POLYTECHNIC UNIVERSITY OF CATALONIA (UPC)

MASTER THESIS

Simultaneous trajectory and contact optimization with an augmented Lagrangian algorithm

---

Author
Joaquim ORTIZ DE HARO

Supervisors
Dr. Nicolas MANSARD
Dr. Justin CARPENTIER

A thesis submitted in fulfillment of the requirements for the degree of Master of Mathematics (MAMME)

Gepetto, LAAS-CNRS

INSA Toulouse

FME-UPC Barcelona

February 18, 2019

Abstract

Simultaneous trajectory and contact optimization with an augmented Lagrangian algorithm

by Joaquim ORTIZ DE HARO

Legged robots do not require a flat and smooth surface to advance and have, therefore, the potential to work in unstructured and complex environments. These robots, like humans and animals, can climb stairs and ladders and cross challenging terrain, which is essential for exploration and rescue applications.

However, walking is a complex task. The robot movement must be a consequence of creating contacts with the environment. Every time a new contact is made the dynamics become discontinuous. Also, legged robots are unstable and subject to many constraints, which restricts the potential movements. Finally, contact planning has a combinatorial structure: the robot chooses the optimal subset of contacts out of an infinite number of possible contact points.

Trajectory optimization in robotics is often decoupled into two independent modules. A problem formulation, that models the robot motion and transforms it into a mathematical optimization problem, and the resolution of this optimization problem with a suitable optimization solver. We rather envision this question as a two-way interaction between the two modules, that can not be considered separately. Indeed, the robotics problem formulation must also be a consequence of the understanding of the capabilities of the numerical optimization algorithm.

We propose a simultaneous trajectory and contact optimization with an augmented Lagrangian algorithm. The contacts with the environment are modeled in a continuous way with complementarity constraints. For example, the normal contact model is defined by two inequalities: positive force and distance to the surface, and a complementarity relation: either the force or the distance to the surface is zero.

These constraints are added as path constraints in a continuous optimal control formulation. Using a direct collocation, the formulation is converted into a non-linear constrained optimization problem. Optimization problems with complementarity constraints have a degenerated and challenging structure and do not fulfil basic constraints qualifications used by standard numerical solvers. Therefore, we propose to solve it using an augmented Lagrangian algorithm, which offers a good behaviour under this type of constraints and can be efficiently warmstarted.

In this thesis we present our preliminary results with a robot manipulator interacting with the environment. Optimizing simultaneously trajectory and contacts, we compute interesting motions such as multiple surface touching and pick and place tasks. We also outline our future plans to generate walking motions with legged robots.

Acknowledgements

I would like to thank my supervisors Dr. Nicolas Mansard and Dr. Justin Carpentier for guiding me in this exciting research project. During 6 months they have enthusiastically shared their knowledge and long term vision. Moreover, their passion for robotics and numerical optimization has inspired my current and future research interests and encouraged me to pursue a PhD.

I would also like to thank my friends and colleagues of the Gepetto Team, for a friendly welcome and fruitful discussions.

Contents

Chapter 1

Legged robots and trajectory optimization

1.1 Robotics and optimal control, a short introduction

1.1.1 Robot kinematics and dynamics

Geometry

Robot has become now a wide concept in our society and the research community. In this project, we call robot to a mechanical system composed of a set of rigid bodies attached between them using joints.

The configuration of the robot is defined by the values of its joints (for example, the turned angle of a revolution joint). However, we are usually interested in the position of its end effector. The end effector of the robot is one part of the body used to perform some useful task. For example, in manipulators the end effector carries a tool. In walking robots, the feet are the end effectors and are used to make contacts with the ground. The end effector placement $x_e \in SE(3)$ has 6 degrees of freedom (3 translations and 3 rotations) and lies in $SE(3)$, the space of rigid body motions (translation and rotation).

The robot direct geometry function $\phi(\cdot)$ (often named forward kinematic function in the literature) is the function that maps the configuration space of the robot (joint values) to the placement of the end effector. This function is non-linear and is a composition of elementary translations and rotations.

When the robot has a fixed base, for example a manipulator, its configuration is defined by the joint values $q \in \mathbb{R}^n$.

$$\begin{aligned} \phi &: \mathbb{R}^n \to SE(3) \\ \phi(q) &= x_e \end{aligned} \tag{1.1}$$

However, when the robot has a mobile base (meaning it can move around, like a car or a quadruped), its configuration is defined by the joints values and the placement of the base $q \in \mathbb{R}^n \times SE(3)$.

$$\begin{aligned} \phi &: \mathbb{R}^n \times SE(3) \to SE(3) \\ \phi(q) &= x_e \end{aligned} \tag{1.2}$$

The relation between joint and end effector velocities is linear for a given configuration. $J(q)$ is called the Jacobian and plays central role in robotics.

$$J(q) = \frac{\partial x_e}{\partial q} = \frac{\partial \phi(q)}{\partial q} \tag{1.3}$$

$$\dot{x}_e = \frac{d}{dt} x_e, \quad \dot{q} = \frac{d}{dt} q \tag{1.4}$$

$$\dot{x}_e = J(q)\dot{q} \tag{1.5}$$

The time derivative of the end effector placement is represented by the velocity $\dot{x}_e$, which contains its linear and angular components. $\dot{q}$ is the time derivative of the robot configuration. Accelerations quantities are related by:

$$\ddot{x} = \dot{J}(q)\dot{q} + J(q)\ddot{q} \tag{1.6}$$

Dynamics

So far, we have presented only kinematics relations (position, velocity and acceleration). However, a dynamic model is required to study the forces that the end effector can produce and the torques required to fulfil a task. Dynamics are necessary to work with contacts, where force cannot be controlled through velocity. Controlling forces is essential for friendly robotics and human interaction.

The relation between forces and movement is modelled with the following equation: (derived either applying Newton-Euler equation $F = ma$; $I\dot{\omega} + \omega \times I\omega = \tau$ or Lagrangian mechanics to a multibody system [15])

$$M(q)\ddot{q} + C(q, \dot{q}) + g(q) = \tau \tag{1.7}$$

where $M(q)$ denotes the mass matrix, $C(q, \dot{q})$ the Coriolis and centrifugal effects and $g(q)$ the gravity force. q, $\dot{q}$, $\ddot{q}$ are the robot configuration, velocities and accelerations. The configuration q represents the joint values (fixed base robot $q \in \mathbb{R}^n$) or the joint values and the base placement (mobile robot $q \in \mathbb{R}^n \times SE(3)$). Finally, τ is the collection of joint torques.

If the system is in contact with the environment, contact forces that are acting on the system are also included in the dynamics. The contact forces are exerting an equivalent torque at the joint level, given by:

$$\tau_c = J(q)^T f_c \tag{1.8}$$

where f_c is a vector of contact forces and $J(q)$ is the Jacobian of the position of the end effector where the force is applied. Therefore, the term τ in (1.7) is an addition of the joint torques and the contact force term τ_c.

In the particular case of mobile robots, such as walking robots, the configuration state $q \in \mathbb{R}^n \times SE(3)$ can be partitioned between an actuated (joints) and an underactuated (free floating base) part.

$$\begin{bmatrix} M_u \\ M_a \end{bmatrix} \ddot{q} + \begin{bmatrix} C_u \\ C_a \end{bmatrix} + \begin{bmatrix} g_u \\ g_a \end{bmatrix} = \begin{bmatrix} 0 \\ \tau_a \end{bmatrix} + \sum_k \begin{bmatrix} J_{k,u}^T \\ J_{k,a}^T \end{bmatrix} f_k \tag{1.9}$$

where f_k and J_k denote force and position Jacobian at contact k. The first 6 rows of (1.9) correspond to the underactuted dynamics. On the other side, the n last rows of (1.9) are the

classic model of a robot manipulator in contact. Denoting by p and L the linear and angular momentum, the first 6 rows can be rewritten as:

$$\begin{aligned} \dot{p} &= \sum f_k - mg \\ \dot{L} &= \sum (r_k - c) \times f_k \end{aligned} \tag{1.10}$$

where mg denotes the gravity force vector, r_k the position of contact k and c the center of mass position. This formulation is called Centroidal Dynamics and has applications in control and trajectory optimization [8]. It is a reduced model (only 6 variables) in comparison to a full body configuration. However, feasibility constraints implied by the whole body are difficult to express with this formulation.

1.1.2 Optimal control formulation

The trajectory optimization can be formulated as a continuous optimal control problem. Optimal control deals with the problem of finding a control law for a dynamic system so that a desired task is fulfilled in an optimal way. There are two type of variables: the state $x(t)$, a description of the configuration of the system; and the control $u(t)$, a chosen input that steers the evolution of the dynamical system. The evolution of the dynamical system (meaning how the state changes along time) is modelled with an ordinary differential equation $f(\cdot)$:

$$\dot{x}(t) = f(x(t), u(t)) \tag{1.11}$$

In particular, robot dynamics are modelled with a second order differential equation (1.7).

The goal of the control $u(t)$ is to reduce some cost function, for example, minimize the energy or the distance between the final configuration of the system and a predefined goal.

$$E(x(T)) + \int_0^T l(x(t), u(t))dt \tag{1.12}$$

where $E(\cdot)$ denotes the terminal cost and $l(\cdot)$ the path cost. The initial configuration $x(0)$ of the system is normally fixed, and the motion has a predefined duration T.

$$t \in [0, T] \tag{1.13}$$

$$x(0) = x_0 \tag{1.14}$$

Finally, the trajectory and the control inputs can be subject to some restrictions, called path constraints $h(\cdot)$.

$$h(x(t), u(t)) \geq 0 \tag{1.15}$$

The knowledge of an initial state x_0 and a control input trajectory $u(t)\ \forall t \in [0, T]$ determines the whole state trajectory $x(t)\ \forall t \in [0, T]$.

Formally, the problem is formulated as:

$$x(t) : \mathbb{R} \to \mathbb{R}^n \quad u(t) : \mathbb{R} \to \mathbb{R}^m$$

$$\min_{x(\cdot),u(\cdot)} E(x(T)) + \int_0^T l(x(t), u(t))dt \tag{1.16}$$

subject to : (1.17)

$$\begin{aligned} & f(x(t), u(t), t) = \dot{x}(t), \quad \forall t \in [0, T] \\ & h(x(t), u(t), t) \leq 0, \quad \forall t \in [0, T] \\ & x(0) = x_0 \end{aligned}$$

This optimization problem has an infinite dimension, as $u(t)$ and $x(t)$ belong to $\infty-$dimensional function space. An analytic solution exists only in simple examples. The problem can be solved numerical with either a direct or indirect method. Direct methods parameterize the problem with a finite number of variables, such that (1.16) becomes a finite dimension non linear program. This approach is informally described as: "First discretize, then optimize".

Direct methods are divided into two families:

- Shooting: only the control is an optimization variable. The state is computed implicitly using the dynamics equation. It can be either single shooting (only x_0 and control are optimization variables) or multiple shooting (few states at intermediate time frames are added as optimization variables)
- Collocation: both state and control are optimization variables. The dynamics are enforced as a constraint at some discrete times frames, called collocation points.

1.2 Challenges and approaches to multi-contact locomotion

Trajectory optimization for legged robots can be formulated as an optimal control problem (1.16) using the kinematic and dynamic model of the robot (1.7). Compared to other robotics systems, such as wheeled or flying robots, legged robots move by sequentially making contacts. The complexity of the problem lies mainly on three aspects:

- *Discontinuous hybrid dynamics*. The trajectory is divided into several phases. Inside each phase, all the contacts points remain constant and the dynamics are continuous (contacts forces change smoothly). However, when contacts are created the dynamics become discontinuous. The concatenation of contact phases is called the contact sequence.
- *Unstable dynamics*. Walking robots are unstable, specially bipeds. Stability restricts the possible motion of the robot. Moreover, it is an underactuated system and only the joint positions can be directly controlled with the motors. To move around, the legged robot must interact with the environment.
- *Contact combinatorial problem*. A contact sequence is related to a sequence of discrete choices of contact positions. Decomposing the trajectory into movement phases is hard in complex environments, where robots should make contacts with different limbs (foots and hands) and different candidate surfaces acyclically. There are successful methods for flat-surface walking with cyclic gaits, showing good results on

simulation and real robots. However, they use strong terrain assumptions and do not generalize to complex and acyclic motions. A key issue is to avoid the combinatorial structure when considering at the same time the possible contacts and the potential paths. A naive combinatoric approach of selecting different subsets of discrete contacts and then computing trajectories has exponential time complexity and does not scale for complex motions.

Overall, trajectory and contact planning requires a simultaneous handling of three subproblems: ***P1***: planning a guide path for the root of the robot; ***P2***: planning a discrete sequence of contacts. ***P3***: generating a trajectory between two postures of the contact sequence. These three problems should be solved simultaneously to get an optimal solution. [28]

We now introduce 4 different approaches to solve this problem. Our goal is to give a complete view of the field, highlighting the main ideas and contributions to this open problem. From these four ideas, our method is more related to the paper "A direct method for trajectory optimization of rigid bodies through contacts, Posa et al." [24] (Section 1.2.2). In this short review, we do not cover discrete contact formulations, another interesting approach. This family of algorithms are based on a discrete parametrization of the contact sequence and use mixed integer programming solvers [1].

1.2.1 Separation of trajectory and contact planning

A standard approach is to decouple the multi-contact locomotion problem into a sequence of easier subproblems. Each subproblem represents a module of a unidirectional pipeline, and they are solved individually and sequentially, using a suitable optimization algorithm (and heuristics) in each step. Here the main challenge is to design the modules and mutual interactions between them in a way that the generated motion is close to the global optimal one, while keeping the computation time low. This is the approach currently used by the team Gepetto at LASS-CNRS, where the trajectory optimization problem is divided into three modules: a contact sequence planner, a centroidal pattern generator and a whole body motion generation [28], [9]. A short introduction of the modules is presented here:

- The contact sequence planner computes a sequence of contacts considering the root (center of mass) and the reachability sets of the end-effectors (3D space around the root where the end effector can touch the environment). It first plans a path for the root that is close enough to the obstacles to create contacts but not too close to avoid collision. Then it generates a discrete sequence of whole-body configurations that are in static equilibrium along this path. This guarantees that the contact sequence is feasible for the following centroidal and whole body optimization.
- Centroidal pattern generator: with the fixed contact points, an optimal control formulation based on centroidal dynamics is used to compute the contact forces and the centroidal trajectory (1.10).
- Whole-body motion generator: given the contacts and the centroidal trajectory, the position of the end effectors and joint values is optimized, using either a smooth optimization (optimal control formulation) or a rapidly-exploring random tree (a sampling based exploration method) if the environment is cluttered.

This pipeline has already been implemented in real robots for walking and climbing stairs. In simulation, they are able to generate complex and acyclic motions in challenging terrain.

1.2.2 Direct trajectory optimization through contact

Michael Posa et al. [24] propose a trajectory optimization formulation that does not need a predefined contact sequence or modes, so that the optimization algorithm computes simultaneously the optimal contacts and whole body movements. They use a direct collocation method and contacts are modelled using complementarity constraints. This restriction imposes, with a continuous formulation, that either the contact force is zero or the distance to the surface is zero. The problem becomes a mathematical program with complementarity constraints (MPCC) and is optimized using sequential quadratic programming (SQP).

Complementarity constraints (Section 4.1), direct collocation (Section 1.2.2) and sequential quadratic programming (Section 2.2.3) are explained in following chapters of this thesis.

The motivation of this work is to remove the need of a predefined contact sequence. Optimizing over fixed contact sequences do not scale well for large systems, as the possible number of sequences grows exponentially with the number of contacts. In addition, these contacts sequences are required to be feasible, which is challenging to ensure in complex environments.

The optimization is formulated with a direct control approach, where both state and control are variables and dynamic equations are imposed as constraints (Section 1.2.2).

We only present here the planar (2D) contact dynamics (normal and tangential forces are always on same plane). The reader is referred to the original paper for an extension to 3D contacts.

The tangential force is split into a positive f_x^+ and a negative component f_x^-. The new slack variable γ represents the magnitude of the relative tangential velocity. Both normal and tangential contact models are represented with complementarity constraints. These constraints are imposed as paths constraints in the optimal control formulation.

$$
\begin{aligned}
f = [f_x^+,\ -f_x^-,\ f_z] \quad & \text{contact forces (z is normal direction)} \\
f_x^+, -f_x^-, f_z, \gamma \geq 0 & \\
x_e(q) \geq 0 \quad & \text{distance to surface} \\
\mu f_z - f_x^+ - f_x^- \geq 0 & \\
\gamma + \psi(q,\dot{q}) \geq 0 & \\
\gamma - \psi(q,\dot{q}) \geq 0 & \\
x_e(q) f_z = 0 & \\
(\mu f_z - f_x^+ - f_x^-)\gamma = 0 & \\
(\gamma + \psi(q,\dot{q}))f_x^+ = 0 & \\
(\gamma - \psi(q,\dot{q}))f_x^- = 0 &
\end{aligned}
\tag{1.18}
$$

$\psi(q,\dot{q})$ is the tangential velocity and μ the friction coefficient. These constraints prevent contact forces at a distance, enforce the friction cone and ensure that, if the contact point is sliding, the tangential force properly lies on the edge of the cone and directly opposes the direction of motion.

The optimization is solved with the SQP solver SNOPT [16]. In complex examples, a sequence of problems is solved by relaxing the complementarity constraint with a smooth approximation. This has the effect of allowing intermediate subproblems to exert contact forces at a small distance and experimentally improves the solutions.

In the paper, this formulation is applied to four examples, with computations times between few minutes and up to one hour. The most complex environment is the Fast Runnner, a planar model with 13 degree of freedom. The optimized trajectory finds a mode sequence (considering also the saturation of joint limits) different from the initial guess. Extra features are added to ensure convergence. Apart from the sequential relaxation of complementarity constraints, additional linear constraints are added to guide the solver away from poorly conditioned or infeasible regions.

1.2.3 Contact invariant optimization

Mordatch et al. [22] propose an approach to optimize simultaneously both the contacts and the trajectory. It is based on introducing a set of scalar continuous variables that indicates if contacts are active. These new variables affect both the cost and the dynamics (enabling and disabling contact forces). The domain of application includes complex behaviours in complex environments and it is not only limited to walking motions.

In this formulation, each end effector of the robot that can make contacts with the environment is numbered with and index i and has an associated continuous variable $c_i \geq 0$.

When c_i is large, contact i must be active (end effector touching some surface), while if c_i is small, contact i is inactive.

The time T is discretized in N intervals and K phases, with c_i constant inside each phase. Therefore variable c_i is a piecewise constant function $c_i(t) = c_{i,\phi(t)}$ where $\phi(t)$ indicates the movement phase at time t.

The state variables are the position and orientation of the root (center of mass) and the end effectors. The trajectory is represented using splines (whose coefficients are the optimization variables). The joint values can be implicitly computed using inverse kinematics.

The total cost includes four terms: $L_C(x)$ the novel contact invariant cost, $L_P(x)$ a physics violation cost , $L_T(x)$ the task cost (for example the terminal position) , $L_H(x)$ an optional term to provide hints to guide early phases of the optimization. The optimization variable x represents the parameterization of the state variables.

We briefly describe the different cost components, which clarifies the role of this new variables c_i

- Contact invariant cost

$$L_C(x) = \sum_{t,i} c_{i,\phi(t)}(\|e_i\|^2 + \|\dot{e}\|^2) \tag{1.19}$$

  The contact invariant cost penalizes the product of the variable c_i with a movement and position error metric of the end effector. $\dot{e}$ is the end effector velocity and e_i is a 4D contact violation error. It encodes the difference between the position and orientation with respect to a smooth formulation of the closest point. The interpretation of this cost is that, when c_i is large (meaning that there is a contact), the velocity and position errors of the end effector should be small.

- Physic Violation cost

  In the paper, the authors present the physics violation cost with a general formulation using a whole body model. However, they use a simplified model in the simulations.

In the general case (with a whole body model), movement, forces and torques are related by the dynamic equation (1.7). With this equation, the resulting sum of torques τ is computed from $q, \dot{q}$, and $\ddot{q}$. Denoting by s all the kinematic quantities: $\tau = \tau(s)$. External forces f^* and joint torques u^* are then computed by solving the following quadratic problem.

$$\min_{f,u} \left\| J^T f + Bu - \tau \right\| + fWf + uRu \tag{1.20}$$
$$\text{subject to} Af \leq b$$

where $f \in \mathbb{R}^{6E}$ is the concatenation of vector forces acting on the body (with E the number of end effectors). $Af \leq b$ represents a linear approximation of the friction cone. J is the Jacobian of the position of the end effectors. $A = A(s)\,, J = J(s)\,, b = b(s)$ depend on the robot configuration, that is fixed for this quadratic problem. R is a positive definite matrix $W = W(c_i)$ is a diagonal force regularization matrix depending on the auxiliary contact variables c_i. The 6 diagonal elements corresponding to the contact i are set to:

$$W_{j,j} = \frac{k_0}{c_i^2 + k_1} \quad \forall j : 6i - 5 < j < 6i \tag{1.21}$$

where $k_0, k_1 > 0$ are two small constant values. Using the optimal values f^* and u^* of (1.20) the physics violation cost is:

$$L_P(s) = \sum \left\| J(s)^T f^*(s) + Bu^*(s) - \tau(s) \right\|^2 \tag{1.22}$$

This cost will never reach zero because of the regularization matrix W in the quadratic problem (1.20). Therefore, the generated motion is never physically consistent.

The optimal trade-off between the costs L_C and L_P is achieved when c_i is large only for the end-effectors in contact which are necessary to generate the trajectory. The intuition is as follows: if all c_i are large, W matrix is small and the physics cost tend to zero, but the contact cost increases. Otherwise, if all c_i are small, the contact cost is zero but the physics cost grows.

- The task cost L_T includes the high level goals of the movement and generic penalization on control and acceleration.
- The hint cost L_H is introduced in the first iterations to ensure convergence. This hint is based on keeping the ZMP (Zero Moment Point) in the convex hull of the support region.

The optimization problem is solved using a L-BFGS solver. A continuation scheme is implemented by weighting differently the four terms in the cost through the iterations: in the first steps, only task and hint cost are active. Later, the hint cost is deactivated and the physics and contact costs are added with gradually increasing weights.

Execution times are around 2 and 10 minutes and the authors synthesize a variety of complex behaviours: climbing, getting-up, kicks, punches and interaction between characters and objects.

1.2.4 Phase parametrization

Winkler et al. [31] introduce a trajectory optimization formulation based on using continuous variables to parameterize the phase of the movement. The optimization automatically determines step-timings, footholds and 6D body motion over non-flat terrain.

The formulation is based on the continuous variables $\Delta T_{i,j}$, that represents duration of phase j of foot i. A phase of an individual foot can be either *flight* or *contact*, and they are set to be an alternating sequence. The number of phases is defined by the user, although time intervals can be compressed to have zero time duration. The gait pattern is completely defined by the set of $\Delta T_{i,j}$.

The robot is represented using simplified centroidal dynamics, working only with the center of mass (COM) position and orientation, feet position, and contact forces. An approximated kinematic restriction between the root and feet position is enforced in Cartesian space.

$$\begin{aligned} m\ddot{r}(t) &= \sum f_i(t) - mg \\ I\dot{\omega}(t) + \dot{\omega}(t) \times I\omega(t) &= \sum f_i(t) \times (r(t) - p_i(t)) \\ |R(\theta)[p_i(t) - r(t)] - \bar{p}_i| &< b \end{aligned} \tag{1.23}$$

where $r(t)$ is the COM position, $R(\theta)$ the rotation matrix, and $\omega(t)$ its angular velocity. The inertial I is assumed constant. m and g denote the mass and gravity vector. $f_i(t)$ and $p_i(t)$ are the force and position vectors of the end effector i. For the kinematics restriction, $\bar{p}_i$ is a constant reference of the relative foot position and b denotes the box size.

The trajectory optimization is formulated with an optimal control direct approach (using collocation) where both state and controls are optimization variables. The optimization variables are the COM linear and angular position, the contact forces, the feet position and the time duration of phase of each foot $\Delta T_{i,j}$. There is no cost function, thus the optimization solver computes only a feasible motion.

Regarding the contact formulation: the number of phases $2m_i$ for foot i is chosen before the optimization. Each foot, denoted with index i, has the following related variables:

$$\begin{aligned} &\Delta T_{i,1}, ..., \Delta T_{i,2m_i} \geq 0 \\ &p_i(t, \Delta T_{i,1}, ...) \in \mathbb{R}^3 \\ &f_i(t, \Delta T_{i,1}, ...) \in \mathbb{R}^3 \\ &\Delta T_{i,j} \in \mathcal{C}_\rangle \quad j = 1, 3, .. \quad \text{contact phase} \\ &\Delta T_{i,j} \notin \mathcal{C}_\rangle \quad j = 2, 4, ... \quad \text{flying phase} \end{aligned} \tag{1.24}$$

Constraints are specialized for either flying or contact phase, avoiding the use of complementarity constraints. Slippage is not allowed by the formulation and the tangential forces are restricted to be inside a linear approximation of the friction cone.

$$
\begin{aligned}
\sum_j \Delta T_{i,j} &= T \\
\text{foot in contact} \quad & (t \in \mathcal{C}_i) \\
\dot{p}_i(t \in \mathcal{C}_i) &= 0 && \text{no slip} \\
p_i^z(t \in \mathcal{C}_i) &= h_{terrain}(p_i^{xy}) && \text{terrain height} \\
f_i(t \in \mathcal{C}_i) \cdot n(p_i^{xy}) &\geq 0 && \text{pushing normal force} \\
f_i(t \in \mathcal{C}_i) &\in \mathcal{F}(\mu) && \text{linear approximation of friction cone} \\
\text{foot in air} \quad & (t \notin \mathcal{C}_i) \\
f_i(t \notin \mathcal{C}_i) &= 0
\end{aligned}
\tag{1.25}
$$

where $h_{terrain}(p_i^{xy})$ is the terrain height at position $(x, y) = p_i^{xy}$, and $n(p_i^{xy})$ its normal vector. $\dot{p}_i(t)$ denotes the foot velocity and $p_i^z(t)$ the foot height. $f_i(t)$ and $p_i(t)$ denote force and foot position. μ denotes the friction coefficient.

COM, feet and forces are parametrized with piecewise polynomials (third degree polynomials or constant functions). In the flight phase, forces are set to zero and feet position is parametrized with 3 third-degree polynomial (piecewise). In contact phase, the position is constant and the force is now parameterized equivalently. Smooth continuity on forces and feet positions between phases is imposed as a restriction.

Motions are generated for several legged robots in simulation: bipeds, single leg robots and quadrupeds (tested also partially with the real robot). The optimization problem is solved with the Interior Point Solver IPOPT. Computational time depends on the task and terrain complexity, taking 300 ms to compute two steps of a quadruped or 20 seconds for highly dynamical 5 seconds of motion. Once the trajectory for the COM and end effectors is optimized, joint values are computed with inverse kinematics.

Chapter 2

Numerical optimization

2.1 The role of numerical optimization in robotics

We start the second chapter with an introduction to the theory of constraint optimization: problem formulation, basic definitions and theorems that characterize the optimal solution. This theoretic results are the basis of the numerical optimization algorithms that will be presented later.

The three discussed algorithms are interior points (IP), sequential quadratic programming (SQP) and augmented Lagrangian (AL). SQP has been traditionally the most common choice in robotics, whereas interior points are becoming popular nowadays.

However, in general, the focus on robotics research has usually been on proposing new formulations and defining the problems in an original and ingenious way. Little or no effort is dedicated to the interaction between the chosen problem definition and the optimization method. For example, a standard practice is to use off-the-shelf implementations, testing maybe some of them, and selecting the one that experimentally works better in each case.

Therefore, the choice of the optimization algorithm depends more on software considerations: open source codes, programming language and robust implementation than on the underlying differences between methods.

The third numerical optimization algorithm is called augmented Lagrangian, and has still not gained much attention of the robotics researchers [29]. This will be our choice for simultaneously optimizing trajectories with contacts. This decision is justified in Section 2.3 and 4.1.3.

We believe that understanding numerical optimization algorithms is essential: not only to choose one of them, but to formulate the problem in a way that fits the chosen algorithm. Therefore, in this chapter we present the three algorithms. The goal of this introduction is to give a general understanding and intuition of the underlying principles. We refer to [23] for a complete discussion.

2.2 Constrained optimization

Optimization problems in robotics are described with a cost function $f(x)$ and a set of constraints $\{c_i(x)\}$. This is called constrained optimization, and the goal is to find the value of the variables $x \in \mathbb{R}^n$ that fulfils the constraints with a minimum cost.

The problem is formulated as:

$$\begin{gathered} x \in \mathbb{R}^n \quad \text{variables} \\ f : \mathbb{R}^n \to \mathbb{R} \quad \text{cost or objective} \\ c_i : \mathbb{R}^n \to \mathbb{R} \quad \forall i = 1, ..., m \quad \text{constraints} \\ \mathcal{I} \text{ (inequality)}, \ \mathcal{E} \text{ (equality)} : \ \text{finite sets of indices} \\ i \in \mathcal{I} \text{ or } i \in \mathcal{E} \quad \forall i = 1, ..., m \end{gathered}$$

$$\begin{gathered} \min_x f(x) \\ \text{subject to:} \quad c_i(x) = 0, \ \ i \in \mathcal{E}; \quad c_i(x) \geq 0, \ \ i \in \mathcal{I} \end{gathered} \tag{2.1}$$

Before stating the first order necessary condition for x^* to be a minimizer of (2.1), we define the following concepts:

- Lagrangian function and multipliers
  Each constrained optimization problem has an associated Lagrangian function $\mathcal{L}(x, \lambda)$ that plays an important role on the theory and implementation of constrained optimization algorithms. The variables $\lambda_i \in \mathbb{R}$, $i = 1, ..., m$ (where m is the number of constraints) are called the Lagrange multipliers.

$$\mathcal{L}(x, \lambda) = f(x) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(x) \tag{2.2}$$

- Active set
  The active set $\mathcal{A}(x)$ at any feasible x consists of the equality constraint indices from $\mathcal{E}$ together with the indices of the inequality constraints $i \in \mathcal{I}$ for which $c_i(x) = 0$.

$$\mathcal{A}(x) = \mathcal{E} \cup \{i \in \mathcal{I} \mid c_i(x) = 0\} \tag{2.3}$$

- Feasible set.
  Set of all points that satisfy the constraints.

$$\Omega = \{x \mid \quad c_i(x) = 0, \ \ i \in \mathcal{E}; \quad c_i(x) \geq 0, \ \ i \in \mathcal{I}\} \tag{2.4}$$

  Given a feasible point x and the active constraint set $\mathcal{A}(x)$, the set of linearized feasible directions $\mathcal{F}(x)$ is:

$$\begin{aligned} \mathcal{F}(x) = \{d \in \mathbb{R}^n \mid d^T \nabla c_i(x) = 0, \quad & i \in \mathcal{E}; \\ \text{and } \ d^T \nabla c_i(x) \geq 0, \quad & i \in \mathcal{A}(x) \cap \mathcal{I}\} \end{aligned} \tag{2.5}$$

- Linear independence constraint qualification (LICQ)
  Given the point x and the active set $\mathcal{A}(x)$ we say that the linear independence constraint qualification holds if the set of active constraint gradients $\{\nabla c_i(x),\ i \in \mathcal{A}(x)\}$ is linearly independent

The First-Order Necessary Conditions Theorem establishes a relation on the gradients of the objective and constraint functions at the constrained optimum. These conditions are the foundation for many numeric algorithms. These algorithms are usually based on finding the variable x, and multipliers λ that fulfil the first order necessary conditions of optimality.

First-Order Necessary Conditions Theorem (KKT Conditions): . Suppose that $x^* \in \mathbb{R}^n$ is a local solution of (2.1), that the cost and constraints functions are continuously differentiable, and that the LICQ holds at x. Then there exists a Lagrange multiplier vector $\lambda^* \in \mathbb{R}^m$ such that:

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla f(x^*) - \sum_i \lambda_i^* \nabla c_i(x^*) = 0 \tag{2.6}$$

$$c_i(x^*) = 0, \quad \text{for all} \quad i \in \mathcal{E} \tag{2.7}$$

$$c_i(x^*) \geq 0, \quad \text{for all} \quad i \in \mathcal{I} \tag{2.8}$$

$$\lambda_i^* \geq 0, \quad \text{for all} \quad i \in \mathcal{I} \tag{2.9}$$

$$\lambda_i^* c_i(x^*) = 0, \quad \text{for all} \quad i \in \mathcal{E} \cup \mathcal{I} \tag{2.10}$$

These conditions are usually called the KKT Conditions (Karush–Kuhn–Tucker).

Points that fulfil KKT are candidates for local minimizers. Second order necessary conditions can be then applied to points that already fulfil KKT conditions. This new test consists on checking the curvature of the Lagrangian function along the directions $w \in \mathbb{R}^n$ that belong to the linearized feasible set $\mathcal{F}$.

A formal definition can be found in [23]. As an intuition and similar to unconstrained minimization, second order information can distinguish apart points that already fulfil KKT between locally constrained maximum, minimum and stationary points.

Second order conditions are important for convergence analysis and to assure that the found solution is actually a local minimizer. However, numerical optimization algorithms are based mainly on finding points that fulfil KKT conditions.

2.2.1 Augmented Lagrangian

Overview: the augmented Lagrangian algorithm solves the constrained minimization (2.1) by solving a sequence of unconstrained minimizations of the augmented Lagrangian function $\mathcal{L}_A(x, \lambda; \mu)$ and updating the estimate of the Lagrange multipliers λ.

$$\mathcal{L}_A(x, \lambda; \mu) = f(x) - \sum \lambda_i c_i(x) + \frac{\mu}{2} \sum c_i^2(x) \tag{2.11}$$

As its name suggests, (2.11) corresponds to the standard Lagrangian with the addition of the constraints with a squared penalty. In fact, the augmented Lagrangian is closely related to squared penalty methods.

Squared penalty methods solve the original constrained minimization (2.1) by solving a sequence of unconstrained problems Q_k, where the constraint violation is added to the objective function with a quadratic penalty function. The initial guess for a subproblem Q_k is the solution of last subproblem, and the penalty parameter is always increased.

$$\begin{gathered} Q_k(x; \mu_k) = f(x) + \frac{\mu_k}{2} \sum c_i(x) \\ \text{solve} \quad \min_x Q_k \quad \text{for sequence } \{\mu_k\} \to \infty \end{gathered} \tag{2.12}$$

The main disadvantage of square penalty methods is the ill conditioning (almost singular Hessian matrix of $Q_k(x, \mu_k)$) induced by the sequence $\{\mu_k\} \to \infty$. To reduce this problem, the augmented Lagrangian formulation introduces an estimate of the Lagrange multiplier $\lambda \in \mathbb{R}^m$. Another advantage is that it can be warmstarted more efficiently using both a guess of primal, x, and dual variables, λ.

Augmented Lagrangian for equality constraint optimization: Let's assume now that the optimization problem has only equality constraints $c_i(x) = 0 \; \forall i$ (later we will show how to generalize this formulation to handle inequalities). The augmented Lagrangian algorithm consists of iteratively alternating two steps while increasing the penalty parameter : an approximate minimization of the augmented Lagrangian with respect to the primal variables (2.13) and an update on the multipliers estimate (2.14). This algorithm generates a sequence of (x^k, λ^k) that converges to the optimum (x^*, λ^*) of the constrained problem (2.1). The framework is specified in Algorithm 1 [23].

$$0 \approx \nabla \mathcal{L}_A(x^k, \lambda; \mu) = \nabla f(x) - \sum_{i=1}^{m} [\lambda_i - \mu c_i(x)] \nabla c_i(x) \tag{2.13}$$

$$\lambda_i^{k+1} = \lambda_i^k - \mu c_i(x) \tag{2.14}$$

The update on the multipliers is deduced by comparing (2.13) with the optimality conditions (2.15) (2.6) of the original constrained problem (2.1).

$$\nabla f(x) - \sum \lambda_i \nabla c_i(x) = 0 \tag{2.15}$$

Algorithm 1 Augmented Lagrangian Framework

Given μ_0 , tolerance $\tau_0 > 0$ and x_0^s and λ_0
for $k = 0, 1, 2, ...$ **do**
  Find x_k minimizer of $\mathcal{L}_A(\cdot, \lambda_k, \mu_k)$ terminating when $\|\nabla \mathcal{L}_A(\cdot, \lambda_k, \mu_k)\| \leq \tau_k$
  if Convergence (2.6) on primal and dual feasibility of constrained problem (2.1) **then**
    Terminate with solution x_k and λ_k
  end if
  Update Lagrange multipliers with $\lambda_{k+1}^i = \lambda_{k+1}^i - \mu_k c^i(x_k)$
  Choose new penalty parameter $\mu_{k+1} \geq \mu_k$
  Set starting point for the next iteration $x_{k+1}^s = x_k$
  Select tolerance τ_{k+1}
end for

Regarding the sequence $\{\mu_k\}$, feasibility can be achieved even for reasonably small values of μ. The relation between penalty and infeasibility is better than in classical penalty methods, specially when λ_i^k is close to the optimal λ_i^*. [23]

Bound Constraint Formulation and inequalities. The standard approach to handle inequality constraints $c_i(x) \geq 0, \quad i \in \mathcal{I}$ is by introducing slack variables s_i. These new variables are added to the optimization variables, and the inequalities are converted into equality's plus bounds.

$$c_i(x) \geq 0 \rightarrow c_i(x) - s_i = 0, \;\; s_i \geq 0, \quad \forall\, i \in \mathcal{I} \tag{2.16}$$

Coming back to the alternating algorithmic framework, the bound constraints are imposed in the first step: the minimization of the augmented Lagrangian with respect to the primal variables. It becomes now a bound constrained minimization (instead of unconstrained).

$$\min_x \mathcal{L}_A(x; \lambda, \mu) \quad \text{s.t.} \quad l \leq x \leq u \tag{2.17}$$

where x is now a concatenation of original and slack variables, and l,u the lower and upper bounds. The update of the multiplier step remains as in the "only-equality" case. There is no multiplier for bound constraints, as they are applied directly in the inner solver.

$$\lambda_i^{k+1} = \lambda_i^k - \mu c_i(x) \tag{2.18}$$

Bound constrained optimization can be solved efficiently (although not as fast as unconstrained minimization). A popular technique is the gradient or Newton projection method, an efficient alternative to general active set approaches (Section 2.2.3). It is based on projected line searches that bend around the box (projecting a vector to a box is a very fast operation!). Standard active set approaches are slower, as the constraints (in this case the bounds) get in an out the active set (set of constraints active in a given iteration) one by one.

In a practical implementation, the update rule for the penalty parameter is also important to get good performance. Convergence analysis shows that the sequence $\{\mu_k\}$ must be non decreasing $\mu_k \leq \mu_{k+1}$. However, the best rule on when and how to increase the penalty is still unclear and very problem dependent. During the optimization, primal (constraint violation) and dual (gradient of Lagrangian) errors are monitored. The parameter μ_k tries to get a good balance between both objectives.

In Algorithm 2, we describe a standard bound constrained augmented Lagrangian algorithm , as presented in [23]. This is the algorithm implemented in LANCELOT [11], a popular commercial code. $P(\cdot)$ denotes an orthogonal projection of a vector into the box defined by the bound constraints.

Algorithm 2 Practical Augmented Lagrangian Algorithm

Choose initial x_0 and λ_0
Choose convergence tolerances η_* and ω_*
Set $\mu_0 = 10$, $\omega_0 = 1/\mu_0$ and $\eta_0 = 1/\mu_0^{0.1}$
for $k = 0, 1, 2, \ldots$ **do**
  Find x_k minimizer of subproblem (2.17) such that
    $\left\| x_k - P(x_k - \nabla_x \mathcal{L}_A(x_k, \lambda^k; \mu_k), l, u) \right\| \leq \omega_k$
  if $\|c(x_k)\| \leq \eta_k$ **then**
    (*test primal dual convergence*)
    if $\|c(x_k)\| \leq \eta_*$ and $\left\| x_k - P(x_k - \nabla_x \mathcal{L}_A(x_k, \lambda^k; \mu_k), l, u) \right\| \leq \omega_*$ **then**
      stop with x_k and λ_k
    end if
    (*update multipliers*)
    $\lambda_{k+1} = \lambda_k - \mu_k c(x_k)$
    $\mu_{k+1} = \mu_k$
    $\eta_{k+1} = \eta_k / \mu_{k+1}^{0.9}$
    $\omega_{k+1} = \omega_k / \mu_{k+1}$
  else
    (*increase penalty parameter*)
    $\lambda_{k+1} = \lambda_k$
    $\mu_{k+1} = 10\mu_k$
    $\eta_{k+1} = 1/\mu_{k+1}^{0.1}$
    $\omega_{k+1} = 1/\mu_{k+1}$
  end if
end for

2.2.2 Interior points methods

Interior points methods can be viewed as continuation (on a disturbed KKT system (2.6)) or barrier methods. Here, the barrier approach derivation is presented. The barrier problem associated to (2.1) is:

$$\begin{aligned} c_E(x) &= [c_i(x) : i \in \mathcal{E}] \\ c_I(x) &= [c_i(x) : i \in \mathcal{I}] \\ \min_{x,s} & f(x) - \mu \sum_{i=1}^{m} \log s_i \\ \text{s.t.} \quad & c_E(x) = 0 \\ & c_I(x) - s = 0 \end{aligned} \tag{2.19}$$

where $\mu \geq 0$ is the barrier parameter and $\log(\cdot)$ the natural logarithm. The equation $s \geq 0$ appears implicitly, as the minimization of the barrier term prevents that the components of s become too close to zero. To find a solution of original constrained problem (2.1), the barrier problem is solved for a sequence of μ_k that converges to zero.

The KKT conditions of 2.19 are as follows:

$$\begin{aligned} \nabla f(x) - A_E^T(x)y - A_I^T(x)z &= 0 \\ Sz - \mu e &= 0 \\ c_E(x) &= 0 \\ c_I(x) - s &= 0 \end{aligned} \tag{2.20}$$

where $A_E(x)$ and $A_I(x)$ are, respectively, the Jacobian matrices of the vector functions $c_E(x)$ and $c_I(x)$ and y, z their Lagrange multipliers. S is a diagonal matrix with $S_{ii} = s_i$ and $e = [1, 1, ...]^T$.

The second equation is obtained by multiplying by S the equation (2.21) arising from $\nabla_s \mathcal{L}(x, s, y, z; \mu) = 0$ (so that it becomes quadratic instead of rational). This KKT conditions of the barrier problem are known as the disturbed KKT conditions of the original problem.

$$-\mu S^{-1} e + z = 0 \tag{2.21}$$

For fixed μ, and starting from an initial guess (x_0, s_0, y_0, z_0) Newton's method is applied to the nonlinear system (2.20) to compute a primal-dual step.

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} & 0 & -A_E^T(x) & -A_I^T(x) \\ 0 & Z & 0 & S \\ A_E(x) & 0 & 0 & 0 \\ A_I(x) & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_s \\ p_y \\ p_z \end{bmatrix} = - \begin{bmatrix} \nabla f - A_E^T(x)y - A_I^T(x)z \\ Sz - \mu e \\ c_E(x) \\ c_I(x) - s \end{bmatrix} \tag{2.22}$$

$$\mathcal{L}(x, s, y, z) = f(x) - y^T c_E(x) - z^T(c_I(x) - s) \tag{2.23}$$

The new iterate is computed with:

$$\begin{aligned} x^+ &= x + \alpha_s^{\max} p_x \qquad y^+ = y + \alpha_z^{\max} p_y \\ s^+ &= s + \alpha_s^{\max} p_s \qquad z^+ = z + \alpha_z^{\max} p_z \end{aligned} \tag{2.24}$$

The step size ($\alpha_s^{\max}, \alpha_z^{\max}$) is controlled to prevent that the variables s and z approach 0 to quickly.

$$\begin{aligned} \alpha_s^{\max} &= \max\{\alpha \in (0,1] : s + \alpha p_s \geq (1-\tau)s\} \\ \alpha_z^{\max} &= \max\{\alpha \in (0,1] : z + \alpha p_z \geq (1-\tau)z\} \end{aligned} \tag{2.25}$$

with $\tau \in (0,1)$ (close to 1)

The primal-dual matrix 2.22 remains non singular as the iterations converge to a solution that satisfies second order sufficient conditions and strict complementarity (a constraint and its associated multiplier are not both zero) . The rate of convergence is superlinear.

Based on the disturbed KKT conditions 2.20, the following error function is defined.

$$E(x,s,y,z,\mu) = \max\left\{\left\|\nabla f - A_E(x)^T y - A_I(x)^T z\right\|, \|Sz - \mu e\|, \|c_E(x)\|, \|c_I(x) - s\|\right\}$$

(2.26)

A basic interior point algorithm consists on computing iteratively a primal-dual step with (2.22) and decreasing the barrier parameter. It has of two loops: the inner loop solves the disturbed KKT conditions for a fixed value of μ, that is decreased in the outter loop. This framework is shown in Algorithm 3.

Algorithm 3 Basic Interior-Point Algorithm

Choose initial x_0 and $s_0 \geq 0$, compute initial values for y_0 and $z_0 \geq 0$
Choose $\mu_0 \geq 0$ and parameters $\sigma, \tau \in (0,1)$
repeat
  repeat
    Solve 2.22 to obtain a search direction $p = (p_x, p_s, p_y, p_z)$
    Compute $\alpha_s^{\max}, \alpha_z^{\max}$ using 2.25
    Compute $(x_{k+1}, s_{k+1}, y_{k+1}, z_{k+1})$
  until $E(x_k, s_k, y_k, z_k; u_k) \leq \mu_k$
  Choose $\mu_k \in (0, \sigma\mu_k)$
until stopping test for (2.1) (Original Problem)

This simple algorithm provides the basis of modern interior point methods, although various modifications are needed to work with non convexity and non linearity. Also, the sequence of penalty parameters can be chosen in different ways.

IPOPT

One of the most successful code implementations of an interior point algorithm is IPOPT, a primal-dual interior-point algorithm with a filter line-search [30].

Recently, this code has become popular in robotics and is extensively used for solving constrained non linear programming in different applications. The main difference with respect to the presented basic algorithm (Algorithm 3) is the addition of a filter line-search to compute the length of the steps α_s and α_z.

$$\alpha_s \in (0, \alpha_s^{\max}] \quad \alpha_z \in (0, \alpha_z^{\max}] \tag{2.27}$$

where $\alpha_s^{\max}, \alpha_z^{\max}$ are the computed step length in the basic formulation in (2.25).

Filter methods consists on formulating the step length selection as a bi-objective optimization with two goals: minimizing the objective function $\varphi(x) = f(x) - \mu \log(s)$ and the constraint violation $\theta(x) := \|c(x)\|$. Trial points $x^+ = x + \alpha p_x$ are tested in a backtracking line search and are accepted if they lead to sufficient decrease of bi-objective. Along the whole optimization process, the algorithm maintains a "filter" that gets dynamically updated and contains those combinations of constraint violation values θ and objective function values φ, that are "prohibited" for a successful trial point.

The details on how the filter is updated and the step size is selected are carefully explained in the original paper [30].

The implementation includes algorithmic features such as second-order corrections, an efficient and robust feasibility restoration phase and inertia correction. Also, special heuristics are added to allow faster performance. Overall IPOPT is considered one of the best non linear optimizers for general use, and is often used by the research community as a black-box tool.

2.2.3 Sequential quadratic programming

The sequential quadratic programming algorithm generate steps by solving a sequence of quadratic subproblems. At each iteration, the original problem (2.1) is approximated by a quadratic program (QP, an optimization problem with quadratic cost and linear constraints). Starting from the current iterate (x_k, λ_k) , the new step p is generated with:

$$\begin{aligned} \min_{p} \quad & f_k + \nabla f_k^T p + \frac{1}{2} p^T \nabla^2_{xx} \mathcal{L}_k p \\ \text{s.t} \quad & \nabla c_{ik}^T p + c_{ik} = 0, \quad i \in \mathcal{E} \\ & \nabla c_{ik}^T p + c_{ik} \geq 0, \quad i \in \mathcal{I} \end{aligned} \tag{2.28}$$

where the subindex k means that the corresponding matrix, vector and scalar functions are evaluated at (x_k, λ_k) and remain constant inside the subproblem.

The next iterate is then $(x_k + p_k, \lambda_{k+1})$ where p_k and λ_{k+1} are the solution and multipliers of (2.28).

Therefore, quadratic programming (solving a QP) is the core unit of the SQP algorithm. We provide a (very) short introduction. We start with an equality constrained QP and then extend the formulation to handle inequalities using an active set algorithm.

Quadratic Program. An equality constrained QP program is stated as :

$$\begin{aligned} \min_{x} \quad & \tfrac{1}{2} x^T G x + x^T c \\ \text{s.t} \quad & Ax = b \end{aligned} \tag{2.29}$$

First order necessary conditions of optimality (KKT) (2.6) , state that the solution (x^*, λ^*) of (2.2.3) has to be solution of:

$$\begin{bmatrix} G & -A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -c \\ b \end{bmatrix} \tag{2.30}$$

Under some assumptions (A has full row rank and the Hessian matrix G is definite positive on the span generated by the null space of A) the system (2.30) has only one solution that corresponds to the global optimum (x^*, λ^*). Therefore, solving the linear system provides the solution of an equality constrained QP. Note that, in practice, either the system (2.30) is written in an equivalent form that is most useful for computations or it is solved iteratively.

Active set. To deal with inequalities, an active set method is used. The main challenge is that we do not know a priori which constraints are active at the optimum. The active set method is an iterative process: at each step an equality-QP is solved using as constraints all the equality constraints and some of the inequality constraints (the active ones). This subset of constraints is called the working (active) set. Once a step is generated by the solution of the equality-QP, it is checked whether a new inequality must enter or leave the active set:

- If the step violates an inequality that was not active, it is shortened to remain feasible and the blocking constraint becomes active in the next iteration.
- If an inequality that already belongs to the active set is preventing a further minimization of the objective, it becomes inactive.

It can be shown that active set method finds the solution in a finite number of steps.

Back to SQP If the SQP method identifies the unknown optimal active set (and does not change its guess at following iterations), then it acts like a Newton method for equality-constrained optimization and converges rapidly (it is equivalent to applying the Newton Method on the KKT conditions).

Solving a sequence of QP subproblems is the basis of this approach. However, a robust an efficient implementation depends also on features to handle inconsistent linearizations, step-acceptance mechanisms and appropriate quasi Newton approximations.

SQP methods often use a merit function to decide the length of the step p computed from the QP subproblem (2.28). That is, deciding the α for generating the next iterate: $x_{k+1} = x_k + \alpha p$. The merit function is a weighted sum of the objective value and the constraint violation. For example, the l_1 merit function $\phi_1(x;\mu)$ with penalty parameter μ. For simplicity, we assume now that all constraints are in the form of equalities. Inequalities can be converted to equalities by adding slack variables $s \geq 0$, and this slack can be considered outside the merit function.

$$\phi_1(x;\mu) = f(x) + \mu \left\| c(x) \right\|_1 \tag{2.31}$$

The length of α is selected so that the following sufficient decrease condition holds.

$$\phi_1(x_k + \alpha p;\mu) \leq \phi(x_k;\mu_k) + \ \eta\alpha D(\phi_1(x_k;\mu);p_k) \quad \eta \in (0,1) \tag{2.32}$$

$D(\phi_1(x_k;\mu);p_k)$ denotes the directional derivative of ϕ_1 in the direction p_k. This requirement is analogous to Armijo conditions for unconstrained optimization. One strategy to choose μ is to consider the effect of the step on a (quadratic) model of the merit function, see [23].

SLSQP

Sequential least squares quadratic programming [20] is a minor variation of the original SQP. It is the algorithm implemented in the optimization module of the Python library Scipy, the most popular Python package for numerical computations. Therefore, the algorithm is usually used as a "first-try" solver for optimizing constrained problems in robotics and science.

In SLSQP, the quadratic subproblem (2.28) is replaced by a linear least squares, so that a dedicated solver for least squares optimization can be used.

$$\begin{aligned} \min_p \ & \left\| D^{1/2} L^T p + D^{-1/2} L^{-1} \nabla f_k \right\| \\ \text{s.t} \ & \nabla c_{ik}^T p + c_{ik} = 0 \quad i \in \mathcal{E} \\ & \nabla c_{ik}^T p + c_{ik} \geq 0 \quad i \in \mathcal{I} \end{aligned} \tag{2.33}$$

where LDL^T is a factorization of the matrix $\nabla_{xx}\mathcal{L}_k$ (D and L are, respectively, diagonal and lower triangular matrices) In Scipy $\nabla_{xx}\mathcal{L}_k$ is approximated with a quasi Newton method.

2.3 Strengths of the augmented Lagrangian algorithm

This short introduction to interior points (IP), sequential quadratic programming (SQP) and augmented Lagrangian (AL) already shows some of the big underlying differences. Each solver deals with constraints in a different way: either with log barriers (IP), quadratic penalty (AUG) or linearization and active set (SQP). Moreover, they are based on solving a different sequence of subproblems: perturbed KKT (IP), bound constrained optimization (AL), and QP (SQP).

From this quick and meaningful comparison, the main strengths of the augmented Lagrangian can be highlighted:

- *No matrix factorization*. The bound constraint subproblems are (normally) convex and conjugate gradients methods are used to compute Newton search directions without matrix factorization.
- *Simplicity*. Augmented Lagrangian is an algorithmic framework on top of a standard bound constrained solver.
- *Quadratic penalty on constraints*. The quadratic penalty acts as a natural regularization and relaxation of constraints in the first iterations. This behaviour is ideal for optimizing degenerated problems, such as problems with complementarity constraints (where constraints do not satisfy standard constraint qualifications).
- *Warmstart*. They can be warmstarted with a primal-dual guess, both from feasible and infeasible regions. Warmstart means using the optimal solution of a previously solved similar problem, with the goal of reducing the number of iterations to reach the optimum.

In our formulation of simultaneous trajectory and contact optimization (Chapter 4), we take advantage of these four properties. Specifically, we leverage its robustness and performance dealing with complementarity constraints (Section 4.1.3). Using this new type of constraints, contacts are formulated in a continuous manner and can be optimized together with the trajectory.

Chapter 3

Benchmarking numerical optimization solvers in robotics

The three optimization algorithms presented in Chapter 2: interior points, sequential quadratic programming and augmented Lagrangian; are now evaluated in a classical robotics problem: low-dimensional posture generation. These problems are called inverse geometry or inverse kinematics, and consist in finding a robot configuration (static, not a trajectory) that fulfils a given task in an optimal way. It is the basic unit for high dimensional problems of trajectory optimization and optimal control, that can be understood as a simultaneous optimization of a coupled sequence of postures.

The augmented Lagrangian algorithm is analysed in detail, with a focus on the inner bound constrained solver, the penalty update strategy and the warmstarst behaviour.

3.1 Key challenges in optimization for robotics

Inverse geometry and trajectory optimization in robotics present the following challenges:

- Non convex objective function

  Robots are usually build of a kinematic chain: a tree of rigid bodies attached by joints. Therefore position and orientation of the links are defined by products and additions of trigonometric functions. Starting from a distant initial guess, the objective function is not always convex. The algorithm must be able to optimize over concave regions, where the Newton direction is not a descending direction. Although these algorithms compute local optimum, some degree of globalization is always desired. Line searches or trust region features are required.

- Constrained

  Non linear constraints appear always in robotics. We have presented in Chapter 2 the most popular constrained optimization algorithms.

- Rotations $SO(3)$ and functions defined on manifolds

  Modelling free-base robots (that is, robots that do not have any fixed point, for example legged robots) is challenging because the orientation of the base is represented with a rotation matrix $R \in \mathbb{R}^{3\times3}$ s.t. $RR^T = Id$. The space of rotation matrices is not Euclidean. For example, the addition of two rotation matrices is not a rotation matrix. Unfortunately, most solvers implementations work only in Euclidean space and non linear parametrizations (for example Euler angles) of rotation matrices are

employed. However, when rotations plays and important role [7], dedicated solvers must be built. This also highlights the importance of having numerical methods that can be understood and easily implemented, without relying on black box codes.

- Degenerated problems

  Problem in robotics are sometimes degenerated. For example, it might exist an infinite number of configurations that reach a posture. In this cases, numerical algorithms should converge to any given solution (without iterating forever), displaying the status of the solution. Also, constraints might become linearly dependent at the solution. This break one classical assumption on constrained numerical optimization: the linear independence of the constraints gradients at the optimum.

- Lack of benchmarks

  In the robotics community there are not public available benchmarks, where researches can test and compare their new algorithms. Usually research groups test a new algorithm in their own problems, sometimes without releasing the code or the problems. This differs from other fields, such as computer vision and Machine Learning, where new results are now compared to existing methods.

  Doing benchmarks with real robots is impossible, but it is necessary to build a benchmark in a simulated environment. This benchmark should allow for both new formulations and resolutions. For example, in the numerical optimization community they use Cuter/Cutest testing environment [17]. Without defined benchmarks is sometimes difficult to assess which are the best algorithms and ideas in the state of the art.

3.2 Inverse geometry problems

Inverse geometry (inverse kinematics) consists in finding a robot configuration that fulfils a given task in an optimal way. A typical example is to find the configuration of a robot manipulator such that the end effector of the robot is at a given position and orientation. For legged robots, a usual objective is to generate a stable posture while fulfilling a given task with one of its limbs.

These problems cannot, in general, be solved analytically, and only numeric solutions are available. The cost function is not necessarily convex. Also the feasible set is not always convex. Gradient-based numerical optimization algorithms (presented in Chapter 2 and evaluated here) can be used to find local optimum. This local optimum is a point that fulfills the necessary first and second order conditions of optimality (2.6). For finding global optimum, it is necessary to use other optimization algorithms based on exploration and sampling (such as rapidly-exploring random trees).

Finding local optimum is not, in practice, a big limitation, as normally we can start the optimization from a reasonable initial guess. The algorithm will then find the best configuration around this guess. Also, both the feasible set and the cost are usually convex around the local optimum.

The dimension of the problem (number of variables) is small (7 for a fixed based manipulator arm, around 30 for simple humanoid model).

3.2.1 Problem definition

We define two classical inverse geometry problems using two different robot models: a robot arm and a quadruped robot. In both cases, a placement task and joint limits are set as constraints and the objective function is a regularization on the configuration.

Robot arm
The robot arm is a KUKA LWR4, a redundant manipulator with 7 degrees of freedom. The base is fixed and the task is to position the end effector at a given position (constraint), while remaining close to a predefined configuration (objective). The function $p(q) : \mathbb{R}^7 \rightarrow \mathbb{R}^3$ maps joints values to end effector position and represents the kinematic chain of the arm.

$$\begin{aligned} \text{joint values} \quad & q \in \mathbb{R}^7 \\ \text{reference joints} \quad & q_{ref} \in \mathbb{R}^7 \\ \text{end effector goal position} \quad & p^* \in \mathbb{R}^3 \\ \text{end effector function} \quad & p(q) : \mathbb{R}^7 \rightarrow \mathbb{R}^3 \\ \text{joint limits} \quad & l, u \in \mathbb{R}^7 \end{aligned}$$

$$\begin{aligned} \min_q \quad & \left\| q - q_{ref} \right\|^2 \\ \text{subject to} \quad & p(q) = p^* \\ & l < q < u \end{aligned} \tag{3.1}$$

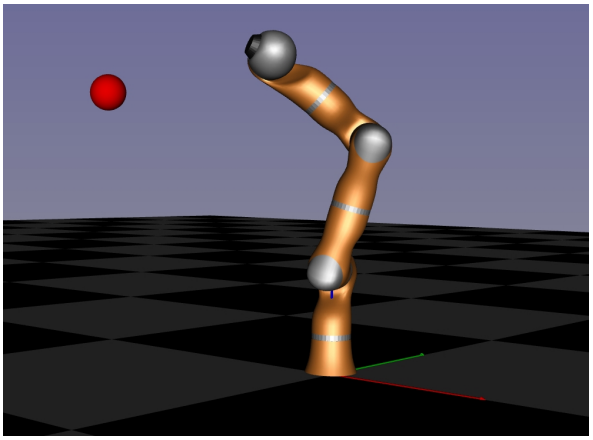

FIGURE 3.1: Inverse geometry problem with a robot arm. Displaying a random initial configuration. The red ball indicates the end effector goal position (formulated as a constraint).

Quadruped
The quadruped model is called Hyq and has 12 joints (3 in each leg). The translation of the free-floating base is also optimized. This translation defines the position of a predefined point of the robot, for example the geometric center of its main body. The task is to reach a point with one feet, while keeping the other three on the ground. One of the feet in the ground is completely fix, as a reference.

$$
\begin{aligned}
\text{floating base translation} \quad & x \in \mathbb{R}^3 \\
\text{joint values} \quad & q \in \mathbb{R}^{12} \\
\text{variables (joints and base)} \quad & y = [\, x \,,\, q \,] \in \mathbb{R}^{15} \\
\text{feet position function} \quad & p_i(y) : \mathbb{R}^{15} \to \mathbb{R}^3 \quad i = 0,1,2,3 \\
\text{feet height} \quad & z_i(y) : \mathbb{R}^{15} \to \mathbb{R} \quad i = 0,1,2,3 \\
\text{reference joints and base} \quad & y_{ref} \in \mathbb{R}^{15} \\
\text{Foot 0 goal position} \quad & p^* \in \mathbb{R}^3 \\
\text{Foot 3 reference (z=0)} \quad & p_{ref} \in \mathbb{R}^3 \\
\text{joint limits} \quad & l, u \in \mathbb{R}^{12}
\end{aligned}
$$

$$
\begin{aligned}
\min_y \quad & \left\| y - y_{ref} \right\|^2 \\
\text{subject to} \quad & p_0(y) = p^* \\
& z_i(y) = 0 \quad i = 1,2 \\
& p_3(y) = p_{ref} \text{ (this includes } z_3 = 0) \\
& l < q < u
\end{aligned}
\tag{3.2}
$$

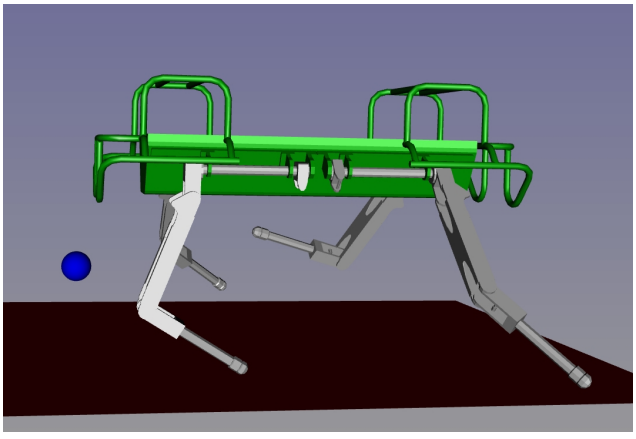

FIGURE 3.2: Inverse geometry problem with a quadruped. Displaying a random initial configuration. The blue ball indicates the goal position of foot 0 (formulated as a constraint). The 3 other feet should lie on the ground (dark plane).

3.2.2 Results

The optimization algorithms are tested using the following implementations:

- Augmented Lagrangian: we use the code of NLP.py [4] [5], a Python environment for large-scale optimization. This package is still in development, and not yet a robust production code. Getting a good performance has required to go into the source code and do minor modification on the penalty update strategy and convergence criteria of the different iterative processes.
- Interior points: we use the code Ipopt [30] (state of the art), executed from Python with the wrapper package Pyipopt.
- Sequential quadratic programming: we use the algorithm SLSQP [20] (least squares version of SQP, implemented in Fortran) from the package Scipy.

Random initial guess

Non linear numerical optimization algorithms are iterative processes and need an initial guess (not necessarily good) of the solution. We first report the results starting from different random configurations (Figure 3.3, 3.4)

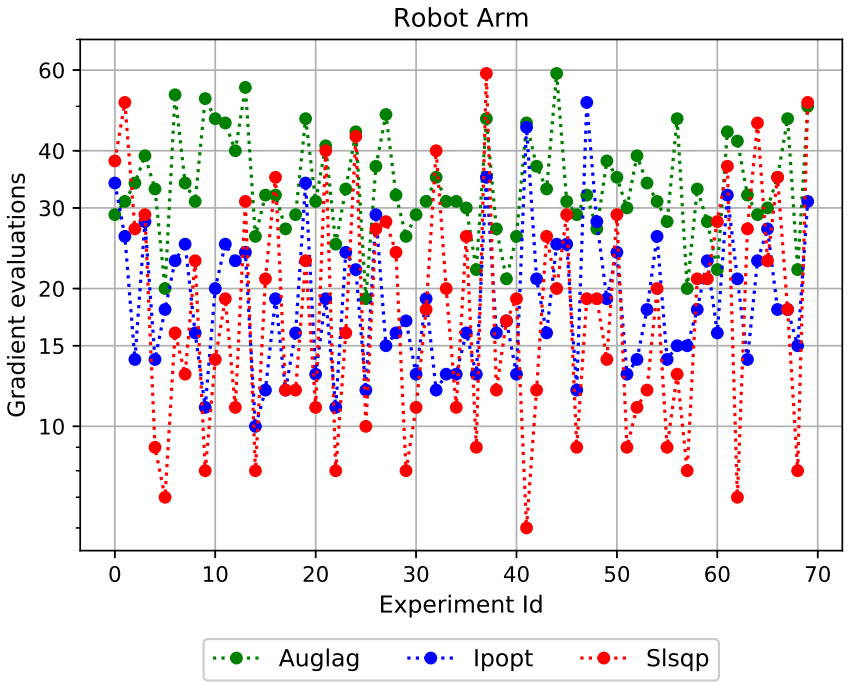

FIGURE 3.3: Robot arm inverse geometry problem, starting from a random initial guess.

Making a fair comparison between the solvers is difficult. Execution speed cannot be compared because augmented Lagrangian is in Python and Ipopt and SLSQP are in Fortran. Also, comparing the number of iterations is unfair, as the workload is different in each case. We decided to report the number of gradient evaluations. Ipopt is the most competitive solver for this well posed and low scale optimization problems. Slsqp shows an irregular behaviour and is very sensitive to the initial conditions.

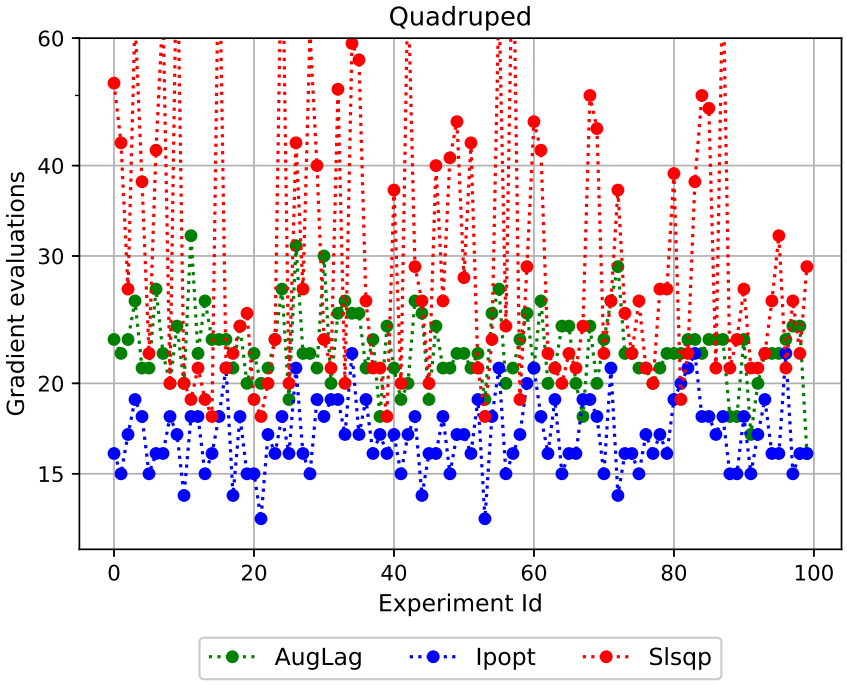

FIGURE 3.4: Quadruped inverse geometry problem, starting from a random initial guess.

Warmstart

Often, in robotics, a "warmstart" is available. That is, a good initial guess of the solution. This happens when solving a sequence of closely related problems. The solution of the first problem can be used as the initial guess of the next one (to warmstart!). For example, the configuration of a manipulator that reaches a given point can be used as an initial guess to optimize the posture that reaches a close point.

Warmstarts are classified as primal, primal-dual or dual, depending on which of the variables of the previous solution are used: primal x , dual λ or both.

In this benchmark, only primal warmstart is used in SLQP and IPOPT. SQP methods are initialized normally using only the primal variables. For example, this is the case of the Slsqp implementation in Scipy. Regarding interior points methods, two reasons justify this decision: the Python wrapper Pyipopt does not support a primal-dual warmstart. Spending time developments this feature was not possible in the scope of this thesis. Experimental results have shown that interior points cannot be warmstarted efficiently, specially when starting from an infeasible region or too close to the constraints [19]. Yet our benchmark would have benefit from this comparison, that we keep as a future task.

On the other hand, a primal-dual warmstart is used in the augmented Lagrangian. This formulation has a clear and explicit notion of the Lagrange multiplier that can be exploited with a good warmstart.

In the new experiments (Figure 3.5, 3.6), the scalar disturbance parameterizes the "distance" to a reference problem, whose optimal values are used as warmstart. This disturbance is applied only on the cost function.

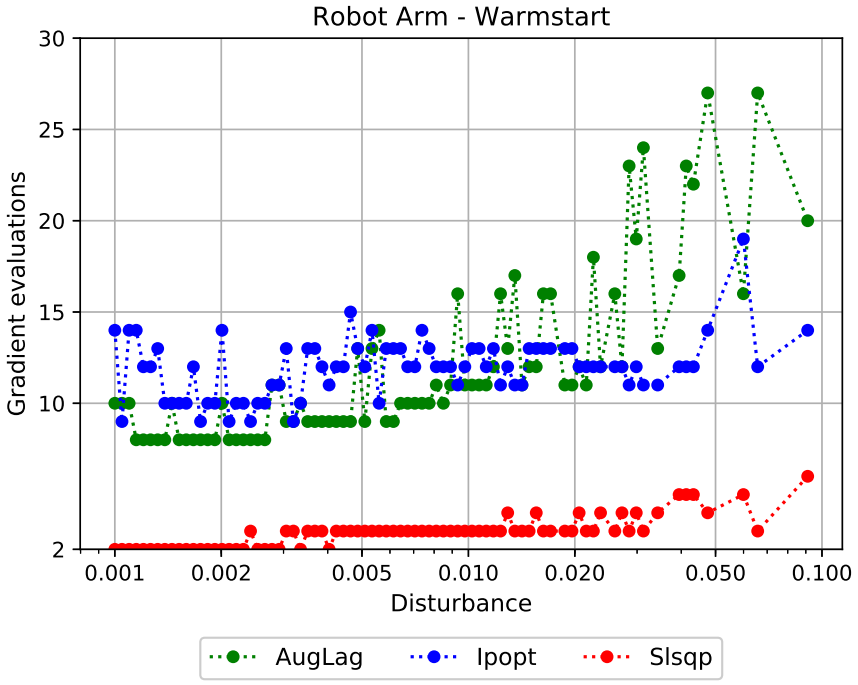

FIGURE 3.5: Robot arm inverse geometry problem, starting from the optimal values of a reference problem (warmstart). The problem variation with respect to the reference is parameterized by the disturbance.

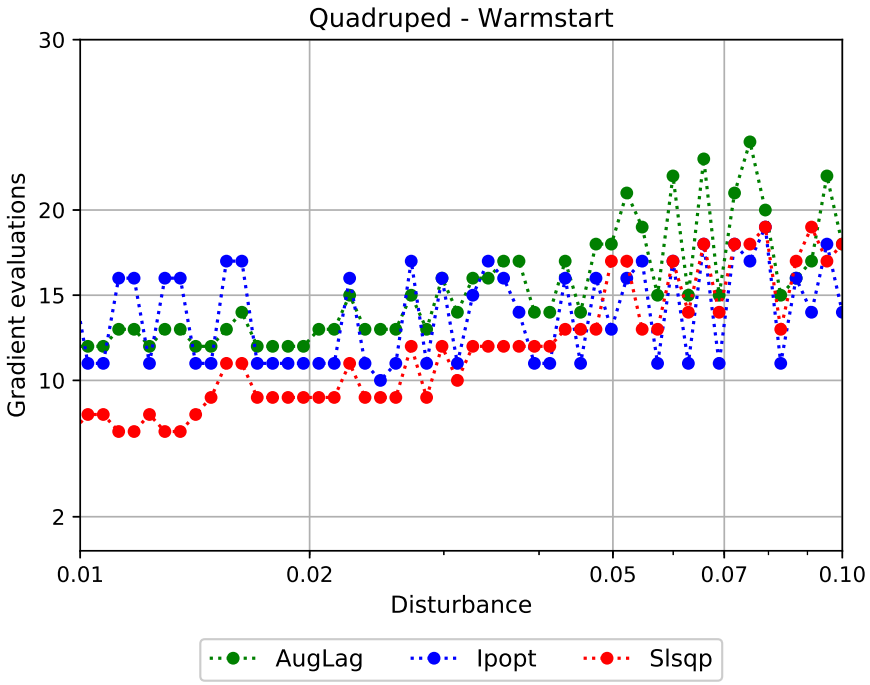

FIGURE 3.6: Quadruped inverse geometry problem, starting from the optimal values of a reference problem (warmstart). The problem variation with respect to the reference is parameterized by the disturbance.

The augmented Lagrangian is warmstarted efficiently (reduction of the number of iterations for smaller disturbances), showing a competitive behaviour in this conditions. Slsqp is now the most efficient solver, and its strategy of constraint linearization and QP solving is successful around the optimum. On the other hand, Ipopt has a good performance but it is not able to warmstart (primal) as the number of iterations does not decrease for small disturbances.

Conclusion

This small benchmark gives a first idea of the performance of the solvers on small size and well posed problems. We have chosen the number of gradient evaluations as metric, which penalizes the augmented Lagrangian as it does not use matrix factorization. In a real benchmark, computation time of efficient implementations should be compared.

Also, the results show a lot of variability between problems and even inside the same model. This highlights, again, the necessity of building a benchmark with a huge number of problems to decide which are the best algorithms.

Finally, in these inverse geometry problems, augmented Lagrangian does not outperform the other solvers. This in normal, because Ipopt and Slsqp are indeed efficient implementations of successful algorithms that will beat augmented Lagrangian in lots of different situations.

However, in our simultaneous contact and trajectory optimization, the optimization will be a large scale problem and complementarity constraints will play an important role. We hold that, under this challenges, the augmented Lagrangian approach will show its potential as an efficient solver for robotics.

3.3 Experimental analysis of augmented Lagrangian

Further understanding of the behaviour of the augmented Lagrangian solver also requires a numeric evaluation of some of its key components. In this section, we report some comparisons and thoughts on the choose of inner solver, penalty update and warmstart stratgey.

3.3.1 Inner solver: bound constrained optimization

The augmented Lagrangian solver is based on solving a sequence of bound constrained problems. Therefore, the performance of this solver is essential. For example, the toolbox NLP.py implements a Python version of TRON [21]. TRON is a state of the art solver for large scale bound constrained optimization, based on trust region projected newton search. The Python version does not use any preconditioner. We compare this with L-BFGS [32] (a bound constrained low-memory quasi Netwon method) and Slsqp from the package Scipy. In our problems TRON shows clearly a better performance (Figure 3.7 , 3.8).

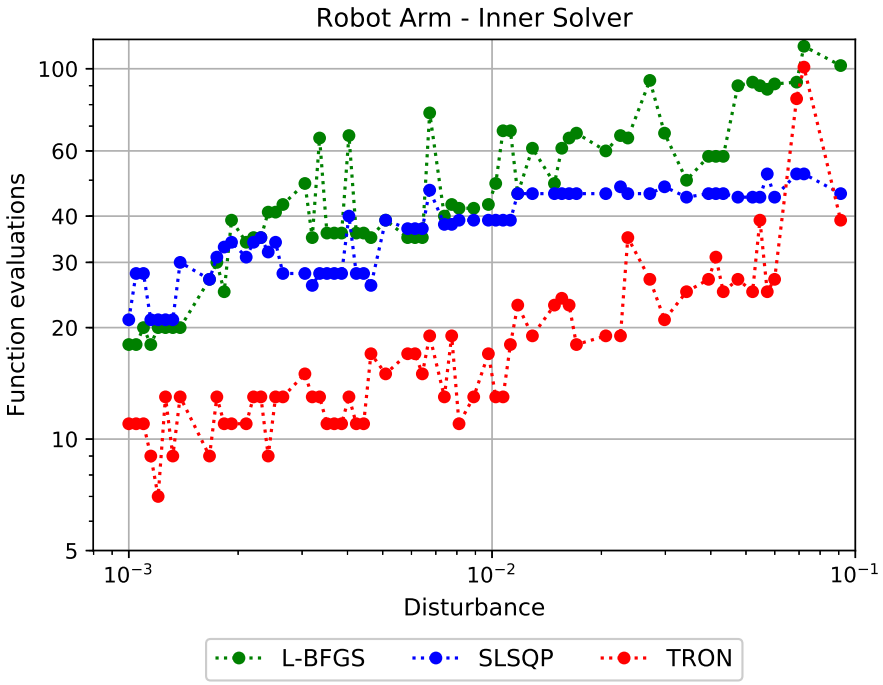

FIGURE 3.7: Robot arm inverse geometry problem, evaluation of inner solvers in augmented Lagrangian. Optimization starts from the optimal values of a reference problem (warmstart). The problem variation with respect to the reference is parameterized by the disturbance.

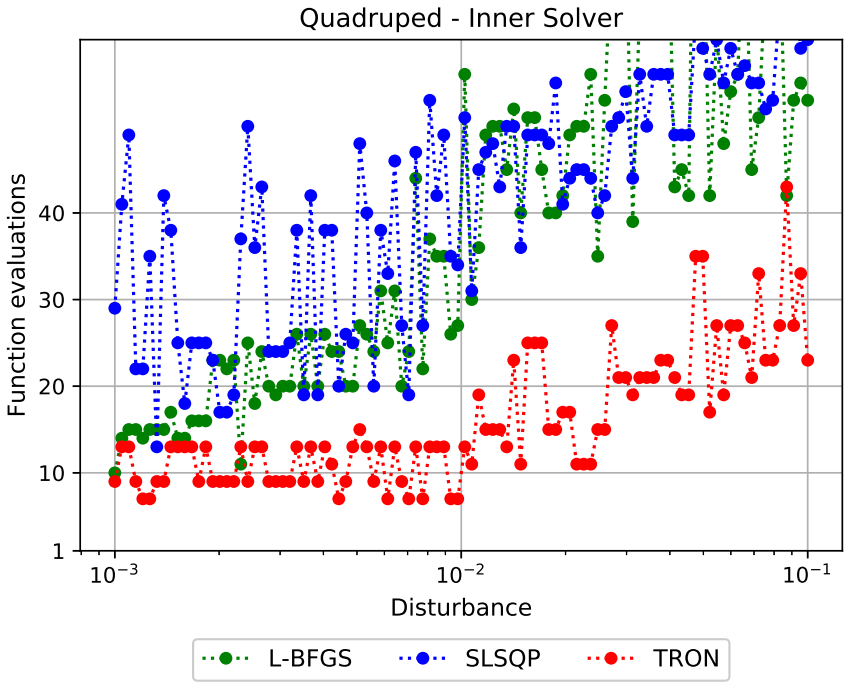

FIGURE 3.8: Quadruped inverse geometry problem, evaluation of inner solvers in augmented Lagrangian. Optimization starts from the optimal values of a reference problem (warmstart). The problem variation with respect to the reference is parameterized by the disturbance.

3.3.2 Penalty update

The formulation of the augmented Lagrangian and its convergence properties are well studied. However, it is still not clear which is the best strategy to update the penalty. Convergence analysis shows that the sequence should be non decreasing: $\mu_k \leq \mu_{k+1}$, but does not provide any further insight on an optimal sequence.

During the project, different strategies were evaluated. In particular, the possibility to reduce the penalty value in some iterations [6] , [13].

Minor variations on the standard penalty strategy were also tested. Although the performance was improving in particular settings, no strategy was performing consistently better. Therefore, the standard penalty update from [23] is adopted for future steps.

3.3.3 Warmstart

We have tested three different warmstart strategies (Fig. 3.9, 3.10). We denote with $(x_s\,, \lambda_s)$ the starting values of the current problem and with $(x_*^-\,, \lambda_*^-)$ the optimal solution of the reference problem. The following methods are evaluated:

- *Primal*: $x_s = x_*^-$ and $\lambda_s = 0$.
- *Primal-dual*: $(x_s\,, \lambda_s) = (x_*^-\,, \lambda_*^-)$
- *Least Squares*: Setting primal variables $(x_s = x_*^-)$, and computing the dual variables λ_s as the solution of a least squares problem using KKT conditions.

$$
\begin{aligned}
& b = \nabla f(x_s^k) \in \mathbb{R}^{n\times 1}, \quad \lambda \in \mathbb{R}^{m\times 1} \\
& A = [\, a_i \,] \in \mathbb{R}^{n\times m}, \quad a_i = \nabla c_i(x_s^k) \in \mathbb{R}^{n\times 1} \\
& \min_{\lambda} \|b - A\lambda\|^2
\end{aligned} \tag{3.3}
$$

  $\lambda = \lambda_*^-$ is not the solution because $f(x)$ and $c(x)$ are perturbed with respect to the reference problem. This is quadratic program: when $m \leq n$ the unique solution is $\lambda = A^+ b$, where $A^+ = (A^T A)^{-1} A^T$ denotes the pseudoinverse.

In general, the least square warmstart has a more irregular behaviour. When the disturbance is small, an update on the multipliers is enough to reach optimality. Also the least square warmstart is more computationally expensive because it requires solving a quadratic program. Therefore, the standard primal-dual warmstart is preferred because of its stability and efficiency.

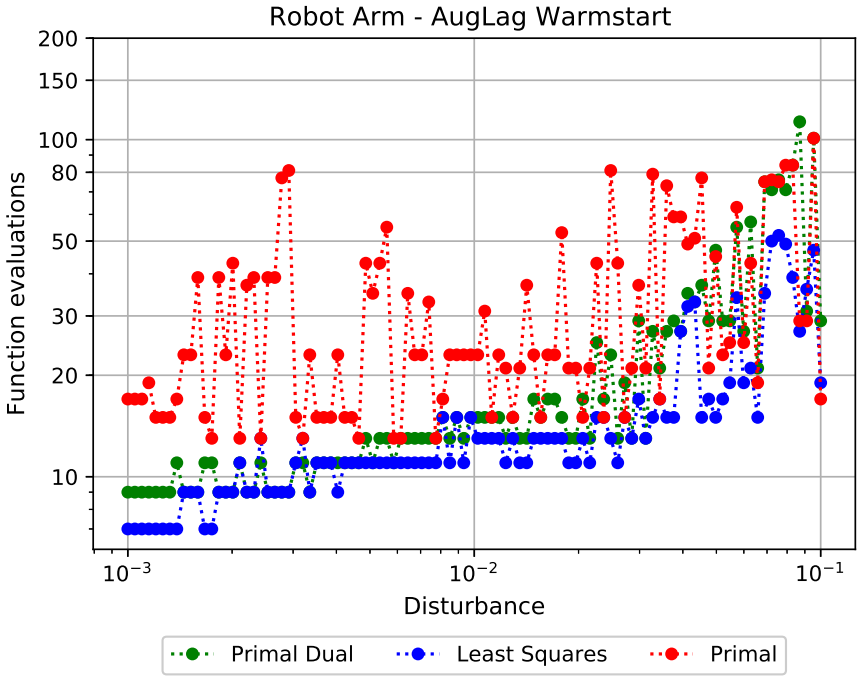

FIGURE 3.9: Robot arm inverse geometry problem, evaluation of different warmstart strategies for the augmented Lagrangian. The problem variation with respect to the reference is parameterized by the disturbance.

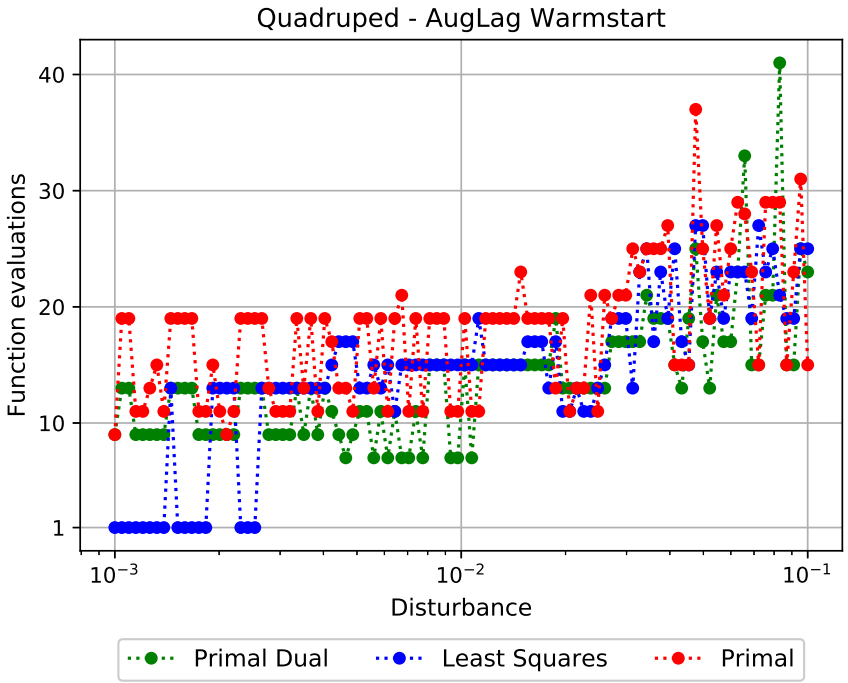

FIGURE 3.10: Quadruped inverse geometry problem, evaluation of different warmstart strategies for the augmented Lagrangian. The problem variation with respect to the reference is parameterized by the disturbance.

Chapter 4

Simultaneous trajectory and contact optimization

In this chapter we propose a method to optimize simultaneously trajectories and contacts for multi-contact manipulation and legged locomotion.

Our pipeline is composed of two interconnected modules: a robotics problem formulation using optimal control and an augmented Lagrangian algorithm to solve the optimization problem.

In the previous chapters we have evaluated these two modules in a rather independent way. First, optimal control in robotics and the challenges of legged locomotion have been presented and discussed (Chapter 1). Second, numerical optimization algorithms have been analysed, focusing on its algorithmic differences (Chapter 2) and a preliminary numeric evaluation in robotics problems (Chapter 3).

Now, we highlight how both modules interacts, explaining and justifying the key elements of the pipeline. Contacts are represented with complementarity constraints, a continuous formulation that can be naturally included inside the optimal control problem. Combining this formulation with an Augmented Lagrangian algorithm, we show that contact and trajectory can be optimized simultaneously.

Finally, we report the first results with a robot arm interacting with the environment.

4.1 Contacts as complementarity constraints

Legged robots must make contacts with the environment to move and interact. We model contacts using complementarity constraints and impose them as path constraints in the optimal control formulation (4.7), (4.10). This eliminates the need of pre-defining contact points, phase times or contact schedules and does not impose any structure on the generated motion.

4.1.1 Contact model

Normal contacts are complementary in the following sense: both the normal force and the distance to the surface must be positive, as normal contacts forces can only push and the robot cannot penetrate the surface. Moreover, either the distance to the surface is zero (contact) or the normal force is zero (no contact).

$$\begin{aligned} z = 0 \quad & \text{horizontal surface (Reference)} \\ z_e \quad & \text{distance to the surface} \\ f_z \quad & \text{normal force} \end{aligned} \tag{4.1}$$

$$\begin{aligned} f_z \geq 0 \quad & \text{force (Push)} \\ z_e \geq 0 \quad & \text{distance (above)} \\ f_z \cdot z_e = 0 \quad & \text{either zero distance (contact) or zero force} \end{aligned} \tag{4.2}$$

A complete contact model is compound of a normal (4.2) and a tangential contact model. Tangential forces lie on contact the surface, and its resulting magnitude f_τ is restricted by the friction coefficient μ and the normal force f_z .

$$\begin{aligned} z = 0 \quad & \text{surface (Reference)} \\ f_x, f_y \quad & \text{components of tangential force} \\ \sqrt{f_x^2 + f_y^2} = f_\tau \leq \mu f_z \quad & \text{friction cone} \end{aligned} \tag{4.3}$$

If $f_\tau < \mu f_z$ the tangential velocity ψ of the contact point is zero. If $f_\tau = \mu f_z$ the tangential velocity ψ is different from zero and points in the opposite direction of the tangential force.

Currently our formulation only considers two cases (although it can be easily extended to the general case $\mu \geq 0$)

- $\mu = 0$ (ice surface model). The tangential forces are 0 and there is no restriction on the tangential velocity.
- $\mu \to \infty$ (no-slip model). The tangential forces can be arbitrarily large when $f_z > 0$. The no-slip constraint can be easily stated using the normal force:

$$\begin{aligned} f_z \geq 0 \quad & \text{normal force} \\ \psi_x, \psi_y \quad & \text{Components of tangential velocity} \\ \psi_x \, f_z = 0 & \\ \psi_y \, f_z = 0 & \end{aligned} \tag{4.4}$$

4.1.2 Complementarity constraints and difficulties of MPCC

The normal contact model (4.2) corresponds to the standard mathematical definition of complementarity constraints. An optimization with complementarity constraints is called MPCC (mathematical program with complementarity constraints, see [26] for a general introduction on stationarity, optimality, and sensitivity). In its general form, complementarity of constraints $g(x)$ and $h(x)$ is written as:

$$\begin{aligned} g(x) &\geq 0 \\ h(x) &\geq 0 \\ g(x)h(x) &= 0 \end{aligned} \tag{4.5}$$

This type of constraints is usually challenging for standard numerical solvers. The main difficulties are:

- The optimization problem is degenerated at $g(x) = 0$, $h(x) = 0$. The linearized feasible set (first orders feasible directions) does not model the real the feasible set (2.4), (2.5). This inconsistency will affect the behaviour of methods that rely on constraint linearization.
- They do not satisfy the majority of the established constraint qualifications. For example, the gradients are not linearly independent (lack of LICQ)

$$\frac{\partial}{\partial x} g(x)h(x) = g(x)\nabla h(x) + h(x)\nabla g(x) \tag{4.6}$$

From a theoretic point of view, this requires the introduction and study of new stationary concepts and constraint qualifications. On the practical side, researcher have studied the behavior of general nonlinear optimization algorithms when applied to MPCC, proposing modifications to adapt them to this new problem. For example, an extension of interior points [25], SQP [16] and a general regularization [27].

4.1.3 Augmented Lagrangian to solve MPCC

The augmented Lagrangian formulation has already gained attention from the numerical optimization community for dealing with complementarity constraints. Convergence results and numerical experiments show the potential of this algorithm. For a detailed convergence analysis we directly refer the reader to [2] and [18], and provide here a qualitative and general idea.

Augmented Lagrangian formulation is appealing for these problems because it is based on solving unconstrained subproblems and does not linearize explicitly the constraints. Adding the constraints in the cost with a quadratic penalty is a natural way to relax and regularize the complementarity in the first iterations.

The iterative sequence produced by the algorithm is not required to be feasible (except clearly for the last iteration). Then, it is possible to go from a starting point on one side of the feasible set, for example $h(x) = 0$, to the other, $g(x) = 0$, without crossing the degenerated point $h(x) = g(x) = 0$.

This approach does not require any external continuation strategy or added formulation, as the framework already contains a penalty parameter that relaxes the complementarity constraints.

A standard augmented Lagrangian algorithm has been shown to compete and outperform specialized versions of interior points and sequential quadratic programming for this type of constraints [18]. The authors recommend augmented Lagrangian when the focus is robustness and globalization properties. The difference between these dedicated solvers from its classical version is the implementation of an elastic mode (penalizing infeasible linearizations) and a regularization. This also highlights further potential for improvement for augmented Lagrangian, probably employing the special structure of the MPCC.

4.2 Optimal control formulation

The trajectory and contact optimization is formulated as a continuous optimal control problem that includes the contact model (4.1) as a path constraint. (See Section 1.1.2 for a short introduction to optimal control).

$$x(t) : \mathbb{R} \to \mathbb{R}^n \quad u(t) : \mathbb{R} \to \mathbb{R}^m$$

$$\min_{x(\cdot),u(\cdot)} E(x(T)) + \int_0^T l(x(t), u(t))\, dt \tag{4.7}$$

$$\text{subject to :} \tag{4.8}$$

$$\begin{aligned} & f(x(t), u(t), t) = \dot{x}(t), \quad && \forall t \in [0, T] \\ & h(x(t), u(t), t) \leq 0, && \forall t \in [0, T] \\ & x(0) = x_0 \end{aligned}$$

where $x(t)$ is the state and $u(t)$ is the control. $E(\cdot)$, $l(\cdot)$, $f(\cdot)$ and $h(\cdot)$ are, respectively, the terminal cost, the path cost, the dynamics and the path constraints.

The optimization problem (4.7) is solved numerically with a direct method. Direct methods parameterize the problem with a finite number of variables, and convert it to a finite dimension non linear program (this process is known as transcription). This approach is informally described as: "First discretize, then optimize".

There are two families of direct methods: shooting (only the control is a decision variable, and the state is computed with the dynamics) and collocation (both control and state are decision variables, and the dynamics are imposed at given set of time stamps, called collocation points).

In this project we will use collocation for two reasons:

- *Stability*. Good stability compared to single shooting. Single shooting methods fail in unstable and chaotic dynamics, where small variations in the initial conditions of the trajectory have a huge effect on the final state. This instability of single shooting can be mitigated with multiple shooting methods.
- *Simplicity*. Compared to multiple shooting, both methods have a good stability but the implementation of collocation is more intuitive.

In our approach (direct method, collocation), the time is discretized in N intervals. The state is modelled as a piece wise third-order polynomial and the control as piece wise constant. The dynamic model is a second order differential equation. The control includes the external forces and either joint acceleration (kinematic model) or joint torques (dynamic model).

$$
\begin{aligned}
&N \text{ intervals} \\
&t_k \text{ time grid} \quad k = 0, ..., N \\
&t_{k,i} \in [t_k, t_{k+1}] \text{ Collocation times} \quad i = 1, ..., M \\
&\bar{u}_k \text{ parameterize } u(t) \in [t_k, t_{k+1}] \\
&\bar{x}_k \text{ parameterize } x(t) \in [t_k, t_{k+1}] \\
&\bar{x} = [\bar{x}_k] \quad \bar{u} = [\bar{u}_k]
\end{aligned}
\tag{4.9}
$$

$$
\begin{aligned}
&\min_{\bar{x},\bar{u}} \bar{E}(\bar{x}_{N-1}) + \sum_k \bar{l}(\bar{x}_k, \bar{u}_k) \\
&\text{Dynamics at } t_{k,i} \quad i = 1, ..., M, \; k = 1, ..., N \\
&\text{Path Constraints at } t_{k,i} \quad k = 1, ..., N, \; i = 1, ..., M \\
&\text{Continuity } x(t) \quad \text{at } t_k \quad k = 1, ..., N \\
&\text{Initial Conditions at } t_0
\end{aligned}
\tag{4.10}
$$

The term $\sum \bar{l}(\bar{x}_k, \bar{u}_k)$ approximates the integral in (4.7) and $\bar{E}(\bar{x}_{N-1})$ corresponds to $E(x(T))$ taking into account the chosen parametrization. Initial conditions are set on the state position and velocity.

State and Control Parametrization. Inside each time interval, the state is defined as a piecewise 3rd order polynomial and it is parameterized by its coefficients. The parametrization is linear: all kinematic quantities evaluated at a given time are a linear function of the coefficients. In this project, both 3rd order standard (4.11) and Lagrange polynomials (4.12) have been studied and tested. Standard polynomials are chosen because, thanks to better scaling properties (details below), they fit better inside the augmented Lagrangian Formulation.

$$
\begin{aligned}
&\text{(standard)} \\
&p(t;\, t_0) = a + b(t - t_0) + c(t - t_0)^2 + d(t - t_0)^3 \\
&p(t_0) = a, \; \dot{p}(t_0) = b, \; \ddot{p}(t_0) = 2c, \; \dddot{p}(t_0) = 6d
\end{aligned}
\tag{4.11}
$$

$$
\begin{aligned}
&\text{(Lagrange)} \\
&t_r = \begin{bmatrix} t_0, \; t_1, \; t_2, \; t_3 \end{bmatrix} \\
&p(t;\, t_r) = a\, l_0(t, t_r) + b\, l_1(t, t_r) + c\, l_2(t, t_r) + d\, l_3(t, t_r) \\
&l_i(t, t_r) = \prod_{j \neq i} \frac{t - t_j}{t_i - t_j} \\
&p(t_0) = a, \; p(t_1) = b, \; p(t_2) = c, \; p(t_3) = d
\end{aligned}
\tag{4.12}
$$

Each polynomial representation has its own strengths. The advantage of Lagrange polynomials is that joint limits constraints (very common in robotics) can be applied directly as bound constraints to the coefficients (which are the optimization variables). In a numerical algorithm, dealing with bounds is faster than with linear equalities.

Standard polynomials are a sparser representation: for example, acceleration only depends on two coefficients. Also, kinematics at the beginning of the interval can be set directly as bounds on the coefficients. They are better scaled when computing velocities and accelerations. The influence of the coefficients is either 0 or is well-balanced with the contributions of the other coefficients.

Numerically, this means that the gradient of kinematic quantities with respect to the polynomial coefficients is better scaled using standard polynomials (elements in gradient vectors and Jacobians have the same magnitude).

The control is parameterized as a piecewise constant function, with two steps inside each interval. The choice of the control model is arbitrary, and depends on the physical interpretation of the problem. In our case, control represents either forces or accelerations. In robots with powerful actuators, it is reasonable to assume that these inputs can be produced in a discontinuous manner. Also, contact forces with the environment are discontinuous.

$$t \in [t_k, t_{k+1}] \tag{4.13}$$

$$u(t) = \begin{bmatrix} u_{k0} & \text{if} & t \leq t_s \\ u_{k1} & \text{if} & t > t_s \end{bmatrix} \tag{4.14}$$

$$t_s = \frac{t_{k+1} + t_k}{2} \tag{4.15}$$

Dynamics

Dynamic equations are imposed at two internal time frames of each interval (collocation points). Therefore, the 3rd degree polynomial is well defined by four equations: two dynamic constraints and an initial position and velocity. Initial position and velocity of the first interval correspond to the initial conditions. In the following intervals, initial position and velocity are constrained to be continuous with the end of the previous interval.

Continuity

The control is discontinuous between intervals. For the state, continuity on position and velocity is imposed as a constraint.

Path constraints

Path constraints can be evaluated either at a coarse time grid t_k $k = 0, ..., N$ or at a finer grid, using, for example, also the collocation points $t_{k,i}$. We impose path constraints at a fine grid, using the collocation points. Path constrains will include the complementarity constraints arising from the contact model.

4.3 Conditioning of direct collocation in optimal control

Optimal control with direct collocation are, in general, not well scaled problems. Position, velocity and acceleration at any given time are a linear function of the coefficients of the parametrization. The relation is linear, but the gradient is not well scaled, in the sense that their values can be from different magnitude. We clarify this idea with a little example:

$$\begin{aligned}
\text{Variables} \quad & x = [a, b, c, d] \\
\text{fixed time} \quad & T = 0.01 \\
& p_T(x) = a + bT + dT^2 + cT^3 \\
\text{Optimization Variable:} \quad & x = [a, b, c, d] \\
\text{constraint function} \quad & c(x): \; p_T(x) = p_0 \\
\text{constraint jacobian} \quad & \nabla c(x): \; [1, T, T^2, T^3] \\
\text{Evaluated at T=0.01} \quad & \nabla c(x): \; [1, 0.01, 0.01^2, 0.01^3]
\end{aligned} \tag{4.16}$$

The augmented Lagrangian formulation is based on solving a sequence of minimizations of the augmented Lagrangian of the problem (4.17), with Hessian (4.18).

$$\mathcal{L}_A(x; \lambda, \mu) = f(x) - \sum_i \left(\lambda_i c_i(x) - \tfrac{1}{2}\mu c_i^2(x)\right) \tag{4.17}$$

$$\nabla^2_{xx}\mathcal{L}_A(x; \lambda, \mu) = \nabla^2 f(x) - \sum \left(\left(\lambda_i - \mu c_i(x)\right) \nabla^2 c_i(x) - \mu \nabla c_i(x)^T \nabla c_i(x) \right) \tag{4.18}$$

Note that the gradient of the constraints appears "squared" $\nabla c_i(x)^T \nabla c_i(x)$ in the Hessian of the augmented Lagrangian. Therefore, the bad scale of the gradients is now squared in the hessian, making the problem ill conditioned in some cases. Ill conditioned means that the condition number κ of the matrix $\nabla^2_{xx}\mathcal{L}_A$ is large.

$$\kappa(A) = \frac{|\sigma_{\max}(A)|}{|\sigma_{\min}(A)|} \tag{4.19}$$

where A is squared matrix and $\sigma_{\max}$ and $\sigma_{\min}$ its maximum and minimum (in modulus) eigenvalues.

The identity matrix has a condition number of 1. In an ill conditioned system, the steepest descent direction $(-\nabla f(x))$ is not a good search direction. Making steps following this direction will converge only with a very large number of iterations. Therefore, using exact second order information of the Hessian is essential.

A bad condition number also has a terrible effect on the performance of the conjugate gradients method: an iterative method for solving large scale linear systems (in the context of optimization: $\nabla^2 f(x_k)\; p = -\nabla f(x_k)$). We can accelerate the performance of conjugate gradients method by transforming the linear system (with a change of variables) to get a better condition number on the system matrix. This is known as preconditioning.

$$\begin{aligned}
&\min_x f(x) && \text{ill conditioned problem} \\
&x = S\bar{x} && \text{change of variables} \\
&\min_{\bar{x}} f(S\bar{x}) && \text{equivalent problem, better condition}
\end{aligned} \tag{4.20}$$

The performance of quasi Newton methods, where the hessian is estimated from gradient information, will also decrease. Estimating a bad conditioned Hessian is harder than a Hessian close to the identity matrix.

Moreover and on top of the scaling arising from the optimal control, augmented Lagrangian formulations always get ill-conditioned when penalty increases. Although the penalty remains smaller than standard penalty methods, it still presents a difficulty.

4.4 Development of a dedicated solver

We have built our own augmented Lagrangian solver to work with these optimal control problems. It can handle correctly the ill-conditioning of the system and is fast for our development purposes. Speed is achieved by using a compiled unconstrained solver of Scipy as the basic unit inside the bound constrained solver.

Augmented Lagrangian in NLP.py

The augmented Lagrangian solver available in the Python package NLP.py has been used in the previous benchmark on inverse geometry problems (Chapter 3). The performance was good and the package provides a flexible optimization environment that can be easily extended and modified.

However, it was not robust against the ill conditioning of the problem coming from the optimal control formulation (see Section 4.3). Moreover, testing large scale problems became tedious and inefficient. NLP.py is completely written in Python, an interpreted programming language that is slower than compiled languages, such as Cpp and Fortran.

Preconditioning the system efficiently (for speed and convergence) was time-consuming and complex. The core element of the augmented Lagrangian in NLP.py is the trust region solver TRON [21] with a truncated conjugate gradients. However, compared to the original implementation of TRON, it does not include a preconditioner. (TRON uses a limited-memory Cholesky factorization of the Hessian as preconditioner)

Dedicated augmented Lagrangian

Our dedicated augmented Lagrangian solver is faster (less execution time) because it relies on a compiled implementation of an unconstrained solver of Scipy. Also, at the cost of computing a matrix factorization, is robust against ill-conditioning.

The chosen unconstrained solver is called "Trust-Exact", a trust region unconstrained minimization in which quadratic subproblems are solved almost exactly [12]. This algorithm requires the gradient and the Hessian (which is not required to be positive definite). As it computes a factorization of the Hessian, it is suited for small and medium size problems, but not large scale settings.

The bound constraints are added following an active set approach (see active set in Section 2.2.3). At each step, a subset of bounds is imposed as equalities and the other are ignored. The unconstrained solver solves the reduced problem (only using the "free variables") stopping either at converge or when it generates an infeasible step with respect to the inactive bounds. The second stopping criterium is implemented through a software trick, as the unconstrained solver does not handle bound limits.

A projected line search is used to compute the size (and projection) of the last step. When the step is infeasible, the projected line search bends around the box with a normal projection.

In augmented Lagrangian, bound constraints problems are solved in a sequence. Normally, the set of bound that are active are identified during the first bound constraints optimizations and are then maintained through the later outter iterations.

The magic step

The magic step (inspired by the package NLP.py) consists on exploiting the structure of the problem coming from the slack variables inside the bound constrained solver. For notation simplicity, we suppose that the optimization problem has only inequalities.

$$\begin{gathered} x \in \mathbb{R}^n \\ \min_x f(x) \quad \text{s.t} \quad c_i(x) \geq 0 \;\; i = 1, ..., m \quad \text{(original)} \end{gathered} \tag{4.21}$$

$$\begin{gathered} x \in \mathbb{R}^n, \quad s \in \mathbb{R}^m \\ \min_{x,s} f(x) \quad \text{s.t} \quad c_i(x) - s_i = 0 \quad i = 1, ..., m \quad \text{(with slack)} \end{gathered} \tag{4.22}$$

$$c(x) : \mathbb{R}^n \to \mathbb{R}^m \quad \nabla_x c(x) : \mathbb{R}^n \to \mathbb{R}^{m \times n} \tag{4.23}$$

$$\nabla \mathcal{L}_A(x, s, \lambda, \mu) = \left[\nabla f(x),\ 0_m\right] - \left(\lambda - \mu \left(c(x) - s\right)\right) \left[\nabla_x c(x),\ -I_m\right] \tag{4.24}$$

By setting $\nabla_s \mathcal{L}_A(x, s, \lambda, \mu) = 0$ an explicit step for the slack variables can be computed (subject to $s \geq 0$). In practice, the magical can be applied after the standard minimization with respect to all variables x, s. With this feature, slack variables can enter and leave the active set in a dynamic and efficient way.

The focus of the current implementation is on robustness and not speed. The performance is satisfactory on all the tested optimal control problems (around 500 variables). However, it remains unclear how it will scale to large problems. Also, it is important to point out that, with the actual problems size, computing the hessian and Jacobian matrices, using for-loops in Python, takes more time than matrix vector products and matrix factorizations.

4.5 Results

In this section we report our preliminary results. We apply the whole pipeline to a robot manipulator that makes contacts with the environment to perform a set of different motions and tasks. Trajectory and contacts are optimized simultaneously with an augmented Lagrangian algorithm. The results are promising and show the potential of this new formulation. We also discuss our plans to apply this formulation to generate a walking motion for a quadruped robot, highlighting the main challenges.

4.5.1 Development of a polynomial collocation module

We have implemented a module that generates a non linear program from a continuous optimal control problem using collocation. The proposed polynomial collocation formulation is generic and follows [14]. Developing our own module gives us control over the whole collocation scheme, with flexibility on the choice of the state parametrization and, specially, on how complementarity is formulated and implemented.

The software pipeline (from model to solution) has three main independent components: a model, the polynomial collocation, and the solver.

- *Model* implements the functions (with its Jacobians) that define the continuous optimal control problem. It includes: dynamic equations, path and terminal cost, path constraints and initial and terminal constraints. We use the Pinocchio library [10] for computing robot kinematics and dynamics.
- *Polynomial collocation* converts the continuous optimal control problem defined by the model into a non linear constrained program using a collocation (direct method).
- *Solver* finds the solution of the optimization problem. We interact with the following solvers: interior points (Pyipopt), sequential quadratic programming (Slsqp from Scipy) and augmented Lagrangian (NLP.py package and our dedicated implementation)

The pipeline is developed and tested using three different dynamic systems:

- a 1D block. It has an analytic solution and is a quadratic program.
- Cart Pole. Underactuated system with unstable dynamics.
- 7-joint manipulator arm LWR KUKA 4 with a position task.

All solvers converge to the same locally optimum solution (there are no complementarity constraints in these three problems). The results we obtain with the augmented Lagrangian are validated with the other solvers.

These first experiments highlight that augmented Lagrangian is prone to bad conditioning when applied to collocation (Section 4.3). Standard 3rd degree polynomial parametrizations are better scaled than Lagrangian Polynomials and will be latter used in more complex problems.

4.5.2 Multi-contact motions with a robot manipulator

We simultaneously optimize trajectory and contacts of a robot manipulator that touches the environment with its end effector. A set of different motions in increasing complexity have been computed: trajectories touching one wall, two walls and pick and place tasks.

Contact is modelled through complementarity constraints (4.25), (4.26); set as path constraints at a dense grid discretization. (see Section 4.1)

$$
\begin{aligned}
&\text{(Normal Contact model)} \\
&f_n \geq 0 \quad \text{Normal force} \\
&p_n \geq 0 \quad \text{distance to surface} \\
&f_n \; p_n = 0
\end{aligned}
\tag{4.25}
$$

$$
\begin{aligned}
&\text{(Tangential Contact model)} \\
&v_1, v_2 \quad \text{Tangential velocities} \\
&\text{if } \mu = 0: \\
&\qquad v_1, v_2 \quad \text{free} \\
&\text{if } \mu \to \infty: \\
&\qquad f_n \; v_1 = 0 \\
&\qquad f_n \; v_2 = 0
\end{aligned}
\tag{4.26}
$$

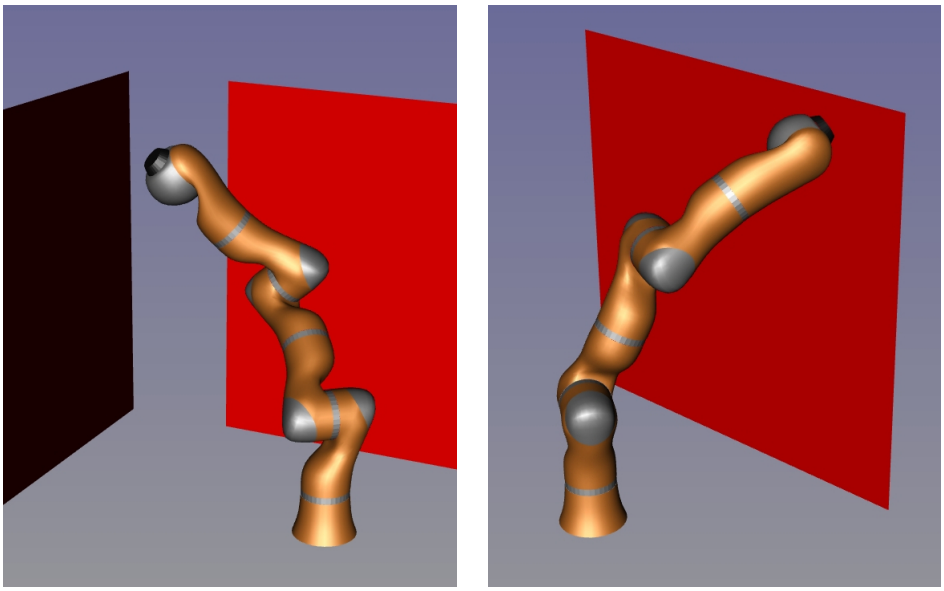

FIGURE 4.1: Simultaneous trajectory and contact optimization with a robot manipulator. The environment is modelled with planes. Left: Touching two walls motion. Right: Pick and place task.

Motion 1: Touching two walls

The environment has two perpendicular walls. We compute a trajectory that begins with the end effector at a desired position and exerts forces on different directions at different times (Fig. 4.2). The order is predefined by constraining the force value at a chosen frame. The problem is formulated using no friction ($\mu = 0$) and no slip ($\mu \to \infty$) tangential contact models. The cost function regularizes the robot configuration, joint control and contact forces. The optimal control formulation of this problem is described in detail in Appendix A. The initial guess for the optimization is a fixed robot around the starting goal. Optimized trajectories are shown in Fig. 4.3, 4.4.

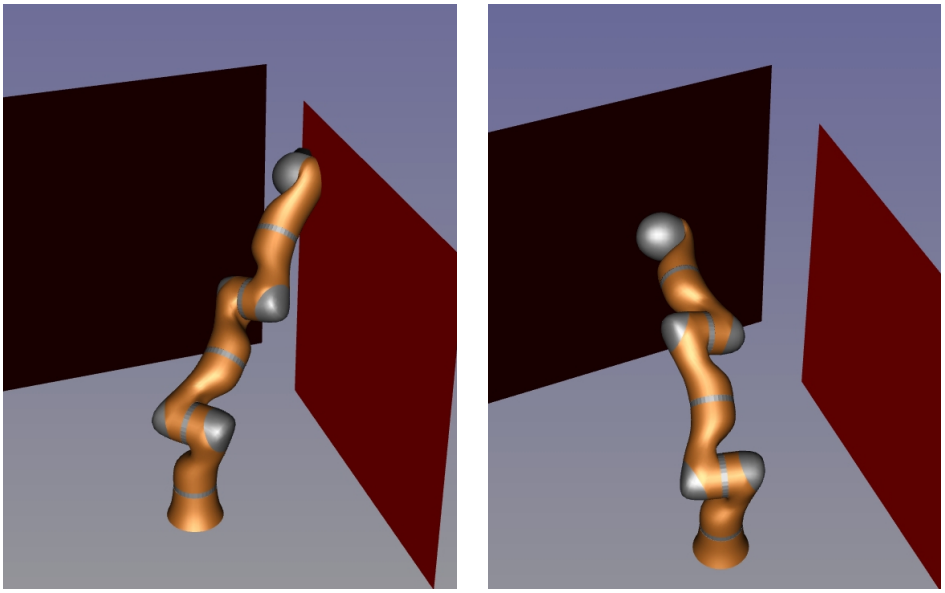

FIGURE 4.2: Touching two walls motion. Left: Robot configuration when it makes contact with the first wall. Right: Final position of the motion, touching the second wall.

Motion 2: Pick and place

The second motion represents a pick and place task and is inspired by a manipulator taking one object from a flat surface and leaving it in another position (Fig. 4.5). To achieve this behaviour, the initial and terminal force values are constrained, and the start and end robot configurations are optimized to be close to two user-defined references.

A key aspect is to find a cost function that encourages lifting the end effector from the surface (so that a straight line motion on (or slightly above) the surface is avoided). Although this could be an efficient motion in terms of energy, it is not reliable for execution on a real robot. The desired "flying" phase is achieved by adding a term to the cost that maximizes the distance of the end effector to the surface during the trajectory. The cost function also contains a term to regularizes the robot configuration, joint control and contact forces. The optimal control formulation of this problem is described in detail in Appendix A. The initial guess for the optimization is a fixed robot around the starting goal. Different trajectories can be generated changing the weights between initial and terminal cost, control regularization and distance to surface maximization. Optimized trajectories are shown in Fig. 4.6, 4.7.

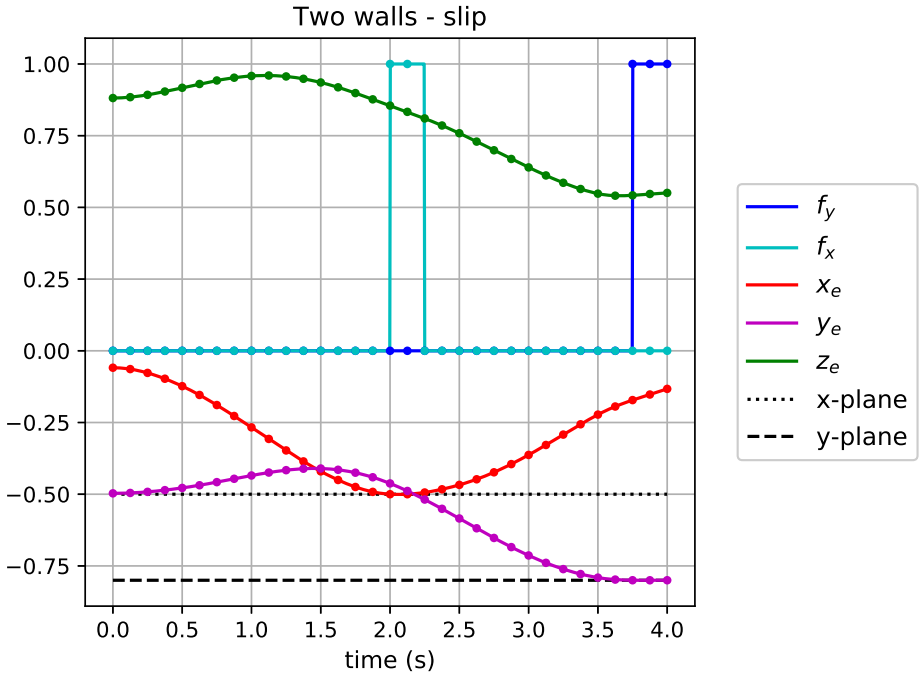

FIGURE 4.3: Touching two walls motion with $\mu = 0$. Showing end effector trajectory x_e, y_e, z_e and contact forces. Dots indicate collocation points. No friction contact model, $\mu = 0$ (free tangential velocity).

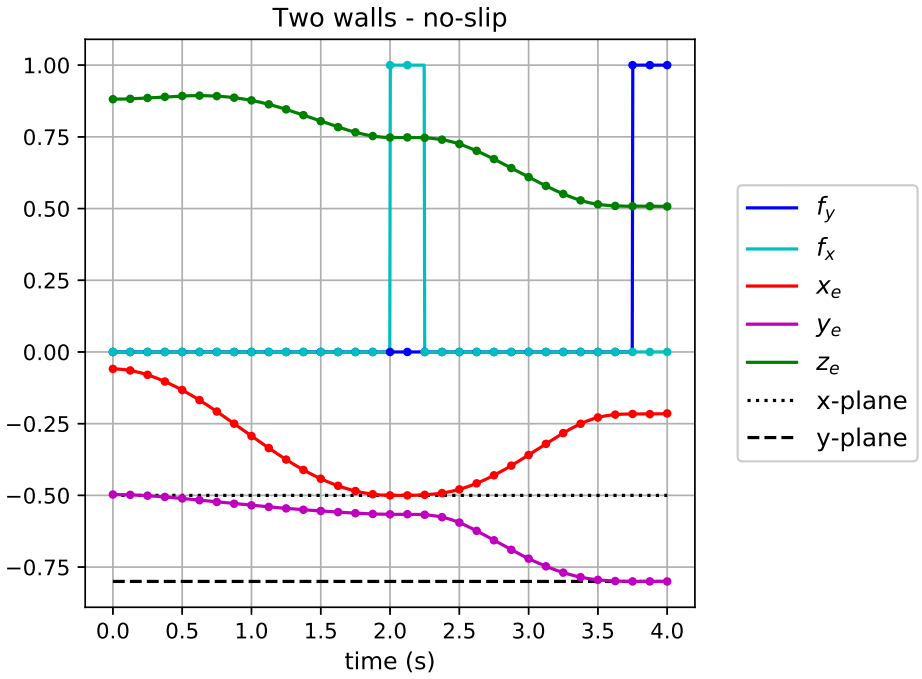

FIGURE 4.4: Touching two walls motion with $\mu \to \infty$. Showing end effector trajectory x_e, y_e, z_e and contact forces. Dots indicate collocation points. No slip contact model, $\mu \to \infty$ (zero tangential velocity when normal force is strictly positive).

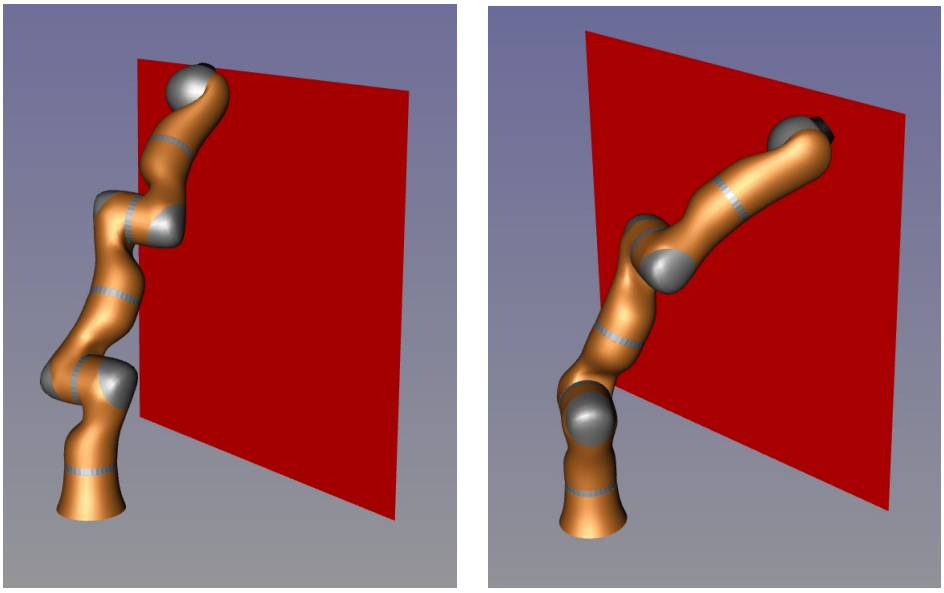

FIGURE 4.5: Pick and place motion. Left: initial configuration of optimized trajectory (in contact) Right: final configuration of optimized trajectory (in contact).

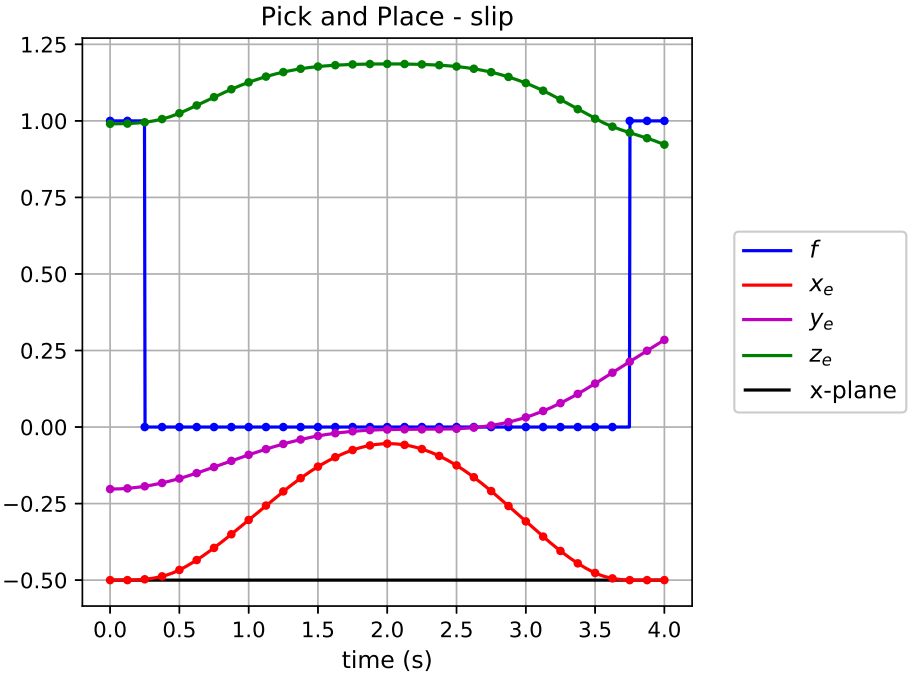

FIGURE 4.6: Pick and place motion with $\mu = 0$. Showing end effector trajectory x_e, y_e, z_e and contact force. Dots indicate collocation points. No friction contact model, $\mu = 0$ (free tangential velocity).

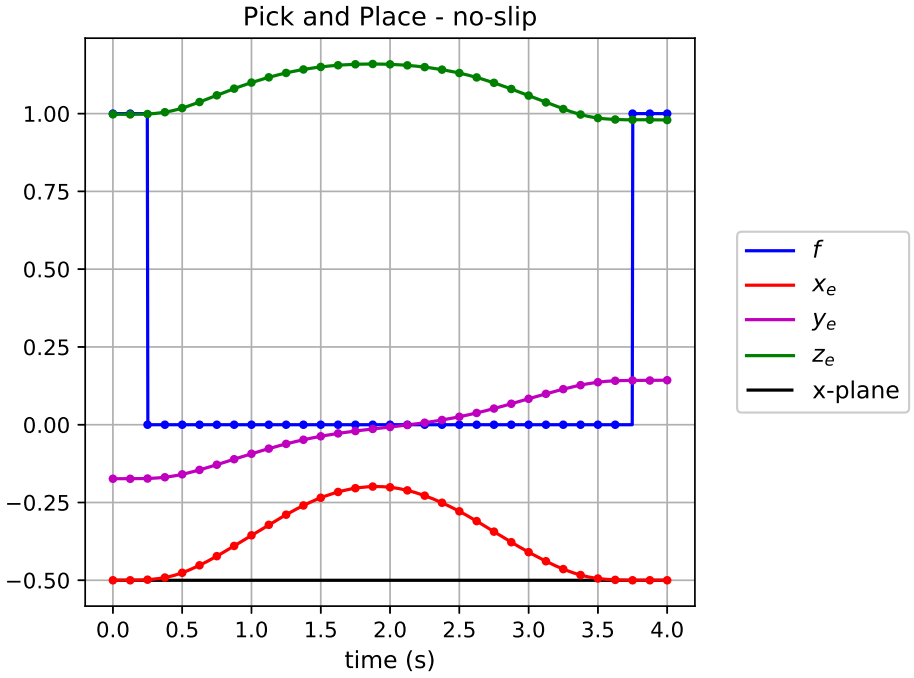

FIGURE 4.7: Pick and place motion with $\mu \to \infty$. Showing end effector trajectory x_e, y_e, z_e and contact force. Dots indicate collocation points. No friction contact model, $\mu = 0$ (free tangential velocity).

Comments on the results

In average, the problems are solved with 6 outter iterations (penalty increase and multiplier update) and 40 inner iterations (accumulative bound constrained iterations) of the augmented Lagrangian. Computation time (from a far initial guess) ranges from 5 to 30 seconds. Most of the time is spend in the computation of the Hessian and Jacobian matrices, that is done in Python without parallelization.

Ipopt and Slsqp have not been directly tested on these problems. However, they have been evaluated on inverse geometry problems with complementarity constraints, where the posture of a quadruped is optimized without predefining the contacts. In this problem, they were either failing or slower than the augmented Lagrangian, that delivered good and robust results.

Testing the modified versions of Ipopt and Slsqp for handling with complementarity constraints remains as a future task. Notice that also the augmented Lagrangian Algorithm is applied in its classical form, without any modification for dealing with the complementarity structure.

4.5.3 Legged locomotion

The motivation of this project is to generate walking motions for a quadruped robot. Now the contacts are not only a way to interact with the environment, but also a requirement to move the robot. Although the presented formulation can be directly applied to this robot, the complexity of the system could require new contributions to achieve robustness and fast performance. In comparison to a robot manipulator making contacts, the main challenges are the following:

- Large dimension of the system

  A single configuration of a quadruped robot has 18 state variables. (12 joints + 6 parameters of the free floating base) and 24 control variables (12 joint torques or acceleration + 3 forces in each feet) Our manipulator had 7 state variables and around 10 controls (7 torques or acceleration and 1-2 forces on the end effector). The time discretization should be dense (more time intervals) to allow a good resolution for deciding when the foot starts and ends contacts. Also, time horizon is typically larger when several gait cycles are optimized. Overall, the bigger dimension could require a sparse resolution, avoiding matrix factorization and exploiting the block structure of collocation scheme.

- Initial guess.

  As locomotion is a complex and unstable control system, we require a reasonably good initial guess trajectory. This arises four interesting open question: how good should this initial trajectory be? How far is the found optimal solution from the initial guess? Can the optimization algorithm completely change the gait pattern? And finally, is the found local optimum close to the real global optimum of the problem? Moreover, the potential of hint functions or virtual restrictions in the early stages of the optimization to improve convergence should be evaluated.

- Suitable objective function

  Optimizing the trajectory and contacts simultaneously is still not well studied in the community. In this context, there are few references about which objective function will induce a good motion, similar to human and animal locomotion. Minimizing energy consumption is a clear candidate, but new metrics to represent the robustness and stability of the trajectory are essential to transfer these algorithms to real robots.

Chapter 5

Conclusion and perspectives

Conclusion

Optimization of a motion with contacts is an open problem, in particular in locomotion. Discontinuous dynamics, instability and the combinatorial structure of contact planning are the main challenges in this field. Researchers in robotics have proposed different ideas pointing in quite different directions. For example, decoupling the contact and trajectory optimization as two different subproblems; or parameterizing the contact sequence in a continuous manner .

In this thesis, we have proposed to optimize simultaneously trajectory and contacts, focusing on both the problem formulation and the numerical optimization algorithm.

On the formulation part, we have modelled contacts using complementarity constraints, and imposed them as path constraints inside our trajectory optimization pipeline. The trajectory optimization is converted into a finite dimensional problem using direct collocation.

On the optimization side, we have used an augmented Lagrangian optimization algorithm. The augmented Lagrangian successfully handles complementarity constraints, can be easily warmstarted and is suitable for large scale problems, as it can be implemented without matrix factorizations. We use our own augmented Lagrangian implementation, based on projected line searches and trust region, yet implemented with matrix factorization.

The whole pipeline, from high-level transcription of the motion problem to the optimization with augmented Lagrangian, has been designed to work as a single unit. The interaction between components is bidirectional: the problem is formulated to fit the optimization algorithm, and the optimization algorithm is chosen based on the problem definition.

We have shown the results of our pipeline with a robot manipulator that interacts with the environment. We are able to optimize motions such as touching walls and pick and place tasks, without pre-defining any structure on the contacts sequence.

Augmented Lagrangian is not a common choice in robotics, where researchers usually choose off-the-shelf implementations of interior points and sequential quadratic programming solvers .

Understanding the underlying mechanism of numerical solvers is essential for solving complex robotics problems. During this project we have also studied the behaviour of interior points and sequential quadratic programming. Together with augmented Lagrangian, they have been evaluated in two low-dimensional inverse geometry problems. Although fair comparisons are difficult and depend on the chosen implementation, the three algorithms are competitive. The interior points solver shows the best performance from a random initial guess. However, generic implementations of interior points and sequential quadratic

programming are not efficient in a problem with complementarity constraints, where augmented Lagrangian is the clear winner.

Perspectives

Current results with the manipulator arm show the potential of this formulation to optimize trajectories for legged robots. New objective functions that encourage an efficient, robust and stable motion, and at the same time, do not deteriorate the performance of the augmented Lagrangian algorithm are a necessary contribution.

Our current augmented Lagrangian Algorithm relies on matrix factorization which does not exploit the structure of this algorithm, that can be implemented in a matrix free approach. Given the bad scaling of direct collocation, a preconditioner is essential. Building an efficient preconditioner, based on the numerical optimization literature and open source codes [21], will enable the optimization of trajectories with large systems and long time horizons. We are also be curious to see how robust codes like Algencan [3] and Lancelot [11] can solve this problem.

Trajectory and contact optimization is, in general, not a convex problem. Gradient-based methods will recover, if successful, a locally optimum solution. Our long term view is that a theoretic and convergence analysis of these problems will be extremely useful and could result in new techniques and algorithms. Also, it will highlight how continuous optimal control should be combined with sampling and search algorithm to achieve a better performance.

This project has strengthened our initial intuition that augmented Lagrangian should be the method of choice for implementing sparse solvers for robotics problems, specially in multi-contact environments. The augmented Lagrangian algorithm is a simple framework that relies on an inner solver. Therefore, available solvers in robotics can be extended to more general settings by adding new and challenging constraints with the augmented Lagrangian formulation.

Appendix A

Formulation details of multi-contact motions with a robot manipulator

In this appendix, we specify the formulation of the continuous optimal control problems of Section 4.5.2. They correspond to the two motions generated with the robot manipulator making contacts with the environments. These problems are converted into a finite dimension non linear optimization program using direct collocation (4.10). The optimization problem is solved with our dedicated Augmented Lagrangian solver (Section 4.4).

Motion 1: touching two walls

$$\begin{aligned} &\text{Joint position} && q(t) : \mathbb{R} \to \mathbb{R}^7 \\ &\text{Joint control} && u(t) : \mathbb{R} \to \mathbb{R}^7 \\ &\text{Force x-plane} && f_x(t) : \mathbb{R} \to \mathbb{R} \\ &\text{Force y-plane} && f_y(t) : \mathbb{R} \to \mathbb{R} \end{aligned} \tag{A.1}$$

$$\begin{aligned} &\text{End effector position} && p(q) : \mathbb{R}^7 \to \mathbb{R}^3 \\ &\text{End effector velocity} && v(q, \dot{q}) : \mathbb{R}^7 \times \mathbb{R}^7 \to \mathbb{R}^3 \\ &\text{3D starting position} && p^* \in \mathbb{R}^3 \\ &\text{Joint limits} && l,\ u\ \in\ \mathbb{R}^7 \\ &\text{Regularization weights} && k_q, k_u, k_f \in \mathbb{R}_{>0} \end{aligned} \tag{A.2}$$

$$\text{x-Plane, y-Plane} \quad x_{\text{plane}}\ ,\ y_{\text{plane}} \in \mathbb{R} \tag{A.3}$$

$$\min_{q(\cdot), u(\cdot), f_x(\cdot), f_y(\cdot)} \int_0^T k_u \left\|u\right\|^2 + k_q \left\|q\right\|^2 + k_f\ f_x^2 + k_f\ f_y^2\ \ dt \tag{A.4}$$

subject to: (A.5)

$$\begin{aligned} & p(q(0)) = p^* && \text{initial conditions} \\ & \dot{q}(0) = 0 \\ & f_x(T/2) = 1 && \text{touch two walls} \\ & f_y(T) = 1 \\ & \forall\ t \in [0, T] : \\ & \quad u(t) = \ddot{q}(t) && \text{dynamics} \\ & \quad l < q(t) < u && \text{joint limits} \\ & \quad f_x(t)\ (p_x(q(t)) - x_{\text{plane}}) = 0 && \text{normal contact} \\ & \quad f_y(t)\ (p_y(q(t)) - y_{\text{plane}}) = 0 \\ & \quad f_y(t),\ f_x(t) \geq 0 \\ & \quad p_x(q(t)) - x_{\text{plane}}\ ,\ p_y(q(t)) - y_{\text{plane}} \geq 0 \\ & \quad \text{if } \mu \to \infty : && \text{tangential contact} \\ & \qquad f_x(t)\ v_y(q(t), \dot{q}(t)) = 0 \\ & \qquad f_x(t)\ v_z(q(t), \dot{q}(t)) = 0 \\ & \qquad f_y(t)\ v_x(q(t), \dot{q}(t)) = 0 \\ & \qquad f_y(t)\ v_z(q(t), \dot{q}(t)) = 0 \end{aligned}$$

Motion 2: pick and place

$$\begin{aligned} &\text{Joint position} \quad q(t) : \mathbb{R} \to \mathbb{R}^7 \\ &\text{Joint control} \quad u(t) : \mathbb{R} \to \mathbb{R}^7 \\ &\text{force x-plane} \quad f_x(t) : \mathbb{R} \to \mathbb{R} \end{aligned} \tag{A.6}$$

$$\begin{aligned} &\text{End effector position} \quad p(q) : \mathbb{R}^3 \to \mathbb{R}^7 \\ &\text{End effector velocity} \quad v(q, \dot{q}) : \mathbb{R}^7 \times \mathbb{R}^7 \to \mathbb{R}^3 \\ &\text{Start, end reference configurations} \quad q_0^*,\ q_T^* \in \mathbb{R}^7 \\ &\text{Joint limits} \quad l,\ u \in \mathbb{R}^7 \\ &\text{Regularization weights} \quad k_0, k_T, k_q, k_u, k_f, k_x \in \mathbb{R}_{>0} \\ &\text{x-Plane} \quad x_{\text{plane}} \in \mathbb{R} \end{aligned} \tag{A.7, A.8}$$

$$\begin{aligned} \min_{q(\cdot), u(\cdot), f_x(\cdot)} \; & k_0 \left\| q(0) - q_o^* \right\|^2 + k_T \left\| q(T) - q_T^* \right\|^2 \\ & + \int_0^T k_u \left\| u(t) \right\|^2 + k_q \left\| q(t) \right\|^2 + k_f \, f_x^2(t) - k_x p_x(q(t)) \; dt \end{aligned} \tag{A.9}$$

subject to: (A.10)

$$\begin{aligned} &\dot{q}(0) = 0 && \text{initial conditions} \\ &f_x(0) = 1 && \text{pick and place} \\ &f_x(T) = 1 \\ &\forall\, t \in [0, T] : \\ &\quad u(t) = \ddot{q}(t) && \text{dynamics} \\ &\quad l < q(t) < u && \text{joint limits} \\ &\quad f_x(t) \; (p_x(q(t)) - x_{\text{plane}}) = 0 && \text{normal contact} \\ &\quad f_x(t),\ p_x(q(t)) - x_{\text{plane}} \geq 0 \\ &\quad \text{if } \mu \to \infty : && \text{tangential contact} \\ &\qquad f_x(t) \; v_y(q(t), \dot{q}(t)) = 0 \\ &\qquad f_x(t) \; v_z(q(t), \dot{q}(t)) = 0 \end{aligned}$$

Bibliography

[1] Bernardo Aceituno-Cabezas et al. "Simultaneous contact, gait, and motion planning for robust multilegged locomotion via mixed-integer convex optimization". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 2531–2538.

[2] Roberto Andreani, Leonardo D Secchin, and Paulo JS Silva. "Convergence properties of a second order augmented Lagrangian method for mathematical programs with complementarity constraints". In: *SIAM Journal on Optimization* 28.3 (2018), pp. 2574–2600.

[3] Roberto Andreani et al. "On augmented Lagrangian methods with general lower-level constraints". In: *SIAM Journal on Optimization* 18.4 (2007), pp. 1286–1309.

[4] Sylvain Arreckx, Dominique Orban, and Nikolaj van Omme. "NLP.py: An Object-Oriented Environment for Large-Scale Optimization". In: *Cahier du GERAD G* 2016 (2016), p. 42.

[5] Sylvain Arreckx et al. "A matrix-free augmented Lagrangian algorithm with application to large-scale structural design optimization". In: *Optimization and Engineering* 17.2 (2016), pp. 359–384.

[6] Ernesto G Birgin and José Mario Martínez. "Augmented Lagrangian method with nonmonotone penalty parameters for constrained optimization". In: *Computational Optimization and Applications* 51.3 (2012), pp. 941–965.

[7] Stanislas Brossette, Adrien Escande, and Abderrahmane Kheddar. "Multicontact Postures Computation on Manifolds". In: *IEEE Transactions on Robotics* (2018).

[8] Justin Carpentier et al. "A versatile and efficient pattern generator for generalized legged locomotion". In: *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE. 2016, pp. 3555–3561.

[9] Justin Carpentier et al. "Multi-contact Locomotion of Legged Robots in Complex Environments The Loco3D project". In: *RSS Workshop on Challenges in Dynamic Legged Locomotion*. 2017, 3p.

[10] Justin Carpentier et al. "The Pinocchio C++ library–A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives". In: *International Symposium on System Integrations*. 2019.

[11] Andrew R Conn, GIM Gould, and Philippe L Toint. *LANCELOT: a Fortran package for large-scale nonlinear optimization (Release A)*. Vol. 17. Springer Science & Business Media, 2013.

[12] Andrew R Conn, Nicholas IM Gould, and Ph L Toint. *Trust region methods*. Vol. 1. Siam, 2000.

[13] Frank E Curtis, Hao Jiang, and Daniel P Robinson. "An adaptive augmented Lagrangian method for large-scale constrained optimization". In: *Mathematical Programming* 152.1-2 (2015), pp. 201–245.

[14] Moriz Diehl and Gros Sebastien. *Numerical Optimal Control*. Freiburg, 2017.

[15] Roy Featherstone. *Rigid body dynamics algorithms*. Springer, 2014.

[16] Philip E Gill, Walter Murray, and Michael A Saunders. "SNOPT: An SQP algorithm for large-scale constrained optimization". In: *SIAM review* 47.1 (2005), pp. 99–131.

[17] Nicholas IM Gould, Dominique Orban, and Philippe L Toint. "CUTEst: a constrained and unconstrained testing environment with safe threads for mathematical optimization". In: *Computational Optimization and Applications* 60.3 (2015), pp. 545–557.

[18] Alexey F Izmailov, Mikhail V Solodov, and EI Uskov. "Global convergence of augmented Lagrangian methods applied to optimization problems with degenerate constraints, including problems with complementarity constraints". In: *SIAM Journal on Optimization* 22.4 (2012), pp. 1579–1606.

[19] Elizabeth John and E Alper Yıldırım. "Implementation of warm-start strategies in interior-point methods for linear programming in fixed dimension". In: *Computational Optimization and Applications* 41.2 (2008), pp. 151–183.

[20] Dieter Kraft. "A software package for sequential quadratic programming". In: 88-28 (1988).

[21] Chih-Jen Lin and Jorge J Moré. "Newton's method for large bound-constrained optimization problems". In: *SIAM Journal on Optimization* 9.4 (1999), pp. 1100–1127.

[22] Igor Mordatch, Emanuel Todorov, and Zoran Popović. "Discovery of complex behaviors through contact-invariant optimization". In: *ACM Transactions on Graphics (TOG)* 31.4 (2012), p. 43.

[23] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. second. New York, NY, USA: Springer, 2006.

[24] Michael Posa, Cecilia Cantu, and Russ Tedrake. "A direct method for trajectory optimization of rigid bodies through contact". In: *The International Journal of Robotics Research* 33.1 (2014), pp. 69–81.

[25] Arvind U Raghunathan and Lorenz T Biegler. "An interior point method for mathematical programs with complementarity constraints (MPCCs)". In: *SIAM Journal on Optimization* 15.3 (2005), pp. 720–750.

[26] Holger Scheel and Stefan Scholtes. "Mathematical programs with complementarity constraints: Stationarity, optimality, and sensitivity". In: *Mathematics of Operations Research* 25.1 (2000), pp. 1–22.

[27] Stefan Scholtes. "Convergence properties of a regularization scheme for mathematical programs with complementarity constraints". In: *SIAM Journal on Optimization* 11.4 (2001), pp. 918–936.

[28] Steve Tonneau et al. "An efficient acyclic contact planner for multiped robots". In: *IEEE Transactions on Robotics* (2018).

[29] Marc Toussaint. "A Novel Augmented Lagrangian Approach for Inequalities and Convergent Any-Time Non-Central Updates". In: *arXiv preprint arXiv:1412.4329* (2014).

[30] Andreas Wächter and Lorenz T Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical programming* 106.1 (2006), pp. 25–57.

[31] Alexander W Winkler et al. "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1560–1567.

[32] Ciyou Zhu et al. "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization". In: *ACM Transactions on Mathematical Software (TOMS)* 23.4 (1997), pp. 550–560.

This report is based on the Latex Template *Masters/Doctoral Thesis* from LatexTemplates made by Vel, Steve Gunn and Sunil Patel.