ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCHOOL OF ENGINEERING

DEPARTMENT of
ELECTRICAL, ELECTRONIC AND INFORMATION ENGINEERING
“Guglielmo Marconi”
DEI

MASTER’S DEGREE IN
AUTOMATION ENGINEERING

MASTER THESIS
in
Distributed Control Systems

A First-order Closed-loop Methodology for Nonlinear Optimal Control

CANDIDATE:

Lorenzo Sforni

SUPERVISOR:

Chiar.mo Prof. Giuseppe Notarstefano

CO-SUPERVISOR:

Dott. Ivano Notarnicola

Academic Year
2019/2020

Session
II

Contents

Abstract **3**

Introduction **4**

Motivations 4
Literature 5
Contributions 6
Organization 8

1 Nonlinear Optimal Control Problem **9**

1.1 Problem Formulation 9
1.2 The Projection Operator Newton Method for Trajectory Optimization 11
1.3 Gradient-based Optimization Methods 16

2 Heavy-Ball Closed-Loop Sequential Method **24**

2.1 Heavy-Ball Sequential Optimization 24
2.2 Heavy-Ball for Constrained Systems 32

3 Algorithm Implementation **36**

3.1 Python Code 36
3.2 Inverse Pendulum Problem 38

Conclusions **52**

A Basics on Optimization **53**

A.1 Descent Methods for Unconstrained Optimization 53
A.2 Sequential Quadratic Programming 54
A.3 Linear Quadratic Regulator 54
A.4 Armijo Rule – Backtracking Line Search 56

Bibliography **58**

Abstract

This thesis is focused on state-of-art numerical optimization methods for nonlinear (discrete-time) optimal control. These challenging problems arise when dealing with complex tasks for autonomous systems (e.g. vehicles or robots) which require the generation of a trajectory that satisfies the system dynamics and, possibly, input and state constraints due to, e.g, actuator limits or safety region of operation.

A general formulation is proposed that allows the implementation of different descent optimization algorithms on optimal control problems exploiting the beneficial effects of state feedback in terms of efficiency and stability. The main idea is the following: at each iteration a new (infeasible) state-input curve is conveniently updated by any descent method, e.g, gradient descent or Newton methods, then a nonlinear feedback controller maps the curve to a trajectory satisfying the dynamics.

Thanks to its inherent flexibility, this strategy provides the opportunity to speed-up the resolution of optimization problems by conveniently choosing the descent method. This thesis proposes, for example, to exploit the Heavy-ball method to speed up the convergence.

It is important to underline that this methodology enjoys recursive feasibility during the algorithm evolution, i.e. at each iteration a system trajectory is available. This feature is extremely important in real-time control schemes since it allows one to stop the algorithm at any iteration and yet have a (suboptimal) system trajectory.

Furthermore, tasks which require the introduction of state and input constraints can be managed introducing an approximate barrier function which embeds the constraints within the cost function.

The second main contribution of this thesis is an original Python toolbox developed in order to implement and compare different optimization methods. Moreover, thanks to a modular approach, with just few adjustments it is possible to change system, cost function and constraints.

Introduction

Motivations

In many and various fields of engineering, from autonomous vehicles to power grids, higher and higher performances are sought together with equally stricter bounds in terms of cost, power consumption and any other consumable resources. These user-defined requirements are translated in control theory as optimality criteria: performance indexes associated to the system under control. This is the field of optimal control which aims at finding a control law for a dynamical system over a period of time such that those objective functions are optimized.

Since it was formalized thanks to the work of Lev Potryagin and Richard Bellman in the 1950s, a lot has been written about dealing with linear and nonlinear systems, however, as the goals' boldness increases, traditional strategies may not be enough. In particular, negative effects of unknown or uncertain dynamics become more and more important as higher performances are requested. This framework becomes even harsher when uncertainties about the environment are introduced. For example, let us consider a small quadrotor which is required to perform a minimum-time trajectory. The optimization is usually performed on a simplified model of both vehicle and environment which may neglect some hidden effects such as drag and propeller thrust.

This thesis can be considered as a first step of a wider and more ambitious project focused on the study and design of optimal control strategies for systems subjected to uncertainties. In this work, indeed, state-of-art numerical optimization methods for nonlinear (discrete-time) optimal control are investigated in order to provide a strong and solid methodological basis for the work to come. The focus on sequential optimization methods, indeed, is a necessary step motivated by the attractive perspective to evaluate how well-known sequential optimization methods could be extended in order to deal with an uncertain system.

Moreover, these sequential algorithms represent a solid basis when evaluating a possible extension towards the control of complex systems governed by a graph dynamics, namely, systems whose behavior is modeled as the interaction between a set of several agents which interact and/or communicate through a graph structure. Sequential algorithm of this kind, in fact, have been already successfully extended to a distributed framework.

Literature

The goal of an optimal control problem is to define a control law capable of minimizing a given cost function and producing a trajectory which satisfies the system dynamics. It worth noticing that an optimal control problem differentiates from a generic optimization problem due to presence of a set of constraints which represent the dynamics of the system and these constraint relations are fixed by the physics of the problem.

Analytical methods for the optimal trajectory cannot be found except for simple cost functions and system dynamics, thus in the majority of practical cases numerical optimization methods must be used.

Two major approaches can be distinguished to formulate and numerically solve a discrete time optimal control problem, the simultaneous and the sequential approach.

The first strategy, the simultaneous approach, exploits the fact that an optimal control problem is large and structured and can thus in principle be solved by any numerical solver. In this approach, all original variables, i.e. controls and states remain optimization variables of the problem. Its name stems from the fact that the solver has to simultaneously solve both, the “simulation” (dynamics) and the “optimization” problem. Usually feasibility of the constraints, namely satisfaction of the model equations, is obtained only asymptotically, therefore simultaneous methods are sometimes referred to as an infeasible path approaches. The direct multiple shooting [1] and direct collocation [2] methods are simultaneous approaches.

The second methods exploit the dynamics constraints in order to eliminate nearly all states by a forward simulation, and in this way we could reduce the variable space of the problem by keeping only the control as decision variables. The resulting problem is called *reduced problem*. The simulation is used to recover the state trajectory, therefore the dynamics is always satisfied. The reduced problem can be tackled by several optimization methods. This strategy is called the sequential approach, because the simulation problem and optimization problem are solved sequentially, one after the other.

A first-order sequential method is proposed in [3]: this is an open-loop sequential method where the original constrained (by the dynamics) problem is remapped as an unconstrained one exploiting a Lagrangian approach. This algorithm constitutes the foundation of the closed-loop sequential method hereby proposed.

In [4], a second-order Newton based sequential method is proposed. This method, the Projection Operator Newton method for Trajectory Optimization (PRONTO), exploits a feedback system to exponentially stabilize the trajectory as the time horizon goes to infinity. This also guarantees a numerical stability in the optimization procedure. Moreover, this method guarantees recursive feasibility during the algorithm evolution, i.e., at each iteration a system trajectory is available.

A major contribution to this work comes from [5] where a closed-loop methodology, inspired by both [3] and [4], is proposed directly in a distributed setup. With respect to [3], the descent direction, in particular, consists of variations of both states and inputs and the trajectory update is performed in closed-loop. Moreover, differently from [4], the descent direction is a generic state-input curve that is not required to statisfy any linearized dynamics.

Another tool widely exploited to solve optimal control problems is dynamic programming [6]. Dynamic programming (DP) is a very different approach to solve optimal control problems than the ones presented previously. The methodology was developed in the fifties and sixties of the 20th century, most prominently by Richard Bellman [7] who also coined the term dynamic programming. Interestingly, dynamic programming is easiest to apply to systems with discrete state and control spaces. When DP is applied to discrete time systems with continuous state spaces, some approximations have to be made, usually by discretization. Generally, this discretization leads to exponential growth of computational cost with respect to the dimension of the state space, what Bellman called "the curse of dimensionality". It is the only but major drawback of DP and limits its practical applicability. On the positive side, DP can easily deal with all kinds of hybrid systems or non-differentiable dynamics, and it even allows us to treat stochastic optimal control with recourse, or minimax games, without much additional effort.

Dealing with optimal control problems, it is important to consider and evaluate the presence of constraints both over the states trajectory and the inputs. In particular, in [8], an approximate barrier function which incorporates the constraints into an unconstrained trajectory functional. This results in a relaxed optimization problem in which constraints are embedded within the cost function. Since this barrier function approach is an interior function method, the approximated solution always satisfies given constraints.

Contributions

In this work a novel methodology for nonlinear optimal control problems with input and state constraints is developed. This methodology relies and extends the idea presented in [4] of using feedback as a mean to solve optimal control problems in a robust and flexible way. The general concept is the following: at each iteration, conveniently update via any descent method a new (infeasible) state-input curve and map the curve to a trajectory satisfying the dynamics thanks to a nonlinear feedback controller, acting as projection operator.

This results in a very flexible and versatile strategy, which allows us to adopt and evaluate different optimization methods in order to speed up the algorithm convergence, thus the resolution of the optimization problem.

In particular, in this thesis the Heavy-ball method, proposed in [9], is incorporated within a closed-loop strategy, evolution of the one proposed in [3]. Here the unpractical open loop integration of the dynamics is substituted by a more efficient and stable closed-loop structure exploiting the beneficial effects of state feedback.

It is important to underline that, thanks to the projection operator, this methodology enjoys recursive feasibility during the algorithm evolution, i.e. at each iteration a system trajectory is available. This feature is extremely important in real-time control schemes since it allows one to stop the algorithm at any iteration and yet have a (suboptimal) system trajectory.

Furthermore, tasks which require the introduction of state and input constraints can be managed introducing an approximate barrier function as described in [8]. This results in a relaxed optimization problem in which constraints are embedded within the cost function. Since this barrier function approach is an interior function method, the approximated solution always satisfies given constraints.

Another important contribution of this thesis consists in a Python implementation of the proposed algorithm. In particular, it allows to compare methods effectiveness and it provides an useful test-bench for the proposed algorithm addressed to the resolution of any discrete time optimal control problem, not only a particular case study. In fact, a ready-to-use platform is presented: the user needs just to define the system dynamics, the cost function and the given constraints.

As a future development of this work, the presented sequential methodology is a key step towards the study of a strategy capable of handling optimal control problems when uncertainties are considered. In fact, this methodology could be extended to a stochastic framework where uncertainties are modeled as disturbances characterized by proper probability distributions. For example, an approach to be investigated is the stochastic gradient descent algorithm. This approach allows the designer to consider, at each iteration of the optimizer, a descent direction associated just to a single disturbance's realization. In fact, as shown also in [10], under certain properties of the gradient function, it can be adopted as an unbiased estimator of the whole stochastic process. Since this algorithm defines, with few information, a descent direction, it fits well the proposed methodology.

Organization

This thesis is organized as follows. In Chapter 1, it is described the problem set-up and the fundamental methodologies are discussed. In particular, it is described PRONTO, from [4], the open loop sequential method in [3] and its closed loop evolution.

In Chapter 2, the proposed first-order closed-loop algorithm enhanced by an Heavy-ball trajectory update is introduced. First an unconstrained set-up is considered, secondly state and input constraints are introduced and managed through the introduction of a barrier function, as proposed in continuous time in [8].

In Chapter 3, the Python implementation of the algorithm is described. Moreover, an illustrative example of the algorithm proposed in the previous chapter is included.

Finally, basics about optimization are added in the Appendix.

Chapter 1

Nonlinear Optimal Control Problem

Optimal control regards the optimization of dynamical systems. Dynamical systems are identified with processes that are evolving in time and that can be characterized by *states* x which allow us to predict the future behavior of the system. Often, the evolution of the system's dynamics can be steered by a suitable choice of inputs here denoted as *controls* u. Typically, these controls shall be chosen optimally in order to optimize an *objective function* and to satisfy some *constraints*.

1.1 Problem Formulation

In this work, the main focus is around *discrete time systems*, namely systems where the time in which the system evolves only takes values on a predefined time grid, usually assumed to be integers. In particular, in order to denote the system state x at a certain discrete-time instant t, $t \in \mathbb{N}$, the notation x_t is adopted. The same index notation is applied to the other time-varying variables. The whole time horizon is denoted as $\mathbb{T} = \{t \in \mathbb{N} \mid t \in [0, T]\}$.

In the general, time-variant, case these systems are characterized by the dynamics:

$$x_{t+1} = f_t(x_t, u_t) \qquad t = 0, \dots, T \tag{1.1}$$

where $f_t : \mathbb{R}^{n_x \times n_u} \to \mathbb{R}^{n_x}$, $f_t \in C^2$ represent the system dynamics, which maps, at each time instant t, a pair of state x_t and input u_t, into the system's state at the next time instant x_{t+1} on a time horizon of length T, with control input vectors $u_0, \dots, u_{T-1} \in \mathbb{R}^{n_u}$ and $T + 1$ state vectors $x_0, \dots, x_T \in \mathbb{R}^{n_x}$.

A typical property of a dynamic system is that knowledge of an initial state x_0 and a control input trajectory $u(t)$ for all $t \in [0, T-1]$ allows one to determine the evolution of the state trajectory $x(t)$ for all $t \in [1, T]$, where T is the final time. In particular, knowing the initial state x_0 and the sequence of control input $u_0, \dots, u_{T-1}$, it is possible to recursively evaluate the dynamics function $f_t(x_t, u_t)$ and retrieve the whole states sequence $x_1, \dots, x_T$.

The aim of optimal control is to define an input sequence capable to optimize a given objective, or cost, function.

In particular, in this work, discrete-time optimal control problems of this form are considered:

$$\begin{aligned} \min_{\substack{x_1,\dots,x_N \\ u_0,\dots,u_{T-1}}} \quad & \sum_{t=0}^{T-1} \ell_t(x_t, u_t) + m(x_T) && (1.2a)\\ \text{subj. to} \quad & x_{t+1} = f_t(x_t, u_t) \qquad t \in \mathbb{T}_{[0,T-1]} && (1.2b)\\ & g(x_t, u_t) \leq 0 && (1.2c)\\ & x_0 = x_{\text{init}} \end{aligned}$$

where $T \in \mathbb{N}$ discrete time horizon, $x_t \in \mathbb{R}^{n_x}$ state of the system at time t, $u_t \in \mathbb{R}^{n_u}$ control input at time t, $\ell(x_t, u_t) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ instantaneous cost function, $m(x_T) : \mathbb{R}^{n_x} \to \mathbb{R}$ terminal cost function, $g_t(x_t, u_t) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_g}$ instantaneous inequality constraints function with n_g number of constraints.

Functions, f, l, m are supposed to be sufficiently regular. Both f_t and ℓ_t may be time-invariant. In this work, in fact, $f_t(x_t, u_t) \equiv f(x_t, u_t)$, $f_t : \mathbb{R}^{n_x \times n_u} \to \mathbb{R}^{n_x}$, and $\ell_t(x_t, u_t) \equiv \ell(x_t, u_t)$, $\ell(x_t, u_t) : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ for all $t \in \mathbb{T}_{[0,T]}$

Accordingly to the kind of functions and systems considered, different strategies can be applied in order to define the optimal control sequence.

To ease the notation, let introduce a stacked version of the optimization variables, i.e., let $\mathbf{x} \in \mathbb{R}^{n_x T}$ and $\mathbf{u} \in \mathbb{R}^{n_u T}$ be defined as:

$$\begin{aligned} \mathbf{x} &:= \left[x_1^\top, \dots, x_T^\top\right]^\top \\ \mathbf{u} &:= \left[u_0^\top, \dots, u_{T-1}^\top\right]^\top \end{aligned}$$

In this context it is possible to introduce the concept of trajectory:

Definition 1 (Trajectory). *A pair $(\mathbf{x}, \mathbf{u})$ with $\mathbf{x} \in \mathbb{R}^{n_x T}$ and $\mathbf{u} \in \mathbb{R}^{n_u T}$ is called trajectory if the components satisfy the dynamics, namely, $x_{t+1} = f(x_t, u_t)$ for all $t \in \mathbb{T}$.*

Otherwise, a generic pair $(\boldsymbol{\alpha}, \boldsymbol{\mu})$ with $\boldsymbol{\alpha} \in \mathbb{R}^{n_x T}$ and $\boldsymbol{\mu} \in \mathbb{R}^{n_u T}$ is called *curve*.

In particular, two major strategies can be distinguished to formulate and numerically solve a discrete time optimal control problem, the simultaneous and the sequential approach. While the simultaneous exploits as optimization variables both $\mathbf{x}$ and $\mathbf{u}$ quite often resulting in model equations (1.1) satisfied only once the iterations are converged, the sequential approach exploits the knowledge of the system dynamics to reduce the variable space.

The sequential approach exploits model equations (1.1) in order to keep, as optimization variables, only $\mathbf{u}$. States $x_1, \ldots, x_T$ are eliminated recursively by:

$$\begin{aligned}\phi_0(\mathbf{u}) &= x_0 \\ \phi_{t+1}(\mathbf{u}) &= f(\phi_t(\mathbf{u}), u_t)\end{aligned} \tag{1.4a}$$

where $\phi_t : \mathbb{R}^{n_u T} \to \mathbb{R}^{n_x}$ for all $t \in \mathbb{T}_{[0,T]}$.

In this way the problem (1.2) is equivalent to the reduced problem:

$$\min_{u_0,\ldots,u_{T-1}} \sum_{t=0}^{T-1} \ell(\phi_t(\mathbf{u}), u_t) + m(x_T) \tag{1.5a}$$

$$\text{subj. to} \quad \mathbf{u} \in \mathbb{R}^{n_u T} \tag{1.5b}$$

$$g_t(\phi_t(\mathbf{u}), u_t) \leq 0 \tag{1.5c}$$

Until Section 2.2, however, inequalities contraints are neglected, leading to problem:

$$\begin{aligned}\min_{u_0,\ldots,u_{T-1}} & \sum_{t=0}^{T-1} \ell(\phi_t(\mathbf{u}), u_t) + m(x_T) \\ \text{subj. to} \quad & \mathbf{u} \in \mathbb{R}^{n_u T}\end{aligned} \tag{1.6}$$

which is an unconstrained problem with unique optimization variable $\mathbf{u}$.

It is important to underline that, in this work, analyzed methods are designed in order to produce, at each iteration, a trajectory capable of satisfying the dynamic of the system, namely, a *feasible* trajectory is always available (even if sub-optimal). Thus, the algorithm can be stopped at any iteration with an important guarantee: a trajectory to track is available.

1.2 The Projection Operator Newton Method for Trajectory Optimization

The projection operator approach for trajectory optimization, introduced in [4], is an iterative algorithm which, in its easiest formulation, allows one to perform local Newton (or quasi-Newton) optimization of the cost functional over the set of trajectories of a nonlinear system with fixed initial conditions.

By now, a simplified optimal control problem is considered, i.e. inequalities constraints are neglected. These are going to be addressed later in Section 2.2, thanks to the introduction of a suitable approximate barrier function. Thus, problem (1.2) can then be expressed as:

$$\begin{aligned} \min_{\substack{x_1,\ldots,x_N \\ u_0,\ldots,u_{T-1}}} \; & \sum_{t=0}^{T-1} \ell(x_t,u_t) + m(x_T) \\ \text{subj. to} \quad & x_{t+1} = f(x_t,u_t) \qquad t \in \mathbb{T}_{[0,T-1]} \\ & x_0 = x_{\text{init}} \end{aligned} \tag{1.7}$$

To ease the notation, let us introduce:

$$\xi = (\boldsymbol{\alpha}, \boldsymbol{\mu}) \tag{1.8a}$$

$$\eta = (\mathbf{x}, \mathbf{u}) \tag{1.8b}$$

Remembering that $x_t, \alpha_t \in \mathbb{R}^{n_x}$ $\forall t \in \mathbb{T}_{[0,T]}$ and $u_t, \mu_t \in \mathbb{R}^{n_u}$ $\forall t \in \mathbb{T}_{[0,T]}$ and accordingly to the previously introduced notation:

$$\boldsymbol{\alpha} := \left[\alpha_1^\top, \ldots, \alpha_T^\top\right]^\top$$

$$\boldsymbol{\mu} := \left[\mu_0^\top, \ldots, \mu_{T-1}^\top\right]^\top$$

It is possible to observe that equation (1.8a) represents a generic state-input curve $\xi \in \mathbb{R}^{n_x T} \times \mathbb{R}^{n_u T}$, not necessarily feasible, and equation (1.8b) represents a system trajectory $\eta \in \mathbb{R}^{n_x T} \times \mathbb{R}^{n_u T}$ satisfying Definition 1.

In particular, given a generic state-input curve $\xi = (\boldsymbol{\alpha}, \boldsymbol{\mu})$, it is possible to associate a trajectory $\eta = (\mathbf{x}, \mathbf{u})$ obtained by setting $x_0 = x_{\text{init}}$ and by forward simulation of the following closed-loop dynamics (the projection operator):

$$\begin{aligned} u_t &= \mu_t + K_t(\alpha_t - x_t) \\ x_{t+1} &= f(x_t, u_t) \end{aligned} \tag{1.10}$$

the gain matrix $K_t \in \mathbb{R}^{n_u \times n_x}$ is exploited to give the operator stability properties, namely it has to exponentially stabilize the trajectory $x_t \in \mathbb{T}_{[0,T]}, u_t \in \mathbb{T}_{[0,T-1]}$ as $T \to +\infty$. A suitable gain can be constructed via different strategies, e.g., it can be the solution of a finite horizon linear regulator problem associated to the system linearized about a trajectory $(\mathbf{x}, \mathbf{u})$.

In fact, defining the trajectory manifold, namely the space defined by all the system trajectories, as $\mathcal{T}$, to work on that trajectory manifold it is necessary, as shown in [11], to project state-control curves onto $\mathcal{T}$ by using a local linear time-varying trajectory tracking controller, the so-called projection operator (1.10), which relies on the use of a redundant representation of a system trajectory.

This procedure is then denoted as:

$$\eta = \mathcal{P}(\xi) \tag{1.11}$$

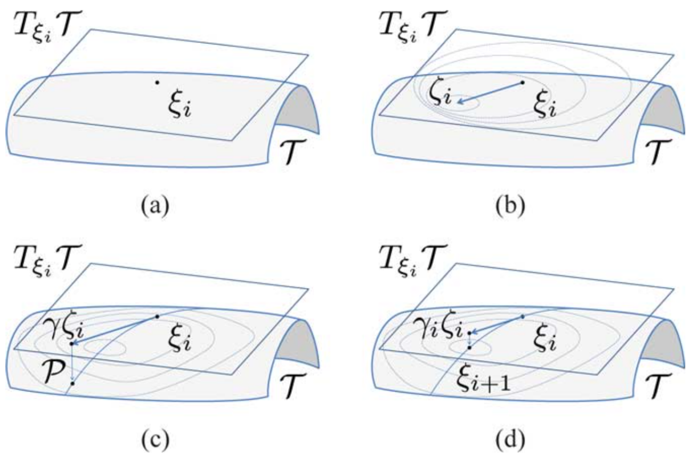


Figure 1.1: Projection operator approach: the linearization of the control system about the trajectory ξ_i defines the tangent space to the trajectory $\mathcal{T}$ manifold at ξ_i; (b) the constrained minimization over the tangent space of the second-order approximation of the extended cost functional yields the search direction ζ_i ; (c) the optimal step size γ_i is computed through a line search along ζ_i ; (d) the search direction and step size are combined to obtain a new update trajectory ξ_{i+1} . (a)Trajectory manifold. (b) Search direction. (c) Line search. (d) Update.

where $\mathcal{P} : \mathbb{R}^{n_x T} \times \mathbb{R}^{n_u T} \to \mathbb{R}^{n_x T} \times \mathbb{R}^{n_u T}$ represents the projection operator. Note that, if ξ is a trajectory of the system, then it is a fixed point of $\mathcal{P}$, namely $\xi = \mathcal{P}(\xi)$. Formally, $\xi \in \mathcal{T}$ if and only if $\xi = \mathcal{P}(\xi)$.

With a suitable projection operator in hand, $\xi = (\boldsymbol{\alpha}, \boldsymbol{\mu})$ can be used as a redundant representation of the trajectory $\eta = \mathcal{P}(\xi)$.

Thus, the idea behind [4] is to exploit the projection operator in an iterative algorithm such that, at each iteration, the new update trajectory ξ^{k+1} is computed projecting the curve $\xi^k + \beta^k \zeta^k$ onto the trajectory manifold $\mathcal{T}$, where ζ^k is the descent direction obtained by the resolution of a Linear Quadratic Regulator (LQR) problem about the current trajectory ξ^k and $\beta^k \in \mathbb{R}$ a conveniently chosen step-size. This method is defined as PRojection Operator Newton method for Trajectory Optimization (PRONTO).

This approach could be synthetically expressed as:

$$\xi^{k+1} = \mathcal{P}(\xi^k + \beta^k \zeta^k) \qquad k = 1, 2, \ldots \tag{1.12}$$

An illustration of the approach is shown in figure 1.1

Including the projection operator, problem (1.7) is then rewritten as:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{u},\boldsymbol{\alpha},\boldsymbol{\mu}} \; & \sum_{t=0}^{T-1} \ell(x_t, u_t) + m(x_T) \\ \text{subj. to} \quad & x_{t+1} = f(x_t, u_t) \qquad t \in \mathbb{T}_{[0,T-1]} \\ & u_t = \mu_t + K_t(\alpha_t - x_t) \\ & x_0 = \alpha_0 = x_{\text{init}} \end{aligned} \tag{1.13}$$

Introducing the functional:

$$F(\boldsymbol{\alpha}, \boldsymbol{\mu}) = \sum_{t=0}^{T-1} \ell(\alpha_t, \mu_t) + m(\alpha_T)$$

the original optimal control problem (1.13) is equivalent to the constrained optimization problem:

$$\min_{(\boldsymbol{\alpha},\boldsymbol{\mu}) \in \mathcal{T}} F(\boldsymbol{\alpha}, \boldsymbol{\mu}) \tag{1.14}$$

Thus, with the concept of projection operator in mind, it can be recast as an unconstrained optimal control problem given by:

$$\min_{\boldsymbol{\alpha},\boldsymbol{\mu}} F(\mathcal{P}(\boldsymbol{\alpha}, \boldsymbol{\mu})) \tag{1.15}$$

These two problems, as shown in continuous time in [4], are essentially equivalent in the sense that a solution to the constrained problem (1.14) is a solution to the unconstrained problem (1.15), while a solution to the second problem is, projected by $\mathcal{P}$, a solution to the first problem.

Using these facts, a trajectory optimization methodology is effectively developed in an unconstrained manner by working with the cost functional $J(\boldsymbol{\alpha}, \boldsymbol{\mu}) = F(\mathcal{P}(\boldsymbol{\alpha}, \boldsymbol{\mu}))$.

In fact, exploiting the functional $J(\boldsymbol{\alpha}, \boldsymbol{\mu})$, the constrained optimal control problem can be equivalently written as:

$$\begin{aligned} \min_{\boldsymbol{\alpha},\boldsymbol{\mu}} \quad & J(\boldsymbol{\alpha}, \boldsymbol{\mu}) \\ \text{subj. to} \quad & \boldsymbol{\alpha} \in \mathbb{R}^{n_x T}, \boldsymbol{\mu} \in \mathbb{R}^{n_u T} \end{aligned} \tag{1.16}$$

Then, a (projected) Newton method can be applied for unconstrained optimization to solve problem (1.16). Specifically, given an initial guess $(\mathbf{x}^0, \mathbf{u}^0)$, for all $k \geq 0$ the method reads:

$$(\mathbf{x}^{k+1}, \mathbf{u}^{k+1}) = \mathcal{P}((\mathbf{x}^k, \mathbf{u}^k) + (\mathbf{z}^k, \mathbf{v}^k))$$

where the update direction $(\mathbf{z}^k, \mathbf{v}^k)$, $\mathbf{z}^k \in \mathbb{R}^{n_x T}$, $\mathbf{v}^k \in \mathbb{R}^{n_u T}$ is obtained from a quadratic (first or second-order) approximation of $J(\boldsymbol{\alpha}, \boldsymbol{\mu})$ about the current iterate $(\mathbf{x}^k, \mathbf{u}^k)$. Notice that since Newton method works only locally, one can also use a globalization technique by introducing a step-size $\beta^k \in (0,1]$ in the update and obtain the following modified (projected) Newton method:

$$(\mathbf{x}^{k+1}, \mathbf{u}^{k+1}) = \mathcal{P}((\mathbf{x}^k, \mathbf{u}^k) + \beta^k(\mathbf{z}^k, \mathbf{v}^k))$$

As for the update direction, it can be obtained by solving the following quadratic program about a second order approximation of $J(\boldsymbol{\alpha}, \boldsymbol{\mu})$:

$$(\mathbf{z}^k, \mathbf{v}^k) = \arg \min_{\Delta\mathbf{x}, \Delta\mathbf{u}} \nabla J(\boldsymbol{\alpha}^k, \boldsymbol{\mu}^k)^\top \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{u} \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{u} \end{bmatrix}^\top H^k \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{u} \end{bmatrix} \tag{1.17}$$

where H^k can be chosen in several ways, giving rise to different Newton methods, e.g. a first order method if $H^k = \mathbf{I}$, a second-order Newton method if $H^k = \nabla^2 J(\boldsymbol{\alpha}^k, \boldsymbol{\mu}^k)$. In this work a first order approximation is adopted.

Being $J(\boldsymbol{\alpha}, \boldsymbol{\mu})$ a composition, its derivatives involve, by the chain rule, the derivatives of the projection operator as well. It can be proved [11] that the projection operator derivative is a projection operator as well. Specifically, it is a projection operator on the tangent space $T_{(\mathbf{x}^k,\mathbf{u}^k)}\mathcal{T}$, where $T_{(\mathbf{x}^k,\mathbf{u}^k)}$ denotes the space tangent to the trajectory manifold $\mathcal{T}$ in correspondence of a linearization about the current trajectory $(\mathbf{x}^k, \mathbf{u}^k)$. Hence, one can restrict the search domain of $(\Delta\mathbf{x}, \Delta\mathbf{u})$ in problem (1.17) over the tangent space, i.e., the set of trajectories satisfying the linearization of the system dynamics about $(\mathbf{x}^k, \mathbf{u}^k)$ (with initial conditions equal to zero).

Therefore, problem (1.17), boils down to:

$$\begin{aligned} (\mathbf{z}^k, \mathbf{v}^k) = \arg \min_{\Delta\mathbf{x}, \Delta\mathbf{u}} & \begin{bmatrix} 1 \\ \Delta\mathbf{x} \\ \Delta\mathbf{u} \end{bmatrix}^\top \begin{bmatrix} 0 & \nabla J(\boldsymbol{\alpha}^k, \boldsymbol{\mu}^k)^\top \\ \nabla J(\boldsymbol{\alpha}^k, \boldsymbol{\mu}^k) & H^k \end{bmatrix} \begin{bmatrix} 1 \\ \Delta\mathbf{x} \\ \Delta\mathbf{u} \end{bmatrix} \\ \text{subj. to } & \Delta x_{t+1} = A_t^k \Delta x_t + B_t^k \Delta u_t, \; t \in \mathbb{T}_{[0,T-1]} \\ & \Delta x_0 = 0 \end{aligned} \tag{1.18}$$

Notice that (1.18) is a time-varying linear quadratic regulator problem since the objective function is a quadratic function incorporating derivatives of both the cost and the dynamics, while the constraint is linear since it describes the linearized dynamics. This method is summarized in algorithm 1 remembering that:

$$\begin{aligned} a_t^k &= \nabla_{x_t} \ell(x_t^k, u_t^k) \\ b_t^k &= \nabla_{u_t} \ell(x_t^k, u_t^k) \\ A_t^k &= \nabla_{x_t} f(x_t^k, u_t^k)^\top \\ B_t^k &= \nabla_{u_t} f(x_t^k, u_t^k)^\top \end{aligned}$$

Algorithm 1 PRONTO

```
Require: trajectory $(\mathbf{x^0}, \mathbf{u^0})$ with $x_0^0 = x_{init}$
  for $k = 0, 1, 2, \dots$ do
    compute $K_t^k$ for all $t \in \mathbb{T}_{[0,T-1]}$
    compute
      $$\begin{aligned} (\mathbf{z}^k, \mathbf{v}^k) = \arg\min_{\Delta\mathbf{x},\Delta\mathbf{u}} & \sum_{t=0}^{T-1} \begin{bmatrix} a_t^k \\ b_t^k \end{bmatrix}^\top \begin{bmatrix} \Delta x_t \\ \Delta u_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{u} \end{bmatrix}^\top H^k \begin{bmatrix} \Delta\mathbf{x} \\ \Delta\mathbf{u} \end{bmatrix} \\ \text{subj. to } & \Delta x_{t+1} = A_t^k \Delta x_t + B_t^k \Delta u_t,\ t \in \mathbb{T}_{[0,T-1]} \\ & \Delta x_0 = 0 \end{aligned}$$
    compute step-size $\beta^k$
    set initial condition $x_0^{k+1} = x_{init}$
    for $t = 0, \dots, T-1$ do
      compute new curve
        $$\begin{aligned} \alpha_t^{k+1} &= x_t^k + \beta z_t^k \\ \mu_t^{k+1} &= u_t^k + \beta v_t^k \end{aligned}$$
      compute new trajectory
        $$\begin{aligned} u_t^{k+1} &= \mu_t^k + K_t^k(\alpha_t^{k+1} - x_t^{k+1}) \\ x_{t+1}^{k+1} &= f(x_t^{k+1}, u_t^{k+1}) \end{aligned}$$
    end for
  end for
```

1.3 Gradient-based Optimization Methods

Even if PRONTO proved its effectiveness, it results in a method were the descent direction is constrained to lie on the tangent space to the trajectory manifold. Now, the interest is to study an iterative method were the descent direction is not constrained to satisy any linearized dynamics. Thus, some sequential, gradient based approaches are considered. Recalling the unconstrained version of problem (1.2):

$$\min_{\mathbf{x},\mathbf{u}} \sum_{t=0}^{T-1} \ell(x_t, u_t) + m(x_T) \tag{1.19a}$$

$$\text{subj. to} \quad x_{t+1} = f(x_t, u_t) \qquad t \in \mathbb{T}_{[0,T-1]} \tag{1.19b}$$

$$x_0 = x_{\text{init}}$$

for a given $x_{\text{init}} \in \mathbb{R}^{n_x}$.

It is possible to write a compact version of the cost function (1.19a), introducing $F : \mathbb{R}^{n_x T} \times \mathbb{R}^{n_u T} \to \mathbb{R}$ defined as:

$$F(\mathbf{x}, \mathbf{u}) = \sum_{t=0}^{T-1} \ell(x_t, u_t) + m(x_T) \tag{1.20}$$

Problem (1.19) then reads as:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{u}} \quad & F(\mathbf{x}, \mathbf{u}) \\ \text{subj. to} \quad & x_{t+1} = f(x_t, u_t) \qquad t \in \mathbb{T}_{[0,T-1]} \\ & x_0 = x_{\text{init}} \end{aligned} \tag{1.21}$$

1.3.1 Open-loop Sequential Method

A first sequential gradient-based approach is originally proposed in Section 1.9 of [3].

Applying a sequential approach previously discussed to problem (1.21), let us introduce an implicit formulation of the (open-loop) dynamics:

$$\begin{aligned} & x_{t+1} = f(x_t, u_t) \qquad t \in \mathbb{T}_{[0,T-1]} \\ & x_0 = x_{\text{init}} \end{aligned} \tag{1.22}$$

Let:

$$h : \mathbb{R}^{n_x T} \times \mathbb{R}^{n_u T} \to \mathbb{R}^{n_x T} \tag{1.23}$$

be defined as:

$$h(\mathbf{x}, \mathbf{u}) = \begin{bmatrix} f(x_0, u_0) - x_1 \\ \vdots \\ f(x_{T-1}, u_{T-1}) - x_T \end{bmatrix} \tag{1.24}$$

Then, the optimal control problem (1.21) can be compactly rewritten as:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{u}} \quad & F(\mathbf{x}, \mathbf{u}) \\ \text{subj. to} \quad & h(\mathbf{x}, \mathbf{u}) = 0 \end{aligned} \tag{1.25}$$

with $\mathbf{x} \in \mathbb{R}^{n_x}$ and $\mathbf{u} \in \mathbb{R}^{n_u T}$. Notice that we ignore the initial condition $x_0 = x_{\text{init}}$ since we embed it in the first constraint of $h(\mathbf{x}, \mathbf{u}) = 0$, i.e. imposing $f(x_{\text{init}}, u_0) - x_1 = 0$.

It is then possible to make the following assumption:

Assumption 1. *There exists a smooth map $\phi : \mathbb{R}^{n_x T} \times \mathbb{R}^{n_u T} \to \mathbb{R}^{n_x T}$ such that:*

$$h(\phi(\mathbf{u}), \mathbf{u}) = 0$$

for all $\mathbf{u} \in \mathbb{R}^{n_u T}$.

In light of Assumption 1, given a generic input vector $\mathbf{u} \in \mathbb{R}^{n_u T}$ we have that:

$$\phi_{t+1}(\mathbf{u}) = f(\mathbf{u}, u_t) \qquad t \in \mathbb{T}_{[0,T-1]} \tag{1.26}$$

with ϕ_t being the t-th component of ϕ. Hence, a state trajectory $\mathbf{x}$ of the system $x_{t+1} = f(x_t, u_t)$ can be obtained from $\mathbf{u}$ by setting:

$$x_t = \phi_t(\mathbf{u}) \qquad t \in \mathbb{T}_{[1,T]} \tag{1.27}$$

Then, Problem (1.19) can be rewritten in the reduced form:

$$\begin{aligned} &\min_{u_0,\dots,u_{T-1}} \sum_{t=0}^{T-1} F(\phi(\mathbf{u}), \mathbf{u}) \\ &\text{subj. to} \quad \mathbf{u} \in \mathbb{R}^{n_u T} \end{aligned} \tag{1.28}$$

which is an *unconstrained* optimization problem. Therefore, we can resort to descent methods to solve this problem.

In this case, numerical solutions to problem (1.28) are computed via a gradient (steepest) descent method in Algorithm 2 where, at each iteration k a trajectory $(\mathbf{x}^{k+1}, \mathbf{u}^{k+1})$ is available. The descent direction $\mathbf{v}^k$ is computed via (1.29) where $a_t^k \in \mathbb{R}^{n_x}$, $b_t^k \in \mathbb{R}^{n_u}$, $A_t^k \in \mathbb{R}^{n_x \times n_x}$, $B_t^k \in \mathbb{R}^{n_x \times n_m}$ are defined as:

$$\begin{aligned} a_t^k &= \nabla_{x_t} \ell(x_t^k, u_t^k) \\ b_t^k &= \nabla_{u_t} \ell(x_t^k, u_t^k) \\ A_t^k &= \nabla_{x_t} f(x_t^k, u_t^k)^\top \\ B_t^k &= \nabla_{u_t} f(x_t^k, u_t^k)^\top \end{aligned}$$

The trajectory update is then performed in (1.30) via the open loop dynamics (1.1).

Algorithm 2 enjoys dynamic feasibility at each iteration, (i.e., a trajectory satisfying the dynamics is available at each iteration). Nevertheless, the algorithm suffers of numerical instability, because of the open loop update.

Algorithm 2 Open-Loop Sequential Method

```
Require: trajectory $(\mathbf{x^0}, \mathbf{u^0})$ with $x_0^0 = x_{init}$
  for $k = 0, 1, 2, \dots$ do
    set $\lambda_T^k = \nabla m(x_T^k)$
    for $t = T-1, \dots, 0$ do
      compute
          $\lambda_t^k = A_t^{k\top} \lambda_{t+1}^k + a_t^k$
          $v_t^k = -B_t^{k\top} \lambda_{t+1}^k - b_t^k$          (1.29)
    end for
    compute step-size $\beta^k$
    set initial condition $x_0^{k+1} = x_{init}$
    for $t = 0, \dots, T-1$ do
      compute new trajectory
          $u_t^{k+1} = u_t^k + \beta^k v_t^k$
          $x_{t+1}^{k+1} = f(x_t^k, u_t^k)$          (1.30)
    end for
  end for
```

1.3.2 Closed-loop Sequential Method

In this section it is presented a closed-loop algorithm to solve the nonlinear optimal control problem (1.19). A first example of this strategy is originally proposed in a distributed framework in [5].

The basic idea is to exploit the advantages, in terms of stability, of the closed-loop dynamics introduced via the projection operator (1.10) in PRONTO. This goal is achieved introducing a redundant constraint in the dynamics in order to enforce it to enjoy a closed loop structure.

Namely, problem (1.19) is reformulated as:

$$\min_{\mathbf{x},\mathbf{u},\boldsymbol{\alpha},\boldsymbol{\mu}} \sum_{t=0}^{T-1} \ell(x_t, u_t) + m(x_T) \tag{1.31a}$$

$$\text{subj. to} \quad x_{t+1} = f(x_t, u_t) \qquad t \in \mathbb{T}_{[0,T-1]} \tag{1.31b}$$

$$u_t = \mu_t + K_t(\alpha_t - x_t) \tag{1.31c}$$

$$x_0 = \alpha_0 = x_{init}$$

where the redundant constraint is represented by (1.31c), which introduces a state feedback through matrix K_t, to be suitably chosen, together with a feed-forward action represented by μ_t. Notice that $x_t, \alpha_t \in \mathbb{R}^{n_x}$ $\forall t \in \mathbb{T}_{[0,T]}$ and $u_t, \mu_t \in \mathbb{R}^{n_u}$ $\forall t \in \mathbb{T}_{[0,T-1]}$.

It is important to underline that the pair $(\boldsymbol{\alpha}, \boldsymbol{\mu})$ represents a generic state-input curve that may not statisfy the dynamics.

Gains $K_t \in \mathbb{R}^{n_u \times n_x}$ are supposed to be suitably assigned, for example solving the Linear Quadratic Regulator (LQR) problem with respect to a linearization of the nonlinear system about a reference trajectory.

In order to adopt a sequential approach, it is necessary to express implicitly the nonlinear dynamics in (1.31b) and (1.31c).
Let

$$h : \mathbb{R}^{n_x T} \times \mathbb{R}^{n_u T} \times \mathbb{R}^{n_x T} \times \mathbb{R}^{n_u T} \to \mathbb{R}^{n_x T + n_u T}$$

be defined as:

$$h(\mathbf{x}, \mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \begin{bmatrix} f(x_0, u_0) - x_1 \\ \vdots \\ f(x_{T-1}, u_{T-1}) - x_T \\ \mu_0 + K_0(\alpha_0 - x_0) - u_0 \\ \vdots \\ \mu_{T-1} + K_0(\alpha_{T-1} - x_{T-1}) - u_{T-1} \end{bmatrix} \tag{1.32}$$

Then, the optimal control problem can be compactly rewritten as:

$$\begin{aligned} \min_{\mathbf{x}, \mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\mu}} \quad & F(\mathbf{x}, \mathbf{u}) \\ \text{subj. to} \quad & h(\mathbf{x}, \mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = 0 \end{aligned} \tag{1.33}$$

where $\mathbf{x}, \boldsymbol{\alpha} \in \mathbb{R}^{n_x T}$ and $\mathbf{u}, \boldsymbol{\mu} \in \mathbb{R}^{n_u T}$ and F as:

$$F(\mathbf{x}, \mathbf{u}) = \sum_{t=0}^{T-1} \ell(x_t, u_t) + m(x_T) \tag{1.34}$$

Recalling the role of $\phi(\mathbf{u})$ in the previously introduced open-loop Sequential method (cf. Assumption 1), a similar reasoning is exploited in this section. Let us introduce a map that transform a generic state-input curve $(\boldsymbol{\alpha}, \boldsymbol{\mu})$ into a trajectory $(\mathbf{x}, \mathbf{u})$ of (1.31b). Specifically, it is made the following assumption:

Assumption 2. *There exists a smooth map $\phi : \mathbb{R}^{n_x T} \times \mathbb{R}^{n_u T} \to \mathbb{R}^{n_x T}$ and $\gamma : \mathbb{R}^{n_x T} \times \mathbb{R}^{n_u T} \to \mathbb{R}^{n_u T}$ such that:*

$$h(\phi(\boldsymbol{\alpha}, \boldsymbol{\mu}), \gamma(\boldsymbol{\alpha}, \boldsymbol{\mu}), \boldsymbol{\alpha}, \boldsymbol{\mu}) = 0$$

for all $(\boldsymbol{\alpha}, \boldsymbol{\mu}) \in \mathbb{R}^{n_x T} \times \mathbb{R}^{n_u T}$.

In light of Assumption 2, given a generic state-input curve $(\boldsymbol{\alpha}, \boldsymbol{\mu})$ we have that:

$$\phi_{t+1} = f(\phi_t(\boldsymbol{\alpha}, \boldsymbol{\mu}), \gamma_t(\boldsymbol{\alpha}, \boldsymbol{\mu})) \qquad t \in \mathbb{T}_{[0,T-1]} \tag{1.35}$$

with ϕ_t, γ_t being the t-th component of ϕ and γ respectively. Hence, a trajectory $(\mathbf{x}, \mathbf{u})$ of the system $x_{t+1} = f(x_t, u_t)$ can be obtained from $(\boldsymbol{\alpha}, \boldsymbol{\mu})$ by setting:

$$x_t = \phi_t(\boldsymbol{\alpha}, \boldsymbol{\mu}) \qquad t \in \mathbb{T}_{[1,T]} \tag{1.36a}$$

$$u_t = \gamma_t(\boldsymbol{\alpha}, \boldsymbol{\mu}) \qquad t \in \mathbb{T}_{[0,T-1]} \tag{1.36b}$$

In light of these assumptions, the projection operator procedure can be then denoted as:

$$(\mathbf{x}, \mathbf{u}) = \mathcal{P}(\boldsymbol{\alpha}, \boldsymbol{\mu}) = \begin{bmatrix} \phi(\boldsymbol{\alpha}, \boldsymbol{\mu}) \\ \gamma(\boldsymbol{\alpha}, \boldsymbol{\mu}) \end{bmatrix} \tag{1.37}$$

Finally, problem (1.31) can be rewritten in the reduced form:

$$\begin{aligned} \min_{u_0,\dots,u_{T-1}} \quad & \sum_{t=0}^{T-1} \ell(\phi_t(\boldsymbol{\alpha}, \boldsymbol{\mu}), \gamma_t(\boldsymbol{\alpha}, \boldsymbol{\mu})) + m(\phi_T(\boldsymbol{\alpha}, \boldsymbol{\mu})) \\ \text{subj. to} \quad & \boldsymbol{\alpha} \in \mathbb{R}^{n_x T}, \boldsymbol{\mu} \in \mathbb{R}^{n_u T} \end{aligned} \tag{1.38}$$

which is an *unconstrained* optimization problem. Therefore, we can resort to descent methods to solve this problem.

In general, the current solution iteratively evolves as:

$$\begin{bmatrix} \boldsymbol{\alpha}^{k+1} \\ \boldsymbol{\mu}^{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{x}^k \\ \mathbf{u}^k \end{bmatrix} + \beta^k \begin{bmatrix} \mathbf{z}^k \\ \mathbf{v}^k \end{bmatrix} \tag{1.39}$$

where, $\mathbf{z}^k \in \mathbb{R}^{n_x T}$, $\mathbf{v}^k \in \mathbb{R}^{n_u T}$ are descent directions.

Accordingly to this iterative approach, at each step, a properly defined algorithm provides a descent direction $(\mathbf{z}^k, \mathbf{v}^k)$ moving from the cost functional $J(\boldsymbol{\alpha}^k, \boldsymbol{\mu}^k)$. Then a step size β^k is computed, accordingly to some criteria, e.g. backtracking line search. Combining the search direction $(\mathbf{z}^k, \mathbf{v}^k)$ with step size β^k a new update trajectory $(\mathbf{x}^{k+1}, \mathbf{u}^{k+1})$ is computed by projecting the curve $(\boldsymbol{\alpha}^{k+1}, \boldsymbol{\mu}^{k+1})$ onto the trajectory manifold $\mathcal{T}$ and the algorithm restarts (unless a termination condition is met).

This methodology is schematically illustrated in Algorithm 3. Thanks to its inherent flexibility, this strategy provides the opportunity to speed-up the resolution of optimization problems by conveniently choosing the descent method. It is important to underline that, thanks to the projection operator, this methodology enjoys recursive feasibility during the algorithm evolution, i.e. at each iteration a system trajectory is available. This feature is extremely important in real-time control schemes since it allows one to stop the algorithm at any iteration and yet have a (suboptimal) system trajectory.

Algorithm 3 Closed-Loop Projected Methodology

```
Require: trajectory $(\mathbf{x}^0, \mathbf{u}^0)$ with $x_0^0 = x_{init}$
  for $k = 0, 1, 2, \dots$ do
    Compute $K_t,\ t = 0, \dots, T-1$
    Compute a descent direction $(\mathbf{z}^k, \mathbf{v}^k)$ based on $(\mathbf{x}^k, \mathbf{u}^k)$
    Compute a new curve $(\boldsymbol{\alpha}^{k+1}, \boldsymbol{\mu}^{k+1})$ based on $(\boldsymbol{\alpha}^k, \boldsymbol{\mu}^k)$ and $(\mathbf{z}^k, \mathbf{v}^k)$.
    Update the new trajectory
        $(\mathbf{x}^{k+1}, \mathbf{u}^{k+1}) = \mathcal{P}(\boldsymbol{\alpha}^{k+1}, \boldsymbol{\mu}^{k+1})$
  end for
```

There exist also a projected version of the curve update, where the new curve $(\boldsymbol{\alpha}^{k+1}, \boldsymbol{\mu}^{k+1})$ is based directly on $(\mathbf{x}^k, \mathbf{u}^k)$ and $(\mathbf{z}^k, \mathbf{v}^k)$.

To solve problem (1.38), for example, it is possible to resort to a gradient method which reads as, for all $k \geq 0$:

$$\begin{aligned}\boldsymbol{\alpha}^{k+1} &= \boldsymbol{\alpha}^k - \beta^k \nabla_{\boldsymbol{\alpha}} J(\boldsymbol{\alpha}^k, \boldsymbol{\mu}^k)\\ \boldsymbol{\mu}^{k+1} &= \boldsymbol{\mu}^k - \beta^k \nabla_{\boldsymbol{\mu}} J(\boldsymbol{\alpha}^k, \boldsymbol{\mu}^k)\end{aligned} \tag{1.40}$$

Theorem 1. *There exists a step-size β^k such that any limit point $(\boldsymbol{\alpha}^*, \boldsymbol{\mu}^*)$ of the sequence $\{\boldsymbol{\alpha}^k, \boldsymbol{\mu}^k\}_{k\geq 0}$ generated by* (1.40) *is a stationary point of $F(\phi(\boldsymbol{\alpha}, \boldsymbol{\mu}), \gamma(\boldsymbol{\alpha}, \boldsymbol{\mu}))$*

In this case, the gradient of the cost function J involves the gradient of both mapping function ϕ and γ which may be difficult to compute. This difficulty may be overcome exploiting the characterization of ϕ and γ given by Assumption 2 and introducing an equivalent problem computed via the Lagrangian functional. This approach is going to be detailed later in Chapter 2.

Eventually, this method is summarized in Algorithm 4, where, at each iteration k a trajectory $(\mathbf{x}^{k+1}, \mathbf{u}^{k+1})$ is available. The descent direction $\mathbf{v}^k$ is computed via (1.41) where $a_t^k \in \mathbb{R}^{n_x}$, $b_t^k \in \mathbb{R}^{n_u}$, $A_t^k \in \mathbb{R}^{n_x \times n_x}$, $B_t^k \in \mathbb{R}^{n_x \times n_m}$ are defined as:

$$\begin{aligned}a_t^k &= \nabla_{x_t} \ell(x_t^k, u_t^k)\\ b_t^k &= \nabla_{u_t} \ell(x_t^k, u_t^k)\\ A_t^k &= \nabla_{x_t} f(x_t^k, u_t^k)^\top\\ B_t^k &= \nabla_{u_t} f(x_t^k, u_t^k)^\top\end{aligned}$$

The trajectory update is then performed in (1.42) via the closed loop dynamics (1.10), where $K_t^k \in \mathbb{R}^{n_u \times n_x}$ solution of the Linear Quadratic Regulator problem of a linearization of the system about $(\mathbf{x}^k, \mathbf{u}^k)$.

Differently from [4], the descent direction of this algorithm is a generic state-input curve that is not required to satisfy any linearized dynamics.

As a comparison with Algorithm 2, note that, by setting $K_t^k \equiv 0 \; \forall t$, equations (1.41) and (1.42) become equations (1.29) and (1.30) respectively.

Finally, note that dynamic feasibility is guaranteed at each iteration.

Algorithm 4 Closed-Loop Sequential Method

```
Require: trajectory $(\mathbf{x^0}, \mathbf{u^0})$ with $x_0^0 = x_{init}$
  for $k = 0, 1, 2, \dots$ do
    compute $K_t^k$ for all $t \in \mathbb{T}_{[0,T-1]}$
    set $\lambda_T^k = \nabla m(x_T^k)$
    for $t = T-1, \dots, 0$ do
      compute
        $$\begin{aligned} v_t^k &= -B_t^{k\top} \lambda_{t+1}^k - b_t^k \\ z_t^k &= K_t^{k\top} v_t^k \\ \lambda_t^k &= (A_t^k - B_t^k K_t^k)^\top \lambda_{t+1}^k + a_t^k - K_t^{k\top} b_t^k \end{aligned} \tag{1.41}$$
    end for
    compute step-size $\beta^k$
    set initial condition $x_0^{k+1} = x_{init}$
    for $t = 0, \dots, T-1$ do
      compute new curve
        $$\begin{aligned} \alpha_t^{k+1} &= \alpha_t^k + \beta z_t^k \\ \mu_t^{k+1} &= \mu_t^k + \beta v_t^k \end{aligned}$$
      compute new trajectory
        $$\begin{aligned} u_t^{k+1} &= \mu_t^k + K_t^k(\alpha_t^{k+1} - x_t^{k+1}) \\ x_{t+1}^{k+1} &= f(x_t^{k+1}, u_t^{k+1}) \end{aligned} \tag{1.42}$$
    end for
  end for
```

Chapter 2

Heavy-Ball Closed-Loop Sequential Method

A contribution of this thesis relies in developing the methodology firstly proposed for the distributed setup in [5] towards the resolution of centralized optimal control problems. This work proposes the Heavy-Ball method, in order to speed up algorithm convergence. Nevertheless, thanks to the flexibility provided by the proposed methodology, different descent methods could be easily considered and implemented in the sequential optimization algorithm with very few effort from the designer.

Furthermore, the presence of inequality constraints over the system dynamics is eventually considered. In this case, the previously developed method is extended with a barrier function, which allows to reformulate constrained optimization problems as unconstrained ones suitable for this methodology.

2.1 Heavy-Ball Sequential Optimization

The Heavy-Ball Method is a two-step procedure firstly introduced by Polyak in [9]. It is usually considered as an improvement with respect to the gradient descent when dealing with optimization problems [12].
The intuition is simple: the iterates of gradient descent tend to bounce between the walls of narrow "valleys" on the objective surface. The left panel of figure 2.1 shows the iterates of gradient descent bouncing from wall to wall. Considering the optimization problem:

$$\min_{x} \quad f(x)$$

where $x \in \mathbb{R}^{n_x}$, $f : \mathbb{R}^{n_x} \to \mathbb{R}$, and f continuous and twice differentiable, namely $f \in C^2$.

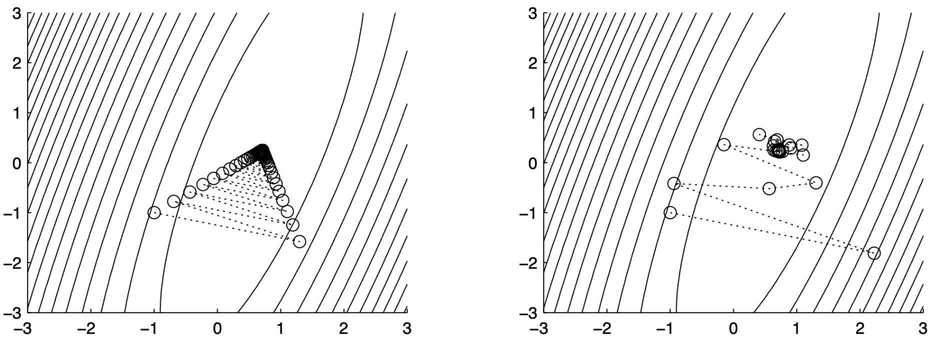


Figure 2.1: The iterates of gradient descent (left panel) and the heavy ball method (right panel) starting at $(-1, -1)$

For this problem the gradient descent update reads, for all iterations $k \geq 0$, as:

$$x^{k+1} = x^k - \beta^k \nabla f(x^k)$$

where $\beta^k \in \mathbb{R}$ is the step-size. To avoid bouncing, the heavy ball method adds a momentum term with step-size $\gamma^k \in \mathbb{R}$ to the gradient step for all $k > 0$:

$$x^{k+1} = x^k - \beta^k \nabla f(x^k) + \gamma^k (x^k - x^{k-1})$$

The term $x^k - x^{k-1}$ nudges x^{k+1} to the direction of the previous step (hence momentum), the right panel of figure 2.1 shows the effects of adding momentum.

This methods owes its name due to the fact that each iteration is equivalent to a discretization of a second order ODE which models the motion of a body in a potential field given by f with friction.

The Closed-Loop Projected Methodology proposed in Algorithm 3 in Section 1.3.2 allows a straightforward implementation of this approach. In fact, as already shown in Section 1.3.2, thanks to the introduction of a redundant constraint which forces the control input to apply both a feedforward and a feedback action, the following optimal control problem:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{u},\boldsymbol{\alpha},\boldsymbol{\mu}} \quad & \sum_{t=0}^{T-1} \ell(x_t, u_t) + m(x_T) \\ \text{subj. to} \quad & x_{t+1} = f(x_t, u_t) \qquad t \in \mathbb{T}_{[0,T-1]} \\ & u_t = \mu_t + K_t(\alpha_t - x_t) \\ & x_0 = \alpha_0 = x_{init} \end{aligned} \tag{2.1}$$

can be recast as:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{u},\boldsymbol{\alpha},\boldsymbol{\mu}} \quad & F(\mathbf{x}, \mathbf{u}) \\ \text{subj. to} \quad & h(\mathbf{x}, \mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = 0 \end{aligned} \tag{2.2}$$

where $\mathbf{x}, \boldsymbol{\alpha} \in \mathbb{R}^{n_x T}$ and $\mathbf{u}, \boldsymbol{\mu} \in \mathbb{R}^{n_u T}$ and F as:

$$F(\mathbf{x}, \mathbf{u}) = \sum_{t=0}^{T-1} \ell(x_t, u_t) + m(x_T) \tag{2.3}$$

and $h : \mathbb{R}^{n_x T} \times \mathbb{R}^{n_u T} \times \mathbb{R}^{n_x T} \times \mathbb{R}^{n_u T} \to \mathbb{R}^{n_x T + n_u T}$ defined as:

$$h(\mathbf{x}, \mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = \begin{bmatrix} f(x_0, u_0) - x_1 \\ \vdots \\ f(x_{T-1}, u_{T-1}) - x_T \\ \mu_0 + K_0(\alpha_0 - x_0) - u_0 \\ \vdots \\ \mu_{T-1} + K_0(\alpha_{T-1} - x_{T-1}) - u_{T-1} \end{bmatrix} \tag{2.4}$$

Then, in light of Assumption 2, problem (2.1) can be written in the reduced form as:

$$\begin{aligned} \min_{u_0, \dots, u_{T-1}} & \sum_{t=0}^{T-1} \ell(\phi_t(\boldsymbol{\alpha}, \boldsymbol{\mu}), \gamma_t(\boldsymbol{\alpha}, \boldsymbol{\mu})) + m(\phi_T(\boldsymbol{\alpha}, \boldsymbol{\mu})) \\ \text{subj. to} \quad & \boldsymbol{\alpha} \in \mathbb{R}^{n_x T}, \boldsymbol{\mu} \in \mathbb{R}^{n_u T} \end{aligned} \tag{2.5}$$

this unconstrained optimization problem can be solved via a descent method such as Heavy-Ball. In this case, each curve update iteration for $k \geq 0$ reads as:

$$\begin{bmatrix} \boldsymbol{\alpha}^{k+1} \\ \boldsymbol{\mu}^{k+1} \end{bmatrix} = \begin{bmatrix} \boldsymbol{\alpha}^{k} \\ \boldsymbol{\mu}^{k} \end{bmatrix} + \beta^k \begin{bmatrix} \mathbf{z}^{k} \\ \mathbf{v}^{k} \end{bmatrix} + \gamma^k \begin{bmatrix} \boldsymbol{\alpha}^{k} - \boldsymbol{\alpha}^{k-1} \\ \boldsymbol{\mu}^{k} - \boldsymbol{\mu}^{k-1} \end{bmatrix}$$

Where β^k and γ^k are conveniently chosen step-sizes and descent direction $(\mathbf{z}^k, \mathbf{v}^k)$ with $\mathbf{z}^k \in \mathbb{R}^{n_x T}$, $\mathbf{v}^k \in \mathbb{R}^{n_u T}$. This new curve is then conveniently projected onto the trajectory manifold.

In this thesis, however, it is implemented a projected version of the curve update 2.1, which has proven to be faster and more stable with respect to the "unprojected" version within the scope of the considered tests. This implementation reads as:

$$\begin{bmatrix} \boldsymbol{\alpha}^{k+1} \\ \boldsymbol{\mu}^{k+1} \end{bmatrix} = \begin{bmatrix} \mathbf{x}^{k} \\ \mathbf{u}^{k} \end{bmatrix} + \beta^k \begin{bmatrix} \mathbf{z}^{k} \\ \mathbf{v}^{k} \end{bmatrix} + \gamma^k \begin{bmatrix} \mathbf{x}^{k} - \mathbf{x}^{k-1} \\ \mathbf{u}^{k} - \mathbf{u}^{k-1} \end{bmatrix}$$

In this way we have:

$$\begin{aligned} u_t^{k+1} &= \mu_t^k + K_t^k \left[x_t^k - x_t^{k+1} + \beta^k z_t^k + \gamma^k (x_t^k - x_t^{k-1}) \right] \\ x_{t+1}^{k+1} &= f(x_t^{k+1}, u_t^{k+1}) \end{aligned} \tag{2.6}$$

The application of this method is illustrated in Algorithm 5.

Algorithm 5 Application of the Projected Heavy-Ball Method

```
Require: trajectory $(\mathbf{x}^0, \mathbf{u}^0)$ with $x_0^0 = x_{init}$
  for $k = 0, 1, 2, \dots$ do
    Compute $K_t,\ t = 0, \dots, T-1$
    Compute a descent direction $(\mathbf{z}^k, \mathbf{v}^k)$ based on $(\mathbf{x}^k, \mathbf{v}^k)$
    Compute $\beta^k, \gamma^k$
    Compute a new curve
      $$\begin{bmatrix}\boldsymbol{\alpha}^{k+1}\\ \boldsymbol{\mu}^{k+1}\end{bmatrix} = \begin{bmatrix}\mathbf{x}^{k}\\ \mathbf{u}^{k}\end{bmatrix} + \beta^k \begin{bmatrix}\mathbf{z}^{k}\\ \mathbf{v}^{k}\end{bmatrix} + \gamma^k \begin{bmatrix}\mathbf{x}^{k} - \mathbf{x}^{k-1}\\ \mathbf{u}^{k} - \mathbf{u}^{k-1}\end{bmatrix}$$
    Update the new trajectory
      $$(\mathbf{x}^{k+1}, \mathbf{u}^{k+1}) = \mathcal{P}(\boldsymbol{\alpha}^{k+1}, \boldsymbol{\mu}^{k+1})$$
  end for
```

In this work this method is applied together with the closed-loop sequential optimization method discussed previously. In particular, recalling the function composition:

$$\begin{aligned}
J(\boldsymbol{\alpha}, \boldsymbol{\mu}) &= F(\mathcal{P}(\boldsymbol{\alpha}, \boldsymbol{\mu}))\\
&= F(\phi(\boldsymbol{\alpha}, \boldsymbol{\mu}), \gamma(\boldsymbol{\alpha}, \boldsymbol{\mu}))\\
&= \sum_{t=0}^{T-1} \ell(\phi_t(\boldsymbol{\alpha}, \boldsymbol{\mu}), \gamma_t(\boldsymbol{\alpha}, \boldsymbol{\mu})) + m(\phi_T(\boldsymbol{\alpha}, \boldsymbol{\mu}))
\end{aligned}$$

where ϕ and γ are smooth maps satisfying Assumption 2 and ϕ_t, γ_t being the t-th component of ϕ and γ respectively.

This function composition allows us to recast problem (2.5) as:

$$\begin{aligned}
\min_{\boldsymbol{\alpha}, \boldsymbol{\mu}} \quad & J(\boldsymbol{\alpha}, \boldsymbol{\mu})\\
\text{subj. to} \quad & \boldsymbol{\alpha} \in \mathbb{R}^{n_x T}, \boldsymbol{\mu} \in \mathbb{R}^{n_u T}
\end{aligned} \tag{2.7}$$

The Heavy-Ball method applied to (2.7) reads, for all $k \geq 0$, following the steepest descent direction:

$$\begin{aligned}
\boldsymbol{\alpha}^{k+1} &= \mathbf{x}^k - \beta^k \nabla_{\boldsymbol{\alpha}} J(\boldsymbol{\alpha}^k, \boldsymbol{\mu}^k) + \gamma^k (\mathbf{x}^k - \mathbf{x}^{k-1})\\
\boldsymbol{\mu}^{k+1} &= \mathbf{u}^k - \beta^k \nabla_{\boldsymbol{\mu}} J(\boldsymbol{\alpha}^k, \boldsymbol{\mu}^k) + \gamma^k (\mathbf{u}^k - \mathbf{u}^{k-1})
\end{aligned} \tag{2.8}$$

Theorem 2. *There exists a step-size β^k, γ^k such that any limit point $(\boldsymbol{\alpha}^*, \boldsymbol{\mu}^*)$ of the sequence $\{\boldsymbol{\alpha}^k, \boldsymbol{\mu}^k\}_{k \geq 0}$ generated by (2.8) is a stationary point of $F(\phi(\boldsymbol{\alpha}, \boldsymbol{\mu}), \gamma(\boldsymbol{\alpha}, \boldsymbol{\mu}))$*

In this case, the gradient of the cost function J involves the gradient of both mapping function ϕ and γ which may be difficult to compute. This difficulty may be overcome exploiting the characterization of ϕ and γ given by Assumption 2 and introducing an equivalent problem computed via the Lagrangian functional.

Let us introduce the Lagrangian multiplier vector:

$$\boldsymbol{\lambda} := \left[\lambda_1^\top, \dots, \lambda_T^\top, \tilde{\lambda}_1^\top, \dots, \tilde{\lambda}_T^\top\right]^\top \in \mathbb{R}^{n_x T + n_u T} \tag{2.9}$$

with each $\lambda_t \in \mathbb{R}^{n_x}$ and $\tilde{\lambda}_t \in \mathbb{R}^{n_u}$, associated to the dynamic constraint $h(\mathbf{x}, \mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\mu}) = 0$ of problem 2.2. The Lagrangian of that problem then reads:

$$\mathcal{L}(\mathbf{x}, \mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{\lambda}) := F(\mathbf{x}, \mathbf{u}) + h(\mathbf{x}, \mathbf{u}, \boldsymbol{\alpha}, \boldsymbol{\mu})^\top \boldsymbol{\lambda} \tag{2.10}$$

A nice property of the Lagrangian is that it holds :

$$\begin{aligned} \mathcal{L}(\phi(\boldsymbol{\alpha}, \boldsymbol{\mu}), \gamma(\boldsymbol{\alpha}, \boldsymbol{\mu}), \boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{\lambda}) &= F(\phi(\boldsymbol{\alpha}, \boldsymbol{\mu}), \gamma(\boldsymbol{\alpha}, \boldsymbol{\mu})) \\ &= J(\boldsymbol{\alpha}, \boldsymbol{\mu}) \end{aligned} \tag{2.11}$$

for all $\boldsymbol{\alpha} \in \mathbb{R}^{n_x T}$, $\mu \in \mathbb{R}^{n_u T}$ and for all $\boldsymbol{\lambda} \in \mathbb{R}^{n_x T + n_u T}$, i.e., the value of $\mathcal{L}(\phi(\boldsymbol{\alpha}, \boldsymbol{\mu}), \gamma(\boldsymbol{\alpha}, \boldsymbol{\mu}), \boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{\lambda})$ does not depend on $\boldsymbol{\lambda}$, due to the fact that, by Assumption 2:

$$h(\phi(\boldsymbol{\alpha}, \boldsymbol{\mu}), \gamma(\boldsymbol{\alpha}, \boldsymbol{\mu}), \boldsymbol{\alpha}, \boldsymbol{\mu}) = 0$$

for all $(\boldsymbol{\alpha}, \boldsymbol{\mu}) \in \mathbb{R}^{n_x T} \times \mathbb{R}^{n_u T}$.

Thus, problem (2.7) is equivalent to

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \boldsymbol{\mu}} \quad & \mathcal{L}(\phi(\boldsymbol{\alpha}, \boldsymbol{\mu}), \gamma(\boldsymbol{\alpha}, \boldsymbol{\mu}), \boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{\lambda}) \\ \text{subj. to} \quad & \boldsymbol{\alpha} \in \mathbb{R}^{n_x T}, \boldsymbol{\mu} \in \mathbb{R}^{n_u T} \end{aligned} \tag{2.12}$$

for any value of $\boldsymbol{\lambda}$. Then, $\boldsymbol{\lambda}$ can be considered as a degree of freedom rather than an optimization variable.

Let $\boldsymbol{\lambda} \in \mathbb{R}^{n_x T + n_u T}$ be given, then introduce $\mathcal{L}_\lambda : \mathbb{R}^{n_x} \times \mathbb{R}^{n_u} \to \mathbb{R}$ defined as:

$$\mathcal{L}_\lambda := \mathcal{L}(\phi(\boldsymbol{\alpha}, \boldsymbol{\mu}), \gamma(\boldsymbol{\alpha}, \boldsymbol{\mu}), \boldsymbol{\alpha}, \boldsymbol{\mu}, \boldsymbol{\lambda}) \tag{2.13}$$

The gradient of $\mathcal{L}_\lambda(\boldsymbol{\alpha}, \boldsymbol{\mu})$ is:

$$\nabla \mathcal{L}_\lambda = \begin{bmatrix} \nabla_{\boldsymbol{\alpha}} \mathcal{L}_\lambda(\boldsymbol{\alpha}, \boldsymbol{\mu}) \\ \nabla_{\boldsymbol{\mu}} \mathcal{L}_\lambda(\boldsymbol{\alpha}, \boldsymbol{\mu}) \end{bmatrix} \tag{2.14}$$

in which:

$$\begin{aligned}
\nabla_{\boldsymbol{\alpha}}\mathcal{L}_\lambda(\boldsymbol{\alpha},\boldsymbol{\mu}) &= \nabla_{\boldsymbol{\alpha}}\phi\nabla_{\mathbf{x}}\mathcal{L} + \nabla_{\boldsymbol{\alpha}}\gamma\nabla_{\mathbf{u}}\mathcal{L} + \nabla_{\boldsymbol{\alpha}}\mathcal{L} \\
&= \nabla_{\boldsymbol{\alpha}}\phi(\boldsymbol{\alpha},\boldsymbol{\mu})\Big(\nabla_{\mathbf{x}}F(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu})) \\
&\quad + \nabla_{\mathbf{x}}h(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu}),\boldsymbol{\alpha},\boldsymbol{\mu})^\top\boldsymbol{\lambda}\Big) \\
&\quad + \nabla_{\boldsymbol{\alpha}}\gamma(\boldsymbol{\alpha},\boldsymbol{\mu})\Big(\nabla_{\mathbf{u}}F(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu})) \\
&\quad + \nabla_{\mathbf{u}}h(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu}),\boldsymbol{\alpha},\boldsymbol{\mu})^\top\boldsymbol{\lambda}\Big) \\
&\quad + \nabla_{\boldsymbol{\alpha}}h(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu}),\boldsymbol{\alpha},\boldsymbol{\mu})^\top\boldsymbol{\lambda}
\end{aligned} \tag{2.15}$$

and:

$$\begin{aligned}
\nabla_{\boldsymbol{\mu}}\mathcal{L}_\lambda(\boldsymbol{\alpha},\boldsymbol{\mu}) &= \nabla_{\boldsymbol{\mu}}\phi\nabla_{\mathbf{x}}\mathcal{L} + \nabla_{\boldsymbol{\mu}}\gamma\nabla_{\mathbf{u}}\mathcal{L} + \nabla_{\boldsymbol{\mu}}\mathcal{L} \\
&= \nabla_{\boldsymbol{\mu}}\phi(\boldsymbol{\alpha},\boldsymbol{\mu})\Big(\nabla_{\mathbf{x}}F(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu})) \\
&\quad + \nabla_{\mathbf{x}}h(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu}),\boldsymbol{\alpha},\boldsymbol{\mu})^\top\boldsymbol{\lambda}\Big) \\
&\quad + \nabla_{\boldsymbol{\mu}}\gamma(\boldsymbol{\alpha},\boldsymbol{\mu})\Big(\nabla_{\mathbf{u}}F(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu})) \\
&\quad + \nabla_{\mathbf{u}}h(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu}),\boldsymbol{\alpha},\boldsymbol{\mu})^\top\boldsymbol{\lambda}\Big) \\
&\quad + \nabla_{\boldsymbol{\mu}}h(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu}),\boldsymbol{\alpha},\boldsymbol{\mu})^\top\boldsymbol{\lambda}
\end{aligned} \tag{2.16}$$

Notice that $\nabla\mathcal{L}_\lambda$ in (2.14) has a very peculiar structure. In fact, its expression can be simplified by choosing a proper value for $\boldsymbol{\lambda}$. Specifically, let $\boldsymbol{\lambda}(\boldsymbol{\alpha},\boldsymbol{\mu}) \in \mathbb{R}^{n_xT+n_uT}$ annihilate the terms pre-multiplied by $\nabla\phi$ and $\nabla\gamma$, i.e. let $\boldsymbol{\lambda}(\boldsymbol{\alpha},\boldsymbol{\mu})$ be such that:

$$\begin{aligned}
&\nabla_{\mathbf{x}}F(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu}))+ \\
&\qquad +\nabla_{\mathbf{x}}h(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu}),\boldsymbol{\alpha},\boldsymbol{\mu})^\top\boldsymbol{\lambda}(\boldsymbol{\alpha},\boldsymbol{\mu}) = 0 \\
&\nabla_{\mathbf{u}}F(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu}))+ \\
&\qquad +\nabla_{\mathbf{u}}h(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu}),\boldsymbol{\alpha},\boldsymbol{\mu})^\top\boldsymbol{\lambda}(\boldsymbol{\alpha},\boldsymbol{\mu}) = 0
\end{aligned} \tag{2.17}$$

Then, whenever $\boldsymbol{\lambda} = \boldsymbol{\lambda}(\boldsymbol{\alpha},\boldsymbol{\mu})$ we obtain:

$$\begin{aligned}
\nabla_{\boldsymbol{\alpha}}\mathcal{L}_\lambda(\boldsymbol{\alpha},\boldsymbol{\mu}) &= \nabla_{\boldsymbol{\alpha}}h(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu}),\boldsymbol{\alpha},\boldsymbol{\mu})^\top\boldsymbol{\lambda} \\
\nabla_{\boldsymbol{\mu}}\mathcal{L}_\lambda(\boldsymbol{\alpha},\boldsymbol{\mu}) &= \nabla_{\boldsymbol{\mu}}h(\phi(\boldsymbol{\alpha},\boldsymbol{\mu}),\gamma(\boldsymbol{\alpha},\boldsymbol{\mu}),\boldsymbol{\alpha},\boldsymbol{\mu})^\top\boldsymbol{\lambda}
\end{aligned} \tag{2.18}$$

With (2.17) and (2.18), it is possible to pose a gradient method to minimize problem (2.12).

Formally, the Heavy-Ball projected descent algorithm, equation (2.1), applied to (2.12) reads, for all $k \geq 0$:

$$\begin{aligned}
\boldsymbol{\alpha}^{k+1} &= \mathbf{x}^k - \beta^k\nabla_{\boldsymbol{\alpha}}\mathcal{L}(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k;\boldsymbol{\lambda}^k) + \gamma^k(\mathbf{x}^k - \mathbf{x}^{k-1}) \\
\boldsymbol{\mu}^{k+1} &= \mathbf{u}^k - \beta^k\nabla_{\boldsymbol{\mu}}\mathcal{L}(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k;\boldsymbol{\lambda}^k) + \gamma^k(\mathbf{u}^k - \mathbf{u}^{k-1})
\end{aligned} \tag{2.19}$$

Choosing $\boldsymbol{\lambda}^k$ such that:

$$\begin{aligned}&\nabla_{\mathbf{x}} F(\phi(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k),\gamma(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k))+\\&\qquad+\nabla_{\mathbf{x}} h(\phi(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k),\gamma(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k),\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k)^\top\boldsymbol{\lambda}^k=0\\&\nabla_{\mathbf{u}} F(\phi(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k),\gamma(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k))+\\&\qquad+\nabla_{\mathbf{u}} h(\phi(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k),\gamma(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k),\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k)^\top\boldsymbol{\lambda}^k=0\end{aligned}\tag{2.20}$$

it is possible to prove that (2.19) becomes:

$$\begin{aligned}\boldsymbol{\alpha}^{k+1}&=\mathbf{x}^k-\beta^k\nabla_{\boldsymbol{\alpha}} h(\phi(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k),\gamma(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k),\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k)^\top\boldsymbol{\lambda}^k+\\&\qquad+\gamma^k(\mathbf{x}^k-\mathbf{x}^{k-1})\\\boldsymbol{\mu}^{k+1}&=\mathbf{u}^k-\beta^k\nabla_{\boldsymbol{\mu}} h(\phi(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k),\gamma(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k),\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k)^\top\boldsymbol{\lambda}^k+\\&\qquad+\gamma^k(\mathbf{u}^k-\mathbf{u}^{k-1})\end{aligned}\tag{2.21}$$

For given $(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k)$, let:

$$\begin{aligned}a_t^k&=\nabla_{x_t}\ell(\phi_t(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k),\gamma_t(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k))\\b_t^k&=\nabla_{u_t}\ell(\phi_t(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k),\gamma_t(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k))\\A_t^k&=\nabla_{x_t}f(\phi_t(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k),\gamma_t(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k))^\top\\B_t^k&=\nabla_{u_t}f(\phi_t(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k),\gamma_t(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k))^\top\end{aligned}\tag{2.22}$$

Then, a vector $\boldsymbol{\lambda}^k\in\mathbb{R}^{n_xT+n_uT}$ satisfying (2.20) can be obtained by imposing:

$$\begin{aligned}a_t^k+A_t^{k\top}\lambda_{t+1}^k-\lambda_t^k-K_t^{k\top}\tilde{\lambda}_t^k&=0\qquad t\in\mathbb{T}_{[1,T-1]}\\\nabla m(\phi_T(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k)-\lambda_T^k&=0\\b_t^k+B_t^{k\top}\lambda_{t+1}^k-\tilde{\lambda}_t^k&=0\qquad t\in\mathbb{T}_{[0,T]}\end{aligned}$$

which can be rearranged as:

$$\lambda_t^k=\left(A_t^k-B_t^kK_t^k\right)^\top\lambda_{t+1}^k+a_t^k-K_t^{k\top}b_t^k\tag{2.23a}$$

$$\lambda_T^k=\nabla m(\phi_T(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k))\tag{2.23b}$$

$$\tilde{\lambda}_t^k=b_t^k+B_t^{k\top}\lambda_{t+1}^k\tag{2.23c}$$

for all $t\in\mathbb{T}_{[1,T-1]}$. This is a linear dynamical system that is meant to be backward simulated in t.

Once $\boldsymbol{\lambda}^k$ is computed, the gradient of the Lagrangian function, namely the descent direction, can be computed. In particular, $\nabla_{\boldsymbol{\alpha}}\mathcal{L}$ can be obtained using:

$$\begin{aligned}z_t^k:=-\nabla_{\alpha_t}\mathcal{L}(\boldsymbol{\alpha}^k,\boldsymbol{\mu}^k)&=-K_t^{k\top}\tilde{\lambda}_t^k\\&=-K_t^{k\top}\left(b_t^k+B_t^{k\top}\lambda_{t+1}^k\right)\end{aligned}\tag{2.24}$$

for all $t \in \mathbb{T}_{[1,T]}$, $z_t^k \in \mathbb{R}^{n_x}$. While, the gradient $\nabla_{\boldsymbol{\mu}} \mathcal{L}$ can be obtained using:

$$\begin{aligned} v_t^k := -\nabla_{\mu_t} \mathcal{L}(\boldsymbol{\alpha}^k, \boldsymbol{\mu}^k) &= -\tilde{\lambda}_t^k \\ &= -b_t^k - B_t^{k\top} \lambda_{t+1}^k \end{aligned} \tag{2.25}$$

for all $t \in \mathbb{T}_{[0,T-1]}$.

Thus, the Heavy-Ball method applied to a closed-loop sequential optimization strategy is recapped by Algorithm 6.

This approach, with respect to the strategy proposed in [4], offers as main advantage the possibility of keeping the descent direction free and not constrained to the space tangent to the current trajectory. In fact, differently from [3] and [4], the descent direction of this algorithm is a generic state-input curve that is not required to satisfy any linearized dynamics. Moreover, note that dynamic feasibility is guaranteed at each iteration. Thus, the algorithm can be stopped at any iteration with a twofold guarantee: it is available a system trajectory $(\mathbf{x}^k, \mathbf{u}^k)$ and a controller K_t^k $t \in \mathbb{T}_{[0,T-1]}$ to track it.

Algorithm 6 Closed-Loop Sequential Method enhanced by Heavy-Ball

```
Require: trajectory $(\mathbf{x^0}, \mathbf{u^0})$ with $x_0^0 = x_{init}$
  for $k = 0, 1, 2, \dots$ do
    compute $K_t^k$ for all $t \in \mathbb{T}_{[0,T-1]}$
    set $\lambda_T^k = \nabla m(x_T^k)$
    for $t = T-1, \dots, 0$ do
      compute
        $v_t^k = -B_t^{k\top} \lambda_{t+1}^k - b_t^k$
        $z_t^k = K_t^{k\top} v_t^k$
        $\lambda_t^k = (A_t^k - B_t^k K_t^k)^\top \lambda_{t+1}^k + a_t^k - K_t^{k\top} b_t^k$
    end for
    compute step-size $\beta^k, \gamma^k$
    set initial condition $x_0^{k+1} = x_{init}$
    for $t = 0, \dots, T-1$ do
      compute new curve
        $\alpha_t^{k+1} = x_t^k + \beta^k z_t^k + \gamma^k (x_t^k - x_t^{k-1})$
        $\mu_t^{k+1} = u_t^k + \beta^k v_t^k + \gamma^k (u_t^k - u_t^{k-1})$
      compute new trajectory
        $u_t^{k+1} = \mu_t^k + K_t^k (\alpha_t^{k+1} - x_t^{k+1})$
        $x_{t+1}^{k+1} = f(x_t^{k+1}, u_t^{k+1})$
    end for
  end for
```

2.2 Heavy-Ball for Constrained Systems

In the vast majority of optimal control some sort of constraints needs to be considered, e.g. maximum power provided by the actuators, path constraints, etc. In this case, the optimal control problem is defined as:

$$\begin{aligned} \min_{\substack{x_1,\dots,x_N \\ u_0,\dots,u_{T-1}}} \; & \sum_{t=0}^{T-1} \ell(x_t, u_t) + m(x_T) \\ \text{subj. to} \; & x_{t+1} = f(x_t, u_t) \qquad t \in \mathbb{T}_{[0,T-1]} \\ & g_t(x_t, u_t) \le 0 \\ & x_0 = x_{\text{init}} \end{aligned} \tag{2.26}$$

To be consistent with the proposed methodology, we are interested in developing unconstrained optimal control strategies that can be used to approximate solutions to the constrained optimal control problem. In particular, a barrier function approach is considered. This strategy, in fact, proved to be quite effective for solving convex optimization problems [13].

In that approach, a solution to a convex problem:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & g_j(x) \leq 0 \qquad j = 1, \dots, n_g \end{aligned}$$

is found by solving a sequence of convex problems:

$$\min_{x \in X} \quad f(x) - \varepsilon \sum_{j=1}^{n_g} \log(-g_j(x))$$

Where $X = \{x \in \mathbb{R}^n : \ g_j(x) < 0\}$. Using a decreasing sequence, it is possible to follow the path of solutions $\varepsilon \to x^*_\varepsilon$ to a solution $x^* = \lim_{\varepsilon \to 0} x^*_\varepsilon$.

Introducing n_g constraints in problem (2.26), it becomes:

$$\begin{aligned} \min_{\substack{x_1, \dots, x_N \\ u_0, \dots, u_{T-1}}} \quad & \sum_{t=0}^{T-1} \ell(x_t, u_t) + m(x_T) \\ \text{subj. to} \quad & x_{t+1} = f(x_t, u_t) \qquad t \in \mathbb{T}_{[0, T-1]} \\ & g_j(t, x_t, u_t) \leq 0 \qquad j = 1, \dots, n_g \\ & x_0 = x_{\text{init}} \end{aligned}$$

Following the strategy proposed by [8], an analog of the barrier function method for use on discrete time constrained trajectory optimization problems is of the form:

$$\begin{aligned} \min_{\substack{x_1, \dots, x_N \\ u_0, \dots, u_{T-1}}} \quad & \sum_{t=0}^{T-1} \left[\ell(x_t, u_t) - \varepsilon \sum_{j=1}^{n_g} \log(-g_j(t, x_t, u_t)) \right] + m(x_T) \\ \text{subj. to} \quad & x_{t+1} = f(x_t, u_t) \qquad t \in \mathbb{T}_{[0, T-1]} \\ & x_0 = x_{\text{init}} \end{aligned} \tag{2.27}$$

with $(\mathbf{x}, \mathbf{u})$ such that $g_j(\mathbf{x}, \mathbf{u}) < 0$, for all $j = 1, \dots, n_g$.

This approach, however, presents a key difficulty: infeasibility is not tolerated at all. That is, it is not possible to evaluate the objective in (2.27) unless $(\mathbf{x}, \mathbf{u})$ is a feasible trajectory. In fact, for an unfeasible trajectory, namely $g_j > 0$ the approximate barrier function is undefined. To solve this issue, as proposed in [8], an approximate logarithmic barrier function is introduced $\beta_\delta(\cdot)$, $0 \leq \delta \leq 1$:

$$\beta_\delta(z) = \begin{cases} -\log(z) & z > \delta \\ \frac{k-1}{k} \left[\left(\frac{z - k\delta}{(k-1)\delta} \right)^k - 1 \right] - \log(\delta) & z \leq \delta \end{cases} \tag{2.28}$$

The C^2 function (2.28) retains many of the important properties of the log barrier function: $z \to -\log(z)$ while expanding the domain of finite values from $(0, \infty)$ to $(-\infty, \infty)$. Both functions are (strictly) convex and strictly decreasing on their domains.

Returning to the original problem, using the constraints in (2.26) to define the approximate barrier functional:

$$b_\delta = \sum_{t=0}^{T} \sum_{j=1}^{n_g} \beta_\delta(-g_j(t, x_t, u_t))$$

then, problem (2.26) can be rewritten as:

$$\begin{aligned} \min_{\substack{x_1,\dots,x_N \\ u_0,\dots,u_{T-1}}} \quad & \sum_{t=0}^{T-1} \left[\ell(x_t, u_t) - \varepsilon \sum_{j=1}^{n_g} \beta_\delta(-g_j(t, x_t, u_t)) \right] + m(x_T) \\ \text{subj. to} \quad & x_{t+1} = f(x_t, u_t) \qquad t \in \mathbb{T}_{[0,T-1]} \\ & x_0 = x_{\text{init}} \end{aligned} \tag{2.29}$$

in which the only constraint is represented by the system dynamics.

For a given curve $(\boldsymbol{\alpha}, \boldsymbol{\mu})$, then, it is possible to introduce the unconstrained optimal control problem

$$\min_{(\boldsymbol{\alpha}, \boldsymbol{\mu}) \in \mathcal{T}} F(\boldsymbol{\alpha}, \boldsymbol{\mu}) + \varepsilon b_\delta(\boldsymbol{\alpha}, \boldsymbol{\mu}) \tag{2.30}$$

This problem is an approximation of (2.26) as much as (2.27) is, the only difference is that this functional can be evaluated on any curve, while (2.27) may only be evaluated on feasible curves. As in the finite dimensional case, a trajectory that is locally optimal solution of (2.27) is also solution of (2.30) provided $\delta > 0$ sufficiently small.

This functional permits to exploit the projection operator. In particular, the constrained optimal control problem exploiting the projection operator can be written as:

$$\begin{aligned} \min_{\boldsymbol{\alpha}, \boldsymbol{\mu}} \quad & J_{\varepsilon,\delta}(\boldsymbol{\alpha}, \boldsymbol{\mu}) \\ \text{subj. to} \quad & \boldsymbol{\alpha} \in \mathbb{R}^{n_x T}, \boldsymbol{\mu} \in \mathbb{R}^{n_u T} \end{aligned} \tag{2.31}$$

where:

$$J_{\varepsilon,\delta}(\boldsymbol{\alpha}, \boldsymbol{\mu}) = F(\mathcal{P}(\boldsymbol{\alpha}, \boldsymbol{\mu})) + \varepsilon b_\delta(\mathcal{P}(\boldsymbol{\alpha}, \boldsymbol{\mu}))$$

The adopted strategy is to start with a reasonably large ε and δ, e.g. $\varepsilon = \delta = 1$. Then, for the current ε and δ, problem (2.31) is solved using the proposed methodology. Then, the trajectory is updated to the current optimal. Next, ε and δ are decreased. Then, go back to the optimization step and continue.

This approach proved its validity together with the heavy-ball sequential method previously proposed. In fact, the approximate barrier functional introduces a nonlinear term within the cost function which can be handled by the suggested methodology.

Moreover, it is important to underline that this strategy works also exploiting other sequential optimization methods.

Chapter 3

Algorithm Implementation

In order to test the proposed method and compare its effectiveness with respect to other algorithms an implementation coded in Python has been developed. The whole code is structured in a way such that flexibility with respect to cost function, methodology and system is granted, also in forecast of a possible definition of a proper Python toolbox.

3.1 Python Code

The code produced together with this work is aimed to provide an useful test-bench for the proposed algorithm addressed to the resolution of any discrete time optimal control problem, not only a particular case study. Thus, flexibility and modularity represent the core concepts around which the whole code has been developed.

Furthermore, the whole code has been developed following the strategy depicted by Algorithm 3, namely the core functionalities which must be provided are: calculating a suitable descent direction and producing a (unfeasible) curve which is going to be projected as a feasible trajectory.

Alongside with some auxiliary functions, five main functional segments are defined:

- Sequential Optimization: the *main* function of this toolbox, allows the user to define parameters such as reference trajectory, number of iterations... It provides the optimal trajectory.
- Descent Method: produces the descent direction according to different approaches e.g. open-loop sequential, closed-loop sequential and first order PRONTO
- Update Trajectory: evaluates the new curve and perform the projection to a feasible trajectory

- System: allows the user to define the system under control. Receiving the control input and current state, it returns the next state (allowing dynamics integration) and the system Jacobian (needed for first order optimization).
- Cost: implements the cost function. Returns the current cost and cost Jacobian with respect to the current control input and state. Thanks to the previously discussed barrier function, it is possible to include in this function also the desired constraints.

Among these fundamental elements, some auxiliary functions are defined such as a Linear Quadratic Regulator problem solver, an useful tool in different applications e.g. gain matrices calculations and first order PRONTO.

A basic representation of the workflow behind the code is represented in figure 3.1. The whole sequential optimization phase is fully costumizable by the user, in terms of descent methods and approaches to compute the descent direction.

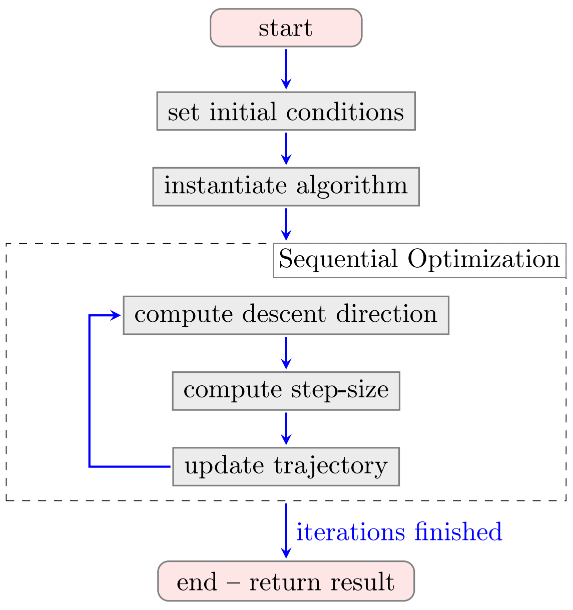


Figure 3.1: Code workflow

In figure 3.2, the basic code modules adopted to compute the descent direction at each iteration are represented. Notice tha both the system dynamics and the cost functions are defined by the user.

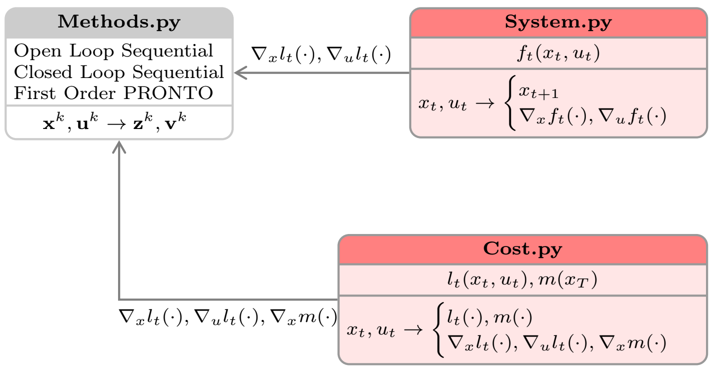

Methods.py
Open Loop Sequential
Closed Loop Sequential
First Order PRONTO
$\mathbf{x}^k, \mathbf{u}^k \to \mathbf{z}^k, \mathbf{v}^k$
$\nabla_x l_t(\cdot), \nabla_u l_t(\cdot)$
System.py
$f_t(x_t, u_t)$
$x_t, u_t \to \begin{cases} x_{t+1} \\ \nabla_x f_t(\cdot), \nabla_u f_t(\cdot) \end{cases}$
Cost.py
$l_t(x_t, u_t), m(x_T)$
$\nabla_x l_t(\cdot), \nabla_u l_t(\cdot), \nabla_x m(\cdot)$
$x_t, u_t \to \begin{cases} l_t(\cdot), m(\cdot) \\ \nabla_x l_t(\cdot), \nabla_u l_t(\cdot), \nabla_x m(\cdot) \end{cases}$

Figure 3.2: Descent direction computation – in red the user defined elements

In figure 3.3, the basic code modules adopted to update the current trajectory at each iteration are represented. Notice tha both the system dynamics and the cost functions are defined by the user.

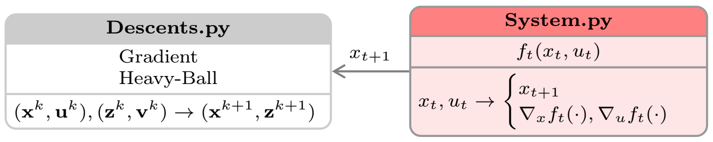

Descents.py
Gradient
Heavy-Ball
$(\mathbf{x}^k, \mathbf{u}^k), (\mathbf{z}^k, \mathbf{v}^k) \to (\mathbf{x}^{k+1}, \mathbf{z}^{k+1})$
x_{t+1}
System.py
$f_t(x_t, u_t)$
$x_t, u_t \to \begin{cases} x_{t+1} \\ \nabla_x f_t(\cdot), \nabla_u f_t(\cdot) \end{cases}$

Figure 3.3: Current trajectory update – in red the user defined elements

3.2 Inverse Pendulum Problem

The presented methodology and implementation is designed to be applied to any kind of nonlinear discrete-time optimal control problem in many and various field of engineering, e.g. trajectory optimization [14] and dynamics exploration [15].

In this work, for the sake of clarity, a simple and comprehensive example is considered: an inverted pendulum system, a classical case study in control systems theory. Even though simple, this framework still allows to consider nonlinear dynamics, trajectory tracking and multiple constraints both on states and inputs.

3.2.1 Problem

A simple inverted pendulum is hereby considered: Where k represents the friction coefficient, u is the input torque, g gravity, m mass of the pendulum, considered applied at its end, and l pendulum's length.

The system dynamics is represented as:

$$ml^2\ddot{\theta} = mgl\sin(\theta) - kl\dot{\theta} + u$$

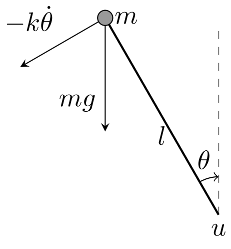


Figure 3.4: The inverse pendulum setup

In order to provide a state-space representation, the angular position θ reads as $x_1 \in \mathbb{R}$ while angular velocity $\dot{\theta}$ as $x_2 \in \mathbb{R}$. Thus, system dynamics can be rewritten in the state space as:

$$\begin{cases} \dot{x}_1 = x_2 \\ \dot{x}_2 = \frac{g}{l}\sin(x_1) - \frac{k}{ml}x_2 + \frac{1}{ml^2}u \end{cases}$$

Defining the state vector $x \in \mathbb{R}^2$:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

The nonlinear system can be written in the form:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ \frac{g}{l}\sin(x_1) - \frac{k}{ml}x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix} u \tag{3.1}$$

Which can be written in a more compact way as:

$$\dot{x} = f(x) + bu$$

Where $b = \begin{bmatrix} 0 & \frac{1}{ml^2} \end{bmatrix}^\top$. It worth noticing that, in this case, the input enters linearly.

By now, provided model is inherently a continuous time system. Thus, in order to be suitable for the developed discrete-time methodology, a discretization process needs to be performed.
In this case, discretization via Euler's method is considered. This is a basic explicit method for numerical integration of ordinary differential equations. In particular, given an Ordinary Differential Equation (ODE):

$$\dot{y}(t) = f(t, y(t)) \qquad y(t_0) = y_0$$

it is chosen a value $d \in \mathbb{R}$ discretization time step and set $t_n = t_0 + nd$. Now, one step of the Euler method from t_n to $t_{n+1} = t_n + d$ is:

$$y_{n+1} = y_n + df(t_n, y_n) \tag{3.2}$$

the value of y_n is an approximation of the solution to the ODE at time t_n, namely: $y_n \approx y(t_n)$.

Applying the approach of (3.2) to the continuous time system (3.1) results in the following discrete time system:

$$\begin{bmatrix} x_{1,t+1} \\ x_{2,t+1} \end{bmatrix} = \begin{bmatrix} x_{1,t} \\ x_{2,t} \end{bmatrix} + d \begin{bmatrix} x_{2,t} \\ \frac{g}{l}\sin(x_{1,t}) - \frac{k}{ml}x_{2,t} \end{bmatrix} + d \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix} u_t \tag{3.3}$$

which can be written in a more compact way as:

$$x_{t+1} = x_t + df(x_t) + bu_t$$

The sampling time chosen in this study is $d = 10^{-3}$.

The system's parameters are reported in Table 3.1.

$\boldsymbol{l}$	$\boldsymbol{m}$	$\boldsymbol{k}$
$[m]$	$[kg]$	
1	1	0.5

Table 3.1: System's parameters

It is important to underline that in these applications the initial conditions are chosen as:

$$x_{\text{init}} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

In the proposed methods, the system's dynamics Jacobian is required, which is defined as:

$$\nabla_x f(x^t, u^t)^\top = \begin{bmatrix} 1 & d \\ d\left[\frac{g}{l}\cos(x_1^t)\right] & 1 - d\left[\frac{k}{ml}\right] \end{bmatrix}$$

$$\nabla_u f(x^t, u^t)^\top = \begin{bmatrix} 0 \\ d\left[\frac{1}{ml^2}\right] \end{bmatrix}$$

Cost function

The cost function chosen for this illustrative example is a classical quadratic function, this kind of functions are often adopted as they allow to express, as quadratic form, the deviations of the variables of interest from their desired values. The algorithm thus finds those controller settings that minimize undesired deviations, however, it may not be possible to achieve the desired values of all target variables This property is particularly useful when dealing with trajectory tracking.

Moving from the generic definition:

$$F(\mathbf{x}, \mathbf{u}) = \sum_{t=0}^{T-1} \ell_t(x_t, u_t) + m(x_T)$$

the quadratic cost function considered reads:

$$F(\mathbf{x}, \mathbf{u}) = \sum_{t=0}^{T-1} \left(x_t^\top Q_t x_t + u_t^\top R_t u_t \right) + x_T^\top Q_f x_T \tag{3.4}$$

where, considering $n_x = 2$, $n_u = 1$, $Q_t \in \mathbb{R}^{n_x \times n_x}$, $Q_f \in \mathbb{R}^{n_x \times n_x}$ and $R \in \mathbb{R}^{n_u \times n_u}$. In particular:

$$\begin{aligned} \ell_t(x_t, u_t) &= x_t^\top Q x_t + u_t^\top R u_t \\ m_t(x_t) &= x_T^\top Q_f x_T \end{aligned}$$

In this work, constant matrices are chosen, namely $Q_t \equiv Q$ and $R_t \equiv R$ for all $t \in \mathbb{T}_{[0,T-1]}$. Defined as:

$$\begin{aligned} Q &= \begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix} \\ Q_f &= \begin{bmatrix} 10^2 & 0 \\ 0 & 10^4 \end{bmatrix} \\ R &= \begin{bmatrix} 0.5 \end{bmatrix} \end{aligned}$$

the higher cost associated to the second state - the angular velocity - at the final time $t = T$ is motivated by the desire to move the pendulum to a steady final position.

In the proposed methods, the gradient of the cost function is required, which is defined as:

$$\begin{aligned} \nabla_x \ell(x^t, u^t) &= 2Qx_t \\ \nabla_u \ell(x^t, u^t) &= 2Rx_t \\ \nabla_x m(x_T) &= 2Q_f x_T \end{aligned}$$

Trajectory Tracking

In many applications, it is useful for each state and input to track an user-defined reference trajectory. Under these circumstances, a quadratic cost function is particularly useful. In fact, these sequential algorithm can be easily extended towards this goal just modifying the cost function weighting the distance from the reference trajectory $\mathbf{x}^{\text{ref}}$ defined for all $t \in \mathbb{T}_{[0,T-1]}$:

$$J(\mathbf{x}, \mathbf{u}) = \sum_{t=0}^{T-1} \ell(x_t, u_t, x_t^{\text{ref}}, u_t^{\text{ref}}) + m(x_T, x_T^{\text{ref}})$$

In particular, considering a quadratic cost:

$$F(\mathbf{x},\mathbf{u}) = \sum_{t=0}^{T-1} \Big((x_t - x_t^{\mathrm{ref}})^\top Q (x_t - x_t^{\mathrm{ref}}) + (u_t - u_t^{\mathrm{ref}})^\top R (u_t - u_t^{\mathrm{ref}}) \Big) + (x_T - x_T^{\mathrm{ref}})^\top Q_f (x_T - x_T^{\mathrm{ref}})$$

Once this cost function is defined, algorithm is ready to go without further changes.
It is important to underline that, due to the implementation of this reference trajectory within the cost function does not guarantee perfect, zero-error tracking throughout the whole trajectory. However, this allows the designer to not care about dynamic feasibility of the reference signal, the optimization algorithm, in fact, minimizes the deviation from the reference value at any instant while keeping, also thanks to the projection operator, the trajectory feasible.

In this practical example, the trajectory to be tracked is associated just to the first state x_1, the angular position of the pendulum. In particular, it is requested to move the pendulum to the origin, namely $x_1 = \theta = 0$ and then, after few moments, to $x_1 = \theta = 0.5$ rad $\approx 28\,$deg. This reference trajectory is represented in figure 3.5.

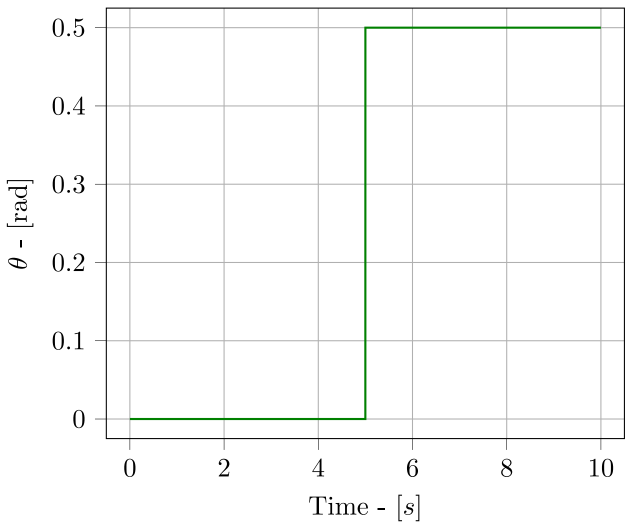


Figure 3.5: Angular position reference trajectory

On the other hand, reference trajectory such as $\mathbf{x}_2^{\mathrm{ref}} \equiv 0$ and $\mathbf{u}^{\mathrm{ref}} \equiv 0$ are chosen for both the second state - the angular velocity - and the input, in order to force them to be as small as possible. In particular, the request above $\mathbf{x}_2^{\mathrm{ref}}$, could represent the need for the optimizer to define the optimal way to connect two different zero-velocity trajectories.

Considering the presence of a trajectory to be tracked, the gradient of the cost function becomes:

$$
\begin{aligned}
\nabla_x \ell(x^t, u^t, x_t^{\text{ref}}, u_t^{\text{ref}}) &= 2Qx_t - 2Qx_t^{\text{ref}} \\
\nabla_u \ell(x^t, u^t, x_t^{\text{ref}}, u_t^{\text{ref}}) &= 2Rx_t - 2Ru_t^{\text{ref}} \\
\nabla_x m(x_T, x_T^{\text{ref}}) &= 2Q_f x_T - 2Q_f x_T^{\text{ref}}
\end{aligned}
$$

Constraints

Constraints, accordingly to the proposed methodology, are handled by an approximate barrier function as proposed in continuous time in [8]. This term, which intervenes as a cost function, makes the optimization problem nonlinear also in the cost.

In this work, two main constraints are introduced, one on the input u (mimicking the maximum torque available from the actuator) and another over the angular speed $\dot{\theta} = x_2$ (which may be an external requirement). Formally, they are expressed as:

$$
\begin{aligned}
|u_t - u_{\max}| &\le 0 \\
|x_{2,t} - x_{2,\max}| &\le 0
\end{aligned}
\qquad \forall t \in \mathbb{T}_{[0,T-1]}
$$

which then maps into 4 inequalities contraints, namely $n_g = 4$:

$$
\begin{aligned}
g_1(x_t, u_t) &= u_t - u_{\max} \\
g_2(x_t, u_t) &= u_{\max} - u_t \\
g_3(x_t, u_t) &= x_{2,t} - x_{2,\max} \\
g_4(x_t, u_t) &= x_{2,\max} - x_{2,t}
\end{aligned}
\qquad \forall t \in \mathbb{T}_{[0,T-1]}
$$

These contraints are then approximated as a nonlinear term in the cost function which reads, for each $t \in \mathbb{T}_{[0,T-1]}$, as:

$$
-\varepsilon \sum_{j=1}^{n_g} \log(-g_j(t, x_t, u_t)) \tag{3.5}
$$

In this example, the only constraint implemented is related to the angular velocity x_2, it is requested, in fact, to keep bounded the angular speed, namely:

$$
-0.2 \le x_2 \le 0.2
$$

Algorithms

In this illustrative implementation two optimization algorithms are compared. As a reference, it is chosen first-order PRONTO, the first order version of Algorithm 1. In fact, a first order approximation of the cost function is considered during the resolution of the SQP problem associated to the research for the descent direction. Namely $H^k = \mathbf{I}\ \forall k \geq 0$, where $\mathbf{I}$ is the identity matrix.

On the other hand, the proposed Heavy-Ball sequential method is tested, as illustrated in Algorithm 6. In this case a fixed step-size $\gamma^k = 0.1$ is adopted $\forall k \geq 0$.

3.2.2 Results

The proposed algorithm is hereby implemented in the previously discussed practical example. In particular, its performances are compared with the first order implementation of PRONTO.

As previously discussed, one of the major advantages of the proposed technique is related to the fact that the descent direction is free and not constrained to the space tangent to the current trajectory.

From these simulations, the Heavy-Ball enhanced sequential optimization proved faster than PRONTO.

Unconstrained Optimization

A first test is performed dealing with the following optimization problem:

$$\begin{aligned} \min_{\substack{x_1,\ldots,x_N \\ u_0,\ldots,u_{T-1}}} \quad & \sum_{t=0}^{T-1} \Big[(x_t - x_t^*)^\top Q(x_t - x_t^*) + (u_t - u_t^*)^\top R(u_t - u_t^*)\Big] \\ & + (x_T - x_T^*)^\top Q_f (x_T - x_T^*) \qquad (3.6) \\ \text{subj. to} \quad & x_{t+1} = f(x_t, u_t) \qquad t \in \mathbb{T}_{[0,T-1]} \\ & x_0 = x_{\text{init}} \end{aligned}$$

It is possible to notice that no external constraints, with the exception of dynamics, are considered.

In figure 3.6 the behavior of the first-order version of PRONTO is compared to the proposed Heavy-Ball sequential algorithm. It is possible to observe that Heavy-Ball sequential converges faster to the optimal cost. This is more evident in figure 3.7, where the cost error between the first and last iteration is represented. In this graphs, to ease the notation, it is defined as:

$$F^k = F(\mathbf{x}^k, \mathbf{u}^k)$$

namely, the cost at each iteration and as:

$$F^* = F(\mathbf{x}^*, \mathbf{u}^*)$$

the cost at the last iteration, where $(\mathbf{x}^*, \mathbf{u}^*)$ as the corresponding trajectory.

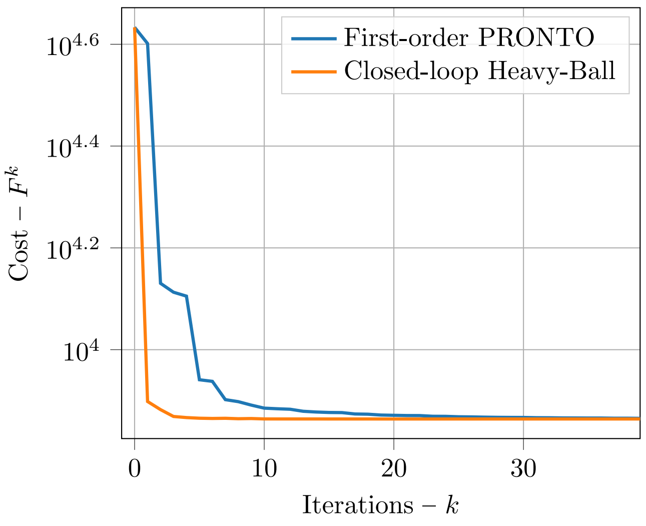


Figure 3.6: Algorithm comparison – First-order PRONTO and Heavy-Ball Sequential

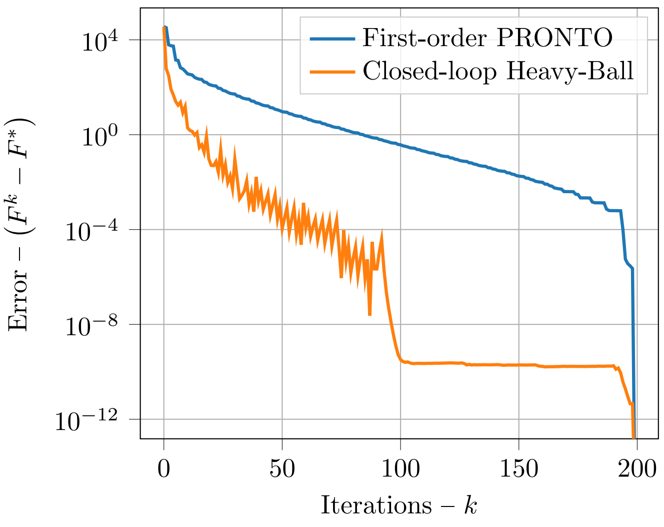
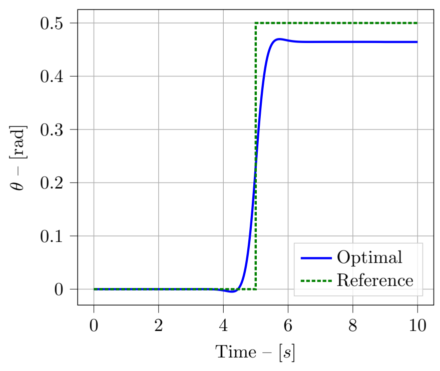
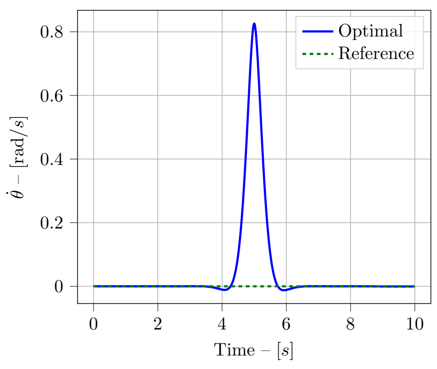


Figure 3.7: Cost evolution – First-order PRONTO and Heavy-Ball Sequential, difference between the cost at each iteration F^k and at the last one F^*

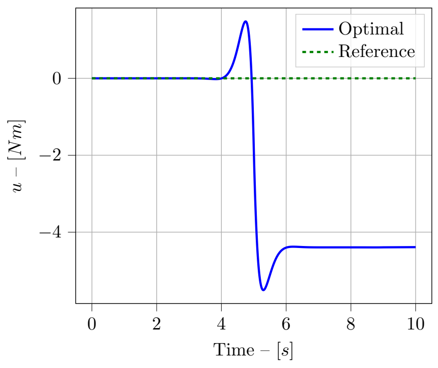
Finally, the optimal trajectory is depicted in figure 3.8. It is important to underline that the same optimal trajectory is achieved both via first-order PRONTO and Heavy-Ball Sequential.



(a) Position



(b) Velocity



(c) Input

Figure 3.8: Optimal trajectory – in blue the optimal trajectory, in dashed green the reference signals

From figure 3.8a it may seem that the optimizer failed with respect to the task of tracking the position reference signal. However, it is necessary to keep in mind that, due to the chosen cost function, the optimization process was also trying to minimize the distance between the input and its reference signal (figure 3.8c).

Constrained Optimization

The efficiency of the proposed methodology is proved also considering a constrained scenario, where, together with the dynamics constraints, some external restrictions are applied to the system. This is useful in many applications where limitations over input, trajectory and speed are considered.

The problem hereby optimized introduces a constraint over the maximum speed achievable during the pendulum motion.

$$\begin{aligned}
\min_{\substack{x_1,\dots,x_N\\u_0,\dots,u_{T-1}}} \quad & \sum_{t=0}^{T-1}\Big[(x_t - x_t^*)^\top Q(x_t - x_t^*) + (u_t - u_t^*)^\top R(u_t - u_t^*)\Big] \\
& \quad + (x_T - x_T^*)^\top Q_f(x_T - x_T^*) \\
\text{subj. to} \quad & x_{t+1} = f(x_t, u_t) \qquad t \in \mathbb{T}_{[0,T-1]} \\
& x_0 = x_{\text{init}} \\
& x_{2,t} \leq x_{2,\max} \qquad \forall t \in \mathbb{T}_{[0,T-1]} \\
& x_{2,t} \geq x_{2,\min} \qquad \forall t \in \mathbb{T}_{[0,T-1]}
\end{aligned} \tag{3.7}$$

In figure 3.9 the behavior of the two algorithms, still first-order PRONTO and Heavy-Ball sequential is represented. Since the barrier function shrinks every 10 iterations, its effects are evident from the graphs in the form of those *stairs*.

Considering the optimization of constrained system, the convergence speed of the two algorithms does not present huge differences, as it is possible to observe in figure 3.10. However, the Heavy-Ball sequential method, as shown in figure 3.11, within very few iterations produces a trajectory which follows the position reference signal.

In this graphs, to ease the notation, it is defined as:

$$F^k = F(\mathbf{x}^k, \mathbf{u}^k)$$

namely, the cost at each iteration and as:

$$F^* = F(\mathbf{x}^*, \mathbf{u}^*)$$

the cost at the last iteration, where $(\mathbf{x}^*, \mathbf{u}^*)$ the corresponding trajectory.

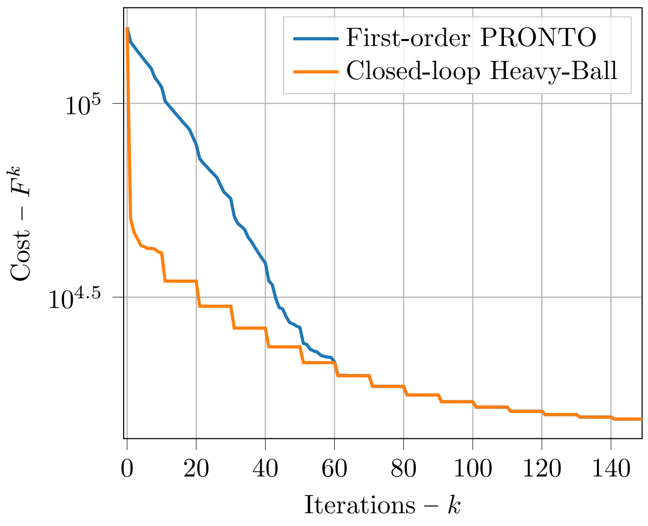


Figure 3.9: Algorithm comparison – First-order PRONTO and Heavy-Ball Sequential

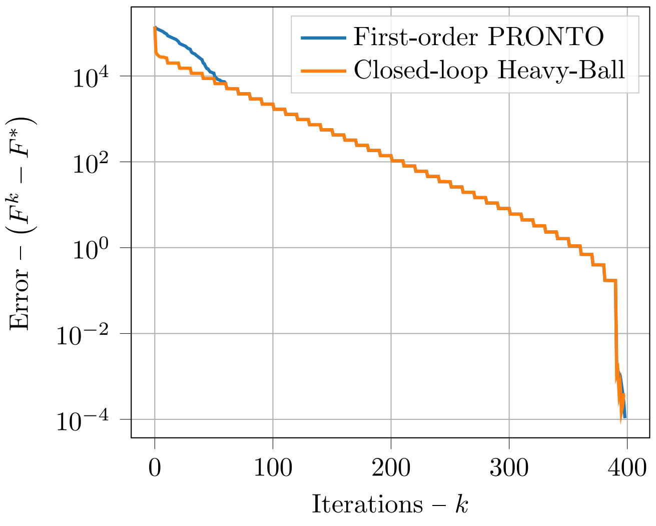
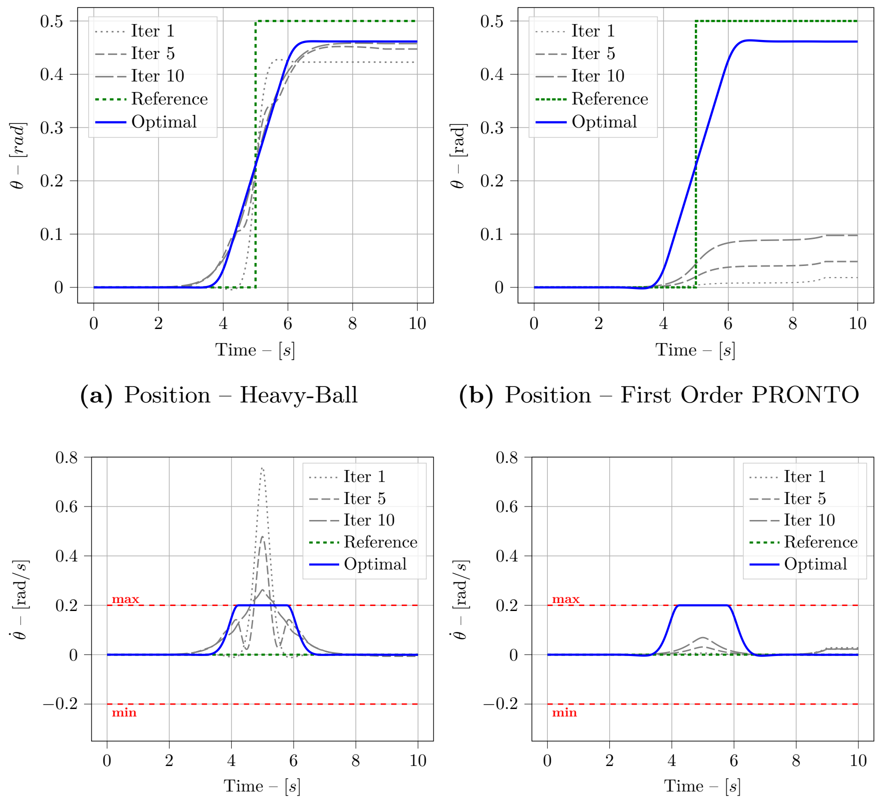


Figure 3.10: Cost evolution – First-order PRONTO and Heavy-Ball Sequential, difference between the cost at each iteration F^k and at the last one F^*



(a) Position – Heavy-Ball **(b)** Position – First Order PRONTO

(c) Velocity – Heavy-Ball **(d)** Velocity – First Order PRONTO

Figure 3.11: State trajectory evolution – in dashed grey the corresponding trajectory at each iteration

From figure 3.11c and 3.11d the effects of the barrier function applied to the constrained problem can be observed. In particular, it is possible to see how, iteration after iteration, the velocity trajectory is pushed towards the boundaries.

Finally, the optimal trajectory is depicted in figure 3.12. It is important to underline that the same optimal trajectory is achieved both via first-order PRONTO and Heavy-Ball Sequential.

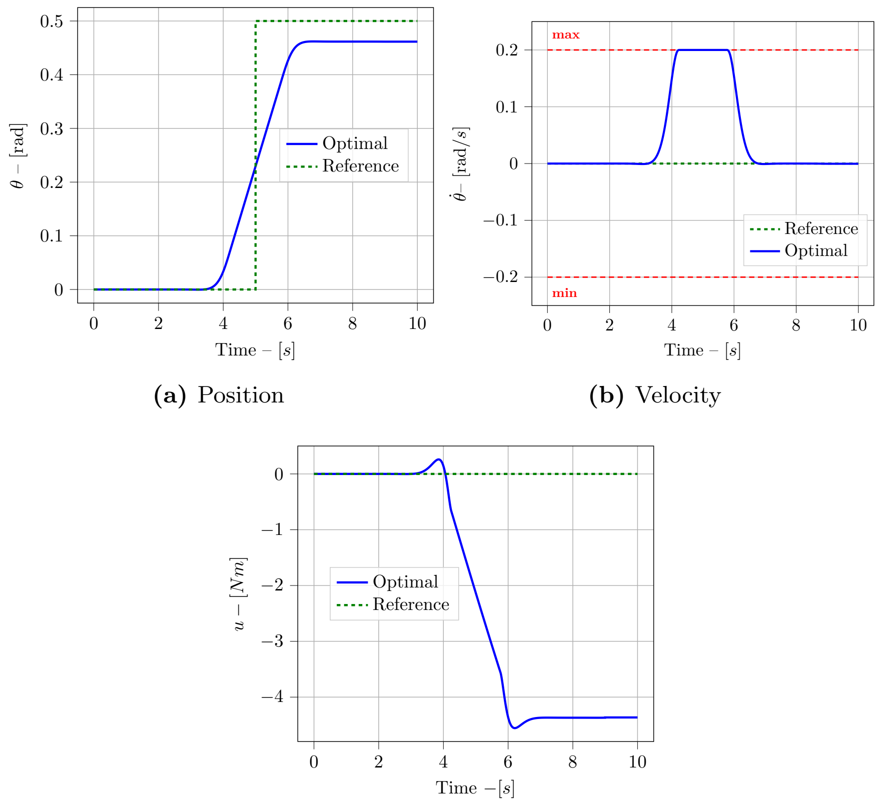
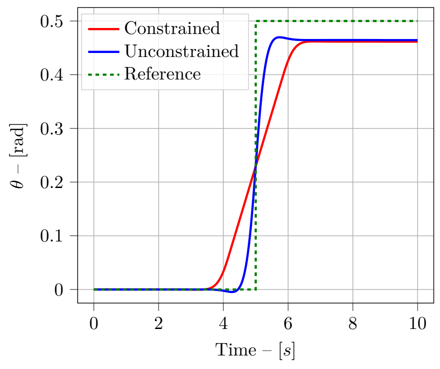


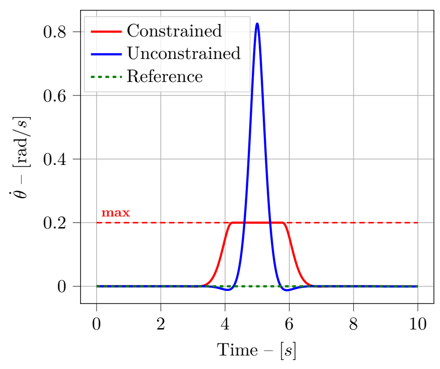
Figure 3.12: Optimal trajectory – in blue the optimal trajectory, in dashed green the reference signals.

The effectiveness of the barrier function can be observed in particular from figure 3.12b. In order to obtain an optimal trajectory compatible with the constraints, in fact, the velocity gets saturated to its maximum (feasible) value.

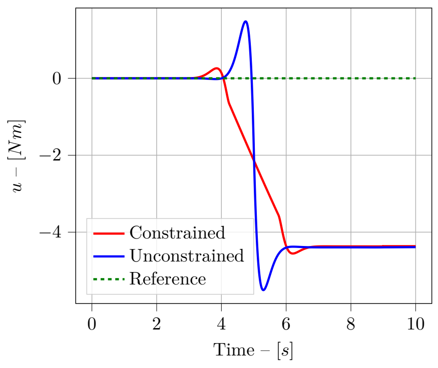
A comparison between the constrained and unconstrained state input trajectory is represented in figure 3.13.



(a) Position



(b) Velocity



(c) Input

Figure 3.13: Unconstrained and constrained optimization – comparison between constrained (red) and unconstrained (blue) optimal trajectories. In green the reference signals.

Conclusions

In this thesis a novel methodology for nonlinear optimal control problems with input and state constraints is proposed. In particular, the proposed approach generalizes a so-called closed-loop sequential method. The dynamics is enhanced by the introduction of a closed-loop structure exploiting the beneficial effects of state feedback. This results in a more stable and efficient algorithm with respect to its open-loop counterparts.

This methodology is further improved by the exploitation of the Heavy-ball method in order to speed-up algorithm's convergence.

State and input constraints are effectively managed thanks to the introduction of an approximate barrier function alongside with the traditional problem cost function. This results in a relaxed optimization problem in which constraints are embedded within the cost function.

It is important to underline that proposed method is designed in order to produce, at each iteration, a trajectory capable of satisfying the dynamic of the system, namely, a feasible trajectory is always available even if sub-optimal.

Numerical computations finally show the effectiveness of the proposed strategy both in an unconstrained and constrained framework.

Appendix A

Basics on Optimization

A.1 Descent Methods for Unconstrained Optimization

Considering a generic unconstrained optimization problem:

$$\begin{aligned} \min_{x} \quad & \ell(x) \\ \text{subj. to} \quad & x \in \mathbb{R}^{n_x} \end{aligned} \tag{A.1}$$

Most interesting algorithms for this unconstrained optimization problem rely on an important idea, called iterative descent that works as follows: starting from an initial guess x^0 and successively generate vectors $x^1, x^2, \ldots$ such that the cost is decreased at each iteration, that is:

$$\ell(x^{k+1}) < \ell(x^k) \quad k = 0, 1, \ldots$$

In doing so, current solution estimate is successively improved, hoping to decrease the cost all the way to its minimum.

The candidate solution at iteration $k \in \mathbb{N}$ is, in general, updated accordingly to:

$$x^{k+1} = x^k + \beta^k \zeta^k \tag{A.2}$$

where $\beta^k \in \mathbb{R}$ step-size, $\zeta^k \in \mathbb{R}^{n_x}$ descent direction.
The gradient method applied to A.2 reads:

$$x^{k+1} = x^k - \beta^k \nabla \ell(x^k) \tag{A.3}$$

The Newton method applied to A.2 reads:

$$x^{k+1} = x^k - \beta^k \left(\nabla^2 \ell(x^k)\right)^{-1} \nabla \ell(x^k) \tag{A.4}$$

In this case the notation x^k refers to a vector x at the algorithm's k-th iteration.

A.2 Sequential Quadratic Programming

Considering a generic constrained optimization problem:

$$\begin{aligned} \min_{x\in\mathbb{R}^{n_x}} \quad & \ell(x) \\ \text{subj. to} \quad & h(x) = 0 \\ & g(x) \leq 0 \end{aligned} \tag{A.5}$$

where $\ell : \mathbb{R}^{n_x} \to \mathbb{R}$, $h : \mathbb{R}^{n_x} \to \mathbb{R}^p$ and $g : \mathbb{R}^{n_x} \to \mathbb{R}^r$ are twice continuosly differentiable functions. We denote the feasible set for problem (A.5) by X.

The Sequential Quadratic Programming (SQP) is an iterative procedure which models the optimization problem (A.5) for a given iterate x^k, by a Quadratic Programming (QP) problem, solves that such approximation, and then uses the solution to construct a new iterate x^{k+1}.

The QP approximations should reflect the local properties of (A.5) with respect to the current iterate x^k. Therefore, a natural idea is to replace the objective functional by its local quadratic approximation while constraint functions h and g by their local affine approximations. This leads to the following basic SQP algorithm which obtains a step Δx by solving:

$$\begin{aligned} \Delta x^k = \arg\min_{\Delta x} \quad & \nabla \ell(x^k)^\top \Delta x + \frac{1}{2}\Delta x^\top H^k \Delta x \\ \text{subj. to} \quad & h(x^k) + \nabla h(x^k)^\top \Delta x = 0 \\ & g(x^k) + \nabla g(x^k)^\top \Delta x \leq 0 \end{aligned} \tag{A.6}$$

Then, set

$$x^{k+1} = x^k + \Delta x^k \tag{A.7}$$

A.3 Linear Quadratic Regulator

Considering the following optimal control problem:

$$\begin{aligned} \min_{\substack{x_1,\dots,x_N \\ u_0,\dots,u_{T-1}}} \quad & \sum_{t=0}^{T-1} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^\top \begin{bmatrix} Q_t & S_t^\top \\ S_t & R_t \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix} + x_T^\top Q_T x_T \\ \text{subj. to} \quad & x_{t+1} = A_t x_t + B_t u_t \quad t \in \mathbb{T}_{[0,T-1]} \\ & x_0 = x_{\text{init}} \end{aligned} \tag{A.8}$$

with $Q_t = Q_t^\top \geq 0$, $Q_T = Q_T^\top \geq 0$ and $R_t = R_t^\top > 0$.

Using, for example, dynamic programming, it can be proved that this problem admits a closed-form solution that relies on the following steps.

Set $P_T = Q_T$. Then, recursively compute backward in time for $t = T?1, \dots, 0$

$$\begin{aligned} P_t = Q_t &+ A_t^\top P_{t+1} A_t \\ &- (S_t^\top + A_t^\top P_{t+1} B_t)(R_t + B_t^\top P_{t+1} B_t)^{-1}(S_t + B_t^\top P_{t+1} A_t) \end{aligned}$$

which is commonly known as difference Riccati equation.
Once matrices P_t have been obtained the optimal input can be computed as a feedback of the state given by:

$$\begin{aligned} K_t &= -(R_t + B_t^\top P_{t+1} B_t)^{-1}(S_t + B_t^\top P_{t+1} A_t) \\ u_t^* &= K_t x_t^* \\ x_{t+1}^* &= (A_t + B_t K_t) x_t^* \end{aligned}$$

for all $t \in \mathbb{T}_{[0,T-1]}$ and with $x_0^* = x_{\text{init}}$.

A more general LQR formulation may include affine dynamics, i.e. $x_{t+1} = A_t x_t + B_t u_t + c_t$. Moreover, one can consider a desired reference state-input signal $(\mathbf{x}^d, \mathbf{u}^d)$ to be optimally tracked. This task can be formalized using the following "tracking objective"

$$\begin{bmatrix} x_t - x_t^d \\ u_t - u_t^d \end{bmatrix}^\top \begin{bmatrix} Q_t & S_t^\top \\ S_t & R_t \end{bmatrix} \begin{bmatrix} x_t - x_t^d \\ u_t - u_t^d \end{bmatrix} \tag{A.9}$$

which can be equivalently posed, ignoring some constant terms, as a linear-quadratic minimization:

$$\sum_{t=0}^{T-1} \left[2 \begin{bmatrix} q_t \\ r_t \end{bmatrix}^\top \begin{bmatrix} x_t \\ u_t \end{bmatrix} + \begin{bmatrix} x_t \\ u_t \end{bmatrix}^\top \begin{bmatrix} Q_t & S_t^\top \\ S_t & R_t \end{bmatrix} \begin{bmatrix} x_t \\ u_t \end{bmatrix} \right] \tag{A.10}$$

with $q_t = -Q_t x_t^d - S_t^\top u_t^d$ and $r_t = -S_t x_t^d - R_t u_t^d$. One can also consider terminal costs and finally rearrage terms as:

$$\begin{aligned} \min_{\substack{x_1,\dots,x_N \\ u_0,\dots,u_{T-1}}} \quad & \sum_{t=0}^{T-1} \begin{bmatrix} 1 \\ x_t \\ u_t \end{bmatrix}^\top \begin{bmatrix} 0 & q_t^\top & r_t^\top \\ q_t & Q_t & S_t^\top \\ r_t & S_t & R_t \end{bmatrix} \begin{bmatrix} 1 \\ x_t \\ u_t \end{bmatrix} + \begin{bmatrix} 1 \\ x_t \end{bmatrix}^\top \begin{bmatrix} 0 & q_T^\top \\ q_T & Q_T \end{bmatrix} \begin{bmatrix} 1 \\ x_t \end{bmatrix} \\ \text{subj. to} \quad & \begin{bmatrix} 1 \\ x_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ c_t & A_t \end{bmatrix} \begin{bmatrix} 1 \\ x_t \end{bmatrix} + \begin{bmatrix} 0 \\ B_t \end{bmatrix} u_t \qquad t \in \mathbb{T}_{[0,T-1]} \\ & x_0 = x_{init} \end{aligned} \tag{A.11}$$

By defining an augmented state:

$$\tilde{x}_t = \begin{bmatrix} 1 \\ x_t \end{bmatrix}$$

and the corresponding matrices, one obtains the same structure in (A.8) and can apply the same formulas.

A.4 Armijo Rule – Backtracking Line Search

In order to improve the efficiency of an iterative minimization algorithm, it is possible to adopt an iterative algorithm back-tracking line search strategy, namely, a variable step-size β_t^k defined accordingly to the Armijo-Goldstein conditions is introduced [16]. This is aline search method to determine the maximum amount to move along a given search direction.

Given a starting position $\mathbf{x}$ and a search direction $\mathbf{z}$, the task of a line search is to determine a step size $\beta > 0$ that adequately reduces the objective function $f : \mathbb{R}^n \to \mathbb{R}$ (assumed smooth), i.e., to find a value of β that reduces $f(\mathbf{x} + \beta\mathbf{z})$ relative to $f(\mathbf{x})$.

Starting with a maximum candidate step size value $\beta_0 > 0$, using search control parameters $\tau \in (0, 1)$ and $c \in (0, 1)$, the backtracking line search algorithm can be expressed as follows:

1. Set $t = -cm$ and iteration counter $j = 0$ where $m = \nabla f(x)^\top \mathbf{z}$.
2. Until the condition is satisfied that $f(\mathbf{x}) - f(\mathbf{x} + \beta_j\mathbf{z}) \geq \beta_j t$, repeatedly increment j and set $\beta_j = \tau\beta_{j-1}$.
3. Return β_j as the solution.

Bibliography

[1] 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.

[2] O. Von Stryk, "Numerical solution of optimal control problems by direct collocation," in *Optimal control*, Springer, 1993, pp. 129–143.

[3] D. P. Bertsekas, W. Hager, and O. Mangasarian, *Nonlinear programming.* Athena Scientific Belmont, MA, 1998.

[4] J. Hauser, "A projection operator approach to the optimization of trajectory functionals," *IFAC Proceedings Volumes*, vol. 35, no. 1, pp. 377–382, 2002.

[5] S. Spedicato and G. Notarstefano, "Cloud-assisted distributed nonlinear optimal control for dynamics over graph," *IFAC-PapersOnLine*, vol. 51, no. 23, pp. 361–366, 2018.

[6] D. P. Bertsekas, *Dynamic programming and optimal control*, 2. Athena scientific Belmont, MA, 1995, vol. 1.

[7] R. Bellman, *Dynamic Programming.* Princeton University Press, 1957.

[8] J. Hauser and A. Saccon, "A barrier function method for the optimization of trajectory functionals with constraints," in *Proceedings of the 45th IEEE Conference on Decision and Control*, IEEE, 2006, pp. 864–869.

[9] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *USSR Computational Mathematics and Mathematical Physics*, vol. 4, no. 5, pp. 1–17, 1964.

[10] D. P. Bertsekas and J. N. Tsitsiklis, "Gradient convergence in gradient methods with errors," *SIAM Journal on Optimization*, vol. 10, no. 3, pp. 627–642, 2000.

[11] J. Hauser and D. G. Meyer, "The trajectory manifold of a nonlinear control system," in *Proceedings of the 37th IEEE Conference on Decision and Control (Cat. No. 98CH36171)*, IEEE, vol. 1, 1998, pp. 1034–1039.

[12] E. Ghadimi, H. R. Feyzmahdavian, and M. Johansson, “Global convergence of the heavy-ball method for convex optimization,” in *2015 European control conference (ECC)*, IEEE, 2015, pp. 310–315.

[13] S. Boyd and L. Vandenberghe, *Convex optimization.* Cambridge university press, 2004.

[14] S. Spedicato and G. Notarstefano, “Minimum-time trajectory generation for quadrotors in constrained environments,” *IEEE Transactions on Control Systems Technology*, vol. 26, no. 4, pp. 1335–1344, 2017.

[15] A. Rucco, G. Notarstefano, and J. Hauser, “Optimal control based dynamics exploration of a rigid car with longitudinal load transfer,” *IEEE Transactions on Control Systems Technology*, vol. 22, no. 3, pp. 1070–1077, 2013.

[16] L. Armijo, “Minimization of functions having lipschitz continuous first partial derivatives,” *Pacific Journal of mathematics*, vol. 16, no. 1, pp. 1–3, 1966.