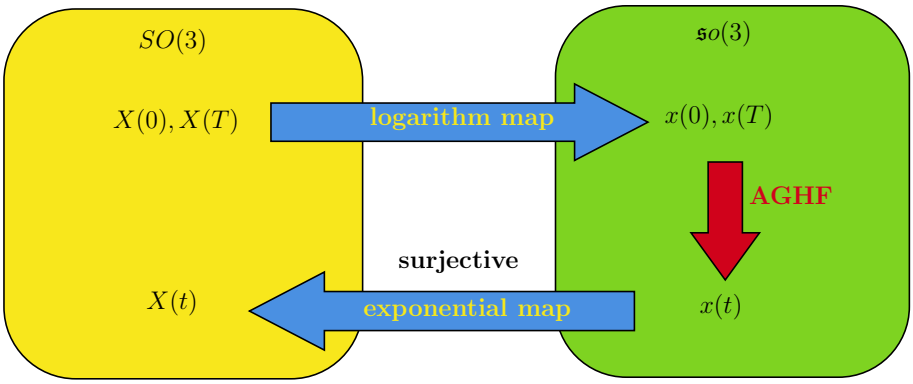


Figure 6.1: Workflow for planning with exponential map

Hence, the goal of the trajectory planning problem is to generate an admissible curve (where admissible is defined from using parameters from the task at hand) valued in the Lie group $SO(3)$. One can describe such a curve either intrinsically, by choosing an appropriate parametrization of $SO(3)$, or extrinsically, e.g., plan motion in Euclidean space $\mathbb{R}^n$ (with $n = 9$ for $SO(3)$) and enforce constraints that make the resulting motion a curve in $SO(3)$; in other words, embed $SO(3)$ in Euclidean space and use the AGHF in Euclidean space with constraints defining $SO(3)$.

The first approach can be implemented using, for example, the exponential map $\exp : \mathfrak{so}(3) \to SO(3)$, which is known to be *surjective* [57]. The work flow is shown in 6.1.Hence, using matrix logarithms, one could translate the requirements of the planning task (i.e., initial and final states, constraints, etc) from $SO(3)$ to $\mathfrak{so}(3)$, plan the motion in $\mathfrak{so}(3) \simeq \mathbb{R}^3$ using the AGHF, and use the matrix exponential to translate the result into a motion in $SO(3)$. Since $\mathfrak{so}(3)$ is a vector space, the theory developed in the previous chapters applies straightforwardly.

The main issue with the approach depicted in Fig. 6.1 lies in the fact that the exponential map is *not* injective. While there is a good understanding of how to determine a subset $D \subset \mathfrak{so}(3)$ so that $\exp(D) = SO(3)$, the logarithm map $\log : SO(3) \to D$ is necessarily *discontinuous*. This is a consequence of the fact that the topology of D (which is trivial) and that of $SO(3)$ are fundamentally different. Furthermore, the set D is represented by linear inequalities, which also add a layer of complexity to this approach.

We use here an approach that could be considered as a hybrid of the completely intrinsic method, which runs into the issues explained above, and

a completely extrinsic approach, which is computationally more demanding as it introduces extra state variables *and* constraints to make up for the lack of adequate parameterization of the state-space. The method we rely on is based on the quaternion representation of $SO(3)$ [58].

Before we elaborate the details of the method, it is necessary to introduce some notations and basic operations.

Notations: The operator $\widehat{\cdot}$ maps a vector $x = [x_1, x_2, x_3]^\top \in \mathbb{R}^3$ to the skew-symmetric matrix in $\mathfrak{so}(3)$:

$$\widehat{x} = \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix} \tag{6.4}$$

The operator $\mathrm{vec}(\cdot)$ is the matrix vectorization that reshapes a matrix $A = [a_1, a_2, \cdots, a_m] \in \mathbb{R}^{n\times m}$ with $a_i \in \mathbb{R}^n$ representing the i-th column, to one column vector by concatenating all columns:

$$\mathrm{vec}(A) = [a_1^\top, a_2^\top, \cdots, a_m^\top]^\top \tag{6.5}$$

Let $q = [q_w, q_x, q_y, q_z]^\top \in \mathbb{R}^4$ be a 4D unit vector representing a so-called *unit quaternion*. We can associate to it a rotation matrix $R = [r_1, r_2, r_3] \in SO(3)$, with r_i representing the columns of R. The mapping from q to rotation matrix R is as follow:

$$\begin{aligned} R(q) &= \begin{bmatrix} r_1(q) & r_2(q) & r_3(q) \end{bmatrix} \\ &= \begin{bmatrix} 2(q_w^2 + q_x^2) - 1 & 2(q_x q_y - q_w q_z) & 2(q_x q_z + q_w q_y) \\ 2(q_x q_y + q_w q_z) & 2(q_w^2 + q_y^2) - 1 & 2(q_y q_z - q_w q_x) \\ 2(q_x q_z - q_w q_y) & 2(q_y q_z + q_w q_x) & 2(q_w^2 + q_z^2) - 1 \end{bmatrix} \end{aligned} \tag{6.6}$$

Reciprocally, to each rotation matrix, we can associate two possible quaternions: given the rotation matrix

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{6.7}$$

We can set :

$$
\begin{aligned}
q_w &= \pm\sqrt{1 + r_{11} + r_{22} + r_{33}}/2 \\
q_x &= (r_{32} - r_{23})/4q_w \\
q_y &= (r_{13} - r_{31})/4q_w \\
q_z &= (r_{21} - r_{12})/4q_w.
\end{aligned} \tag{6.8}
$$

Note that this conversion is only accurate under some certain conditions of the trace of R. For more details of the conversion, we refer the reader to [59].

In the conversion (6.8), the computation of component q_w involves a square root operation, which gives two solutions of q_w, as indicated. Indeed, each rotation matrix can be converted to two distinct unit quaternions; in fact, the quaternions are a *double cover* of $SO(3)$.

6.1 Representing elements of $SO(3)$ and its Tangent Space

The usual representations of rotations are via rotation matrices, Euler angles, or unit quaternions. The Euler angles are a seemingly a good approach to do this, but suffer from the presence of singularities and the Gimbal lock problem [60]. Rotation matrices are a faithful (i.e., injective and surjective) representation of 3D orientations. The transformation operation (e.g., action of a rotation on a point or rigid object) using rotation matrix is easily implemented by a choice of right- or left-multiplication. However, rotation matrices require 9 state variables for their representation, and 6 constraints.

The unit quaternions also have provide a surjective representation of orientations, but they require only 4 state variables for their representation. The rotation operations using quaternions are easily implemented as well. This representation is, however, not injective although the level of redundancy introduced is more manageable than in the case of the exponential map and logarithm. This lack of injectivity might be a critical issue for some motion planning tasks. In this work, both the rotation matrix and quaternion representations are studied. For simplicity, we say "quaternion" refers to unit quaternion, unless otherwise stated.

6.1.1 Representation via Quaternions

For rotational dynamics, the equations of motion are usually written in terms of angular velocity and angular acceleration. Denote the angular velocity by $\omega \in \mathbb{R}^3$, expressed in the body fixed frame.

First, we use quaternions to represent the orientation state (also referred to as **pose**) $X(t) = q(t)$, and let $q(t) = [q_w(t), q_x(t), q_y(t), q_z(t)]^\top$ be the real vector representation of unit quaternion trajectory. Let the quaternion form of angular velocity be $\vec{\omega} = [0, \omega^\top]^\top$, the relation between the time derivative of unit quaternion and angular velocity is concise:

$$\dot{q} = \frac{1}{2} q \otimes \vec{\omega} \tag{6.9}$$

where $\otimes$ is quaternion multiplication operator. This can be equivalently expressed as:

$$\dot{q} = F_q(q)\omega \tag{6.10}$$

where $F_q \in \mathbb{R}^{4\times 3}$:

$$F_q = \frac{1}{2}\begin{bmatrix} -q_x & -q_y & -q_z \\ q_w & -q_z & q_y \\ q_z & q_w & -q_x \\ -q_y & q_x & q_w \end{bmatrix} \tag{6.11}$$

It can be observed from (6.11) that the columns of F_q are orthogonal for **arbitrary** $q \in \mathbb{R}^4$ (including quaternions that are not unit length).

6.1.2 Representation via Rotation Matrices

Now we use rotation matrix as the orientation state, $X(t) = R(t)$, and let $R(t) = [r_1(t), r_2(t), r_3(t)] \in SO(3)$ be the rotation matrix trajectory, with $r_i(t)$ representing the columns of $R(t)$. The time derivative of R is:

$$\dot{R} = R\widehat{\omega} \tag{6.12}$$

where $\widehat{\cdot}$ the usual skew symmetric operator defined in (6.4). Eq. (6.12) can be vectorized:

$$\mathrm{vec}(\dot{R}) = F_R(R)\omega \tag{6.13}$$

where $F_R \in \mathbb{R}^{9\times 3}$ is given by

$$F_R = \begin{bmatrix} O_{3\times 1} & -r_3 & r_2 \\ r_3 & O_{3\times 1} & -r_1 \\ -r_2 & r_1 & O_{3\times 1} \end{bmatrix} \tag{6.14}$$

and vec$(\cdot)$ is the matrix vectorization operator defined in (6.5).

The matrix R being a rotation matrix implies that r_1, r_2 and r_3 are orthogonal. This further implies that the columns of F_R are orthogonal. Note that the orthogonality of the columns of F_R relies on R being an element in $SO(3)$. When formulating the AGHF, one needs to augment the matrix F_q (6.11) or F_R (6.14) to a full rank square matrix $\bar{F}$, which involves finding the orthogonal basis of the complementary spaces to the column spans of F_q and F_R. While F_q needs to be augmented with one more column, F_R needs to be augmented by 6 additional columns, which is computationally inefficient. This is a clearly a drawback compared with the quaternion. More details of discussion is addressed in Sec. 6.2.2.

6.2 Riemannian Metric and AGHF for $SO(3)$

To construct the Riemannian metric, we need to find the inadmissible directions or constrained directions F_c for the tangent vector of the (actuated) curve. The columns of F_c span the inadmissible directions and generally be constructed via Gram-Schmidt procedure using the F from the original system. Here, we recall that the inadmissible directions we are referring to are the ones that violate the constraints on $R(t) \in SO(3)$ or $q(t)$ being a unit quaternion. Specifically, the direction of the tangent matrix $\dot{R}(t)$ is inadmissible if $R(t)+\dot{R}(t)\delta t \notin SO(3)$ for infinitely small δt. And the direction of $\dot{q}(t)$ is inadmissible if $\|q(t)+\dot{q}(t)\delta t\| \neq 1$, for infinitely small δt. There could be additional constraints due to the planning task.

In the case where the angular velocity ω is the control (e.g. kinematics unicycle), the system is driftless and we can find the inadmissible directions directly based on F_q or F_R.

6.2.1 Driftless Case with Quaternions

For the case of represention of the orientation via quaternions, since F_q has orthogonal columns, it is straightforward to construct the matrix $\bar{F}$: set

$$\bar{F}(q) := \Big(F_c(q)|F_q(q)\Big) \in \mathbb{R}^{4\times 4} \tag{6.15}$$

where

$$F_c(q) = \frac{1}{2}\begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}^\top = \frac{1}{2}q. \tag{6.16}$$

With F_q as in Eq. (6.11), it is easy to verify that $\bar{F}(q)$ is orthogonal for arbitrary $[q_w, q_x, q_y, q_z]^\top \in \mathbb{R}^4$ with $\|q\| \neq 0$.

The energy of a curve in the space of unit quaternions is as usual given by

$$\mathcal{A}(q(\cdot)) := \int_0^T \frac{1}{2}\dot{q}^\top G(q)\dot{q}dt. \tag{6.17}$$

which of course shares the same minimizer as the curve length defined by (3.5) without the drift term F_d.

The Riemannian metric G can be constructed the same way as before:

$$G(q) := (\bar{F}(q)^{-1})^\top D\bar{F}(q)^{-1} \tag{6.18}$$

for an appropriately defined matrix D. For example, we take $D := \mathrm{diag}([\lambda, 1, 1, 1])$ when there are no constraints on the orientation; for curves that are constrained to have no curvature around the x-axis, one can use $D := \mathrm{diag}([\lambda, \lambda, 1, 1])$.

The steady state solution of GHF is a unit quaternion trajectory, which can be translated to the orientation of the rigid body. The system dynamics (6.10) guarantees that the magnitude of $q(t)$ is constant, therefore the initial value $q(0)$ has to be unit length. The final value $q(T)$, if specified, also has to be unit length. As mentioned earlier, the quaternion representation is not faithful in that different quaternions (in fact, two of them) can represent the *same* orientation. This double cover of orientations by quaternions requires special attention when defining the initial and final state. For example, if the initial state is q_0 and the final state is the conjugate of q_0, the planned motion will perform a rotation by a full circle, even though the initial and final state represent the same orientation, and one would thus expect the

planning algorithm to yield a constant curve.

6.2.2 Driftless Case with Rotation Matrices

We now deal with the case where the orientation is given by a rotation matrix R. The vectorized state $\mathrm{vec}(R)$ is in $\mathbb{R}^9$.

$$\bar{F}(R) := \Big(F_c(R)|F_R(R)\Big) \in \mathbb{R}^{9\times 9}, \tag{6.19}$$

where the columns of $F_c(R)$ span the constrained directions. and there is no constraint that the matrix R is a valid rotation matrix. Therefore, the orthogonality F_R does not hold in general, as discussed in 6.1.2.

Lemma 6.1. *Let $R \in SO(n)$, denote by $T_RSO(n)$ the tangent space of $SO(n)$ at R and by $\mathrm{vec}(T_RSO(n)) \subset \mathbb{R}^{n^2}$ its vectorization described according to Eq. (6.5). The orthogonal vector space to $\mathrm{vec}(T_RSO(n))$ in $\mathbb{R}^{n^2}$ for the Euclidean metric is given by $\mathrm{vec}(RSym) \subset \mathbb{R}^{n^2}$, where Sym is the vector space of symmetric matrices.*

Proof. The admissible directions are given by the tangent space of $SO(n)$ at R, and it is well known that

$$T_RSO(n) = \{R\Omega \mid \Omega \in \mathfrak{so}(n)\},$$

where $\mathfrak{so}(n)$ is the vector space of $n \times n$ skew symmetric matrix.

We will show that the inadmissible directions are given by:

$$T_RSO(n)^\perp = \{RS \mid S \in \mathbb{R}^n \text{ and } S = S^\top\}$$

which is the orthogonal space of $T_RSO(n)$, and S is symmetric.

Indeed, for the Euclidean inner product, take an arbitrary element in $T_RSO(n)$, $R\Omega$, and an arbitrary element in $T_RSO(n)^\perp$, RS. Computing their inner product, we have

$$< R\Omega, RS >= \mathrm{Tr}(\Omega^\top R^\top RS) = -\mathrm{Tr}(\Omega S)$$

with the help of the orthogonality of R.

Recall that $\mathrm{Tr}(AB) = \mathrm{Tr}(B^\top A^\top)$ and $\mathrm{Tr}(ABC) = \mathrm{Tr}(BCA) = \mathrm{Tr}(CAB)$. Hence if $A = -A^\top$ and $B = B^\top$, we get $\mathrm{Tr}(AB) = -\mathrm{Tr}(BA) = -\mathrm{Tr}(AB)$, which leads to $\mathrm{Tr}(AB) = 0$.

Therefore with $\Omega = -\Omega^\top$ and $S = S^\top$, we have $< R\Omega, RS >= -\mathrm{Tr}(\Omega S) = 0$. In addition, we have the following inner product property $< A, B >= \mathrm{vec}(A)^\top \mathrm{vec}(B)$, thus we have

$$\mathrm{vec}(R\Omega)^\top \mathrm{vec}(RS) = 0$$

the two vectors are orthogonal. □

Hence, to determine the 6 columns of F_c, it suffices to introduce the 6 basis for the space of 3×3 symmetric matrices, say $S_1, \ldots S_6$ with

$$S_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad S_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad S_3 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$S_4 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad S_5 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad S_6 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

and then the i-th column of F_c denoted by $F_{c,i}$ is $\mathrm{vec}(RS_i)$.

6.3 Control Extraction and Integrated Path

Once the solution $q(t, s_{max})$ or $R(t, s_{max})$ of the AGHF is obtained for large enough s_{max}, the control ω can be extracted.

In the driftless case, the angular velocity (i.e., the control) can be extracted as follows:

$$\omega(t) := \begin{pmatrix} O_{3\times 1} & I_{3\times 3} \end{pmatrix} \bar{F}(q(t, s_{\max}))^{-1} \dot{q}(t, s_{\max}) \tag{6.20}$$

for quaternions, and

$$\omega(t) := \begin{pmatrix} O_{3\times 6} & I_{3\times 3} \end{pmatrix} \bar{F}(\mathrm{vec}(R(t, s_{\max})))^{-1} \mathrm{vec}(\dot{R}(t, s_{\max})) \tag{6.21}$$

for rotation matrices. Note that the matrix $\bar{F}$ in (6.20) is derived from (6.15)

with (6.16), and $\bar{F}$ in (6.21) is derived from (6.19) if one can find a proper $F_c \in \mathbb{R}^{9\times 6}$.

In case there is a drift term in the dynamics, e.g., in case of a kinodynamic system where the controls are forces and torques, we take the standard approach of letting the angular velocity ω be a state whose evolution is determined by the angular acceleration, which then becomes the control.

The planned motion is given by the integrated path from (6.10) or (6.13) with extracted $\omega(t)$ (or acceleration in case of a second order system). The regular Runge–Kutta integration and interpolation rule does not apply for rotation matrices or unit quaternions since the integrated orientation state can deviate from the constraints stemming from being a rotation matrix or a unit quaternion. Therefore, the integration of the orientation state need special care to ensure that these constraints are met to the best extent possible. One approach to do so it to use *multiplicative* update rules, instead of addition update rules:

For the quaternion, once the angular velocity ω is extracted from (6.20) or integrated from angular acceleration, the integrated path $\hat{q}(t)$ can be obtained via the integration rule:

$$\hat{q}^{i+1} = \exp(\frac{1}{2}\Delta t\vec{\omega}^i) \otimes \hat{q}^i \tag{6.22}$$

where i denote the current integration step and Δt is the integration step size. $\exp(\cdot)$ is quaternion exponential.

For the rotation matrix, once the angular velocity ω is extracted from (6.21) or integrated from angular acceleration, the integrated path $\hat{R}(t)$ can be obtained via the integration rule:

$$\hat{R}^{i+1} = \hat{R}^i \exp(\Delta t\widehat{\omega}^i) \tag{6.23}$$

where i denote the current integration step and Δt is the integration step size. $\exp(\cdot)$ is matrix exponential.

6.4 Planning on $SE(3)$

For the motion of a rigid body in 3D space, the complete configuration space is $SE(3)$, which contains both the orientation $X \in SO(3)$ and the position

$p \in \mathbb{R}^3$. We now elaborate the AGHF formulation for $SE(3)$ planning. For the sake of computational ease, the orientation is represented by quaternion.

Let $p = [p_x, p_y, p_z]^\top \in \mathbb{R}^3$, $q \in \mathbb{R}^4$ be the position and quaternion trajectory, and $v = [v_x, v_y, v_z]^\top \in \mathbb{R}^3$, $\omega \in \mathbb{R}^3$ be the translational and angular velocity measured in body fixed frame, respectively. The time derivative of p and q can be mapped from v and ω by:

$$\dot{p} = R(q)v \tag{6.24}$$

$$\dot{q} = F_q(q)\omega \tag{6.25}$$

where the first equation is a transformation of the translational velocity from body fixed frame to global frame, the second equation is the same as (6.10).

6.4.1 AGHF formulation for $SE(3)$: 3D Unicycle

We illustrate the implementation of our method for planning in $SE(3)$ by using the dynamical 3D unicycle system, also known as a 3D Dubins car. The 3D unicycle is a rigid body that can only move along its body fixed x axis translationally, and steer around its body fixed y,z axis. This system is nonholonomic. In order to constrain the translational velocity v and angular velocity ω, we augment the system such that the state is $x = [p^\top, q^\top, v^\top, \omega^\top]^\top \in \mathbb{R}^{13}$. The set of constraints on v and ω is now a state constraint and can be formulated with the method in Sec. 4.1. The control is now comprised of the translational and angular accelerations.

The control is:

$$u = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{v} \\ \dot{\omega} \end{bmatrix} = \begin{bmatrix} \dot{v}_x \\ \dot{\omega}_y \\ \dot{\omega}_z \end{bmatrix} \in \mathbb{R}^3 \tag{6.26}$$

which is the acceleration along its x axis, $\dot{v}_x$, and angular acceleration about its y and z axis, $\dot{\omega}_y$ and $\dot{\omega}_z$.

The resulting dynamical system is

$$\dot{x} = F_d(x) + F(x)u \tag{6.27}$$

where

$$F_d(x) = \begin{bmatrix} R(q)v \\ F_q(q)\omega \\ O_{6\times 1} \end{bmatrix} \quad F(x) = \begin{bmatrix} O_{7\times 3} \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{bmatrix} \tag{6.28}$$

and $F_q(q)$ is defined in (6.11).

The *planning task* is to solve for a motion $x(t)$ that starts from initial state x_{init} at $t = 0$ and ends at final state x_{fin} at $t = T$, with constraints on v and ω. The key step of the AGHF method is to construct the metric G. We define the augmented control directions (3.16) to be:

$$\bar{F} := \Big[F_c | F\Big] \tag{6.29}$$

where the inadmissible directions F_c is:

$$F_c = \left[\begin{array}{c|c} I_7 & O_{7\times 3} \\ \hline & O_{1\times 3} \\ O_{6\times 7} & I_3 \\ & O_{2\times 3} \end{array}\right] \tag{6.30}$$

so that $\bar{F}$ is invertible. The metric G is defined similarly to (3.17), with D matrix:

$$D = \text{diag}(\underbrace{\lambda, \cdots, \lambda}_{10}, 1, 1, 1) \tag{6.31}$$

where the first 7 λ's on the diagonal of D ensure that $\dot{p}$ and $\dot{q}$ are only steered by the drift term F_d, reflecting the double integrator-like nature, and the remaining 3 λ's penalize the inadmissible accelerations (angular acceleration about body fixed y, z axis, and translational acceleration along body fixed x axis).

In addition, we impose range constraints on the magnitudes of ω_y and ω_z:

$$\begin{aligned} -1 \le \omega_y \le 1 \\ -1 \le \omega_z \le 1; \end{aligned} \tag{6.32}$$

these constraints are enforced using the method described in Sec. 4.1.

Upon deriving the AGHF from the above-defined metric, one can solve the motion planning problem with the following BC and IC and a large s_{max}:

$$\begin{aligned} x(0,s) &= x_{\text{init}} \\ x(T,s) &= x_{\text{fin}} \\ x(t,0) &= x_{guess}(t) \end{aligned} \tag{6.33}$$

6.4.2 Results

A 3D "parallel parking" motion is generated. The 3D unicycle start at position $p(0) = [0,0,0]^\top$ with orientation $q(0) = [0.9659, 0, 0, 0.2588]^\top$ which corresponds to a rotation of $\pi/6$ around body x axis. The goal is to reach $p(T) = [0, 0.2, 0.2]^\top$ with same orientation $p(T) = p(0)$ with $T = 1$. The initial guess $x_{guess}(t)$ is straight line connecting the specified BCs. Note that such straight line satisfies the inequality constraints (6.32). The large constant λ is 5×10^4.

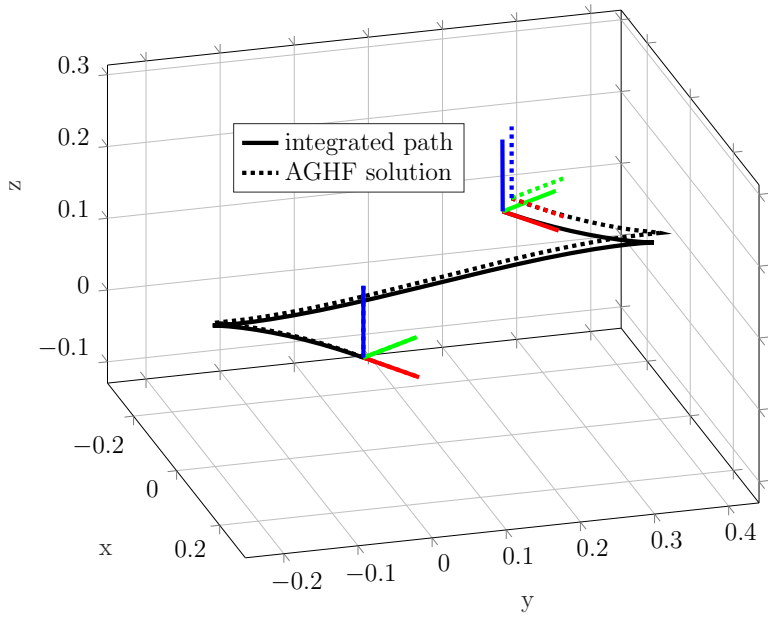
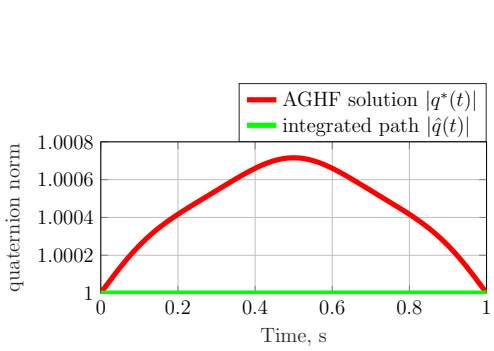


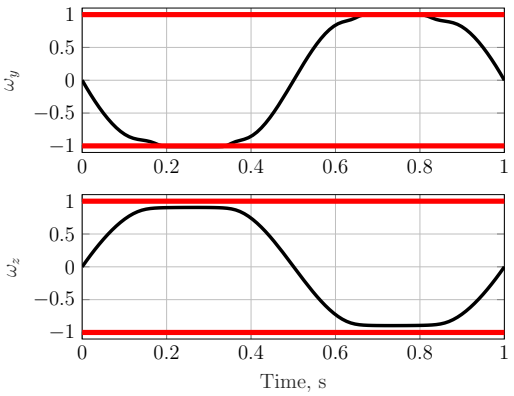
Figure 6.2: 3D unicycle parallel path. The solid line is the integrated path and the dotted line is the AGHF solution. Black lines are xyz path and the RGB frames are body fixed frame at initial and final state.

The planned motion is shown in Fig. 6.2. The black curve is the 3D position

trajectory (p variable) and the RGB (red/green/blue) segment are body fixed frames representing the initial and final orientations of the car. The integrated path, which is the planned motion, is indicated by solid lines. The AGHF solution, $x^*(t)$, is indicated by the dotted lines. Similar as the 2D unicyle parallel parking example in Sec. 3.4, the algorithm finds a "Z-shaped" path. Due to the constraint on ω_y and ω_z (shown in Fig. 6.3b), the unicycle cannot steer too quickly. The planned path tends to use larger translational velocity and thus travel a longer distance to overcome the limited range of angular velocity. Finally, the error between the AGHF solution and the integrated path converges to zero when λ tends to infinity, as discussed in Theorem 3.1.



(a) The norm of quaternion with respect to time.



(b) Admissible angular velocities: ω_y and ω_z. Red lines are the upper and lower limit of the angular velocity

Figure 6.3: Norm of quaternion and constraints on angualr velocity

To conclude this subsection, we look into the numerical errors in the planning process. The norm of the integrated quaternion trajectory $\hat{q}(t)$ has to be 1 for all t in order to represent a valid orientation trajectory; we show the norm of $\hat{q}(t)$ and the AGHF solution $q^*(t)$ are shown in Fig. 6.3a. We observe that $\|q^*(t)\|$ is not preserved and slightly off from 1 for $0 \leq t \leq T$, but the deviation is minimal and would only impact a physical implementation of the obtained control minimally.

6.5 Soft Continuum Arm (SCA) Estimation

We now apply the framework developed in the previous sections to a soft robotic problem. The soft robot we have in mind is depicted in Fig. 6.4. It consists schematically of a tube, with one end called the base, and the other end the end-effector. The connection between the two seemingly different problems is two-fold: first, a soft robot made of a single flexible arm is described by a *curve* in $SE(3)$; recall that a point in $SE(3)$ contains both an element of $SO(3)$ and a vector of $\mathbb{R}^3$. Second, the main principle we rely on for estimation is the one of **least action**: the robot pose we observe should be the one that **minimizes energy**, perhaps under given constraints. Our method, by design, minimizes an energy functional, and is thus applicable here with minimal modifications.

Precisely, the position and orientation of each cross section of the soft robot determines a unique element of $SE(3)$: say $x \in SE(3)$. A diagram of the SCA cross section is shown in Fig. 6.4. Different cross sections are parameterized by the robot length t measured from robot's base to the cross section, $t \in [0, T]$ where T is the soft robot length. The robot pose can then be described by a differentiable curve $x(t) : [0, T] \to SE(3)$. In this section, we thus use t as the 1D independent variable describing the location of each cross section along the robot, and we use ω to denote the curvature of the robot.

The SCA robot is directly actuated by the *internal pressure* along the robot. The internal pressure determines the curvature along the robot, thus can control the robot to form different pose. A typical structure of the SCA is to have multiple extensible tubes adhered together. The robot can twist and bend by properly controlling the internal pressure of each tube. The model we propose is fairly general, and not tied to a particular physical implementation. We refer to [61] for a description of an example of robots covered by our framework. In this section, we directly use the curvature as the control, instead of the internal pressure.

The estimation method proposed in this chapter aims to estimate the pose and the external load, with limited sensor data. The problem has a common general set-up with the 3D unicycle planning problem, with the SCA robot length being the motion duration in motion planning, as we now illustrate. The additional constraint needed, compared to the 3D unicycle

motion planning, is:

The soft robot does not extend or shear. These constraints can easily be encoded in the velocity in (6.24) written in the frame of the cross-section of the robot. By convention, we set $v = [1, 0, 0]^\top$ and assume that the x-axis in the body frame is perpendicular and outward pointing to a cross-section. The perpendicularity of v with the cross section ensures that neighboring cross-sections are not sheared; the fact that v has a constant magnitude ensures that the robot is inextensible.

Due to the fact that v is known and constant, we can obtain the *pose of the robot from the curvature profile* $\omega(t)$. This model is a special case of the Cosserat model [62].

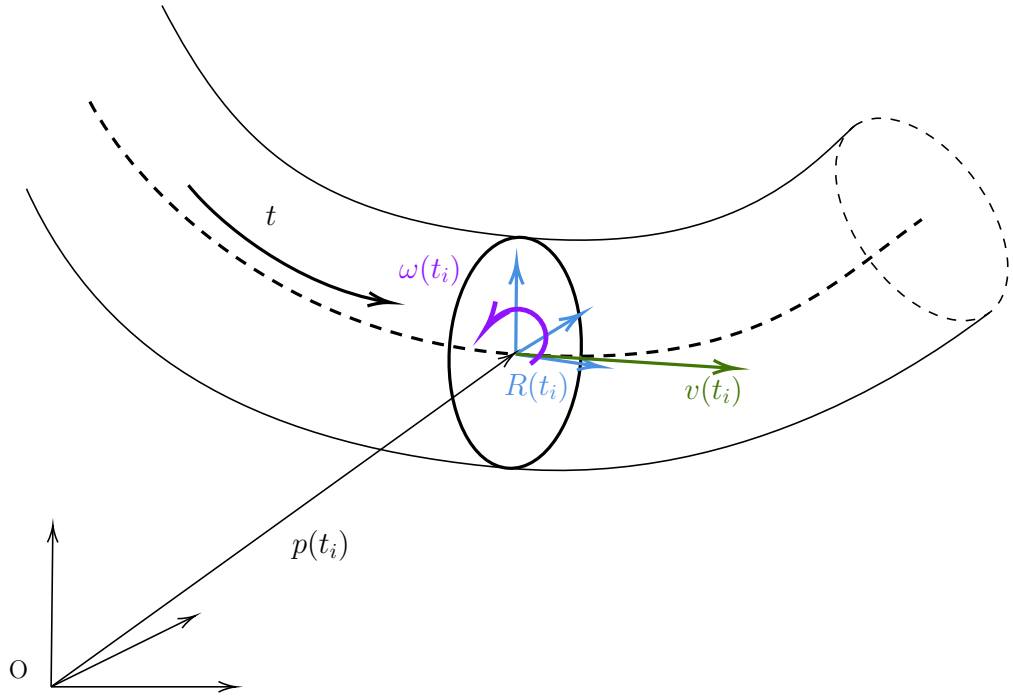


Figure 6.4: SCA diagram for cross section t_i, with position $p(t_i)$, orientation $R(t_i)$, and tangent vector $v(t_i)$.

6.5.1 SCA Forward Model

Without loss of generality, we assume the robot length is 1, so that $t \in [0, 1]$. The pose of the cross section at t, $x(t) = [p^\top(t), q^\top(t)]^\top$, is described by the position $p(t) = [p_x, p_y, p_z]^\top \in \mathbb{R}^3$ and the orientation $q(t)$ of the cross section, where $q(t)$ is a unit quaternion.

In addition, the **resting pose** of the robot is described by a **natural curvature profile** $\omega^\star(t)$. The natural curvature is zero if and only if the natural pose is a straight line. For example, $\omega^\star(t) = [0, 0, \pi]^\top$ gives a circular natural pose around the z-axis. The natural curvature is typically a result of actuated internal pressure. The difference between the natural curvature of the robot and the curvature under load is denoted by $\Delta\omega(t)$. The orientation is determined by the net curvature: $\omega = \Delta\omega + \omega^\star$. Denote $\frac{dp}{dt}$, $\frac{dq}{dt}$ by $\dot{p}$, $\dot{q}$. their evolution along length t can be expressed by (6.24) and (6.25), with $v = [1, 0, 0]^\top$, reflecting the fact that the robot is not stretching or shearing as mentioned above.

6.5.2 SCA Pose Estimation and Affine Geometric Heat Flow

The SCA pose estimation problem is to find the pose curve $x(t)$ of the soft robot, for $0 \leq t \leq T$, given the state at the base $x(0)$ and possibly the state at tip, $x(T)$, and the external load and constraints. On the one hand, owing to the principle of least action, the pose in static equilibrium should be the curve that minimizes energy of the robot. On the other hand, the planning problem in $SE(3)$ introduced in Sec. 6.4 is to find a curve $x(t)$, which is interpreted as a trajectory for a system, of duration T, with given initial and final states $x(0)$ and $x(T)$, and given some motion constraints. Our approach to the problem was to cast a feasible trajectory as a trajectory of minimal length, and it is well known that minimizers of the energy functional are also minimizers of the length functional [28].

Thus, we can use the AGHF to find a pose that minimize the total energy due to deformation since this pose is also the trajectory of a 3D Dubins car with appropriate constraints and boundary conditions. Hence, we arrive at the following conclusion:

The motion planning problem in $SE(3)$ solved in Sec. 6.4 and the SCA estimation problem are equivalent.

We will elaborate below on this point: first, the SCA pose estimation problem is formulated by the AGHF method in Sec. 6.5.3. Then in Prop. 6.1, we show that the energy of actuated curve (3.8) derived from the SCA model coincide with the elastic potential energy of SCA in the estimation problem.

6.5.3 AGHF Formulation for SCA Estimation

Recall that the deformation due to external load is characterized by the curvature change $\Delta\omega = [\Delta\omega_x \Delta\omega_y, \Delta\omega_z]^\top$, which serve as the control and is to be found by the estimation algorithm. Let the state vector be the pose of SCA, $x = [p^\top, q^\top]^\top$, the system dynamics is:

$$\dot{x} = F_d(x) + F(x)\Delta\omega \tag{6.34}$$

where

$$F_d(x) = \begin{bmatrix} R(q) \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^\top \\ F_q(q)\omega^\star \end{bmatrix} \quad F(x) = \begin{bmatrix} O_{3\times 3} \\ F_q(q) \end{bmatrix} \tag{6.35}$$

can be derived from (6.24) and (6.25) with net curvature $\omega = \Delta\omega + \omega^\star$ and $v = [1, 0, 0]^\top$.

The constrained directions can be constructed with help of (6.15) and (6.16):

$$F_c(x) := \begin{bmatrix} I_{3\times 3} & O_{3\times 1} \\ O_{4\times 3} & \frac{1}{2}q \end{bmatrix} \tag{6.36}$$

The resulting $\bar{F}$ is:

$$\bar{F}(x) := \Big(F_c | F_q\Big) = \begin{bmatrix} I_{3\times 3} & O_{3\times 4} \\ O_{4\times 3} & \begin{bmatrix} \frac{1}{2}q & F_q(q) \end{bmatrix} \end{bmatrix} \tag{6.37}$$

which is an orthogonal matrix. The metric is as usual given by:

$$G(x) := (\bar{F}(x)^{-1})^\top D \bar{F}(x)^{-1} \tag{6.38}$$

with the penalty matrix $D := \mathrm{diag}([\lambda, \lambda, \lambda, \lambda, b_x, b_y, b_z])$, where b_x,b_y and b_z are twisting and bending stiffness, and λ is the large constant. The matrix D can be also called stiffness matrix in this section, because the constant λ can be interpreted as virtual stiffness in the direction of the inadmissible motions; these could be, for example, stretching and shearing. It is also the virtual stiffness λ that constrains the quaternion states to be unit length. More specifically, the deformation in the stretching and shearing direction, or the deformation of quaternion magnitude way from unit norm, will generate a virtual elastic potential energy due to the virtual stiffness λ.

The Lagrangian for energy of the actuated curve is the same as the one

given in (3.8), and we restate it here:

$$L(x, \dot{x}) = \frac{1}{2}(\dot{x} - F_d(x))^\top G(x)(\dot{x} - F_d(x)). \tag{6.39}$$

Once the AGHF derived from (6.39) is numerically solved, and the steady state solution of AGHF $x^*(t, s_{max})$ is obtained, the control $\Delta\omega(t)$ is obtained according to:

$$\Delta\omega(t) := \begin{pmatrix} O_{3\times4} & I_{3\times3} \end{pmatrix} \bar{F}(x(t, s_{\max}))^{-1} \dot{x}(t, s_{\max}) \tag{6.40}$$

which gives the curvature change from natural curvature. The estimated quaternion curve $\hat{q}(t)$ can be obtained by integration of (6.22) with net curvature $\omega(t) = \Delta\omega(t) + \omega^*(t)$. The estimated position curve $\hat{p}(t)$ can be obtained by integrating $\dot{\hat{p}}(t) = R(\hat{q}(t))[1, 0, 0]^\top$. Finally, the estimated pose is given by $\hat{x}(t) = [\hat{p}(t), \hat{q}(t)]^\top$.

In the following steps, we show the equivalence between the Lagrangian (6.39) and the elastic potential in the SCA, then augment the Lagrangian with additional terms that reflect marker constraints, gravity and external force.

Proposition 6.1. *The energy of actuated curve, given by* (6.39)*, coincides with the elastic potential energy of the SCA.*

Proof. In Sec. 6.5.3, the AGHF formulation of the SCA is derived. The actuation matrix F is augmented by $\bar{F}$ in (6.37) to include all admissible and inadmissible controls. The admissible control is the admissible deformation, namely, the curvature change $\Delta\omega$. Let $u_c \in \mathbb{R}^4$ be the control in inadmissible directions, which can be interpreted as the robot deformation in inadmissible directions, i.e, the deformation in the stretching and shearing direction, or the deformation of quaternion against unit norm. Same as the definition of curvature $\Delta\omega$, the inadmissible deformation is a local measure, meaning that it is the deformation along infinitesimal robot length at cross section t.

The complete augmented system with inadmissible control is:

$$\dot{x} = F_d(x) + \bar{F}(x) \begin{bmatrix} u_c \\ u \end{bmatrix} \tag{6.41}$$

Integrating the Lagrangian (6.39) along the robot length, with $\dot{x} - F_d(x)$

term substituted by (6.41) and G replaced by its definition (6.38), we get the energy of actuated curve $\mathcal{A}(x(\cdot))$:

$$\begin{aligned}
\mathcal{A}(x(\cdot)) &= \int_0^1 L(x, \dot{x})dt \\
&= \int_0^1 \frac{1}{2}(\dot{x} - F_d(x))^\top G(x)(\dot{x} - F_d(x))dt \\
&= \int_0^1 \frac{1}{2} \begin{bmatrix} u_c & u \end{bmatrix} \bar{F}^\top(x)(\bar{F}(x)^{-1})^\top D\bar{F}(x)^{-1}\bar{F}(x) \begin{bmatrix} u_c \\ u \end{bmatrix} dt \\
&= \int_0^1 \frac{1}{2}(b_x\Delta\omega_x^2 + b_y\Delta\omega_y^2 + b_z\Delta\omega_z^2 + \lambda u_c^\top u_c)dt \\
&= E_{elastic}
\end{aligned} \tag{6.42}$$

which is the total elastic potential that consists of both elastic potential from admissible deformation and inadmissible deformation. Therefore, the energy of actuated curve coincides with the elastic potential energy. □

With large virtual stiffness λ, the AGHF finds the pose with minimum deformation on inadmissible directions and maintains the unit norm of quaternion.

6.5.4 Integrating vision data via a marker potential

We now show how to use the framework we introduced above to estimate, from vision data captured by a camera, the pose of a SCA. The camera maps a point in 3D space to a point on the 2D image. The use of vision to perform this task is often preferred to having embedded sensors in the SCA, mostly for cost and reliability reasons. The goal is to estimate the 3D pose based on the image of the robot. To facilitate the interpretation of visual data, *markers* are placed on the robot and the locations of the markers are known. Once the image of the pose is obtained from the camera, the actual 3D pose is to be estimated using the positions of the markers on the image.

Let

$$C : \mathbb{R}^3 \to \mathbb{R}^2 : p \mapsto [C_x(p), C_y(p)]^\top$$

be the *camera mapping*, sending a point $p \in \mathbb{R}^3$ in 3D space to the point $[C_x, C_y]^\top \in \mathbb{R}^2$ on 2D camera image. The number of markers is k_m.

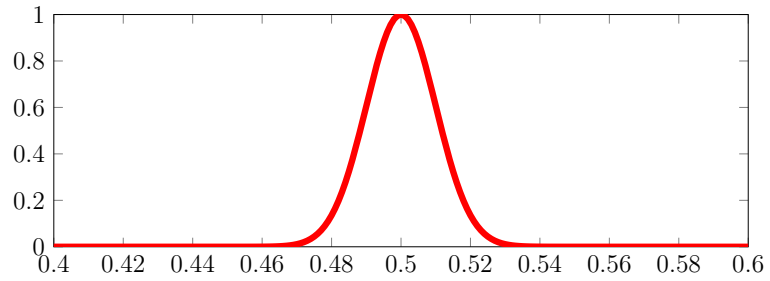


Figure 6.5: An example of activation function at $t = 0.5$

A marker position on the image can be treated as a state-dependent constraint [14] that is only active near the cross-section containing the marker. The installation location of the i-th marker on the robot can be indicated by $t_i \in [0, 1]$. Let $C(p(t_i))$ be the corresponding marker position on the image. A new state can be added for each of the marker position constraints:

$$\zeta_i(t) := \int_0^t ||C(p) - C_i||A_i(\tau)d\tau \longrightarrow \dot{\zeta}_i := ||C(p) - C_i||A_i(t).$$

With the same derivation in Sec. 5.3.3, the augmented Lagrangian with the marker constraints is:

$$L(x, \dot{x}) = \underbrace{\frac{1}{2}(\dot{x} - F_d(x))^\top G(x)(\dot{x} - F_d(x))}_{\text{elastic potential}} + \underbrace{\sum_{i=1}^{k_m} \lambda_i ||C(p) - C_i||^2 A_i^2(t)}_{\text{marker potential}} \tag{6.43}$$

where $C_i \in \mathbb{R}^2$ is the i-th marker position on image, λ_i is a positive constant and $A_i(t) \in \mathbb{R}$ is an activation function that "activates" the constraint $C(p) = C_i$ only at $t = t_i$, as we now elaborate on. We use the following bell-shaped function centered at t_i:

$$A_i(t) = e^{-(t-t_i)^2/\beta^2} \tag{6.44}$$

where β controls the width of the function. An example of the activation function is shown in Fig. 6.5 for $t = 0.5$. With this activation function, the penalty on violation of $C(p) = C_i$ is active near $t = t_i$: the penalty function A_i is large and to minimize the length, the flow will force $\|C(p) - C_i\|$ to be small. Reciprocally, the constraint is not active if t is not near t_i: A_i is small and the value of $\|C(p) - C_i\|$ could be large without affecting the length.

6.5.5 Gravity and External Load Potential

Gravity potential Up to this point, the gravity has not been considered. The potential energy due to gravity is a function of the SCA pose. Assuming the mass is distributed uniformly along the center line of the robot, the potential energy of an infinitesimal segment of the robot is $p_z(t)gdm$ where the mass of the segment is $dm = \rho dt$, ρ being the density with unit kg/m. The total gravity potential energy along the robot is then:

$$PE = \int_0^1 \rho g p_z(t) dt \tag{6.45}$$

With the gravity potential taken into account, the Lagragian becomes:

$$L(x, \dot{x}) = \frac{1}{2} \underbrace{(\dot{x} - F_d(x))^\top G(x)(\dot{x} - F_d(x))}_{\text{elastic potential}} + \underbrace{\sum_{i=1}^{k_m} \lambda_i ||C(p) - C_i||^2 A_i^2(t)}_{\text{marker potential}} + \underbrace{\rho g p_z(t)}_{\text{potential energy}} \tag{6.46}$$

Now the Lagrangian includes elastic potential, gravity potential, as well as the artificial marker potential.

External load In this work, we consider a constant force applied on the tip of the soft robot as the only external load. We refer to it as **the tip force** and denote it by F_{tip}. The *work* done by the tip force at the section t is :

$$W_{load}(p(t)) = (p(t) - p^0(1))^\top F_{tip} \tag{6.47}$$

where $p(1)$ is the loaded tip position and $p^0(1)$ denote the tip position when the robot is at the unloaded pose, which is itself determined by natural curvature ω^*. Eq. (6.47) states the fact that the work W_{load}, done by constant tip force F_{tip}, is the product of the force and the displacement of the tip. The initial position of the tip is obtained from the pose of the unloaded SCA, which is known and determined by the natural curvature. Therefore the work W_{load} is a function of the loaded tip position $p(1)$, if the force is also known. We can conclude that the tip force F_{tip} is a conservative force because the work is only determined by the loaded tip position $p(1)$. As a result, the work

done by F_{tip} can be considered to be a potential energy term.

The work W_{load} can be treated as a terminal cost and transformed into integral form:

$$\begin{aligned} W_{load}(p(1)) &= W_{load}(p(0)) + \int_0^1 \frac{d(W_{load}(p(t)))}{dt} dt \\ &= \underbrace{(p(0) - p^0(1))^\top F_{tip}}_{\text{work 1}} + \underbrace{\int_0^1 \dot{p}^\top F_{tip} dt}_{\text{work 2}} \end{aligned} \tag{6.48}$$

in which the work done by the force is split into two part: work 1 is calculated by replacing $W_{load}(p(0))$ with its expression from Eq. (6.47). This work can be interpreted as the work done by F_{tip} if the force is applied on the unloaded robot at tip initially and moved along the robot quasistatically to the base of the robot, $p(0)$, work 2 is the work done by F_{tip} from the robot base to loaded tip, along the loaded robot.

The total potential energy is conserved because all external forces (gravity and F_{tip}) are conservative. Therefore, to reflect the effect of F_{tip}, we can **subtract** the work W_{load} from the Lagrangian (6.46). Work 1 in (6.48) is constant because $p(0)$ is the fixed base position and $p^0(1)$ is determined by the natural curvature, we only need to include work 2 in the total potential. Therefore we have:

$$\begin{aligned} L(x, \dot{x}) = \frac{1}{2} \underbrace{(\dot{x} - F_d(x))^\top G(x)(\dot{x} - F_d(x))}_{\text{elastic potential}} + \underbrace{\sum_{i=1}^{k_m} \lambda_i ||C(p) - C_i||^2 A_i^2(t)}_{\text{marker potential}} \\ + \underbrace{\rho g p_z(t)}_{\text{gravity potential}} - \underbrace{\dot{p}^\top F_{tip}}_{\text{load work}} \end{aligned} \tag{6.49}$$

where the term "load work" is work 2 without integration, which is the work done by F_{tip} per unit length along the robot.

With this formulation, minimizing the Lagrangian is reflecting the principle of least action. The AGHF is able to find the robot pose with the effect of gravity and a known external load F_{tip} on the tip. It should be noted that, when solving the AGHF, the final value of $x(t)$ should be free so that the algorithm can find the feasible pose with proper tip state.

We can now summarize the content of the above two section in the following

result:

Proposition 6.2. *Consider a SCA with pose $x(t) \in SE(3)$ for $t \in [0,1]$, and markers at positions C_i. Then the Lagrangian density of the SCA is given by Eq.* (6.49)*.*

6.5.6 External Force Estimation

The tip force F_{tip} can be a result of the external load carried by the tip, or due to contact with other objects. If the robot pose $x(t)$ is known, the external force F_{tip} can be extracted from the internal torque balance along the robot. However, it may happen in practical settings that the robot pose under the effect of F_{tip} is not directly known but needs to be estimated from limited sensor data, e.g, marker images or tip position only. In this section, we demonstrate first how to extract the force F_{tip} assuming the robot pose is known via torque balance analysis, then we describe an estimation procedure for the pose under effect of the unknown force F_{tip} using AGHF. The available data is the tip position $p_{tip} \in \mathbb{R}^3$ from sensor. We assume the only external wrench acting on the robot are the force from the load F_{tip}, applied to the tip, and the distributed gravity along the robot. The robot has no contact with the environment except for the tip and the base.

6.5.6.1 Force Estimation

We now show that the force F_{tip} can indeed be recovered from the knowledge of the pose of the SCA. The main idea we exploit here is the one of Torque Balance at a static equilibrium. Recall that $\Delta\omega$ is the curvature change due to external load or variations in the internal pressure (actuation) and serves as control to the system, as in model (6.34). Precisely, we have the following result:

Proposition 6.3 (Force estimation from torque balance)**.** *Consider an SCA as described in Sec. 6.5.3 with known pose $x(t)$ and control $\Delta\omega(t)$ and unknown external load F_{tip}. Then F_{tip} is obtained obtained by solving the internal torque balance Equation* (6.50) *with Equations* (6.51) *to* (6.53)*.*

The proof will be the content of this section. The proposition provides the ingredients for estimating the tip force F_{tip}. A prerequisite is to obtain the

robot pose $x(t)$ and the control $\Delta\omega(t)$, which will be discussed in the next section.

When the robot is in a static pose, the net torque acting on any of the cross sections is *balanced*, as shown in Fig. 6.6. The net torque acting on a cross section t includes, internal torque due to the local curvature deviation from natural curvature $\Delta\omega(t)$, torque generated by F_{tip}, and the torque generated by the weight of robot segment $[t, 1]$:

$$M_{load}(t) + M_g(t) = M(t) \tag{6.50}$$

where M is the internal torque, $M_{load}(t)$ is the torque generated by F_{tip} and M_g is the torque generated by gravity of robot segment $[t, 1]$. We have

$$M_{load}(t) = (p(1) - p(t)) \times F_{tip} \tag{6.51}$$

$$M_g(t) = \int_t^1 (p(\tau) - p(t)) \times \begin{bmatrix} 0 \\ 0 \\ -\rho g \end{bmatrix} d\tau \tag{6.52}$$

$$M(t) = R(q(t))B\Delta\omega(t) \tag{6.53}$$

where $B = \mathrm{diag}([b_z, b_y, b_z])$ is the stiffness matrix of the twisting and bending. Note again that $\Delta\omega(t) = [\Delta\omega_x(t), \Delta\omega_y(t), \Delta\omega_z(t)]^\top$ is curvature difference from the natural curvature $\omega^\star$, and serves as the control.

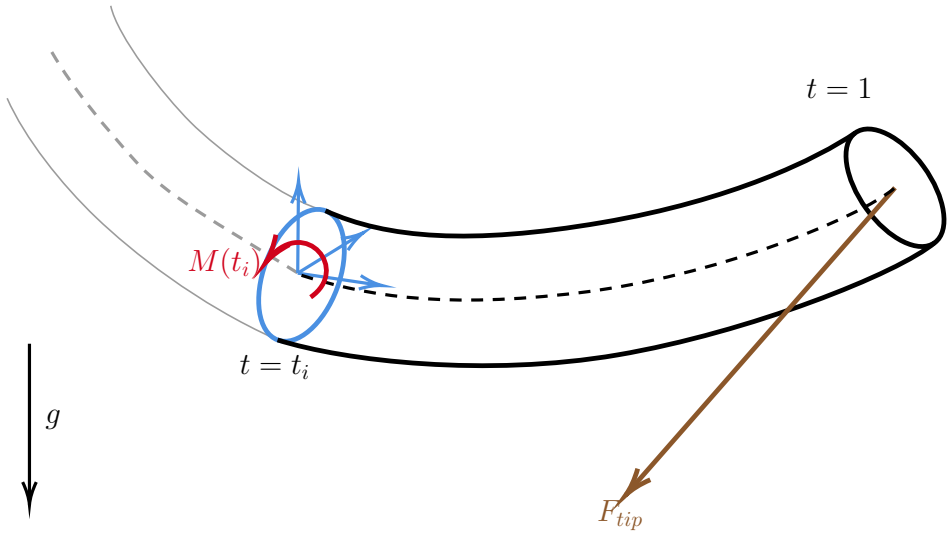


Figure 6.6: Torque balance at cross section t_i: the internal torque $M(t_i)$ is balanced with the torque generated by gravity of the robot segment $[t_i, 1]$ and the torque generated by tip force F_{tip}

If the robot pose $x(t) = [p(t), q(t)]^\top$ and the corresponding $\Delta\omega(t)$ are available, the force F_{tip} can be obtained from the torque balance (6.50). The operation $(p(1) - p(t)) \times (\cdot)$ in (6.51) is equivalent to multiplication of $(\cdot)$ by the skew-symmetric matrix $\widehat{p}(1) - \widehat{p}(t)$ whose rank is 2 for any $p(t)$. By itself, it is thus insufficient to solve for $F_{tip} \in \mathbb{R}^3$. However, the equality (6.50) holds for *all points* along the robot. Since more than one point can be used to solve for F_{tip}, e.g. let t_1 and t_2 be two different cross sections on the robot, and $\text{rank}(\begin{bmatrix}\widehat{p}(1) - \widehat{p}(t_1) & \widehat{p}(1) - \widehat{p}(t_2)\end{bmatrix}) = 3$, then $M_{load}(t_1) + M_g(t_1) = M(t_1)$ and $M_{load}(t_2) + M_g(t_2) = M(t_2)$ together provide sufficient number of independent linear equations to solve for $F_{tip} \in \mathbb{R}^3$. To do so, it is enough to only use the data points in a small segment of the robot, or several selected data points along the entire robot. Note that in the case more than one point is used, an *overdetermined* system is created and least square method is used to solve for the force.

6.5.6.2 Pose Estimation for Unknown External Force

From the internal torque balance analysis, the force F_{tip} can be solved using the pose $x(t)$ under the effect of gravity and tip force. However, it is often the case in practical settings that the pose is not directly available and has to be estimated from limited sensor data. The AGHF derived from

Lagrangian (6.49) can be used in a straightforward manner to find the pose and tip position for a *known force* F_{tip}. However if the force is *unknown*, extra information is needed to estimate the pose. In this section, we assume the tip position is known from sensor data when the robot is loaded with the force F_{tip} at the tip, and we estimate the pose based on the tip position.

Proposition 6.4 (Pose estimation with unknown tip force)**.** *Given the base cross section state x_{init} and tip position p_{tip} with unknown tip force F_{tip} applied, the robot pose $x(t)$ can be obtained by solving the AGHF defined by Lagrangian* (6.56) *with BC $x(0) = x_{init}$ and $p(1) = p_{tip}$.*

The proposition above uses the AGHF method to extract the missing ingredients for the tip force estimation: the robot pose under the effect of unknown tip force. Recall that to solve the tip force from (6.50), the pose $x(t)$ has to be known in advance. The proof of the proposition will be the content of this section.

Both the gravity force and tip force are conservative and generate potential energy, as discussed in Sec. 6.5.5. When the robot is in static equilibrium with the effect of gravity and F_{tip}, the static pose should minimize the total potential. Using (6.49) with no marker use, since $x(t)$ is known, we have that the potential energy is:

$$L(x, \dot{x}) = \frac{1}{2} \underbrace{(\dot{x} - F_d(x))^\top G(x)(\dot{x} - F_d(x))}_{\text{elastic potential}} + \underbrace{\rho g p_z(t)}_{\text{gravity potential}} - \underbrace{\dot{p}^\top F_{tip}}_{\text{load work}} \tag{6.54}$$

$$x^\star(t) = \arg\min_{x(t)} \int_0^1 L(x, \dot{x}) dt \qquad \text{for free } x(1). \tag{6.55}$$

The potential energy from (6.54) should be minimized according to the *principle of least action*, with the tip state $x(1)$ being free. The optimal solution $x^*(t)$ is then the static pose with F_{tip} applied to the tip.

The work done by the external force F_{tip} can be represented by $\int \dot{p}^\top F_{tip} dt$. The integral $\int_0^1 \dot{p}^\top F_{tip} dt = p(1)^\top F_{tip}$ is determined only by the constant F_{tip} if the tip position $p(1)$ is known from sensor data. Whether the force F_{tip} is known or not, the work done by F_{tip} is independent of the path of robot. Hence the $\dot{p}^\top F_{tip}$ term can be eliminated from (6.54) on condition that the tip position is provided. Again, according to principle of least action, the static pose in equilibrium with an *unknown* constant tip force can be found

by:

$$L(x, \dot{x}) = \frac{1}{2}(\dot{x} - F_d(x))^\top G(x)(\dot{x} - F_d(x)) + \rho g p_z \tag{6.56}$$

$$x^\star(t) = \arg\min_{x(t)} \int_0^1 L(x, \dot{x}) dt \qquad \text{for fixed } p(1) \tag{6.57}$$

The discussion above shows two scenarios for the pose with tip force applied. If the tip force is known, the pose should be obtained by minimizing (6.54) with free tip state. If the same tip force is applied but the force is unknown, the tip position should be provided and the pose can be obtained by minimizing (6.56) with tip position fixed to provided tip position. The pose in equilibrium should be the minimizer of the action functionals for both scenarios. Thus the pose can be obtained by the AGHF method using either (6.54) with *known* tip force and *free* tip state, or (6.56) with *known* tip position. It should be noted that it remains an open question if the pose in equilibrium is local or global minimizer and if the pose is unique. In the force estimation task, the tip force can be extracted from the pose obtained by the tip position, using the internal torque balance (6.50).

6.5.6.3 Consistency of Force Estimation

To numerically validate the method proposed to estimate F_{tip},a two-step validation process is conducted. This also allows us to have a broad estimation of what numerical errors the method could incur.

Ground Truth In this step, ground truth data is generated. A known force F_{tip} is applied on a SCA robot whose stiffness for inadmissible deformations, λ^*, and stiffness for admissible deformations (twisting and bending), b_x,b_y and b_z, are known parameters. The robot pose with F_{tip} applied, $x_{nom}(t)$, is the pose of least potential energy, and can be obtained via AGHF using (6.54) and (6.55). From $x_{nom}(t)$, the tip position p_{tip} is recorded and will be used for force estimation. The known force F_{tip}, the stiffness λ^*, b_x,b_y,b_z, and the pose $x_{nom}(t)$ serve as a ground truth.

Force Estimation In this step, the tip force is *unknown* and the only data available to estimate the force is the tip position, p_{tip}, recorded in Ground Truth step. To obtain the pose via AGHF using (6.56), (6.57) and

$p(1) = p_{tip}$ fixed, the stiffness for both inadmissible and admissible directions are needed. The admissible ones, b_x, b_y and b_z are usually given as robot material parameters and can be directly used in AGHF method. However the inadmissible stiffness, λ, are usually assumed to be infinite since it is orders of magnitude larger than the admissible stiffness. Hence, a large guess of λ is used. The discrepancy of the guess between ground truth λ^* is the main source of force estimation error, and will be discussed in next section. The pose obtained using the tip position p_{tip} is the reconstruction of the pose with tip force applied. We denote the reconstructed pose by $x_{rec}(t)$. If the guess of λ is accurate, namely, if $\lambda = \lambda^*$, the pose $x_{rec}(t)$ should be identical with the ground truth pose $x_{nom}(t)$. Finally we can estimate the force using the reconstructed pose $x_{rec}(t)$ and torque balance condition (6.50). It is necessary to note again that the estimated poses $x_{nom}(t)$ and $x_{rec}(t)$ are both integrated paths, which represent valid R^3 and unit quaternion curves.

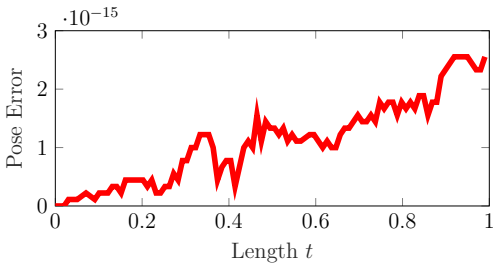


Figure 6.7: Consistency error $||x_{nom}(t) - x_{rec}(t)||$

To verify that the consistency, ground truth values $F_{tip} = [200, -400, -600]^\top$ and with $\lambda^* = 1 \times 10^5$, $[b_x, b_y, b_z] = [100, 100, 100]$, $\omega^*(t) = [0, 0, \pi/2]^\top$ are used in the Ground Turth step. The ground truth pose $x_{nom}(t)$ is obtained and the tip position p_{tip} is recorded. Then the Force Estimation step is conducted with p_{tip} and the guess $\lambda = \lambda^*$. As a result, the pose $x_{rec}(t)$ is obtained and the tip force is extracted. We define the pose consistency error to be $||x_{nom}(t) - x_{rec}(t)||$ and the plot of the error is shown in Fig. 6.7. The error is in the scale of maximum precision of numerical floating numbers. Therefore, the two poses can be considered identical, the optimization problems (6.55) and (6.57) are consistent and lead to the same pose.

Fig. 6.8 shows the extracted force along the robot using $x_{rec}(t)$, where 2000 grids are used to represent the pose along the robot length. The force F_{tip}

is solved by using the torque balance equation (6.50) of each grid and its adjacent grids. Now the extracted force is a function of the length t. As shown in Fig. 6.8, the extracted force is constant along the robot and the same as the ground truth force $[200, -400, -600]^\top$. This is expected since we choose $\lambda = \lambda^*$ in this consistency check. The force being constant implied that, at each cross section, the internal torque and gravity torque can be balanced by the same tip force, which is the extracted force. The unstable behavior of the extracted force near the tip is resulted from bad numerical scaling due to almost singular matrix $\widehat{p}(1) - \widehat{p}(t)$ when $t \approx 1$. As a result, the data near the tip should be discarded to avoid large error.

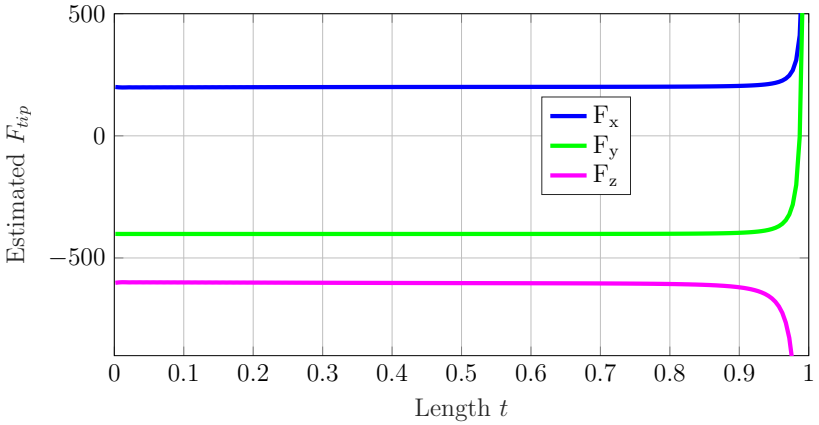


Figure 6.8: Estimated force along the robot

The consistency check shows that the reconstructed pose $x_{rec}(t)$ is identical with the nominal pose $x_{nom}(t)$ if the guess of λ in Force Estimation step is the same as the ground truth λ^* in Ground Truth step, that is to say, if the guess of inadmissible stiffness value is accurate.

6.5.6.4 Estimation Accuracy and inadmissible Stiffness

The stiffness for inadmissible deformations plays an important role in the force estimation accuracy. Recalled from Sec. 6.1, the λ is interpreted as the stiffness of inadmissible deformations. These deformations include the physical stretching and shearing deformation which are physical deformations, as well as the deformation of quaternion away from unit norm, which is an artificial deformation. To investigate the effect of inadmissible stiffness, the Ground Truth step in the previous section is repeated with a range of ground truth λ^*. For each λ^*, the ground truth pose $x_{nom}(t)$ is obtained and the tip

position is recorded. For each of the recorded tip position, the reconstructed pose $x_{rec}(t)$ is generated using a range of guess for λ, and the force is extracted for each $x_{rec}(t)$.

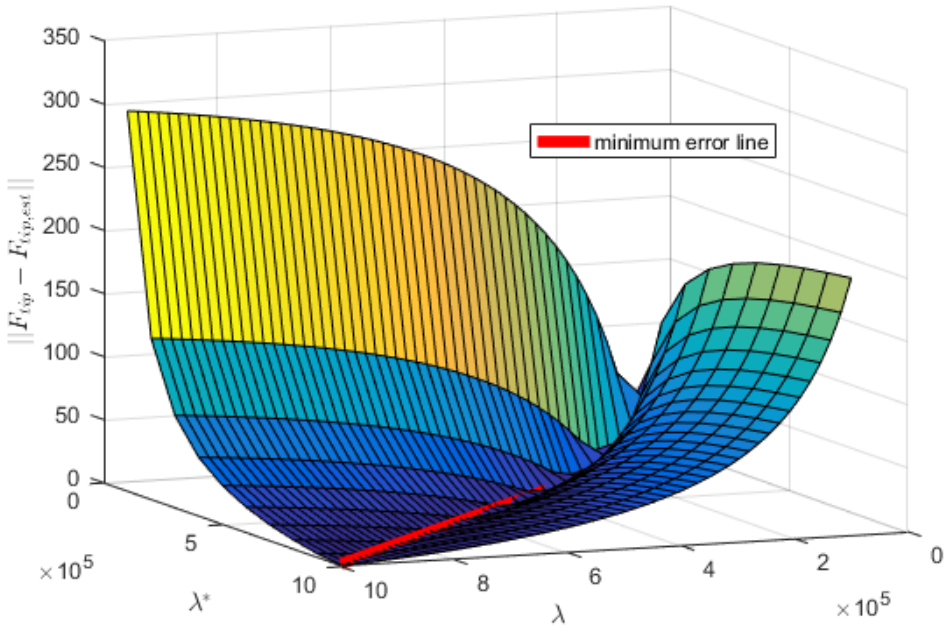


Figure 6.9: 3D surface for $\lambda^* \in [1 \times 10^5, 10 \times 10^5]$ vs $\lambda \in [1 \times 10^5, 10 \times 10^5]$ vs estimation error $||F_{tip} - F_{tip,est}||$

We select ground truth range $\lambda^* \in [1 \times 10^5, 10 \times 10^5]$ and guess range $\lambda \in [1 \times 10^5, 10 \times 10^5]$. For each of the reconstructed poses, the estimated force, $F_{tip,est}$, is extracted by solving the overdetermined equations formed by toruqe balance (6.50) of all sampled cross sections along the robot (except for the segment near tip). As a result, each pose only produces one estimated force. Fig. 6.9 shows the estimation error $||F_{tip} - F_{tip,est}||$ for the selected range of λ^* and λ. For each value of λ^*, the estimation error has a minimum of zero when the $\lambda = \lambda^*$ is zero (red line), which interprets the fact that the force estimation is exact when the inadmissible stiffness used in force estimation is consistent with the inadmissible stiffness of the ground truth. The error gets larger when the discrepancy between λ and λ^* gets larger. An important observation is that, as λ^* increases, the surface is more flat around the minimum error line $\lambda = \lambda^*$. This implies that, a same amount of discrepancy between λ and λ^* makes less estimation error as λ^* increases. For example, the estimation error with discrepancy $\lambda - \lambda^* = 1 \times 10^5$ for

$\lambda^* = 10 \times 10^5$ is smaller than the estimation error with same discrepancy $\lambda - \lambda^* = 1 \times 10^5$ but for $\lambda^* = 9 \times 10^5$. In conclusion, to have an accurate estimation, the inadmissible stiffness λ used in estimation has to be as close to the ground truth value λ^* as possible. However, the effect of discrepancy of inadmissible stiffness on estimation error is small when the ground truth λ^* is large.

The conclusion above helps to conduct the estimation using experimental data. To estimate the force experimentally, an estimation of the inadmissible stiffness is required. If the robot is soft in stretching and shearing, meaning that ground truth λ^* is small, an accurate estimation of λ^* is needed to ensure a good accuracy of force estimation since the assumption of an unshearable robot is in fact not met. However, if the robot is extremely stiff in stretching and shearing, meaning the that ground truth λ^* is large, a rough estimation of λ is acceptable since the discrepancy makes less effect on the force estimation accuracy for large inadmissible stiffness.

6.6 GHF of $SO(3)$ from ground up: key points

In this section, an approach to directly plan motions in $SO(3)$ space with AGHF method is summarized. Suppose the $SO(3)$ state $R(t) = [r_1(t), r_2(t), r_3(t)] \in SO(3)$ is the rotation matrix, with r_i representing the columns. The tangent space at the identity is denoted by $so(3)$; an element of $so(3)$ is denoted by $\widehat{\omega}$ where $\omega \in \mathbb{R}^3$ is the angular velocity in some chosen frame. The $\widehat{\cdot}$ operator is the screw symmetric operator. A metric $G(R)$ can be used to define the length, $L(R, \omega) = \int_0^T \omega^\top G(R)\omega dt$ for motion duration T.

Denote the Euler Lagrange equation for $SO(3)$ curve by $EL(x, \omega)$, which is derived in [63] as:

$$EL(x,\omega) = -\Big\{ \frac{d}{dt}\Big(\frac{\partial L(x,\omega)}{\partial \omega}\Big) + \omega \times \frac{\partial L(x,\omega)}{\partial \omega} + \sum_{i=1}^{3} r_i \times \frac{\partial L(x,\omega)}{\partial r_i} \Big\} \quad (6.58)$$

The heat flow is equivalent as:

$$\frac{dR}{ds} = -R * \widehat{EL}(R,\omega) \quad (6.59)$$

which is the evolution along s (same as $\dot{R} = R\widehat{\omega}$ along t).

As discussed in 6.3, the regular Runge–Kutta integration and interpolation rule does not apply for $SO(3)$ since the integrated or interpolated state doesn't live in $SO(3)$. The integration rule (6.23) should be used. This update formula applies both for updating $R(t, s)$ when solving PDE, and for obtaining the integrated path after solving the motion.

For interpolation, angle-axis interpolation can be used. Given two ration matrices $R(0) = R_1$ and $R(\Delta t) = R_2$, and let $R_1^\top R_2 = R(\theta, v)$, where θ and v are angle and axis in global frame for the rotation from R_1 to R_2. A linear interpolation for orientation and angular velocity can be done along this angle-axis rotation:

$$R(t) = R_1 R(\theta \frac{t}{\Delta t}, v) \tag{6.60}$$

$$\omega(t) = \frac{\theta}{\Delta t} v \tag{6.61}$$

To sum up, the method based on $SO(3)$ requires a customized PDE solver that uses $SO(3)$ integration and interpolation techniques. At this point, since we intend to plan motions that include both $\mathbb{R}^n$ and $SO(n)$ curve, a general purpose PDE solver is preferred. Therefore we chose to approximate $SO(3)$ curve using $\mathbb{R}^n$ curve.

CHAPTER 7

COMPUTATIONAL COMPLEXITY ANALYSIS

In this chapter, the computational complexity of the AGHF approach is analyzed and compared with other methods of motion planning in robotics. The purpose is to emphasis the novelty and potential of the AGHF method on the computation and implementation level. The analysis starts with a brief overview of the fundamentals of the most commonly used motion planning algorithms for complex robot systems, with a focus on trajectory optimization and direct collocation methods. Then, we classify these methods according to how the numerical solving process is carried out, and compare them with the solver that AGHF algorithm based on, which is PDE solver relying on the method of lines (MOL). In order to have a quantitive and intuitive comparison, an illustrating example is implemented in both direct collocation method and AGHF method.

7.1 Fundamentals of Trajectory Optimization Algorithms

Motion planning is a broad and multi-disciplinary research area. Searching based method like A* [64], sampling based method like Probabilistic Road Map (PRM) [65] and Rapidly-exploring Random Tree (RRT) [66, 67] are notable branches in motion planning. These methods are widely applied in ground mobile robot and manipulators, where the system is usually fully actuated and nonholonomic constraints are absent. In the past decade, as robot hardware and software rapidly developed, the robot systems became more complex and dynamic, typified by humanoid robots and legged robots. Underactuation, kinodynamic nature, nonholonomic dynamics and varies type of constraints are main challenges when planning motion for these type of robots. Aiming to address all these issues, methods based on trajectory

optimization were introduced. Trajectory Optimization (TO) is the process of designing a trajectory that minimizes some measure of performance while satisfying a set of constraints, and can be considered a subfield of optimal control theory. The aforementioned issues are then transferred to a solver via encoding the dynamics and constraints into optimality conditions.

We refer the reader to surveys [68, 69, 70] for detailed review and classification of the methods in TO. Fundamental concepts are summarized in this section. The TO aims to solve the following problem: find the control $u(t) \in \mathbb{R}^m$ from initial time t_0 to terminal time t_f such that the following cost function is minimized.

$$J = \Phi(x(t_0), t_0, x(t_f), t_f) + \int_{t_0}^{t_f} L(x(t), u(t), t)dt \tag{7.1}$$

subject to the dynamics

$$\dot{x} = f(x, u, t) \tag{7.2}$$

with path constraints

$$C_{min} \leq C(x, u, t) \leq C_{max} \tag{7.3}$$

and boundary conditions

$$\phi_{min} \leq \phi(x(t_0), t_0, x(t_f), t_f) \leq \phi_{max} \tag{7.4}$$

7.1.1 Indirect Methods

In an indirect method, the TO problem is solved as an optimal control problem, where calculus of variations is used to find the first-order conditions for optimality. Typically, these conditions are derived using the augmented Hamiltonian which involves system states, co-states, and Lagrange multipliers. The first-order necessary condition is given by the following ingredients: The Hamiltonian system dynamics that is the differentiation of the augmented Hamiltonian. Pontryagin's Minimum Principle (PMP) that is used to determine the optimal control, the transversality condition which determines the initial and final value of co-states and complementary slackness conditions for the Lagrange multipliers. The ingredient above form a Hamiltonian boundary-value problem (HBVP), whose extremals are optimal trajectories.

Numerically, solving for the extremals of the HBVP is achieved by first discretizing the state, co-state and control trajectories (i.e., introducing vectors x, p, u whose i-th entries represent $x(t_i), p(t_i)$ and $u(t_i)$, respectively), then converting the first-order necessary conditions into system of differential algebraic equations (DAE). Typical numerical algorithms to solve the resulting problem include indirect shooting, indirect multiple shooting, and indirect collocation.

On the one hand, the first-order necessary condition serving as a optimality metric ensures a high accuracy of the solution when converged, which is the major advantage of the indirect method.

On the other hand, indirect methods are extremely sensitive to the boundary conditions and initial guess. It is usual that small changes in BC or initial guess lead to "wild" trajectories which exceed the numerical range of the computer, as summarized in [71]. In addition, deriving the first-order necessary condition is tedious and sometimes impossible for complex robotic systems. Moreover, introduction of the co-states will double the state space when solving the corresponding DAE problem. Due to these factors, indirect methods are mostly used in aerospace applications where high accuracy planning is required.

7.1.2 Direct Methods

In a direct method, the TO problem is converted to a constrained parameter optimization problem and thus can be solved by general purpose nonlinear programming (NLP) solvers. The control and/or state trajectories are discretized and approximated by a class of properly chosen functions. The system dynamics is either constrained by integration or collocation rules applied on the discretization points. Then the values of discretization points are solved by NLP solvers with integration or collocation constraints. By assembly of the discretized states and control, the optimal trajectory can be obtained. Generally, the direct methods are easy to implement compared with indirect methods, because only discretization and constraint construction are needed, without deriving the first order condition for optimality. However the accuracy of direct method is lower than indirect method since the system dynamics is enforced only on the discretization points. Typical direct methods

include direct shooting and direct collocation.

Direct shooting method approximates the control with specified function, e.g., polynomials, and optimize the function parameters. The state trajectory can be expressed by numerical integration of system dynamics using approximated control function. The NLP solves for the optimal values for the initial state and control function parameter that minimize the specified cost and satisfies the target state constraints. The method is suitable for systems with simple control and constraints.

According to the 2009 survey [69], direct collocation methods are "arguably the most powerful methods for solving general optimal control problems". This is backed up by powerful large-scale sparse NLP solvers that have been developed in the past two decades. Indeed, as the complexity of the TO problem increases, the advantage of direct collocation is more prominent. This is mainly due to two factors. Firstly, the transcription from optimal control to parameter optimization is achieved in an efficient and flexible manner, which allows complex system dynamics, constraints boundary conditions to be simply implemented. Secondly, the NLP solvers converge with bad initial guesses for a solution, and are efficient by utilizing the sparsity of the costs and constraints. On account of the properties above, the direct collocation method is preferred in motion planning for complex robotic systems such as legged robot.

In direct collocation method, the state and control are discretized. Parametric 'basis' functions are used to approximate the trajectory locally over some time interval, or globally over the whole duration of the motion. The key step is to construct the collocations constraints with a proper collocation method that approximates the system dynamics, e.g. Trapezoidal or Simpson-Hermite. The cost is converted to a summation according to the chosen collocation method. Path constraints and boundary conditions are converted to constraints on the state and control at specific discretization points. The resulting NLP problem's size depends on how finely one discretizes the trajectory and, depending on the type of parametric functions used, the cost and constraints representations can be made sparse. An illustrative example of direct collocation is given in Sec. 7.4, with a detailed problem formulation. On the computational side, most NLP solvers eventually employ Newton's method to find the decision variables that satisfy the Karush-Kuhn-Tucker (KKT) condition.

7.1.3 Dynamic Programming

Another branch for solving optimal control problems is dynamic programming (DP). Unlike the direct and indirect methods that find an open-loop local optimal solution, DP finds a closed-loop control policy which gives globally optimal with the help of Hamilton-Jacobi-Bellman (HJB) equation. The trade off is that the HJB equation is PDE w.r.t to time and the states. The computational complexity scales exponentially with the dimension of the state-space; this phenomenon is often referred to as the"curse of dimensionality". This drawback of DP is in itself sufficient to exclude DP from complex robotic applications.

7.2 Solving Trajectory Optimization

In the previous section, fundamentals of different TO and optimal control methods were introduced. The methods are distinguished by how the TO or the optimal control problem is formulated and converted to a numerical problem. From another perspective, if categorized by the type of numerical solver used, the methods fall to two main branches. On the one hand, all the direct and indirect methods produce DAEs, which are mostly solved by Newton's method. On the other hand, the DP can be solved by PDE solvers.

7.2.1 Newton's Method for Indirect and Direct Methods

As mentioned earlier, the indirect methods solves the HBVP which forms a DAE problem. The DAE takes the form:

$$f(x, \dot{x}, t) = 0 \quad \text{for } t \in [t_0, t_f] \tag{7.5}$$

where $x = x(t) \in \mathbb{R}^n$ is the state trajectory and f is a nonlinear function of the state, state derivative and time. To solve the DAE numerically, the most common numerical treatment is to discretize the state and estimate the derivative with finite difference, for example, backward difference:

$$f(x_i, \frac{x_i - x_{i-1}}{\Delta t}, t) = 0 \quad \text{for the } i\text{-th discretization point} \tag{7.6}$$

where Δt is the step size of the discretization. More advanced derivative approximations can be applied to replace $\dot{x}$ with the state x evaluated at nearby points. Applying (7.6) for all the time discretization points, the DAE system (7.5) is converted to a root finding problem.

For direct methods, the TO problem is converted to NLP problem which find the discretized state and control that satisfy the first-order optimality condition, typically the KKT condition. This is also a root finding problem.

The fundamental approach to solve root finding algebraic equations is Newton's method, or some variations of it, which solves the following problem:

$$f(x) = 0 \tag{7.7}$$

where $x \in \mathbb{R}^n$ is the unknown variable vector. Starting with some initial guess x_0, the method produces a series of iterates x_i according to:

$$x_{i+1} = x_i - \left[\frac{\partial f}{\partial x}\right]_{x_i}^{-1} f(x_i) \tag{7.8}$$

where $\frac{\partial f}{\partial x}$ is the Jacobian of the function f.

Close to a root of (7.7), the Newton's method has quadratic convergence. However, when the initial guess is bad, the method may fail to find a solution: if the guess is too far away from the root, or the derivative of the guess is zero, the Newton's method might not converge. In addition, if the function is not continuously differentiable in the neighbourhood of the root, the method might diverge. For complex robotic applications, all of the above scenarios are likely to happen, due to the complexity of the system dynamics and constraints. Furthermore, Newton's method requires calculating the function derivative. In the case of NLP converted from TO, the function derivative is actually the second order derivative of the original Lagrangian, which is usually hard to obtain analytically.

7.2.2 Solving PDE for Dynamic Programming

As discussed in Sec. 7.1, DP relies on solving the so-called HJB equation, which is a PDE for a function whose domain is the state space of the system. Developing numerical methods to solve PDEs is an active area of research. Finite element methods form one broadly used class of methods. In finite

elements methods, the domain of the independent variables is discretized and the derivatives of the unknown functions are approximated by finite difference. The PDE is then converted to a system of algebraic equations.

A different approach is the method of lines (MOL), which approximates the PDE by a system of time varying ODEs [72]. The MOL discretizes the domain of the *spatial* variables only. The derivatives with respect to the spatial variables are approximated by finite difference. Then, for each of the discretized points in spatial domain, an ODE is solved along the time variable. A major appeal of the MOL is that it relies on standard, general-purpose solvers developed for ODEs.

Other methods like spectral methods also come in handy in some particular areas. Nevertheless, the efficiency of DP mainly depends on the complexity and dimension of the HJB, regardless what PDE solver is used. For complex robotic applications with large number of states, the DP is still "cursed" by the dimensionality.

7.3 Computational Complexity of AGHF Method

The AGHF (3.6) is a parabolic PDE of the form:

$$\frac{\partial x(t,s)}{\partial s} = \Psi(x(t,s), \dot{x}(t,s), \ddot{x}(t,s), t) \tag{7.9}$$

Several properties of this specific PDE should be noted. Because the AGHF describes a smooth deformation of the state trajectory into a shortest path, the dependent variables of PDE are the states of the control system (3.3), and not the control. Furthermore, **the evolution of the state only depends on time variable t and the deformation variable s, which implies that the domain of the unknown functions is of dimension 2.** This precludes the appearance of the "curse of dimensionality". With the dimension of PDE fixed to 2, the complexity of solving this PDE numerically scales polynomially with the number of states. More specifically, with the number of discretization points on t and s fixed, the computational complexity is *linearly proportional* to the number of states. This is a major computational advantage over the HJB based methods, since the dependent variable of HJB is the value function and the dimension of independent variable is the number

of states, which scales the complexity exponentially.

Due the the special structure of the AGHF (7.9) (2 dimensional parabolic PDE), the PDE can be solved efficiently using MOL methods. The deformation variable s is treated as the time variable of PDE and is continuous. The time variable t is now a spatial variable of the PDE and is discretized. The derivative terms $\dot{x}$ and $\ddot{x}$ are approximated by finite difference. For each of the discretized state at time t_i, $x(t_i, s)$, an ODE can be solved with the initial value $x(t_i, 0)$. If the system has a state vector $x \in \mathbb{R}^n$ and the states are discretized in t with N discretization points. The total number of ODE is $n \times M$.

The accuracy of AGHF method depends on the magnitude of the constant λ. Increasing λ renders the resulting system of ODEs stiff. Since the AGHF is solved by MOL method, we are able to choose stiff ODE solvers. For instance, we used Matlab's PDE tool box *pdepe* which uses the stiff ODE solver *ode15s*.

In general, a large s_{max} is used to obtain the steady state AGHF solution, whereas a large λ guarantees that error between AGHF solution and the integrated path, denoted by $e = \int_0^T |x^\star(t) - \tilde{x}(t)| dt$, is small. The planning error e can serve as a measure of violation of dynamics and constraints, and it can be shown to converge to zero as $\lambda, s \to \infty$, see Fig. 7.1. For the AGHF method proposed for the hybrid dynamics in Chapter. 5, given large enough λ, the steady state AGHF solution is a feasible trajectory for the approximated robot dynamics instead of the actual hybrid robot system. The model accuracy increases with larger α and β of the smooth approximation of Heaviside step function (5.33), (5.34), resulting in a better approximation of the discrete transition between different modes. That is, with sufficiently large α and β, the violation of constraints near the switching time is negligible.

A practical way to speedup the solving process of the AGHF is to use higher tolerance for the PDE solver. Higher tolerance forces the solver to use larger step size for integration along s direction, thus uses less time for solving. The trade off is that the accuracy along s will be lower. However, the accuracy along s will only affect the transient behavior but has no effect on the steady state solution $x(t, s_{max})$, since the steady state is determined by λ.

The symbolic expression of AGHF has to be derived before solving it as a PDE. It is a one-time computation for each system. In this work, the symbolic expression of the AGHF is solved using `Symbolic Toolbox` in

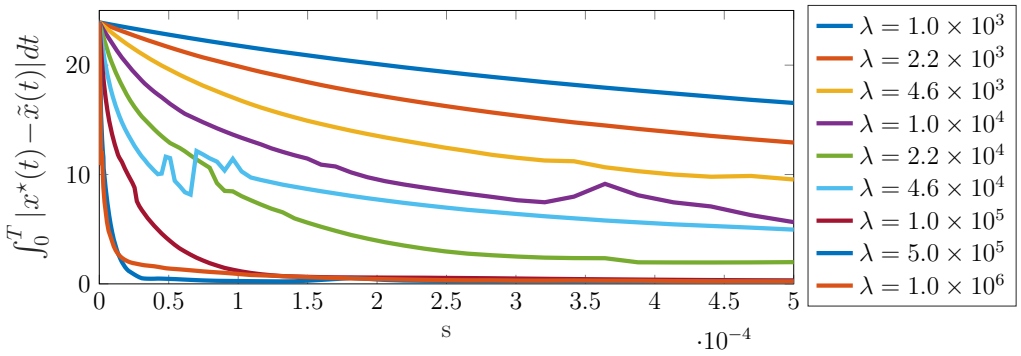


Figure 7.1: Convergence of the planning error e along s and λ for the 2 leg hopping example

`MATLAB`. Alternatively, the AGHF can be calculated numerically, e.g, using finite difference.

7.4 Comparison of the Methods

To benchmark the computational performance of AGHF, in this section, the 3D dynamic unicycle planning example given in Sec. 6.4 is implemented in both direct collocation method and AGHF method. However, the comparison is not intended to serve as an conclusive judgement since the performances are determined by various factors of the implementation, such as selected NLP, PDE solvers, and code efficiency. To simplify the planning task, the constraints on translational and angular velocities (6.32) are relaxed.

7.4.1 Problem Formulation

Now we formulate the 3D unicycle planning problem using both direct collocation method and AGHF method.

Direct Collocation Formulation

The sample direct collocation method with trapezoidal collocation in [73] is implemented for the 3D unicycle. Let the state x and control u be equally discretized in time interval $[0, T]$, with k intervals. Then the step size is $\Delta t = \frac{T}{k}$ and the i-th time stamp is $t_i = ih$ for $i = 0, 1, \cdots, k$. Note that $t_0 = 0$ and $t_k = T$. Let the state and control at time t_i be x_i and u_i. The

planning problem is to find the $k+1$ discrete states $[x_0, x_1, \cdots, x_k]$ and control $[u_0, u_1, \cdots, u_k]$.

Trapezoidal collocation is used to enforce the system dynamics (6.27):

$$x_{i+1} = x_i + \frac{h}{2}\big(f(x_i, u_i) + f(x_{i+1}, u_{i+1})\big) \tag{7.10}$$

for $i = 0, 1, \cdots, k-1$.

In addition, the boundary conditions are assigned as equality constraints:

$$\begin{aligned} x_0 &= x_{\text{init}} \\ x_k &= x_{\text{fin}} \end{aligned} \tag{7.11}$$

The goal is to minimize the Lagrangian $L = u^\top u$. Applying the trapezoidal rule again, the total cost is:

$$J = \sum_0^{k-1} \frac{h}{2}(u_i^\top u_i + u_{i+1}^\top u_{i+1}) \tag{7.12}$$

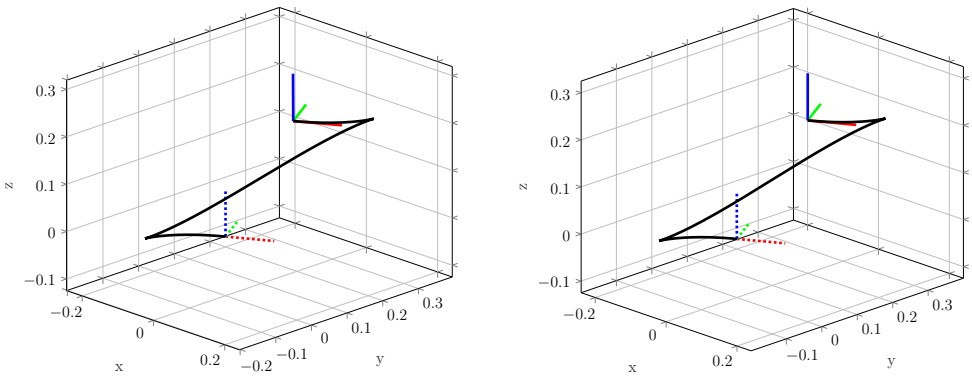
To solve the resulting NLP problem, the NLP formulation platform *CasADI* [74] is used with NLP solver *Ipopt* [75].

AGHF Formulation

The AGHF formulation is the same as Sec. 6.4 with the constraint (6.32) removed. For solving the AGHF equation, we used Matlab's PDE tool box *pdepe* which internally implemented the stiff ODE solver *ode15s*.

7.4.2 3D Dynamical Unicycle Parallel Parking

To compare the generated motions, a 3D parallel parking problem is tested with both methods. The same BC and IC are used for AGHF and direct collocation method. The 3D unicycle start at position $p(0) = [0, 0, 0]^\top$ with orientation $q(0) = [0.9659, 0, 0, 0.2588]^\top$ which corresponds to a rotation of $\pi/6$ around body fixed x axis. The goal is to reach $p(T) = [0, 0.2, 0.2]^\top$ with same orientation $p(T) = p(0)$ with $T = 1$. The initial guess is straight lines connecting the specified BCs, for both AGHF and direct collocation method. The number of discretized time interval is 50 for both methods: the state and



(a) AGHF Path (b) Direct Collocation Path

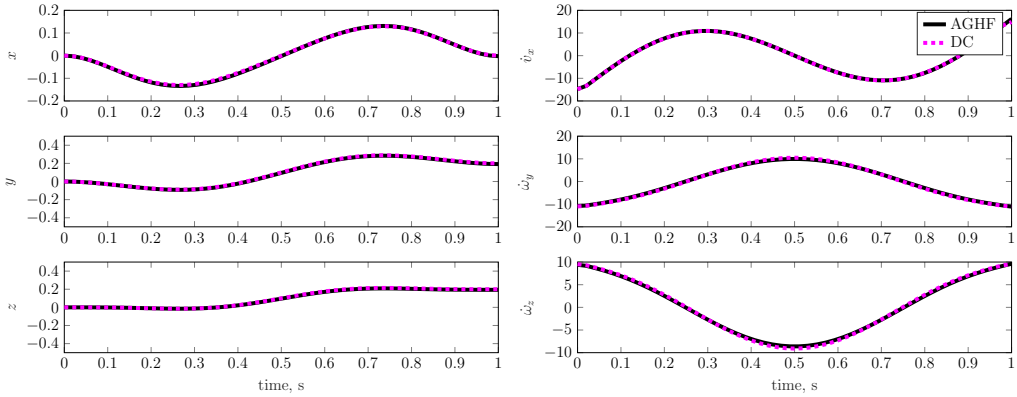
Figure 7.2: Comparison of the motion generated motion from AGHF and direct collocation. The RGB axis shows the initial (dotted) and final (solid) orientations. Solid black line is the x-y-z trajectory.

control in direct collocation are discretized with 50 time intervals, while the state in AGHF methods are solved with 50 sets of ODEs in parallel. For the AGHF method, $\lambda = 50000$ is used to have a good planning accuracy. This planning task serves as a basic task and will be modified for more benchmarks.

The generated motion is shown in Fig. 7.2. Due to the nonholonomic dynamics that the unicyle can only accelerate along its body fixed x axis translationally and around its y, z axis rotationally, the unicyle has to travel a "Z" shape path to reach the target position and orientation. Both methods obtained the same motion which minimized the accelerations. Fig. 7.3 shows a more precise comparison of the generated unicycle position and control, which are almost the same for both methods. The result implies that, given the same initial guess, both methods find the same feasible solution, which is a local minimum near the initial guess. However, the methods are not expected to find the same solution if the same initial guess is given, in general. This will be illustrated in Sec. 7.4.4.

7.4.3 Runtime and Planning Accuracy

As discussed in 7.3, the runtime of AGHF method is partially determined by the constant λ, because it affects the integration step size, if the PDE solver utilizes variable step-size ODE solvers. On the other hand, the runtime is also affected by the number of grid points on time t, which determines how many ODEs are solved in the MOL method. For a fixed λ, the runtime depends



(a) AGHF (b) Direct Collocation

Figure 7.3: Comparison of position and control for AGHF and direct collocation.

solely on the discretization along t. For direct collocation, the runtime is dependent on the number of decision variables, the number of constraints, and the complexity of constraints of the NLP. The number of decision variables and constraints are proportional to the number of discretized grid points along t, while the complexity of constraints is related to the collocation rule. Therefore if the collocation rule is chosen, the runtime solely depends on the number of discretization points.

To benchmark both methods, the same motion planning task in 7.4.2 is repeated with a range of discretization intervals, on a desktop computer with Intel i7-6700 CPU. For each discretization intervals, the algorithms are executed for several times and the average runtime is measured. The tested runtimes are tabulated in Tab. 7.1. The runtime of AGHF is shorter compared with direct collocation. With increasing number of intervals, the increase in runtime is more drastic for direct collocation. Another runtime comparison is shown in Tab. 7.2, where the AGHF method is implemented in *Julia* and the direct collocation method uses Legendre collocation. The PDE solver in *Julia* utilizes MOL and is highly parallelized, therefore the runtime of AGHF can be further reduced compared with using *pdepe*.

In this task, both methods are able to plan the control that steers the state to target final state. With the system dynamics (6.27) and the planned control, the state trajectory $x(t)$ can be simulated by integration, which is the actual motion when given the planned control and can be used to test the planning accuracy. The planning error can be defined as the difference

intervals	20	50	100	150	200	300	500
AGHF (*pdepe*)	0.62 s	1.28 s	1.98 s	2.24 s	4.92 s	7.64 s	7.19 s
DC (Trapezoidal)	0.84 s	1.72 s	1.97 s	8.62 s	15.76 s	15.20 s	28.40 s

Table 7.1: runtime for varies of discretization intervals for AGHF with *pdepe*, and trapezoidal Direct Collocation (DC)

intervals	50	100	200	300	400	500	600
AGHF (*Julia*)	0.27 s	0.50 s	0.99 s	1.51 s	2.02 s	2.58 s	3.23 s
DC (Legendre)	0.29 s	0.50 s	3.00 s	6.81 s	59.16 s	3.68 s	>300 s

Table 7.2: runtime for varies of discretization intervals for AGHF with *Julia*, and Legendre Direct Collocation (DC)

between the simulated final state $x(T)$ and the desired final state x_{fin}:

$$e = |x(T) - x_{\text{fin}}| \tag{7.13}$$

Using the same range of discretization intervals as in Tab. 7.1, the motion planning in 7.4.2 is repeated with both methods. For each planned motion, the state trajectory is simulated with the planned control. To keep the comparison simple, Euler integration and 5000 integration intervals are used. Then the planning error (7.13) is measured, as shown in Fig. 7.4. As more intervals are used, the error decreases and reaches a steady state above zero. The nonzero planning error is caused by the integration error. For the AGHF method, as the number of interval approaches infinity, the planning error is reaching a nonzero steady state which is slightly higher than the steady state value of the direct collocation method. This is because the planning error of AGHF method is determined by the constant λ as discussed in Sec. 7.3. Larger λ gives smaller planning error. Therefore, if the integration error is negligible, increasing the number of discretization points helps to drive the planning error of the direct collocation method to zero, but drive the planning error of AGHF method to its nonzero steady state. However, it is observed from Tab. 7.1 that the runtime of direct collocation experiences more drastic rise when number of intervals increases. To sum up, one can achieve better planning accuracy and faster runtime using AGHF method by tuning the number of intervals and constant λ.

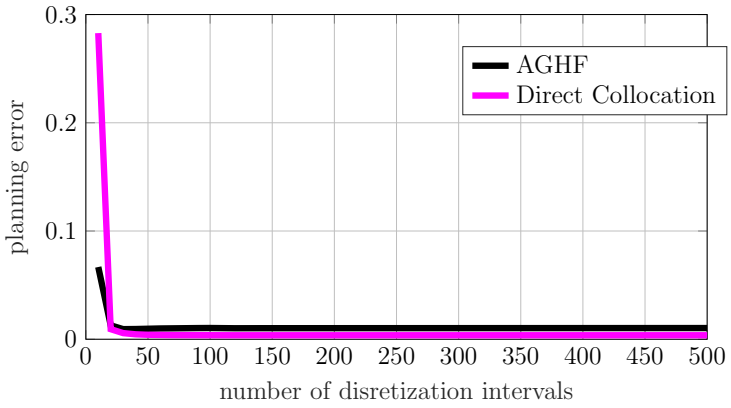


Figure 7.4: Comparison of the planning error for range of discretization intervals

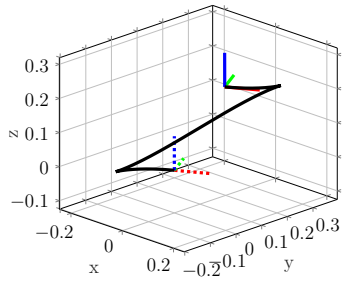
7.4.4 Sensitivity to Initial Guess

The example in Sec. 7.4.2 shows that both methods find the same solution, given the same BC and initial guess. However, this is not guaranteed in general.

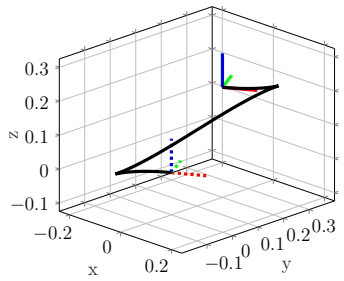
To have a clear view of the effect of initial guess, several different initial guesses are used for the same motion planning problem described in Sec. 7.4.2. The initial guess now is the original straight line connecting x_{init} and x_{fin}, with an added sinusoidal offset $A\sin(\omega_{pert}\frac{2\pi}{T}t)$ on each of the state. That is, the straight line initial guess is perturbed by an oscillation of amplitude A and frequency ω_{pert}. The PDE solver *pdepe* in *Matlab* is used for the AGHF method and the toolbox *CasADI* with NLP solver *Ipopt* is used for the direct collocation method.

A range of A and ω_{pert} are used to see the effect of initial guess on the planned motion, for both methods. The planned motion is shown in Fig. 7.5. First, we choose a high frequency $\omega_{pert} = 10$. $A = 1, 10, 20$ are tested. With low amplitude $A = 1$, as shown in Fig. 7.5a, Fig. 7.5b, both of the methods obtain the same motion, which is the same as the original planned motion in Sec. 7.4.2. The runtime is around 1s for both methods. This implies that the perturbation is not large enough to move the initial guess too far away from this local minimum. With higher amplitude $A = 10$, as shown in Fig. 7.5c and Fig. 7.5d, the methods obtained different solutions. The AGHF solved for a feasible motion in 2s. The motion is still a "Z" shape path, but is different from the planned motion for $A = 1$. This implies that, after being

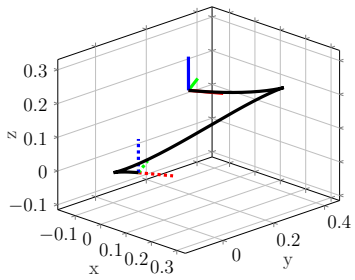
perturbed, the initial guess is "closer" to this new local minimum than the planned motion for $A = 1$. The direct collocation method obtains a motion in 10s but the motion is not feasible since the change in state is too large between each discretization point. With even higher amplitude $A = 20$, as shown in Fig. 7.5e and Fig. 7.5f, the trend is more obvious. The AGHF is still able to find a feasible but different solution, while the direct collocation method fails to find one. Based on these observations, one can conclude that AGHF method is better when the initial guess is fluctuating.



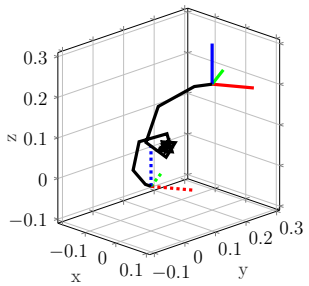
(a) AGHF, with $A = 1$, $\omega_{pert} = 10$



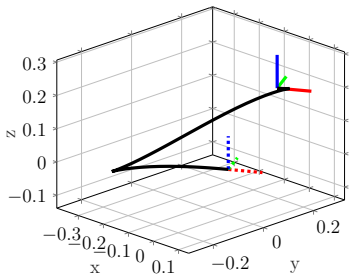
(b) Direct Collocation, with $A = 1$, $\omega_{pert} = 10$



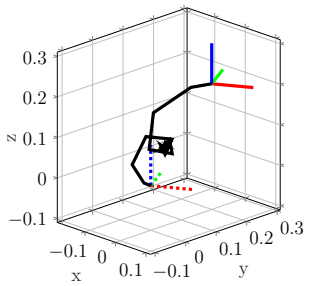
(c) AGHF, with $A = 10$, $\omega_{pert} = 10$



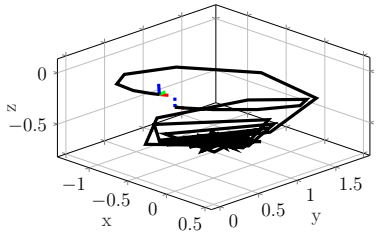
(d) Direct Collocation, with $A = 10$, $\omega_{pert} = 10$



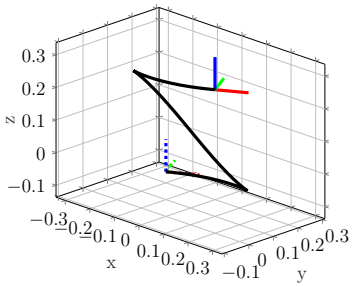
(e) AGHF, with $A = 20$, $\omega_{pert} = 10$



(f) Direct Collocation, with $A = 20$, $\omega_{pert} = 10$



(g) AGHF, with $A = 100$, $\omega_{pert} = 0.5$



(h) Direct Collocation, with $A = 100$, $\omega_{pert} = 0.5$

Figure 7.5: Planned motions with straight line initial guess perturbed by $A\sin(\omega_{pert}\frac{2\pi}{T}t)$. The RGB axis shows the initial (dotted) and final (solid) orientations. Solid black line is the x-y-z trajectory.

.

REFERENCES

[1] S. Seok, A. Wang, M. Y. Chuah, D. J. Hyun, J. Lee, D. M. Otten, J. H. Lang, and S. Kim, "Design principles for energy-efficient legged locomotion and implementation on the mit cheetah robot," *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 3, pp. 1117–1129, 2015.

[2] D. W. Haldane, M. M. Plecnik, J. K. Yim, and R. S. Fearing, "Robotic vertical jumping agility via series-elastic power modulation," *Science Robotics*, vol. 1, no. 1, 2016.

[3] N. K. Uppalapati and G. Krishnan, "Design and modeling of soft continuum manipulators using parallel asymmetric combination of fiber-reinforced elastomers," *Journal of Mechanisms and Robotics*, vol. 13, no. 1, 2020.

[4] C. R. Hargraves and S. W. Paris, "Direct trajectory optimization using nonlinear programming and collocation," *Journal of Guidance, Control, and Dynamics*, vol. 10, no. 4, pp. 338–342, 1987.

[5] O. Von Stryk and R. Bulirsch, "Direct and indirect methods for trajectory optimization," *Annals of operations research*, vol. 37, no. 1, pp. 357–373, 1992.

[6] O. Von Stryk, "Numerical solution of optimal control problems by direct collocation," in *Optimal Control*. Springer, 1993, pp. 129–143.

[7] M. A. Belabbas and S. Liu, "New method for motion planning for nonholonomic systems using partial differential equations," in *2017 American Control Conference (ACC)*, May 2017, pp. 4189–4194.

[8] S. Liu and M.-A. Belabbas, "A homotopy method for motion planning," *Arxiv 1901.10094*, 2019.

[9] F. Jean, *Control of Nonholonomic Systems: from Sub-Riemannian Geometry to Motion Planning*. Springer, 2014.

[10] S. Liu, Y. Fan, and M.-A. Belabbas, "Affine geometric heat flow and motion planning for dynamic systems," *IFAC-PapersOnLine*, vol. 52, no. 16, pp. 168–173, 2019.

[11] Y. Fan, S. Liu, and M.-A. Belabbas, "Mid-air motion planning of floating robot using heat flow method," *IFAC-PapersOnLine*, vol. 52, no. 22, pp. 19–24, 2019.

[12] Y. Fan, S. Liu, and M.-A. Belabbas, "Mid-air motion planning of floating robot using heat flow method with state constraints," *IFAC Mechatronics*, 2019, accepted to publish in 2020.

[13] D. Liberzon, *Switching in systems and control*. Springer, 2003.

[14] Y. Fan, S. Liu, and M. Belabbas, "Geometric heat flow method for legged locomotion planning," *IEEE Control Systems Letters*, vol. 5, no. 3, pp. 941–946, 2021.

[15] K.-S. Chou and X.-P. Zhu, *The curve shortening problem*. CRC Press, 2001.

[16] T. Colding, W. Minicozzi, E. Pedersen et al., "Mean curvature flow," *Bulletin of the American Mathematical Society*, vol. 52, no. 2, pp. 297–333, 2015.

[17] M. P. d. Carmo, *Riemannian geometry*. Birkhäuser, 1992.

[18] J. P. Laumond, S. Sekhavat, and F. Lamiraux, *Robot Motion Planning and Control*, J. P. Laumond, Ed. Springer, 1998.

[19] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[20] R. W. Brockett, *Control Theory and Singular Riemannian Geometry*. New York: Springer, 1982, pp. 11–27.

[21] Z. Li and J. Canny, *Nonholonomic Motion Planning*, ser. The Springer International Series in Engineering and Computer Science. Springer US, 1993.

[22] S. Wang, J. B. Hoagg, and T. M. Seigler, "Orientation control on so(3) with piecewise sinusoids," *Automatica*, vol. 100, pp. 114 – 122, 2019.

[23] H.-B. Dürr, M. S. Stanković, C. Ebenbauer, and K. H. Johansson, "Lie bracket approximation of extremum seeking systems," *Automatica*, vol. 49, no. 6, pp. 1538–1552, 2013.

[24] S. Michalowsky, B. Gharesifard, and C. Ebenbauer, "Distributed extremum seeking over directed graphs," in *Decision and Control (CDC), 2017 IEEE 56th Annual Conference on*. IEEE, 2017, pp. 2095–2101.

[25] A. D. Luca and G. Oriolo, *Modelling and Control of Nonholonomic Mechanical Systems*. Springer, 1995, pp. 277–342.

[26] J. . Godhavn, A. Balluchi, L. Crawford, and S. Sastry, "Path planning for nonholonomic systems with drift," in *Proceedings of the 1997 American Control Conference*, vol. 1, June 1997, pp. 532–536.

[27] M. Reyhanoglu, A. van der Schaft, N. H. Mcclamroch, and I. Kolmanovsky, "Dynamics and control of a class of underactuated mechanical systems," *IEEE Transactions on Automatic Control*, vol. 44, no. 9, pp. 1663–1671, Sep 1999.

[28] J. Jost, *Riemannian Geometry and Geometric Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[29] S. Liu, "Nonlinear and switched systems: Geometric motion planning, non-monotonic lyapunov functions and input-to-state stability," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 2020.

[30] T. Kane and M. Scher, "A dynamical explanation of the falling cat phenomenon," *International journal of solids and structures*, vol. 5, no. 7, pp. 663–670, 1969.

[31] A. Jusufi, D. I. Goldman, S. Revzen, and R. J. Full, "Active tails enhance arboreal acrobatics in geckos," *Proceedings of the National Academy of Sciences*, vol. 105, no. 11, pp. 4215–4219, 2008.

[32] E. Chang-Siu, T. Libby, M. Tomizuka, and R. J. Full, "A lizard-inspired active tail enables rapid maneuvers and dynamic stabilization in a terrestrial robot," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sep. 2011, pp. 1887–1894.

[33] J. Zhao, T. Zhao, N. Xi, M. W. Mutka, and L. Xiao, "Msu tailbot: Controlling aerial maneuver of a miniature-tailed jumping robot," *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 6, pp. 2903–2914, Dec 2015.

[34] J. Hodgins and M. H. Raibert, "Biped gymnastics," *Dynamically Stable Legged Locomotion*, p. 79, 1988.

[35] R. R. Playter and M. H. Raibert, "Control of a biped somersault in 3d," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 1, July 1992, pp. 582–589.

[36] E. Papadopoulos, I. Fragkos, and I. Tortopidis, "On robot gymnastics planning with non-zero angular momentum," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, April 2007, pp. 1443–1448.

[37] E. Papadopoulos and S. Dubowsky, "On the nature of control algorithms for free-floating space manipulators," *IEEE Transactions on Robotics and Automation*, vol. 7, no. 6, pp. 750–758, Dec 1991.

[38] S. Dubowsky and E. Papadopoulos, "The kinematics, dynamics, and control of free-flying and free-floating space robotic systems," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 5, pp. 531–543, Oct 1993.

[39] H. Park, Sangin Park, and S. Kim, "Variable-speed quadrupedal bounding using impulse planning: Untethered high-speed 3d running of mit cheetah 2," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, May 2015, pp. 5163–5170.

[40] J.-M. Godhavn, A. Balluchi, L. Crawford, and S. Sastry, "Steering of a class of nonholonomic systems with drift terms," *Automatica*, vol. 35, no. 5, pp. 837 – 847, 1999.

[41] R. Shu, A. Siravuru, A. Rai, T. Dear, K. Sreenath, and H. Choset, "Optimal control for geometric motion planning of a robot diver," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct 2016, pp. 4780–4785.

[42] J. Bettez-Bouchard and C. Gosselin, "Development and experimental validation of a reorientation algorithm for a free-floating serial manipulator," in *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016, pp. 2733–2738.

[43] J. Koschorreck and K. Mombaur, "Modeling and optimal control of human platform diving with somersaults and twists," *Optimization and Engineering*, vol. 13, no. 1, pp. 29–56, Mar 2012.

[44] S. Bhattacharya, "Search-based path planning with homotopy class constraints," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[45] M. W. Spong and M. Vidyasagar, *Robot dynamics and control*. John Wiley & Sons, 2008.

[46] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa, "Biped walking pattern generation by using preview control of zero-moment point," in *IEEE International Conference on Robotics and Automation*, vol. 2, Sep. 2003, pp. 1620–1626.

[47] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *The International Journal of Robotics Research*, vol. 33, no. 1, pp. 69–81, 2014.

[48] M. Neunert, F. Farshidian, A. W. Winkler, and J. Buchli, "Trajectory optimization through contacts and automatic gait discovery for quadrupeds," *IEEE Rob. and Aut. Lett.*, vol. 2, pp. 1502–1509, 2017.

[49] D. Orin, A. Goswami, and S.-H. Lee, "Centroidal dynamics of a humanoid robot," *Autonomous Robots*, no. 2-3, pp. 161–176, 2013.

[50] H. Dai, A. Valenzuela, and R. Tedrake, "Whole-body motion planning with centroidal dynamics and full kinematics," in *IEEE-RAS International Conference on Humanoid Robots*, 2014, pp. 295–302.

[51] B. Aceituno-Cabezas, C. Mastalli, H. Dai, M. Focchi, A. Radulescu, D. G. Caldwell, J. Cappelletto, J. C. Grieco, G. Fernández-López, and C. Semini, "Simultaneous contact, gait, and motion planning for robust multilegged locomotion via mixed-integer convex optimization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 2531–2538, 2017.

[52] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1560–1567, 2018.

[53] H. Dai and R. Tedrake, "Planning robust walking motion on uneven terrain via convex optimization," in *IEEE-RAS 16th International Conference on Humanoid Robots*, 2016, pp. 579–586.

[54] B. Ponton, A. Herzog, S. Schaal, and L. Righetti, "A convex model of humanoid momentum dynamics for multi-contact motion generation," in *2016 IEEE-RAS 16th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2016, pp. 842–849.

[55] R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *2014 IEEE-RAS international conference on humanoid robots*. IEEE, 2014, pp. 279–286.

[56] S. Liu, Y. Fan, and M.-A. Belabbas, "Geometric motion planning for affine control systems with indefinite boundary conditions and free terminal time," *arXiv preprint arXiv:2001.04540*, 2020.

[57] F. S. Grassia, "Practical parameterization of rotations using the exponential map," *Journal of graphics tools*, vol. 3, no. 3, pp. 29–48, 1998.

[58] K. Shoemake, "Animating rotation with quaternion curves," in *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, 1985, pp. 245–254.

[59] M. Baker. [Online]. Available: https://www.euclideanspace.com/maths/geometry/rotations/conversions/

[60] J. Strickland, "What is a gimbal–and what does it have to do with nasa," *HowStuffWorks, last modified May*, vol. 20, 2008.

[61] A. AlBeladi, G. Krishnan, M.-A. Belabbas, and S. Hutchinson, "Vision-based shape reconstruction of soft continuum arms using a geometric strain parametrization," *arXiv preprint arXiv:2011.09106*, 2020.

[62] M. B. Rubin, *Cosserat theories: shells, rods and points*. Springer Science & Business Media, 2013, vol. 79.

[63] T. Lee, M. Leok, and N. H. McClamroch, "Global formulations of lagrangian and hamiltonian dynamics on manifolds," *Springer*, vol. 13, p. 31, 2017.

[64] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[65] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[66] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[67] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[68] J. T. Betts, "Survey of numerical methods for trajectory optimization," *Journal of guidance, control, and dynamics*, vol. 21, no. 2, pp. 193–207, 1998.

[69] A. V. Rao, "A survey of numerical methods for optimal control," *Advances in the Astronautical Sciences*, vol. 135, no. 1, pp. 497–528, 2009.

[70] A. Shirazi, J. Ceberio, and J. A. Lozano, "Spacecraft trajectory optimization: A review of models, objectives, approaches and solutions," *Progress in Aerospace Sciences*, vol. 102, pp. 76–98, 2018.

[71] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[72] S. Hamdi, W. E. Schiesser, and G. W. Griffiths, "Method of lines," *Scholarpedia*, vol. 2, no. 7, p. 2859, 2007.

[73] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Review*, vol. 59, no. 4, pp. 849–904, 2017.

[74] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, In Press, 2018.

[75] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.