DEGREE PROJECT IN ELECTRICAL ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2022

A Trust-Region Method for Multiple Shooting Optimal Control

Shaohui Yang

KTH ROYAL INSTITUTE OF TECHNOLOGY
ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Author

Shaohui Yang — shaohuiy@kth.se
Master in Systems, Control and Robotics
KTH Royal Institute of Technology

Place for Project

Robotic Systems Lab, ETH Zurich, Switzerland

Examiner

Prof. Karl Henrik Johansson
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology

Supervisors

Prof. Yvonne R. Stürz
School of Electrical Engineering and Computer Science
KTH Royal Institute of Technology

Prof. Marco Hutter
Robotic Systems Lab, Department of Mechanical and Process Engineering
ETH Zurich

Dr. Farbod Farshidian
Robotic Systems Lab, Department of Mechanical and Process Engineering
ETH Zurich

Ruben Grandia
Robotic Systems Lab, Department of Mechanical and Process Engineering
ETH Zurich

Defence Date

25 January 2022

Abstract (English)

In recent years, mobile robots have gained tremendous attention from the entire society: the industry is aiming at selling more intelligent products while the academia is improving their performance from all perspectives. Real world examples include autnomous driving vehicles, multirotors, legged robots, etc. One of the challenging tasks commonly faced by all game players, and all robotics platforms, is to plan motion or locomotion of the robot, calculate an optimal trajectory according to certain criterion and control it accordingly. Difficulty of solving such task usually arises from high-dimensionality and complexity of the system dynamics, fast changing conditions imposed as constraints and necessity for real-time deployment.

This work proposes a method over the aforementioned mission by solving an optimal control problem in a receding horizon fashion. Unlike the existing Sequential Linear Quadratic [1] algorithm which is a continuous-time variant of Differential Dynamic Programming [2], we tackle the problem in a discretized multiple shooting fashion. Sequential Quadratic Programming is employed as optimization technique to solve the constrained Nonlinear Programming iteratively. Moreover, we apply trust region method in the sub Quadratic Programming to handle potential indefiniteness of Hessian matrix as well as to improve robustness of the solver. Simulation and benchmark with previous method have been conducted on robotics platforms to show the effectiveness of our solution and superiority under certain circumstances. Experiments have demonstrated that our method is capable of generating trajectories under complicated scenarios where the Hessian matrix contains negative eigenvalues (e.g. obstacle avoidance).

Keywords

Trust Region Method, Optimal Control, Model Predictive Control, Sequential Quadratic Programming, Quadratic Programming, Indefinte Hessian, Mobile Robots

Abstract (Swedish)

De senaste åren har mobila robotar fått enorm uppmärksamhet från hela samhället: branschen siktar på att sälja mer intelligenta produkter samtidigt som akademin förbättrar sina prestationer ur alla perspektiv. Exempel på verkligheten inkluderar autonoma körande fordon, multirotorer, robotar med ben, etc. En av de utmanande uppgifterna som vanligtvis alla spelare och alla robotplattformar står inför är att planera robotens rörelse eller rörelse, beräkna en optimal bana enligt vissa kriterier och kontrollera det därefter. Svårigheter att lösa en sådan uppgift beror vanligtvis på hög dimensionalitet och komplexitet hos systemdynamiken, snabbt föränderliga villkor som åläggs som begränsningar och nödvändighet för realtidsdistribution.

Detta arbete föreslår en metod över det tidigare nämnda uppdraget genom att lösa ett optimalt kontrollproblem på ett vikande horisont. Till skillnad från den befintliga Sequential Linear Quadratic [1] algoritmen som är en kontinuerlig tidsvariant av Differential Dynamic Programming [2], tar vi oss an problemet på ett diskretiserat multipelfotograferingssätt. Sekventiell kvadratisk programmering används som optimeringsteknik för att lösa den begränsade olinjära programmeringen iterativt. Dessutom tillämpar vi trust region-metoden i den sub-kvadratiska programmeringen för att hantera potentiell obestämdhet av hessisk matris samt för att förbättra lösarens robusthet. Simulering och benchmark med tidigare metod har utförts på robotplattformar för att visa effektiviteten hos vår lösning och överlägsenhet under vissa omständigheter. Experiment har visat att vår metod är kapabel att generera banor under komplicerade scenarier där den hessiska matrisen innehåller negativa egenvärden (t.ex. undvikande av hinder).

Nyckelord

Trust Region Method, Optimal Control, Model Predictive Control, Sekventiell kvadratisk programmering, kvadratisk programmering, obestämd hessian, mobila robotar

Acknowledgements

Before stepping into the main content, a brief introduction of the author goes first. My name is Shaohui Yang and I started my master program Systems, Control and Robotics(SCR) at KTH Royal Institute of Technology, Stockholm, Sweden since 2019 August. My exchange program at ETH Zurich started from Autumn 2020, during which I conducted a semester project in Robotics Systems Lab under the supervision of Ruben Grandia and Jan Carius. This master thesis project, which was conducted in Spring 2021, could be viewed as a continuation of my previous work.

I would like to thank Ruben Grandia and Farbod Farshidian for their great effort and supervision. Also, appreciation goes to Prof. Marco Hutter, the director of Robotics Systems Lab, who has provided me such wonderful opportunity to work in one of the most famous quadrupedal labs! Finally, I would like to express my gratitude to Prof. Karl H. Johansson and Prof. Yvonne R. Stürz, who have kindly agreed to be my thesis examiner and supervisor at KTH respectively.

Acronyms and Symbols

Symbols

$\mathbb{R}$	Set of real numbers
$\mathscr{L}$	Lagrangian function
I_d	Identity matrix, size subject to context. The subscript d is for abbreviation of identity.

Acronyms and Abbreviations

CoM	Center of Mass
OCP	Optimal Control Problem
TO	Trajectory Optimization
MPC	Model Predictive Control
NLP	Non-Linear Programming
DDP	Differential Dynamic Programming
iLQR	iterative Linear Quadratic Regulator
LQR	Linear Quadratic Regulator
LQG	Linear Quadratic Gaussian
SLQ	Sequential Linear Quadratic
AL	Augmented Lagrangian
ALTRO	Augmented Lagrangian TRajectory Optimizer
HDDP	Hybrid Differential Dynamic Programming
MDDP	Multiple-shooting Differential Dynamic Programming
SQP	Sequential Quadratic Programming
QP	Quadratic Programming
TR	Trust Region
LM	Levenberg-Marquardt
LS	Line Search
RK4	Runge–Kutta Method upto 4th Order
KKT	Karush–Kuhn–Tucker
ODE	Ordinary Differential Equation
PDE	Partial Differential Equation

Contents

Chapter 1

Introduction

Given certain high-level task by human, general mobile robots typically complete the following four steps in consecutive orders: (state) estimation, (environmental) perception, planning and control. This work focuses on the final two steps on complicated robotics platforms: planning and control.

Planning is about generating future motions for a robot to finish the high-level task under constraints imposed by internal system dynamics and outside environments. Motions are then carried out by controllers under the usual assumption that the planning result can always be followed tightly. In this procedure, feedback must be incorporated in planning and control simultaneously. Thus, techniques synthesizing these two moves with arbitrary types of constraints and intuitive cost functions naturally grabs the attention of robotists. Among them is the most widely-spread framework: model predictive control (MPC).

In Sec. 1.1, we first reveal the reasons why MPC is becoming more popular and attractive in the field of robots' locomotion / motion planning, trajectory generation and control. Different formulation for optimal control problem (OCP), which is the major sub-problem of MPC, will be treated afterwards in Sec. 1.2. Sec. 1.3 covers how to solve the OCP numerically with a focus on sequential quadratic programming (SQP) and quadratic programming (QP). Introductory words and literature review are coupled in this chapter.

1.1 MPC in Planning and Control

The distinction between planning and control can be quite clear for some robotics systems such as quadrotors and mobile wheeled platforms: motion planner generates time-indexed splines or polynomials specifying the position $p(t)$, velocity $v(t)$, acceleration $a(t)$ and jerk $j(t)$ of the bot center of mass (CoM) while satisfying certain constraints, after which low-level controller follow the trajectory as tightly as possible [3]. This is feasible due to the simplicity of their dynamics in which the status of CoM is capable of representing most body movements under most circumstances. On the contrary, however, this is hardly the case for legged robots where the system dynamics involve tens of states such that planning for CoM only is insufficient. Meanwhile, unlike quadrotor motion planning where dynamic feasibility (not exceeding maximum velocity v_{max} and acceleration bounds a_{max}) is the only internal constraint, there are far more constraints to be considered in legged robots: contact constraints, normal force constraints, friction cone constraints, etc [4]. The planning and control phases for legged robots are highly coupled, so it seems to be unwise to handle them separately.

Thus, recent trends on deploying model predictive control (MPC) on legged robots [1],[5],[6],[7] have been gradually blurring the boundary between planning and control. MPC solves an open-loop optimal control problem with given objective function and constraints over certain horizon length repeatedly and only executes the first part of the resulting input sequence, until new information about states is observed. Such repeating optimization scheme incorporates feedback mechanism implicitly. While guaranteeing recursive feasibility and closed-loop stability, the receding horizon strategy for high dimensional problems suffers from realtively low update rate due to the complexity of solving large-scale optimization problem online. Another arising problem is that solvers typically perform well locally but sometimes undersirable ill-conditioning of the problem and other disturbances will complicate the solving procedure. Consequently, it requires extra effort on improving solver robustness and enabling more generality to it before deploying MPC on legged robots directly. Such two points will be the concentration of this work.

However, it must be pointed out that even though this work proposes MPC solver that performs more robustly and is able to handle certain cases of indefinite Hessian matrices than previous methods, using the optimization result from MPC directly as pure **feedforward** controller on real hardware platforms sometimes will not work in practice. This is especially the case when solver rate is relatively low (under 100 HZ). One shall either add a light-weight motion tracker that operates at much higher frequency, or develop feedback MPC approach as in [1], [6] that all originate from the real-time iteration algorithm proposed in [8], or, if possible, both. In this work, thanks to the Riccati-based optimal control solver (HPIPM [9]), our controller is composed of **feedforward** term u_{ff} and **feedback** matrix term K.

Though the major challenge for legged robot locomotion planning appears to be the low control policy updating rate, deeper implication lies in two perspectives: **how the optimal control problem is formulated** and **what kinds of optimization techniques are involved to solve it**. These two aspects determine the problem's true complexity when deployed on real-world hardware. Consecutive sections will given brief introduction to them respectively.

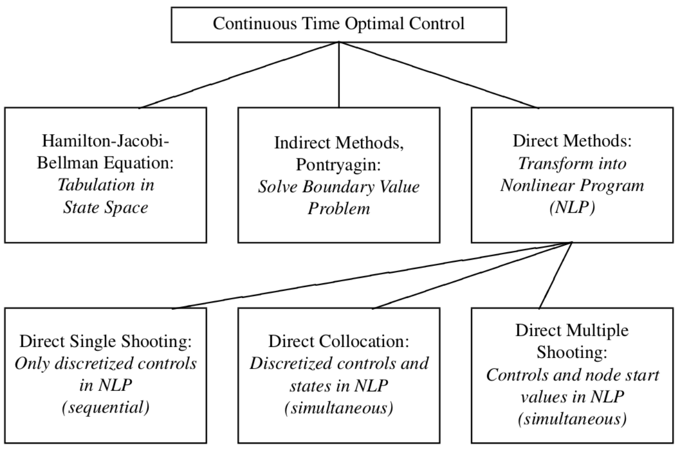
Continuous Time Optimal Control
Hamilton-Jacobi-Bellman Equation: Tabulation in State Space
Indirect Methods, Pontryagin: Solve Boundary Value Problem
Direct Methods: Transform into Nonlinear Program (NLP)
Direct Single Shooting: Only discretized controls in NLP (sequential)
Direct Collocation: Discretized controls and states in NLP (simultaneous)
Direct Multiple Shooting: Controls and node start values in NLP (simultaneous)

Figure 1.1: Family tree of continuous OCP. Image from page 63 of [10].

1.2 How to Formulate OCP

In continuous optimal control problem, we would like to optimize an objective function value under ODE models of system dynamics, path constraints and terminal constraints. Fig. 1.2 shows the variables and constraints. Instead of being too involved into detailes here, we simply point out that the main difficulties lie in nonlinear functions in every aspects and inequality handling. Details will come in Chap. 3.

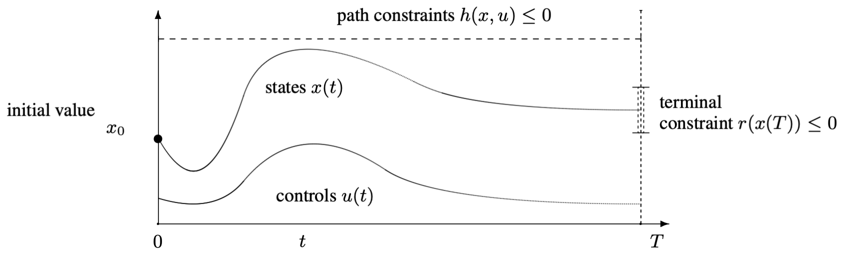

Figure 1.2: Graphical illustration of continuous OCP. Image from page 61 of [10].

We briefly introduce different formulations for continuous optimal control problem in this section, a sub-problem of MPC that is solved in each iteration. If certain methods have been deployed on legged robots, a short review will also be included. As been pointed out by [10], there are three families of methods to address the infinite dimensional continuous optimal control problem (See Fig 1.1).

Before that, we would like to talk a bit regarding to the terminology "optimal control problem" (OCP) and "trajectory optimization" (TO), whose usage in literature are rather chaotic and confusing. There are different definitions for these two terms. Here, we take the opinion from [11] that these two words are pointing at the same set of mathematical problems and tools and will be mentioned interchangably hereafter.

1.2.1 State-space Approach

State-space Approach solves the Hamilton–Jacobi–Bellman (HJB) equation which gives necessary and sufficient condition for optimality of the solution. The process involves numerical solutions of nonlinear partial differential equation (PDE). It used to attract lots of attention due to its principal of optimality but it is always hard to circulate the curse of dimensionality. However, Dynamic Programming-based algorithms do exist and we will introduce the successful ones in the "Differential Dynamic Programming" part.

1.2.2 Indirect Method

Indirect method derives a boundary value problem (BVP) by leveraging the necessary conditions for optimality. BVP in ordinary differential equations (ODE) are then solved numerically, either by shooting or collocation technique (to be covered later). The indirect nature comes from the spirit "first optimize, then discretize". Indirect method includes Pontryagin's Maximum Principle, Euler-Lagrange differential equations and calculus of variations. Such methods suffer from the difficulty that differential equations beneath BVP are difficult to solve if system model and cost functions are strongly nonlinear, nonconvex and unstable.

1.2.3 Direct Method

Direct method transforms the continuous OCP to a discrete one, formulates the discrete OCP to an NLP and solves it. Instead of tackling problems with infinite dimension directly, all transformed optimization are finite dimensional. A summarizing sentence would be "first discretize, then optimize". With well-developed NLP techniques, direct method is good at handling equality/inequality constraints, which usually cause great troubles during the formulation of optimality conditions in indirect methods. However, solver speed for large-scale problems remains unsatisfactory for general nonlinearity and nonconvexity. So direct method typically tends to work well when exploiting specific problem's sparse structure.

Three different ways of formulating such nonlinear programming exist in literature, namely direct single shooting, direct multiple shooting and direct collocation.

Single Shooting

Direct single shooting method [12] is shown in Fig 1.3. The discretized control inputs $u(t;q)$ are parameterized by vector $q = (q_0, \ldots, q_{N-1})$. The intermediate state trajectory $x(t;q)$ is viewed as implicit functions of the control inputs with the usage of suitable differential equation integrator. The optimization is done with respect to the parameterization variables q, so only the inputs are directly optimized. The advantage is that such formulation is dense in structure and less in variable numbers. The drawback is that one has to explicitly integrates along the optimized inputs to retrieve the state trajectory and during the forward integration, unstable dynamic systems might diverge. Robustness and stability of single shooting method is found to be unsatisfactory for the application to legged robots.

Multiple Shooting

Direct multiple shooting method [13] is shown in Fig 1.4. Here both the inputs and intermediate states are considered as optimization variable in the nonlinear programming. Thus, more degrees of freedom will be endorsed. The mid-way states offer foothold for system dynamics integration, so the divergence of dynamic is limited in each step forward. Therefore multiple shooting suffers less from divergence when compared with single shooting. Moreover, multiple shooting needs initialization through the state, which is much more intuitive than input.

More decision variables results in a larger dimension NLP with sparse structure. Structure-exploited solvers should be used for better performance. This method will be the main focus of this work.

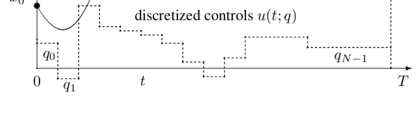

Figure 1.3: Single shooting, from page 66 of [10].

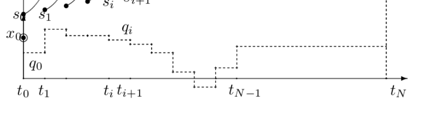

Figure 1.4: Multiple shooting, from page 68 of [10].

Collocation

Direct collocation is a bit different with the two shooting methods mentioned before: the control inputs and states are parametrized functions. Instead of modeling the system dynamics accurately with forward integration, the state involvement is imposed as equality constraints at discrete ”collocation” points. We will not go into the details of this method and interested readers are referred to [10].

1.2.4 Differential Dynamic Programming

Differential Dynamic Programming (DDP) [2] is listed out separately due to two reasons. First, there exists different opinions on how to categorize it. Second, it is the current state-of-the-art and has been deployed on platforms such as ANYmal [14] with great success.

The process of DDP is as follows: we start with an initial state and control policy, simulate forward to get state and input trajectory (”rollout”) and perform taylor expansion around the rollout trajectory with derivatives up to second order. Thus, a quadratic approximation to the original problem is constructed. DDP then solves a backward Riccati recursion to compute the incremental change to the current trajectory. The backward Riccati also brings a time-varying state feedback matrix that will be used in the next rollout iteration.

It divides the big problems into smaller parts and conquer them one-by-one utilizing the Bellman’s principal of optimality in Riccati update equation. So some researchers view it as indirect method [7]. Others think that the decision variables are only control inputs and DDP is optimizing the discretized version of the problem anyway, which qualifies it as a special type of direct single shooting method that cannot handle inequality constraints.

Apart from the discussion on classification, DDP-based methods have drawn lots of attention recently and two main variants have been developed, namely iterative Linear Quadratic Regulator (iLQR) [15] and Sequential Linear Quadratic (SLQ) [16],[1], both of which no longer require the second order information. The advantage of substituting Hessian with certain manipulation of first order information is that computational burden decreases significantly, with the price that quadratic convergence rate degenerates to linear. A series of Gauss-Newton based DDP methods that combined with multiple shooting was proposed in [17], aiming at integrating respective advantages together.

A lot of SLQ/iLQR variants are proposed to handle different types of complicated nonlinear constraints. State-input equality ones are considered in the continuous SLQ settings in [1]. Inequality constraints handling for SLQ with the usage of soft barrier function is enabled in [6]. Introduction of such penalty will aggravate the nonlinearity of entire problem, but it at least frees the solver from active set or interior point method, so trade-off happens here.

Another stream of constraint handling methods in SLQ/iLQR concentrates their focus on Augmented Lagrangian (AL) method, where two loops operate iteratively: an inner loop solver returns state & input trajectory followed by an outer loop updates the lagrangian multipliers and penalty parameters. Building on top of the projection method to eliminate state-input equality constraints in [1], [7] applies AL on inequality and state-only equality constraints, in order to avoid ill-conditioning issues that usually coupled with barrier function related penalty methods. ALTRO [18] also leverages the same strategy. Its another claimed contribution is the introduction of matrix square root by Cholesky factorization to accelerate backward

Riccati recursion, which, according to [19] and [20], has become a quite common technique for optimal control solvers.

HDDP [21] adds a hybrid combination of different nonlinear mathematical programming techniques into DDP: QP subproblems are solved via trust region and range space active-set method and Augmented Lagrangian's inner/outer loop is utilized. MDDP [22] is the successor of HDDP [21] with the spirit of cutting phases in optimal control problem into smaller pieces named "legs", adding boundary and linkage conditions into consecutive "legs" and appling HDDP [21] within each "leg" such that the sensitivity of the entire problem got splitted into subintervals and parallel computing becomes possible.

It is worthy to point out that SLQ/iLQR methods all embrace feedback directly in designing the control policy, which make them suitable to handle disturbance in reality, especially when combined with the real-time iteration scheme [8]. Other efforts towards a better performance on hardware for SLQ includes frequency domain effort for feedback MPC [5], [6].

1.3 How to Solve OCP

As explained above, model predictive control solves an optimal control problem in receding horizon fashion. Depending on the formulation of OCP we choose, different solving mechanics exist:

- If we choose the DDP-based SLQ/iLQR method for individual OCP, we will have to implement the backward Riccati recursion by ourselves. This is the case in OCS2 [23] and ALTRO [18], so it is quite easy to manually create an outer loop for Lagrangian multiplier update and suitable for AL methods when handling constraints. Instead of going into details of this part, we argue that the previous Literature Review paragraph 1.2.4 for DDP suffices.

- It is also reasonable to choose the direct method, which is good at handling any constraints in the resulting nonlinear programming (NLP) originated from the infinite dimensional OCP, because mature solvers for NLP exist a lot. Most of them can handle various kinds of constraints and it is usually up to the users to choose what they need. This is the main focus of this report.

1.3.1 Two Methods for NLP Formulated by Direct Method

Interior point (IP) method [24] and sequential quadratic programming (SQP) are two widely used methods for solving NLP. IP method converts the nonlinear optimization problem to a nonlinear complementarity problem, which is solved by generic interior-point Newton algorithm. SQP solves sequence of QP, each of which is equivalent to solving the Newton-KKT matrix according to first-order optimality for the delta variables [25].

One of the reasons why SQP attracts more attension is that some IP requires the starting guess to be feasible while SQP does not. Sometimes obtaining a feasible guessing of the nonlinear programming is as difficult as getting the optimal solution. Another reason is about warm-starting. Since MPC works in receding horizon, it is straightforward to use the past optimization result as current starting point. But this is not the case for IP method. Hence, we will focus on SQP.

1.3.2 Existing SQP Frameworks

Several great frameworks provide out-of-box SQP solvers, such as SNOPT [26], Acados [27], CasADi [28]. We will not use them directly since either the QP subproblems in SQP are not reformualted to exploit more rigorously the specific sparsity structure that is encoutered in discrete OCP, or an expensive interface is required to generate and pass by partial derivatives.

1.3.3 Existing OCP QP Solvers

An overview of solvers that exploit the structure of optimal control QP can be found in [29].

1.4 Connection between MPC, OCP, SQP and Legged Robot Locomotion

We would like to plan locomotion for legged robot, which means that our resulting state and input trajectory must satisfy the complicated nonlinear system dynamics and lots of (in)equality constraints. It would be wonderful if our planner can return real-time trajectory planning.

We decide to solve the optimization problem in a receding horizon fashion, thus MPC is involved. In each iteration of MPC, an OCP is solved. We review different methods to formulate the OCP and reach the conclusion that multiple shooting method is a better choice over single shooting. When things come to the solving process, we introduce SQP to solve the discrete OCP. Later chapters will bring details during these transitions.

1.5 Step Selection in Optimization Problems

Everything above is introducing how optimal control problem is formulated and solved. Admittedly speaking, OCP has a lot of problem specified structures for exploitation. However, in the end, it still belongs to the big set of optimization problems. Hence, all techniques designated for tackling general optmization problems can be applied to OCP.

One of the key points is the following: assume we are at x_k in the kth iteration. What will be the proper step p_k such that the next iteration point $x_{k+1} = x_k + p_k$ lowers the objective function value and satisfies the constraints?

There are two major approaches to generate p_k, namely line search and trust region. The line search approach first finds a descent direction d_k along which the objective function will be reduced and then computes a step size α that determines how far x_k should move along that direction. The update rule is $x_{k+1} = x_k + p_k = x_k + \alpha d_k$. The descent direction can be computed by various methods.

The trust region approach constructs a model function (usually quadratic) around the current guess x_k, "trusts" the model and returns an optimal step p_k only within certain "region". If an adequate model of the objective function is found within the trust region, then the region is expanded; conversely, if the approximation is poor, then the region is contracted. Details will come in later chapters.

Chapter 2

Methodology

In this chapter, we present the entire mathematical pipeline of solving a continuous optimal control problem formulated in multiple shooting style with trust region method used. The resulting trajectory is calculated in a receding horizon fashion. Most of the equations and deduction in this chapter are converted to C++ code as part of OCS2 package [1].

2.1 Problem Definition

Ideally speaking, we would like to solve the following continuous optimal control problem (or its simplified variants) for mobile robot with state trajectory $x(\cdot) : \mathbb{R} \to \mathbb{R}^n$, input trajectory $u(\cdot) : \mathbb{R} \to \mathbb{R}^m$ and time horizon length $T \in \mathbb{R}$ as decision variables. Due to limited computational resources, the temporal variable T will not appear in the final OCP formulation. Its appearance here only shows the rigorousness and generality of our starting point. $f_c : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^n, g : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^p, h : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^q, L : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}, E : \mathbb{R}^n \to \mathbb{R}$

$$
\begin{aligned}
\underset{x(\cdot),u(\cdot),T}{\text{minimize}} \quad & \int_0^T L(x(t), u(t))dt + E(x(T)) & \\
\text{subject to} \quad & x(0) - \bar{x}_0 = 0 & \text{fixed initial state} \\
& \dot{x}(t) - f_c(x(t), u(t), t) = 0 & \text{time dependent model} \\
& g(x(t), u(t), t) = 0 & \text{state-input equality constraints} \\
& h(x(t), u(t), t) \geq 0 & \text{inequality constraints}
\end{aligned}
\tag{2.1}
$$

The system model, constraints and cost function are viewed as given conditions. They may bear high nonlinearity at the same time. Other than that, there are still individuals to be noticed:

- $L(x(t), u(t))$ is the stage cost along system evolution and $E(x(T))$ is the final stage cost. In classical LQR, these functions are typically quadratic.
- The ODE models a time dependent system $f_c(x, u, t)$. For simple models such as ballbot or drone, f_c does not evolve along with time. However, this is not the case for legged robots since gait patterns will be different from time to time.
- State estimator returns the current state $\bar{x}_0$ of robot as initial value.

[1]Code available at `https://bitbucket.org/leggedrobotics/ocs2_dev/src/master/`, branch sqp_trail_mpcnode

- Equality constraints $g(x, u, t)$ are introduced by contacts with ground (stance condition), dynamic consistency and swing task. $g(x(t), u(t), t)$ is time dependent because gait patterns of the legged robot are time switching. We assume that the gait patterns (e.g. stance, dynamic walk, trot, skipping and their hybrid mode changing time respectively) are either specified by the user or returned by high-level motion planner.

- Inequality constraints $h(x, u, t)$ encodes torque, friction and kinematic limits.

As [6] suggested, we are using relaxed barrier function to absorb inequality constraints to cost function. 2.1 becomes to the following:

$$\begin{aligned}
&\underset{x(\cdot),u(\cdot),T}{\text{minimize}} && \int_0^T L(x(t),u(t)) + P(h(x(t),u(t),t))dt + E(x(T)) && \\
&\text{subject to} && x(0) - \bar{x}_0 = 0 && \text{fixed initial state} \\
& && \dot{x}(t) - f_c(x(t),u(t),t) = 0 && \text{time dependent model} \\
& && g(x(t),u(t),t) = 0 && \text{state-input equality constraints} \\
& && \cancel{h(x(t),u(t),t)} \geq 0 && \text{inequality constraints as penalty}
\end{aligned} \tag{2.2}$$

In the following sections, we will present how to approximately solve the continuous OCP via SQP with trust region method by 1. discretization; 2. linearization to quadratic approximation; 3. elimination of state-input equality constraints; 4. QP sub-problem combined with trust region method.

2.2 Discretization of Continuous OCP

2.2.1 Formulation with Multiple Shooting

We transform the continuous OCP 2.2 to a discrete one in a multiple shooting fashion, which means we view both state and input as decision variables. Note that $\tilde{L}(x_k, u_k) = (t_{k+1} - t_k)[L(x_k, u_k) + P(h(x_k, u_k, t_k))]$ where we use Forward Euler method for stage cost and penalty term in the integration.

$$\begin{aligned}
&\underset{x_0,u_0,x_1,\ldots,u_{N-1},x_N}{\text{minimize}} && \sum_{k=0}^{N-1} \tilde{L}(x_k,u_k) + E(x_N) && \\
&\text{subject to} && x_0 = \bar{x}_0 && \text{fixed initial value} \\
& && x_{k+1} = f_d(x_k,u_k,t_k) && \text{discrete time dependent model} \\
& && g(x_k,u_k,t_k) = 0 && \text{equality enforced at discrete time steps} \\
& && \{t_k\}\&N && \text{temporal settings, treated as given}
\end{aligned} \tag{2.3}$$

Similar to the continuous case, there are several things noticeable

- We are solving the problem in a multiple shooting fashion, so the degrees of freedom in optimization come from both the state trajectory $\{x_0, x_1, \ldots, x_N\}$ and the input $\{u_0, u_1, \ldots, u_{N-1}\}$. The equality constraint induced by evolution between consecutive stages (See Fig.1.4) is represented by the discrete time model.

- The discretization time steps $\{t_k\}$ are specified by user or returned by high-level motion planner. In continuous setting, the switching time between different gait patterns matters because we have to know when to apply which constraints. In discrete setting, other than switching times, the number of discrete samples in each mode also matters. The horizon length N is user-defined. We are abandoning temporal variables from degree of freedom because spatial-temporal adversary optimization is one of the central challenges in optimal control problem. Otherwise, it would be too costly to handle in real-time.

- Equality constraints are enforced at given time instants only. We could also use the dynamics to formulate constraints at other time instants but we choose not to do so.
- The conversion between continuous and discrete model (cost function) involves explicit integration such as Runge–Kutta methods: 1st order corresponds to Forward Euler; 4th order is the commonly used RK4 method. See Appendix B for more details.

2.3 SQP and Discrete OCP

In this section, we first introduce Newton-KKT method to solve the linear equality constrained quadratic programming. Then we extend the Newton-KKT system to general nonlinear optimization problems such that sequential quadratic programming is assembled. Finally, we show how to fit the discrete OCP 2.3 into the SQP framework.

2.3.1 QP with Linear Equality Constraints and KKT matrix

The content of this subsection will be heavily used in the coming subsection because in SQP we solve a QP per iteration. We will go over some properties and specialties of linear equality constrained QP to provide sufficient background information. This part is available at Chap.16 in [30]. Consider a QP with $Q \in \mathbb{R}^{n\times n}$, $A \in \mathbb{R}^{m\times n}$, $m \leq n$.

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & \frac{1}{2}x^{\mathrm{T}}Qx + x^{\mathrm{T}}c \\ \text{subject to} \quad & Ax = b \end{aligned} \tag{2.4}$$

The corresponding Lagrangian function is

$$\mathscr{L}(x,\lambda) = \frac{1}{2}x^{\mathrm{T}}Qx + x^{\mathrm{T}}c - \lambda^{\mathrm{T}}(Ax - b) \tag{2.5}$$

The first-order conditions state that if x^* is a local minimizer of 2.4, then there exists Lagrangian multiplier λ^* such that

$$\nabla_x\mathscr{L}(x^*,\lambda^*) = Qx^* + c - A^{\mathrm{T}}\lambda^* = 0 \tag{2.6a}$$

$$Ax^* - b = 0 \tag{2.6b}$$

By viewing the necessary conditions in 2.6 as a linear system

$$\begin{bmatrix} Q & -A^{\mathrm{T}} \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -c \\ b \end{bmatrix} \tag{2.7}$$

If we substitute $x^* = x + \varDelta x$ with x to be the current guess of x^* and $\varDelta x$ to be the step, then 2.7 is equivalent to

$$\begin{bmatrix} Q & A^{\mathrm{T}} \\ A & 0 \end{bmatrix} \begin{bmatrix} -\varDelta x \\ \lambda^* \end{bmatrix} = \begin{bmatrix} g \\ h \end{bmatrix} \tag{2.8}$$

with $g = c + Qx$, $h = Ax - b$, $\Delta x = x^* - x$. The sign change of A^{T} is subject to the newly formed variables $g, h, \Delta x$. In the literature, KKT matrix may refer to the matrices in 2.7 or 2.8 depending on different scenarios. If any one of them is invertible, so is the other one. Here we present conditions that lead to nonsingular KKT matrix in 2.8. The proof is available in Appendix A.

- Matrix A has full row rank
- $Z^{\mathrm{T}}QZ$ is positive definite, where $Z = \begin{bmatrix} z_1 & z_2 & \dots & z_{n-m} \end{bmatrix}$ such that $\{z_1, z_2, \dots, z_{n-m}\}$ form a basis of the null space of matrix A.

2.3.2 Sequential Quadratic Programming Formulation

Consider a general nonlinear optimization problem with objective function $f : \mathbb{R}^n \to \mathbb{R}$ and equality constraints $g : \mathbb{R}^n \to \mathbb{R}^m, m \leq n$

$$\begin{aligned} &\underset{x \in \mathbb{R}^n}{\text{minimize}} && f(x) \\ &\text{subject to} && g(x) = 0 \end{aligned} \tag{2.9}$$

The Lagrangian function corresponding to the nonlinear programming is

$$\mathscr{L}(x, \lambda) = f(x) - \lambda^{\mathrm{T}} g(x) \tag{2.10}$$

The first-order conditions state that if x^* is a local minimizer of 2.9, then there exists Lagrangian multiplier λ^* such that

$$\nabla_x \mathscr{L}(x^*, \lambda^*) = \nabla f(x^*) - G(x^*)^{\mathrm{T}} \lambda^* = 0 \tag{2.11a}$$

$$g(x^*) = 0 \tag{2.11b}$$

where $G(x)^{\mathrm{T}} = \begin{bmatrix} \nabla g_1(x) & \nabla g_2(x) & \dots & \nabla g_m(x) \end{bmatrix} \in \mathbb{R}^{n \times m}$ is the Jacobian of equality constraints. By viewing the necessary conditions in 2.11 as a big function

$$\mathscr{F}(x, \lambda) = \begin{bmatrix} \nabla f(x) - G(x)^{\mathrm{T}} \lambda \\ g(x) \end{bmatrix} = 0 \tag{2.12}$$

and apply the Newton's method in root finding to $\mathscr{F} = 0$ based on the current guess

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ \lambda_k \end{bmatrix} + \begin{bmatrix} \Delta x_k \\ \Delta \lambda_k \end{bmatrix} \tag{2.13}$$

with the updated step satisfying

$$\begin{gathered} \mathscr{F}'(x_k, \lambda_k) \begin{bmatrix} \Delta x_k \\ \Delta \lambda_k \end{bmatrix} = -\mathscr{F}(x_k, \lambda_k) \\ \begin{bmatrix} \nabla_x^2 \mathscr{L}(x_k, \lambda_k) & -G(x_k)^{\mathrm{T}} \\ G(x_k) & 0 \end{bmatrix} \begin{bmatrix} \Delta x_k \\ \Delta \lambda_k \end{bmatrix} = \begin{bmatrix} -\nabla f(x_k) + G(x_k)^{\mathrm{T}} \lambda_k \\ -g(x_k) \end{bmatrix} \end{gathered} \tag{2.14}$$

For a Newton step to be meaningful, the KKT matrix in 2.14 has to be nonsingular. We note that $\mathscr{F}'(x_k, \lambda_k)$ has similar structure with 2.7, so the sufficient conditions for invertiblity are also alike:

- The Jacobian of equality constraints $G(x)$ has full row rank
- $\forall d \neq 0$ such that $G(x)d = 0$, $d^{\mathrm{T}} \nabla_x^2 \mathscr{L}(x, \lambda) d > 0$

As been pointed out in the previous subsection, QP and KKT matrix have one-to-one onto bijection, so it is natural to map 2.14 to a quadratic programming (QP). Note that the constant offset $f(x_k)$ in objective function does not matter.

$$\begin{aligned} \underset{\Delta x_k \in \mathbb{R}^n}{\text{minimize}} \quad & f(x_k) + \nabla f(x_k)^{\mathrm{T}} \Delta x_k + \frac{1}{2} \Delta x_k^{\mathrm{T}} \nabla_x^2 \mathscr{L}(x_k, \lambda_k) \Delta x_k \\ \text{subject to} \quad & G(x_k)\Delta x_k + g(x_k) = 0 \end{aligned} \tag{2.15}$$

Solving 2.15 is equivalent to solving the following KKT matrix equation where Δx_k^* and μ_k^* are the primal and dual solutions respectively:

$$\begin{bmatrix} \nabla_x^2 \mathscr{L}(x_k, \lambda_k) & -G(x_k)^{\mathrm{T}} \\ G(x_k) & 0 \end{bmatrix} \begin{bmatrix} \Delta x_k^* \\ \mu_k^* \end{bmatrix} = \begin{bmatrix} -\nabla f(x_k) \\ -g(x_k) \end{bmatrix} \tag{2.16}$$

2.14 and 2.16 are equivalent up to the simple transformation $\mu_k^* = \lambda_k + \Delta\lambda_k = \lambda_{k+1}$

Everything above summarizes the whole procedure of basic SQP. We are aiming at solving 2.9. We start from a guess x_0, λ_0. We then solve the QP formulated by 2.15 in each iteration and retrieve the optimal value $\Delta x_k^*, \mu_k^*$. In the next iteration, $x_{k+1} = x_k + \Delta x_k^*$ and $\lambda_{k+1} = \mu_k^*$ are plugged into the new QP. The original nonlinear programming is solved via a sequence of QP, hence then name Sequential Quadratic Programming(SQP). Since the sequence of QP is equivalent to standard Newton's method with step size $\alpha = 1$, local convergence $x_k \to x^*, \lambda_k \to \lambda^*, k \to \infty$ with (x^*, λ^*) a pair of local minimizer, is guaranteed with quadratic convergence rate provided that $\|x_0 - x^*\|$ is sufficiently small [25].

Here we only present the basic version of SQP algorithms. We also list several improvable techniques, some of which will be discussed below

- In practical usage, the Hessian of Lagrangian $\nabla_x^2 \mathscr{L}(x_k, \lambda_k)$ is expensive to compute in most cases. So BFGS methods are used to obtain an approximation $B_k \approx \nabla_x^2 \mathscr{L}(x_k, \lambda_k)$.

- Step size can be leveraged in the update step of SQP: $x_{k+1} = x_k + \alpha \Delta x_k$ and $\lambda_{k+1} = \lambda_k + \alpha \Delta\lambda_k$. In the Newton-KKT setup, introduction of α will significantly decrease the theoretical convergence rate, usually to linear [25]. However, if approximation of Hessian of Lagrangian B_k is used, line search typically returns better practical performance.

- If line search can be leveraged to adjust the update step, so can trust region, especially when the Hessian matrix in 2.15 is indefinite.

2.3.3 Discrete OCP into SQP Framework

This subsection will be the addition of the previous two subsections. We rewrite the nonlinear optimization problem 2.3 into the following pattern with $\omega = \{x_0, u_0, x_1, u_1, \ldots, x_{N-1}, u_{N-1}, x_N\}$ and $\lambda = \{\lambda_1^{\mathscr{M}}, \lambda_2^{\mathscr{M}}, \ldots, \lambda_N^{\mathscr{M}}, \lambda_0^{\mathscr{E}}, \lambda_1^{\mathscr{E}}, \ldots, \lambda_{N-1}^{\mathscr{E}}\}$. The upper calligraphic letter $\mathscr{M}$ and $\mathscr{E}$ for λ means Lagrangian multiplier for model of dynamic systems and equality constraint respectively.

$$\begin{aligned} \underset{\omega}{\text{minimize}} \quad & f(\omega) \\ \text{subject to} \quad & g(\omega) = 0 \end{aligned} \tag{2.17}$$

where

$$f(\omega) := \sum_{k=0}^{N-1} \tilde{L}(x_k, u_k) + E(x_N) \tag{2.18a}$$

$$g(\omega) := \begin{bmatrix} x_1 - f_d(x_0, u_0, t_0) \\ x_2 - f_d(x_1, u_1, t_1) \\ \vdots \\ x_N - f_d(x_{N-1}, u_{N-1}, t_{N-1}) \\ g(x_0, u_0, t_0) \\ g(x_1, u_1, t_1) \\ \vdots \\ g(x_{N-1}, u_{N-1}, t_{N-1}) \end{bmatrix} \tag{2.18b}$$

The Lagrangian function will be

$$\begin{aligned} \mathscr{L}(\omega, \lambda) &= f(\omega) - \lambda^{\mathrm{T}} g(\omega) \\ &= \sum_{k=0}^{N-1} L(x_k, u_k) + E(x_N) - \sum_{k=0}^{N-1} (\lambda_{k+1}^{\mathscr{M}})^{\mathrm{T}} (x_{k+1} - f_d(x_k, u_k, t_k)) - \sum_{k=0}^{N-1} (\lambda_k^{\mathscr{E}})^{\mathrm{T}} g(x_k, u_k, t_k) \end{aligned} \tag{2.19}$$

The first order derivative $\nabla_\omega \mathscr{L}(\omega, \lambda)$ is the following

$$\nabla_{x_k} \mathscr{L}(\omega, \lambda) = \begin{cases} \nabla_{x_0} L(x_0, u_0) + \frac{\partial f_d}{\partial x_0}(x_0, u_0, t_0)^{\mathrm{T}} \lambda_1^{\mathscr{M}} - \frac{\partial g}{\partial x_0}(x_0, u_0, t_0)^{\mathrm{T}} \lambda_1^{\mathscr{E}} & k = 0 \\ \nabla_{x_k} L(x_k, u_k) - \lambda_k^{\mathscr{M}} + \frac{\partial f_d}{\partial x_k}(x_k, u_k, t_k)^{\mathrm{T}} \lambda_{k+1}^{\mathscr{M}} - \frac{\partial g}{\partial x_k}(x_k, u_k, t_k)^{\mathrm{T}} \lambda_k^{\mathscr{E}} & k = 1, \ldots, N-1 \\ \nabla_{x_N} E(x_N) - \lambda_N^{\mathscr{M}} & k = N \end{cases} \tag{2.20a}$$

$$\nabla_{u_k} \mathscr{L}(\omega, \lambda) = \nabla_{u_k} L(x_k, u_k) + \frac{\partial f_d}{\partial u_k}(x_k, u_k, t_k)^{\mathrm{T}} \lambda_{k+1}^{\mathscr{M}} - \frac{\partial g}{\partial u_k}(x_k, u_k, t_k)^{\mathrm{T}} \lambda_k^{\mathscr{E}} \quad k = 0, 1, \ldots, N-1 \tag{2.20b}$$

2.20a and 2.20b indicates the Hessian of Lagrangian is block-diagonal, i.e, $\forall k \neq l$.

$$\frac{\partial}{\partial x_k} \frac{\partial}{\partial x_l} \mathscr{L} = 0 \tag{2.21}$$

$$\frac{\partial}{\partial x_k} \frac{\partial}{\partial u_l} \mathscr{L} = 0 \tag{2.22}$$

$$\frac{\partial}{\partial u_k} \frac{\partial}{\partial u_l} \mathscr{L} = 0 \tag{2.23}$$

The Jacobian matrix of $g(\omega)$ is

$$G(\omega) = \begin{bmatrix} -\frac{\partial f_d}{\partial x_0} & -\frac{\partial f_d}{\partial u_0} & I_d & & & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & & -\frac{\partial f_d}{\partial x_{N-1}} & -\frac{\partial f_d}{\partial u_{N-1}} & I_d \\ \frac{\partial g}{\partial x_0} & \frac{\partial g}{\partial u_0} & & & & & \\ & \ddots & \ddots & \ddots & \ddots & \ddots & \\ & & & & \frac{\partial g}{\partial x_{N-1}} & \frac{\partial g}{\partial u_{N-1}} & \end{bmatrix} \tag{2.24}$$

In order for the Newton-KKT matrix corresponding to 2.17 to be invertible, we assume the following:

- $G(\omega)$ always has full row rank because for the first part N dynamic equality, there are identity blocks. As for the second part $N-1$ state-input equality, full rankness of the matrix $\frac{\partial g}{\partial u_k}$ will be discussed in 2.4.2.
- $\forall d \neq 0$ such that $G(\omega)d = 0$, $d^{\mathrm{T}}\nabla^2_{\omega}\mathscr{L}(\omega,\lambda)d > 0$ is yet to be validated. The calculation of $\nabla^2_{\omega}\mathscr{L}(\omega,\lambda)$ involves third-order tensor of $f_d(x,u,t)$ and $g(x,u,t)$, which are costly to get. We thus use the approximated version $B_k \approx \nabla^2_{\omega}\mathscr{L}(\omega,\lambda)$. Bearing the upcoming trust region method 2.6.1 in mind, the potential positive definiteness or indefiniteness of B_k as well as how shall we handle them are to be discussed in Chap. 3.

Following the same pattern as in standard SQP 2.15, we can solve the below QP in each iteration with $\Delta\omega = \{\Delta x_0, \Delta u_0, \Delta x_1, \Delta u_1, \dots, \Delta x_{N-1}, \Delta u_{N-1}, \Delta x_N\}$ for an optimal $\Delta\omega^*$. Final solution of 2.3 will be approximated with the update step $\omega^{i+1} = \omega^i + \Delta\omega^*$ and $\lambda^{i+1} = \lambda^{QP}$. The block-diagonality of $\nabla^2_{\omega}\mathscr{L}(\omega,\lambda)$ and stage-wise cost function of $L(x_k,u_k)$ determine the sparse and multi-stage structure:

$$\begin{aligned}
\underset{\Delta\omega}{\text{minimize}} \quad & \frac{1}{2}\sum_{k=0}^{N-1}\begin{bmatrix}\Delta x_k\\ \Delta u_k\end{bmatrix}^{\mathrm{T}} P_k \begin{bmatrix}\Delta x_k\\ \Delta u_k\end{bmatrix} + \frac{1}{2}\Delta x_N^{\mathrm{T}} P_N \Delta x_N + \sum_{k=0}^{N-1}\begin{bmatrix}\Delta x_k\\ \Delta u_k\end{bmatrix}^{\mathrm{T}} p_k + \Delta x_N^{\mathrm{T}} p_N \\
\text{subject to} \quad & \Delta x_{k+1} = A_k \Delta x_k + B_k \Delta u_k + b_k \\
& C_k \Delta x_k + D_k \Delta u_k + e_k = 0
\end{aligned} \tag{2.25}$$

with

$$P_k = \nabla^2_{(x_k,u_k)}\mathscr{L} \tag{2.26a}$$
$$P_N = \nabla^2_{x_N}\mathscr{L} \tag{2.26b}$$
$$p_k = \nabla_{(x_k,u_k)} L(x_k,u_k) \tag{2.26c}$$
$$p_N = \nabla_{x_N} E(x_N) \tag{2.26d}$$
$$A_k = \frac{\partial f_d}{\partial x_k}(x_k,u_k,t_k) \tag{2.26e}$$
$$B_k = \frac{\partial f_d}{\partial u_k}(x_k,u_k,t_k) \tag{2.26f}$$
$$b_k = f_d(x_k,u_k,t_k) - x_{k+1} \tag{2.26g}$$
$$C_k = \frac{\partial g}{\partial x_k}(x_k,u_k,t_k) \tag{2.26h}$$
$$D_k = \frac{\partial g}{\partial u_k}(x_k,u_k,t_k) \tag{2.26i}$$
$$e_k = g(x_k,u_k,t_k) \tag{2.26j}$$
$$\{x_0,u_0,x_1,u_1,\dots,x_{N-1},u_{N-1},x_N\} = \omega^i \tag{2.26k}$$

2.26k means that the guess of $\{x_0,x_1,\dots,x_N\}$, $\{u_0,u_1,\dots,u_{N-1}\}$ in 2.25 are retrieved from the current i-th iteration ω^i. The upper index i, which indicates the SQP iteration number, is not to be confused with lower index k for state and input variable in different stages.

As we have introduced beforehand, there are different choices for the discrete model f_d, thus different A_k in 2.26e and B_k in 2.26f:

- Forward Euler B.1:

$$A_k = I_d + \Delta t\frac{\partial f_c}{\partial x_k}(x_k,u_k,t_k) \tag{2.27}$$

$$B_k = \Delta t\frac{\partial f_c}{\partial u_k}(x_k,u_k,t_k) \tag{2.28}$$

- RK4 B.2:

$$A_k = I_d + \frac{\Delta t}{6}(\frac{\partial k_1}{\partial x_k} + 2\frac{\partial k_2}{\partial x_k} + 2\frac{\partial k_3}{\partial x_k} + \frac{\partial k_4}{\partial x_k}) \tag{2.29}$$

$$B_k = \frac{\Delta t}{6}(\frac{\partial k_1}{\partial u_k} + 2\frac{\partial k_2}{\partial u_k} + 2\frac{\partial k_3}{\partial u_k} + \frac{\partial k_4}{\partial u_k}) \tag{2.30}$$

where $\frac{\partial k_i}{\partial x_k}$ and $\frac{\partial k_i}{\partial u_k}, i = 1, 2, 3, 4$ are derived in Appendix B.

The resulting QP in 2.25, which is often referred as OCP QP in the literature, has sparse structure because the cost is defined stage-wise and dynamic equality constraints coupled pairs of consecutive stages. There are plenty of solvers that could exploit this characteristic. More details will be covered in implementation chapter 3.

2.4 Elimination of State-Input Equality Constraints

One way of dealing with the linearized state-input equality constraint $C_k \Delta x_k + D_k \Delta u_k + e_k = 0$ in 2.25 is to add it as two inequalities. But if the selected solver such as HPIPM [9] decides to solve the OCP QP with interior point method, several iterations will be taken to find the solution. Meanwhile, the underlying nonlinear equality constraint $g(x, u, t)$ in 2.1 for legged robots has special feature: $D_k = \frac{\partial g}{\partial u_k}(x_k, u_k, t_k)$ has full row rank. This subsection aims at taking fully advantage of these properties.

2.4.1 General Elimination with QR Decomposition

Consider the following optimization problem $x \in \mathbb{R}^n, A \in \mathbb{R}^{m\times n}, m \leq n$

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & Ax = b \quad \text{A has full row rank} \end{aligned} \tag{2.31}$$

Full rankness of A implies the feasibility of QR decomposition

$$A^{\mathrm{T}} = \begin{bmatrix} Q_1 & Q_2 \end{bmatrix} \begin{bmatrix} R \\ 0 \end{bmatrix} \tag{2.32}$$

with $Q_1 \in \mathbb{R}^{n\times m}, Q_2 \in \mathbb{R}^{n\times(n-m)}$ both have orthonormal columns, $\begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ is orthogonal and $R \in \mathbb{R}^{m\times m}$ is upper-triangular and invertible. 2.32 implies that

$$\begin{aligned} AQ_1 &= R^{\mathrm{T}} \\ AQ_2 &= 0 \end{aligned} \tag{2.33}$$

Since $\begin{bmatrix} Q_1 & Q_2 \end{bmatrix}$ is nonsingular, any $x \in \mathbb{R}^n$ that satisfies $Ax = b$ may be represented by

$$x = Q_1 x_1 + Q_2 x_2 \tag{2.34}$$

with certain $x_1 \in \mathbb{R}^m, x_2 \in \mathbb{R}^{n-m}$. Then $Ax = b \Rightarrow AQ_1x_1 + AQ_2x_2 = R^{\mathrm{T}}x_1 = b$. With AQ_1 invertible, $x_1 = (R^{\mathrm{T}})^{-1}b$, so the following holds

$$Ax = b \iff \exists x_2 \in \mathbb{R}^{n-m} \text{ s.t. } x = Q_1(R^{\mathrm{T}})^{-1}b + Q_2x_2 \tag{2.35}$$

By substituting the equality constraints with 2.35, 2.31 is equivalent to

$$\min_{x_2 \in \mathbb{R}^{n-m}} f(Q_1(R^{\mathrm{T}})^{-1}b + Q_2x_2) \tag{2.36}$$

2.4.2 Elimination Combined with OCP QP

Now we focus on the elimination of $C_k \Delta x_k + D_k \Delta u_k + e_k = 0$ where $D_k = \frac{\partial g}{\partial u_k}$ has full rank. We first apply QR decomposition to D_k.

$$D_k^{\mathrm{T}} = \begin{bmatrix} Q_k^1 & Q_k^2 \end{bmatrix} \begin{bmatrix} R_k^1 \\ 0 \end{bmatrix} \tag{2.37}$$

with $Q_k^1 \in \mathbb{R}^{m \times p}, Q_k^2 \in \mathbb{R}^{m \times (m-p)}, R_k^1 \in \mathbb{R}^{p \times p}$. Note that the upper indices of Q_k^1, Q_k^2 and R_k^1 matrices do not mean power and $g(x, u, t) \in \mathbb{R}^p$. By mimicking 2.35, the variable Δu_k can be rewritten as:

$$\begin{aligned} \Delta u_k &= Q_k^2 \Delta \tilde{u}_k + Q_k^1 (R_k^1)^{-\mathrm{T}} (-C_k \Delta x_k - e_k) \\ &= Q_k^2 \Delta \tilde{u}_k - P_{ke} (C_k \Delta x_k + e_k) \\ &= Q_k^2 \Delta \tilde{u}_k - P_{kx} \Delta x_k - P_{ke} e_k \end{aligned} \tag{2.38}$$

with $\Delta \tilde{u}_k \in \mathbb{R}^{m-p}$ and the following new notations for simplicity:

$$P_{kx} := Q_k^1 (R_k^1)^{-\mathrm{T}} C_k \qquad P_{ke} := Q_k^1 (R_k^1)^{-\mathrm{T}} \tag{2.39}$$

The new variable $\Delta \tilde{u}_k$ is not constrained anymore and $C_k \Delta x_k + D_k \Delta u_k + e_k = 0$ will be discarded. However, since the cost functions and dynamic equality constraints are all expressed in terms of Δu_k, it is necessary to re-express them with $\Delta \tilde{u}_k$.

Let's start with the cost terms. We first rewrite 2.25 into the following form such that the inner structure of P_k and p_k with respect to state and control variables are more visible and intuitive

$$\sum_{k=0}^{N-1} \left(\frac{1}{2} \begin{bmatrix} \Delta u_k \\ \Delta x_k \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} R_k & P_k \\ P_k^{\mathrm{T}} & Q_k \end{bmatrix} \begin{bmatrix} \Delta u_k \\ \Delta x_k \end{bmatrix} + \begin{bmatrix} \Delta u_k \\ \Delta x_k \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} r_k \\ q_k \end{bmatrix} \right) + \frac{1}{2} \Delta x_N^{\mathrm{T}} Q_N \Delta x_N + \Delta x_N^{\mathrm{T}} q_N \tag{2.40}$$

Note that P_k in 2.25 and 2.40 are totally different. The first order terms in 2.40 are rewritten as

$$\begin{aligned} \begin{bmatrix} \Delta u_k \\ \Delta x_k \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} r_k \\ q_k \end{bmatrix} &= \begin{bmatrix} Q_k^2 \Delta \tilde{u}_k - P_{kx} \Delta x_k - P_{ke} e_k) \\ \Delta x_k \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} r_k \\ q_k \end{bmatrix} \\ &= \left(\Delta \tilde{u}_k^{\mathrm{T}} (Q_k^2)^{\mathrm{T}} - \Delta x_k^{\mathrm{T}} P_{kx}^{\mathrm{T}} - e_k^{\mathrm{T}} P_{ke}^{\mathrm{T}} \right) r_k + \Delta x_k^{\mathrm{T}} q_k \\ &= \begin{bmatrix} \Delta \tilde{u}_k \\ \Delta x_k \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} (Q_k^2)^{\mathrm{T}} r_k \\ q_k - P_{kx}^{\mathrm{T}} r_k \end{bmatrix} - e_k^{\mathrm{T}} P_{ke}^{\mathrm{T}} r_k \\ &= \begin{bmatrix} \Delta \tilde{u}_k \\ \Delta x_k \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \tilde{r}_k^1 \\ \tilde{q}_k^1 \end{bmatrix} + C_k^1 \end{aligned} \tag{2.41}$$

with the newly defined notations

$$\begin{aligned} \tilde{r}_k^1 &:= (Q_k^2)^{\mathrm{T}} r_k \\ \tilde{q}_k^1 &:= q_k - P_{kx}^{\mathrm{T}} r_k \\ C_k^1 &:= -e_k^{\mathrm{T}} P_{ke}^{\mathrm{T}} r_k \end{aligned} \tag{2.42}$$

Then the second order term $\frac{1}{2} \Delta u_k^{\mathrm{T}} R_k \Delta u_k$ of 2.40 in terms of new $\Delta \tilde{u}_k$ from 2.38

$$\begin{aligned} \frac{1}{2} \Delta u_k^{\mathrm{T}} R_k \Delta u_k = \frac{1}{2} \Delta \tilde{u}_k^{\mathrm{T}} (Q_k^2)^{\mathrm{T}} R_k Q_k^2 \Delta \tilde{u}_k + \frac{1}{2} \Delta x_k^{\mathrm{T}} P_{kx}^{\mathrm{T}} R_k P_{kx} \Delta x_k + \frac{1}{2} e_k^{\mathrm{T}} P_{ke}^{\mathrm{T}} R_k P_{ke} e_k \\ - \frac{1}{2} \Delta \tilde{u}_k^{\mathrm{T}} (Q_k^2)^{\mathrm{T}} R_k P_{kx} \Delta x_k - \frac{1}{2} \Delta x_k^{\mathrm{T}} P_{kx}^{\mathrm{T}} R_k (Q_k^2) \Delta \tilde{u}_k \\ - \Delta \tilde{u}_k^{\mathrm{T}} (Q_k^2)^{\mathrm{T}} R_k P_{ke} e_k - \Delta x_k^{\mathrm{T}} P_{kx}^{\mathrm{T}} R_k P_{ke} e_k \end{aligned} \tag{2.43}$$

The second order term $\frac{1}{2}\Delta u_k^{\mathrm{T}} P_k \Delta x_k$ and $\frac{1}{2}\Delta x_k^{\mathrm{T}} P_k^{\mathrm{T}} \Delta u_k$ of 2.40 followed by

$$\frac{1}{2}\Delta u_k^{\mathrm{T}} P_k \Delta x_k = \frac{1}{2}\Delta \tilde{u}_k^{\mathrm{T}} (Q_k^2)^{\mathrm{T}} P_k \Delta x_k - \frac{1}{2}\Delta x_k^{\mathrm{T}} P_{kx}^{\mathrm{T}} P_k \Delta x_k - \frac{1}{2} e_k^{\mathrm{T}} P_{ke}^{\mathrm{T}} P_k \Delta x_k \tag{2.44a}$$

$$\frac{1}{2}\Delta x_k^{\mathrm{T}} P_k^{\mathrm{T}} \Delta u_k = \frac{1}{2}\Delta x_k^{\mathrm{T}} P_k^{\mathrm{T}} (Q_k^2) \Delta \tilde{u}_k - \frac{1}{2}\Delta x_k^{\mathrm{T}} P_k^{\mathrm{T}} P_{kx} \Delta x_k - \frac{1}{2}\Delta x_k^{\mathrm{T}} P_k^{\mathrm{T}} P_{ke} e_k \tag{2.44b}$$

Note there is no change to the second order term $\frac{1}{2}\Delta x_k^{\mathrm{T}} Q_k \Delta x_k$. Combining all the second order terms from 2.43,2.44a,2.44b, the second order terms of 2.40 are equivalent to:

$$\begin{aligned}
&\frac{1}{2}\Delta u_k^{\mathrm{T}} R_k \Delta u_k + \frac{1}{2}\Delta x_k^{\mathrm{T}} Q_k \Delta x_k + \frac{1}{2}\Delta u_k^{\mathrm{T}} P_k \Delta x_k + \frac{1}{2}\Delta x_k^{\mathrm{T}} P_k^{\mathrm{T}} \Delta u_k \\
&= \frac{1}{2}\Delta \tilde{u}_k^{\mathrm{T}} \tilde{R}_k \Delta \tilde{u}_k + \frac{1}{2}\Delta x_k^{\mathrm{T}} \tilde{Q}_k \Delta x_k + \frac{1}{2}\Delta \tilde{u}_k^{\mathrm{T}} \tilde{S}_k \Delta x_k + \frac{1}{2}\Delta x_k^{\mathrm{T}} \tilde{S}_k^{\mathrm{T}} \Delta \tilde{u}_k + \begin{bmatrix} \Delta \tilde{u}_k \\ \Delta x_k \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \tilde{r}_k^2 \\ \tilde{q}_k^2 \end{bmatrix} + C_k^2
\end{aligned} \tag{2.45}$$

with the newly defined notations

$$\begin{aligned}
\tilde{R}_k &:= (Q_k^2)^{\mathrm{T}} R_k Q_k^2 \\
\tilde{Q}_k &:= Q_k - P_{kx}^{\mathrm{T}} P_k - P_k^{\mathrm{T}} P_{kx} + P_{kx}^{\mathrm{T}} R_k P_{kx} \\
\tilde{S}_k &:= (Q_k^2)^{\mathrm{T}} P_k - (Q_k^2)^{\mathrm{T}} R_k P_{kx} \\
\tilde{r}_k^2 &:= -(Q_k^2)^{\mathrm{T}} R_k P_{ke} e_k \\
\tilde{q}_k^2 &:= -P_k^{\mathrm{T}} P_{ke} e_k - P_{kx}^{\mathrm{T}} R_k P_{ke} e_k \\
C_k^2 &:= \frac{1}{2} e_k^{\mathrm{T}} P_{ke}^{\mathrm{T}} R_k P_{ke} e_k
\end{aligned} \tag{2.46}$$

The new first-order terms 2.41 and second-order terms 2.45 composite the new cost expressed in $\Delta \tilde{u}_k$

$$\sum_{k=0}^{N-1} \left(\frac{1}{2} \begin{bmatrix} \Delta \tilde{u}_k \\ \Delta x_k \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \tilde{R}_k & \tilde{S}_k \\ \tilde{P}_k^{\mathrm{T}} & \tilde{Q}_k \end{bmatrix} \begin{bmatrix} \Delta \tilde{u}_k \\ \Delta x_k \end{bmatrix} + \begin{bmatrix} \Delta \tilde{u}_k \\ \Delta x_k \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \tilde{r}_k \\ \tilde{q}_k \end{bmatrix} + C_k \right) + \frac{1}{2}\Delta x_N^{\mathrm{T}} Q_N \Delta x_N + \Delta x_N^{\mathrm{T}} q_N \tag{2.47}$$

with the newly defined notations

$$\begin{aligned}
\tilde{r}_k &:= \tilde{r}_k^1 + \tilde{r}_k^2 \\
\tilde{q}_k &:= \tilde{q}_k^1 + \tilde{q}_k^2 \\
C_k &= C_k^1 + C_k^2
\end{aligned} \tag{2.48}$$

The substitution is finished in the cost function. Now we plug the new variable $\Delta u_k = Q_k^2 \Delta \tilde{u}_k - P_{kx} \Delta x_k - P_{ke} e_k$ into the dynamic equality constraints

$$\begin{aligned}
\Delta x_{k+1} &= A_k \Delta x_k + B_k \Delta u_k + b_k \\
&= A_k \Delta x_k + B_k Q_k^2 \Delta \tilde{u}_k - B_k P_{kx} \Delta x_k - B_k P_{ke} e_k + b_k \\
&= (A_k - B_k P_{kx}) \Delta x_k + B_k Q_k^2 \Delta \tilde{u}_k + b_k - B_k P_{ke} e_k \\
&= \tilde{A}_k \Delta x_k + \tilde{B}_k \Delta \tilde{u}_k + \tilde{b}_k
\end{aligned} \tag{2.49}$$

with the newly defined notations

$$\begin{aligned}
\tilde{A}_k &:= (A_k - B_k P_{kx}) \\
\tilde{B}_k &:= B_k Q_k^2 \\
\tilde{b}_k &:= b_k - B_k P_{ke} e_k
\end{aligned} \tag{2.50}$$

Until now, we have finished replacing the old variable Δu_k with new variable $\Delta \tilde{u}_k$ so the state-input equality constraints can be always satisfied by the reparametrization. 2.25 is equivalent to

$$\begin{aligned} \underset{\Delta\tilde{\omega}}{\text{minimize}} \quad & \frac{1}{2}\sum_{k=0}^{N-1}\begin{bmatrix}\Delta x_k\\ \Delta\tilde{u}_k\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}\tilde{Q}_k & \tilde{P}_k^{\mathrm{T}}\\ \tilde{S}_k & \tilde{R}_k\end{bmatrix}\begin{bmatrix}\Delta x_k\\ \Delta\tilde{u}_k\end{bmatrix} + \frac{1}{2}\Delta x_N^{\mathrm{T}} Q_N \Delta x_N + \sum_{k=0}^{N-1}\begin{bmatrix}\Delta x_k\\ \Delta\tilde{u}_k\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}\tilde{r}_k\\ \tilde{q}_k\end{bmatrix} + \Delta x_N^{\mathrm{T}} q_N \\ \text{subject to} \quad & \Delta x_{k+1} = \tilde{A}_k\Delta x_k + \tilde{B}_k\Delta\tilde{u}_k + \tilde{b}_k \end{aligned} \tag{2.51}$$

with $\Delta\tilde{\omega} = \{\Delta x_0, \Delta\tilde{u}_0, \Delta x_1, \Delta\tilde{u}_1, \ldots, \Delta x_{N-1}, \Delta\tilde{u}_{N-1}, \Delta x_N\}$, the new update step. The update rules are $\tilde{\omega}^{i+1} = \tilde{\omega}^i + \Delta\tilde{\omega}$, $\tilde{\lambda}^{i+1} = \tilde{\lambda}^{QP}$. However, it is worthy noting that a transformation from $\Delta\tilde{u}$ in a lower dimensional space to Δu with the right dimension of input is necessary with the help of 2.38.

2.5 Review of Riccati Recursion for Full Version of LQR

We call 2.51 the full version of LQR problem since traditional LQR does not include linear cost terms r_k, q_k and second order cross cost term P_k. Here we present the backward Riccati recursion for the full LQR. Though this is not something new, it acts as a prerequisite for the following trust region section. Rewriting the objective function of 2.51 by deleting all the tilde symbols for simplicity

$$\sum_{k=0}^{N-1}\frac{1}{2}\begin{bmatrix}\Delta u_k\\ \Delta x_k\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}R_k & P_k\\ P_k^{\mathrm{T}} & Q_k\end{bmatrix}\begin{bmatrix}\Delta u_k\\ \Delta x_k\end{bmatrix} + \begin{bmatrix}\Delta u_k\\ \Delta x_k\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}r_k\\ q_k\end{bmatrix} + \frac{1}{2}\Delta x_N^{\mathrm{T}} Q_N\Delta x_N + \Delta x_N^{\mathrm{T}} q_N \tag{2.52}$$

Let's assume that the cost-to-go starting from stage $k+1$ is

$$V_{k+1}(z) = \frac{1}{2}z^{\mathrm{T}} S_{k+1} z + z^{\mathrm{T}} s_{k+1} + c_{k+1} \tag{2.53}$$

Then the cost-to-go starting from stage k is

$$\begin{aligned} V_k(z) &= \frac{1}{2}z^{\mathrm{T}} Q_k z + z^{\mathrm{T}} q_k + \beta \\ \beta &= \min_w\Big(\frac{1}{2}w^{\mathrm{T}} P_k z + \frac{1}{2}z^{\mathrm{T}} P_k^{\mathrm{T}} w + \frac{1}{2}w^{\mathrm{T}} R_k w + w^{\mathrm{T}} r_k + V_{k+1}(A_k z + B_k w + b_k)\Big) \end{aligned} \tag{2.54}$$

Considering 2.53, the last term in 2.54 is

$$\begin{aligned} V_{k+1}(A_k z + B_k w + b_k) &= \frac{1}{2}(A_k z + B_k w + b_k)^{\mathrm{T}} S_{k+1}(A_k z + B_k w + b_k) + (A_k z + B_k w + b_k)^{\mathrm{T}} s_{k+1} + c_{k+1} \\ &= \frac{1}{2}w^{\mathrm{T}} B_k^{\mathrm{T}} S_{k+1} B_k w + w^{\mathrm{T}}(B_k^{\mathrm{T}} S_{k+1} A_k z + B_k^{\mathrm{T}} S_{k+1} b_k + B_k^{\mathrm{T}} s_{k+1}) + \alpha \end{aligned} \tag{2.55}$$

α is irrelavent to ω. It is defined as

$$\alpha := \frac{1}{2}z^{\mathrm{T}} A_k^{\mathrm{T}} S_{k+1} A_k z + z^{\mathrm{T}}(A_k^{\mathrm{T}} S_{k+1} b_k + A_k^{\mathrm{T}} s_{k+1}) + \frac{1}{2}b_k^{\mathrm{T}} S_{k+1} b_k + b_k^{\mathrm{T}} s_{k+1} + c_{k+1} \tag{2.56}$$

Let's focus on the β part in 2.54 here:

$$\begin{aligned} & \min_w \quad \frac{1}{2}w^{\mathrm{T}} P_k z + \frac{1}{2}z^{\mathrm{T}} P_k^{\mathrm{T}} w + \frac{1}{2}w^{\mathrm{T}} R_k w + w^{\mathrm{T}} r_k + V_{k+1}(A_k z + B_k w + b_k) \\ \rightarrow \quad & \min_w \quad \frac{1}{2}w^{\mathrm{T}}(R_k + B_k^{\mathrm{T}} S_{k+1} B_k) w + w^{\mathrm{T}}(P_k z + B_k^{\mathrm{T}} S_{k+1} A_k z + r_k + B_k^{\mathrm{T}} S_{k+1} b_k + B_k^{\mathrm{T}} s_{k+1}) + \alpha \end{aligned} \tag{2.57}$$

Recall the following unconstrained QP problem and its optimal solution:

$$\begin{aligned}\min_x \quad & \frac{1}{2}x^{\mathrm{T}}Qx + x^{\mathrm{T}}q \\ x^* &= -Q^{-1}q \\ \min_x \quad & \frac{1}{2}x^{\mathrm{T}}Qx + x^{\mathrm{T}}q = -\frac{1}{2}q^{\mathrm{T}}Q^{-1}q\end{aligned} \tag{2.58}$$

So the optimum w^* corresponding to the β part is:

$$\begin{aligned}w^* =& -(R_k + B_k^{\mathrm{T}}S_{k+1}B_k)^{-1}(P_k + B_k^{\mathrm{T}}S_{k+1}A_k)z \\ & -(R_k + B_k^{\mathrm{T}}S_{k+1}B_k)^{-1}(r_k + B_k^{\mathrm{T}}S_{k+1}b_k + B_k^{\mathrm{T}}s_{k+1}) \\ =& K_k z + k_k\end{aligned} \tag{2.59}$$

With the feedback matrix K_k and feedforward term k_k defined to be

$$K_k = -(R_k + B_k^{\mathrm{T}}S_{k+1}B_k)^{-1}(P_k + B_k^{\mathrm{T}}S_{k+1}A_k) \tag{2.60}$$

$$k_k = -(R_k + B_k^{\mathrm{T}}S_{k+1}B_k)^{-1}(r_k + B_k^{\mathrm{T}}S_{k+1}b_k + B_k^{\mathrm{T}}s_{k+1}) \tag{2.61}$$

Plugging the w^* back, the term β is

$$\begin{aligned}\beta =& -\frac{1}{2}z^{\mathrm{T}}(P_k + B_k^{\mathrm{T}}S_{k+1}A_k)^{\mathrm{T}}(R_k + B_k^{\mathrm{T}}S_{k+1}B_k)^{-1}(P_k + B_k^{\mathrm{T}}S_{k+1}A_k)z \\ & - z^{\mathrm{T}}(P_k + B_k^{\mathrm{T}}S_{k+1}A_k)^{\mathrm{T}}(R_k + B_k^{\mathrm{T}}S_{k+1}B_k)^{-1}(r_k + B_k^{\mathrm{T}}S_{k+1}b_k + B_k^{\mathrm{T}}s_{k+1}) \\ & - \frac{1}{2}(r_k + B_k^{\mathrm{T}}S_{k+1}b_k + B_k^{\mathrm{T}}s_{k+1})^{\mathrm{T}}(R_k + B_k^{\mathrm{T}}S_{k+1}B_k)^{-1}(r_k + B_k^{\mathrm{T}}S_{k+1}b_k + B_k^{\mathrm{T}}s_{k+1}) + \alpha\end{aligned} \tag{2.62}$$

Now go back to $V_k(z)$. Since $V_k(z) = \frac{1}{2}z^{\mathrm{T}}S_kz + z^{\mathrm{T}}s_k + c_k = \frac{1}{2}z^{\mathrm{T}}Q_kz + z^{\mathrm{T}}q_k + \beta$, by matching term-to-term in $V_k(z)$, we will end up with the update formula for S_k, s_k, c_k from $S_{k+1}, s_{k+1}, c_{k+1}$:

$$S_k = Q_k + A_k^{\mathrm{T}}S_{k+1}A_k - (P_k + B_k^{\mathrm{T}}S_{k+1}A_k)^{\mathrm{T}}(R_k + B_k^{\mathrm{T}}S_{k+1}B_k)^{-1}(P_k + B_k^{\mathrm{T}}S_{k+1}A_k) \tag{2.63}$$

$$\begin{aligned}s_k =& q_k + A_k^{\mathrm{T}}S_{k+1}b_k + A_k^{\mathrm{T}}s_{k+1} \\ & - (P_k + B_k^{\mathrm{T}}S_{k+1}A_k)^{\mathrm{T}}(R_k + B_k^{\mathrm{T}}S_{k+1}B_k)^{-1}(r_k + B_k^{\mathrm{T}}S_{k+1}b_k + B_k^{\mathrm{T}}s_{k+1})\end{aligned} \tag{2.64}$$

$$\begin{aligned}c_k =& \frac{1}{2}b_k^{\mathrm{T}}S_{k+1}b_k + b_k^{\mathrm{T}}s_{k+1} + c_{k+1} \\ & - \frac{1}{2}(r_k + B_k^{\mathrm{T}}S_{k+1}b_k + B_k^{\mathrm{T}}s_{k+1})^{\mathrm{T}}(R_k + B_k^{\mathrm{T}}S_{k+1}B_k)^{-1}(r_k + B_k^{\mathrm{T}}S_{k+1}b_k + B_k^{\mathrm{T}}s_{k+1})\end{aligned} \tag{2.65}$$

The backward Riccati recursion starts from

$$S_N = Q_N \tag{2.66}$$

$$s_N = q_N \tag{2.67}$$

$$c_N = 0 \tag{2.68}$$

2.6 Trust Region in Sub-QP Problem

We first introduce the general setup of trust region method in nonlinear optimization problems. Then we show the combination of TR philosophy with 2.51.

2.6.1 Trust Region Subproblems

Trust region and line search methods are both globalization strategies that return update step for optimizing functions. The step is calculated based on a quadratic approximation of the objective function.

Assume that we are minimizing an unconstrained function $f(x)$ and a current guess x_k is given. The quadratic model $m_k(p)$ is constructed from Tayler expansion:

$$m_k(p) = f(x_k) + \nabla f(x_k)^{\mathrm{T}} p + \frac{1}{2} p^{\mathrm{T}} B_k p \tag{2.69}$$

The matrix related to second order term B_k can either be the true Hessian matrix $\nabla^2 f(x_k)$ or some approximation updated via BFGS law. In the former case, the approximation error is $\mathcal{O}(\|p\|^3)$.

The name of "Trust Region Method" originates from that the quadratic model $m_k(p)$ may approximate the true function $f(x)$ with different quality from case to case. So it is quite natural to limit our model within a fixed range, which is called "Trust Region Radius" and the optimization only happens within the region. If model is reliable and accurately mimicing the behaviours of objective function, trust region radius can be increased to allow more ambitious steps. On the contrary, if the returned step is performing poorly, we will abandon it and shrink the radius. Mathematically speaking, we would like to solve:

$$\begin{aligned} \min_{p \in \mathbb{R}^n} \quad & m_k(p) = f(x_k) + \nabla f(x_k)^{\mathrm{T}} p + \frac{1}{2} p^{\mathrm{T}} B_k p \\ \text{subject to} \quad & \|p\| \leq \delta_k \end{aligned} \tag{2.70}$$

The sub-index k indicates we are in the k-th iteration of optimization. δ_k is the trust region radius. $\|p\|$ may represent ℓ_1, ℓ_2, ℓ_∞ norms. In the literature, polynomial time algorithm exitsts if ℓ_2 norm is used in 2.70. We present the necessary and sufficient conditions for p^* to be the global solution of $\min_{p \in \mathbb{R}^n} m(p) = f + \nabla g^{\mathrm{T}} p + \frac{1}{2} p^{\mathrm{T}} B p \quad \text{s.t.} \|p\|_2 \leq \delta$: there exists $\lambda \geq 0$ such that

$$\begin{aligned} (B + \lambda I) p^* &= -g \\ \lambda(\delta - \|p^*\|_2) &= 0 \\ (B + \lambda I) &\geq 0 \end{aligned} \tag{2.71}$$

The proof of such claim can be found in Sec 4.3 in [30] or Sec 7.2 in [31]. If p^* lies interiorly, then $\lambda = 0$. If p^* lies on the boundary, then we may use Newton's method for root finding to solver $\|p(\lambda)\|_2^2 - \delta^2 = 0$ to find λ. In such case, ℓ_2 norm trust region has a similar taste with the Levenburg-Marquardt algorithm.

If ℓ_1 or ℓ_∞ norm is used, then we are unfortunate to face a NP hard problem since both of them have sharp corners where iterative step may get trapped. However, ℓ_∞ still remains a popular choice over ℓ_1 since box bounds constraints are easier to check.

Once 2.70 is solved (whatever norm type is utilized), we then measure the quality of update step p_k^* by the following equation:

$$\rho_k = \frac{f(x_k) - f(x_k + p_k^*)}{m_k(0) - m_k(p_k^*)} \tag{2.72}$$

ρ_k is the division result between actual reduction and predicted reduction. Since the optimal step p_k^* comes from minimizing m_k, the predicted reduction is always positive. If $\rho_k < 0$, then we need to reject the step because we are not making any progress in the original function $f(x)$ and shrink the radius in the next iteration. If ρ_k is close to 1, which is a great indication that our model is approximating well, we will accept the step and enlarge trust region radius. If ρ_k is lies in between these two conditions, we may mildly enlarge or shrink the radius and decide from case to case whether to accept the step or not.

2.6.2 An Iterative Algorithm for QP with Trust Region ℓ_2 Norm

Ideally speaking, we would like to solve the following equations (which are equivalent to 2.70) with exact λ^M, s^M, but most time an approximate solution λ^+, s^+ suffices.

$$\begin{aligned} H(\lambda^M)s^M &= -g \\ H(\lambda^M) &= H + \lambda^M I \geq 0 \\ \lambda^M &\geq 0 \\ \lambda^M(\|s^M\|_2 - \delta) &= 0 \end{aligned} \tag{2.73}$$

Algorithm 7.3.4 from [31] is one of the algorithms that runs iteratively and finally provides us an approximate solution λ^+, s^+. It starts with a guess $\lambda \in (\lambda^L, \lambda^U)$. Some of the sets are defined as $\mathscr{N} = \{\lambda | \lambda \leq \max[0, -\lambda_1]\}, \mathscr{L} = \{\lambda | \max[0, -\lambda_1] < \lambda \leq \lambda^M\}, \mathscr{G} = \{\lambda | \lambda > \lambda^M\}$. $\mathscr{N}$ means not feasible, $\mathscr{L}, \mathscr{G}$ means less / greater than the model minimizer. λ^M is the solution to 2.73, λ_1 is the unknown smallest eigenvalue of H.

1. Attempt to factorize $H(\lambda) = H + \lambda I = LL^T$ if not already tried.
   (a) If the factorization succeeds, then $\lambda \in \mathscr{F}$. Solve $LL^T s = -g$. If $\|s\|_2 < \delta$, then $\lambda \in \mathscr{G}$; check for interior convergence. Otherwise (i.e. the factorization fails),$\lambda \in \mathscr{L}$. Otherwise, $\lambda \in \mathscr{N}$.
2. If $\lambda \in \mathscr{G}$, replace λ^{U} by λ. Otherwise, replace λ^{L} by λ.
3. If $\lambda \in \mathscr{F}$
   (a) Solve $Lw = s$ and set $\lambda^+ = \lambda + \left(\frac{\|s\|_2 - \delta}{\delta}\right)\left(\frac{\|s\|_2^2}{\|w\|_2^2}\right)$. (This is Newton's method update.)
   (b) If $\lambda \in \mathscr{G}$
      i. Use the LINPACK method to find a unit vector u to make $\langle u, H(\lambda)u\rangle$ small.
      ii. Replace λ^{L} by $\max\left[\lambda^{\mathrm{L}}, \lambda - \langle u, H(\lambda)u\rangle\right]$
      iii. Find the root α of the equation $\|s + \alpha u\|_2 = \delta$ which makes the model $q(s + \alpha u)$ smallest, and replace s by $s + \alpha u$ Otherwise (i.e., $\lambda \notin \mathscr{F}$)
   (c) Use the partial factorization to find ϵ and v such that $\left(H(\lambda) + \epsilon e_k e_k^T\right) v = 0$
   (d) Replace λ^{L} by $\max\left[\lambda^{\mathrm{L}}, \lambda + \frac{\epsilon}{\|v\|_2^2}\right]$.
4. Check for termination.
5. If $\lambda \in \mathscr{L}$ and $g \neq 0$, replace λ with λ^+. Otherwise, if $\lambda \in \mathscr{G}$, attempt to factorize $H\left(\lambda^+\right) = LL^T$.

(a) If the factorization succeeds, then $\lambda^+ \in \mathscr{L}$. Replace λ with λ^+.

(b) Otherwise, $\lambda^+ \in \mathscr{N}$. Replace λ^{L} by $\max\left[\lambda^{\mathrm{L}}, \lambda^+\right]$, check λ^{L} for interior convergence; otherwise replace λ by $\max\left[\sqrt{\lambda^{\mathrm{L}}\lambda^{\mathrm{U}}}, \lambda^{\mathrm{L}} + \theta\left(\lambda^{\mathrm{U}} - \lambda^{\mathrm{L}}\right)\right]$.

Interested readers may check the LINPACK method and termination conditions themselves. It is necessary to include this part since the algorithm acts as a prerequiste to the coming trust region method applied to Riccati recursion session.

2.6.3 Trust Region with OCP QP

Building on top of 2.51, we apply trust region thought and formulate the following:

$$\begin{aligned}
\min_{\Delta u_k, \Delta x_k} \quad & \sum_{k=0}^{N-1} \frac{1}{2} \begin{bmatrix} \Delta u_k \\ \Delta x_k \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} R_k & P_k \\ P_k^{\mathrm{T}} & Q_k \end{bmatrix} \begin{bmatrix} \Delta u_k \\ \Delta x_k \end{bmatrix} + \begin{bmatrix} \Delta u_k \\ \Delta x_k \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} r_k \\ q_k \end{bmatrix} + \frac{1}{2} \Delta x_N^{\mathrm{T}} Q_N \Delta x_N + \Delta x_N^{\mathrm{T}} q_N \\
\text{subject to} \quad & \Delta x_{k+1} = A_k \Delta x_k + B_k \Delta u_k + b_k \\
& \|\Delta u_k\| \le \delta_k^u, \quad \|\Delta x_k\| \le \delta_k^x
\end{aligned} \tag{2.74}$$

It is still unclear whether we should include trust region radius limit on $\delta x, \delta u$ at the same time or not. For generality, 2.74 includes both. However, as later chapters will show, it seems that inclusion of δu suffice.

Depending on ℓ_2 or ℓ_∞ is used, there are different approaches to solve 2.74, which will be discussed in the coming subsections.

Trust Region with OCP QP with ℓ_2 Norm

Given Δx_0, we would like to

$$\begin{aligned}
\min_{\Delta u_k, \Delta x_k} \quad & \sum_{k=0}^{N-1} \frac{1}{2} \begin{bmatrix} \Delta u_k \\ \Delta x_k \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} R_k & P_k \\ P_k^{\mathrm{T}} & Q_k \end{bmatrix} \begin{bmatrix} \Delta u_k \\ \Delta x_k \end{bmatrix} + \begin{bmatrix} \Delta u_k \\ \Delta x_k \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} r_k \\ q_k \end{bmatrix} + \frac{1}{2} \Delta x_N^{\mathrm{T}} Q_N \Delta x_N + \Delta x_N^{\mathrm{T}} q_N \\
\text{subject to} \quad & \Delta x_{k+1} = A_k \Delta x_k + B_k \Delta u_k + b_k \\
& \|\Delta u_k\|_2 \le \delta_k
\end{aligned} \tag{2.75}$$

Here we only use the trust region radius to limit δu_k because this is the real decision variable. Without proof (Some effort regarding to it can be found in Appendix C), we claim that there exists $\{\lambda_k^*\}_{k=0}^{N-1}$ such that the following equations hold. Note that here there is no additivity of $\lambda_k I$ into S_k because we are limiting the control variables only.

$$
\begin{aligned}
S_N^* &= Q_N, s_N^* = q_N\\
H_k^* &= R_k + B_k^{\mathrm{T}} S_{k+1}^* B_k\\
\mathbb{H}_k^* &= H_k^* + \lambda_k^* I\\
G_k^* &= P_k + B_k^{\mathrm{T}} S_{k+1}^* A_k\\
L_k^* &= -(\mathbb{H}_k^*)^{-1} G_k^*\\
S_k^* &= Q_k + A_k^{\mathrm{T}} S_{k+1}^* A_k + (L_k^*)^{\mathrm{T}} H_k^* L_k^* + (L_k^*)^{\mathrm{T}} G_k^* + (G_k^*)^{\mathrm{T}} L_k^*\\
&= Q_k + A_k^{\mathrm{T}} S_{k+1}^* A_k + (G_k^*)^{\mathrm{T}} (\mathbb{H}_k^*)^{-1} H_k^* (\mathbb{H}_k^*)^{-1} G_k^* - (G_k^*)^{\mathrm{T}} (\mathbb{H}_k^*)^{-1} (G_k^*) - (G_k^*)^{\mathrm{T}} (\mathbb{H}_k^*)^{-1} (G_k^*)\\
g_k^* &= r_k + B_k^{\mathrm{T}} S_{k+1}^* b_k + B_k^{\mathrm{T}} s_{k+1}^*\\
l_k^* &= -(\mathbb{H}_k^*)^{-1} g_k^*\\
s_k^* &= q_k + A_k^{\mathrm{T}} S_{k+1}^* b_k + A_k^{\mathrm{T}} s_{k+1}^* + (L_k^*)^{\mathrm{T}} H_k^* l_k^* + (L_k^*)^{\mathrm{T}} g_k^* + (G_k^*)^{\mathrm{T}} l_k^*\\
&= q_k + A_k^{\mathrm{T}} S_{k+1}^* b_k + A_k^{\mathrm{T}} s_{k+1}^* + (G_k^*)^{\mathrm{T}} (\mathbb{H}_k^*)^{-1} H_k^* (\mathbb{H}_k^*)^{-1} g_k^* - (G_k^*)^{\mathrm{T}} (\mathbb{H}_k^*)^{-1} g_k^* - (G_k^*)^{\mathrm{T}} (\mathbb{H}_k^*)^{-1} g_k^*
\end{aligned}
\tag{2.76}
$$

Starting with given Δx_0^*, the optimal solution set of controls and states is:

$$
\begin{aligned}
\Delta u_k^* &= L_k^* \Delta x_k^* + l_k^*\\
\Delta x_{k+1}^* &= A_k \Delta x_k^* + B_k \Delta u_k^* + b_k
\end{aligned}
\tag{2.77}
$$

For each stage $k \in \{0, \dots, N-1\}$, the following are satisfied:

$$
\begin{aligned}
\mathbb{H}_k^* &\geq 0\\
\lambda_k^* (\|\Delta u_k^*\| - \delta_k) &= 0
\end{aligned}
\tag{2.78}
$$

We propose the following iterative algorithm to find the optimal $\{\lambda_k^*\}$. Assume in the current iteration, we have $\{\lambda_k^L \leq \lambda_k \leq \lambda_k^U\}_{k=0}^{N-1}, \{\delta_k\}_{k=0}^{N-1}, \{\theta_k\}_{k=0}^{N-1}$

1. Try to finish the backward Riccati recursion using solvers (e.g. HPIPM) in the following way for all stages at once. 2.75 can be equivalently viewed as a sparse QP trust region problem just as 2.70. Note the riccati recursion is the same as factorizing the giant Hessian matrix, but only in a step-by-step sense. A proof of such result can be found in [32]. So this is similar to step 1 in Algorithm 7.3.4. The updating formula very similar with those in 2.6.3, but different in the sense that $\tilde{}$ means current guess and * means the optimal solution.

$$
\begin{aligned}
\tilde{S}_N &= Q_N, \tilde{s}_N = q_N\\
\tilde{H}_k &= R_k + B_k^{\mathrm{T}} \tilde{S}_{k+1} B_k\\
\tilde{\mathbb{H}}_k &= \tilde{H}_k + \lambda_k I\\
\tilde{G}_k &= P_k + B_k^{\mathrm{T}} \tilde{S}_{k+1} A_k\\
\tilde{L}_k &= -\tilde{\mathbb{H}}_k^{-1} \tilde{G}_k\\
\tilde{S}_k &= Q_k + A_k^{\mathrm{T}} \tilde{S}_{k+1} A_k + \tilde{L}_k^{\mathrm{T}} \tilde{H}_k \tilde{L}_k + \tilde{L}_k^{\mathrm{T}} \tilde{G}_k + \tilde{G}_k^{\mathrm{T}} \tilde{L}_k\\
&= Q_k + A_k^{\mathrm{T}} \tilde{S}_{k+1} A_k + \tilde{G}_k^{\mathrm{T}} \tilde{\mathbb{H}}_k^{-1} \tilde{H}_k \tilde{\mathbb{H}}_k^{-1} \tilde{G}_k - \tilde{G}_k^{\mathrm{T}} \tilde{\mathbb{H}}_k^{-1} \tilde{G}_k - \tilde{G}_k^{\mathrm{T}} \tilde{\mathbb{H}}_k^{-1} \tilde{G}_k\\
\tilde{g}_k &= r_k + B_k^{\mathrm{T}} \tilde{S}_{k+1} b_k + B_k^{\mathrm{T}} \tilde{s}_{k+1}\\
\tilde{l}_k &= -\tilde{\mathbb{H}}_k^{-1} \tilde{g}_k\\
\tilde{s}_k &= q_k + A_k^{\mathrm{T}} \tilde{S}_{k+1} b_k + A_k^{\mathrm{T}} \tilde{s}_{k+1} + \tilde{L}_k^{\mathrm{T}} \tilde{H}_k \tilde{l}_k + \tilde{L}_k^{\mathrm{T}} \tilde{g}_k + \tilde{G}_k^{\mathrm{T}} \tilde{l}_k\\
&= q_k + A_k^{\mathrm{T}} \tilde{S}_{k+1} b_k + A_k^{\mathrm{T}} \tilde{s}_{k+1} + \tilde{G}_k^{\mathrm{T}} \tilde{\mathbb{H}}_k^{-1} \tilde{H}_k \tilde{\mathbb{H}}_k^{-1} \tilde{g}_k - \tilde{G}_k^{\mathrm{T}} \tilde{\mathbb{H}}_k^{-1} \tilde{g}_k - \tilde{G}_k^{\mathrm{T}} \tilde{\mathbb{H}}_k^{-1} \tilde{g}_k
\end{aligned}
\tag{2.79}
$$

2. Assume the riccati recursion of HPIPM succeeds, which means $\lambda_k \in \mathscr{F}_k$ for all stage. We then run a forward pass to calculate the optimal $\{\Delta x_k, \Delta u_k\}$ corresponding to the current set of $\{\lambda_k\}$.

3. In this step, we calculate the new $\{\Delta x_k^+, \Delta u_k^+\}$ and $\{\lambda_k^+\}$.

   (a) Start with $\delta x_0^+ = \delta x_0 = 0$. At stage k, we would like to use trust region method to solve the following (Note that here we already have the value of δx_k^+, so there is no concern about optimizing with respect to unsettled decision variables):

$$\begin{aligned} \min_{\Delta u_k} \quad & \frac{1}{2}\Delta u_k^{\mathrm{T}}(R_k + B_k^{\mathrm{T}}\tilde{S}_{k+1}B_k)\Delta u_k + \left(r_k + B_k^{\mathrm{T}}\tilde{S}_{k+1}b_k + B_k^{\mathrm{T}}\tilde{s}_{k+1} + (P_k + B_k^{\mathrm{T}}\tilde{S}_{k+1}A_k)\Delta x_k^+\right)^{\mathrm{T}}\Delta u_k \\ \text{subject to} \quad & \|\Delta u_k\|_2 \leq \delta_k \end{aligned} \tag{2.80}$$

   (b) We denote $\mathscr{H}_k = R_k + B_k^{\mathrm{T}}\tilde{S}_{k+1}B_k$ and $\mathscr{g}_k = r_k + B_k^{\mathrm{T}}\tilde{S}_{k+1}b_k + B_k^{\mathrm{T}}\tilde{s}_{k+1} + (P_k + B_k^{\mathrm{T}}\tilde{S}_{k+1}A_k)\Delta x_k^+$. Note that these two variables are functions of $\lambda_{k+1}, \ldots, \lambda_N$.

      i. In the backward Riccati pass, we have already factorized $\mathscr{H}_k(\lambda_k) = \mathscr{H}_k + \lambda_k I = L_k^{\mathrm{T}}L_k$. Since we assume it succeeds, $\lambda_k \in \mathscr{F}_k$. Then solve $L_k^{\mathrm{T}}L_k\Delta u_k = -\mathscr{g}_k$. If $\|\Delta u_k\| < \delta_k$, then $\lambda_k \in \mathscr{G}_k$; check for interior convergence. If $\|\Delta u_k\| > \delta_k$, $\lambda_k \in \mathscr{L}_k$.

      ii. From now on, just follow the steps in Alogrithm 7.3.4.

   (c) After the algorithm, we will have a new set of $\{\lambda_k^+\}_{k=0}^N$ as well as new bounds $\lambda_k^{L+}, \lambda_k^{U+}$. Also, we have retrieved Δu_k^+ that corresponds to λ_k^+. We calculate the next state $\Delta x_{k+1}^+ = A_k\Delta x_k^+ + B_k\Delta u_k^+ + b_k$. This method must be done sequentially.

   One potential point worth investigating is that we may consider to add the sensitivity of future $\lambda_{k+1}, \ldots, \lambda_N$ to the update formula of λ_k

4. Go back to step 1 and loop until termination.

Some discussion about this set-up:

- If the entire riccati recursion of HPIPM fails, we don't know which λ_k cause the problem and thus a violent way to deal with it is to simply increase the lower bound for every λ_k.

- This proposed algorithm is not to be confused with the trust region update rule for $\{\delta_k\}_{k=0}^N$. The above algorithm is mimicing Algorithm 7.3.4 in [31] under Riccati recursion (OCP structure) setting that instead of $\lambda \in \mathbb{R} \leftrightarrow s \in \mathbb{R}^n$, we are dealing with $\{\lambda_k \in \mathbb{R}^n\}_{k=0}^N \leftrightarrow \{\Delta u_k \in \mathbb{R}^m\}_{k=0}^N$. So the above iterative algorithm works for finding the optimal $\{\Delta u_k \in \mathbb{R}^m\}_{k=0}^N$ corresponding to the set of $\{\delta_k\}_{k=0}^N$. The trust region update rule means $\{\delta_k^i\}_{k=0}^N \rightarrow \{\delta_k^{i+1}\}_{k=0}^N$ based on the ratio between predicted reduction and actualt reduction in 2.72.

- The constraints on each stage $\|\Delta u_k\| \leq \delta_k$ might seem to be overwhelmed. More importantly, according to the author's best knowledge, there exists no theoratical proof for stage-wise ℓ_2 trust region LQR setup as in 2.75 stating that the assumption in 2.6.3 is correct. However, if we only limit the entire ℓ_2 norm of delta input variables, we will end up with only one inequality constraint. According to the deduction 2.71 presented above, for the following problem,

$$\begin{aligned}\min_{\Delta u_k,\Delta x_k} \quad & \sum_{k=0}^{N-1} \frac{1}{2}\begin{bmatrix}\Delta u_k\\ \Delta x_k\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}R_k & P_k\\ P_k^{\mathrm{T}} & Q_k\end{bmatrix}\begin{bmatrix}\Delta u_k\\ \Delta x_k\end{bmatrix} + \begin{bmatrix}\Delta u_k\\ \Delta x_k\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}r_k\\ q_k\end{bmatrix} + \frac{1}{2}\Delta x_N^{\mathrm{T}} Q_N \Delta x_N + \Delta x_N^{\mathrm{T}} q_N \\ \text{subject to} \quad & \Delta x_{k+1} = A_k \Delta x_k + B_k \Delta u_k + b_k \\ & \| \begin{bmatrix}\Delta u_0\\ \Delta u_1\\ \vdots\\ \Delta u_{k-1}\end{bmatrix} \|_2 \le \delta^u\end{aligned} \tag{2.81}$$

There exists $\lambda^* \in \mathbb{R}$ such that (On the contrary, recall in 2.75 we assume the existence of $\{\lambda_k^*\}_{k=0}^{N-1}$)

$$\begin{aligned} S_N^* &= Q_N, s_N^* = q_N \\ H_k^* &= R_k + B_k^{\mathrm{T}} S_{k+1}^* B_k \\ \mathbb{H}_k^* &= H_k^* + \lambda^* I \\ G_k^* &= P_k + B_k^{\mathrm{T}} S_{k+1}^* A_k \\ L_k^* &= -(\mathbb{H}_k^*)^{-1} G_k^* \\ S_k^* &= Q_k + A_k^{\mathrm{T}} S_{k+1}^* A_k + (L_k^*)^{\mathrm{T}} H_k^* L_k^* + (L_k^*)^{\mathrm{T}} G_k^* + (G_k^*)^{\mathrm{T}} L_k^* \\ &= Q_k + A_k^{\mathrm{T}} S_{k+1}^* A_k + (G_k^*)^{\mathrm{T}} (\mathbb{H}_k^*)^{-1} H_k^* (\mathbb{H}_k^*)^{-1} G_k^* - (G_k^*)^{\mathrm{T}} (\mathbb{H}_k^*)^{-1} (G_k^*) - (G_k^*)^{\mathrm{T}} (\mathbb{H}_k^*)^{-1} (G_k^*) \\ g_k^* &= r_k + B_k^{\mathrm{T}} S_{k+1}^* b_k + B_k^{\mathrm{T}} s_{k+1}^* \\ l_k^* &= -(\mathbb{H}_k^*)^{-1} g_k^* \\ s_k^* &= q_k + A_k^{\mathrm{T}} S_{k+1}^* b_k + A_k^{\mathrm{T}} s_{k+1}^* + (L_k^*)^{\mathrm{T}} H_k^* l_k^* + (L_k^*)^{\mathrm{T}} g_k^* + (G_k^*)^{\mathrm{T}} l_k^* \\ &= q_k + A_k^{\mathrm{T}} S_{k+1}^* b_k + A_k^{\mathrm{T}} s_{k+1}^* + (G_k^*)^{\mathrm{T}} (\mathbb{H}_k^*)^{-1} H_k^* (\mathbb{H}_k^*)^{-1} g_k^* - (G_k^*)^{\mathrm{T}} (\mathbb{H}_k^*)^{-1} g_k^* - (G_k^*)^{\mathrm{T}} (\mathbb{H}_k^*)^{-1} g_k^* \end{aligned} \tag{2.82}$$

Starting with given Δx_0^*, the optimal solution set of controls and states is:

$$\begin{aligned} \Delta u_k^* &= L_k^* \Delta x_k^* + l_k^* \\ \Delta x_{k+1}^* &= A_k \Delta x_k^* + B_k \Delta u_k^* + b_k \end{aligned} \tag{2.83}$$

For each stage $k \in \{0, \ldots, N-1\}, \mathbb{H}_k^* \ge 0$. Also, $\lambda^*(\| \begin{bmatrix}\Delta u_0\\ \Delta u_1\\ \vdots\\ \Delta u_{k-1}\end{bmatrix} \|_2 - \delta^u) = 0$.

Similar iterative algorithm can be used to find λ^*.

Trust Region with OCP QP with ℓ_∞ Norm

In this thesis, we solve the following problem with the Riccati-based OCP solver HPIPM [9]:

$$\begin{aligned}\min_{\Delta u_k,\Delta x_k} \quad & \sum_{k=0}^{N-1} \frac{1}{2}\begin{bmatrix}\Delta u_k\\ \Delta x_k\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}R_k & P_k\\ P_k^{\mathrm{T}} & Q_k\end{bmatrix}\begin{bmatrix}\Delta u_k\\ \Delta x_k\end{bmatrix} + \begin{bmatrix}\Delta u_k\\ \Delta x_k\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}r_k\\ q_k\end{bmatrix} + \frac{1}{2}\Delta x_N^{\mathrm{T}} Q_N \Delta x_N + \Delta x_N^{\mathrm{T}} q_N \\ \text{subject to} \quad & \Delta x_{k+1} = A_k \Delta x_k + B_k \Delta u_k + b_k \\ & \|\Delta u_k\|_\infty \le \delta_k^u\end{aligned} \tag{2.84}$$

The major reason for choosing the stage-wise ℓ_∞ norm trust region LQR setup is that HPIPM [9] already provides interior point solver for such box bounding problem. Our initial plan was to get ℓ_∞ version working and then head to ℓ_2, where deeper and more sophisticated involvement into HPIPM [9] by modifying the original block LQR matrix factorization will be a must. Due to time limit, we only achieve a working solution for the ℓ_∞ version.

2.7 Little Conclusion

We conclude this chapter by stating that 2.84 is the sequence of QP we are solving in iterative way for an approximate solution for 2.1. We feed 2.84 into a OCP QP solver directly. Details about implementation will be covered in next chapter.

Chapter 3

Implementation

A sequential quadratic programming solver [1] for optimal control problem with multiple shooting fashion is implemented under the C++ ROS framework OCS2 [23]. Matrix operations such as QR decomposition are handled by Eigen [33]. A model predictive control wrapper [2] is written for the solver such that receding horizon control is achieved in the simulation environment.

3.1 Construction of SQP Solver

As [27] pointed out, two most expensive steps in SQP schemes are solving process in QP sub-problems as well as integration and sensitivity analysis (gradient propagation) of dynamical system. In our framework, the first step is addressed with QP solver HPIPM [9] that exploits the structure of OCP QP 2.84. The second step mainly comes from the auto-differentiation mechanism for system dynamics, cost functions and constraints as part of the OCS2 [23] framework. We discuss them in details separately.

3.1.1 QP Solvers for SQP

If we rewrite the OCP QP 2.84 (first neglect the trust region radius constraints) into the following matrix form

$$\begin{aligned} \underset{\tilde{\omega}}{\text{minimize}} \quad & \frac{1}{2}\tilde{\omega}^{\mathrm{T}} Q \tilde{\omega} + \tilde{\omega}^{\mathrm{T}} c \\ \text{subject to} \quad & A\tilde{\omega} = b \end{aligned} \tag{3.1}$$

it is straightforward to see that the matrices Q, A and vectors c, b are all sparse, i.e, most of the entries are zero. The reason is that costs are stage-wise and equality constraints only embody consecutive states and inputs. Here we list common strategies to tackle QP in 3.1:

1. Directly feed it into sparse QP solver such as OOQP [34]. Undoubtedly, such method will be faster than applying a dense solver who assumes the matrices and vectors are non-zero everywhere. Yet, we still want faster speed.

[1]Code available at `https://bitbucket.org/leggedrobotics/ocs2_dev/src/master/`, branch sqp_trail_mpcnode

[2]Code available at `https://bitbucket.org/leggedrobotics/ocs2_anymal/src/master/`, branch sqp_trail_mpcnode

2. Feed it to multi-stage structure-exploiting QP solver such as HPIPM [9]. HPIPM implements primal-dual interior-point method arising from KKT conditions. When solving the resulting system of linear equations from KKT matrix, special backward Riccati recursion that handles constraints is used [20]. Meanwhile, dense linear algebra library BLASFEO [35] is used for moderate size matrix calculation. Up to our best knowledge, this might be the state-of-the-art for solving OCP QP at present.

3. Condense the OCP QP to a dense QP, solve it with a dense solver such as [36] and project the dense solution back to the original sparse space. Condensing will eliminate the system dynamic constraints and leave the inputs as degree of freedom only. Since condensing itself typically has high complexity, we will not use it here.

4. Partially condense the OCP QP to a stage between method 2 and 3. Not all state variables are eliminated. Rather, the system dynamic evolvement happens between blocks of original states. This method can trade-off between the advantages and disadvantages of method 2 and 3.

We have chosen method 2 with HPIPM [9], which dedicates to solving OCP QP with inequality constraints and slack variables. The inequality constraints are shown here because not only HPIPM is capable of handling them, but also in our implementation, $\underline{u}_n, \overline{u}_n$ provides lower and upper bounds for u, which corresponds to the ℓ_∞ norm perfectly.

$$
\begin{aligned}
\min_{x,u,s} \quad & \sum_{n=0}^{N} \frac{1}{2} \begin{bmatrix} u_n \\ x_n \\ 1 \end{bmatrix}^T \begin{bmatrix} R_n & S_n & r_n \\ S_n^T & Q_n & q_n \\ r_n^T & q_n^T & 0 \end{bmatrix} \begin{bmatrix} u_n \\ x_n \\ 1 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} s_n^l \\ s_n^u \\ 1 \end{bmatrix}^T \begin{bmatrix} Z_n^l & 0 & z_n^l \\ 0 & Z_n^u & z_n^u \\ (z_n^l)^T & (z_n^u)^T & 0 \end{bmatrix} \begin{bmatrix} s_n^l \\ s_n^u \\ 1 \end{bmatrix} \\
\text{s.t} \quad & \\
& x_{n+1} = A_n x_n + B_n u_n + b_n, && n = 0, \dots, N-1, \\
& \begin{bmatrix} \underline{u}_n \\ \underline{x}_n \\ \underline{d}_n \end{bmatrix} \leq \begin{bmatrix} J_{b,u,n} & 0 \\ 0 & J_{b,x,n} \\ D_n & C_n \end{bmatrix} \begin{bmatrix} u_n \\ x_n \end{bmatrix} + \begin{bmatrix} J_{s,u,n} \\ J_{s,x,n} \\ J_{s,g,n} \end{bmatrix} s_n^l, && n = 0, \dots, N, \\
& \begin{bmatrix} J_{b,u,n} & 0 \\ 0 & J_{b,x,n} \\ D_n & C_n \end{bmatrix} \begin{bmatrix} u_n \\ x_n \end{bmatrix} - \begin{bmatrix} J_{s,u,n} \\ J_{s,x,n} \\ J_{s,g,n} \end{bmatrix} s_n^u \leq \begin{bmatrix} \overline{u}_n \\ \overline{x}_n \\ \overline{d}_n \end{bmatrix}, && n = 0, \dots, N, \\
& s_n^l \geq \underline{s}_n^l, && n = 0, \dots, N, \\
& s_n^u \geq \underline{s}_n^u, && n = 0, \dots, N,
\end{aligned}
\tag{3.2}
$$

where u_n are the control inputs, x_n are the states, s_n^l (s_n^u) are the slack variables of the soft lower (upper) constraints and $\underline{s}_n^l$ and $\underline{s}_n^u$ are the lower bounds on lower and upper slacks, respectively. The matrices $J_{\dots,n}$ are made of rows from identity matrices. Note that all quantities can vary stage-wise. Furthermore, note that the constraint matrix with respect to u and x is the same for the upper and the lower constraints. 3.2 and the elaboration above come from the documentation of HPIPM [37].

Our problem setup 2.84 shares great similarity with 3.2 except that we do not consider general inequality constraints (we do consider the box bounds) and slack variables. As such, HPIPM will return the discrete optimal solution with several iterations of primal-dual interior-point method. This also answers why we eliminate the state-input equality constraints in 2.4.

HPIPM [9] is purely written in C. All matrices and vectors are passed to the OCP QP solving structure via C pointers with double precision, which separates the QP solving process with the OCS2 [23] framework clearly. The speed of this solver will be reflected in chapter 4.

3.1.2 Integration and Sensitivity Analysis

We repeat the original QP sub-problem with equality constraint here

$$\begin{aligned} \underset{\Delta\omega}{\text{minimize}} \quad & \frac{1}{2}\sum_{k=0}^{N-1}\begin{bmatrix}\Delta x_k\\ \Delta u_k\end{bmatrix}^{\mathrm{T}} P_k \begin{bmatrix}\Delta x_k\\ \Delta u_k\end{bmatrix} + \frac{1}{2}\Delta x_N^{\mathrm{T}} P_N \Delta x_N + \sum_{k=0}^{N-1}\begin{bmatrix}\Delta x_k\\ \Delta u_k\end{bmatrix}^{\mathrm{T}} p_k + \Delta x_N^{\mathrm{T}} p_N \\ \text{subject to} \quad & \Delta x_{k+1} = A_k\Delta x_k + B_k\Delta u_k + b_k \\ & C_k\Delta x_k + D_k\Delta u_k + e_k = 0 \end{aligned} \tag{3.3}$$

The first-order information of the equality constraints $C_k = \frac{\partial g}{\partial x_k}$, $D_k = \frac{\partial g}{\partial u_k}$ are directly available from OCS2 [23] API, which are implemented via auto-differential and out of the scope of this report. Gradient of discrete system model $A_k = \frac{\partial f_d}{\partial x_k}, B_k = \frac{\partial f_d}{\partial u_k}$ comes from the provided $\frac{\partial f_c}{\partial x_k}, \frac{\partial f_c}{\partial u_k}$ combined with either 2.28 (Forward Euler) or 2.30 (RK4). $p_k = \nabla_{(x_k,u_k)}L(x_k,u_k), p_N = \nabla_{x_N}E(x_N)$ are also directly available through the provided linear approximator of OCS2 [23].

The main trouble comes from the second order terms

$$P_k = \nabla^2_{(x_k,u_k)}\mathscr{L}, P_N = \nabla^2_{x_N}\mathscr{L} \tag{3.4}$$

As in 2.40, matrix $P_k = \begin{bmatrix} Q_k & P_k^{\mathrm{T}} \\ P_k & R_k \end{bmatrix}$ where the P_k on the left and right has totally different meanings.

$$Q_k = \nabla^2_{x_k}\mathscr{L}, R_k = \nabla^2_{u_k}\mathscr{L}, P_k = \frac{\partial^2\mathscr{L}}{\partial u_k \partial x_k}, P_N = Q_N = \nabla^2_{x_N}\mathscr{L} \tag{3.5}$$

By continuing from 2.20a and 2.20b, we calculate the Hessian of Lagrangian in details:

$$\nabla^2_{x_k}\mathscr{L} = \begin{cases} \nabla^2_{x_k}L(x_k,u_k) + \sum_{j=1}^{n}({}^j\lambda^{\mathscr{M}}_{k+1})\frac{\partial^2({}^jf_d)}{\partial x_k^2}(x_k,u_k,t_k) - \sum_{j=1}^{p}({}^j\lambda^{\mathscr{C}}_{k+1})\frac{\partial^2({}^jg)}{\partial x_k^2}(x_k,u_k,t_k) \\ \qquad k = 0,\ldots,N-1 \\ \nabla^2_{x_N}E(x_N) \end{cases} \tag{3.6a}$$

$$\nabla^2_{u_k}\mathscr{L} = \nabla^2_{u_k}L(x_k,u_k) + \sum_{j=1}^{n}({}^j\lambda^{\mathscr{M}}_{k+1})\frac{\partial^2({}^jf_d)}{\partial u_k^2}(x_k,u_k,t_k) - \sum_{j=1}^{p}({}^j\lambda^{\mathscr{C}}_{k+1})\frac{\partial^2({}^jg)}{\partial u_k^2}(x_k,u_k,t_k) \tag{3.6b}$$

$$k = 0,1,\ldots,N-1$$

$$\frac{\partial^2\mathscr{L}}{\partial u_k\partial x_k} = \frac{\partial^2 L(x_k,u_k)}{\partial u_k\partial x_k} + \sum_{j=1}^{n}({}^j\lambda^{\mathscr{M}}_{k+1})\frac{\partial^2({}^jf_d)}{\partial u_k\partial x_k}(x_k,u_k,t_k) - \sum_{j=1}^{p}({}^j\lambda^{\mathscr{C}}_{k+1})\frac{\partial^2({}^jg)}{\partial u_k\partial x_k}(x_k,u_k,t_k) \tag{3.6c}$$

$$k = 0,1,\ldots,N-1$$

where we explicitly split the third-order tensors into sum of series of Hessian matrices. The upper left index j, either with range $1 \sim n$ or $1 \sim p$, represents the j-th element of the third-order tensor and the j-th element of the Lagrangian multiplier.

Since OCS2 [23] did not provide functions to calculate the third-order tensor, we may use the following approximation to 3.6 in our implementation instead. One more important motivation to do this is that we can control the positive definiteness of Q and R through the cost function.

$$Q_k = \nabla^2_{x_k}\mathscr{L} \approx \begin{cases} \nabla^2_{x_k} L(x_k, u_k) & k = 0, \dots, N-1 \\ \nabla^2_{x_N} E(x_N) & k = N \end{cases} \tag{3.7a}$$

$$R_k = \nabla^2_{u_k}\mathscr{L} \approx \nabla^2_{u_k} L(x_k, u_k) \quad k = 0, 1, \dots, N-1 \tag{3.7b}$$

$$S_k = \frac{\partial^2 \mathscr{L}}{\partial u_k \partial x_k} \approx \frac{\partial^2 L(x_k, u_k)}{\partial u_k \partial x_k} \quad k = 0, 1, \dots, N-1 \tag{3.7c}$$

However, we argue that it is possible to use the true Hessian matrix defined in 3.6, which is the most important motivation of this master thesis – use Trust Region to handle indefinite Hessian in QP. More on this topic will be discussed in Chapter 4.

3.2 Combine SQP Solver with MPC Framework

In model predictive control framework, we use SQP solver to tackle a discrete optimal control problem iteratively. That is to say, at time τ_k, given horizon N_k, latest observed system state $\bar{x}_k$ and discretization time instants $\{t_i^k\}$, we use SQP solver to solve 2.3. Note that index k here stands for the MPC iteration number.

The nomial state and input trajectory corresponding to kth MPC iteration is denoted as $\omega_k^0 = \{x_k^{0,i}, u_k^{0,j}\}$ $i \in \{0, \dots, N_k\}, j \in \{0, \dots, N_k - 1\}$. We first linearize 2.3 all the dynamics, costs and constraints around ω_k^0 and a LQR problem in delta coordinate is formulated just as 2.84. Solving 2.84 will return us $\Delta^*\omega_k^0 = \{\Delta^* x_k^{0,i}, \Delta^* u_k^{0,j}\}$ $i \in \{0, \dots, N_k\}, j \in \{0, \dots, N_k - 1\}$. If the ρ calculated via 2.72 is satisfying, we accept the full step by $\omega_k^1 = \omega_k^0 + \Delta^*\omega_k^0$. If not, we refuse the step, shrink the trust region radius and calculate $\Delta^*\omega_k^0$ again. The sequence of $\omega_k^0, \omega_k^1, \dots, \omega_k^l$ may be repeated for l iterations until convergence for the kth MPC solution.

Assume the computation time of SQP is negligible, we get a local optimum ω_k^* containing the state trajectory and input which is forwarded to simulate the legged robot. At the next time instant τ_{k+1}, where $\tau_{k+1} - \tau_k = \frac{1}{f_{mpc}}$, we receive new $\bar{x}_{k+1}$ and $\{t_i^{k+1}\}$ and a new SQP iteration begins.

A warm start of SQP means that a good guess ω_k^0, λ_k^0 that is close to a local optimum ω_k^*, λ_k^* is used for better performance. Such a good guess is easy to obtain in the MPC framework: simply use the optimal solution $\omega_{k-1}^*, \lambda_{k-1}^*$ one step ahead. However, the discretization time instants $\{t_i^{k-1}\}$ and $\{t_i^k\}$ can be different. In that case, our ω_k^0, λ_k^0 will result from an interpolation between suitable choice of $\omega_{k-1}^*, \lambda_{k-1}^*$.

3.2.1 Controller Composition

One difference of the controller setup between previous semester project and current master thesis is that we used to construct pure feedforward controller while now we are introducing feedforward plus feedback controller.

As stated above, $x^i(u^i)$ represents state(input) trajectory around which the linearzation of i-th iteration happens. $x^{i+1}(u^{i+1})$ represents state(input) trajectory that we hope our robot would follow in reality. We are looking for different ways to calculate the incremental parts between them. The results retrieved from the backward Riccati recursion, denoted as $\Delta x^{*i}, \Delta u^{*i}$, is optimal in the following problem setup:

$$\underset{\Delta u, \Delta x}{\text{minimize}} \quad \sum_{k=0}^{N-1} \frac{1}{2} \begin{bmatrix} \Delta u_k \\ \Delta x_k \\ 1 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} R_k^i & P_k^i & r_k^i \\ (P_k^i)^{\mathrm{T}} & Q_k^i & q_k^i \\ (r_k^i)^{\mathrm{T}} & (q_k^i)^{\mathrm{T}} & 0 \end{bmatrix} \begin{bmatrix} \Delta u_k \\ \Delta x_k \\ 1 \end{bmatrix} + \frac{1}{2} \Delta x_N^{\mathrm{T}} Q_N^i \Delta x_N + \Delta x_N^{\mathrm{T}} q_N^i \tag{3.8}$$

$$\Delta x_{k+1} = A_k^i \Delta x_k + B_k^i \Delta u_k + b_k^i$$
$$\left(C_k^i \Delta x_k + D_k^i \Delta u_k + e_k^i = 0 \right)$$

3.8 is equivalent to 2.51, with or without the state-input equality constraints. By the review of full LQR decuction shown previously, $\Delta x^{*i}, \Delta u^{*i}$ has the following relationship in each stage

$$\Delta u_j^{*i} = K_j^i \Delta x_j^{*i} + k_j^i \quad \forall j \in \{0, 1, \ldots, N-1\} \tag{3.9}$$

To simplify the notations, we eliminate the stage subscript:

$$\Delta u^{*i} = K^i \Delta x^{*i} + k^i \tag{3.10}$$

An intuitive understanding of (3.8) is the following: if the system dynamics, cost function and constraints (may or may not exist) all remain linear as the Taylor first-order expansion around $x^i(u^i)$, $\Delta x^{*i}, \Delta u^{*i}$ will be optimal in the sense that $x^{i+1} - x^i = \Delta x^{*i}$, $u^{i+1} - u^i = \Delta u^{*i}$ and the new trajectory will be exactly $x^{i+1}(u^{i+1})$. However, since everything is nonlinear, the update formula for $x^{i+1}(u^{i+1})$ will be different. Currently, with $\alpha \in [0, 1]$, the update formula is:

$$u^{i+1} = u^i + \alpha \Delta u^{*i} \tag{3.11}$$

$$x^{i+1} = x^i + \alpha \Delta x^{*i} \tag{3.12}$$

and as a result

$$u^{i+1} - u^i = K^i (x^{i+1} - x^i) + \alpha k^i \tag{3.13}$$

In reality, we would like to drive our state x^{real} to be as close to x^{i+1} as possible with input u^{real}, so it is quite reasonable if do the following by mimicing (3.13). We apply the input law (3.13) not for the planned state x^{i+1}, but on the real state x^{real}.

$$u^{real} - u^i = K^i (x^{real} - x^i) + \alpha k^i \tag{3.14}$$

If we substract (3.13) and (3.14), we have

$$u^{real} - u^{i+1} = K^i (x^{real} - x^{i+1}) \tag{3.15}$$

$$u^{real} = u^{i+1} + K^i (x^{real} - x^{i+1}) \tag{3.16}$$

$$u^{real} = u^{i+1} - K^i x^{i+1} + K^i x^{real} \tag{3.17}$$

The linear controller of OCS2 [23] has the convention of

$$u^{real} = u_{ff} + K_{fb} x^{real} \tag{3.18}$$

$$u_{ff} = u^{i+1} - K^i x^{i+1} \tag{3.19}$$

$$K_{fb} = K^i \tag{3.20}$$

Then we move on to the constrained case with projection method. By consistently notating things, the decision variables $\Delta x^{*i}, \Delta\tilde{u}^{*i}$ are retrieved from hpipm. The following holds naturally:

$$\Delta\tilde{u}^{*i} = \tilde{K}^i \Delta x^{*i} + \tilde{k}^i \tag{3.21}$$

Don't forget to include the projection of input

$$\begin{aligned} \Delta u^{*i} &= P_u^i \Delta\tilde{u}^{*i} + P_x^i \Delta x^{*i} + P_e^i & (3.22)\\ &= (P_u^i \tilde{K}^i + P_x^i)\Delta x^{*i} + P_u^i \tilde{k}^i + P_e^i & (3.23)\end{aligned}$$

The update formula is still the same

$$\begin{aligned} u^{i+1} &= u^i + \alpha \Delta u^{*i} & (3.24)\\ x^{i+1} &= x^i + \alpha \Delta x^{*i} & (3.25)\end{aligned}$$

So

$$u^{i+1} - u^i = (P_u^i \tilde{K}^i + P_x^i)(x^{i+1} - x^i) + \alpha(P_u^i \tilde{k}^i + P_e^i) \tag{3.26}$$

Same mimicing policy applies

$$u^{real} - u^i = (P_u^i \tilde{K}^i + P_x^i)(x^{real} - x^i) + \alpha(P_u^i \tilde{k}^i + P_e^i) \tag{3.27}$$

The difference between above two equations is

$$\begin{aligned} u^{real} - u^{i+1} &= (P_u^i \tilde{K}^i + P_x^i)(x^{real} - x^{i+1}) & (3.28)\\ u^{real} &= u^{i+1} - (P_u^i \tilde{K}^i + P_x^i)x^{i+1} + (P_u^i \tilde{K}^i + P_x^i)x^{real} & (3.29)\\ u_f f &= u^{i+1} - (P_u^i \tilde{K}^i + P_x^i)x^{i+1} & (3.30)\\ K_{fb} &= P_u^i \tilde{K}^i + P_x^i & (3.31)\end{aligned}$$

3.2.2 Merit Function Choice

2.2 contains the original $f_{continuous}(\omega)$ to be optimized. The objective function in 2.3 is the dicrete version $f_{discrete}(\omega)$. Based on model approximation, we construct $m(\Delta\omega)$ in 2.84. Under trust region setting, the ratio is copmuted as either:

$$\rho_k = \frac{f_{continuous}(\omega_k) - f_{continuous}(\omega_k + \Delta\omega)}{m(0) - m(\Delta\omega)} \tag{3.32}$$

or

$$\rho_k = \frac{f_{discrete}(\omega_k) - f_{discrete}(\omega_k + \Delta\omega)}{m(0) - m(\Delta\omega)} \tag{3.33}$$

We are not here to discuss the discretization gap between 3.32 and 3.33. This is something can be compensated by higher order RK integration methods. Rather, 3.32 and 3.33 both neglect the inclusion of constraint satisfaction in the actual reduction (numerator). Intuitively speaking, even though a step significantly reduce the value of objective function, but at the cost of high constriant violation, we should not accept it. So here we propose to add extra terms to indicate how well the (in)equality constriants are satisfied.

$$f_{merit} = f_{discrete} + \sum_{k=0}^{N-1} \lambda_k^{\mathrm{T}}(f_d(x_k, u_k, t_k) - x_{k+1}) + \nu_k^{\mathrm{T}} g(x_k, u_k, t_k) \tag{3.34}$$

The set $\{\lambda_k\}_{k=0}^{N-1}$ contains the Lagrangian multipliers corresponding to system evolvment $\Delta x_{k+1} = A_k \Delta x_k + B_k \Delta u_k + b_k$ in 2.84 while $\{\nu_k\}_{k=0}^{N-1}$ are those for

$C_k \varDelta x_k + D_k \varDelta u_k + e_k = 0$ in 2.25. Note that $\{\nu_k\}_{k=0}^{N-1}$ are not directly retrivable through the solver HPIPM [9] because the state-input equality constraints are not present in 2.84 and 2.51. Rather, we need to compute them explicitly by ourselves. From the KKT conditions for 2.51

$$R_k \varDelta u_k + P_k \varDelta x_k + B_k^{\mathrm{T}} \lambda_k + D_k^{\mathrm{T}} \nu_k = 0 \tag{3.35}$$

Recall that in 2.37, we apply QR decomposition to $D_k^{\mathrm{T}} = Q_k^1 R_k^1$ due to its full rankness and Q_k^1 contains orthonormal vectors. As a result,

$$\nu_k = -(R_k^1)^{-1}(Q_k^1)^{\mathrm{T}}(R_k \varDelta u_k + P_k \varDelta x_k + B_k^{\mathrm{T}} \lambda_k) \tag{3.36}$$

There are other ways to formulate merit functions for trust region problems, e.g. Sℓ_1QP [38]. Relevant discussion will appear in the conclusion chapter.

3.2.3 Trajectory Spreading

Now that we have decided to use the Lagrangian multipliers ν_k to weigh the state-input equality constraints, one problem arises: what if the size of $g(x, u, t)$ does not remain the same? This is typically the case when the gait pattern got switched and there is a sudden jump of $dim(g)$, which will result in dimension inconsistency in 3.36. On the contrary, however, there is no such concern for λ_k that weigh the dynamics gaps, because the dimension of state/costate does not change along time.

We apply trajectory spreading technique as in [39]. The solution is quite intuitive and simple to understand, as shown by Fig. 3.1. Around the transitional phase at τ_j or τ_{j+1}, if dimensiona mismatch happens, we will simply use the proper pre or post trajectory $\bar{\alpha}(\cdot, j)$ with the right size of equality constraints.

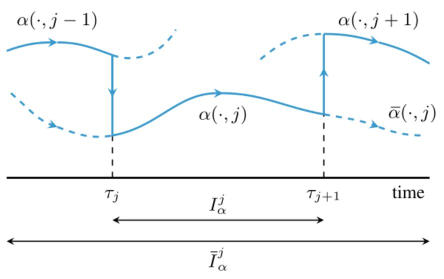

Figure 3.1: $\alpha(\cdot, j)$ is a trajectory defined on time interval $[\tau_j, \tau_{j+1}]$. $\alpha(\cdot, j-1)$ and $\alpha(\cdot, j+1)$ are its pre and post trajectory respectively. There are state jumping gaps at τ_j and τ_{j+1}. The lower dashed lines $\bar{\alpha}(\cdot, j)$ represent pre and post trajectory extension of $\alpha(\cdot, j)$. Image from [39].

3.2.4 Trust Region Radius Update Rule

The core idea of updating the trust region radius δ has been introduced in 2.6.1. A summarizing equation would be

$$\delta_{k+1} = \kappa(\rho_k)\delta_k \tag{3.37}$$

Different strategies are all represented among choice of κ function. There are no essential distinction between one another. Interested readers are referred to sec 3.5 in [38], sec 5.2 in [22] and algorithm 4.1 in [30]. The update rule used in this project is mainly sec 5.2 from [22] with fine-tuned parameters plus constraint violation considerations from [40]. Sometimes even though the computed ρ is not satisfying or even negative, as long as the updating step returned by HPIPM [9] still respects constraint violation, we will still consider to accept the step anyway.

Chapter 4

Result

The proposed multiple shooting based SQP framework for solving optimal control problem has been validated with two models, namely ballbot and ANYmal c-series.

4.1 Ballbot

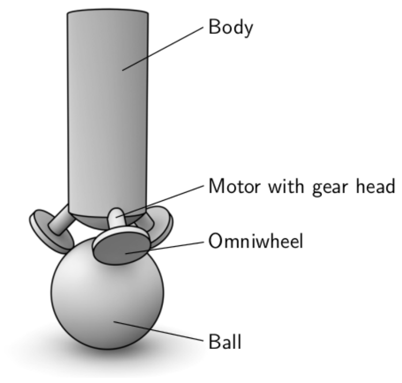

Figure 4.1: Model of ballbot. Image from [41].

Ballbot is a mobile platform with three actuating motors, one ball at the very bottom and a body attached with motors Fig 4.1. The state space representation of this robot consists of 5 states: position along X-axis, position along Y-axis, orientation of the base in Euler angle ZYX convention. In terms of system ODE equations, 10 states are needed to include velocity and angular velocity. Torques of 3 motors form the inputs to the system. Ballbot system does not have any state-input equality constraints.

Given user-defined goal state $x^{goal}, y^{goal}, \theta^{goal}_{yaw}$, our SQP solver combined with the OCS2 MPC framework is able to drive the ballbot to the destination in the obstacle forest quickly without any collision as shown in Fig 4.2, 4.3 and 4.4. Here we explicitly add the distance of ballbot to all obstacles as inequality penalties $h = \sum_{i=1}^{N_{obs}} (x_{ballbot} - x^i_{obs})^2 + (y_{ballbot} - y^i_{obs})^2 - d^2_{safe} > 0$ and apply the relaxed barrier function $p(h)$ to form penalty in the objective.

$$p(h) = \begin{cases} -\mu \ln(h) & if \quad h > \delta, \\ -\mu \ln(\delta) + \mu \frac{1}{2} \left(\left(\frac{h-2\delta}{\delta} \right)^2 - 1 \right) & otherwise, \end{cases} \tag{4.1}$$

where $\mu \geq 0$, and $\delta \geq 0$ are user defined parameters. With the assumption that $h > \delta$, $p(h) = -\mu \ln(h)$, $\frac{d^2p}{dx^2} = -\frac{1}{h(x)}\frac{d^2h}{dx^2} + \frac{1}{h^2(x)}(\frac{dh}{dx})^2$. The first part can contribute to the entire Hessian matrix to formulate an indefinite one, thus we may showcase that trust region SQP is able to handle such case. This is the reason why we set δ to be very small.

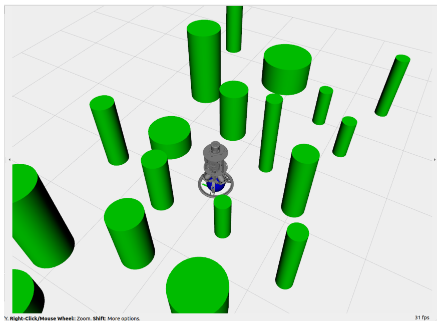
Figure 4.2: Ballbot at starting position, $x = 0, y = 0, \theta_{yaw} = 0°$

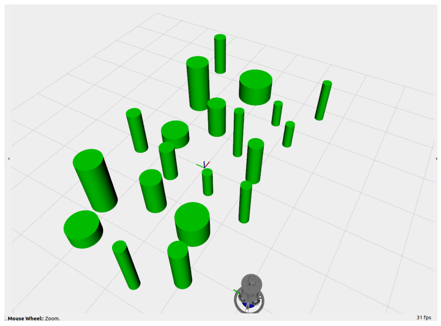
Figure 4.3: Ballbot at ending position, $x = -2, y = -2, \theta_{yaw} = 180°$

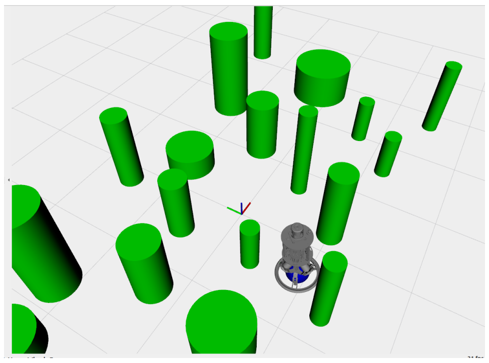
Figure 4.4: Ballbot avoiding obstacles.

We list out how many time are used in each step of SQP solver with different step size selection criterion 4.1 and 4.2. From the table we may conclude that smaller sized problem with simpler dynamic model will not benefit from applying line search over trust region. If the hessian is only slightly indefinite and the negative eigenvalue does not last for too long, line search is still our top choice.

Operation Name	Percentage
Linear Quadratic Problem Approximation	35.1%
Solve QP with HPIPM	23.8%
Trust Region Method	38.2%
Controller Computation	2.9%

Table 4.1: SQP + Trust Region time composition for ballbot

In 4.3, we compare all available solver time for ballbot obstacle avoidance. The conclusion we could draw is that SQP with suitable structure exploiting solver (HPIPM [9]) performs faster than SLQ [1] where the forward and backward pass are re-implemented but not fully optimized.

Operation Name	Percentage
Linear Quadratic Problem Approximation	61.8%
Solve QP with HPIPM	14.8%
Line Search Method	19.6%
Controller Computation	3.8%

Table 4.2: SQP + Line Search time composition for ballbot

Method Type	Average Solver Time
SLQ + Augmented Lagrangian	2.914ms
SQP + Trust Region	0.642ms
SQP + Line Search	0.452ms

Table 4.3: Different solver time comparison for ballbot obstacle avoidance

4.2 ANYmal

ANYmal is a quadrupedal platform with $x \in \mathbb{R}^{24}$, $u \in \mathbb{R}^{24}$. A detailed description of state, input, model dynamics and constraints' types can be found in [42]. Here, we also showcase quadrupedal obstacle avoidance where the parameters for relaxed barrier function are tuned such that the Hessian matrix becomes indefinite when the robot walks near the obstacle, following the same setup as in 4.1. See Fig. 4.5, 4.6 and 4.7.

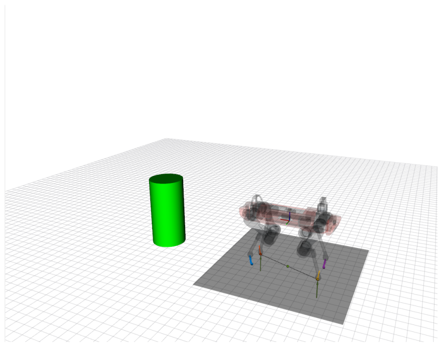
Figure 4.5: Ballbot at starting position, $x = 0, y = 0$

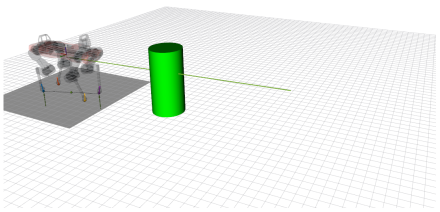
Figure 4.6: Ballbot at ending position, $x = 3, y = 0.1$

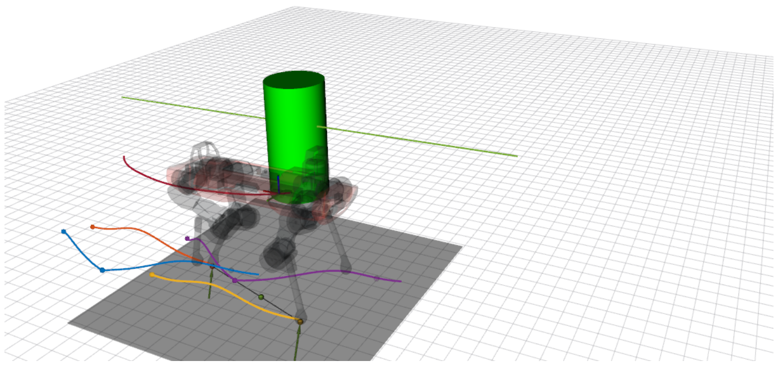

Figure 4.7: Anymal avoiding obstacles.

Fig. 4.8 and 4.9 reflect some detailed numbers during the obstacle avoidance which happens during $6.0 - 6.5s$ approximately. The discontinuity in merit function, total cost and inequality penalty during the time interval indicates the indefiniteness of hessian matrix. High equality constraint (dynamic gap) violation ISE and penalty show that the solver suffers from finding good solutions that obey the dynamics well.

It is interesting to realize that as the robot is approaching the obstacle, the objective function keeps increasing. So ordinary trust region radius update rule is no longer suitable since $\rho < 0$ happens occasionally during the interval. That is why we plug the idea of constraint satisfaction into trust region update rule.

From Fig. 4.9 we also see that when indefinite hessian appears, the SQP solver seldom converges. Meanwhile, the trust region radius is adjusted more violently and rapidly. The norm of Δx and Δu indicates severe peak during the interval.

Method	Operation Name	Absolute Time	Percentage
Line search	LQ Problem Approximation	2.92ms	66.5%
	Solve QP with HPIPM	0.84ms	19.2%
	Update Step Selection	0.49ms	11.3%
	Controller Computation	0.38ms	3.0%
Trust region	LQ Problem Approximation	3.34ms	36.0%
	Solve QP with HPIPM	5.28ms	56.9%
	Update Step Selection	0.54ms	5.8%
	Controller Computation	0.36ms	1.6%

Table 4.4: SQP Trust Region / Line Search time composition for ANYmal trotting

Unlike the ballbot case, we do not include time comparison with SQP + line search in the ANYmal experiment for obstacle avoidance. The primal reason behind is that line search will fail under such large-scale hessian indefiniteness. The internal protection mechanism of HPIPM [9] is no longer capable of handling such circumstances. Yet, it is still reasonable to compare the time usage when solving troting trajectory for the robot 4.4. It is easy to see that the introduction of trust region method will lead to significant growth of QP solving time in HPIPM ($\sim \times 6$). With TR bear in mind, the solver is capable of handling potential indefinite hessian, but the cost is too much. It seems that applying a filter to the hessian matrix rather than feeding it directly to HPIPM would be more efficient.

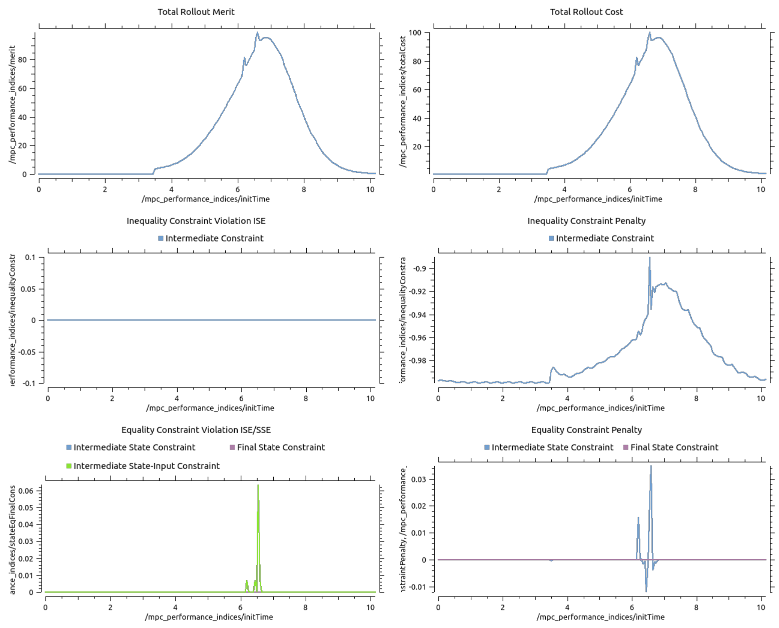

Figure 4.8: Merit function, total cost and constraint violation during ANYmal avoiding obstacles.

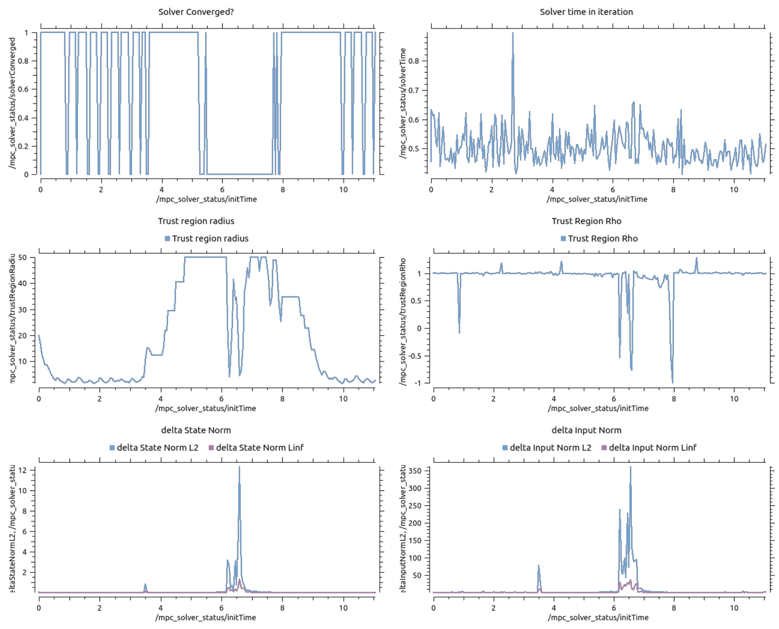

Figure 4.9: Solver status during ANYmal avoiding obstacles.

Chapter 5

Conclusion and Personal Reflections

In this report, we demonstrate the entire pipeline for creating a SQP solver with trust region method applied to each QP for optimal control problem formulated in multiple shooting fashion. We also validate our algorithm on simulation platform provided by OCS2 that robustness and generality are improved in terms that indefinite Hessian (obstacle avoidance) can be handled. Though the results and comparisons presented are mainly qualitative, our method has already shown its potential to deal with more complicated constraining conditions that may contain potential negative eigenvalues.

5.1 Personal Reflections

However, there are a lot of points to be summarized as lessons learnt from this master thesis. Properly dealing with these issues may result in a more successful project.

- Under most cases, a good starting point of any project should be aiming at solving existing real-world problem, rather than exploring randomly in the literature, constructing something new but only seems to be useful and ending up with no suitable scenario to apply the new method.

  In my case, the semester project is aimed at using multiple shooting + SQP to improve the ability of hanlding fast-changing conditions. Its result is already good enough for most cases, after the minor improvement of line search and feedback controller replacing the feedforward one. We abandon the second order terms in Hessian because calculating tensors will be expensive. We manipulate the Hessian matrix to make it postive definite in case there are indeed negative eigenvalues, which is a quick procedure that takes up little resources.

  However, in the master thesis, lots of time are spent to locate something worth to investigate, but not something that already acts as a difficulty. Trying out trust region is great, but the benefit-cost ratio is not ideal: we spend more time to solve OCP in HPIPM due to the box constraints with interior point method and more time on trust region update for potential indefinite Hessian matrix (see 4.4) but in reality 1. we might seldom meet them (only in obstacle avoidance for mobile platforms and inverse kinematics part for armed manipulation); 2. even if there is indefiniteness, converting it to a positive definite one is faster, still guaranteing a descent direction and most importantly, works in reality with great quality.

- It was pointed by [43] that regularizing an indefinite Hessian to be P.D. without changing the reduced Hessian's property (convexification) and maintaining the sparsity that optimal control problem holds specifically requires only linear complexity in terms of horizon length. After convexification, we may feed the modified Hessian to HPIPM and preserve the high speed of solving procedure.

  Systems Control and Optimization Laboratory (syscop) led by Prof. Moritz Diehl in University of Freiburg is one of the most prestigious research group on topic such as model predictive control and optimal control. Nearly all publication of the group try to explore convexity of the problem beforehand and then feed it into Riccati-based solvers [44] [45]. Technically speaking, such discussion should have been resided in the Literature Review part, but I still decide to make it here because too long time has been spent before I realize such issues.

- The current trust region SQP method is not robust in the sense that sometimes HPIPM may not be able to find a solution if the radius is too small or the problem is too ill-conditioned. As suggested by [38], a clever way of avoiding such circumstances would be introducing slack variables in the problem setup 2.84 because sometimes it would be hard for solver to balance between constraint satisfaction and function reduction. Thus, the problem is always solvable and we simply need to incorporate the slack variables as penalty into the merit function. Most importantly, HPIPM provides such interface for slack variables as noted in 3.2 but it was too late for me to realize this.

  Let me review the method proposed by [38]. The original nonlinear optimization problem is

$$\begin{aligned} &\min_x f(x) \\ &\text{subject to } c_i(x) = 0, \quad i \in \mathscr{E} \\ &\qquad\qquad c_i(x) \geq 0, \quad i \in \mathscr{I} \end{aligned} \tag{5.1}$$

  and its SQP iteration around the current guess x_k with trust region is

$$\begin{aligned} &\min_p f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathscr{L}_k p \\ &\text{subject to } \nabla c_i\left(x_k\right)^T p + c_i\left(x_k\right) = 0, \quad i \in \mathscr{E}, \\ &\qquad\qquad \nabla c_i\left(x_k\right)^T p + c_i\left(x_k\right) \geq 0, \quad i \in \mathscr{I}, \\ &\qquad\qquad \|p\| \leq \Delta_k. \end{aligned} \tag{5.2}$$

  The Sℓ_1QP proposed first by Flethcer [46] plugs all equality and inequality constraints into the objective function with ℓ_1 norm penalty

$$\begin{aligned} \min_p \quad q_\mu(p) \stackrel{\text{def}}{=} & f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla_{xx}^2 \mathscr{L}_k p \\ & + \mu \sum_{i \in \mathscr{E}} \left| c_i\left(x_k\right) + \nabla c_i\left(x_k\right)^T p \right| \\ & + \mu \sum_{i \in \mathscr{I}} \left[c_i\left(x_k\right) + \nabla c_i\left(x_k\right)^T p \right]^- \\ \text{subject to } & \|p\|_\infty \leq \Delta_k \end{aligned} \tag{5.3}$$

It is claimed that the following merit function is natural to use

$$\begin{aligned}\phi(x;\mu) &= f(x) + \mu \sum_{i\in\mathscr{E}} |c_i(x)| + \mu \sum_{i\in\mathscr{I}} [c_i(x)]^- \\ &= f(x) + \mu P(c(x))\end{aligned} \tag{5.4}$$

After solving the QP, we get p_k and use it to calculate ρ_k to judge whether we accept this step or not

$$\begin{aligned}\rho_k &= \frac{\text{ared}_k}{\text{pred}_k} = \frac{\phi(x_k,\mu) - \phi(x_k + p_k,\mu)}{q_\mu(0) - q_\mu(p_k)} \\ &= \frac{f(x_k) + \mu P(c(x_k)) - f(x_k + p_k) - \mu P(c(x_k + p_k))}{\mu P(c(x_k)) - \nabla f_k^T p_k - \frac{1}{2} p_k^T \nabla_{xx}^2 \mathscr{L}_k p_k - \mu P(c(x_k) + \nabla c(x_k)^{\mathrm{T}} p_k)}\end{aligned} \tag{5.5}$$

Sℓ_1QP use the same penalty function (ℓ_1 norm) in both q_μ (approximation model) and ϕ (true nonlinear function). In our setting, we used Lagrangian multipliers to weigh the inequality constraints in the merit for true nonlinear function and assume that nothing similar should be included in the approximation model. Though this might sound right and taste like Augmented Lagrangian method, there is no theoretical proof of such. One reason is that it is indeed hard to quantify how well continuous constraints are satisfied by sequence of discrete sampled points.

- One somewhat satisfying result produced by the master thesis would be the iterative algorithm proposed in 2.6.3 to solve OCP with trust region constraint. I believe no similar or relevant algorithm that combines the backward Riccati recursion with addition of λI has been developed in the literature before. However, this has not been proved or validated yet. Our initial plan is to get the ℓ_∞ version working and then turn back to face this one because it would require quite some modifications to C code in HPIPM solver. Honestly speaking, I believe the idea of constraining stage-wise $\|\varDelta u_k\|_2 \leq \delta_k$ may be neither theoratically supported nor the best choice we have. On the contrary, $\left\| \begin{bmatrix} \varDelta u_0 \\ \vdots \\ \varDelta u_{N-1} \end{bmatrix} \right\|_2 \leq \delta$ is strongly supported by existing theories and worth a shot. I actually have spent some time to validate it on Matlab, but the C++ version for OCS2 always goes first in our project.

- The last point I would like to express is that people should take fully advantage of where he stays and what he has in the hand. Robotics System Lab is known for its incredible accumuation on quadrupedal hardware platform. Given that the semester project solver has been working fluently for a long time, a wiser starting point for this master thesis might have been Model Hierarchy Predictive Control as in [47], operator splitting + proximal method + ADMM as in [48], higher level stepping place/gait pattern switching optimization where we use the SQP + line search method as building block and how shall we formulate constraints for OCP from the environment for dog-like robots. All such directions will exert thoroughly the so called "hardware platform advantage" of this famous quadrupedal lab.

However, nobody owns god's view in advance. At the beginning of the thesis, I was surrounded by Ph.D choices and hard to concentrate on topic choice. Anyway, I still enjoy a lot the paper reading session, understanding more of OCS2, discussion with Farbod and Ruben. That would conclude this thesis. Hope this last few words may be helpful to someone else.

Bibliography

[1] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An efficient optimal planning and control framework for quadrupedal locomotion," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 93–100.

[2] D. Jacobson and D. Mayne, *Differential Dynamic Programming*, ser. Modern analytic and computational methods in science and mathematics. American Elsevier Publishing Company, 1970.

[3] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 3529–3536, 2019.

[4] P.-B. Wieber, R. Tedrake, and S. Kuindersma, "Modeling and control of legged robots," in *Springer handbook of robotics*. Springer, 2016, pp. 1203–1234.

[5] R. Grandia, F. Farshidian, A. Dosovitskiy, R. Ranftl, and M. Hutter, "Frequency-aware model predictive control," *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1517–1524, 2019.

[6] R. Grandia, F. Farshidian, R. Ranftl, and M. Hutter, "Feedback mpc for torque-controlled legged robots," *arXiv preprint arXiv:1905.06144*, 2019.

[7] J.-P. Sleiman, F. Farshidian, and M. Hutter, "Constraint handling in continuous-time ddp-based model predictive control," *arXiv preprint arXiv:2101.06067*, 2021.

[8] M. Diehl, H. G. Bock, and J. P. Schlöder, "A real-time iteration scheme for nonlinear optimization in optimal feedback control," *SIAM Journal on control and optimization*, vol. 43, no. 5, pp. 1714–1736, 2005.

[9] G. Frison and M. Diehl, "Hpipm: a high-performance quadratic programming framework for model predictive control," *arXiv preprint arXiv:2003.02547*, 2020.

[10] M. Diehl, "Lecture notes on optimal control and estimation," 2014.

[11] M. Giftthaler, "Towards a unified framework of efficient algorithms for numerical optimal robot control," Ph.D. dissertation, ETH Zurich, 2018.

[12] R. Sargent and G. Sullivan, "The development of an efficient optimal control package," in *Optimization Techniques*. Springer, 1978, pp. 158–168.

[13] H. G. Bock and K.-J. Plitt, "A multiple shooting algorithm for direct solution of optimal control problems," *IFAC Proceedings Volumes*, vol. 17, no. 2, pp. 1603–1608, 1984.

[14] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch *et al.*, "Anymal-a highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS).* IEEE, 2016, pp. 38–44.

[15] E. Todorov and W. Li, "A generalized iterative lqg method for locally-optimal feedback control of constrained nonlinear stochastic systems," in *Proceedings of the 2005, American Control Conference, 2005.* IEEE, 2005, pp. 300–306.

[16] A. Sideris and J. E. Bobrow, "An efficient sequential linear quadratic algorithm for solving nonlinear optimal control problems," in *Proceedings of the 2005, American Control Conference, 2005.* IEEE, 2005, pp. 2275–2280.

[17] M. Giftthaler, M. Neunert, M. Stäuble, J. Buchli, and M. Diehl, "A family of iterative gauss-newton shooting methods for nonlinear optimal control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, 2018, pp. 1–9.

[18] T. A. Howell, B. E. Jackson, and Z. Manchester, "Altro: A fast solver for constrained trajectory optimization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).* IEEE, 2019, pp. 7674–7679.

[19] G. Frison and J. B. Jørgensen, "Efficient implementation of the riccati recursion for solving linear-quadratic control problems," in *2013 IEEE International Conference on Control Applications (CCA).* IEEE, 2013, pp. 1117–1122.

[20] G. Frison, H. H. B. Sørensen, B. Dammann, and J. B. Jørgensen, "High-performance small-scale solvers for linear model predictive control," in *2014 European Control Conference (ECC).* IEEE, 2014, pp. 128–133.

[21] G. Lantoine and R. P. Russell, "A hybrid differential dynamic programming algorithm for constrained optimal control problems. part 1: Theory," *Journal of Optimization Theory and Applications*, vol. 154, no. 2, pp. 382–417, 2012.

[22] E. Pellegrini and R. P. Russell, "A multiple-shooting differential dynamic programming algorithm. part 1: Theory," *Acta Astronautica*, vol. 170, pp. 686–700, 2020.

[23] "Documentation for optimal control for switched systems algorithm (ocs2)," https://leggedrobotics.github.io/ocs2/, accessed: 2021-01-27.

[24] I. Pólik and T. Terlaky, *Interior Point Methods for Nonlinear Optimization.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 215–276.

[25] P. T. Boggs and J. W. Tolle, "Sequential quadratic programming," *Acta numerica*, vol. 4, pp. 1–51, 1995.

[26] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM review*, vol. 47, no. 1, pp. 99–131, 2005.

[27] R. Verschueren, G. Frison, D. Kouzoupis, N. van Duijkeren, A. Zanelli, B. Novoselnik, J. Frey, T. Albin, R. Quirynen, and M. Diehl, "Acados: A modular open-source framework for fast embedded optimal control," *arXiv preprint arXiv:1910.13753*, 2019.

[28] 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*, vol. 11, no. 1, pp. 1–36, 2019.

[29] D. Kouzoupis, G. Frison, A. Zanelli, and M. Diehl, "Recent advances in quadratic programming algorithms for nonlinear model predictive control," *Vietnam Journal of Mathematics*, vol. 46, no. 4, pp. 863–882, 2018.

[30] J. Nocedal and S. Wright, *Numerical optimization.* Springer Science & Business Media, 2006.

[31] *7. The Trust-Region Subproblem*, pp. 169–248.

[32] G. Frison, "Numerical methods for model predictive control," 2012.

[33] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[34] E. M. Gertz and S. J. Wright, "Object-oriented software for quadratic programming," *ACM Transactions on Mathematical Software (TOMS)*, vol. 29, no. 1, pp. 58–81, 2003.

[35] G. Frison, D. Kouzoupis, T. Sartor, A. Zanelli, and M. Diehl, "Blasfeo: Basic linear algebra subroutines for embedded optimization," *ACM Transactions on Mathematical Software (TOMS)*, vol. 44, no. 4, pp. 1–30, 2018.

[36] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpoases: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[37] "Documentation for hpipm," https://github.com/giaf/hpipm/blob/master/doc/guide.pdf, accessed: 2021-01-27.

[38] D. Boiroux and J. B. Jørgensen, "Sequential l1 quadratic programming for nonlinear model predictive control," *IFAC-PapersOnLine*, vol. 52, no. 1, pp. 474–479, 2019.

[39] M. Rijnen, A. Saccon, and H. Nijmeijer, "Reference spreading: Tracking performance for impact trajectories of a 1dof setup," *IEEE Transactions on Control Systems Technology*, vol. 28, no. 3, pp. 1124–1131, 2019.

[40] 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.

[41] P. Fankhauser and C. Gwerder, "Modeling and control of a ballbot," Master's thesis, ETH Zurich, Zürich, 2010, bachelor Thesis ETH Zurich, 2010.

[42] R. Grandia, A. J. Taylor, A. D. Ames, and M. Hutter, "Multi-layered safety for legged robots via control barrier functions and model predictive control," *arXiv preprint arXiv:2011.00032*, 2020.

[43] R. Verschueren, M. Zanon, R. Quirynen, and M. Diehl, "A sparsity preserving convexification procedure for indefinite quadratic programs arising in direct optimal control," *SIAM Journal on Optimization*, vol. 27, no. 3, pp. 2085–2109, 2017.

[44] R. Verschueren, N. van Duijkeren, R. Quirynen, and M. Diehl, "Exploiting convexity in direct optimal control: a sequential convex quadratic programming method," in *2016 IEEE 55th Conference on Decision and Control (CDC).* IEEE, 2016, pp. 1099–1104.

[45] J. Hall, A. Nurkanović, F. Messerer, and M. Diehl, “A sequential convex programming approach to solving quadratic programs and optimal control problems with linear complementarity constraints,” *IEEE Control Systems Letters*, 2021.

[46] R. Fletcher and S. Leyffer, “Nonlinear programming without a penalty function,” *Mathematical programming*, vol. 91, no. 2, pp. 239–269, 2002.

[47] H. Li, R. J. Frei, and P. M. Wensing, “Model hierarchy predictive control of robotic systems,” *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3373–3380, 2021.

[48] V. Sindhwani, R. Roelofs, and M. Kalakrishnan, “Sequential operator splitting for constrained nonlinear optimal control,” in *2017 American Control Conference (ACC)*. IEEE, 2017, pp. 4864–4871.

Appendix A

Proof of Necessary Conditions for a Nonsingular KKT Matrix

We claim that the KKT matrix $\begin{bmatrix} Q & A^{\mathrm{T}} \\ A & 0 \end{bmatrix}$ is invertible if the following two conditions are fulfilled:

- Matrix A has full row rank
- $Z^{\mathrm{T}}QZ$ is positive definite, where $Z = \begin{bmatrix} z_1 & z_2 & \dots & z_{n-m} \end{bmatrix}$ such that $\{z_1, z_2, \dots, z_{n-m}\}$ form a basis of the null space of matrix A.

Now we show why the above two conditions lead to nonsingularity of KKT matrix in 2.8. Suppose there exists $\begin{bmatrix} p \\ q \end{bmatrix}$ such that

$$\begin{bmatrix} Q & A^{\mathrm{T}} \\ A & 0 \end{bmatrix} \begin{bmatrix} p \\ q \end{bmatrix} = 0 \tag{A.1}$$

then we will have $Ap = 0 \Rightarrow p \in \operatorname{span}\{z_1, z_2, \dots, z_{n-m}\} \Rightarrow p = Zr$ for certain $r \in \mathbb{R}^{n-m}$ and $Qp + A^{\mathrm{T}}q = 0$. If we multiple the transpose of $\begin{bmatrix} p \\ q \end{bmatrix}$ beforehand in A.1,

$$\begin{aligned} 0 = \begin{bmatrix} p \\ q \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} Q & A^{\mathrm{T}} \\ A & 0 \end{bmatrix} \begin{bmatrix} p \\ q \end{bmatrix} &= p^{\mathrm{T}}Qp + p^{\mathrm{T}}A^{\mathrm{T}}q + q^{\mathrm{T}}Ap \\ &= p^{\mathrm{T}}Qp \\ &= r^{\mathrm{T}}Z^{\mathrm{T}}QZr \end{aligned} \tag{A.2}$$

Since $Z^{\mathrm{T}}QZ$ is positive definite, $r^{\mathrm{T}}Z^{\mathrm{T}}QZr = 0$ only holds when $r = 0 \Rightarrow p = Zr = 0 \Rightarrow A^{\mathrm{T}}q = 0$. Since A has full row rank, $q = 0$ must hold. So we can see that A.1 holds only if $p = q = 0$, so the KKT matrix in nonsingular.

Appendix B

Integration Methods and Linearization

Forward Euler method:

$$x_{k+1} = f_d(x_k, u_k, t_k) = x_k + f_c(x_k, u_k, t_k)\Delta t \tag{B.1}$$

Runge–Kutta methods with fourth-order:

$$\begin{aligned}
x_{k+1} &= f_d(x_k, u_k, t_k) = x_k + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\
k_1 &= f_c(x_k, u_k, t_k) \\
k_2 &= f_c(x_k + \frac{\Delta t}{2}k_1, u_k, t_k + \frac{\Delta t}{2}) \\
k_3 &= f_c(x_k + \frac{\Delta t}{2}k_2, u_k, t_k + \frac{\Delta t}{2}) \\
k_4 &= f_c(x_k + \Delta t k_3, u_k, t_k + \Delta t)
\end{aligned} \tag{B.2}$$

The linearization results for RK4 method:

$$\frac{\partial k_1}{\partial x_k} = \frac{\partial f_c}{\partial x_k}(x_k, u_k, t_k) \tag{B.3a}$$

$$\begin{aligned}
\frac{\partial k_2}{\partial x_k} &= \frac{\partial f_c}{\partial (x_k + \frac{\Delta t}{2}k_1)}(x_k + \frac{\Delta t}{2}k_1, u_k, t_k + \frac{\Delta t}{2})\frac{\partial (x_k + \frac{\Delta t}{2}k_1)}{\partial x_k} \\
&= \frac{\partial f_c}{\partial (x_k + \frac{\Delta t}{2}k_1)}(x_k + \frac{\Delta t}{2}k_1, u_k, t_k + \frac{\Delta t}{2})(I_d + \frac{\Delta t}{2}\frac{\partial k_1}{\partial x_k})
\end{aligned} \tag{B.3b}$$

$$\begin{aligned}
\frac{\partial k_3}{\partial x_k} &= \frac{\partial f_c}{\partial (x_k + \frac{\Delta t}{2}k_2)}(x_k + \frac{\Delta t}{2}k_2, u_k, t_k + \frac{\Delta t}{2})\frac{\partial (x_k + \frac{\Delta t}{2}k_2)}{\partial x_k} \\
&= \frac{\partial f_c}{\partial (x_k + \frac{\Delta t}{2}k_2)}(x_k + \frac{\Delta t}{2}k_2, u_k, t_k + \frac{\Delta t}{2})(I_d + \frac{\Delta t}{2}\frac{\partial k_2}{\partial x_k})
\end{aligned} \tag{B.3c}$$

$$\begin{aligned}
\frac{\partial k_4}{\partial x_k} &= \frac{\partial f_c}{\partial (x_k + \Delta t k_3)}(x_k + \Delta t k_3, u_k, t_k + \Delta t)\frac{\partial (x_k + \Delta t k_3)}{\partial x_k} \\
&= \frac{\partial f_c}{\partial (x_k + \Delta t k_3)}(x_k + \Delta t k_3, u_k, t_k + \Delta t)(I_d + \Delta t\frac{\partial k_3}{\partial x_k})
\end{aligned} \tag{B.3d}$$

and

$$\frac{\partial k_1}{\partial u_k} = \frac{\partial f_c}{\partial u_k}(x_k, u_k, t_k) \tag{B.4a}$$

$$\begin{aligned}\frac{\partial k_2}{\partial u_k} &= \frac{\partial f_c}{\partial(x_k + \frac{\Delta t}{2}k_1)}(x_k + \frac{\Delta t}{2}k_1, u_k, t_k + \frac{\Delta t}{2})\frac{\partial(x_k + \frac{\Delta t}{2}k_1)}{\partial u_k} + \frac{\partial f_c}{\partial u_k}(x_k + \frac{\Delta t}{2}k_1, u_k, t_k + \frac{\Delta t}{2}) \\ &= \frac{\partial f_c}{\partial(x_k + \frac{\Delta t}{2}k_1)}(x_k + \frac{\Delta t}{2}k_1, u_k, t_k + \frac{\Delta t}{2})\frac{\Delta t}{2}\frac{\partial k_1}{\partial u_k} + \frac{\partial f_c}{\partial u_k}(x_k + \frac{\Delta t}{2}k_1, u_k, t_k + \frac{\Delta t}{2})\end{aligned} \tag{B.4b}$$

$$\begin{aligned}\frac{\partial k_3}{\partial u_k} &= \frac{\partial f_c}{\partial(x_k + \frac{\Delta t}{2}k_2)}(x_k + \frac{\Delta t}{2}k_2, u_k, t_k + \frac{\Delta t}{2})\frac{\partial(x_k + \frac{\Delta t}{2}k_2)}{\partial u_k} + \frac{\partial f_c}{\partial u_k}(x_k + \frac{\Delta t}{2}k_2, u_k, t_k + \frac{\Delta t}{2}) \\ &= \frac{\partial f_c}{\partial(x_k + \frac{\Delta t}{2}k_2)}(x_k + \frac{\Delta t}{2}k_2, u_k, t_k + \frac{\Delta t}{2})\frac{\Delta t}{2}\frac{\partial k_2}{\partial u_k} + \frac{\partial f_c}{\partial u_k}(x_k + \frac{\Delta t}{2}k_2, u_k, t_k + \frac{\Delta t}{2})\end{aligned} \tag{B.4c}$$

$$\begin{aligned}\frac{\partial k_4}{\partial u_k} &= \frac{\partial f_c}{\partial(x_k + \Delta t k_3)}(x_k + \Delta t k_3, u_k, t_k + \Delta t)\frac{\partial(x_k + \Delta t k_3)}{\partial u_k} + \frac{\partial f_c}{\partial u_k}(x_k + \Delta t k_3, u_k, t_k + \Delta t) \\ &= \frac{\partial f_c}{\partial(x_k + \Delta t k_3)}(x_k + \Delta t k_3, u_k, t_k + \Delta t)\Delta t\frac{\partial k_3}{\partial u_k} + \frac{\partial f_c}{\partial u_k}(x_k + \Delta t k_3, u_k, t_k + \Delta t)\end{aligned} \tag{B.4d}$$

Appendix C

Some Effort on ℓ_2 Norm Trust Region Method Combined with Riccati Recursion

Consider the simplest LQR set-up (constant linear dynamics with no offset), in case $N = 1$, we are solving the following problem

$$\begin{aligned} \min_{u_0} \quad & \frac{1}{2}u_0^{\mathrm{T}} R_0 u_0 + \frac{1}{2}\begin{bmatrix} x_0 \\ x_1 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} Q_0 & 0 \\ 0 & Q_1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} \\ \text{subject to} \quad & \begin{bmatrix} x_0 \\ x_1 \end{bmatrix} = \begin{bmatrix} 0 \\ B \end{bmatrix} u_0 + \begin{bmatrix} I \\ A \end{bmatrix} x_0 \end{aligned} \tag{C.1}$$

And it is equivalent to

$$\begin{aligned} \min_{u_0} \quad & \frac{1}{2}u_0^{\mathrm{T}} R_0 u_0 + \frac{1}{2}u_0^{\mathrm{T}} \begin{bmatrix} 0 & B^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} Q_0 & 0 \\ 0 & Q_1 \end{bmatrix} \begin{bmatrix} 0 \\ B \end{bmatrix} u_0 + \frac{1}{2}x_0^{\mathrm{T}} \begin{bmatrix} I & A^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} Q_0 & 0 \\ 0 & Q_1 \end{bmatrix} \begin{bmatrix} I \\ A \end{bmatrix} x_0 + x_0^{\mathrm{T}} \begin{bmatrix} I & A^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} Q_0 & 0 \\ 0 & Q_1 \end{bmatrix} \begin{bmatrix} 0 \\ B \end{bmatrix} u_0 \\ & = \frac{1}{2}u_0^{\mathrm{T}}(R_0 + B^{\mathrm{T}} Q_1 B)u_0 + (B^{\mathrm{T}} Q_1 A x_0)^{\mathrm{T}} u_0 + \frac{1}{2}x_0^{\mathrm{T}}(Q_0 + A^{\mathrm{T}} Q_1 A)x_0 \end{aligned} \tag{C.2}$$

So the optimal solution is $u_0^* = -(R_0 + B^{\mathrm{T}} Q_1 B)^{-1} B^{\mathrm{T}} Q_1 A x_0 = K_0 x_0$ and optimum value function is $\frac{1}{2}x_0^{\mathrm{T}} P_0 x_0$, $P_0 = Q_0 + A^{\mathrm{T}} Q_1 A - A^{\mathrm{T}} Q_1 B(R_0 + B^{\mathrm{T}} Q_1 B)^{-1} B^{\mathrm{T}} Q_1 A$.

For the case $N = 2$,

$$\begin{aligned} \min_{u_0, u_1} \quad & \frac{1}{2}\begin{bmatrix} u_0 \\ u_1 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} R_0 & 0 \\ 0 & R_1 \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \end{bmatrix} + \frac{1}{2}\begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} Q_0 & 0 & 0 \\ 0 & Q_1 & 0 \\ 0 & 0 & Q_2 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} \\ \text{subject to} \quad & \begin{bmatrix} x_0 \\ x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ B & 0 \\ AB & B \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \end{bmatrix} + \begin{bmatrix} I \\ A \\ A^2 \end{bmatrix} x_0 \end{aligned} \tag{C.3}$$

And it is equivalent to

$$\begin{aligned}
&\min_{u_0,u_1}\ \frac{1}{2}\begin{bmatrix}u_0\\u_1\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}R_0 & 0\\0 & R_1\end{bmatrix}\begin{bmatrix}u_0\\u_1\end{bmatrix}+\frac{1}{2}\begin{bmatrix}u_0\\u_1\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}0 & B^{\mathrm{T}} & B^{\mathrm{T}}A^{\mathrm{T}}\\0 & 0 & B^{\mathrm{T}}\end{bmatrix}\begin{bmatrix}Q_0 & 0 & 0\\0 & Q_1 & 0\\0 & 0 & Q_2\end{bmatrix}\begin{bmatrix}0 & 0\\B & 0\\AB & B\end{bmatrix}\begin{bmatrix}u_0\\u_1\end{bmatrix}\\
&+\frac{1}{2}x_0^{\mathrm{T}}\begin{bmatrix}I & A^{\mathrm{T}} & (A^2)^{\mathrm{T}}\end{bmatrix}\begin{bmatrix}Q_0 & 0 & 0\\0 & Q_1 & 0\\0 & 0 & Q_2\end{bmatrix}\begin{bmatrix}I\\A\\A^2\end{bmatrix}x_0+x_0^{\mathrm{T}}\begin{bmatrix}I & A^{\mathrm{T}} & (A^2)^{\mathrm{T}}\end{bmatrix}\begin{bmatrix}Q_0 & 0 & 0\\0 & Q_1 & 0\\0 & 0 & Q_2\end{bmatrix}\begin{bmatrix}0 & 0\\B & 0\\AB & B\end{bmatrix}\begin{bmatrix}u_0\\u_1\end{bmatrix}\\
&=\frac{1}{2}\begin{bmatrix}u_0\\u_1\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}R_0+B^{\mathrm{T}}Q_1B+B^{\mathrm{T}}A^{\mathrm{T}}Q_2AB & B^{\mathrm{T}}A^{\mathrm{T}}Q_2B\\B^{\mathrm{T}}Q_2AB & R_1+B^{\mathrm{T}}Q_2B\end{bmatrix}\begin{bmatrix}u_0\\u_1\end{bmatrix}+\Big(\begin{bmatrix}B^{\mathrm{T}}Q_1A+B^{\mathrm{T}}A^{\mathrm{T}}Q_2A^2\\B^{\mathrm{T}}Q_2A^2\end{bmatrix}x_0\Big)^{\mathrm{T}}\begin{bmatrix}u_0\\u_1\end{bmatrix}\\
&+\frac{1}{2}x_0^{\mathrm{T}}(Q_0+A^{\mathrm{T}}Q_1A+(A^2)^{\mathrm{T}}Q_2A^2)x_0\\
&=\frac{1}{2}\begin{bmatrix}u_0\\u_1\end{bmatrix}^{\mathrm{T}}H_{01}\begin{bmatrix}u_0\\u_1\end{bmatrix}+g_{01}^{\mathrm{T}}\begin{bmatrix}u_0\\u_1\end{bmatrix}+offset
\end{aligned} \tag{C.4}$$

Recall the following Schur complement trick to compute inverse for block matrices:

$$\begin{bmatrix}A & B\\B^{\mathrm{T}} & D\end{bmatrix}^{-1}=\begin{bmatrix}I & 0\\-D^{-1}B^{\mathrm{T}} & I\end{bmatrix}\begin{bmatrix}(A-BD^{-1}B^{\mathrm{T}})^{-1} & 0\\0 & D^{-1}\end{bmatrix}\begin{bmatrix}I & -BD^{-1}\\0 & I\end{bmatrix} \tag{C.5}$$

We may apply Schur complement trick to calculate the inverse of Hessian H_{01}^{-1}:

$$\begin{aligned}
H_{01}^{-1}&=\begin{bmatrix}I & 0\\K_1B & I\end{bmatrix}\begin{bmatrix}(R_0+B^{\mathrm{T}}P_1B)^{-1} & 0\\0 & (R_1+B^{\mathrm{T}}Q_2B)^{-1}\end{bmatrix}\begin{bmatrix}I & B^{\mathrm{T}}K_1^{\mathrm{T}}\\0 & I\end{bmatrix}\\
&=\begin{bmatrix}(R_0+B^{\mathrm{T}}P_1B)^{-1} & (R_0+B^{\mathrm{T}}P_1B)^{-1}B^{\mathrm{T}}K_1^{\mathrm{T}}\\K_1B(R_0+B^{\mathrm{T}}P_1B)^{-1} & (R_1+B^{\mathrm{T}}Q_2B)^{-1}+K_1B(R_0+B^{\mathrm{T}}P_1B)^{-1}B^{\mathrm{T}}K_1^{\mathrm{T}}\end{bmatrix}\\
K_1&=-(R_1+B^{\mathrm{T}}Q_2B)^{-1}B^{\mathrm{T}}Q_2A\\
P_1&=Q_1+A^{\mathrm{T}}Q_2A-A^{\mathrm{T}}Q_2B(R_1+B^{\mathrm{T}}Q_2B)^{-1}B^{\mathrm{T}}Q_2A
\end{aligned} \tag{C.6}$$

And the optimal solutions are

$$\begin{aligned}
\begin{bmatrix}u_0^*\\u_1^*\end{bmatrix}&=-H_{01}^{-1}g_{01}\\
&=-\begin{bmatrix}(R_0+B^{\mathrm{T}}P_1B)^{-1} & (R_0+B^{\mathrm{T}}P_1B)^{-1}B^{\mathrm{T}}K_1^{\mathrm{T}}\\K_1B(R_0+B^{\mathrm{T}}P_1B)^{-1} & (R_1+B^{\mathrm{T}}Q_2B)^{-1}+K_1B(R_0+B^{\mathrm{T}}P_1B)^{-1}B^{\mathrm{T}}K_1^{\mathrm{T}}\end{bmatrix}\begin{bmatrix}B^{\mathrm{T}}Q_1A+B^{\mathrm{T}}A^{\mathrm{T}}Q_2A^2\\B^{\mathrm{T}}Q_2A^2\end{bmatrix}x_0\\
&=-\begin{bmatrix}(R_0+B^{\mathrm{T}}P_1B)^{-1}B^{\mathrm{T}}(Q_1+A^{\mathrm{T}}Q_2A+K_1^{\mathrm{T}}B^{\mathrm{T}}Q_2A)A\\K_1B(R_0+B^{\mathrm{T}}P_1B)^{-1}B^{\mathrm{T}}(Q_1+A^{\mathrm{T}}Q_2A+K_1^{\mathrm{T}}B^{\mathrm{T}}Q_2A)A+(R_1+B^{\mathrm{T}}Q_2B)^{-1}B^{\mathrm{T}}Q_2A^2\end{bmatrix}x_0\\
&=\begin{bmatrix}-(R_0+B^{\mathrm{T}}P_1B)^{-1}B^{\mathrm{T}}P_1A\\-K_1B(R_0+B^{\mathrm{T}}P_1B)^{-1}B^{\mathrm{T}}P_1A+K_1A\end{bmatrix}x_0\\
&=\begin{bmatrix}K_0\\K_1(A+BK_0)\end{bmatrix}x_0\\
K_0&=-(R_0+B^{\mathrm{T}}P_1B)^{-1}B^{\mathrm{T}}P_1A
\end{aligned} \tag{C.7}$$

For the case $N=3$,

$$\begin{aligned}
&\min_{u_0,u_1,u_2} \frac{1}{2}\begin{bmatrix}u_0\\u_1\\u_2\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}R_0&&\\&R_1&\\&&R_2\end{bmatrix}\begin{bmatrix}u_0\\u_1\\u_2\end{bmatrix}+\frac{1}{2}\begin{bmatrix}x_0\\x_1\\x_2\\x_3\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}Q_0&&&\\&Q_1&&\\&&Q_2&\\&&&Q_3\end{bmatrix}\begin{bmatrix}x_0\\x_1\\x_2\\x_3\end{bmatrix}\\
&\text{subject to} \begin{bmatrix}x_0\\x_1\\x_2\\x_3\end{bmatrix}=\begin{bmatrix}&&\\B&&\\AB&B&\\A^2B&AB&B\end{bmatrix}\begin{bmatrix}u_0\\u_1\\u_2\end{bmatrix}+\begin{bmatrix}I\\A\\A^2\\A^3\end{bmatrix}x_0
\end{aligned} \tag{C.8}$$

And it is equivalent to

$$\begin{aligned}
&\min_{u_0,u_1,u_2} \frac{1}{2}\begin{bmatrix}u_0\\u_1\\u_2\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}R_0+B^{\mathrm{T}}Q_1B+B^{\mathrm{T}}A^{\mathrm{T}}Q_2AB+B^{\mathrm{T}}(A^2)^{\mathrm{T}}Q_3A^2B & B^{\mathrm{T}}A^{\mathrm{T}}Q_2B+B^{\mathrm{T}}(A^2)^{\mathrm{T}}Q_3AB & B^{\mathrm{T}}(A^2)^{\mathrm{T}}Q_3B\\ B^{\mathrm{T}}Q_2AB+B^{\mathrm{T}}A^{\mathrm{T}}Q_3A^2B & R_1+B^{\mathrm{T}}Q_2B+B^{\mathrm{T}}A^{\mathrm{T}}Q_3AB & B^{\mathrm{T}}A^{\mathrm{T}}Q_3B\\ B^{\mathrm{T}}Q_3A^2B & B^{\mathrm{T}}Q_3AB & R_2+B^{\mathrm{T}}Q_3B\end{bmatrix}\begin{bmatrix}u_0\\u_1\\u_2\end{bmatrix}\\
&\quad+\left(\begin{bmatrix}B^{\mathrm{T}}Q_1A+B^{\mathrm{T}}A^{\mathrm{T}}Q_2A^2+B^{\mathrm{T}}(A^2)^{\mathrm{T}}Q_3A^3\\ B^{\mathrm{T}}Q_2A^2+B^{\mathrm{T}}A^{\mathrm{T}}Q_3A^3\\ B^{\mathrm{T}}Q_3A^3\end{bmatrix}x_0\right)^{\mathrm{T}}\begin{bmatrix}u_0\\u_1\\u_2\end{bmatrix}+\frac{1}{2}x_0^{\mathrm{T}}(Q_0+A^{\mathrm{T}}Q_1A+(A^2)^{\mathrm{T}}Q_2A^2+(A^3)^{\mathrm{T}}Q_3A^3)x_0\\
&\quad=\frac{1}{2}\begin{bmatrix}u_0\\u_1\\u_2\end{bmatrix}^{\mathrm{T}}H_{012}\begin{bmatrix}u_0\\u_1\\u_2\end{bmatrix}+g_{012}^{\mathrm{T}}\begin{bmatrix}u_0\\u_1\\u_2\end{bmatrix}+offset
\end{aligned} \tag{C.9}$$

We apply the same trick to invert H_{012} matrix, but in block fashion by isolating the last stage

$$\begin{aligned}
H_{012}^{-1}&=\begin{bmatrix}I&\\ \begin{bmatrix}K_2AB&K_2B\end{bmatrix}&I\end{bmatrix}\begin{bmatrix}\begin{bmatrix}R_0+B^{\mathrm{T}}Q_1B+B^{\mathrm{T}}A^{\mathrm{T}}P_2AB & B^{\mathrm{T}}A^{\mathrm{T}}P_2B\\ B^{\mathrm{T}}P_2AB & R_1+B^{\mathrm{T}}P_2B\end{bmatrix}^{-1}&\\ &(R_2+B^{\mathrm{T}}Q_3B)^{-1}\end{bmatrix}\begin{bmatrix}I&\begin{bmatrix}B^{\mathrm{T}}A^{\mathrm{T}}K_2^{\mathrm{T}}\\B^{\mathrm{T}}K_2^{\mathrm{T}}\end{bmatrix}\\&I\end{bmatrix}\\
&=\begin{bmatrix}I&\\ \begin{bmatrix}K_2AB&K_2B\end{bmatrix}&I\end{bmatrix}\begin{bmatrix}H_{01}^{-1}&\\&(R_2+B^{\mathrm{T}}Q_3B)^{-1}\end{bmatrix}\begin{bmatrix}I&\begin{bmatrix}B^{\mathrm{T}}A^{\mathrm{T}}K_2^{\mathrm{T}}\\B^{\mathrm{T}}K_2^{\mathrm{T}}\end{bmatrix}\\&I\end{bmatrix}\\
&=\begin{bmatrix}H_{01}^{-1} & H_{01}^{-1}\begin{bmatrix}B^{\mathrm{T}}A^{\mathrm{T}}K_2^{\mathrm{T}}\\B^{\mathrm{T}}K_2^{\mathrm{T}}\end{bmatrix}\\ \begin{bmatrix}K_2AB&K_2B\end{bmatrix}H_{01}^{-1} & (R_2+B^{\mathrm{T}}Q_3B)^{-1}+\begin{bmatrix}K_2AB&K_2B\end{bmatrix}H_{01}^{-1}\begin{bmatrix}B^{\mathrm{T}}A^{\mathrm{T}}K_2^{\mathrm{T}}\\B^{\mathrm{T}}K_2^{\mathrm{T}}\end{bmatrix}\end{bmatrix}\\
K_2&=-(R_2+B^{\mathrm{T}}Q_3B)^{-1}B^{\mathrm{T}}Q_3A\\
P_2&=Q_2+A^{\mathrm{T}}Q_3A-A^{\mathrm{T}}Q_3B(R_2+B^{\mathrm{T}}Q_3B)^{-1}B^{\mathrm{T}}Q_3A
\end{aligned} \tag{C.10}$$

After care-
ful observation and manipulation, it is found that we don't need to fully expand H_{01}^{-1} to calculate $-H_{012}^{-1}g_{012}$. Rather, we can do the following:

$$\begin{aligned}
\begin{bmatrix} u_0^* \\ u_1^* \\ u_2^* \end{bmatrix} &= -H_{012}^{-1} g_{012} \\
&= -\begin{bmatrix} \alpha = H_{01}^{-1} \begin{bmatrix} B^{\mathrm{T}}Q_1A + B^{\mathrm{T}}A^{\mathrm{T}}Q_2A^2 + B^{\mathrm{T}}(A^2)^{\mathrm{T}}Q_3A^3 \\ B^{\mathrm{T}}Q_2A^2 + B^{\mathrm{T}}A^{\mathrm{T}}Q_3A^3 \end{bmatrix} + H_{01}^{-1} \begin{bmatrix} B^{\mathrm{T}}A^{\mathrm{T}}K_2^{\mathrm{T}} \\ B^{\mathrm{T}}K_2^{\mathrm{T}} \end{bmatrix} B^{\mathrm{T}}Q_3A^3 \\ \begin{bmatrix} K_2AB & K_2B \end{bmatrix} \alpha + (R_2 + B^{\mathrm{T}}Q_3B)^{-1}B^{\mathrm{T}}Q_3A^3 \end{bmatrix} x_0 \\
&= -\begin{bmatrix} \alpha = H_{01}^{-1} \begin{bmatrix} B^{\mathrm{T}}Q_1A + B^{\mathrm{T}}A^{\mathrm{T}}\Big(Q_2 + A^{\mathrm{T}}Q_3A + K_2^{\mathrm{T}}B^{\mathrm{T}}Q_3A\Big)A^2 \\ B^{\mathrm{T}}(Q_2 + A^{\mathrm{T}}Q_3A + K_2^{\mathrm{T}}B^{\mathrm{T}}Q_3A)A^2 \end{bmatrix} \\ \begin{bmatrix} K_2AB & K_2B \end{bmatrix} \alpha - K_2A^2 \end{bmatrix} x_0 \\
&= -\begin{bmatrix} \alpha = H_{01}^{-1} \begin{bmatrix} B^{\mathrm{T}}Q_1A + B^{\mathrm{T}}A^{\mathrm{T}}P_2A^2 \\ B^{\mathrm{T}}P_2A^2 \end{bmatrix} \\ \begin{bmatrix} K_2AB & K_2B \end{bmatrix} \alpha - K_2A^2 \end{bmatrix} x_0 \\
&= -\begin{bmatrix} K_0 \\ K_1(A - BK_0) \\ \begin{bmatrix} K_2AB & K_2B \end{bmatrix} \alpha - K_2A^2 \end{bmatrix} x_0 \\
&= -\begin{bmatrix} K_0 \\ K_1(A - BK_0) \\ \begin{bmatrix} K_2AB & K_2B \end{bmatrix} \begin{bmatrix} K_0 \\ K_1(A - BK_0) \end{bmatrix} - K_2A^2 \end{bmatrix} x_0 \\
&= -\begin{bmatrix} K_0 \\ K_1(A - BK_0) \\ K_2(A - BK_1)(A - BK_0) \end{bmatrix} x_0
\end{aligned} \tag{C.11}$$

The matrix H_{01} and H_{012} are actually the reduced Hessian $Z^{\mathrm{T}}HZ$, if we view $diag(R_i, Q_i)$ as the true Hessian H, system evolvement equality constraints as $Ax = b$ and $AZ = 0$. Indefinte true Hessian may also result in positive definite reduced Hessian. The invertiblity of KKT matrix only depends on positive definiteness of reduced Hessian.

From the solving procedure, we see that inverting the giant Hessian matrix step by step is reasonable. If we add $\|u_k\|_2 \leq \Delta_k$, then it is quite natural to add $\lambda_k I$ matrix into $(R_k + B^{\mathrm{T}}P_{k+1}B)$ and then do the matrix inversion. This is the motivation of the proposed algorithm in 2.6.3.

TRITA-EECS-EX-2022:96