Optimal control and machine learning for humaoid and aerial robots

Mathieu Geisert

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ FÉDÉRALE TOULOUSE MIDI-PYRÉNÉES

Délivré par :

l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *23/04/2018* **par :**

MATHIEU GEISERT

OPTIMAL CONTROL AND MACHINE LEARNING FOR HUMANOID AND AERIAL ROBOTS

JURY

JUAN CORTÈS	Directeur de Recherche	Président du Jury
KAREN LIU	Assistante Professeur	Membre du Jury
PIERRE-YVES OUDEYER	Directeur de Recherche	Membre du Jury
JONAS BUCHLI	Professeur	Membre du Jury
STÉPHANE DONCIEUX	Professeur des Universités	Membre du Jury

École doctorale et spécialité :
EDSYS : Robotique 4200046
Unité de Recherche :
Laboratoire d'Analyse et d'Architecture des Systèmes
Directeur de Thèse :
Nicolas MANSARD
Rapporteurs :
Pierre-Yves Oudeyer et *Karen LIU*

Cette thèse est dédicacée à ma mère et à tous les gens qui m'ont poussé à aller toujours plus loin.

Remerciements

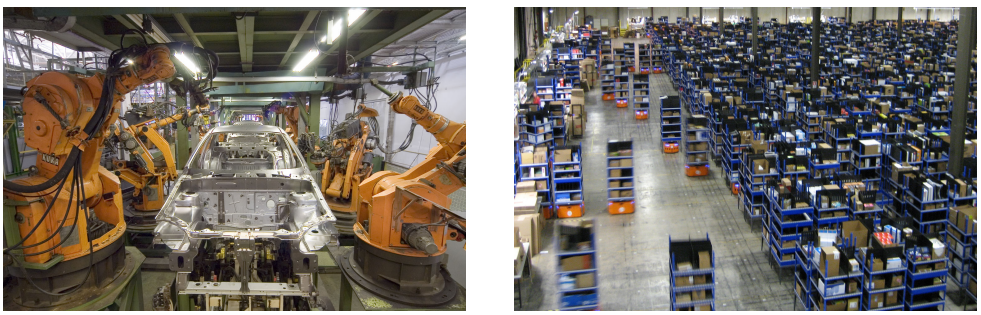
Je remercie en premier mon directeur de thèse Nicolas Mansard qui m'a suivi tout au long de cette thèse, qui a su m'orienter, me motiver et m'aider à accomplir ce travail. Je remercie aussi tout particulièrement Olivier Stasse qui m'a introduit à la robotique humanoïde, m'a énormément appris et soutenu pendant toute la période que j'ai passée au LAAS. Je remercie aussi Florent Lamiraux, sans qui je ne serais probablement pas resté au LAAS et Andrea Del Prete qui a réussi à me montrer comment être plus rigoureux lors de notre collaboration. Plus généralement, je remercie toute l'équipe Gepetto car le travail de thèse n'est pas un travail solitaire, c'est un travail d'équipe avec des collaborations scientifiques, du partage de connaissance, de logiciel et de tâche. Je remercie Justin Carpentier de m'avoir guidé sur beaucoup de points scientifiques et techniques, Joseph Mirabelle et Guilhem Saurel pour tous leurs conseils informatiques, Maximillien Naveau et Mehdi Benallegue pour toutes ces petites discussions sans buts, mais qui au final permettent de mieux cerner les problèmes sur lesquels nous travaillons. Je remercie Florent Lamiraux, Joseph Mirabelle et plus particulièrement Steve Tonneau et Pierre Fernbach pour l'énorme travail qu'ils font pour développer le logiciel de planification de l'équipe et les algorithmes associés, et sans qui je n'aurais jamais pu faire une partie de mon travail sur l'apprentissage automatique. Je remercie aussi toutes les personnes qui travaillent sur le robot et les logiciels associés, c'est-à-dire Olivier Stasse, Andrea Del Prete, Thomas Flaviol, Justin Carpentier, Nicolas Mansard, Maximilien Naveau, Guilhem Saurel, Kévin Giraud, Florent Forget, Rohan Budhiraja, Alexis Mifsud, Mehdi Benallague car sans eux les tests sur nos plateformes robotiques seraient impossibles. Je remercie toute l'équipe Gepetto pour sa bonne ambiance et tous les liens extrêmement forts d'amitié que j'ai eus pendant ces 5 ans et particulièrement avec Maximilien Naveau et Mylène Campana. Pour finir, je veux remercier mes colocs, Ellon Paiva Mendes, Petra Mulloreva, Quang Hung Nguyen, Nassime Blin et Shuai Wang avec qui nous formons une vraie famille et avec qui j'ai passé de très bons moments ici à Toulouse Un dernier mot pour remercier Philippe Souères, le LAAS, sa direction et son équipe administrative qui font un travail de gestion important et qui permet aux étudiants, comme moi de se concentrer sur les travaux techniques et scientifiques.

Contents

Introduction

0.1 Robots and Locomotion

The twentieth and now twenty-first centuries have seen flourish robots almost everywhere. Starting from expensive yet limited robotic arms in industries, robots now populate the world and evolve on lands, seas and also skies. From fixed-base systems like the industrial robots solidly screwed to the ground, robots are now given the ability of locomotion. Robots can evolve in an entire factory as the robots in the warehouses of *Amazon*(Fig. 1). The development of safe and robust controllers allows robots to get out to public environments like shops, metro stations or even streets (Fig. 2). Moreover, technologies come now at an affordable price so robots are becoming tools not only used by companies but also by private individuals (Fig. 3). From all those examples of mobile robots we can see a common characteristic. All those robots are wheeled robots i.e. they use wheels to move in their environment. Wheeled locomotion is simple and efficient way to move but come at a certain cost: the floor needs to be even. This limitation can be particularly incapacitating when moving in wild environments but also in human environments like towns. Sidewalks or stairs are insurmountable obstacles for wheeled robots. Moreover, in the cases of natural catastrophes like earthquakes, even wheel-friendly environments quickly become impracticable for those robots. Improving locomotion capabilities of robots can be done by changing the physical principle used. Instead of slip-free rolling principle, locomotion can take advantage of contact transition (i.e. legged locomotion) or aerodynamics (i.e. aerial locomotion). The main purpose of the work presented in this thesis is to propose motion generation strategies able to take advantage of the extended mobility of aerial and legged robots, with an accent on the latest.



(a) *KUKA* robots in a car factory (b) *Amazon* warehouse

Figure 1: Factory/warehouse robots.

(a) The inventory robot of *PAL Robotics*

(b) The cleaning robot of *Taski*

(c) The autonomous shuttle of *Navya*

Figure 2: Public-environment robots.

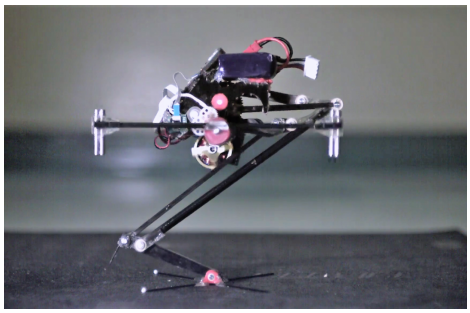
(a) *Spider* vacuum cleaner

(b) *Tesla* self-driving car

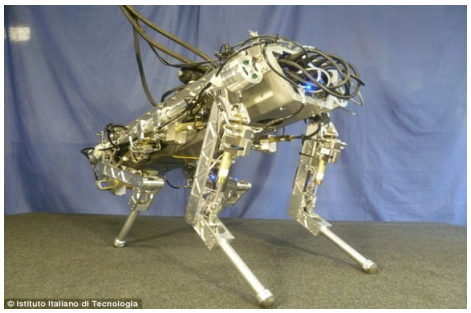
Figure 3: Robots for private individuals.

0.1.1 Legged Robots

Legged robots use articulated kinematic chains to generate contact with the environment and exploit reaction and friction forces to move their body. Legged robots have shown great capabilities to go through uneven terrains but also to climb or jump if needed (Fig. 4).



(a) 1-leg robot *Salto*



(b) 4-leg robot *HyQ*

(c) 6-legs walking tractor of *John Deer*

Figure 4: Legged robots/machines.

0.1.2 Humanoid Robots

Our world has been reshaped by humans for humans. We have been constructing our environment following our morphology: floors, doors, stairs, walkways have all been designed at human scale. Moreover, all tools have also been designed to be used by humans i.e. for beings with two arms and human hands. While for early manipulator robots, robots were surrounded with cages to avoid any unplanned obstacles and railways were installed to ease robot locomotion, we are now able to adapt robots to the environment instead of the opposite. In a human world, humanoid robots are intrinsically well designed to evolve in this world. For instance, dog robots are great to walk outside but would not be able to reach the top drawer in a kitchen or to use an electric drill although human-shape robots could (Fig. 5). In that sense, we are more specifically interested in biped robots (able to walk in narrow cumbersome spaces) with advanced bi-arm manipulation capabilities and perception: humanoid robots! Thanks to the minor environment and tool adaptations that would need humanoid robots, industries like *Airbus* start to see the industrial potential of such robots [Stasse 2014] (Fig. 6).

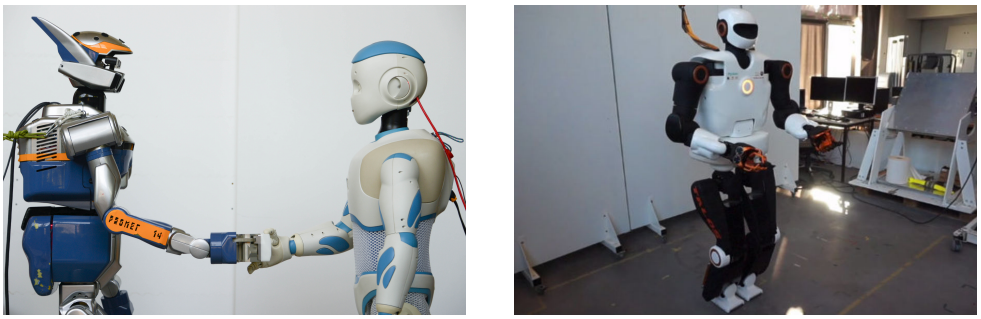


Figure 5: *Gepetto* Humanoid Robots: *HRP-2*, *Romeo* and *Pyrene* [Stasse 2017].

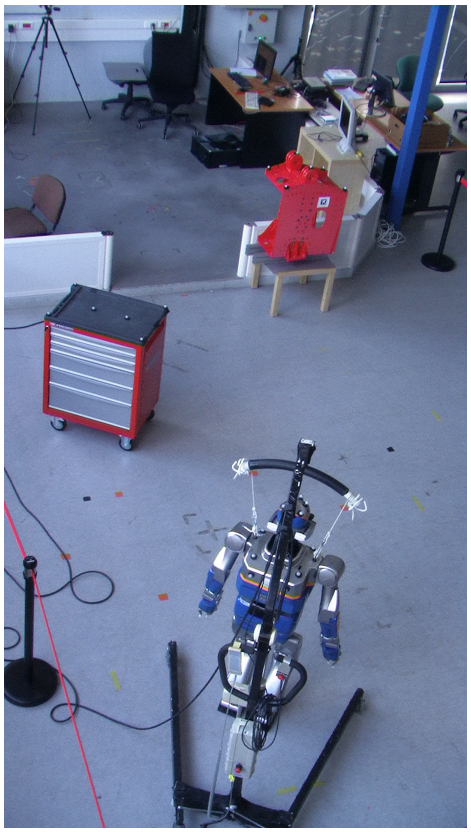


Figure 6: Interactive planning for an *Airbus* project.

0.2 Thesis presentations

In this document we study generic methods to make legged or aerial robots evolve on uneven and/or cluttered environments. Our claim here is that a combination of online *Model Predictive Control* (i.e. direct numerical resolution of *Optimal Control Problems* while the robot is moving) and offline *Machine Learning* (i.e. construction of simple models from offline simulations) are key elements to solve such complex problems in a generic manner. The first chapter of this document introduces the

contributions through an analysis of the locomotion problem in humanoid robotics. We will show that this problem raises several difficulties like underactuation, redundancy or nonconvexity. We analyze the approaches used to solve them and propose new methods or new applications. Chapter 2 acts as a motivation of our research by reporting the application of direct numerical optimal control on flying robots. The first part of this chapter will show how this technique can be used to easily solve complex tasks like optimal time trajectories or pick and place tasks. Moreover, we will show that this technique is fast enough to be used online, and could be directly used to help a teleoperator to navigate in a cluttered environment. The second part will briefly show how learning methods can be used to improve the computation times and the efficiency of the results. The rest of the thesis propose modifications and increment to the nominal *Model Predictive Control* paradigm. In Chapter 3, we address the problem of defining more complex tasks, as it typically arises in humanoid robotics, when multiple terms enter in the cost function and must be ordered into a strict hierarchy. In Chapter 4, we work on the preliminary planning phase, needed to start the *Model Predictive Control* search in the locomotion context. We show how learning techniques can be used to efficiently take into account geometric or dynamic constraints to choose a path.

Chapter 1

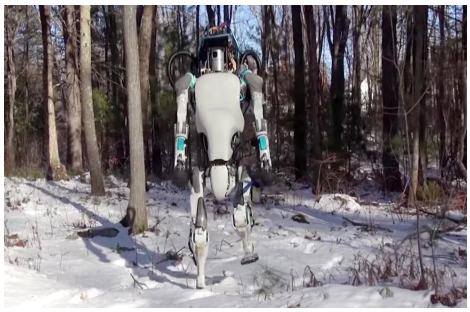
Humanoid Robotics

1.1 Humanoid Robotics: what remains to be solved?

Researchers have succeeded in generating very impressive motions with humanoid robots. As we can see on different videos published by Boston Dynamics, their robots are able to walk on various terrains such as structured environments (Fig. 1.1a), natural scabrous terrains (Fig. 1.1b) and slippery surfaces (Fig. 1.1c). They are also able to stay stable after high perturbations (Fig. 1.1d).

(a) Walking inside

(b) Walking outside



(c) Walking on slippery terrain

(d) Rejecting perturbation

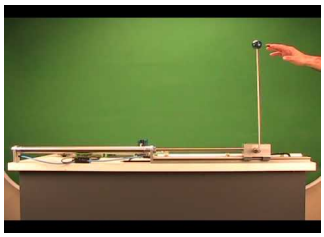
Figure 1.1: ©*Boston Dynamics* Achievements

However humanoid robotics is one of the most difficult disciplines roboticists are trying to solve. If we have a look on the last *DARPA Robotic Challenge* (DRC), we can see by the number of falls that the problem is still unsolved (Fig. 1.2). Why?

Figure 1.2: Humanoid robots falling at the *DRC*.

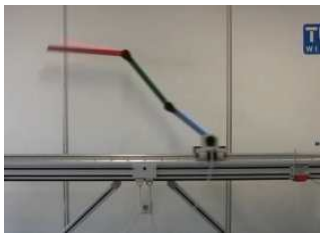
Because humanoid robotics aggregates a bench of challenges that are already difficult to solve by themselves:

- High-dimensional systems: most humanoid robots have at least thirty *Degrees of Freedom* (DoF) and they all need to be actuated simultaneously to generate proper motions.
- Nonlinear dynamics: humanoid robots are made of a succession of rotational joints which introduce nonlinear effects. Thus, any problem involving a cinematic chain is likely to be nonconvex.
- Underactuated systems: humanoid robots do not have motors to directly move their body in the world, they actually are underactuated systems. In fact, balance is challenging and needs to be precisely controlled to be stabilized during a movement.
- Motion redundancy: a task like grasping an object involves up to 7 DoF although humanoid robots achieve it by a coordination of the whole body.
- Robot interaction: humanoid robots move their body in the world by making contacts with the environment, so they need to manage interactions with the environment.
- Constrained problems: Among others, joint limits and friction cones must be respected so the problem is actually a constrained problem.
- Hybrid Systems: To walk, robots also need to change their contacts with the environment so the problem must respect different dynamics at different times.
- Discontinuous/non-differentiable/nonconvex problems: humanoid robots need to choose where they create new contacts and this problem can be discontinuous, non-differentiable and/or nonconvex.
- Compliant control: humanoid robots need to support their own weight and to absorb impacts, so its structure is in comparison lighter and more flexible than other robots. Then, difficulties can appear to control those flexibilities.



(a) Cart-pole

(b) Rotational inverted pendulum



(c) Triple pendulum

Figure 1.3: Examples of experimental underactuated systems.

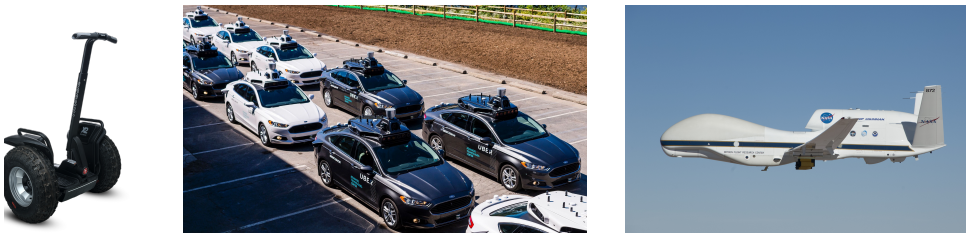
At the end, we have a nonconvex, high-dimensional, overactuated, redundant, constrained, unstable problem with interaction with the environment and a dynamic that changes. Which is a difficult problem to solve... This chapter introduces the contributions of this thesis by exploring the challenges of humanoid robotics and by reviewing the approaches developed by researchers to solve them. Section 1.2 focuses on control of underactuated systems. Next, Section 1.3 describes a method to control redundant systems and Section 1.4 presents techniques to control systems with both underactuation and redundancy. Finally, Section 1.5 analyzes how interactions with the environment (which result in nonconvex problems) can be managed.

1.2 Underactuation

1.2.1 Underactuated systems

Underactuation is not a problem specific to humanoid robotics. Other experimental systems such as cart-poles, rotational inverted pendulums or triple pendulums are underactuated systems (Fig. 1.3). Moreover, they are present in daily life since segways, cars or planes are also underactuated (Fig. 1.4). A system is said underactuated if it is not able to directly command an arbitrary instantaneous acceleration of its state. For instance, a cart-pole (on the left picture of Fig. (Fig. 1.3), the system is an underactuated system because we cannot control the angular acceleration of the pole; a car cannot have a lateral acceleration so it is also a underactuated one (generally a car is said "nonholonom", however a nonholonomic constraint always results in an underactuated system [Tedrake 2009]).

Designing a controller for a fully-actuated system is easier than for an underactuated one. The trajectory does not need to be taken into account, so a controller able to reduce the error between the output and the reference will always converge. The trajectory generated will, of course, change according to the controller parameters but any path can still reach the reference. For those systems, a controller can easily be obtained using feedback linearization [Tedrake 2009]. For underactuated systems, the system is not able to follow an arbitrary trajectory. Therefore, the



(a) Segway (b) Cars (c) Planes

Figure 1.4: Examples of well-known underactuated systems.

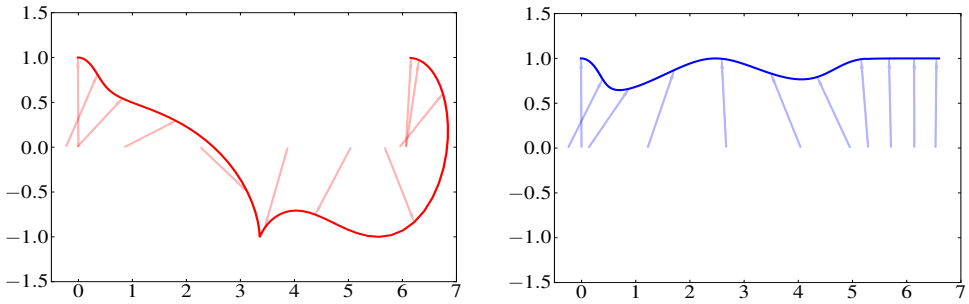


Figure 1.5: Examples of trajectory for a cart-pole system. For both trajectories, the system cannot follow a direct path to the goal position (i.e. the pole always pointing upward) because the system is underactuated.

controller cannot only take into account the current state of the system and reduce the error but needs to follow particular trajectories to get to the goal position (Fig. 1.5).

For instance, for a cartpole system, the pole can be regulated to point upward with a simple state-feedback regulator. However, this regulator cannot directly be used to make the transition from the pole pointing downward to pointing upward because it will not select the right trajectory from the starting state to the end state. To be able to execute this kind of trajectory, the system needs to find a sequence of controls that sways the pole until it is high enough to be able to stabilize it upward. So the future states and controls of the system need to be predicted to know which command needs to be sent right now.

1.2.2 Control of underactuated systems

Different approaches have been proposed to design controllers for underactuated systems. Lyapunov criterion or Poincaré maps are stability criteria that permit to mathematically check if a given system can be stabilized. However, there are no specific ways to obtain a Lyapunov function or to design a controller so this part is left to the ingenuity of researchers. In this thesis we rather consider optimality approaches that offer the premise to systematic methods to synthesize controllers.

Optimal Control formulates the control problem as an optimization problem: objectives, dynamic and system limitations are translated into costs (or rewards) or into constraints in order to formulate an optimization problem:

$$\underset{\underline{x}\in\mathcal{X},\underline{u}\in\mathcal{U}}{\text{minimize}} \int_0^T L(x(t),u(t))dt + E(x(T)) \tag{1.1}$$

subject to

$$\begin{aligned} &\dot{x}(t) = f(x(t),u(t)), \\ &h(x(t),u(t)) \geq 0, \quad h_0(x(0)) \geq 0, \quad h_T(x(T)) \geq 0, \\ &r(x(t),u(t)) = 0, \quad r_0(x(0)) = 0, \quad r_T(x(T)) = 0, \\ &\forall t \in [0,T] \end{aligned} \tag{1.2}$$

where the decision variables are the trajectories in state $\underline{x} : t \in [0,T] \rightarrow x(t) \in \mathcal{X}$ and in control $\underline{u} : t \in [0,T] \rightarrow u(t) \in \mathcal{U}$ (where $\mathcal{X}$ and $\mathcal{U}$ are the state and control spaces, and the underlined symbol is used to differentiate the trajectory from the time value), L represents the integral (or running) cost, E is the terminal cost, f is the system dynamics and the h and r functions represent arbitrary constraints. *Indirect*

Optimal Control uses the *Pontryagin's Maximum Principle* or the *Hamilton-Jacobi-Bellman equations* to construct a controller. The whole state space is analyzed by formulating the problem as a set of differential equations. However, in the general cases the differential equations are too complex to be integrated. Moreover, by analyzing the whole space, those approaches suffer from the *curse of dimensionality* i.e. the analysis complexity grows exponentially with the number of dimensions. Therefore, indirect approaches are mostly used in low-dimensional systems and/or in systems where the differential equations of indirect problems can be simplified enough to be integrated. These techniques are designed to find policies (i.e. controllers which give a command according to the current state) without predicting the future trajectory. Actually, the trajectory prediction is taken into account offline when constructing the policies. They are also the theoretical grounding of the current trend to end-to-end reinforcement learning, made popular by the recent success of deep learning. In these approaches, the policy is approximated offline while only few computations are needed online. In particular, the trajectory that the robot is going to follow is not computed explicitly. While this direction of research is indisputably very ambitious, we do not believe that it is yet realistic to apply on complex dynamic robots, and definitely before years in humanoid robotics. Alternatively, other approaches work by explicitly computing the trajectory that the robot is going to follow, which we are calling here planning. A very classical approach in robotics is named here the *decoupled* approach. This approach solves problems in two steps. First a feasible robot trajectory is computed offline (planning stage), i.e. a valid state trajectory in the sense of the dynamics. Secondly, it finds online the proper controls to track this trajectory and reject perturbations (control stage). This approach is decoupled in the sense that the planning and the control problems are separated into two different stages. In particular, the trajectory is not recom-

puted (or barely recomputed) online. If it is possible, it seems desirable to compute at the same time both trajectory and control. Thus, at the control frequency, a trajectory can be found along with the corresponding controls. This last approach can be regrouped under the name *Predictive Control* or *Preview Control*. To predict trajectories, a model of the robot (and sometimes of the environment) is used directly or indirectly (i.e. via a simulator). So preview control is often called *Model Predictive Control* (MPC). In most cases, preview control is solved using *Direct Optimal Control*. Direct optimal control uses numerical algorithms to find the control commands. The initial *Optimal Control Problem* (OCP) is directly discretized into a finite-dimensional space that computer can handle. Then trajectories and controls can be found by solving the problem using generic numerical optimization algorithms. Contrary to indirect methods which consider the entire space, direct methods only consider at each step a local approximation around a reference trajectory (an initial guess or the result of the previous iteration), and therefore, they do not suffer from integration difficulties and the curse of dimensionality. In practice, the limit between coupled and decoupled approaches is blurred and it is adapted to each application. Due to the relative expensiveness of numerical optimization, MPC is often solved on simplified dynamic problems (e.g. neglecting the propeller dynamic in aerial robotics, only considering the centroidal dynamics and the *Center of Mass*(CoM)/*Center of Pressure*(CoP) "cart-table" model in biped locomotion) and at an intermediate frequency between planning (typically 1Hz or less) and motor control (typically 200Hz or more). For instance, in the next part we will focus on some applications of preview control on humanoid locomotion, where the frequencies of the preview controllers are typically around 10Hz.

1.2.3 Underactuation in Humanoid Robotics: predictive approaches

Because of unilateralism of contact forces, humanoid robots and legged robots in general need to manage their balance to stand-up or walk. If the robot maintains static equilibrium all along its trajectory (i.e. quasi-static walking), the system can be easily controlled. However if we want to achieve a dynamic walk, the torque at ground level is highly constrained by the size of the foot: the system is underactuated. Moreover humanoid robots can be subject to a high number of perturbations (e.g. non-flat surfaces, slippery surfaces, errors in dynamic models, external forces...) that preview control can efficiently reject. Therefore preview control has been shown an effective tool to generate walking patterns for humanoid robots. To be executed at a high enough frequency, walking algorithms use dynamic properties to reduce the number of parameters to control. Those controllers are called *Walking Pattern Generator* (WPG) or shortly *Pattern Generator* (PG). The system is often reduced to a point mass model and can use different stability concepts such as *Zero Momentum Point* (ZMP), *Capture Point* and/or other N-step capturability criteria [Pratt 2006,Wieber 2002]. A very large literature exists for each criterion so this section will mainly focus on some ZMP based algorithms. Kajita [Kajita 2003] first

(a) Quadrotor for cinema (b) Quadrotor for crops (c) Quadrotor toy

Figure 1.6: Quadrotor applications

introduced preview control to generate CoM trajectories that allow humanoid robots to walk dynamically. The system was approximated by a *Linear Inverted Pendulum Model* (also called "cart-table" model). The objectives were set as a quadratic cost function (so the optimization problem is unconstrained). Therefore it could easily be solved as a *Linear Quadratic Regulator* (LQR). Herdt [Herdt 2010b] reformulated the problem to automatically calculate the footstep positions. The problem still used a linear model and a quadratic cost but with additional static variables for the footsteps. To avoid unreachable footsteps and ensure a sustainable walking [Wieber 2008], linear constraints were added on the position of successive feet placements and on the final state. The problem was then explicitly constrained and could not be directly solved with LQR. It was solved using *Quadratic Programming* (QP) instead. Naveau [Naveau 2014] kept the cart-table model but increased difficulty by adding nonlinear constraints (to handle feet orientations and obstacles) and therefore used *Sequential Quadratic Programming* (SQP) to solve its problem. When working with the entire centroidal dynamics (CoM, angular momentum and all contact forces), the dynamic becomes bilinear. With such a dynamic, Capentier developed a WPG [Carpentier 2016] where the problem is formulated as a constrained OCP with general nonlinear dynamics model. This problem was solved using an off-the-shelf *Multiple Shooting* optimal control solver *MUSCOD-II* [Hoffmann 2011] which uses SQP method at its core. As we have seen, preview control solves more and more complex problems. This evolution has been possible thanks to the developments on numerical optimization but also to the improvement of hardware that allows high computation powers in restricted spaces. In the next section, we will see that *Numerical Optimal Control* is a generic method which can be used easily and efficiently on other underactuated robots such as *Unmanned Aerial Vehicles* (UAVs).

1.2.4 Personal contribution: application to aerial robots

Quadrotors have seen a great development during the past decade. It gets from experimental researches to industrial applications (such as cinema, crops analysis or rail-road surveillance) and even to mass consumption goods since it is now sold as a toy for adults and children (see Fig. 1.6). Quadrotors have six DoF (the six

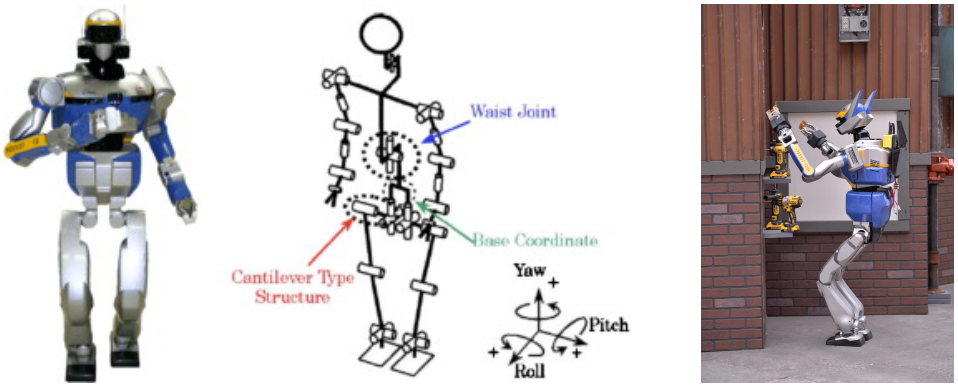
(a) Quadrotor

(b) Quadrotor with pendulum

(c) Aerial manipulator

Figure 1.7: Quadrotor based systems

components of its free-flyer, i.e. three translations and three rotations) for only four actuators (the four propellers). Using its dynamic equations, we can show that quadrotors are fully controllable but are underactuated systems. This underactuation results in constraints, where the horizontal accelerations of the system are constrained by its orientation. The most common approach used for quadrotors is a decoupled approach i.e. motion generation is divided into two separate stages, a trajectory planning stage and a trajectory tracking stage. The dynamic properties of quadrotors allow us to decouple the planning stage and the tracking stage while insuring feasibility. Feasibility is insured using the differential flatness of quadrotors. A system is called differentially flat if there exists a set of linearly independent variables (called flat output) such as all the state and control variables of the system can be expressed as a function of the flat output and a finite number of its derivatives. In the case of quadrotors, we can show that this system is differentially flat and, position and heading can be considered as the flat output [Wang 2012]. The trajectory generation is then greatly facilitated since any smooth trajectory of the flat output can be followed [Mellinger 2011, Faessler 2018]. However, generally speaking, the constraints used to compute trajectories (i.e. constraints on the flat output and its derivatives) do not directly correspond to the original constraints of the system. Therefore, we over-constrain planning to guarantee complete feasibility. For example, linear constraints on trusts become complex when expressed as functions of the flat outputs so these constraints are generally approximated by linear constraints on the flat outputs and its derivatives. While some researchers try to transfer differential flatness to extended systems (e.g. quadrotors with swinging load) or to develop new geometrical properties, we rather believe that MPC is an efficient alternative that can easily be applied on different systems without any analytic development. The first part of Chapter 2 focuses on the usage of preview control to solve different tasks like optimal time reaching (i.e. find the trajectory which reach a position in the shortest among of time), obstacle avoidance or picking; on different aerial robots (see Fig. 1.7). Moreover, we will show that this approach is fast enough to be used online. As an illustration, we will report the



(a) HRP-2 degrees of freedom

(b) example of grasping task (DRC)

Figure 1.8: HRP-2 redundancy

implementation of a teleoperating system to help an operator to safely navigate in a cluttered environment.

While quadrotors have only one solution to statically catch an object (i.e. hover mode over the object), humanoid robots have multiple solutions because they are redundant. In the next section, we will briefly see how redundant systems can be controlled.

1.3 Redundancy

Redundancy refers to the capacity of a system to execute a given motion in different ways. This arises because robots have more DoF than the minimum strictly needed to perform a given motion. For instance, grasping needs up to seven degrees of freedom: six for hand position and orientation and one for the gripper prehension. Therefore, humanoid robots (which have generally over thirty DoF) can be considered as redundant for these tasks (see example with HRP-2 on Fig. 1.8).

1.3.1 Using redundancy to solve multiple tasks

Redundancy implies that there exists a large state space that can solve a given task. Therefore, we can select a particular point in this space to solve another task. This operation can be repeated to solve any number of tasks until there is no degree of freedom left. Based on the above-mentioned, the notion of task hierarchy can be introduced. Let's consider a humanoid robot trying to pick an object. To this end, the motion can be organized in terms of task priority as follows:

- The highest priority task is the safety of the robot i.e. avoiding collision and keeping balance.
- The middle priority task would be the picking task itself.

- Then keeping the object in sight and minimizing torques can be set as low priority tasks.

Each task will be optimized while making sure it will not affect tasks of higher priorities. In this manner, we can make sure that the robot will be safe even if a lower priority task would drive it toward an unsafe state. In our example, the object that the robot needs to pick can be unreachable because it is located too far from the current position of the robot. Thus, the hierarchy will solve at best the picking task while enforcing balance. Equality-constrained optimization can be seen as a two-level hierarchical optimization, where constraints are the high-priority level and cost the low one. However, having a full hierarchy permits to solve problems which would be considered as infeasible for the constrained optimization. Therefore the recurrent question "should a task be set as a cost or a constraint?" is inherently solved. Moreover, if constraints cannot be respected at a certain iteration, a hierarchy can be seen as a way to select which constraints are going to be relaxed [Sherikov 2016] and to make the relaxed constraints converge according to their priority.

1.3.2 Hierarchy of Tasks

The notion of "hierarchy of tasks" was first introduced by [Liegeois 1977] where a two-level hierarchy was used to control the redundancy of a robotic arm. This concept was then developed to integrate any number of task [Siciliano 1991, Baerlocher 1998] and to deal with inequalities [Khatib 1986, Chaumette 2001, Mansard 2008, Kanoun 2011]. Thanks to its robustness and fast computation time, this technique has been widely used to control humanoids for many years [Gienger 2005, Sian 2004, Baerlocher 2004, Sentis 2004, Mansard 2007]. Hierarchy can be obtained at the limit of a multi-objective problem where the relative importance between the objectives is driven to infinity. Consequently it can be approximated by solving an optimization problem where the cost corresponds to a weighted sum of the objective functions. However, when the objectives cannot be solved simultaneously, the solution does not satisfy any of the objectives and corresponds to a trade-off between each of them [Zhao 1994, Faverjon 1987]. Moreover, numerical ill-conditioning is artificially introduced by the large weight ratio, making it difficult to solve when more than five objectives are considered simultaneously. Seminal methods for solving a strict hierarchy are based on pseudo-inverses [Siciliano 1991]. The computation of pseudo-inverse for a task permits to solve the task but also to get a nullspace projector. This nullspace projector can then be used to get a space where lower-priority tasks can evolve. Strict hierarchy problems were later rewritten as a cascade of constrained optimization problems, whose iterative resolutions lead to the hierarchized optimum [De Lasa 2009, Kanoun 2011]. A complete solver implementing a hierarchized active set based on the lexicographic optimality conditions obtained from the cascade was proposed in the case of quadratic costs [Escande 2014]. A generalization to non-quadratic costs remains an open problem [Wieber 2017].

1.3.3 The *Stack of Tasks*

To walk, humanoid robots need to control the trajectory of their feet but also of their CoM. If they want to catch an object or control their balance as a high-wire walker, they also need to control their arms. Moreover, each body of the robot must not enter in collision with the environment or another body. All those tasks can be incompatible but must be solved to generate a feasible motion. Therefore, hierarchy of tasks is a key element to control humanoid robots. Such a hierarchy was implemented several years ago in a control software called the *Stack of Tasks* (SoT). This software allows to solve inverse kinematics [Mansard 2009, Escande 2010] or inverse dynamics [Ramos 2014] as a hierarchy of tasks. SoT has been successfully used on *HPR-2* and *Romeo*, and is being transferred on *Pyrene*. It has been shown to be a reliable tool and is the current whole-body controller used for almost all experiments carried out by the *Gepetto* team. In spite of the fact that hierarchical inverse kinematics or inverse dynamics provide good results to generate motion, it does not use any prediction horizon to anticipate its path. Therefore, algorithms based on this approach can be inefficient if the path needs to avoid several constraints or if the system is underactuated. In the next section, we present approaches to control humanoid robots which are both underactuated and redundant.

1.4 Underactuation and redundancy

As we have seen in previous sections, humanoid robots are underactuated and redundant systems. To walk, a humanoid robot needs to control its whole-body trajectory to move its feet to different positions but also to control the underactuated part of its dynamics i.e. its center of mass trajectory. To cope with this problem, different approaches have been proposed in the literature which can be classified as decoupled, mixed or coupled approaches.

1.4.1 Decoupled approaches

A common approach is to decouple these two problems to solve them separately. We first solve the problem of finding a trajectory for the CoM (and feet) on a two-steps time window, using a simplified model like linearized inverted pendulum [Kajita 2003, Herdt 2010b, Morisawa 2005, Nishiwaki 2009a, Naveau 2014] or a point mass model [Carpentier 2016]. Then, the redundant whole-body problem is solved with hierarchical inverse kinematics (or dynamics) as seen in the previous section. This approach is currently the most used since it allows to reduce a nonlinear problem of high dimension to two linear (for most of the cited papers) problems of smaller dimension. However, several difficulties arise from this decomposition. The solution found by the simplified model can be unfeasible for the whole-body geometry. To overcome this difficulty, we can add constraints/cost to the pattern generator to make sure its result stays in the feasibility set of the whole-body robot. Herdt [Herdt 2010b] used inverse geometry to compute simple linear constraints on

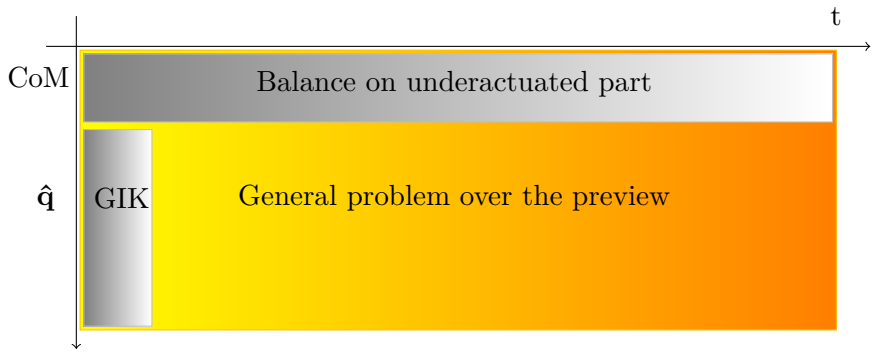


Figure 1.9: Decoupled Approach [Naveau 2016].

successive feet placements. Carpentier [Carpentier 2017a] used random sampling of configurations to generate a cost function able to reveal feasibility of the CoM position with respect to each contact. Moreover, the simplified model does not take into account some part of the dynamics like the orientation of the robot since this variable needs the inertia of the full robot (so its full configuration) to be integrated. Kajita [Kajita 2010] avoided this problem by assuming that the angular momentum was regulated to zero by the whole-body controller. However, it gives rise to constraints that could limit the robot capabilities and there is no certainty that the whole-body controller could fully achieve such regulation. Carpentier [Carpentier 2016] explicitly took into account the angular momentum but minimized it expecting that the result stayed feasible. To take into account geometric and dynamic constraints, Herdt and Perrin [Herdt 2010a] tested a posteriori the results of the final controller (PG and whole-body controller) for different trajectories and constructed bounds on foot placements to ensure that the generated motion stayed in the viability region of their controller. All these techniques are based on the construction of constraints or costs able to reveal feasibility for the complete problem. This thematic will be discussed more deeply later in Section 1.5.

1.4.2 Mixed approaches

In general, the whole-body OCP can be rewritten as a sequence of subproblems. For example, in the case exposed above, the whole problem is decomposed into two subproblems: i) the generation of the walking pattern (CoM) and ii) the tracking of the references (CoM/foot trajectories) with the whole body. Constraints to make sure that the solution of the subproblem is reasonable for the next one in the sequence are needed. Researchers have developed approaches able to bring closer each subpart of the sequence. Those approaches can be seen as a mixed between a fully coupled approaches and decoupled ones.

Sherikov [Sherikov 2014] kept the temporal decomposition of the problem, i.e. the underactuated part of the dynamic system was solved by using a temporal horizon of two steps and the whole-body part only at the current time. However, both problems were included together in the same solver. This allowed the algorithm

to take into account the current whole-body dynamics to compute the Center of Mass trajectory. However this approach could only adapt its plan according to current difficulties of the whole-body and it would not be able to predict any problem that could arise in the future. Dai [Dai 2014] followed the same philosophy but constructs almost a fully coupled problem since the whole-body kinematics was solved over the whole time window. The main difference with the coupled problem was that the dynamics of all joints were not considered and thus the torque limits could not be checked. While computations for the previous approaches were small enough to be executed online on a real robot, for this problem it got closer to fully coupled approaches since it needed several minutes up to several hours. Kajita [Kajita 2003], Nishiwaki [Nishiwaki 2009b] or Naveau [Naveau 2014] also solve the two problems separately but added a *Dynamic Filter* to take into account the whole-body dynamics before sending the commands to the whole-body controller. In [Naveau 2014], the pattern generation problem was firstly solved using a QP algorithm as in [Herdt 2010b], then analytical inverse kinematics was used to calculate the deviation between simplified model and the whole-body one. Using a LQR of the form given by [Kajita 2003] (i.e. without feet repositioning), this deviation was used to generate another CoM trajectory which would be then sent to the whole-body controller. The *Dynamic Filter* can be seen as an additional controller able to merge controls given by the pattern generator and the dynamic constraints of the whole-body. This work was shown to be very efficient in practice because it allows to decrease the tracking error of the ZMP while taking only a small computational time.

1.4.3 Coupled approaches

A fully coupled approach would directly solve the whole-body as a hierarchy of tasks on the whole preview window. The easiest approach to handle hierarchy in an optimal control problem is probably to use a weighted sum cost function. The cost used in the optimization problem corresponds here to a weighted sum of the cost of each task. In that case the strict hierarchical constraints are relaxed, the hierarchy is only enforced by the relative weights between tasks and a strict hierarchy is solved only if the relative weights tend to infinity. This approach was used in [Al Borno 2012], [Tassa 2012] and [Koenemann 2015] to generate whole-body motions on humanoids. However this approach needs long hours of weight tuning and are often restricted to the particular scenarios where weights have been tuned. For instance, the walking motion generated by Al Borno [Al Borno 2012] used a weighted sum of six different cost functions. The corresponding weights needed to be finely tuned to generate the desired movement and to be robust to several situations. Current algorithms do not have the ability to intuit the possible effects of each cost function and to characterize the resulting movements. Therefore, weight tuning is usually done manually with trials and errors. This operation can be extremely long and difficult because the interaction between each task is complex and difficult to tune without already having a functional movement.

By using hierarchies, we can greatly facilitate this step since the hierarchy will manage interactions between tasks itself. The only operation left is to organize task priorities. Moreover, hierarchies are more robust to different scenarios. While a secondary task with small weight but high cost can destabilize a robot controlled with a weighted sum, it cannot do it with a hierarchy of tasks. A hierarchical optimal control problem can be mathematically formulated as follows:

$$g_j^* = \underset{\underline{x}\in\mathcal{X},\underline{u}\in\mathcal{U}}{\text{minimize}} \int_0^T L_j(x(t),u(t))dt + E_j(x(T))$$

subject to

$$g_i(\underline{x},\underline{u}) = g_i^*, \quad \forall i < j$$

and constraints (1.2)

where g_i for $i = 1..K$ are the cost functions of the tasks ordered by their priority. By using the same philosophy as in the SoT (i.e. by using pseudo-inverses and nullspace projectors), Del Prete [Del Prete 2015] showed it was possible to solve this problem with strict hierarchical constraints. The next section presents more precisely the work that has been done to develop this algorithm by exploiting the temporal sparsity of optimal control problems.

1.4.3.1 Differential Dynamic Programming

Differential Dynamic Programming (DDP) is a single shooting resolution scheme for optimal control problems. It has been heavily studied and improved during the recent years because it intrinsically exploits the sparsity of optimal control problems. It was shown to be a very powerful tool to quickly solve difficult problems [Tassa 2012, de Crousaz 2015] while the cost of developing the solver remains reasonable (compared to developing a generic sparse nonlinear solver). Some work focused on fasting computation of the algorithm by using only a linear approximation of the dynamic (iLQR) or by exploiting the square-root form [Geoffroy 2014]. Others tried to overcome the limitations of this approach. Tassa [Tassa 2014] and Xie [Xie 2017] showed that we can modify the DDP algorithm to be able to solve constrained problems (equality and inequality constraints). Moreover, Pellegrini [Pellegrini 2017] showed that it is possible to modify the algorithm to solve a *Multiple Shooting* problem instead of the usual *Single Shooting* one. Recent works showed it is also possible to solve hierarchies of tasks, they are presented in the next sections.

1.4.3.2 Hierarchical Differential Dynamic Programming (HDDP)

The work of Romano and Del Prete [Romano 2015] focused on building a sparse hierarchical algorithm. This algorithm is based on the DDP algorithm and permits a resolution with a linear complexity with respect to the size of the preview window (more precisely the number of time steps) and to the number of tasks. The hierarchy is enforced by adding equality constraints for lower priority tasks using

nullspace projectors. They showed that contrary to hierarchies with a weighted-sum cost, HDDP is able to handle a high number of tasks without suffering from ill-conditioning. However, this algorithm was not designed to handle constraints or costs on controls (cost on controls was only usable as the lowest-priority task) so the algorithm is limited to terminal costs. The next section briefly presents an extension of HDDP where our main purpose was to design an algorithm able to manage more complex cost functions.

1.4.4 Personal contribution: Regularized Hierarchical Differential Dynamic Programming (RHDDP)

Even if hierarchies with integral cost functions were not studied in [Del Prete 2015] nor [Romano 2015], integral costs are important for MPC. Integral costs permit to reach objectives without specifying duration. The system will reach the objectives as fast as possible. And to avoid infinite actuations, we generally add an integral cost on controls. This additional cost is called *task regularization* in this work. Solving a hierarchy of tasks with task regularization using a weighted sum would raise a problem since the weights cannot be tuned to have at the same time i) different weights for each task to have a hierarchy of tasks ii) the same ratio between the weights of each task and the one of regularization. Therefore, either the hierarchy will not be respected or the low-priority tasks will not be achieved because the regularization is too high for them. Hierarchical algorithms would have a clear advantage: they could enforce a hierarchy while allowing a proper regularization for each task. Therefore, we have developed a new version of the HDDP algorithm called *Regularized Hierarchical Dynamic Programming* (RHDDP). While regularization is usually an ad hoc feature, RHDDP was designed by explicitly taking into account the regularization when constructing the problem. A complete study of this algorithm is given in Chapter 3. In chapter 3, RHDDP is mainly applied on non-humanoid robots for two reasons. The first one comes from the simulator used. The simulator used is *Mujoco*, a physics engine to compute robot dynamics and contacts [Todorov 2014]. The contact model used does not exactly solve the contacts as hard constraints but uses approximation to quickly and efficiently get viable results. Even if those small imperfections were completely invisible for traditional optimal control, those imperfections added to the hard hierarchical constraints used in RHDDP were enough to make the system completely constrained by the first task. Therefore, fine tuning were needed to separate the robot dynamics and the noises coming from the contact model. The second one comes from the intrinsic inability of local optimization to discover new contacts with the model used. This problem is discussed in detail in the next section.

1.5 Interaction with the environment

Legged robots move their body by using their legs and/or arms to create reaction forces with the environment. To move further than their limbs, they need to break off contacts and create new ones. Each change of contact allows the robot to change its dynamics to go further. This change in the dynamics generally results in a highly nonconvex cost function with multiple local minima. If we use local optimization, it is unlikely that the robot will create new contact because the robot is like blind and only knows it could push on the ground when its foot is already on it. The next sections develop two different approaches to solve this nonconvex problem. The first one relies on a contact planner to solve separately the nonconvex part relative to contacts, the second one develop different solutions to handle nonconvexity directly in the optimization problem.

1.5.1 Decoupled approach

While in previous sections, the decoupled approaches were mainly designed to reduce computational time, here decoupling comes from the fact that the OCP is nonconvex and therefore can be difficult to solve with optimal control approaches. An alternative method is to directly specify the changing dynamics in the optimization problem. This approach corresponds to all the *Walking Pattern Generator* seen in Section 1.2.3, where gaits are predefined. To be consistent with the dynamics, we also need to add the contact constraint on foot position and orientation. If the floor is even, the contact constraint can be integrated in the optimization problem: the optimization solver is able to choose the optimal foot positions and orientations on the surface [Herdt 2010b]. If the floor is uneven, the optimal solver will once again get trapped into local minima and will not be able to select the best contact surface.

1.5.1.1 *Loco3d*

The *Loco3d* project is a project developed in the *Gepetto* team that aims to build a full pipeline to generate locomotion movements in a complex environment [Carpentier 2017c]. Such pipelines have been also seen in the case of a quadruped robot for the *DARPA Learning Locomotion* project [Kalakrishnan 2010, Zucker 2010]. To overcome the different difficulties of humanoid locomotion, the locomotion problem is solved following a decoupled approach i.e. by constructing a sequence of planners/controllers as follows (Fig. 1.10):

- Planning of an approximate path for the center of the robot.
- Planning of the contacts.
- Planning of a precise trajectory for the center of mass.
- Control of all motors.

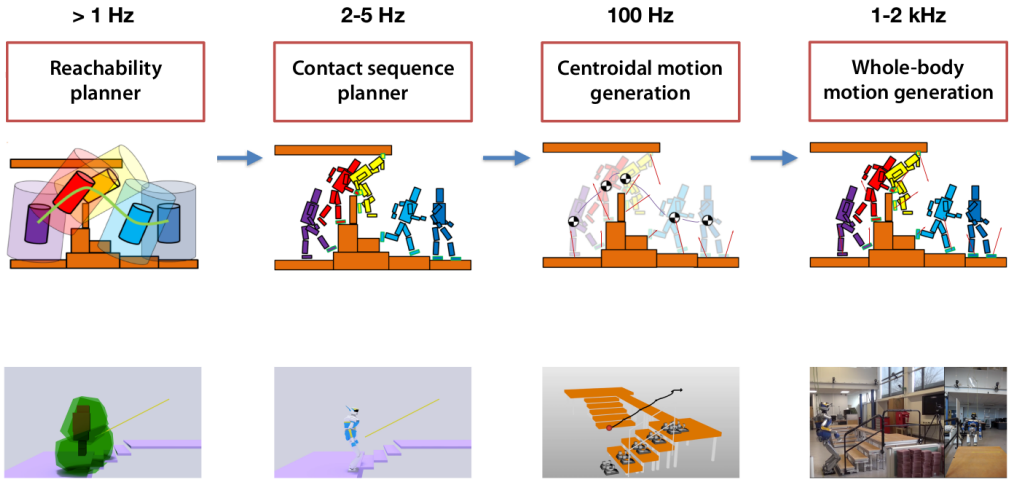


Figure 1.10: *Loco3d* pipeline [Carpentier 2017c, Tonneau 2016]

Therefore in the *Loco3d* project, we use a planning algorithm to select not only the gaits but also the contact surfaces before using the optimal controller. The contact planner used was developed by Tonneau [Tonneau 2016] and is able to quickly find a sequence of poses where the robot is in contact with the environment, in static equilibrium and without collisions. This problem is very complex if we directly consider all variables, but the resolution can be simplified by decoupling it into two stages. The first stage selects a guide trajectory for the center of the robot, the second one selects gaits and contact points to compute a sequence of whole-body poses. As we have quickly seen in Section 1.4.1, with a decoupled approach we need to insure feasibility of the plan computed by the reduced model on the real one. To do that we can construct intermediate model so-called *proxy* [Zaytsev 2015] revealing the feasibility or its probability. In the next sections, we will see two approaches to integrate these proxy models to high-level controllers: *proxy constraints* or *proxy costs*.

1.5.1.2 Proxy constraints

To ensure feasibility, we can add constraints to the planner so the plan stays in a feasible region. As we have seen, in his WPG Herdt used inverse kinematics [Herdt 2010b] or posterior tests [Herdt 2010a] to calculate the feasible region of footsteps. Constraints were then added to the optimization problem to keep the center of the foot in the feasible region. The same approach is often used for footstep planning. Perrin [Perrin 2012] and Orthey [Orthey 2013] both constructed a model of collision for step transitions. Then this model could be used to prune branches of the search tree of an $A*$ algorithm or to validate connections of a *Rapidly-exploring Random Tree* (RRT). Tonneau used the same philosophy and set constraints on the guide trajectory [Tonneau 2015]. To have a feasible trajectory,

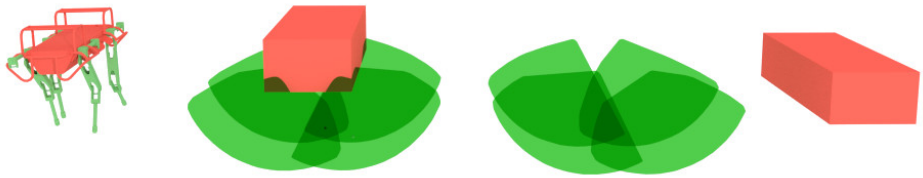


Figure 1.11: Collision space and reachable space [Tonneau 2015].

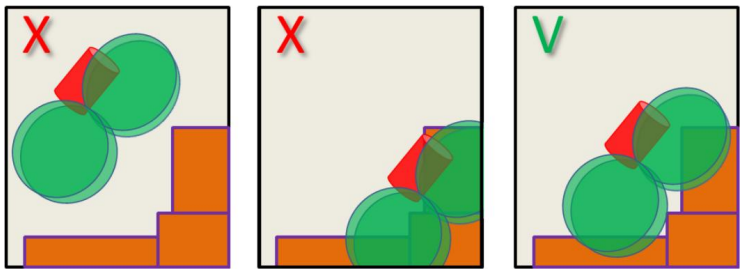


Figure 1.12: Validation of the guide trajectory [Tonneau 2015].

we want the robot to be close to obstacles to be able to generate contacts, but not too close to avoid collisions. By sampling configurations for each robot limbs, Tonneau was able to construct shapes corresponding to the reachable region of each end-effector. Moreover, a shape surrounding the main body was selected to avoid collisions (Fig. 1.11). A guide trajectory could then be planned by selecting the robot positions where obstacles were in collision with the shapes corresponding to reachability (green shapes, Fig. 1.12), but not in collision with the main body (red shape, Fig. 1.12).

1.5.1.3 Proxy costs

Reformulating constraints as additional cost terms might be useful in several situations. Using costs instead of constraints can be useful because it puts a gradation between feasible and infeasible regions. The extra cost terms can be seen as enforcing robust control by pulling the system inside safe regions instead of accepting solutions on the boundaries of the feasible set. Del Prete [Del Prete 2016a] showed that using costs instead of hard constraints can be beneficial because it can still keep the control in a feasible region during a standard walk but it also allows the control to reach dangerous regions when needed (for instance to get back after a perturbation). Usually, the wide range of states are not explored to determine feasibility and only an under-approximation is used so even a unfeasible control for the proxy constraints can actually be feasible under some conditions. Usage of a proxy cost allows to relax those constraints when needed. Moreover, a proxy cost can give more information than constraints by giving information on the robustness of the solution. For instance, Carpentier [Carpentier 2017a] randomly sample configura-

tions to retrieve occupancy measures of the CoM - end-effector relative positions. This measure allows to get the feasible set (occupancy superior to 0) but also to avoid regions where self-collisions often occur. Moreover, a region with a high probability will tend to have a lot of possible limb configurations for the same relative position. Other examples of proxy costs can be found in the *LittleDog Learning Locomotion Project* [Kalakrishnan 2010, Zucker 2010, Kolter 2011]. In those papers, a small quadruped needed to go through an uneven terrain. They first subdivided space and scored each part according to the probability of collision and slippage, using a learned model. Then, they used this scored map to select the less difficult path.

1.5.1.4 Personal Contribution: learning stable poses

To select the robot path, the current state of *Loco3d* uses proxy constraints based on reachability [Tonneau 2016] while several approaches from the *LittleDog Learning Locomotion Project* use proxy costs based on learned data. The first one is used to approximate non-collision with the environment and reachability of the end-effector to the environment. The second one mainly adds consideration on granularity of the surface to avoid slippage. Both approaches show great capabilities, however, they are applied on many-contact problems. For the case of humanoid robots moving with their feet only, the robot creates contacts with the environment using two end-effectors at the same time at maximum. Therefore, the stability criterion, which is not considered in those approaches, becomes a highly important criterion here. The work we have done follows the same philosophy as in [Kalakrishnan 2010, Kolter 2011] but tries to focus on the stability criterion to find a walkable path. Contrary to these works, the balance criterion greatly depends on the position of both feet so we cannot decouple each limb to compute the proxy cost. In this work, we also use the framework of machine learning to get a score for each position. However the data are not scored using expert demonstrations but using heuristics like robustness with respect to a particular foot placement or upper body configuration. A complete description of the algorithm and its results is shown in Chapter 4. In this part, we have seen how we can construct a cascade of controllers

(and ensure its viability) to solve a nonconvex problem. In the next section, we will see different approaches to solve a nonconvex problem without decoupling it.

1.5.2 Coupled approach

To solve interaction with the environment along with the whole dynamic problem, we need to be able to solve a nonconvex OCP. A problem can be considered as nonconvex for several reasons: if the cost is nonconvex but also if the feasible domain is nonconvex or if the (equality) constraints are nonlinear. A simple example of nonconvex problem is the case of environments with obstacles, where deterministic optimal control will often get stuck in local minimum (Fig. 1.13). And this problem will occur if obstacles are considered as constraints (resulting in a nonconvex feasible

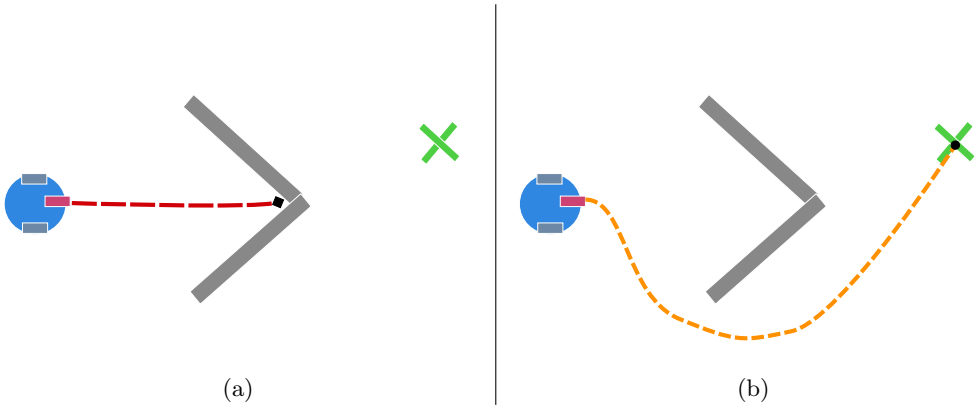


Figure 1.13: Local minima in cluttered environment: examples with a mobile robot. In both cases, trajectories are only local minima. In 1.13a, the robot is stuck in a dead end. In 1.13b, the robot is able to reach the goal position but this not the global optimum since a shorter path exists (on the other side of the obstacle).

set) just as well as costs (resulting in a nonconvex cost function). Direct OCPs using gradient or Newton based optimization algorithms construct local approximations of the cost, of the system dynamics and of the constraints, to iteratively decrease the cost (or increase a reward). By using only local approximation, optimization does not have any information about the global shape of the function. While in motion generation most of our problems are nonconvex, the solvers of the state of the art boil down to convex optimization. Therefore, those algorithms are only able to find local minima and their convergence rates are slow on nonconvex regions. The following sections reveal several approaches that can be used to this problem.

1.5.2.1 Global optimizer

Boyd shows in his course on *Convex Optimization* [Grant 2014] that it is sometimes possible to formulate nonconvex problems as convex ones but that there is no guaranty that such a transformation always exists. In his book, Boyd wrote "The art in local optimization is in solving the problem (in the weakened sense of finding a locally optimal point), once it is formulated. In convex optimization these are reversed: The art and challenge is in problem formulation; once a problem is formulated as a convex optimization problem, it is relatively straightforward to solve it" [Grant 2014]. A generic reformulation is always possible by rewriting any static optimization problem as a problem over the space of measures, hence making it linear, hence convex [Lasserre 2001]. However, the cost to pay for this formulation is to work with a basis in the space of measures, typically moments of the measure. The matrix of moments should be positive definite to have a proper measure, which ends up to a semi-definite problem. However, such reformulation becomes computationally intractable for polynomials of degrees superior to five and dimensions superior to three. Other ad hoc formulations are always possible. For example, Deits [Deits 2014] used *Mixed-Integer Optimization* to solve footstep

planning over a nonconvex feasible set. However, mixed-integer optimization is a NP-hard problem so its application in real time can be difficult.

1.5.2.2 Stochastic optimizer

When it is impossible to achieve a convex formulation of a given problem, we boil down using heuristics combined with stochastic sampling [Hamalainen 2015, Doncieux 2014]. These algorithms are able to get out from local minima by extending or modifying a range of candidate trajectories and therefore they can explore a much larger space. However, optimality conditions can barely be extended further than second order conditions, like in convex programming. Motion planning relies on such heuristic-sampling search (often random) with *Probabilist RoadMap* (PRM) or *Rapidly-exploring Random Tree* (RRT) algorithms. With the complexity induced by robots, it is generally accepted that random search must be achieved, if possible, on a simplified dynamic (e.g. static) while only considering geometric constraints. Kinodynamic planning is, yet, too difficult to be of real practical interest.

1.5.2.3 Reshaping the cost function (cost engineering)

Instead of randomly exploring the whole space, we could also drive the optimization toward a better region. Regularizations or Gauss-Newton approximations change the local approximations of cost functions to stabilize the algorithms or reduce computational times. However, we will see here some techniques that change the global shape of cost functions to make the algorithms converge faster and/or toward better solutions.

Adding costs To get a more convex problem, we can artificially add some terms in the cost function. For instance, for a picking task, we often add a cost on the velocity of the gripper during grasping (see Section 2.5.4). This cost can be seen as artificial since, even with a cost on position only, the (globally) optimal one is a solution where the gripper stands still at the goal position. However, the cost on velocity gives a more convex problem and decreases the number of iterations needed to find a good solution. Another example is adding a cost between the configurations (along trajectories) and a reference one. This can be seen as a way of controlling the degrees of freedom left free but also as a way to select a high manipulability configuration, away from bounds and therefore less prone to local minima.

Homotopic methods We can also adjust the cost function during the search. For instance, the *Contact-Invariant Optimization* (CIO) [Todorov 2011] which uses a relaxation of the complementary constraint of contacts [Posa 2016], can be seen as "a way of reshaping a highly discontinuous and local-minima-prone search space of movements and contacts, into a slightly larger but much better-behaved and continuous search space that enables optimization strategies to find good solutions" [Mor-

datch 2012]. In that case, the problem is first augmented and then converges back to the original problem. This augmentation permits to select the most promising basin of attraction. When formulating the OCP as a multiple-shooting problem, the importance of the continuity is often variably adjusted with a similar philosophy [Diehl 2001]. Tassa [Tassa 2012] and Mordatch [Mordatch 2012] both used this complementary constraint relaxation. The main difference between the two approaches is how they converge back to the original problem. Mordatch solved a problem over a fixed time window and its problem converged back by smoothly reducing the relaxation along iterations. Tassa solved the same problem but on a receding horizon (i.e. MPC), where controls were computed with the relaxed model but executed on a stiffer one. Tassa only relied on robustness and adaptiveness of optimal control to get viable results and therefore had much rougher controls with high impacts.

1.5.2.4 Initial guess

Correctly initializing the candidate solution optimized by the numerical solver is a very important task and can lead to very different results: from unfeasible problems, divergence of the solver or hours of computation, to only few seconds or minutes if rightly initialized. Moreover, we can find global minima if we directly initialize the optimal control problem within its basin of attraction [El Khoury 2013].

Initial guess via heuristics In Section 2.5.2.1, we show that in some cases, even simple heuristics can lead to other optima and therefore a pre-existing knowledge of the solution can be used to initialize optimization. As we will see in the next chapter, an advantage of *Multiple Shooting* methods is that it can be initialized with control and state, and the trajectory does not need to be consistent. Therefore, simple heuristics like static or quasi-static trajectories can be used for initialization.

Initial guess via planning A decoupled approach would be to rely on planning to find a path, which will be then optimized by the optimal control. In that case, the main objective of the planning algorithm is to select the best topology or basin of attraction [Bhattacharya 2016] (Fig. 1.14). For instance, [Bouffard 2009] coupled a planning algorithm and multiple shooting optimal control to find collision-free trajectories. While in *Loco3d* the optimization is restrained by the planner choices (e.g. contact positions), here the optimization is not and can modify the plan computed by the planner. Since planning algorithms only use a simplify approximation of the cost, the output of the planning will not necessarily correspond to the best basin of attraction. The planning algorithm (or the heuristics) usually works on configurations although optimal control generally works on positions, velocities and accelerations or forces/torques so the local minima revealed by the derivatives cannot be treated by the planning part.

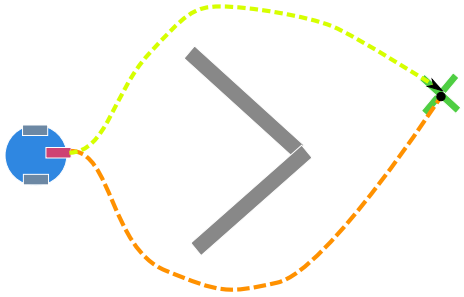


Figure 1.14: Finding the best topology: examples with a mobile robot. Examples of trajectories in the two different topologies of the problem.

Initial guess via learning The experience earned during previous experiments can be used to directly initialize the optimization close to the optimal solution. We can use machine learning techniques to store in an efficient way the results obtained by our optimal control solver at previous experiments. Deep learning has shown the capability to learn complex motions [Mordatch 2015,DeepMind 2017]. However, the trajectories generated by the networks are still too "clumsy" to be directly used on a real robot. Therefore, we believe that machine learning must instead be associated with a model-based controller. Mansard [Mansard 2018] showed the pertinence of such approach by using neural networks to initialize a model predictive controller. The neural network was trained with examples of trajectories generated offline with the controller itself. Moreover, a sampling-based planning algorithm was used to generate new trajectories and possibly find global minima. Data generation and learning were performed in an iterative way: the optimal controller generated the data that were used to construct a model and this model was then used to improve the results of the controller.

1.5.2.5 Personal contribution: learning initial guess on aerial robots

From our initial work on quadrotors [Geisert 2016], we tested the approach of Mansard on different aerial robots. The results are shown in the second part of Chapter 2. While the approach developed in Section 1.5.1.4 and here are both based on *Machine Learning*, the paradigms are quite different. Section 1.5.1.4 uses learning to get a score from the robot state, i.e. learning a function from a relatively high-dimensional space to the real numbers. Instead, here we learn a function from a low-dimensional space (initial and final states) to a high-dimensional space (state/control trajectories). The additional difficulty here is that the output variables are correlated (i.e. trajectories need to respect the dynamic model) and therefore even a "good" approximation could lead to inconsistent results that could destabilize the optimal solver. This is maybe the main reason why we did not succeed yet to generate walking motions with this approach. This chapter has introduced the topics developed in this document by studying some of the challenges

that humanoid robotics is facing. Next chapter shows that the techniques presented here, i.e. *Model Predictive Control* and *Machine Learning*, are versatile and can easily and efficiently be used on other systems such as aerial robots.

1.6 Contributions Summary

Journal

- *Regularized Hierarchical Differential Dynamic Programming.* M Geisert, A Del Prete, N Mansard, F Romano, F Nori. IEEE Transactions on Robotics (TRO 2017).
  pdf: `https://hal.archives-ouvertes.fr/hal-01356992/file/201604_hddp.pdf`
  video: `https://www.youtube.com/watch?v=BqAOmDvlB2Q&t=3s`

Conference

- *Airbus/future of aircraft factory HRP-2 as universal worker proof of concept.* O Stasse, A Orthey, F Morsillo, M Geisert, N Mansard, M Naveau, C Vassallo. IEEE International Conference on Humanoid Robotics (ICHR 2014).
  pdf: `https://hal.archives-ouvertes.fr/hal-01122476/file/14-ichr-airbus.pdf`
  video: `https://www.youtube.com/watch?v=FpMlrfhZCRs`

- *Trajectory Generation for Quadrotor Based Systems using Numerical Optimal Control.* M Geisert, N Mansard. IEEE International Conference on Robotics and Automation (ICRA 2016).
  pdf: `https://hal.archives-ouvertes.fr/hal-01213392/file/root.pdf`
  video: `https://www.youtube.com/watch?v=hcFK32C-bxs`

- *Multi-contact Locomotion of Legged Robots in Complex Environments – The Loco3D project.* J Carpentier, A Del Prete, S Tonneau, T Flayols, F Forget, A Mifsud, K Giraud, D Atchuthan, P Fernbach, R Budhiraja, M Geisert, J Solà, O Stasse, N Mansard. RSS Workshop on Challenges in Dynamic Legged Locomotion (RSS Workshop 2017).
  pdf: `https://hal.laas.fr/hal-01543060/file/loco3d.pdf`

- *Using a Memory of Motion to Efficiently Warm-Start a Nonlinear Predictive Controller.* N Mansard, A Del Prete, M Geisert, S Tonneau, O Stasse. IEEE International Conference on Robotics and Automation (ICRA 2018).
  pdf: `https://hal.archives-ouvertes.fr/hal-01213392/file/root.pdf`
  video: `https://www.youtube.com/watch?v=CbyCa7aeC7k`

ïż£

Chapter 2

Trajectory Generation for Quadrotor Based Systems

The recent works on quadrotors have focused on more and more challenging tasks with increasingly complex systems. Systems are often augmented with slung loads, inverted pendulums or arms, and accomplish complex tasks such as going through a window, grasping, throwing or catching. Quite often, controllers are designed to accomplish a specific task on a specific system using analytic solutions, so each application needs long preparations. On the other hand, the direct multiple shooting approach is able to solve complex problems without any analytic development, by using off-the-shelf optimization solvers. In this first part of this chapter, we show that direct multiple shooting is able to solve a wide range of problems relevant to quadrotor systems, from online trajectory generation for quadrotors, to going through a window for a quadrotor-and-pendulum system, through manipulation tasks for an aerial manipulator. In the second part, we show that performances can be improved by constructing an approximation of the optimal solution and use it to initialize the optimization solver. The algorithm presented in this part allows to construct such approximation without any analytic development and rather relies on automatic methods from optimal control, motion planning and machine learning.

Trajectory Generation using Numerical Optimal Control

2.1 Introduction

Many works have been proposed to control autonomous quadrotors. We focus on two subsets of works: with oscillating loads and with manipulation capabilities. During transportation missions, loads are often carried by means of a cable to avoid aerodynamic perturbations. Various controllers have been designed for this under-actuated system. For tasks involving only transportation, a typical approach is to reduce the energy of the load to limit its swing motion. Swing-free motion can be found using dynamic programming approach [Palunko 2012, Zameroski 2008] or reinforcement learning [Faust 2013, Palunko 2013]. In [Sreenath 2013, Tang 2014] the differential flatness of the system was used to build a cascade of controllers capable of controlling the load position and attitude. In [de Crousaz 2015], dynamic programming was used to calculate aggressive trajectories. Using this technique, authors showed that their method is able to find trajectories to go through a window with a system quadrotor and slung load. However the problem of finding how the quadrotor and the load have to pass through the window is solved beforehand and given to the algorithm in the form of waypoints. In [Brescianini 2013], trajectory of quadrotors throwing and catching an inverted pendulum were generated offline using direct optimal control then executed on the real system using a LQR controller. For tasks such as screwing, assembling or other manipulation tasks, robotic arms can be mounted on the flying vehicle. In [Orsag 2014], a simple symmetric manipulator was used so quadrotor and manipulator controllers can be built separately. When the manipulator becomes more complex, the quadrotor controller has to take into account movements of the arm [Jimenez-Cano 2013, Ghadiok 2011, Thomas 2013]. Once controllers are designed, the problem shift to the trajectory generation. In [Arleo 2013], *Inverse Kinematics* was used to determine trajectories of the joints and the quadrotor from the end-effector trajectory. In most recent works, optimal control was used to generate online picking trajectories [Garimella 2015]. The algorithm used is able to exploit the full system dynamics and generate simultaneous trajectories for the quadrotor and the arm but, because it uses an algorithm similar to *Differential Dynamic Programming* (DDP), it cannot easily handle obstacle constraints. In the expectancy to solve problems involving more and more complex tasks on aerial systems, **we believe that analytic solutions are too restraining and too difficult to exhibit on a generic manner**. An alternative approach, that has not been extensively explored yet, is to rather rely on the field of direct optimal control and numerical optimization, to numerically approximate a valid control sequence, that would then be easily adapted to changes in both the task to achieve and the dynamics of the robot.

The contribution of this chapter is to show the interest of using one of these numerical approaches to design versatile and efficient behaviours on complex aerial vehicles (aerial pendulum and aerial manipulator). We propose a complete formulation to numerically compute an optimal movement. We then report an extensive benchmark of the capabilities of numerical solvers to discover efficient trajectories around obstacles and control these trajectories despite model uncertainties. We also demonstrate that even if these algorithms can look computationally heavy, they can be used for online trajectory generation i.e. in a *Model Predictive Control* (MPC) strategy. Next section presents the optimal control problem and more specifically, the direct multiple shooting formulation used to solve the problem. Section 2.3 describes the three robot models used for the benchmarks (*i.e* quadrotor, quadrotor with pendulum and quadrotor with robotic arm). Section 2.4 presents the initial guess, the model of obstacles and the tools used to generate our results. The last section shows the results obtained in simulation for a wide range of tasks.

2.2 Optimal Control

In this section, we briefly recall the framework of optimal control and define explicitly the formulation of the numerical problem that is used for quadrotors. Meanwhile, we justify why we have selected this particular form among all the possibilities found in the literature.

2.2.1 Indirect and Direct Approaches

In this chapter, trajectories of different quadrotor-based systems are generated using optimal control solvers. The generic optimal control problem can be expressed as follows:

$$\underset{\underline{x},\underline{u}}{\text{minimize}} \int_0^T L(x(t),u(t))dt + E(x(T))$$

subject to

$$\begin{aligned}
&\dot{x}(t) = f(x(t),u(t)),\\
&h(x(t),u(t)) \geq 0, \quad h_0(x(0)) \geq 0, \quad h_T(x(T)) \geq 0,\\
&r(x(t),u(t)) = 0, \quad r_0(x(0)) = 0, \quad r_T(x(T)) = 0,\\
&\forall t \in [0,T]
\end{aligned}$$

where the decision variables are the trajectory in state $\underline{x} : t \in [0,T] \rightarrow x(t) \in \mathcal{X}$ and in control $\underline{u} : t \in [0,T] \rightarrow u(t) \in \mathcal{U}$ (where $\mathcal{X}$ and $\mathcal{U}$ are the state and control spaces, and the underlined symbol is used to differentiate the trajectory from the time value), L represents the integral (or running) cost, E is the terminal cost, f is the system dynamics and the h and r functions represent arbitrary constraints. Two very different approaches may be considered to solve this problem: the indirect one and the direct one. The indirect one changes the problem into an integration (ODE or DAE) problem using the Pontryagin's maximum principle or the Hamilton-Jacobi-Bellman equation. The resulting problem is a differential equation which

unfortunately is often too complex to be integrated as is. When it is possible, this approach provides a complete (and often computationally cheap) solution to the problem. However, this type of approach is usually applied on a specific system and/or task so the differential equation can be simplified enough to be integrated. The direct approach directly solves a discretized approximation of the nominal problem using numerical optimization techniques. It has several advantages: it works directly on the problem so the problem does not need to be reformulated; it is solved by generic solvers; and it can often hope to directly adapt a solution to variations of both the system dynamics and the tasks. The trade off is that the resulting optimization problem is often nonconvex, hence only a partial solution can be found. Our work only focuses on the direct optimal control.

2.2.2 Direct Approaches

The discretization of the nominal problem results in a redundant set of decision variables, $\underline{x}$ and $\underline{u}$, constrained by the system dynamics (redundant in the sense the $\underline{x}$ directly arises from $\underline{u}$). Three different formulations with different properties are typically considered to handle this redundancy:

- Single shooting method integrates controls from an initial (known) state to get the whole trajectory using ODE/DAE solvers and optimizes the cost function over controls. This formulation is interesting because of its simplicity and, since the resulting optimization problem has a lower number of degrees of freedom and no additional constraints, it easily deals with systems of large dimensions [Tassa 2012].

- In collocation method, the cost function is optimized over both states and controls. The relation between states and controls, i.e. the dynamic equation, is discretized and set as constraints for the optimization problem. One state decision variable is added to each integration node (hence the collocation name). In practice, those additional constraints force the solver to solve many inverse dynamic problems, which is costly in general, but allow this method to be initialized with state trajectories, to have fast convergence and to better deal with unstable systems [Diehl 2005].

- Multiple shooting combines advantages of collocation and single shooting by using the same formalism as collocation but with ODE/DAE solvers to integrate the dynamic equation of the system along (longer) time intervals instead of discretizing it. The variables are then a full control trajectory $\underline{u}$ along with a sparse number of state variables $x_1, ..., x_J$ called shooting nodes. These nodes avoid the divergence phenomenon encountered in single shooting, and despite an increase in the number of decision variables, speed up computation [Diehl 2005]. The shooting variables $x_1, ..., x_J$

In this work, we consider quadrotor-based systems. As the system keeps a reasonable state/control dimension (4 to 10 control variables and 12 to 24 state variables),

the resulting optimal control problem has an acceptable dimension. Single shooting would be able to calculate each step very quickly. However, because of the nonlinearity and the instability of the system, the iterative numerical algorithm would only be able to progress slowly and risks to diverge. Multiple shooting and colocation would probably give similar quality results. But thanks to its design where the problem parametrization and the integration of the dynamic are separated, multiple shooting is more flexible and much faster (which is essential for *Model Predictive Control*). Additionally, the relative stability of quadrotors (compared to rockets or humanoid robots) allows to have only few shooting nodes although nonlinearities impose thin steps to integrate the dynamic equations. Therefore, the different tasks presented in this chapter are solved using the direct multiple shooting approach.

2.2.3 Direct Multiple Shooting

In multiple shooting methods, the system trajectory is cut into small time intervals that correspond to shooting intervals. A set of $N-1$ state variables corresponding to the starting point of each shot is introduced:

$$x(t_j) = \{x_j\}_{j=1..N}, \quad x_j \in \mathcal{X}, \quad j = 0, ..., N-1.$$

Thus, integration of the dynamic equation $\dot{x}(t) = f(x(t), u(t))$ will give a piecewise continuous function with discontinuities at each shooting node. We denote it $x(t; x_j, \underline{u}_j)$ for $t_j \leq t < t_{j+1}$, where $\underline{u}_j$ are the control time functions over each shooting interval. Additional constraints are added to force continuity at the shooting nodes. Similarly to states, constraints are discretized over time:

$$\begin{aligned} &h(x_j, u(t_j)) \geq 0, \\ &r(x_j, u(t_j)) = 0, \quad j = 0, ..., N-1. \end{aligned}$$

Therefore, the optimal control problem becomes:

$$\underset{\{x_j, \underline{u}_j\}_{j=1..N-1}}{\text{minimize}} \sum_{j=0}^{N-1} l_j(x_j, \underline{u}_j) + E(x_N)$$

subject to

$$\begin{aligned} x_{j+1} - x(t_{j+1}; x_j, \underline{u}_j) = 0, \qquad & j = 0, ..., N-1, \\ h(x(t_j), u(t_j)) \geq 0, \quad h_0(x(0)) \geq 0, \quad & h_T(x(T)) \geq 0, \\ r(x(t_j), u(t_j)) = 0, \quad r_0(x(0)) = 0, \quad & r_T(x(T)) = 0. \end{aligned} \tag{2.1}$$

where

$$l_j(x_j, u) := \int_{t_j}^{t_{j+1}} L(x(t; x_j, u(t)), u(t)) dt$$

Another advantage of multiple shooting with respect to single shooting can be understood from this final formulation. By explicitly expressing continuity constraints (2.1) over certain points, multiple shooting allows the numerical algorithm to relax

those constraints. One direct implication is that at the beginning of the algorithm, those constraints can be fully relaxed. Hence the initial guess does not need to be continuous. Thus, multiple shooting can easily be initialized by other algorithms which give trajectories in the state space and do not take care of the full dynamic, like planning algorithms [Bouffard 2009]. This formulation has also been found useful to control unstable systems or to handle terminal constraints, because in both cases it can be difficult to find a valid initial guess in the control space. The problem generated using direct multiple shooting method can be condensed [Kirches 2012] (i.e. transformed from a larger and sparse problem to a smaller but dense problem)[1] and then be solved using any *NonLinear Programming* (NLP) solver.

2.2.4 Sequential Quadratic Programming (SQP)

In this section, we present the Sequential Quadratic Programming algorithm which is a NLP algorithm commonly used to solve this kind of problem. In this part, we consider the following standard-form NLP problem:

$$x^\star = \min_x f(x) \quad \text{subject to} \quad h(x) \geq 0, \quad r(x) = 0.$$

From this problem, we can introduce the *Lagrangian function*:

$$\mathcal{L}(x, \lambda, \mu) = f(x) - \lambda^T h(x) - \mu^T r(x)$$

where λ and μ are *Lagrangian multipliers*. The necessary conditions for $x^\star$ to be a local optimum of the NLP are that there exist multipliers $\lambda^\star$ and $\mu^\star$, such that

$$\begin{aligned} \nabla_x \mathcal{L}(x^\star, \lambda^\star, \mu^\star) &= 0, \\ r(x^\star) &= 0, \\ h(x^\star) \geq 0, \quad \mu^\star \geq 0, \quad h(x^\star)^T \mu^\star &= 0. \end{aligned}$$

At each step of the SQP algorithm, the objective function f is approximated by its local quadratic approximation and constraints by their local affine approximations

$$\begin{aligned} f(x) &\approx f(x_k) + \nabla_x f(x_k)^T \Delta x + \frac{1}{2} \Delta x^T H_k \Delta x \\ h(x) &\approx h(x_k) + \nabla_x h(x_k)^T \Delta x, \\ r(x) &\approx r(x_k) + \nabla_x r(x_k)^T \Delta x, \end{aligned}$$

where H_k is a Hessian approximation of $\mathcal{L}(x_k, \lambda_k, \mu_k)$, and $\nabla f(x_k)$, $\nabla h(x_k)$ and $\nabla r(x_k)$ are Jacobians. Thus, at each step of the SQP algorithm, we get the following

[1] Although it is not clear for us that condensation is an indisputable advantage – while sparse resolution seems an interesting and possibly more efficient alternative – we used it in this work to follow the main trend of the domain.

QP subproblem

$$\begin{aligned} \underset{\Delta x \in \{S,U\}}{\text{minimize}} \quad & \nabla_x f(x_k)^T \Delta x + \frac{1}{2} \Delta x^T H_k \Delta x, \\ \text{subject to} \quad & \\ & h(x_k) + \nabla_x h(x_k)^T \Delta x \geq 0, \\ & r(x_k) + \nabla_x r(x_k)^T \Delta x = 0. \end{aligned} \tag{2.2}$$

Which is solved using active-set methods. So, starting from an initial guess (x_0, λ_0, μ_0), SQP solver transforms the NLP into a QP subproblem and iterates:

$$\begin{aligned} x_{k+1} = & \quad x_k + \alpha \Delta x^{QP}, \\ \lambda_{k+1} = & \quad \lambda_k + \alpha \Delta \lambda^{QP}, \\ \mu_{k+1} = & \quad \mu_k + \alpha \Delta \mu^{QP}. \end{aligned}$$

where α can be determined using linesearch methods [Nocedal 2006]. Constraint linearization can result in unsolvable QP subproblems, so in these cases constraints have to be relaxed. One technique is to use ℓ_1 relaxation where the quadratic problem is transformed into

$$\begin{aligned} \underset{\Delta x}{\text{minimize}} \quad & \nabla f(x_k)^T \Delta x + \frac{1}{2} \Delta x^T H_k \Delta x + \nu u + \nu(v + w), \\ \text{subject to} \quad & \\ & h(x_k) + \nabla h(x_k)^T \Delta x \geq -u, \qquad (2.3) \\ & r(x_k) + \nabla r(x_k)^T \Delta x = v - w. \qquad (2.4) \end{aligned}$$

where u, v and w are slack variables and ν is a penalty parameter. The available SQP solvers can differ in the way the Hessian is approximated, the linesearch is done, the QP subproblems are solved or the constraints are relaxed. SQP has been shown to be a powerful tool and due to its superlinear convergence rate and its ability to deal with nonlinear constraints well.

2.2.5 Model Predictive Control (MPC)

MPC techniques use optimal control framework to solve online trajectory generation problems. At each activation, MPC solves an optimal control problem over a sliding time window. In robotics, MPC is usually activated several times per second so algorithms are the same but their applications differ from the offline trajectory generation. At each activation, a solution is found using only one iteration of the NLP solver. Therefore, the solution used will not be an optimal solution but, by using the previous solution to build the initial guess for the next step, MPC will at the same time improve the solution and adapt it to the new state of the system. The faster the algorithm is, the less the previous solution is out-dated so the more the algorithm will be able to improve it instead of just adapt it to the new situation. Thus, fast single shooting methods like DDP are often used

[Garimella 2015]. However, DDP is not directly able to handle constraints, so constraints like obstacle avoidance are often introduced in the problem using the cost function and therefore, we cannot be sure that the system will respect those constraints. In the algorithms used in this paper, the linearized constraints are checked at each step so in the case of convex obstacles, we are sure that the solutions given by MPC will always be collision-free (at least at each shooting node).

2.3 System Dynamics

In order to show the abilities of direct optimal control to easily adapt to various dynamic models, we have performed a benchmark with three different quadrotor-based robots whose models are given in sections 2.3.1, 2.3.2 and 2.3.3.

2.3.1 Quadrotor

The quadrotor is modeled as a rigid body of mass $m_q = 0.9[kg]$ evolving in the 3 dimensional space where effects linked to fluid dynamics are all neglected. Position and orientation of the quadrotor with respect to the inertial frame are respectively noted $\boldsymbol{x}_q \in \mathbb{R}^3$ and $\boldsymbol{\Theta}_q \in SO(3)$ (where $\boldsymbol{\Theta}_q$ can indistinctly be represented by Euler angles or quaternions) and its rotation speed in its local frame is represented by $\boldsymbol{\Omega}_q \in so(3) = \mathbb{R}^3$. Rotor dynamics are neglected so each propeller i produces a thrust $f_i = C_f V_i^2$ and a torque around its main axis $\tau_{zi} = (-1)^{i+1} C_m V_i^2$ where $V_i \in [V_{min}, V_{max}]$ is the motor velocity and C_f, C_m are respectively the lift and drag constants [Benziane 2015]. By neglecting the rotor dynamics, the control is directly the rotor accelerations. Thus, the state vector is $\boldsymbol{x} = [\boldsymbol{x}_q, \boldsymbol{\Theta}_q, \dot{\boldsymbol{x}}_q, \boldsymbol{\Omega}_q, V_1, ..., V_4]^T$ and the system control inputs are $u_i = \dot{V}_i \in [\dot{V}_{min}, \dot{V}_{max}]$, $i \in \{1, ..., 4\}$. The dynamic equations can be written as:

$$m_q \ddot{\boldsymbol{x}}_q = \begin{bmatrix} 0 \\ 0 \\ -m_q g \end{bmatrix} + R_{\boldsymbol{\Theta}_q} \begin{bmatrix} 0 \\ 0 \\ \sum_i C_f V_i^2 \end{bmatrix} \tag{2.5}$$

$$J_q \dot{\boldsymbol{\Omega}}_q + \boldsymbol{\Omega}_q \times (J_q \boldsymbol{\Omega}_q) = \begin{bmatrix} dC_f(V_1^2 - V_3^2) \\ dC_f(V_2^2 - V_4^2) \\ C_m(V_1^2 - V_2^2 + V_3^2 - V_4^2) \end{bmatrix} \tag{2.6}$$

where d is the distance between a rotor and the center of mass of the quadrotor and J_q is the quadrotor inertia matrix, $R_{\boldsymbol{\Theta}_q}$ is the rotation matrix between the world frame and the quadrotor frame, g is the gravitational acceleration and $\times$ represents a cross product.

2.3.2 Quadrotor with Pendulum

The second model corresponds to the same quadrotor, carrying a load attached by a rigid linkage.

2.3.2.1 With a fixed load

We assume that the quadrotor is attached to a load of mass m_p with a weightless rigid bar of length $L = 4[m]$. The rigid bar and the quadrotor are linked with a spherical joint (3 degrees of freedom). To calculate the quadrotor position and the pendulum orientation, the system is modeled as a weightless solid bar with point masses at each tip. Thus, all forces are applied on the axis of the bar and the inertia matrix J_p of the pendulum (bar + point masses) respects $J_{pxx} = J_{pyy}$. Moreover we assume that at the beginning of the trajectory the bar is not rotating around its main axis so Coriolis effects vanish and the dynamic equation can be simplified to:

$$J_p \dot{\boldsymbol{\Omega}}_p = \sum_i T_i \quad \text{and} \quad \omega_{pz} = 0$$

where $\boldsymbol{\Omega}_p$ is the rotation speed of the pendulum with respect to the inertial frame. Thus, the system state vector is $\boldsymbol{x} = [\boldsymbol{x}_q, \boldsymbol{\Theta}_q, \dot{\boldsymbol{x}}_g, \boldsymbol{\Omega}_q, \boldsymbol{\Theta}_p, \omega_{px}, \omega_{py}, V_1, ..., V_4]^T$ where $\boldsymbol{\Theta}_p$ is the orientation of the pendulum frame with respect to the inertial frame, ω_{px} and ω_{py} the rotation speeds of the pendulum in its local frame and $\dot{\boldsymbol{x}}_g$ the velocity of the center of mass of the whole system.

2.3.2.2 Grasping and releasing a load

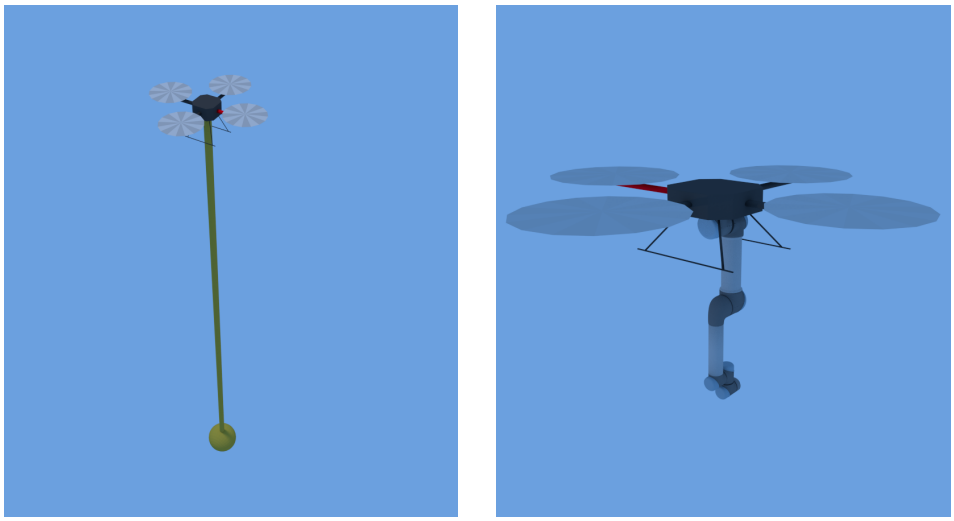
The mass at the end of the pendulum changes when grasping and releasing. We model the load transfer by a smooth variation of m_p, to keep the smoothness of the dynamic formulation (discontinuous transfer would have been possible too, but we believe that the smooth transfer more adequately models the linkage). We assume that the velocity of the end-effector with respect to the inertial frame is low at grasping and releasing time so the part of dynamic equation that corresponds to the variation of the mass can be neglected (this assumption is typically correct at the convergence of the direct optimal control solver).

$$\frac{d\boldsymbol{p}}{dt} = m_q \ddot{\boldsymbol{x}}_q + m_p \ddot{\boldsymbol{x}}_p + \underbrace{\dot{m}_p \dot{\boldsymbol{x}}_p}_{\approx 0}$$

$$\frac{d\boldsymbol{L}_G}{dt} = GQ \times m_q \ddot{\boldsymbol{x}}_q + GP \times m_p \ddot{\boldsymbol{x}}_p + \underbrace{GP \times \dot{m}_p \dot{\boldsymbol{x}}_p}_{\approx 0}$$

where $\boldsymbol{x}_p$ is the position of the load; Q, P and G are the center of mass of respectively the quadrotor, the load and the whole system; $\boldsymbol{p}$ is the linear momentum and $\boldsymbol{L}_G$ is the angular momentum at the center of mass of the whole system. However, when mass of the load changes, the center of mass moves along the pendulum so, at each step, dynamics need to be integrated according to the position/velocity of a new point:

$$||G_2 G_1|| = \frac{L m_q \dot{m}_p}{(m_q + m_p)^2 + \dot{m}_p (m_p + m_q)} dt$$



(a) system quadrotor and pendulum (b) system quadrotor and arm

Figure 2.1: Models of the systems

where G_1 is the position of the center of mass at time t and G_2 is its position at time $t+dt$. Thus, velocity of the center of mass is computed using:

$$\frac{d\dot{\boldsymbol{x}}_G}{dt} = \frac{G_2G_1}{dt} \times \boldsymbol{\Omega}_p + \sum Forces/(m_p + m_q)$$

2.3.3 Aerial Manipulator

In that section, we consider that a robotic arm is placed under a quadrotor. The model of the robot arm used for the tests is the *Universal Robot UR5*. For simplicity, the model of the arm is kept as it is and the model of the quadrotor is adapted to correspond to one that could lift this kind of load (see appendix A.1). Moreover, the quadrotor model is simplified by directly using motor thrusts as controls ($f_i \in [f_{min}, f_{max}]$, $\tau_{zi} = (-1)^{i+1} C_m f_i / C_f$). The *UR5* has 6 degrees of freedom, its state is the joints positions and velocities while the joint torques are chosen as controls. Therefore, the overall system composed of the quadrotor and its arm has state and control vectors of dimension respectively 24 and 10. While the dynamics of the two previous systems were written algebraically, for the aerial manipulator we rather used an algorithmic form, which is more condensed to write, more versatile and correspond better to a realistic use. The dynamic of the whole system is calculated using *Recursive Newton-Euler Algorithm* (RNEA) and *Composite Rigid Body Algorithm* (CRBA) [Featherstone 2007]. To speed up calculation, we used *Pinnocchio* [Mansard 2015], a computationally fast implementation of those algorithms. From those algorithms, we get:

$$M(q)\ddot{q} + b(\dot{q}, q) + g(q) = \tau$$

where $M(q)$ is the mass matrix (computed by CRBA), $b(\dot{q},q)$ and $g(q)$ are the generalized bias and gravitational forces (computed by RNEA) and $\tau = [f_{base}, \tau_{base}, \tau_{joints}]^T$ is the vector of generalized forces, where $f_{base} = [0,0,\sum_i f_i]^T$, τ_{base} are the torques induced by the propellers and τ_{joints} corresponds to the vector of torques acting on each joint of the arm. To add actuator inertia to this model, a term $M_{mot} = J_{mot} * K_{red}^2$ (where J_{mot} is the motor/reducer inertia and K_{red} is the reduction rate of the reducer) is added to the mass matrix:

$$M = M_{crba} + \begin{bmatrix} 0_6 & 0_6 \\ 0_6 & I_6 \end{bmatrix} M_{mot}$$

Then, the mass matrix M is inverted (using a sparse cheap algorithm [Featherstone 2007]) to get the direct dynamic needed by the optimal control. Finally, constraints on positions and controls are added to the optimization to respect joint limits and maximum torques of the *UR5*.

2.4 Implementation Details

2.4.1 Initial Guess

Contrary to single shooting methods where states are implicit variables so the initial guess needs to be specified in the control space, multiple shooting method is initialized with controls and states. This allows to use simple, noncontinuous but powerful initial guess like quasi-static or obstacle-free trajectories and that's why multiple shooting has been shown to be a very useful tool to connect the planning and control parts [Bouffard 2009]. In the experiments, the initial guesses used were mainly quasi-static trajectories between the start and the goal position:

$$\begin{aligned} \boldsymbol{x}_{qj} &= \boldsymbol{x}_{start} + \frac{j}{N}(\boldsymbol{x}_{goal} - \boldsymbol{x}_{start}), && j = 0,...,N, \\ V_i &= \sqrt{mg/4C_f}, && i = 1,...,4. \end{aligned}$$

And all other variables are set to zero.

2.4.2 Obstacle Avoidance

Contrary to Dynamic Programming where the obstacle avoidance problem is often treated outside then re-injected in the optimal control problem in the form of waypoints, direct methods deal easily with constraints on the state space, so the obstacle avoidance problem can easily be inserted in the optimal control problem by adding inequalities on the position of the system. To get concave and differentiable constraints, obstacles are embedded inside ellipsoids. This model allows not only to keep trajectories outside of obstacles even when constraints are linearized but also to guide trajectories around obstacles. So in the optimal control problem, the

constraint for one obstacle is modeled as:

$$(\boldsymbol{x}_q - \boldsymbol{x}_e)^T A(\boldsymbol{x}_q - \boldsymbol{x}_e) - 1 \ \geq \ 0$$

where $\boldsymbol{x}_q$ is the quadrotor position, $\boldsymbol{x}_e$ is the position of the ellipsoid center and $A \in \mathbb{R}^{3\times3}$ is a positive definite matrix. For tasks involving obstacles avoidance, a quasi-static trajectory going through obstacles could be a bad initial guess because trajectory continuity and obstacle avoidance constraints can be incompatible when linearized. In that case, constraints are relaxed until the obstacle is thin enough to be between two nodes where constraints are checked. From this trajectory, at each step the algorithm will decrease the relaxation term u in (2.3) i.e the obstacle constraint will grow back. When the constraints have finally converged back, the result is usually a trajectory where the system goes fast enough to get through the obstacle when constraints are not checked. Thus, when obstacles are present, the initial guesses used are obstacle-free trajectories or at least trajectories where the obstacle constraints do not need to be relaxed.

2.4.3 Rotations

To describe free flying objects, quaternions seem to be a better convention for rotations since there are no singularities. The quadrotor model was tested with quaternions, with explicit or implicit (i.e. by continuity) unit constraints. Tests have shown that in both cases, convergence was much slower than with Euler's angles. This is most probably because the unit constraint of quaternions is a nonlinear constraint that the SQP solver cannot approximate well. Even if this problem can be invisible in some applications, the trajectories generated here are highly dynamic with a large range of orientations and large rotational velocities, so poor approximations result in much slower convergences. Therefore, in all the tests presented in the next sections, rotations are represented using Euler's angles. For the most dynamic trajectories where the system could hit a singularity, the axes are selected such that the orientation does not get close to a singularity.

2.4.4 Experimental Setup

The multiple shooting algorithm has been implemented by Moritz Diehl et al. in an open source optimal control software, the *ACADO toolkit* [Houska 2011]. *ACADO* solves multiple shooting problems thanks to a *Sequential Quadratic Programming* (SQP) algorithm, together with state-of-the-art techniques to condense, relax, integrate and differentiate the problem. This tool has already been shown to be useful to generate complex quadcopter trajectories as throwing and catching an inverted pendulum [Brescianini 2013]. Trajectories presented in the following parts have been generated using this toolkit. Problems were solved on a IntelÂő Xeon(R) CPU E3-1240 v3 @ 3.40GHz with Runge-Kutta 45 integrator and Broyden-Fletcher-Goldfarb-Shanno (BFGS) Hessian approximation. Moreover, controls are discretized as a piecewise constant function, constant between each shooting node.

2.5 Results

In this section, we report various examples of the capabilities of the optimal control problem solver to discover and control complex trajectories with the three models presented above. The general idea is to give a complete list of the capabilities of the approach in terms of range of exploration, speed of computation and robustness to perturbation. For all the reported examples, no external planning method was used to discover the trajectory (although the aggressive flip trajectory (see section 2.5.2.1) was not found from scratch by the solver due to its symmetry). MPC is demonstrated in the cases where we were able to set it up. In general, the computation time is sufficient to enable MPC however, in some cases, we were not able to obtain a reasonable tradeoff between computation approximation and control performance (in particular for the quadrotor with pendulum). The complete implementation of MPC in all the demonstrated examples along with its application on real quadrotors is left for future works. Tests presented in this section can be seen in a video available online (see `https://www.youtube.com/watch?v=hcFK32C-bxs`).

2.5.1 Non-Optimal Trajectories

In the case where the task is simple enough, we can encode it using only constraints. Starting with an initial guess which does not respect the system dynamic but respects the initial and final constraints (quasi-static trajectory), the algorithm is able to find a valid trajectory after few iterations (convergence criteria : *KKT conditions* under 10^{-12}). Tab. 2.1 shows the computation times for trajectories of 20 shooting nodes over 8 seconds.

2.5.2 High-dynamic maneuvers

2.5.2.1 Time optimal trajectories with the quadrotor alone

Direct methods allow to add parameters to the optimization. In this case, resolution of the optimization will be over the controls and parameters state space. These parameters can be used to modify dynamics, cost or constraints. So by adding the final time T as a parameter to optimize and setting it as the cost, the algorithm will try to find a time optimal trajectory respecting the initial and final constraints. Even if the final time T changed over the SQP iterations, the number of nodes of the multiple shooting remains constant. Therefore, variation of the final time is taken into account when dynamic is computed i.e. by the integrator. Fig. 2.2, 2.3 shows the result obtained for position reaching tasks (6 meters lateral displacement with static terminal constraints) in minimum time with the quadrotor system alone. As we can see, commands calculated by the optimal control are *bang-null-bang* as we can expect for a time optimal control. However, the more the algorithm converges to the optimal solution, the more it hits constraints so the longer the QP resolutions are and the smaller the steps are. Therefore, complete convergence is obtained after

System	Task	Time $[s]$ (SQP iterations)
Quadrotor alone	Horizontal displacement of $10[m]$	0.35 (4)
	Horizontal displacement of $30[m]$	0.46 (5)
	Vertical displacement of $10[m]$ + obstacles avoidance	0.51 (6)
Quadrotor + pendulum	Pendulum stabilization after a perturbation ($\omega_{px\,initial} = 2[rad/s]$)	0.78 (4)
	Horizontal displacement of $10[m]$ with inverted pendulum	0.76 (4)
	From pendulum $\psi_{p\,initial} = 0$ to inverted pendulum $\psi_{p\,final} = \pi$	1.99 (7)
Quadrotor + arm	Reaching a position with the end-effector (distance $5[m]$)	6.74 (6)

Table 2.1: Computation times for non-optimal trajectories.

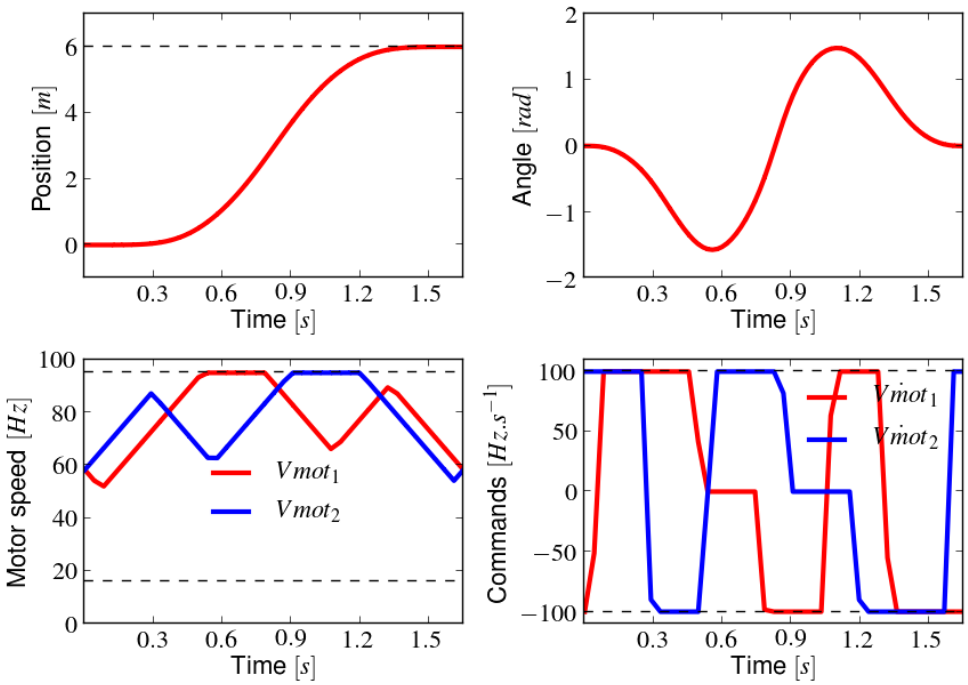


Figure 2.2: Time optimal trajectory for the quadrotor.

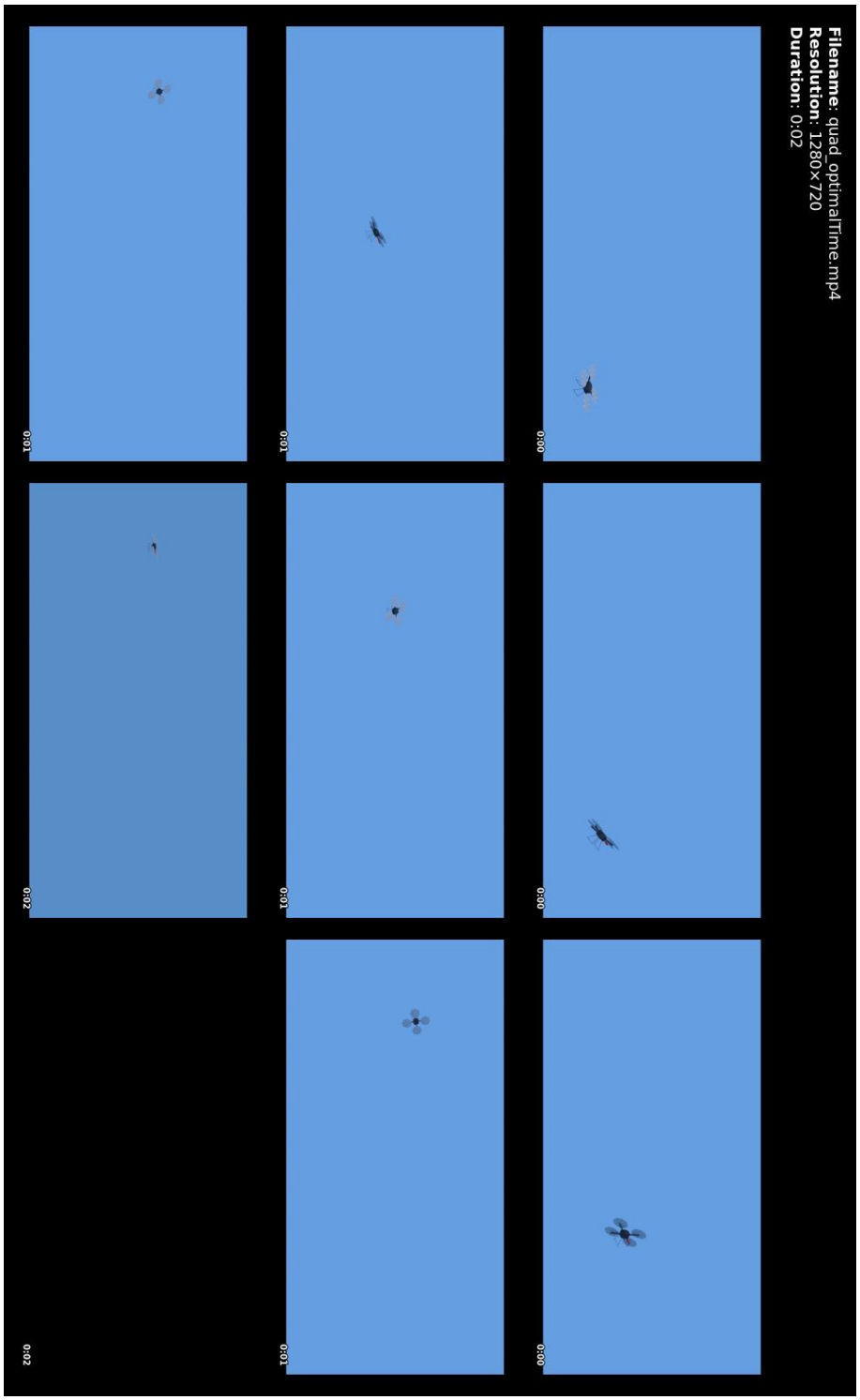


Figure 2.3: Video screen shots: the time optimal trajectory for the quadrotor.

57 seconds (373 SQP iterations) although a solution as $T = 1.05T_{optimal}$ is obtained after only 12.7 seconds (105 SQP iterations).

When the quadrotor is performing a vertical displacement going up, from a certain distance, the optimal time trajectory is a trajectory where the quadrotor is flipping over at the end of the trajectory and use its propellers to slow down (Fig. 2.4). In [Ritz 2011], authors showed that an indirect optimal control was able to find those trajectories. Here, the algorithm used is a local algorithm, so it is not able to jump between valid areas of a non-continuous constraint and is only able to find a local optimum. In this case, the end constraint that the system must be static, is a non-continuous constraint which constrained the final roll and pitch angles to $[\phi_f, \theta_f] = [2k\pi, 2l\pi], \quad [k,l] \in \mathbb{N}^2$ where $[k,l]$ is actually determined by the initial guess. So, with initial guess used (quasi-static with roll $\phi = 0$ and pitch $\theta = 0$), the algorithm is only able to find trajectories where the quadrotor does half a turn in one way then another in the other way instead of a whole turn. Moreover, because of the system symmetry, when the trajectory is strictly vertical, the local optimum is always the trajectory without flip. To find a trajectory with flip, the initial guess needs to be perturbed enough to exit the basin of attraction of the solution without flip. The size of the perturbation depends on the length of the trajectory: for a 40 meters high trajectory, setting a lateral displacement of 1 meter in the initial guess is enough to find the flipping trajectory; for fewer than 20 meters high, a simplified flip needs to be encoded in the initial guess (for instance, for a 20 shooting nodes trajectory, the roll angle is set as $\phi_i = \pi$ for $i = \{16, 17, 18\}$ and $\phi_i = 0$ otherwise).

2.5.2.2 Time optimal trajectories with systems quadrotor with pendulum or arm

Optimal time trajectories for the tasks presented in Tab. 2.1 are visible on Fig. 2.5, 2.6, 2.7, 2.8.

2.5.2.3 Model Predictive Control (MPC)

To speed up calculation when using the algorithm as MPC, the cost function is simply set as:

$$\int_0^T (\boldsymbol{x}_q - \boldsymbol{x}_{goal})^T C_1 (\boldsymbol{x}_q - \boldsymbol{x}_{goal}) + \boldsymbol{\Omega}_q^T C_2 \boldsymbol{\Omega}_q$$

where the term $\boldsymbol{\Omega}_q^T C_2 \boldsymbol{\Omega}_q$ is used for stabilization of the trajectory and C_1, C_2 are weighting matrices. For this test, each 0.2 second a new control is found using the first iteration of a SQP, solving a multiple shooting problem of 20 nodes over a 8 seconds sliding time window. The time used to perform one SQP iteration is variable according to the number of QP iterations, but here each SQP iteration took about 0.1 second. This delay is not taken into account in the simulation but on a real application, MPC can use techniques like *delay compensation* [Diehl 2009] to compensate delays between measures of state and computation of controls. On Fig.

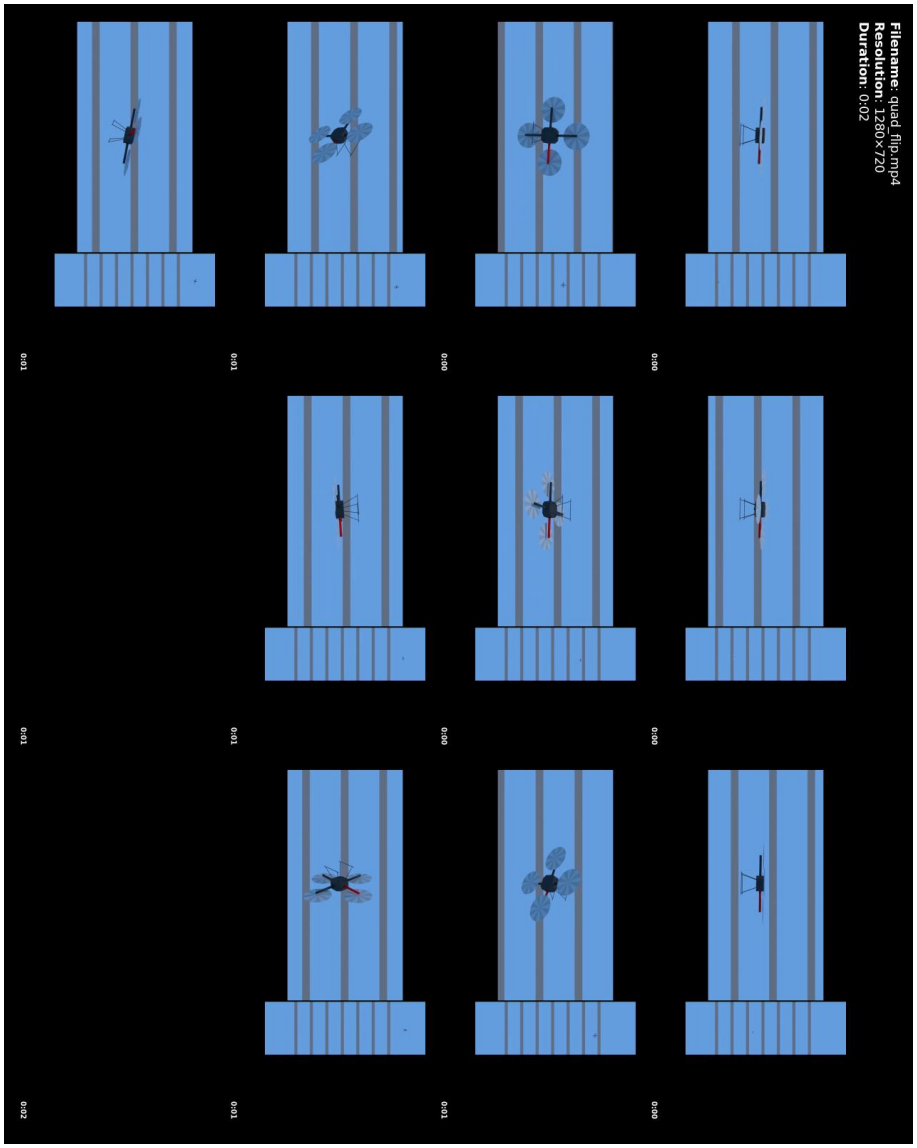


Figure 2.4: Video screen shots: time optimal vertical trajectory with the quadrotor.

Figure 2.5: Video screen shots: time optimal trajectory for the quadrotor with pendulum – From pendulum to inverted pendulum.

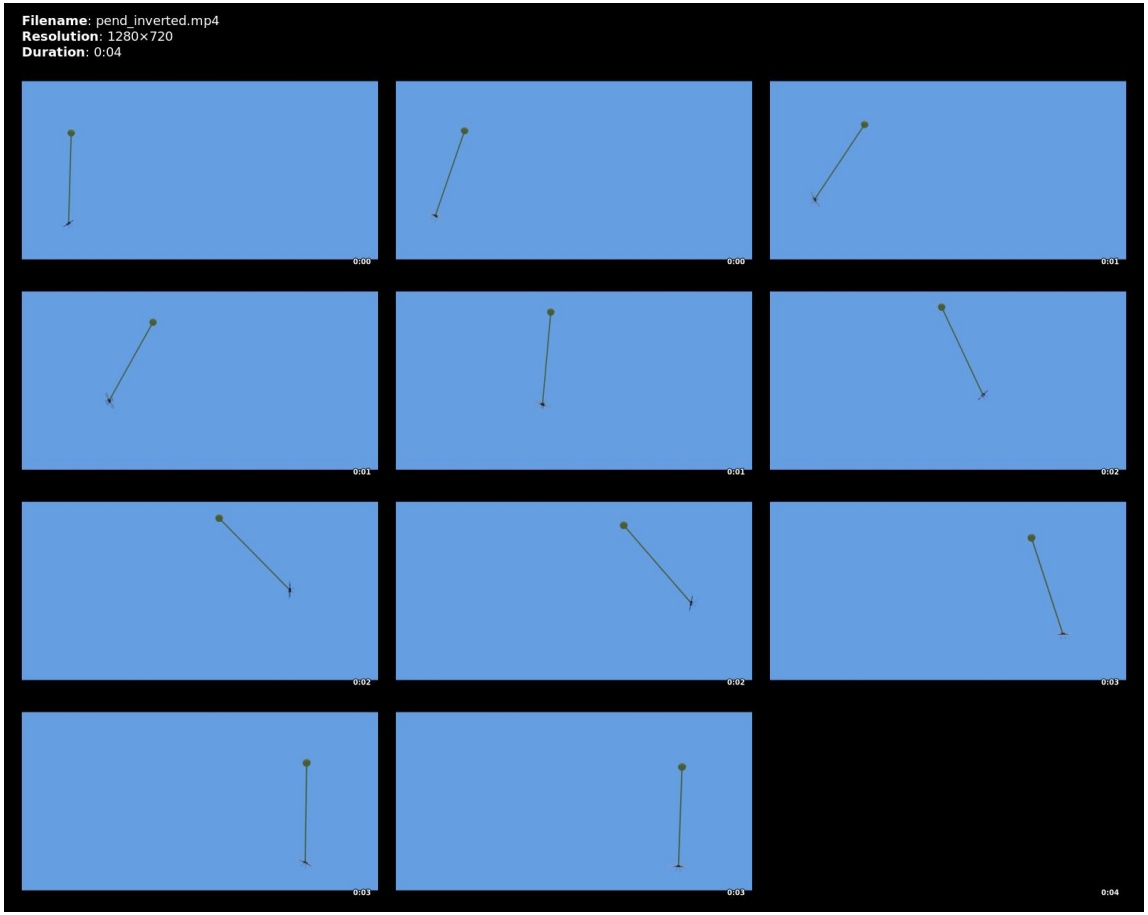


Figure 2.6: Video screen shots: time optimal trajectory for the quadrotor with pendulum – inverted pendulum.

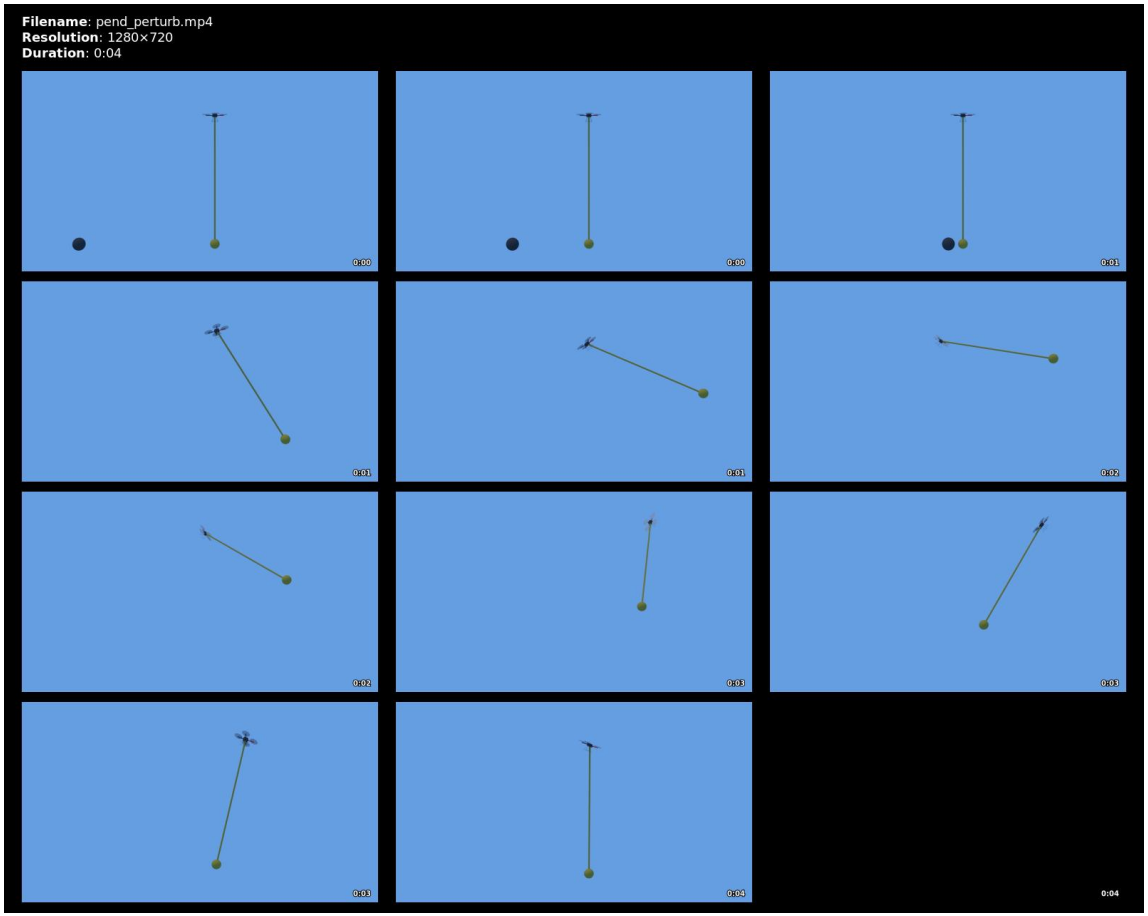


Figure 2.7: Video screen shots: time optimal trajectory for the quadrotor with pendulum – pendulum stabilization after a perturbation.

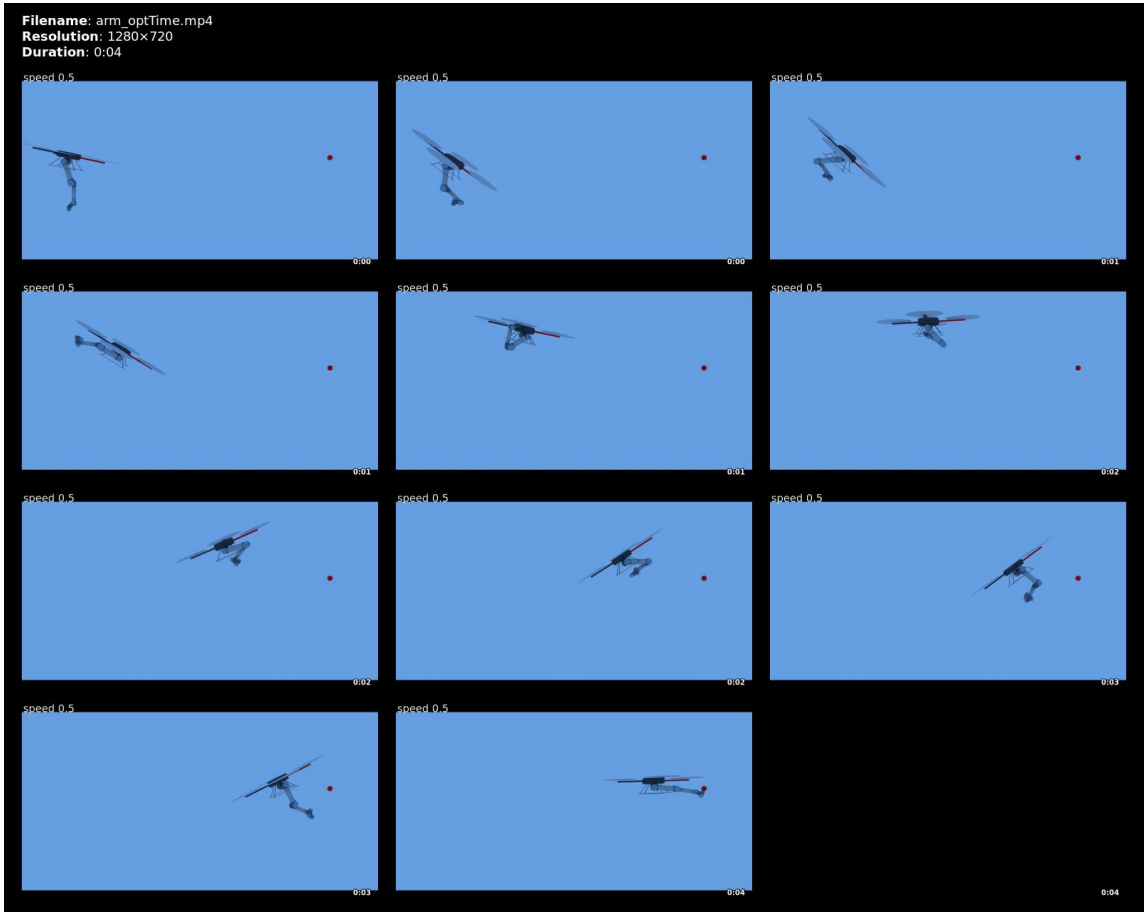


Figure 2.8: Video screen shots: time optimal trajectory for the aerial manipulator – Reaching a position with the end-effector.

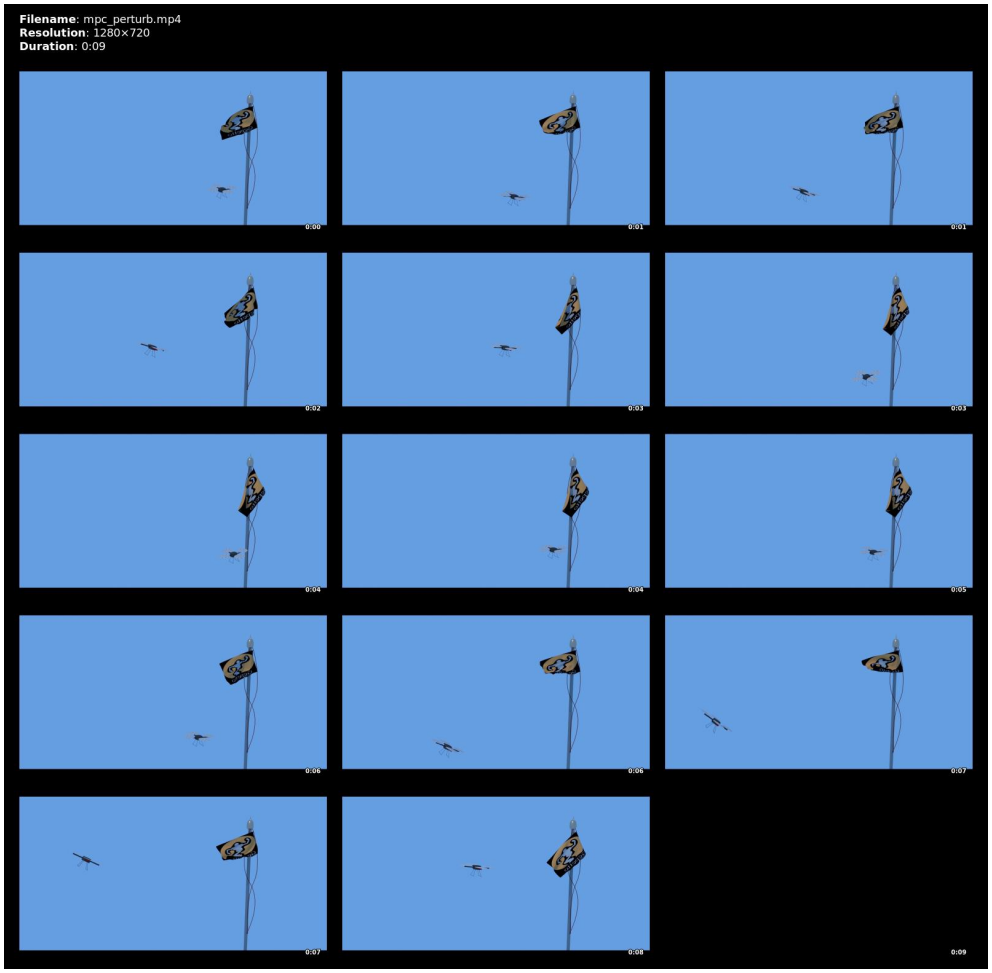


Figure 2.9: Video screen shots: MPC with the quadrotor – wind perturbation.

2.9, MPC stability is tested with wind gusts (wind gusts are modeled as a constant piecewise force acting on the quadrotor and no estimation of the perturbation is inserted into the model).

2.5.3 Point-to-point Trajectories Through Obstacles

2.5.3.1 Time optimal trajectories through obstacles

Fig. 2.10, 2.11 and 2.12 show optimal time trajectories with obstacle avoidance for the three systems. To simplify the obstacles constraints aerial manipulator, we suppose that the distance between the obstacle and its constraints is always greater than the length of the arm. Therefore the obstacle constraints are only applied on the center of the robot.

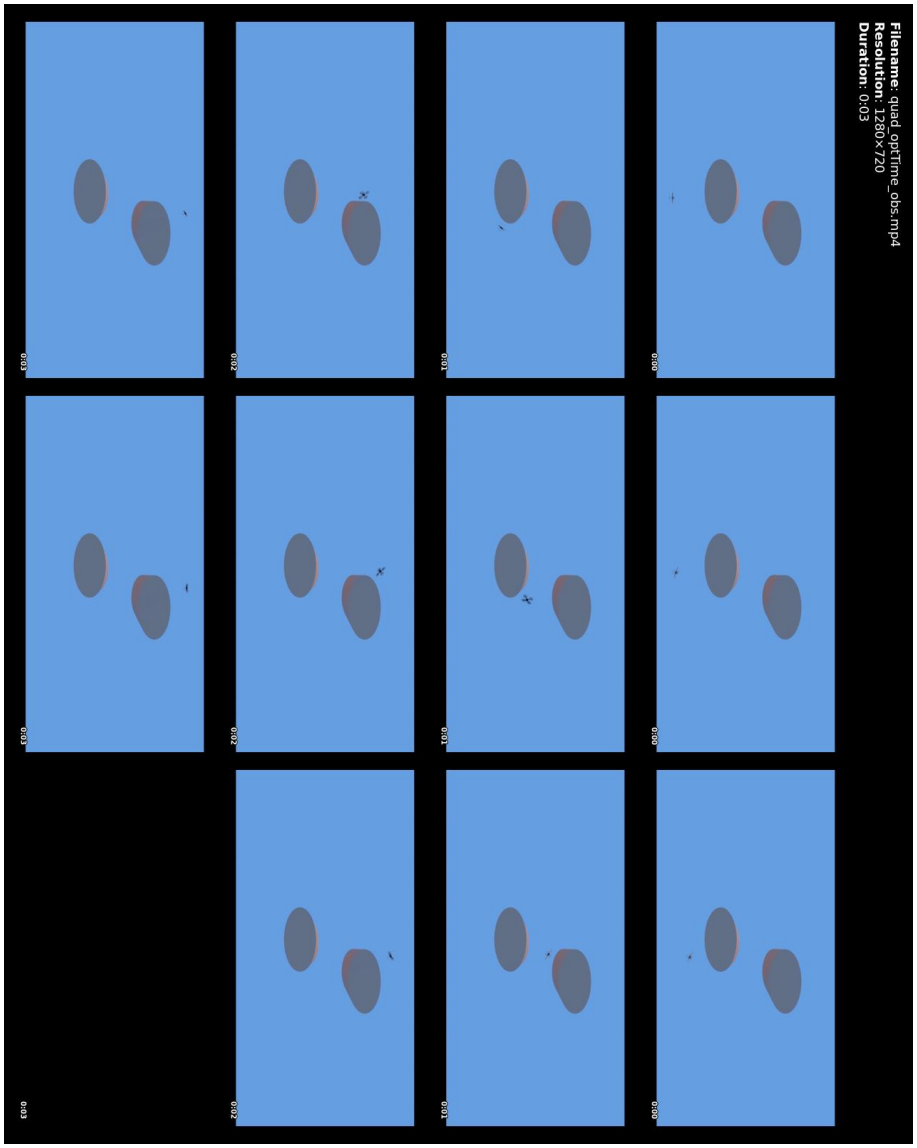


Figure 2.10: Video screen shots: time optimal trajectory with obstacles for the quadrotor.

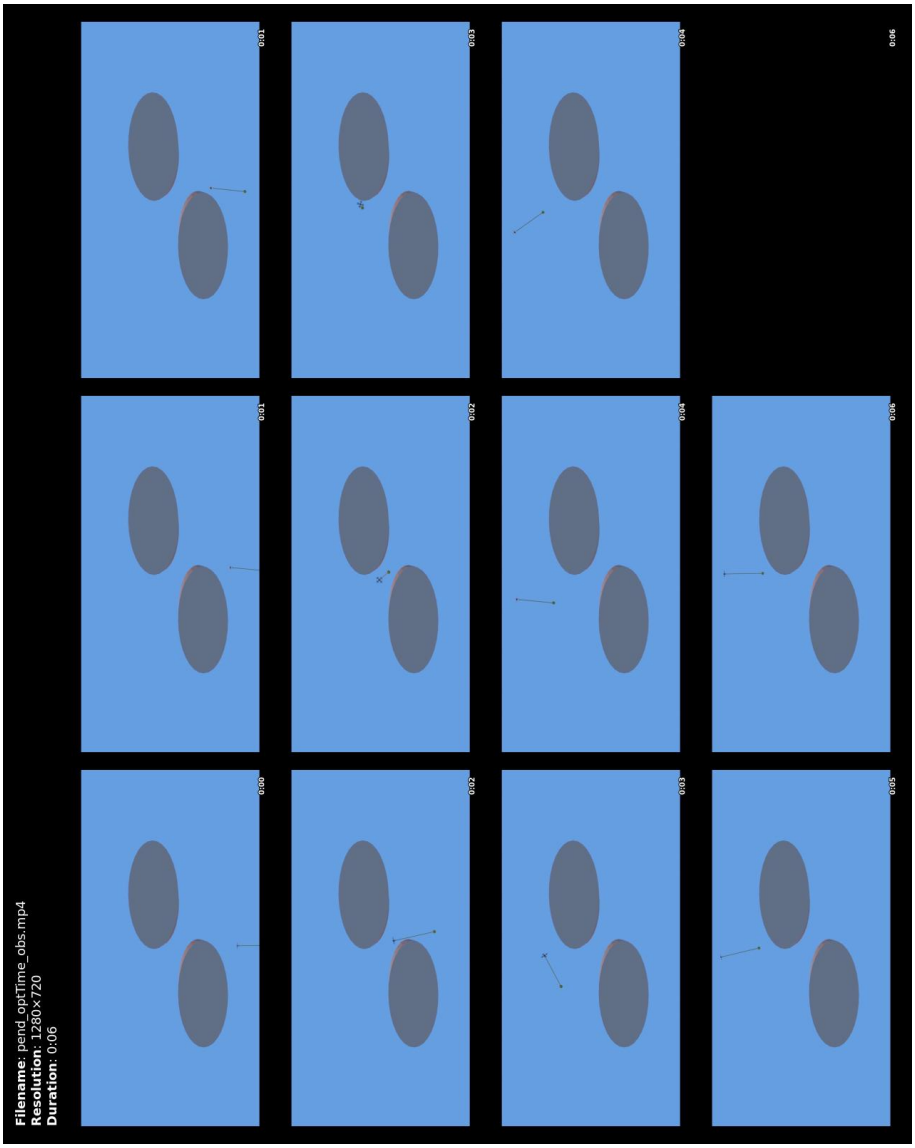


Figure 2.11: Video screen shots: time optimal trajectory with obstacles for the quadrotor with pendulum.

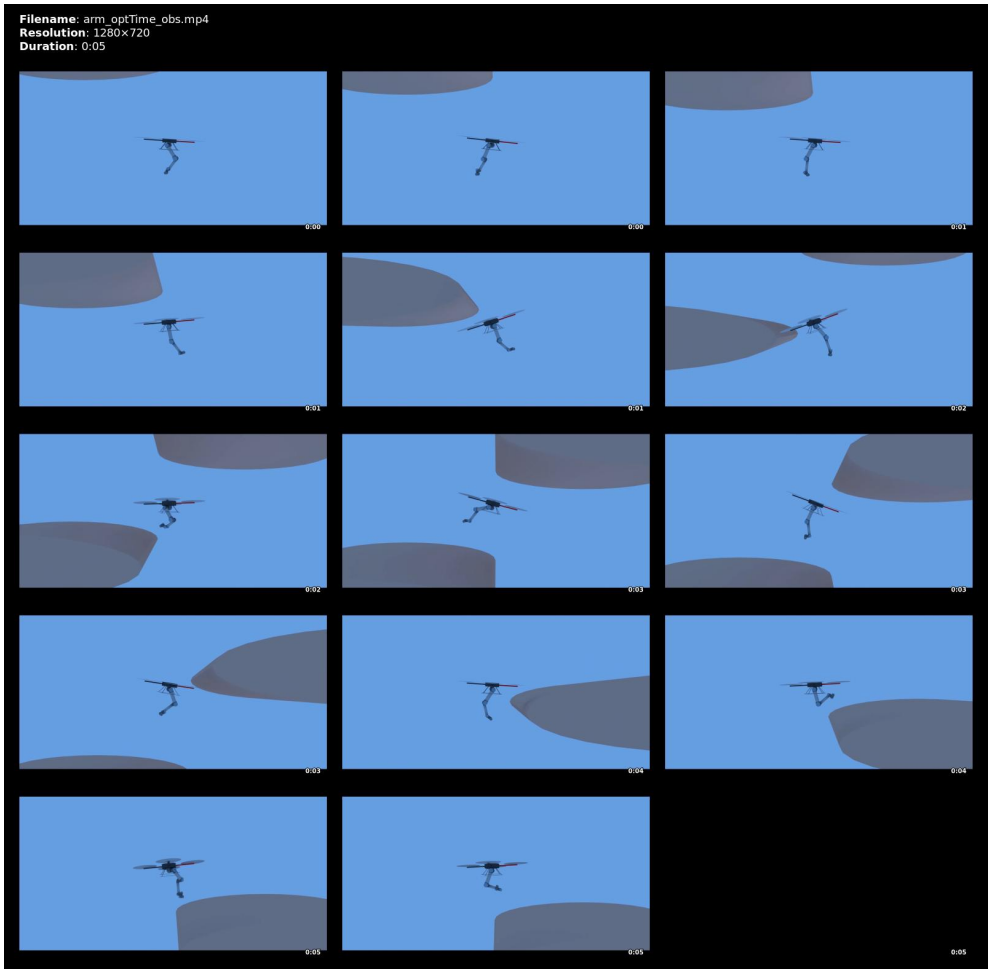


Figure 2.12: Video screen shots: time optimal trajectory with obstacles for the aerial manipulator.

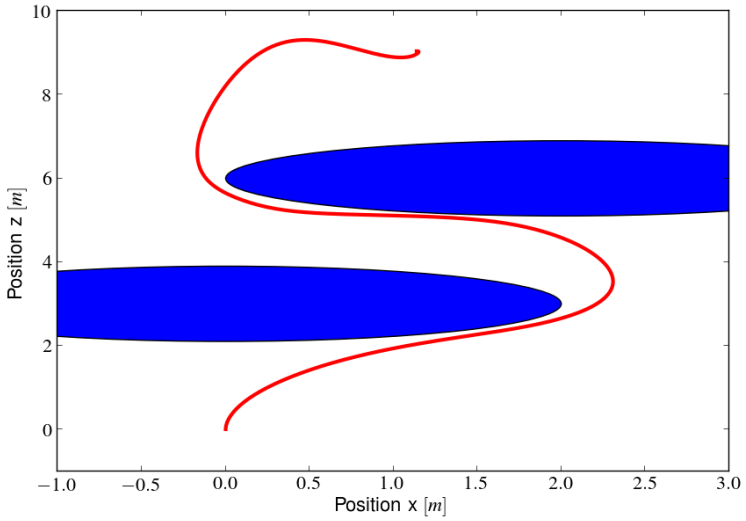


Figure 2.13: Trajectory of the quadrotor using MPC formulation. The blue ellipses represent the obstacle avoidance constraints.

2.5.3.2 MPC

By using simple ellipsoidal constraints for the obstacles, checking those constraints is fast enough to be done online. Fig. 2.13 and 2.14 show an example of MPC with obstacle avoidance for the quadrotor.

2.5.3.3 Going through a window

In this experiment, we ask the quadrotor with pendulum system to go through a window while carrying a load ($m_p = 0.45[kg]$). Since the window height ($2[m]$) is smaller than the pendulum length ($4[m]$), the quadrotor has to find a solution where it swings up the pendulum to get through. The window is supposed infinitely large so only the top and bottom part are modeled as obstacles i.e using ellipsoid constraints. No initial guesses are used (the initial guesses are static trajectories at the initial position) so the algorithm is free to find the best solution according to the cost function. The trajectory needs to be able to spread through the window, so no terminal constraints are set on the pendulum state and the terminal constraint that the system should be static is only applied on the quadrotor.

In this test, the cost function corresponds to the squared distance between the goal position (which is at the other side of the window) and the quadrotor final position $E(x) = C_3||x_q(T) - x_{goal}||^2$. In this case, the quadrotor starts to move back and forth until the pendulum swings high enough to get through the window (see Fig. 2.15, 2.16, computation time : 10.3 seconds).

$$\int_0^T (\boldsymbol{x}_q - \boldsymbol{x}_{goal})^T C_1 (\boldsymbol{x}_q - \boldsymbol{x}_{goal}) + \boldsymbol{\Omega}_q^T C_2 \boldsymbol{\Omega}_q$$

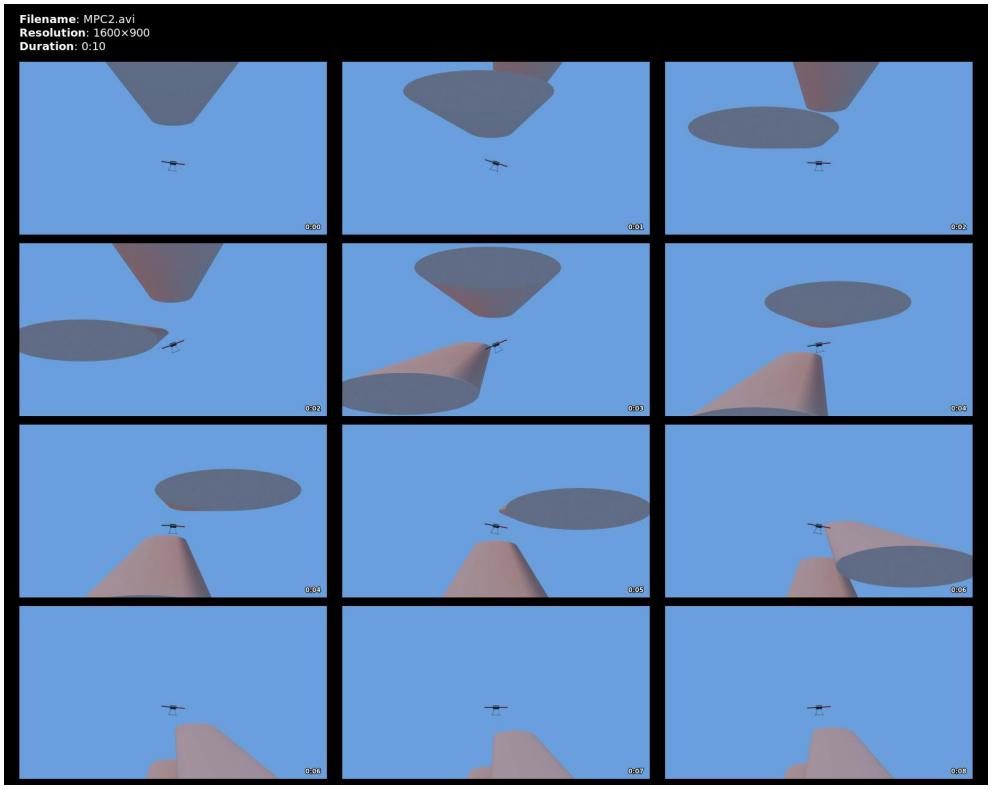


Figure 2.14: Video screen shots of MPC with obstacles contraints for the quadrotor.

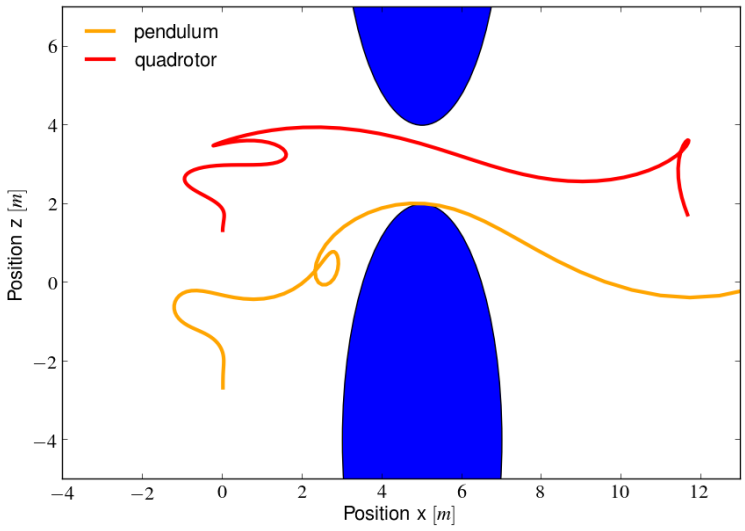


Figure 2.15: Trajectories for the quadrotor with pendulum ($m_q = 0.45[kg]$) while going through a window with terminal cost.

In the second test, a running cost is added $L(x,u) = \int_0^T C_4 ||\boldsymbol{x}_q(t) - \boldsymbol{x}_{goal}||^2 dt$. Here, moving backward at the beginning of the trajectory is costly so instead of moving back and forth, the algorithm is able to find a solution where the quadcopter is moving from left to right to swing the pendulum up (see Fig. 2.17). This kind of behaviour would not be possible if we were using waypoints as in [de Crousaz 2015] because it would restrict the solution to the one encoded by the waypoints.

2.5.4 Pick and Place

In this part, system starts and ends at chosen positions and, on its way, has to pick an object at a certain position then place it to another one. No obstacles are added to the problem so the object is considered as flying. The task is encoded as follows: the initial and final states are set using initial and final constraints and picking/placing tasks are encoded using the cost function:

$$L(x,u) = \frac{\rho}{\sqrt{2\pi}} e^{-\frac{\rho^2 (t-T_{pick})^2}{2}} (\Delta \boldsymbol{x}_{pick}^T C_5 \Delta \boldsymbol{x}_{pick} + \dot{\boldsymbol{x}}_{ee}^T C_6 \dot{\boldsymbol{x}}_{ee}) \tag{2.7}$$

$$+ \frac{\rho}{\sqrt{2\pi}} e^{-\frac{\rho^2 (t-T_{place})^2}{2}} (\Delta \boldsymbol{x}_{place}^T C_7 \Delta \boldsymbol{x}_{place} + \dot{\boldsymbol{x}}_{ee}^T C_8 \dot{\boldsymbol{x}}_{ee}) \tag{2.8}$$

$$\Delta \boldsymbol{x}_{pick} = (\boldsymbol{x}_{ee} - \boldsymbol{x}_{pick})$$

$$\Delta \boldsymbol{x}_{place} = (\boldsymbol{x}_{ee} - \boldsymbol{x}_{place})$$

where $\boldsymbol{x}_{ee}$ is the position of the end-effector, $\boldsymbol{x}_{pick}$ and $\boldsymbol{x}_{place}$ are respectively the positions where the system needs to take and leave the object and C_i are weighing matrices. ρ allows to manage the duration during which the end-effector needs to

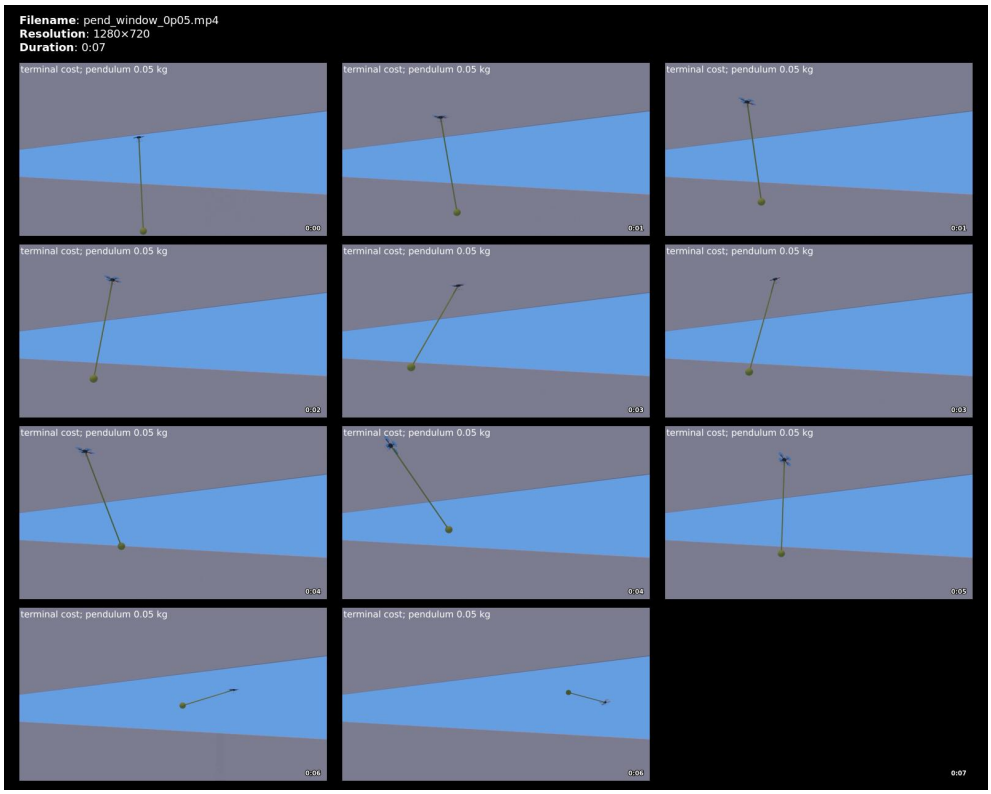


Figure 2.16: Video screen shots: going through a window for the quadrotor with pendulum ($m_q = 0.05[kg]$).

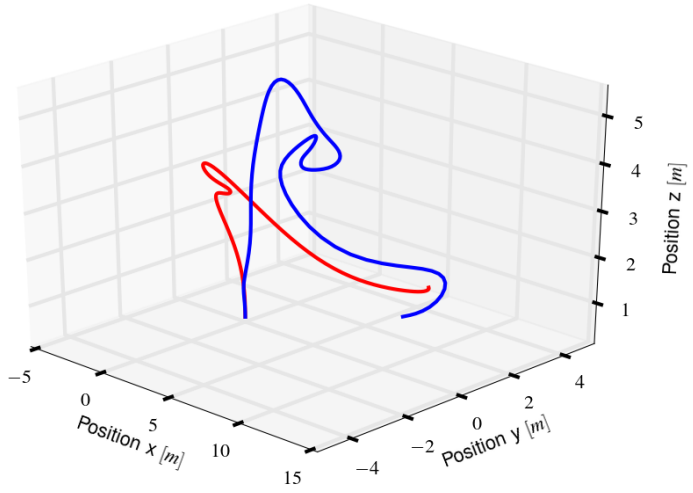


Figure 2.17: Influence of the cost function on going through a window trajectory for the quadrotor with pendulum. In red, the trajectory using terminal cost function, in blue the one with integral cost function.

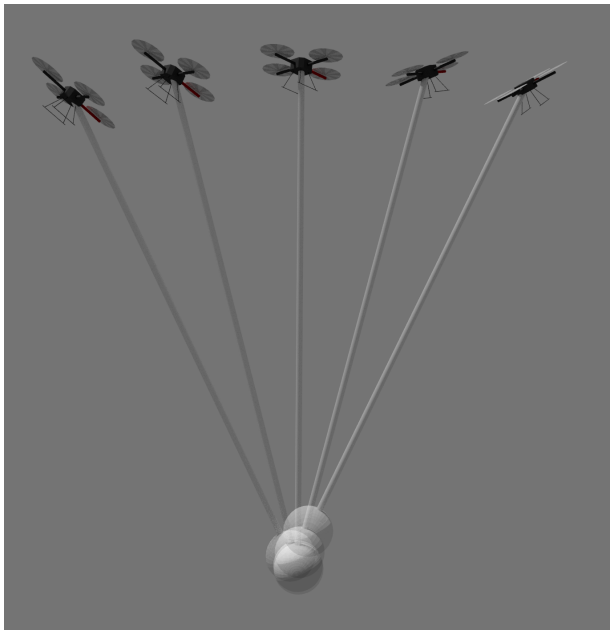


Figure 2.18: Trajectory of the quadrotor and the pendulum for picking task.

be at the picking/placing positions and T_{pick}, T_{place} are parameters that correspond to the time when object is picked/placed. Terms $\dot{\boldsymbol{x}}_{ee} C_i \dot{\boldsymbol{x}}_{ee}$ are used to stabilize the end-effector on the desired position instead of moving around.

For the quadrotor with pendulum, we suppose that the object which needs to be moved has a mass $m_o = 0.55[kg]$. Therefore, the mass at the bottom of the pendulum will increase from $m_q = 0.05[kg]$ to $m_q = 0.6[kg]$ when picking then back to $m_q = 0.05[kg]$ after releasing the object. So mass is set as a trapezoidal function where the slop lasts one interval of the time grid. Fig. 2.19 shows the trajectory of the system when picking the object : by swinging the pendulum then drawing an arc of circle around the object position, the quadrotor is able to keep the bottom of the pendulum around the same place even if its mass changes. To get a nice behavior like this with a under actuated system, the optimization needs to have enough degrees of freedom so the time grid is set as 60 nodes over a 8 seconds trajectory.
For the quadrotor with arm, we consider that the object has a weight which is light enough to be neglected. Therefore, the system keeps the same dynamic during the whole trajectory and T_{pick} and T_{place} can easily be included in the optimization, so they are set as free parameters with constraints that $0 < T_{pick} < T_{place} < T$ and will be optimized by the algorithm. This system has much more actuated degrees of freedom than the last one, thus the algorithm is able to find a end-effector position with a much smaller error (Fig. 2.21) even with a rougher time grid i.e 20 nodes over a 8 seconds trajectory. Fig. 2.22 and 2.23 show that the algorithm is able to exploit the full dynamic of the system: when performing the picking/placing, the arm is able to compensate for the motion of the quadrotor so it does not need to

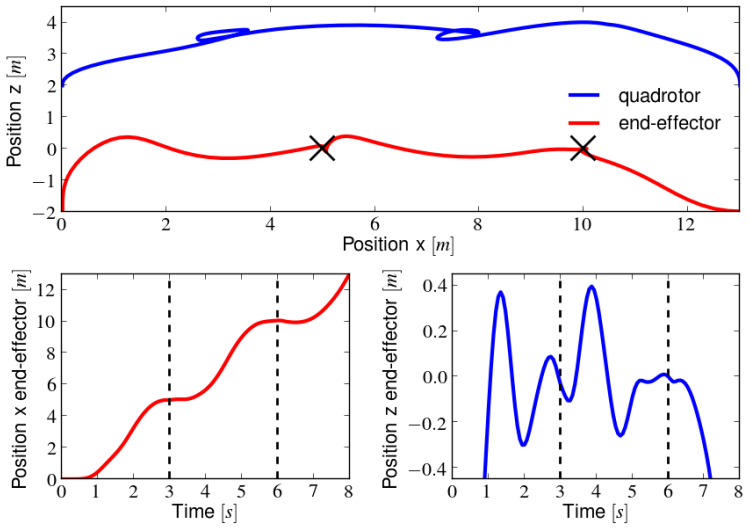


Figure 2.19: Trajectories of the pendulum for picking and placing task.

be in hovering state. For those tests, the algorithm does not only solve a task of placing the end-effector at certain positions, at certain times but also tries to keep it as close as possible, as long as possible. Therefore, the shape of the cost function is much more complex and the algorithm needs a lot of iterations to converge a suitable solution (250 SQP iterations so about 7 minutes for both systems).

2.5.5 Manipulation Tasks

In the *Pick and Place* experiment, time varying functions (2.7) and (2.8) are used to specify waypoints for the end-effector. For more complex manipulation tasks, the cost function can be used to set a complete reference trajectory for the end-effector. As in *Pick and Place*, the optimization needs several minutes to converge to an acceptable solution but allows to exploit the full system dynamic. Fig. 2.24 and 2.25 show trajectories where the aerial manipulator is used to drag an object or to turn a crank handle.

2.6 Application: Smart Teleoperation

Flying quadrotor through a remote control is not a easy task and demand training for the pilot. Moreover, this task can be particularly difficult when evolving in a cluttered environment like a forest (Fig. 2.26), a plant or close to obstacles (e.g for taking pictures of a bridge or an airplane wing for careful inspections). This difficulty yet prevents the implementation of useful applications that might otherwise be performed using a drone.

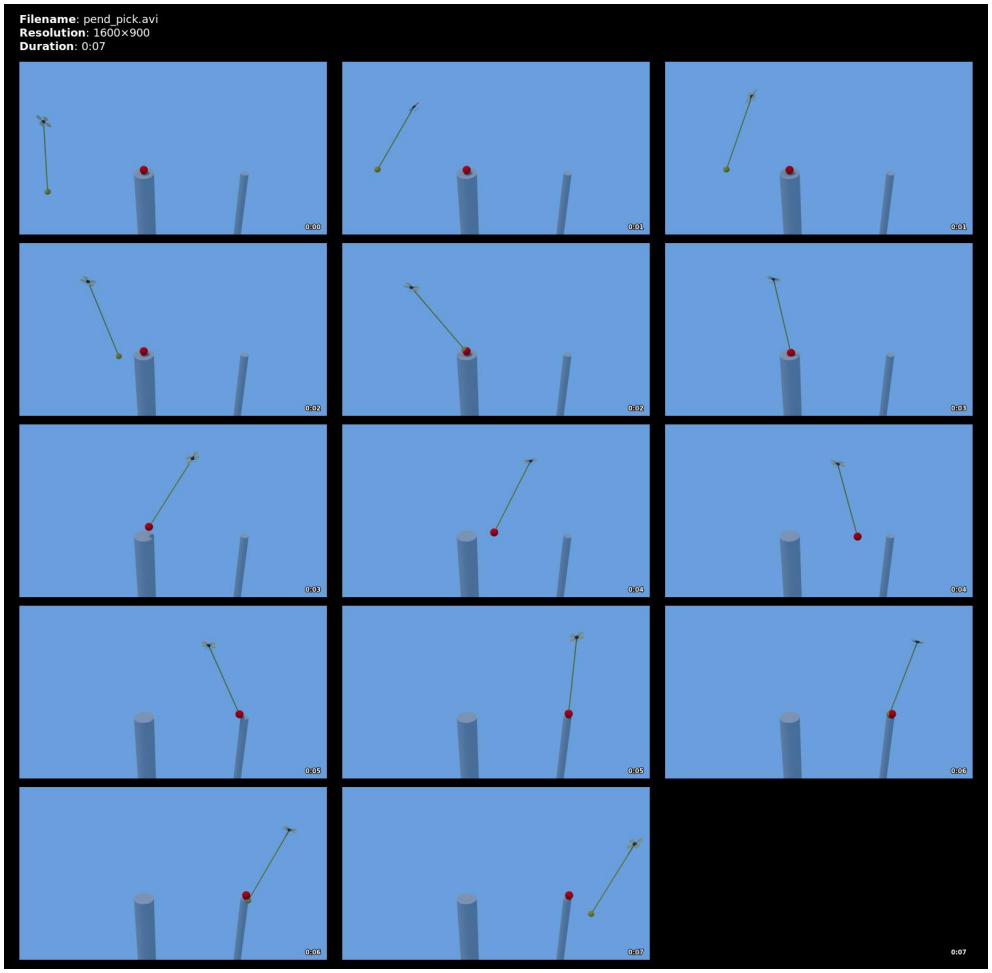


Figure 2.20: Video screen shots: pick and place task for the quadrotor with pendulum.

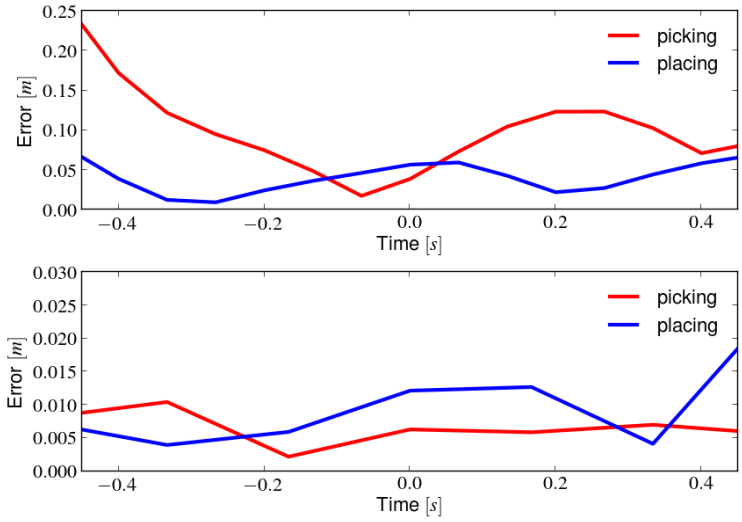


Figure 2.21: Error between picking/placing position and the end-effector position. At the top, error for the quadrotor with pendulum, at the bottom, error for the aerial manipulator. Time scale is centered on the picking/placing time.

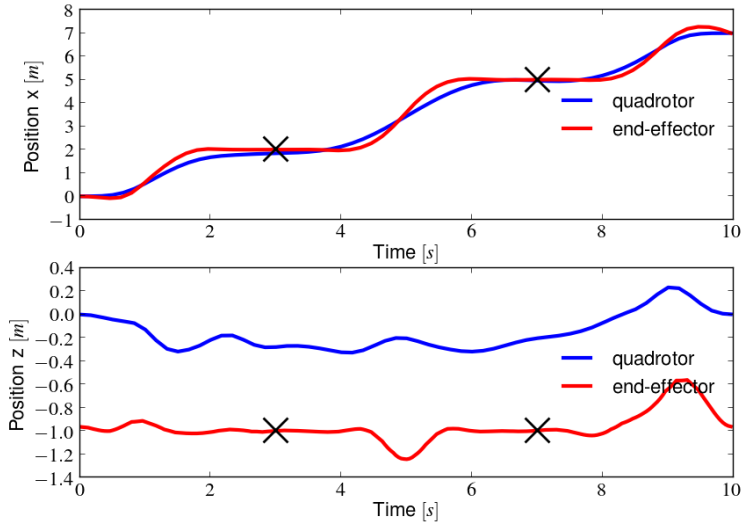


Figure 2.22: End-effector and quadrotor trajectories for the aerial manipulator solving the "pick and place" task.

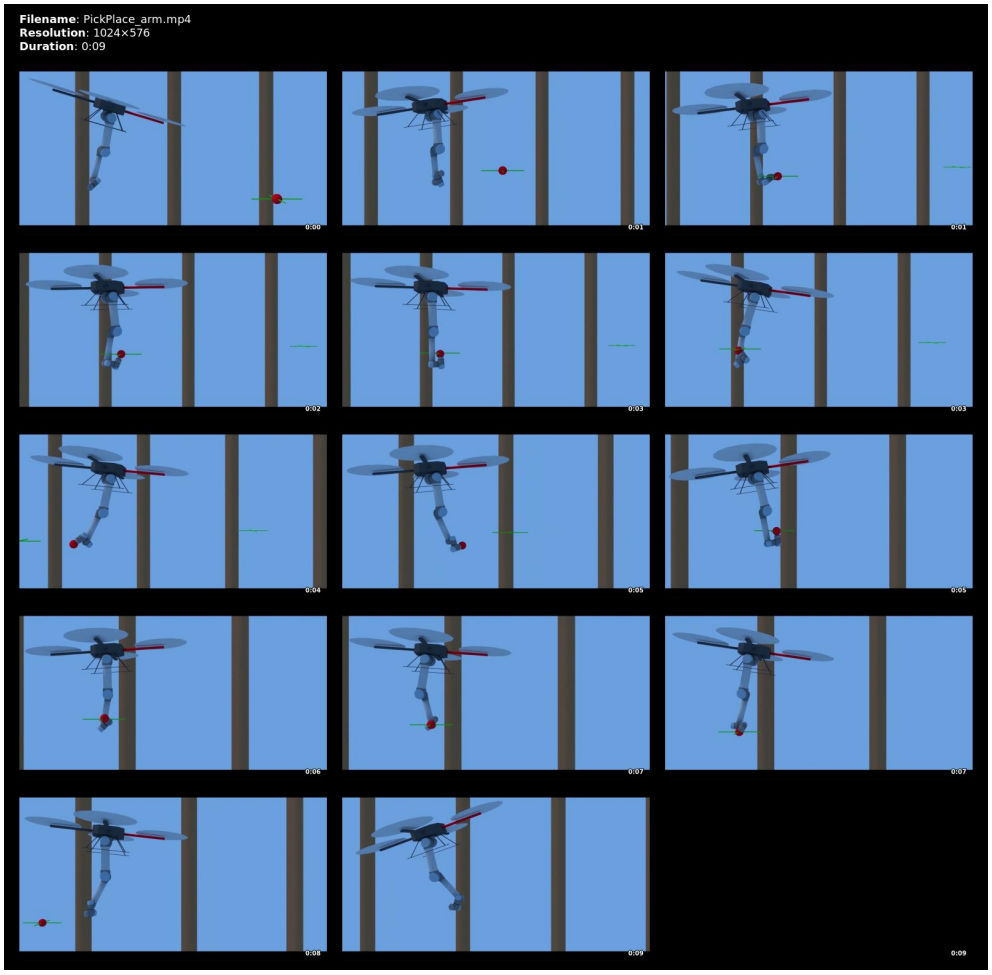


Figure 2.23: Video screen shots: pick and place task with the aerial manipulator.

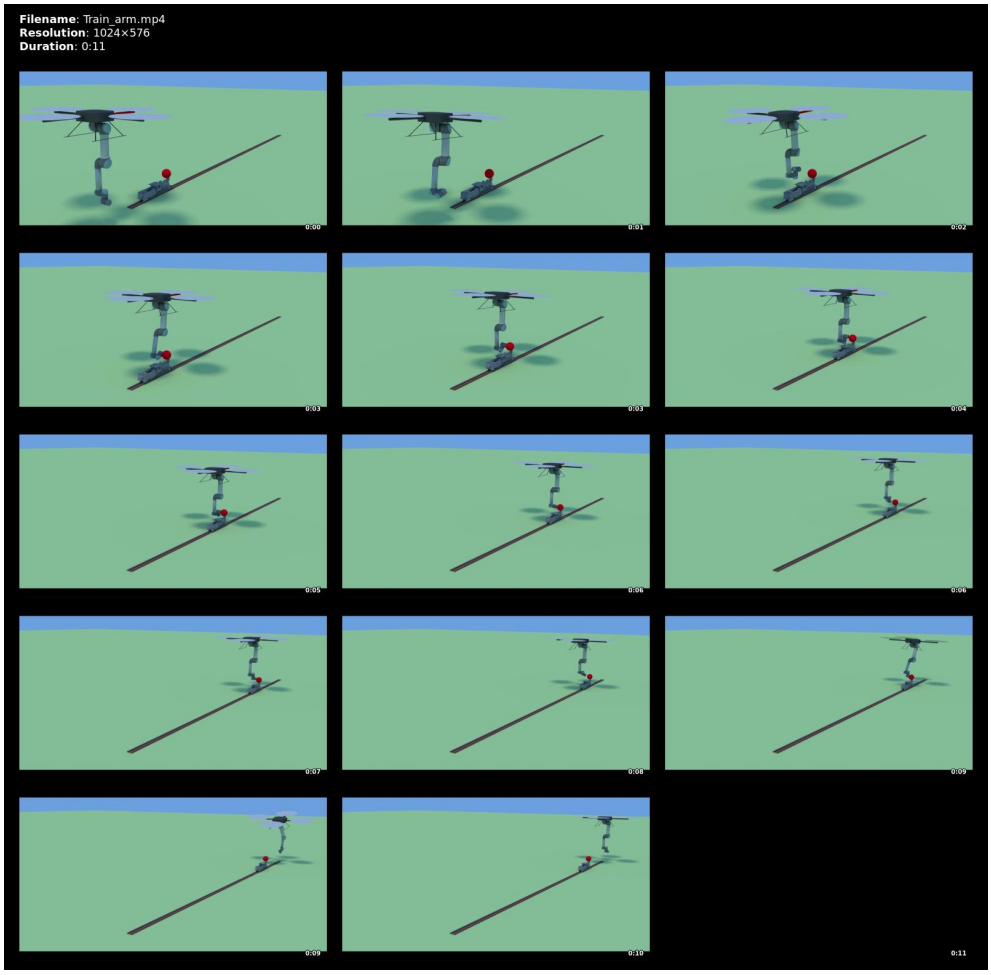


Figure 2.24: Video screen shots: dragging with the aerial manipulator.

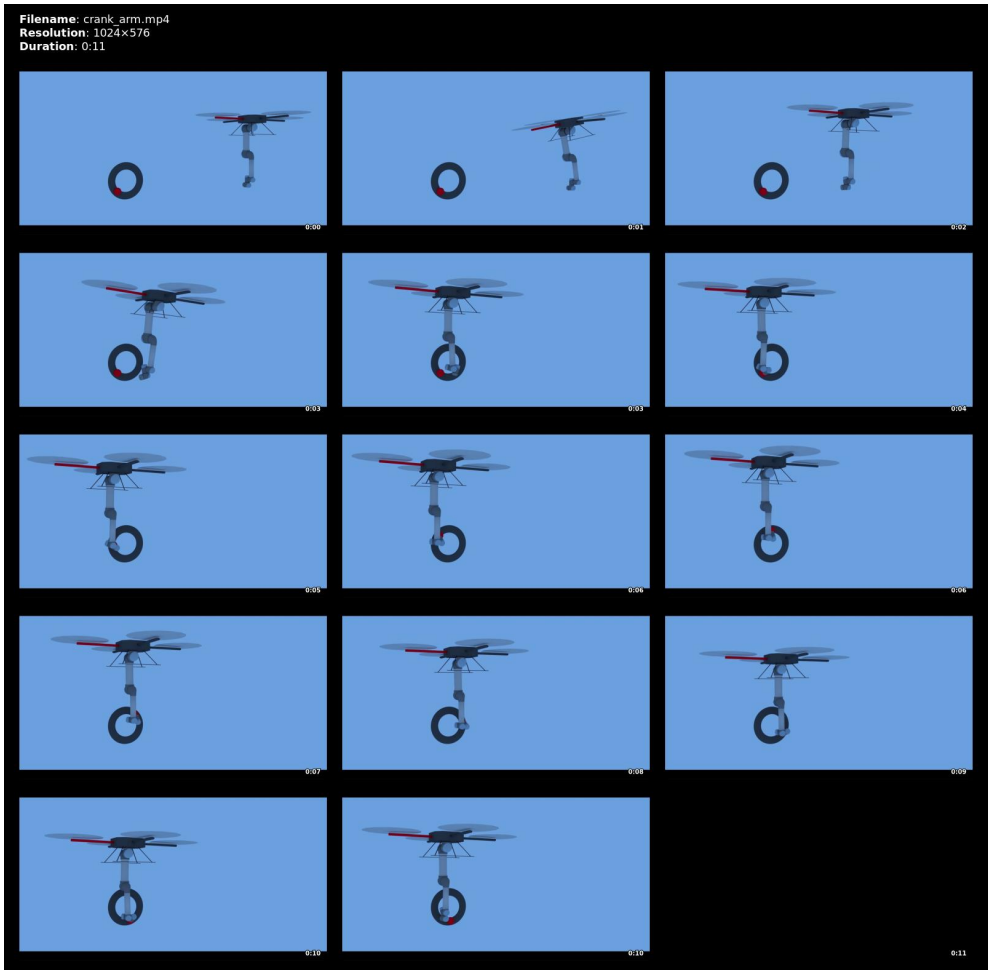


Figure 2.25: Video screen shots: turn a crank handle with the aerial manipulator.

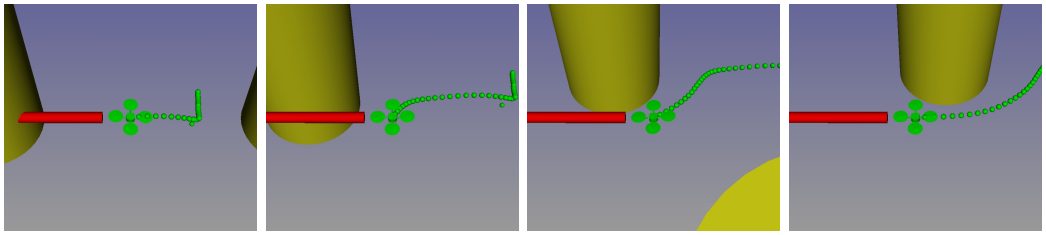


Figure 2.27: Smart teleoperation: the algorithm is able to modify the trajectory to avoid obstacles. The yellow cylinders are obstacles; the red cylinder represents the input velocity commanded by the user; positions of the quadrotor along time are displayed in green.

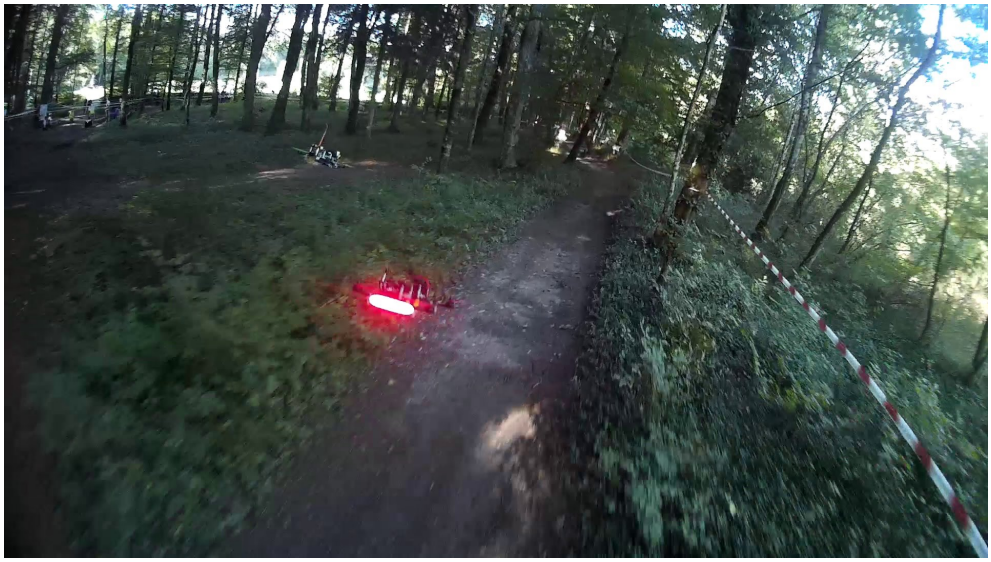


Figure 2.26: NCS Challenge 2014

We propose to build a "smart"teleoperating system to make this task easier and insure safe movements. This system takes the inputs given by the operator through a remote control as a reference velocity and computes the controls of each propeller using MPC formulation. For this application the cost function is set as:

$$\int_0^T (\dot{\boldsymbol{x}}_q - \dot{\boldsymbol{x}}_{ref})^T C_9 (\dot{\boldsymbol{x}}_q - \dot{\boldsymbol{x}}_{ref}) + \boldsymbol{\Omega}_q^T C_{10} \boldsymbol{\Omega}_q$$

where $\dot{\boldsymbol{x}}_{ref}$ is the reference velocity given with the operator. As in the previous sections, obstacle avoidance is done by adding ellipsoidal constraints in the optimization problem. Therefore, if the teleoperator drives the quadrotor toward an obstacle, the teleoperating system will either stop the quadrotor or drive it around to avoid the obstacle (see Fig. 2.27, video `https://www.youtube.com/watch?v=q4qaJmK3RzY`). Tests on simulation show that the computation time is acceptable

(around 35 ms for a preview window of 15 nodes over 1.5 s). The controller is able to correctly control the system even without delay compensation and only starts to show unstable behaviors when the delay becomes greater than twice the computation time. With the current implementation, the algorithm is computationally too expensive to be implemented on a small quadrotor. However, it can easily be integrated in a ground station. A dedicated implementation of the solver, engineered to fit the capabilities of the on-board computer, should also (in our opinion) make it possible without a ground station. In the first part this chapter, we used simple

heuristics (static or quasi-static) or the previous solution (in the case of MPC) to initialize our nonlinear solver. However, the results and the convergence speeds obtained can be improved by using a more efficient initial guess. In the next part, we show that we can also use an automatic method to generate an initial guess already close to the optimal solution.

Warm-starting the Nonlinear Predictive Controller

2.7 Warm start in MPC

As explained in the first part of this chapter, nonlinear MPC boils down to the iterative solution of a nonlinear, often nonconvex, optimization problem. To achieve quick convergence to a good local optimum, we must rely on a good initial candidate for the solver search. This is called a warm start (by opposition to a cold start, where a trivial initial guess – e.g. constant zero control – is provided). When controlling the robot through MPC, the optimization problems only marginally varies when the horizon recedes. Between two control cycles of the robot, the two main variations of the optimization problems are: i) the need to adapt the solution to the new initial state (typically estimated from the sensors) and ii) the new piece of horizon added at the end of the trajectory. In general, there is not enough time to achieve several optimization steps between two control cycles. As soon as the solver get a feasible solution, it is sent to the motor and the horizon recedes. At each new control cycle, the optimization process must also reduce the cost that was only partially optimized in the previous control cycles. If the system is subject to perturbations, the solution found at the previous control cycle needs to be adapted to the new situation. Obviously in that case, the trajectory found in the previous control cycle cannot meet the new optimality conditions. This approach is working well on a system whose dynamics does not vary too much, and where the receding horizon does not imply a big change in the optimum (either because the model is excellent, or because the control frequency – hence the resolution speed of the MPC solver – is quick compared to the dynamic perturbation). If large perturbations occur, this strategy becomes less efficient. In particular, the active set of the underlying QP subproblem (2.2.4) will be badly initialized and will need several (possibly many) iterations to converge. The QP may also become unsolvable, which implies that a relaxation phase is then needed. In both cases, computation time will increase. Thus the control frequency will be reduced and the effects of the perturbations will increase. This could lead to unstable behaviors that also worsen the situation and can typically make the control diverge. Therefore, when the system is subject to disturbances or to changes in the environment, we could need a better initial guess than the previous solution. In this section, we propose to use offline computations to build an approximation of optimal trajectories. This approximation is then used online to initialize our nonlinear solver. The approximation of optimal trajectories is built according to the *Iterative Roadmap Extension and Policy Approximation* algorithm [Mansard 2018].

2.8 Iterative Roadmap Extension and Policy Approximation (IREPA)

The IREPA algorithm aims at building an approximation of the optimal state and control trajectories that can be provided to the MPC solver as warm start. It proceeds by systematically solve many variations of the MPC problem offline in simulation for several initial and final conditions. This dataset of example trajectories is approximated by a neural network. The IREPA algorithm efficiently combines three different approaches to construct a model of the optimal solutions: optimal control, sampling-based planning and machine learning. The main idea of the algorithm is to build concurrently a kinodynamic roadmap and an approximation of the edges of the roadmap, by developing the roadmap using a better approximation and by improving the approximation using the extended roadmap. The roadmap is composed of sampled robot states (i.e. position and velocity, typically randomly sampled) and trajectories to connect some pairs of states. The roadmap edges are computed using the MPC solver, which acts as a steering method [Boeuf 2015]. In the first iteration of the IREPA, the MPC is cold started (a trivial initial guess is provided), which results in many connection failures, and also in some suboptimal edges (either the MPC solver does not converge, and when it converges, it may be to a poor local optimum). Empirically, we observe that only the pairs of states with a small distance (in the state space) can be properly connected in the first iteration. Then three neural networks are optimized from the roadmap edges. The two first networks are built to predict the state and control trajectories connecting two states. The third network is built to predict the value function, i.e. the optimal cost to go from one state to the other. The training is done by achieving a classical regression, while the example dataset is composed of the edges and subtrajectories of the edges. As explained above, the dataset contains some optimal trajectories, but also wrong examples with trajectories that are far to be optimal. The last step of an IREPA iteration is to remove from the dataset these outliers, i.e. edges of the roadmap that are far from the network predictions. In the later IREPA iterations, the MPC solver is warm-started using the network prediction, which are approximations of the optimal trajectory. Although, the roadmap is preferentially extended following the value-function approximation as a metric of the state space. At each new iteration of the IREPA, the roadmap is extended using the learned approximation, and the approximation is refined using the extension. The algorithm is summarized in Alg. 1.

We empirically observed that IREPA is a very efficient solution to drive the MPC to the optimal trajectory. This good behavior is reasonable, considering the complementarity of the pieces that build the algorithm. As we have seen, optimal control can quickly find optimal solutions but it needs to be correctly initialized (otherwise the solver could diverge) and can get trapped into local minima. On the other hand, sampling-based planning generate a high number of trajectories and can explore a large space to connect initial and goal states with (global) optimal

Algorithm 1 Iterative Roadmap Expansion and Policy Approximation (IREPA)

```
Initialize PRM with a given number of state samples
repeat
    Expand PRM
    stop ← True if PRM is fully connected else False
    Optimize approximator RMS
    for every edge E_PRM in the graph do
        (x_0,x_1) ← getStartAndEndNodes(E_PRM)
        E_new ← approximator(x_0, x_1)
        if E_new ≤ E_PRM then
            Replace E_PRM by E_new
            stop ← False
until stop
```

trajectories. However, the algorithms need a *steering method* to compute trajectories connecting pairs of nodes and an optimal solution can only be found with an infinite number of samples. Machine learning is able to automatically construct a model that interpolate data but needs a high number of examples spread over the whole space and they need to correspond to the right set of data (i.e. optimal trajectories here). Thus, drawbacks of each approach can be compensated by the others:

- Sampling-base planning can use optimal control as steering method to connect nodes that are close with locally optimal trajectories.
- Machine learning can use the roadmap generated by the sampling-based planning to generate an approximator of (global) optimal trajectories.
- Optimal control can use the approximator generated with machine learning to be correctly initialized in the basin of attraction of the global optimum.

The IREPA results of a collaborative work, where our main contribution is experimental. We therefore refer the interested reader to the original paper [Mansard 2018], where more details are available.

2.9 Results

In this section, we mainly report the results obtained with a planar UAV with two propellers. Trajectories obtained with other dynamic systems are shown in a video (see `https://www.youtube.com/watch?v=CbyCa7aeC7k`). The system has 3 degrees of freedom and 2 control inputs. The small dimension allows us to more easily plot the behavior of the algorithms and the resulting trajectories of the system. We start by reporting the technical details of our tests. Then we present the main results divided in offline phase (IREPA algorithm) first, and online phase (warm start) last.

2.9.1 Setup

2.9.2 System dynamics and cost

The UAV is modeled by a state of dimension 6: two translations x, z, one rotation θ, and the corresponding velocities $\dot{x}, \dot{z}, \dot{\theta}$. The control inputs are the forces f_1, f_2 applied by the propellers, which must be positive and bounded. The dynamics is:

$$\ddot{x} = -\frac{1}{m}(f_1 + f_2)sin\theta \tag{2.9}$$

$$\ddot{y} = \frac{1}{m}(f_1 + f_2)cos\theta - g \tag{2.10}$$

$$\ddot{\theta} = \frac{l}{I}(f_1 - f_2) \tag{2.11}$$

where m and I are mass and inertia of the UAV, g is gravity and l is the lever arm of the propeller. We set $m = 2.5\text{kg}$, $I = 1.2\text{kg}\,\text{m}^2$, $l = 0.5\text{m}$, following the dimension of the AscTec Falcon. The propellers are limited to $25N$, without continuity constraints. We consider minimum-time trajectory, i.e. the cost function is the trajectory duration. Similar results are obtained with minimum-norm running costs.

2.9.3 Approximators

We used a neural network to implement the approximator function (Value, policy, trajectories). All networks are implemented with 2 hidden layers with ELU activation function. The output layer of the Value approximator is also activated by ELU, while policy and trajectories output layers are hyperbolic tangent (scaled within range limits).

2.9.4 Computational setup

The optimal control solver and neural-network stochastic gradient descent [Kingma 2014] are implemented in C++. The rest of the application (planner, IREPA algorithm, etc) is implemented in Python. The tests were run on a single process on an Intel Xeon E5-2620 (2.1GHz). The algorithm would be easy to parallelize: most of the CPU time is consumed by the steering method, which can be run independently on several cores.

2.9.5 Offline phase

We first empirically validate the convergence of IREPA and its efficiency compared to a kinodynamic PRM.

2.9.6 IREPA convergence

To better visualize the IREPA converge, we first work with a fixed number of nodes in the PRM (30). Fig. 2.28 shows several quantities during IREPA convergence.

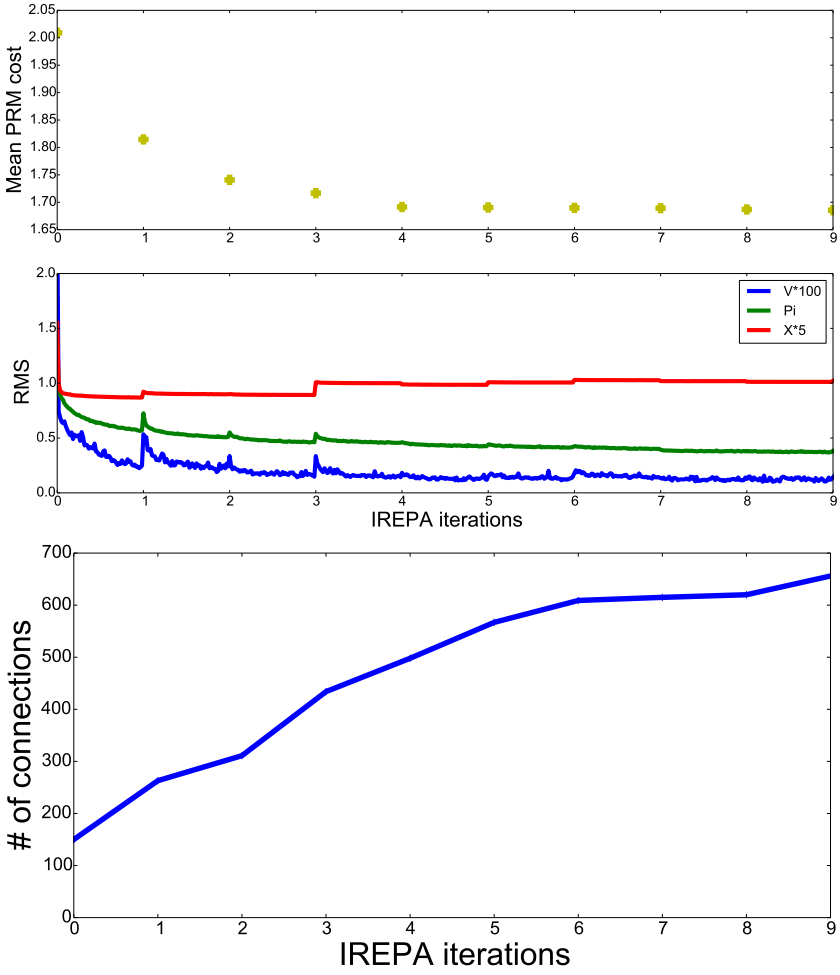


Figure 2.28: Evolution of the PRM and the approximators during the IREPA iterations. Top: decrease of the PRM mean edge cost (on a constant subset of 140 edges) and approximator RMS error. Bottom: number of connections between 30 reference nodes.

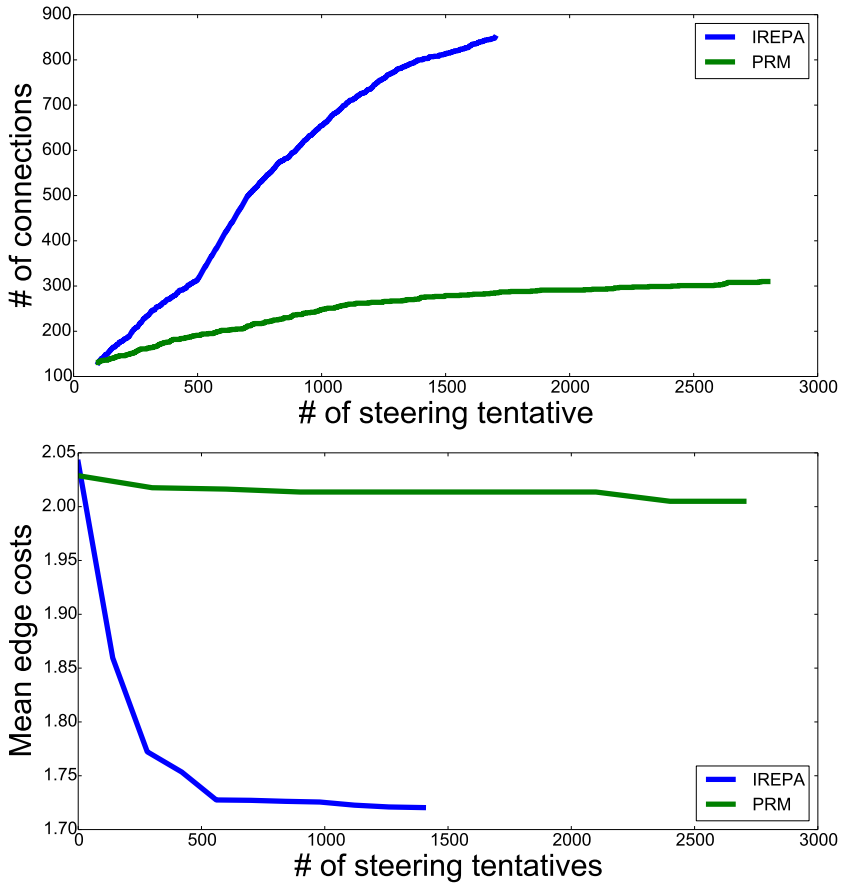


Figure 2.29: Comparison of the progress of IREPA versus PRM algorithms. Top: IREPA requires less evaluation of its (improved) steering method to establish more connections between distant nodes. Bottom: the connections computed by IREPA also have lower costs.

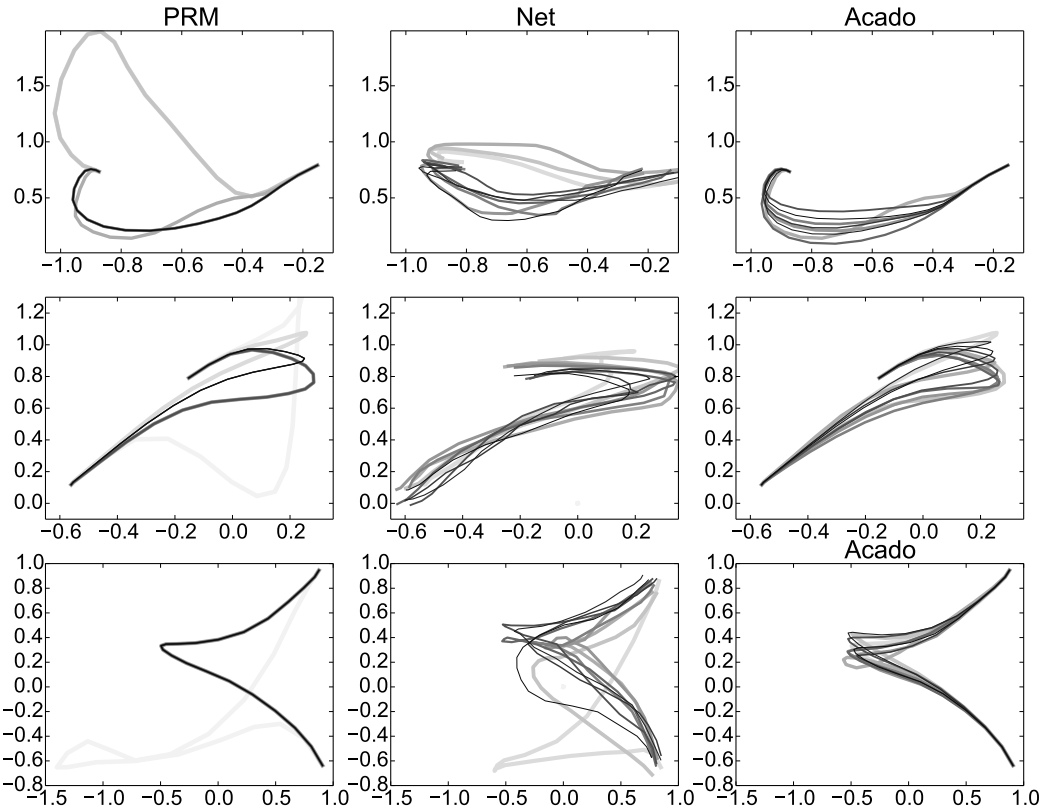


Figure 2.30: Typical examples of the trajectories stored in the PRM (left), in the approximators (middle) and computed by the steering method (right). The trajectories computed in the early IREPA iterations are in light gray, while dark gray is used for the trajectory computed in the last iterations. PRM and approximator trajectories tend to converge toward a same optimum, that the trajectory optimizer will easily optimize.

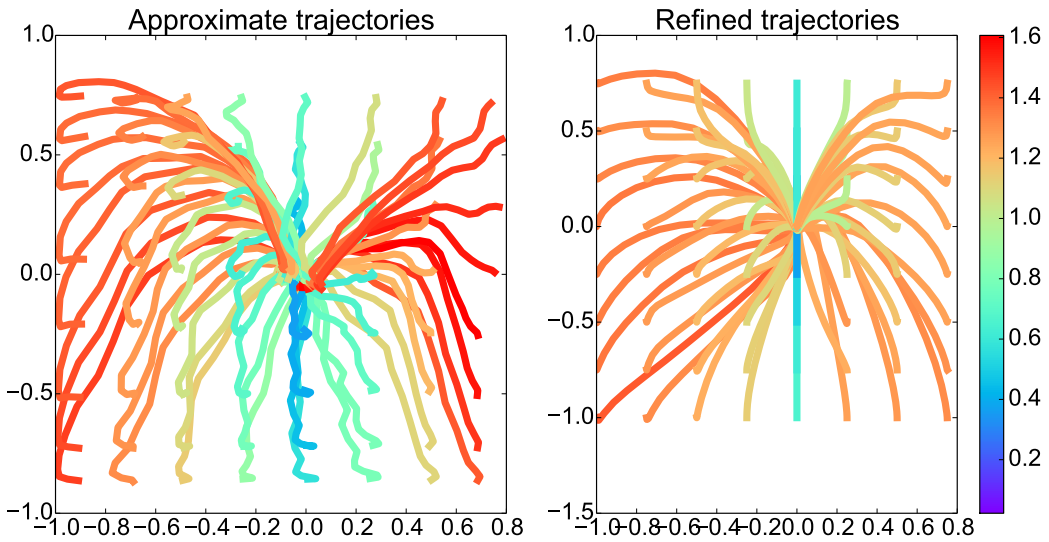
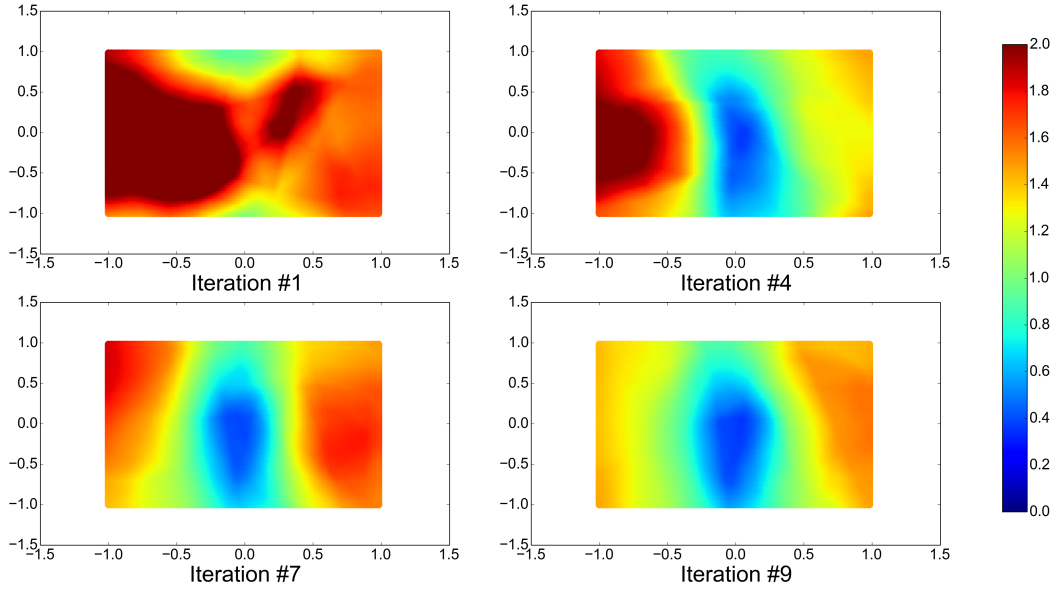


Figure 2.31: Bundle of trajectories (left) stored in the approximators, and (right) refined by the predictive controller. Each point of the plane corresponds to an initial configuration of the UAV (with 0 initial angle and 0 initial velocity), while the goal state is 0. The trajectories stored in the approximators do not exactly respect the boundary conditions. The refined trajectories are smoother, dynamically consistent and respect all the constraints. The color depicts the cost of each trajectory.

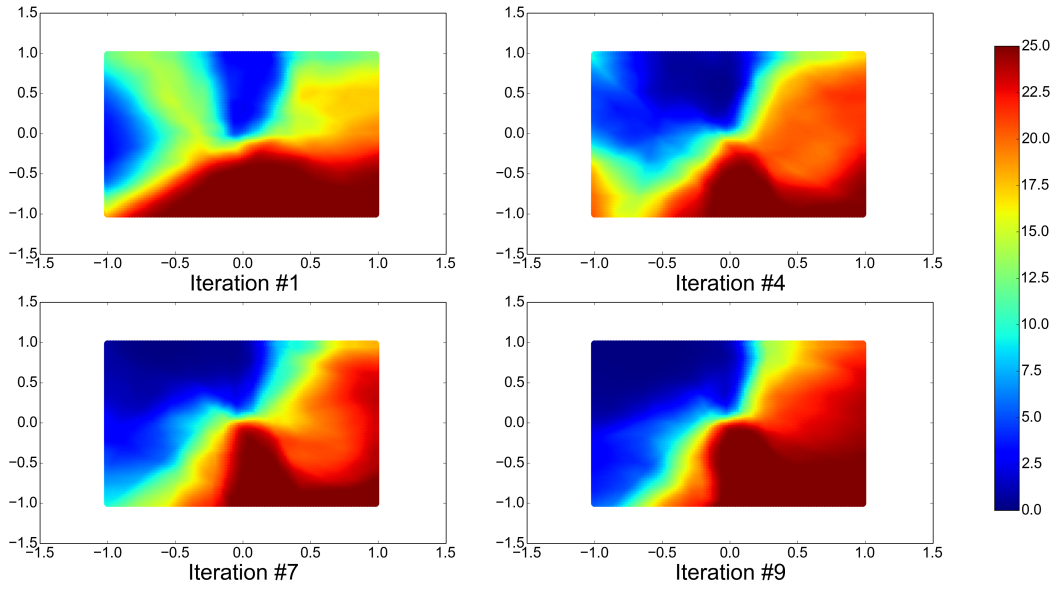
The mean cost on a fixed number of edges (the ones computed in the first iteration) decreases, while better local optimum are computed for the edges thanks to the improved approximators. Reciprocally, we plot the RMS along with the stochastic gradient descent: the RMS error converges to a local minimum at each IREPA iteration. Then, when new and better edges are computed at the next IREPA iteration, the RMS error can leave its local minimum. The number of connections in the PRM constantly grows and converges toward the maximum (830) in 15 iterations.

2.9.7 Propagation of the PRM

Then we compare the propagation of the roadmap when IREPA is used, or when a standard kinodynamic PRM is used. We compare the progress of the algorithms with respect to the number of evaluations of the steering method. Clearly, many evaluations of this local controller fail because the nodes that we try to connect may be out of the visibility range of the method. Once more, we fix the number of nodes in IREPA (only edges are added), while both nodes and edges are added by the PRM. Fig. 2.29 summarizes the results. IREPA is much faster at creating new connections, as the steering method increases its visibility range when the approximators converge. On the contrary, the poor metrics and the constant visibility range of the steering method used in the PRM lead to many connection failures. We also observe the mean cost to connect the 30 nodes initially sampled together.



Value function



Policy function

Figure 2.32: Scatter plot of the value (top) and policy (bottom) functions along IREPA iterations. The axes correspond to X and Z axis of the UAV, while the angle θ and velocities are null. The desired state is the origin (i.e. we plot $V([x, z, \theta = 0, v = 0_3], 0_6)$ and similarly for π). Both functions converge toward symmetric solutions. The policy is bang-bang, as expected.

In IREPA, this cost decreases as lower-cost edges are found. In the standard PRM the cost only decreases when a new path in the graph more efficiently connects two nodes. In conclusion, with respect to a standard PRM, IREPA converges much faster, and to better trajectories.

2.9.8 Results of the offline phase

We display in Fig. 2.30, 2.31 and 2.32 the trajectories computed and stored in the PRM and the approximators. Fig. 2.31 shows the evolution of both the PRM edges and the approximator. In the first IREPA iterations, suboptimal edges are computed as the steering method is cold started. This results in some inconsistency between the trajectories stored in the PRM, hence large RMS errors and poor quality of the trajectory approximators. Consequently, the steering method often fails to converge to a good optimum. Along the IREPA iterations, the trajectories both in the PRM and in the approximators tend to converge toward similar solutions, which are more consistent and closer to optimal. Consequently, the steering method is initialized with a good initial guess, and quickly converges toward an low-cost trajectory. Fig. 2.31 shows a set of trajectories for various initial conditions, stored in the approximator and refined by the predictive controller, at the end of the training phase. Fig. 2.32 shows a 2d projection of the Value function and the policy function. We know that the optimum should be symmetric and the policy function should saturate the controller. We can see that these properties are satisfied when IREPA converges. We also see that the RMS error of the HJB equation is small after convergence.

2.9.9 Online phase

The objective of the IREPA algorithm is to build the initial guess needed to warm start the predictive controller. We now empirically validate that this warm-start is useful and compare the two solutions proposed in the paper.

2.9.9.1 Warm-start

We compare the performances of the predictive controller without any initial guess (cold start), against its performances when warm-started with either the policy approximation (using a roll-out) or using the trajectory approximation. First of all, the predictive controller may often fail to find a admissible trajectory when no initial guess is used (cold start). Rates of success is displayed in Table 2.2. Cold-start results in 60% failures. On the opposite, any of the two warm-start results in between 90% and 95% of success (additional failures after 10 iterations are due to implementation problems). In the meantime, warm-start also results in more efficient trajectories. An important difference clearly appears in Fig. 2.33. The optimizer converges more quickly to the true optimum when initialized with the trajectory approximator $\hat{X}^*$ rather than by a roll-out of the policy $\hat{\pi}$. While the roll-out enforces the dynamic consistency, it also brings some convolution effects

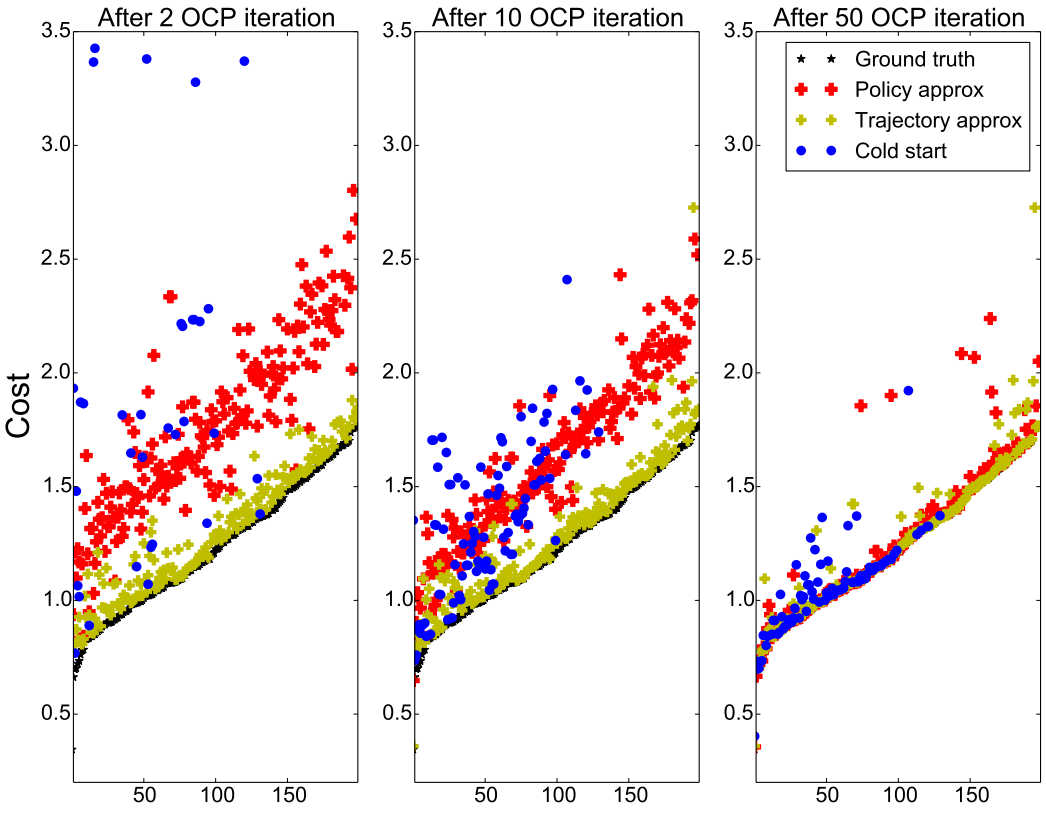


Figure 2.33: Convergence of the predictive controller initialized either by the policy roll-out or by the trajectory approximation. We consider a fixed set of 200 initial points, for which the (global) optimal Value is known. The points are ordered by cost. We plot the cost obtained from the two approximations after 2, 10 and 50 iterations of the predictive controller. Initializing with the trajectory approximation, the solver quickly converges toward the optimal trajectory. Initializing with the policy rollout also leads the solver to a good solution, although 50 iterations are needed. In practice, when controlling a robot, we cannot afford more than 5 iterations.

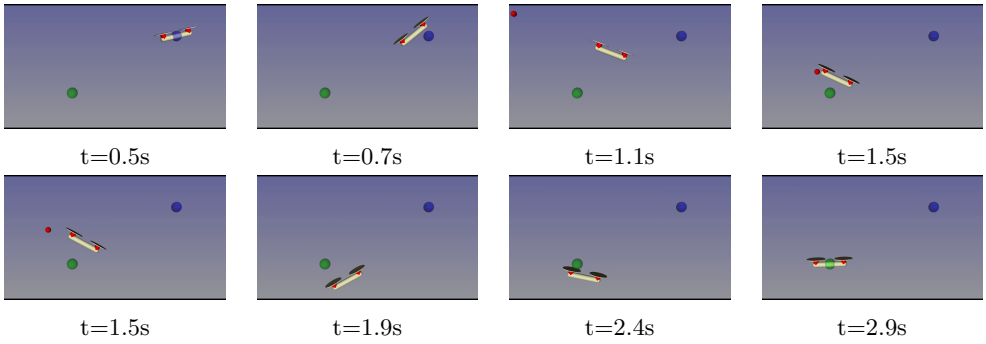


Figure 2.34: snapshots of an example trajectory obtained with the predictive controller. At time 1.29s, the UAV is impacted by an object which suddenly modifies its state. Using the initial guess, the predictive controller quickly rejects the disturbance and converges to the desired equilibrium.

Table 2.2: Success rate (%) of the steering method and average cost (± standard deviation) of the computed trajectories for different warm-start techniques (U: warm start from policy roll-out, J: warm start from state-control trajectory) and at different iterations (It.) of the algorithm.

	U		J		Cold start	
It.	%	Cost	%	Cost	%	Cost
2	90	0.47±0.18	88.5	0.07±0.07	31	2.05±1.85
5	90.5	0.34±0.10	93.5	0.07±0.08	45.5	0.37±0.36
50	85.5	0.04±0.09	85.0	0.03±0.07	42.0	0.06±0.10

due to the dynamic instability, that tends to confuse the solver, at least during the first iterations. After a large number of iterations, both warm starts converge to the same optimum, with similar rates of failure. The critical point with predictive control is to be able to quickly find an admissible trajectory. This is the case with both initial guess. However, on the robot, it is not possible to iterate more than 2 to 5 times. Under this assumption, the initialization by the trajectory approximator is much more efficient.

2.9.9.2 Predictive control

We use the initial guess in a predictive controller. An example of the resulting trajectory is displayed in Fig. 2.34. The robot has to reach a desired steady state. During the movement, it is hit by an object: this impact instantaneously changes its velocity. A new trajectory is first approximated then optimized, leading the robot to reject the disturbance and to reach its final position. Other trajectories are shown in the video (see `https://youtu.be/CbyCa7aeC7k`) with a double pendulum, a birotor, a quadrotor in various situations and a quadrotor with swinging load. Snapshots from the videos are displayed in Figs. 2.35 to 2.38.

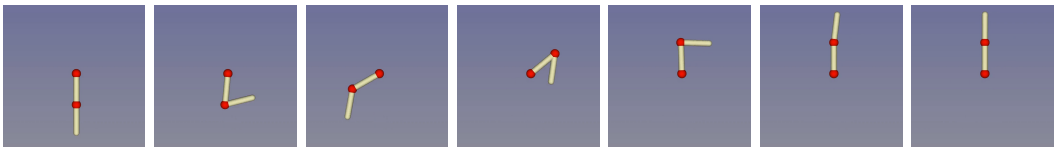


Figure 2.35: Swings of a double pendulum to reach a desired upright position.

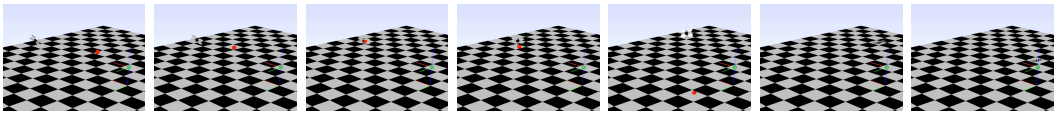


Figure 2.36: Quadcopter trajectory from steady state to steady state. The controller is disturbed by an impact at the middle of the trajectory.

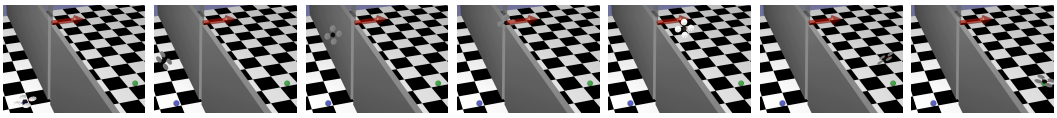


Figure 2.37: Quadcopter trajectory from steady state to a waypoint with tilted angle ($\pi/2$) and nonzero velocity, then to a terminal steady state. The obstacle is not explicitly modeled in the control law.

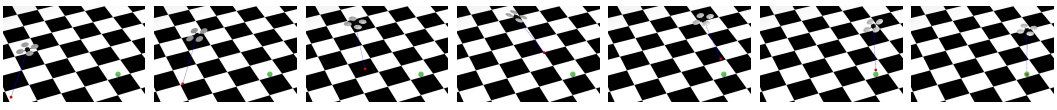


Figure 2.38: Quadcopter trajectory with a swinging load. The robot moves from steady state to steady state.

Table 2.3: Number of iteration before $T < 1.1T_{opt}$

initializer	initial $T[s]$	iteration
IREPA approximator	1.635 (given by the approximator)	2
static	8.0	41
	7.0	38
	6.0	34
	5.0	33
	4.0	31
	3.0	44
	2.5	51
	2.0	95
	1.5	154

2.9.9.3 Characterization of the optimum

The previous results are convincing about the capability of IREPA to drive the MPC solver to feasible solution. We want now to characterize whether the resulting trajectory is indeed the (global) optimum of the problem and compare the number of iterations needed by the nonlinear solver to converge to the optimal solution. We chose a setup where the optimal trajectory is known and experimentally observed how the MPC solver is able to obtain this solution depending on the provided warm start. For this test, we used the 3D quadrotor model presented in Section 2.3.1. The neural network is trained following the same procedure, over 15 IREPA iterations. We consider the quadrotor in hover state subject to a perturbation that instantaneously changes its lateral velocity from $0m/s$ to $5m/s$. The system is controlled to come back to its initial position as fast as possible. As we have seen in Section 2.5.2.1, complete convergence to the optimal solution can be very long. Instead, we consider here that the solution is found if $T < 1.1T_{opt}$, where T is the trajectory duration and $T_{opt} = 1.899s$ is the minimal duration found on this example. The results are compared to a static initial guess (which would be the initial guess found by a MPC controller since the perturbation is considered instantaneous) and with different initial guess for the T parameter. The results are summarized in Tab. 2.3. They show that the initial guess generated by our approximator allows a convergence at least 15 times quicker than a static initial guess.

2.10 Conclusion

The purpose of this chapter was to introduce MPC as an efficient theoretical framework able to generate various kinds of movements for various robot dynamics and task contexts. The main interest of this approach is that the theoretical work and technological implementation do not have to change when the robot, the environment or the task change. We empirically demonstrated that this approach is well suited to address several typical problems of UAVs such as generating and achieving high-dynamic maneuvers, generating oscillatory patterns with a hanging load, dynamically moving through an obstacle field and generating in-contact movements with an aerial manipulator. On the motion-generation side, this study led to scale the computational load using a prototype implementation (around a few seconds of computations to compute a trajectory from scratch, a tenth of a second to update a trajectory in MPC). These performances are sufficient to allow the implementation of the approach on the real robot. Despite the interest of the MPC framework, some open scientific questions remain open before we can simply apply it on real robots. The open questions are a motivation for the next two chapters. A first scientific lock is around the need to engineer the cost function to obtain a good robot behavior. Although the premise of optimal control is that it is enough to choose a simple cost function that encodes the task to be achieved by the robot, defining the cost function is a little bit more complex in practice. The cost function should indeed also regularize the system movements (in particular when closing the loop in a MPC fashion), while we generally have to setup more aggressive cost for exploration (i.e. when using the optimal-control formulation in a planning fashion). Adjusting the relative importance of the several terms that are composing the cost function may quickly become tricky. A practical consequence is that, although the computation costs are very satisfactory, we were not able to set up MPC in a generic manner, apart on the quadrotor alone. In the next chapter we will focus on the construction of cost function more suitable for MPC along with their implementation on physical robots, using a hierarchical strategy. Another important open question is the prior information that is given to the optimizer, to warm-start the search. For cyberphysic systems with a slow or repetitive dynamics (e.g. chemical tanks, tanks), warm-starting the search with the solution of the previous control cycle is most of the time sufficient to ensure a good behavior of the controller. It is, once more, trickier for versatile robots in changing environments, where dynamics, tasks and obstacles quickly evolve and make deprecated the previous solution. Many works available in the literature relies on heuristic solution to build a warm start at run time. We rather believe that quickly building good approximations to warm start the search might be done by combining motion planning and machine learning, based on offline exploration of the robot motion capabilities. In this chapter, we reported a collaborative research action that led to storing the initial guess inside a neural network. Although the approach is interesting, we are not yet able to generalize it on systems whose dynamics are more complex, like legged robots. In the last chapter, we will investigate more deeply this direction, in order to quickly

generate complex locomotion plans using a motion planner, whose internal model results from an offline exploration phase.

Chapter 3

Regularized Hierarchical Differential Dynamic Programming

In the previous chapter, we reported how a classical optimal control method can be applied to a real robotic problem. This chapter presents a new algorithm for optimal control (OC) of nonlinear dynamical systems. The main feature of this algorithm is that it allows the specification of the control objectives as a hierarchy of tasks, each task representing an action that the robot should perform. Each task is described by a cost function that the algorithm tries to minimize, while not affecting the tasks of higher priority. The concept of strict priority allows for an easier and more robust specification of the control objectives, without hand-tuning of task weights. The hierarchy also makes it possible to properly regularize the behavior of each task independently. For the first time, we properly define the problem of regularizing the task cost functions in the presence of a hierarchy and propose an algorithm to compute an approximate solution. Several simulated scenarios with different robots compare our solution with other state-of-the-art methods, validating the interest of the hierarchy in OC and empirically demonstrating the importance of regularization to generate feasible behaviors.

3.1 Introduction

Optimal control algorithms allow the user to generate motion for arbitrary systems by specifying a cost function to minimize. Exact methods for solving optimal control rely on optimality principles by either Bellman or Pontryagin. While solving an optimal control problem using these approaches is intractable in most realistic cases, algorithms exist that compute approximate local optima, which are often good enough in practice [Von Stryk 1993, Mombaur 2005]. However, as shown in the previous chapter, designing a cost function that results in the desired behavior is much more complex than it may look like. Especially for systems with many degrees of freedom (DoFs) a cost function is typically a weighted sum of several elementary costs, each one representing a task that the robot should perform [Tassa 2012, Koenemann 2015]. For instance, to make a humanoid robot walk, we can control the trajectory of its center of mass and its swinging foot, its angular momentum and its whole-body posture. If we also add manipulation objectives, it is clear that the number of tasks rapidly grows. In such cases, finding the right

weights for the different terms of the cost function may be extremely time consuming. Moreover, weights are typically not robust to task variations, e.g. the weights used for walking may be very different from the weights needed for running. **Often, rather than using weights, it is simpler to specify strict priorities between tasks.** This means that in case of conflict between two tasks, we might require the most important task to be achieved at the expense of the other. For instance, the task of avoiding collisions has clearly higher priority than the task of minimizing the motor commands. The concept of strict priorities is indeed widespread in robotics for inverse-kinematics [Escande 2010, Siciliano 1991] and inverse-dynamics [Saab 2013, Del Prete 2015] controllers. In optimal control strict priorities are typically approximated using much larger weights for the high-priority tasks. However this approach does not scale well when the number of priority levels grows because it can lead to poor numerical conditioning. This motivated our first work on Hierarchical Optimal Control (HOC) [Del Prete 2014], in which we introduced strict priorities in the optimal control problem formulation. Later [Romano 2015], we proposed another algorithm to solve the HOC problem, Hierarchical Differential Dynamic Programming (HDDP). In this chapter we propose an improved version of HDDP, which properly handles the regularization of the tasks. While the problem of regularizing the cost function is often overlooked in the literature, we show that it is actually paramount when using strict priorities.

3.1.1 The Role of Regularization

The problem of regularization (or damping) is well-known in robotics [Chiaverini 1994,Decré 2013] and optimization [Dimitrov 2015], but for different reasons. Optimization problems are regularized to avoid poor numerical conditioning, which leads to large numerical errors due to computer arithmetic of finite precision. In this context, the regularization parameters typically take very small values (e.g. 10^{-9}). This regularization, which we refer to as *algorithm regularization*, modifies the original problem, thus introducing a small error in the resulting solution. This error is however largely compensated for by the improved behavior of the numerical solver.

In inverse-dynamics/inverse-kinematics control instead, regularization is used to prevent large motor commands, which occur e.g. in the neighborhood of kinematic singularities [Chiaverini 1994]. In this context, regularization parameters take much larger values (e.g. 10^{-3}). This regularization, which we refer to as *task regularization*, is a core part of the cost functions describing the tasks—rather than a parameter of the algorithm. Task regularization is even more critical in optimal control: without regularization the resulting control would overexploit the robot actuation capabilities (e.g. saturating the motor limits). This behavior is undesirable in most situations in autonomous robots.

The difference between task regularization and algorithm regularization is part of the know-how, but yet not well defined in robotics. This is because in inverse-kinematics/dynamics they are both implemented through the damping factor of the

pseudo-inverses [Chiaverini 1994].

Despite its importance, the problem of task regularization is often not explicitly mentioned, or relegated to a small paragraph towards the end of the chapter. This is because typically task regularization does not affect the mathematical developments that are the subject of the publication. However, this is no longer the case when considering a hierarchical (or lexicographic) optimization [Tazaki 2014, Isermann 1982]. Indeed we will show that the hierarchical minimization of least-squares functions—which has been extensively studied in robotics [Dimitrov 2015, Escande 2014] and is at the core of our algorithm—becomes nonconvex when introducing task regularization. Moreover, while the unregularized hierarchical problem can be defined as the limit of the weighted problem for the ratio of the weights going to infinity, this is no longer the case if we introduce regularization. This implies that the solution of the regularized hierarchical problem cannot be approximated with a classic optimization using large weights. These two properties highlight the fact that task regularization should be taken into account from the very beginning when dealing with hierarchical optimization. The contribution of this chapter goes thus in this direction, by introducing a regularized version of HDDP, called Regularized HDDP (RHDDP).

3.1.2 State of the Art

The problem of hierarchical trajectory optimization has been rarely discussed in the literature. Recently, hierarchical linear model predictive control has been used to generate safe walking motion for a biped robot walking in a crowd [Sherikov 2016]. A similar formulation has been used to solve the minimum-time control problem for industrial robots [Homsi 2016]. In both cases, nonlinear constraints have been linearized, leading to classic hierarchical least-squares problems.

To the best of our knowledge, only one work in the literature dealt with *nonlinear* hierarchical trajectory optimization [Tazaki 2014]. With respect to our approach we can find several differences. The main advantage of their algorithm [Tazaki 2014] is that it can handle inequality constraints. However, this prevented them from exploiting the sparsity of the optimal control problem, which we do thanks to the Differential Dynamic Programming (DDP) formulation. Moreover, they did not deal with the problem of task regularization, which is the main focus of this chapter.

This work is based on two previous conference publications [Del Prete 2014, Romano 2015]. In our first work [Del Prete 2014] we introduced strict task prioritization in the optimal control formulation. This algorithm did not exploit the intrinsic sparsity of the optimal control problem, which can lead to a reduction of the computational complexity from cubic to linear in the number of time steps. In our second work [Romano 2015] we addressed this problem by extending the DDP algorithm to account for strict priorities between the cost functions. In this chapter we present an improved version of the HDDP algorithm, together with extensive simulations with several robotic systems to validate our approach.

3.1.3 chapter Overview

This chapter has three contributions. First, we propose the first well-founded definition and resolution of a hierarchy of quadratic objectives with regularization. Second, we use this solution to derive the first algorithm to solve HOC with regularization, while exploiting the sparsity of the problem. Finally, we demonstrate the importance of our approach by several case studies on various simulated robot models.

Before discussing about HOC, Section 3.2 treats the problem of Hierarchical Quadratic Programming (HQP), which is strongly related to HDDP. The subject of Hierarchical Least-Squares Programming (HLSP, a special case of HQP) is well-known in robotics and has been extensively studied [Dimitrov 2014, Escande 2014, Escande 2010] and applied [Herzog 2016] in recent years. However, the problem of task regularization has never been properly addressed. In particular, we show that the regularized HLSP problem is not convex in general—while HLSP always is. We then propose a convex relaxation of regularized HQP, which gives a (possibly) suboptimal solution that is guaranteed to satisfy the priority constraints.

Section 3.3 introduces the problem of Parametric HQP (PHQP), which consists in minimizing a set of quadratic cost functions with respect to (w.r.t.) a subset of their variables, treating the other variables as problem parameters. This is exactly what happens in the DDP algorithm, where the optimization is performed w.r.t. the control variables, while the state variables are treated as problem parameters. This allows DDP to compute feedback control laws rather than open-loop control trajectories.

Then, Section 3.4 and 3.5 present the Regularized HDDP algorithm, exploiting the results already presented for PHQP. Section 3.6 reports numerical simulations on different robotic systems, comparing RHDDP with HDDP and DDP. Finally, Section 3.7 discusses the results and Section 3.8 draws the conclusions.

3.1.4 Notation

The following notation is used throughout the chapter:

- (A, B) is a short form for $\begin{bmatrix} A^\top & B^\top \end{bmatrix}^\top$.
- $\mathcal{N}(A)$ is the null-space projector of the matrix A.
- $A^\dagger$ is the Moore-Penrose pseudo-inverse of the matrix A.
- $\partial_y g$ is the partial derivative of a multivariable function $g(\cdot)$ with respect to one of its variables y; $\partial_{yz} g$ is the partial second-order derivative with respect to y and z.
- y is a generic variable while x and u are respectively the state and control variable in an optimal control problem. We also denote by X and U the state and control sequences (i.e. $X = (x_0 \dots x_N)$).

- the symbol $\hat{.}$ is used for all quantities related to the regularized costs (e.g. the regularized control law $\hat{u}$).
- the symbol $\tilde{.}$ is used for the hybrid control law.
- the symbol $.^*$ is used for the optimal values.
- the symbol $\bar{.}$ is used for the cumulative quantities, i.e. quantities that are associated to a set of tasks (rather than a single task).

3.2 Hierarchical Quadratic Programming (HQP)

We first recall the Hierarchical Quadratic Programming problem without regularization, as it is typically presented in the literature [Escande 2014]. For applications in robot control, most of the time this problem is not interesting because it results in large/discontinuous motor commands. For this reason, we present then the regularized HQP problem, which does not suffer from this issue, and we discuss its properties.

3.2.1 Problem Statement

Suppose having n_l quadratic functions:

$$g^{(l)}(y) = \frac{1}{2} y^\top H^{(l)} y + h^{(l)\top} y, \quad l = 1, \dots, n_l$$

which we want to minimize in a hierarchical way:

$$\begin{aligned} g^{(l)*} = \underset{y}{\text{minimize}} \quad & g^{(l)}(y) \\ \text{subject to} \quad & g^{(j)}(y) = g^{(j)*} \quad \forall j < l \end{aligned} \tag{3.1}$$

Clearly, we can expect the Hessians of all the functions $g^{(j)}(y)$ to be singular—except for the last one. If that was not the case, all the functions of priority lower than a function with a full-rank Hessian could not be optimized at all. In this form, the priority constraints are quadratic, which make problem (3.1) nonconvex. However, it is well-known that, for the least-squares case, we can replace them with linear constraints (as we will recall in Section 3.2.3), making (3.1) convex. We will show that this is no longer the case if we introduce regularization.

3.2.2 Regularizing the Problem

Suppose having a regularized version of each objective function:

$$\hat{g}^{(j)}(y) = \frac{1}{2} y^\top \hat{H}^{(j)} y + \hat{h}^{(j)\top} y, \quad j = 1, \dots, n_l$$

We assume that the regularized Hessians are positive-definite. A typical case of regularization consists in adding a scaled identify matrix to the Hessians, i.e.

$\hat{H}^{(j)} = H^{(j)} + \lambda I$, while leaving the gradients unvaried, i.e. $\hat{h}^{(j)} = h^{(j)}$. The regularized HQP problem is then:

$$\begin{aligned} \hat{y}^{(l)*} = & \arg\min_{y} \quad \hat{g}^{(l)}(y) \\ & \text{subject to} \quad g^{(j)}(y) = g^{(j)}(\hat{y}^{(j)*}) \quad \forall j < l \end{aligned} \tag{3.2}$$

Note that we do not use the regularized functions in the priority constraints because that would leave no null space to optimize the secondary objectives. Again, because of the quadratic equality constraints, problem (3.2) is not convex. We show now how to replace them with linear constraints that are sufficient (but not necessary) to guarantee the satisfaction of the original quadratic constraints, resulting thus in a convex relaxation of (3.2).

3.2.3 Reformulating the Priority Constraints

Let us use the first two objectives of the hierarchy to illustrate this technique, which can be then easily generalized to the following objectives. The minimization of the first objective is unconstrained, so we can compute $\hat{y}^{(1)*}$ as:

$$\hat{y}^{(1)*} = -(\hat{H}^{(1)})^{-1}\hat{h}^{(1)}$$

The priority constraint for the optimization of the second objective is then:

$$\frac{1}{2} y^\top H^{(1)} y + h^{(1)\top} y = g^{(1)}(\hat{y}^{(1)*})$$

We introduce now a simple change of variable to simplify the derivation: $y = \hat{y}^{(1)*} + y^{(2)}$. This leads us to:

$$\frac{1}{2} y^{(2)\top} H^{(1)} y^{(2)} + \hat{y}^{(1)*\top} H^{(1)} y^{(2)} + h^{(1)\top} y^{(2)} = 0 \tag{3.3}$$

When the HQP problem is unregularized and the functions $g^{(l)}(y)$ are least-squares functions we can replace (3.3) with an equivalent linear constraint, which makes the HQP convex:

$$\frac{1}{2} y^{(2)\top} H^{(1)} y^{(2)} = 0 \quad \iff \quad y^{(2)} = \mathcal{N}(H^{(1)}) z,$$

where z is an arbitrary variable, and we exploited the fact that for an unregularized HQP $\hat{y}^{(1)} = -(H^{(1)})^\dagger h^{(1)}$ and that for least-squares functions $h^{(1)\top}\mathcal{N}(H^{(1)}) = 0$. However, this is not the case for the regularized HQP, which is in general nonconvex. We suggest to replace the quadratic priority constraints (3.3) with linear constraints that are sufficient but not necessary to ensure (3.3). By doing so we get a convex optimization problem whose solution is in general suboptimal for the original problem, but it is guaranteed to satisfy the priority constraints. A sufficient condition for (3.3) to hold is to select $y^{(2)}$ in the null space of $H^{(1)}$ (to nullify the first two

terms) and $h^{(1)\top}$ (to nullify the last term), that is:

$$y^{(2)} = \mathcal{N}((H^{(1)}, h^{(1)\top}))z = N^{(1)}z, \tag{3.4}$$

Note that for a hierarchy of least-squares functions being in the null space of $H^{(1)}$ would be sufficient [Escande 2014] (but still not necessary) because the gradient would always be zero in the null space of the Hessian, i.e. $h^{(1)\top}\mathcal{N}(H^{(1)}) = 0$.

3.2.4 Solving the Second Minimization

The minimization of the second objective is then a QP:

$$\begin{aligned} \underset{y,z}{\text{minimize}} \quad & \hat{g}^{(2)}(y) \\ \text{subject to} \quad & y = \hat{y}^{(1)*} + N^{(1)}z \end{aligned}$$

Eliminating the constraints and setting the gradient of the cost to zero we get:

$$\hat{y}^{(2)*} = \hat{y}^{(1)*} - (N^{(1)}\hat{H}^{(2)}N^{(1)})^{\dagger}(\hat{h}^{(2)} + \hat{H}^{(2)}\hat{y}^{(1)*})$$

3.2.5 Solving the Whole Hierarchy

Generalizing this to an arbitrary number of functions n_l, we get the following recursive solution:

$$\hat{y}^{(l)*} = \hat{y}^{(l-1)*} - \bar{H}^{(l)\dagger}\bar{h}^{(l)},$$

where:

$$\begin{aligned} \bar{H}^{(l)} &\triangleq N^{(l-1)}\hat{H}^{(l)}N^{(l-1)} \\ \bar{h}^{(l)} &\triangleq \hat{h}^{(l)} + \hat{H}^{(l)}\hat{y}^{(l-1)*} \\ N^{(l)} &\triangleq N^{(l-1)}\mathcal{N}((H^{(l)}, h^{(l)\top})N^{(l-1)}), \end{aligned}$$

The recursion is initialized with $N^{(0)} = I$ and $\hat{y}^{(0)*} = 0$.

3.2.6 A Simple Example

We can look at a simple example to give some insights about the proposed convex relaxation of the regularized HQP problem. Let us minimize in a hierarchical way two quadratic functions $g^{(1)}$ and $g^{(2)}$ of the 3-dimensional variable $y = (y_1, y_2, y_3)$. The function $g^{(1)}$ depends only on y_1 and y_2, and its Hessian has rank 2. The function $g^{(2)}$ instead has a full-rank Hessian. The regularized versions of $g^{(1)}$ and $g^{(2)}$ differ only for their Hessians, which are regularized in this way: $\hat{H}^{(l)} = H^{(l)} + \lambda I$.

In this case, the set of solutions of the priority constraint for the second level (i.e. $g^{(1)}(y) = g^{(1)}(\hat{y}^{(1)*})$) is described by the surface of a 3d cylinder (see Fig. 3.1). This cylinder has an axis that is parallel to y_3, and passes through the minimizer of $g^{(1)}$. The diameter of the cylinder is proportional to the regularization parameter λ. For $\lambda = 0$ the cylinder collapses to a line, and hence the priority constraint

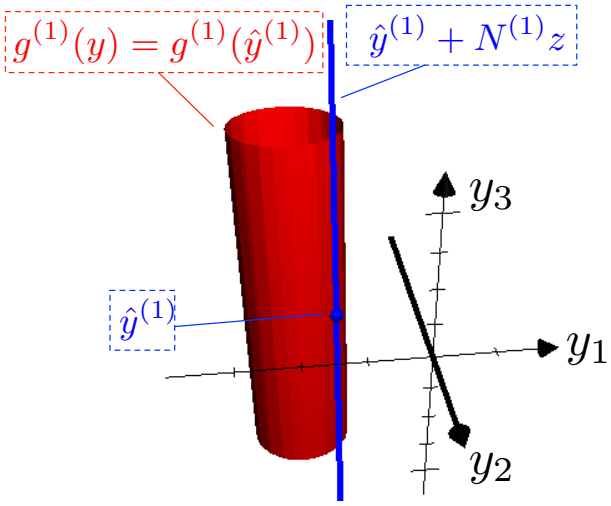


Figure 3.1: 3D example depicting the difference between the nonconvex priority constraints of the regularized HQP problem (red cylinder) and their convex relaxation (blue line) proposed in this chapter.

becomes linear, making the HQP convex. For $\lambda > 0$ the problem is not convex because the constraint of lying over the surface of a cylinder is clearly nonlinear. Rather than looking for the minimizer of $\hat{g}^{(2)}$ over the surface of this cylinder, our convex relaxation looks only over a line, which is parallel to y_3 and passes through $\hat{y}^{(1)}$.

In general, it is difficult to quantify the suboptimality of the resulting solution. Depending on the problem data we could find the global optimum, or we may be significantly suboptimal. Another approach to solve the regularized HQP could be to linearize the quadratic priority constraints and use Sequential Quadratic Programming [Wright 1999]. However, the linear approximation of the quadratic constraints is in general rather poor, and leads the algorithm to take very small steps, slowing down convergence[1]. Alternatively, we could consider the interior of the nonconvex set defined by the priority constraints (which is a convex set) and use an Interior-Point method [Wright 1999]. However, even this approach would require solving several QPs for each level of the hierarchy. Our convex relaxation instead allows us to compute an approximate solution by solving a single QP for each hierarchy level.

[1]We performed some simple tests, which showed that the SQP algorithm can find better solutions than our convex relaxation, but it usually takes tens of iterations just to find a solution of the same quality. Since this is not the main focus of the chapter, for the sake of conciseness we do not report these results here.

3.3 Parametric Hierarchical Quadratic Programming (PHQP)

The problem of HQP becomes more complex if we are not optimizing w.r.t. all the decision variables, but only w.r.t. a subset of them (as it happens in DDP). Let us split the decision variables into two subsets $y = (x, u)$:

$$g^{(l)}(x,u) = \frac{1}{2} x^\top H_{xx}^{(l)} x + \frac{1}{2} u^\top H_{uu}^{(l)} u + x^\top H_{xu}^{(l)} u + h_x^{(l)\top} x + h_u^{(l)\top} u, \qquad j = 1, \ldots, n_l$$

We also have a regularized version of each cost function:

$$\hat{g}^{(l)}(x,u) = \frac{1}{2} x^\top \hat{H}_{xx}^{(l)} x + \frac{1}{2} u^\top \hat{H}_{uu}^{(l)} u + x^\top \hat{H}_{xu}^{(l)} u + \hat{h}_x^{(l)\top} x + \hat{h}_u^{(l)\top} u, \qquad j = 1, \ldots, n_l$$

Rather than optimizing w.r.t. y, we want to optimize w.r.t. u only, treating x as a problem parameter:

$$\begin{aligned} \hat{u}^{(l)*}(x) = \arg\min_u \quad & \hat{g}^{(l)}(x,u) \\ \text{subject to } & g^{(j)}(x,u) = g^{(j)}(x, \hat{u}^{(j)*}) \quad \forall j < l \end{aligned} \tag{3.5}$$

3.3.0.1 Solving the First Minimization

For the first objective we have an unconstrained optimization, which we can solve by computing the cost gradient and setting it equal to zero:

$$\hat{u}^{(1)*} = \underbrace{-(\hat{H}_{uu}^{(1)})^{-1} \hat{h}_u^{(1)}}_{\hat{k}^{(1)}} - \underbrace{(\hat{H}_{uu}^{(1)})^{-1} \hat{H}_{ux}^{(1)}}_{\hat{K}^{(1)}} x$$

3.3.0.2 Solving the Second Minimization

The priority constraint for the optimization of the second objective is then:

$$g^{(1)}(x,u) = g^{(1)}(x, \hat{u}^{(1)*})$$

As before, we introduce a change of variable to simplify the derivation: $u = \hat{u}^{(1)*} + u^{(2)}$. This leads us to:

$$\frac{1}{2} u^{(2)\top} H_{uu}^{(1)} u^{(2)} + (\hat{u}^{(1)*\top} H_{uu}^{(1)} + x^\top H_{xu}^{(1)} + h_u^{(1)\top}) u^{(2)} = 0 \tag{3.6}$$

Similarly to HQP, if the problem were unregularized and the cost functions were least-squares, we could replace this quadratic constraint with an equivalent linear constraint (as we did in [Romano 2015]). Contrary to the regularized HQP, selecting $u^{(2)}$ in the null space of $H_{uu}^{(1)}$ and $h_u^{(1)\top}$ is not sufficient to ensure the satisfaction of

(3.6), because of the term $x^\top H_{xu}^{(1)} u^{(2)}$. A sufficient condition is to select $u^{(2)}$ to be also in the null space of $H_{xu}^{(1)}$:

$$u^{(2)} = \mathcal{N}((H_{uu}^{(1)}, h_u^{(1)\top}, H_{xu}^{(1)}))z = N^{(1)} z \tag{3.7}$$

This new constraint is linear and it is a sufficient condition for the original quadratic constraint. We propose to relax problem (3.5) by replacing (3.6) with (3.7). The solution of the second objective minimization is then:

$$\hat{u}^{(2)*} = -(\bar{H}_{uu}^{(2)})^\dagger \bar{h}_u^{(2)} - (\bar{H}_{uu}^{(2)})^\dagger \bar{H}_{ux}^{(2)} x,$$

where:

$$\begin{aligned} \bar{H}_{uu}^{(2)} &\triangleq N^{(1)} \hat{H}_{uu}^{(2)} N^{(1)} \\ \bar{h}_u^{(2)} &\triangleq \hat{h}_u^{(2)} + \hat{H}_{uu}^{(2)} \hat{k}^{(1)} \\ \bar{H}_{ux}^{(2)} &\triangleq \hat{H}_{ux}^{(2)} - \hat{H}_{uu}^{(2)} \hat{K}^{(1)} \end{aligned}$$

Note that for least-squares functions, being in the null space of H_{uu} would still be sufficient.

3.3.0.3 Solving the Whole Hierarchy

Generalizing this to an arbitrary number of functions n_l, we get the following solution:

$$\hat{u}^* = \bar{k} - \bar{K} x,$$

which can be computed using Algorithm 2.

Algorithm 2 Parametric Hierarchical Quadratic Programming

function PHQP($\{H_{uu}^{(l)}, H_{xu}^{(l)}, h_u^{(l)}, \hat{H}_{uu}^{(l)}, \hat{H}_{xu}^{(l)}, \hat{h}_u^{(l)}\}_l$)
  $N \leftarrow I, \bar{k} \leftarrow 0, \bar{K} \leftarrow 0$
  for l=1:n_l **do**
    $\bar{k} \leftarrow \bar{k} - (N \hat{H}_{uu}^{(l)} N)^\dagger (\hat{h}_u^{(l)} + \hat{H}_{uu}^{(l)} \bar{k})$
    $\bar{K} \leftarrow \bar{K} + (N \hat{H}_{uu}^{(l)} N)^\dagger (\hat{H}_{ux}^{(l)} - \hat{H}_{uu}^{(l)} \bar{K})$
    $N \leftarrow N \mathcal{N}((H_{uu}^{(l)}, h_u^{(l)\top}, H_{xu}^{(l)}) N)$
  return $(\bar{k}, \bar{K})$

3.4 Hierarchical Dynamic Programming

3.4.1 Problem Statement

Let us consider a discrete-time nonlinear dynamical system:

$$x_{i+1} = f(x_i, u_i), \quad \text{for } i = 0, \ldots, N-1, \tag{3.8}$$

where $f(\cdot) : \mathbb{R}^n \times \mathbb{R}^m \mapsto \mathbb{R}^n$ is the dynamics function, $x_i \in \mathbb{R}^n$ is the state at time step i, and $u_i \in \mathbb{R}^m$ is the control at time step i. Assume that we want the system to perform n_l tasks, with task 1 having the highest priority, and task n_l the lowest. The l-th task is represented by an arbitrary cost function:

$$c^{(l)}(X,U) := \sum_{i=0}^{N-1} \phi_i^{(l)}(x_i,u_i) + \phi_N^{(l)}(x_N) \ , \tag{3.9}$$

where $\phi_i^{(l)}$ is the running cost and $\phi_N^{(l)}$ is the final cost. We also have a regularized version of the cost functions:

$$\hat{c}^{(l)}(X,U) := \sum_{i=0}^{N-1} \underbrace{\phi_i^{(l)}(x_i,u_i) + \frac{\lambda^{(l)}}{2}||u_i||^2}_{\hat{\phi}_i^{(l)}(x_i,u_i)} + \phi_N^{(l)}(x_N) \ ,$$

where $\lambda^{(l)} \in \mathbb{R}$ is a regularization parameter. Other regularizations may be used if needed, as long as the Hessian of the regularization function with respect to U is positive definite. Our problem consists in finding the control and state sequences (U^*, X^*) that solve the following hierarchical optimal control problem, denoted as $\text{HOC}^{(\text{l})}$:

$$\begin{aligned} \underset{X,U}{\text{minimize}} \quad & \hat{c}^{(l)}(X,U) \\ \text{subject to} \quad & x_{i+1} = f(x_i,u_i), \quad \text{for } i = 0,\dots,N-1 \\ & x_0 \text{ fixed} \\ & c^{(j)}(X,U) = c^{(j)}(X^{(j)*},U^{(j)*}) \quad \forall j < l \ , \end{aligned} \tag{3.10}$$

for $l = 1$ to n_l, where $(X^{(j)*},U^{(j)*})$ is the optimum obtained by solving the $\text{HOC}^{(\text{j})}$.

3.4.2 Dynamic Programming with Regularization

We tackle problem (3.10) by applying the dynamic programming algorithm [Kirk 1970]. The principle of dynamic programming states that optimizing over the whole trajectory is equivalent to performing a sequence of optimizations over a single time step, starting from the end of the horizon and moving backwards in time. In other words, dynamic programming exploits the sparsity of the OC problem to reformulate it into a sequence of smaller problems (one for each time step). As a consequence, the complexity of the resolution is linear in the number of time steps (while it would be cubic with a nonsparse solver).

Let us start by defining the regularized and unregularized *cost-to-go* at step i

for task l as:

$$c_i^{(l)}(x_i, U_i) \triangleq \sum_{j=i}^{N-1} \phi_j^{(l)}(x_j, u_j) + \phi_N^{(l)}(x_N)$$
$$\hat{c}_i^{(l)}(x_i, U_i) \triangleq \sum_{j=i}^{N-1} \hat{\phi}_j^{(l)}(x_j, u_j) + \phi_N^{(l)}(x_N)$$

The total cost corresponds to the cost-to-go for $i = 0$.

3.4.2.1 First Task

For task 1, we define the value function (i.e. the optimal cost-to-go), for the regularized and unregularized cost:

$$V_i^{(1)}(x_i) \triangleq \underset{U_i}{\text{minimize}}\, c_i^{(1)}(x_i, U_i)$$
$$\hat{V}_i^{(1)}(x_i) \triangleq \underset{\hat{U}_i}{\text{minimize}}\, \hat{c}_i^{(1)}(x_i, \hat{U}_i)$$

We call $U_i^{(1)}(x_i)$ and $\hat{U}_i^{(1)}(x_i)$ the optimal control laws resulting from these minimizations, and $\hat{X}^{(1)}$ the state sequence resulting from applying $\hat{U}_0^{(1)}(x_0)$. By applying Bellman's principle of optimality [Bellman 2015] we can reformulate these minimizations over the whole future control sequence into minimizations over a single control:

$$V_i^{(1)}(x_i) = \underset{u_i}{\text{minimize}} \overbrace{\phi_i^{(1)}(x_i, u_i) + V_{i+1}^{(1)}(f(x_i, u_i))}^{\mathcal{V}_i^{(1)}(x_i, u_i)} \tag{3.11}$$
$$\hat{V}_i^{(1)}(x_i) = \underset{\hat{u}_i}{\text{minimize}} \underbrace{\hat{\phi}_i^{(1)}(x_i, \hat{u}_i) + \hat{V}_{i+1}^{(1)}(f(x_i, \hat{u}_i))}_{\hat{\mathcal{V}}_i^{(1)}(x_i, \hat{u}_i)} \tag{3.12}$$

We call $u_i^{(1)}(x_i)$ and $\hat{u}_i^{(1)}(x_i)$ the optimal control laws resulting from this minimization.

3.4.3 Introducing the Hierarchy

For a task $l > 1$, we define the unregularized value function as the minimum cost-to-go, subject to the constraint of not affecting the cost-to-go of the higher-priority tasks:

$$V_i^{(l)}(x_i) \triangleq \underset{U_i}{\text{minimize}}\, c_i^{(l)}(x_i, U_i)$$
$$\text{subject to } c_i^{(j)}(x_i, U_i) = c_i^{(j)}(x_i, U_i^{(j)}) \quad \forall j < l$$

Again, applying Bellman's principle of optimality we can reformulate this optimization as:

$$\begin{aligned} V_i^{(l)}(x_i) = &\underset{u_i}{\text{minimize}}\ \mathcal{V}_i^{(l)}(x_i, u_i) \\ &\text{subject to}\ \mathcal{V}_i^{(j)}(x_i, u_i) = V_i^{(j)}(x_i) \quad \forall j < l \end{aligned} \tag{3.13}$$

The regularized value function is instead the minimum regularized cost-to-go, subject to the constraint of not affecting the cost-to-go of the higher-priority tasks:

$$\begin{aligned} \hat{V}_i^{(l)}(x_i) \triangleq &\underset{\hat{U}_i}{\text{minimize}}\ \hat{c}_i^{(l)}(x_i, \hat{U}_i) \\ &\text{subject to}\ c_i^{(j)}(x_i, \hat{U}_i) = c_i^{(j)}(\hat{x}_i^{(j)}, \hat{U}_i^{(j)}) \quad \forall j < l \end{aligned} \tag{3.14}$$

Note the difference w.r.t. the constraints of the unregularized value function: here the desired cost-to-go is a function of the optimal state $\hat{x}_i^{(j)}$ rather than the current state x_i. This is due to the fact that $U_i^{(j)}$ is the minimizer of $c_i^{(j)}$, whereas $\hat{U}_i^{(j)}$ is not. In particular, using x_i instead of $\hat{x}_i^{(j)}$ in (3.14) would prevent the secondary tasks from modifying the state sequence found by the first task, that is $\hat{X}^{(1)}$. This is because the control law $\hat{U}^{(1)}$ minimizes the regularized cost, so applying it from a state that does not belong to $\hat{X}^{(1)}$ would result in a different value of the unregularized cost $c_i^{(1)}$. This difference prevents us to directly apply the same reduction as in (3.13).

3.4.4 Reformulation of the Regularized Problem

In order to apply Bellman's principle to reformulate (3.14) as a problem of a single control input, we need to introduce another control law. This control law tries to maintain the unregularized cost-to-go at the same value given by the regularized control law:

$$\tilde{U}_i^{(l)}(x_i) \triangleq \text{find}\, U_i \quad \text{s.t.} \quad c_i^{(l)}(x_i, U_i) = c_i^{(l)}(\hat{x}_i^{(l)}, \hat{U}_i^{(l)})$$

Intuitively, $\tilde{U}^{(l)}$ should behave like the regularized control law in feedforward, but like the unregularized control law in feedback. As long as the state sequence follows $\hat{X}^{(l)}$ we have $\tilde{U}^{(l)} = \hat{U}^{(l)}$. However, if the state sequence is modified by other tasks, $\tilde{U}^{(l)}$ uses the feedback action to maintain the same value of the unregularized cost-to-go. By definition, this new control law allows us to reformulate (3.14) as:

$$\begin{aligned} \hat{V}_i^{(l)}(x_i) \triangleq &\underset{\hat{U}_i}{\text{minimize}}\ \hat{c}_i^{(l)}(x_i, \hat{U}_i) \\ &\text{subject to}\ c_i^{(j)}(x_i, \hat{U}_i) = c_i^{(j)}(x_i^{(j)}, \tilde{U}_i^{(j)}) \quad \forall j < l \end{aligned}$$

Now we can apply Bellman's principle of optimality to reformulate the minimization of the cost function over the whole control sequence as a cascade of minimizations over a single control input. However, we can not do the same for the priority constraints: since $\tilde{U}_i^{(j)}$ is not the minimizer of $c_i^{(j)}$ the principle of optimality does

not apply. This prevents us from directly reformulating the priority constraints as functions of a single control input.

3.4.5 Final HDP Formulation

We propose then to replace the priority constraints with stricter constraints that are functions of a single control input so as to benefit from the computational efficiency of dynamic programming. The new constraints state that the cost-to-go of the higher-priority tasks at each time step must stay the same—which is sufficient but not necessary to guarantee that the total cost stay the same:

$$\begin{aligned} \hat{V}_i^{(l)}(x_i) \triangleq & \underset{\hat{u}_i}{\text{minimize}}\ \hat{\mathcal{V}}_i^{(l)}(x_i, \hat{u}_i) \\ & \text{subject to}\ \tilde{\mathcal{V}}_i^{(j)}(x_i, \hat{u}_i) = \tilde{V}_i^{(j)}(x_i) \quad \forall j < l, \end{aligned} \tag{3.15}$$

where $\tilde{V}_i^{(j)}(x_i)$ is the value function associated to the new control law (i.e. $\tilde{V}_i^{(j)}(x_i) = c_i^{(j)}(x_i^{(j)}, \tilde{U}_i^{(j)})$), and:

$$\tilde{\mathcal{V}}_i^{(l)}(x_i, u_i) = \phi_i^{(l)}(x_i, u_i) + \tilde{V}_{i+1}^{(l)}(f(x_i, u_i))$$

Solving this sequence of optimization problems for $l = 1, \ldots, n_l$ and for $i = N-1, \ldots, 0$, initialized with $\hat{V}_N^{(l)}(x_N) := \phi_N^{(l)}(x_N)$, we could solve (3.10).

3.5 Hierarchical Differential Dynamic Programming

The previous section derived the optimality conditions along samples of the trajectory, resulting in several difficult (typically nonconvex) optimization problems. We devised a discretized version of the conditions to improve the chapter clarity, however they could be quite directly extended to the continuous case as differential conditions. This section proposes a complete algorithm to compute the solution satisfying conditions (3.15). Direct resolution is not tractable in general. We rather follow the route proposed initially by the DDP approach [Jacobson 1970]. We build at each time step a quadratic approximation of the condition, which we can then solve using the proposed PHQP algorithm. The presentation of our method is self contained, but is best understood if the reader has in mind the classic DDP formulation. A concise and modern presentation of it can be found e.g. in [Tassa 2012].

3.5.1 Quadratic Differential Approximation

We start by considering a nominal control sequence $\bar{U} \triangleq (\bar{u}_0, \cdots, \bar{u}_{N-1})$ and the corresponding state sequence $\bar{X} \triangleq (\bar{x}_1, \cdots, \bar{x}_N)$ resulting by applying the former control to the system (3.8). We introduce the new variables of our optimization problem, which are the variation of control and state with respect to their nominal values: $\delta u_i \triangleq u_i - \bar{u}_i$ and $\delta x_i \triangleq x_i - \bar{x}_i$. We can now approximate $\mathcal{V}_i^{(l)}$, $\hat{\mathcal{V}}_i^{(l)}$ and $\tilde{\mathcal{V}}_i^{(l)}$ with their second-order Taylor expansions and optimize the resulting quadratic functions

with the PHQP algorithm. Let us define the local least-squares approximation of $\hat{\mathcal{V}}_i^{(l)}$:

$$\hat{\mathcal{V}}_i^{(l)}(x_i, u_i) \approx \hat{\mathcal{V}}_i^{(l)}(\bar{x}_i, \bar{u}_i) + \frac{1}{2} ||\hat{A}_{x,i}^{(l)\top} \delta x_i + \hat{A}_{u,i}^{(l)\top} \delta u_i + \hat{a}_i||^2$$

In the following, for coherence with our previous works [Romano 2015, Del Prete 2014] and the standard DDP algorithm [Tassa 2012], we prefer to represent the local approximation of $\hat{\mathcal{V}}_i^{(l)}$ with a quadratic form[2]:

$$\begin{aligned} \hat{\mathcal{V}}_i^{(l)}(x_i, u_i) \approx \hat{\mathcal{V}}_i^{(l)}(\bar{x}_i, \bar{u}_i) + \begin{bmatrix} \hat{Q}_{x,i}^{(l)\top} & \hat{Q}_{u,i}^{(l)\top} \end{bmatrix} \begin{bmatrix} \delta x_i \\ \delta u_i \end{bmatrix} + \\ \frac{1}{2} \begin{bmatrix} \delta x_i^\top & \delta u_i^\top \end{bmatrix} \begin{bmatrix} \hat{Q}_{xx,i}^{(l)} & \hat{Q}_{xu,i}^{(l)} \\ \hat{Q}_{ux,i}^{(l)} & \hat{Q}_{uu,i}^{(l)} \end{bmatrix} \begin{bmatrix} \delta x_i \\ \delta u_i \end{bmatrix} \end{aligned} \tag{3.16}$$

The coefficients of the quadratic approximation of $\hat{\mathcal{V}}_i^{(l)}$ can be recursively computed as:

$$\begin{aligned} \hat{Q}_{x,i}^{(l)} &\triangleq \partial_x \phi_i^{(l)} + \partial_x f^\top \partial_x \hat{V}_{i+1}^{(l)} \\ \hat{Q}_{u,i}^{(l)} &\triangleq \partial_u \phi_i^{(l)} + \partial_u f^\top \partial_x \hat{V}_{i+1}^{(l)} + \lambda^{(l)} \bar{u}_i \\ \hat{Q}_{xx,i}^{(l)} &\triangleq \partial_{xx} \phi_i^{(l)} + \partial_x f^\top \partial_{xx} \hat{V}_{i+1}^{(l)} \partial_x f + \partial_x \hat{V}_{i+1}^{(l)} \partial_{xx} f \\ \hat{Q}_{uu,i}^{(l)} &\triangleq \partial_{uu} \phi_i^{(l)} + \partial_u f^\top \partial_{xx} \hat{V}_{i+1}^{(l)} \partial_u f + \partial_x \hat{V}_{i+1}^{(l)} \partial_{uu} f + \lambda^{(l)} I \\ \hat{Q}_{xu,i}^{(l)} &\triangleq \partial_{xu} \phi_i^{(l)} + \partial_x f^\top \partial_{xx} \hat{V}_{i+1}^{(l)} \partial_u f + \partial_x \hat{V}_{i+1}^{(l)} \partial_{xu} f \end{aligned} \tag{3.17}$$

The local quadratic approximation of $\mathcal{V}_i^{(l)}$ and $\tilde{\mathcal{V}}_i^{(l)}$ are defined similarly, removing the symbol $\hat{}$ or replacing it with the symbol $\tilde{}$ (respectively) and setting $\lambda^{(l)}$ to zero. All the derivatives in (3.17) are computed for $x_i = \bar{x}_i$ and $u_i = \bar{u}_i$.

3.5.2 Backward Pass

The computation of (3.17) is initialized with $\hat{V}_N^{(l)}(x_N) = \phi_N^{(l)}(x_N)$. Then we can minimize our quadratic model of $\hat{\mathcal{V}}_i^{(l)}$ using PHQP, which gives us the locally-optimal feedforward and feedback terms for task l at time i (starting from $i = N-1$). Finally, to compute the solution for time step $i-1$ we need to compute $\hat{V}_i^{(l)}(\delta x_i)$ to update our quadratic approximation of $\hat{\mathcal{V}}_{i-1}^{(l)}$. We can do it by substituting the locally-optimal control $\delta \hat{u}_i^{(l)} = \hat{k}_i^{(l)} - \hat{K}_i^{(l)} \delta x_i$ into (3.16), which gives us:

$$\hat{V}_i^{(l)}(\delta x_i) \approx \hat{V}_{s,i}^{(l)} + \hat{V}_{x,i}^{(l)\top} \delta x_i + \frac{1}{2} \delta x_i^\top \hat{V}_{xx,i}^{(l)} \delta x_i \ ,$$

[2] We derive the algorithm without exploiting the least-squares structure, so that all the developments are still valid for a general quadratic approximation. The only exception is the hybrid control law (see Appendix), which we derive only for the least-squares case.

where[3]:

$$\begin{aligned}\hat{V}_{x,i}^{(l)} &= \hat{Q}_{x,i}^{(l)} - \hat{K}_i^{(l)\top}\hat{Q}_{u,i}^{(l)} - \hat{K}_i^{(l)\top}\hat{Q}_{uu,i}^{(l)}\hat{k}_i^{(l)} + \hat{Q}_{xu,i}^{(l)}\hat{k}_i^{(l)} \\ \hat{V}_{xx,i}^{(l)} &= \hat{Q}_{xx,i}^{(l)} + \hat{K}_i^{(l)\top}\hat{Q}_{uu,i}^{(l)}\hat{K}_i^{(l)} - \hat{Q}_{xu,i}^{(l)}\hat{K}_i^{(l)} - \hat{K}_i^{(l)\top}\hat{Q}_{ux,i}^{(l)}\end{aligned} \tag{3.18}$$

For the unregularized value function $V_i^{(l)}$ we have exactly the same equations, but using the unregularized control law rather than the regularized one. For the hybrid value function $\tilde{V}_i^{(l)}$ we need to use the following control law (see proof in the Appendix):

$$\delta\tilde{u}_i^{(l)} = \hat{k}_i^{(l)} - \hat{K}_i^{(l)}\delta\hat{x}_i^{(l)} - K_i^{(l)}(\delta x_i - \delta\hat{x}_i^{(l)}) \tag{3.19}$$

The feedback term of this control law is the same as for the unregularized control law, whereas its feedforward term is:

$$\tilde{k}_i^{(l)} = \hat{k}_i^{(l)} + (K_i^{(l)} - \hat{K}_i^{(l)})\delta\hat{x}_i^{(l)}$$

In our tests, to speed-up the algorithm, we neglected the terms depending on the second-order derivatives of the dynamics. This is the same approximation that distinguishes the iterative LQR algorithm [Todorov 2005, Tassa 2012] from DDP [Mayne 1966]. This allows us also to avoid the computation of some terms, since we have: $\tilde{V}_{xx,i}^{(l)} = V_{xx,i}^{(l)}$, $\tilde{Q}_{uu,i}^{(l)} = Q_{uu,i}^{(l)}$, $\tilde{Q}_{ux,i}^{(l)} = Q_{ux,i}^{(l)}$, $\tilde{Q}_{xx,i}^{(l)} = Q_{xx,i}^{(l)}$.

3.5.3 Regularizing the Optimization

So far we extensively discussed the issue of regularizing the cost functions describing the robot tasks, which we refer to as *task* regularization. The main goal of this regularization is to prevent the use of large control inputs. However, *algorithm* regularization remains necessary in general for a number of reasons that we describe now. The backward pass operates on a local approximation of the original problem (3.10). This model is only valid in the neighborhood of the current solution, so we must avoid taking too large steps. The regularization has a damping effect on near-singular directions, which tends to limit unreasonable step lengths. Moreover the Hessian Q_{uu} may become nonpositive, because the initial problem is nonconvex or because of propagation of numerical errors along the backward pass. Algorithm regularization ensures the soundness of the inversion of the Hessian.

We can achieve all of these goals by adding to all $\hat{Q}_{uu,i}^{(l)}$'s the identity matrix scaled by a sufficiently large scalar parameter $\mu^{(l)}$. Initial values for $\mu^{(l)}$ are given by the user (and may affect a lot the speed of convergence, as shown in our tests). At each iteration the algorithm decides whether to increase or decrease the current value of $\mu^{(l)}$ depending on the eigenvalues of $\hat{Q}_{uu}^{(l)}$ (after projection in the null space of previous tasks, see Algorithm 3) and the line-search parameters (more details on this in Section 3.5.5). The null-space projectors must be computed with the unregularized pseudo-inverses to ensure the proper hierarchy propagation.

[3]We do not report here the value of $\hat{V}_{s,i}^{(l)}$ because it does not affect the computation.

3.5.4 Order of the Operations

Now that we have defined the backward propagation of the value functions, we need to decide how to use them. In particular, we have two options:

1. Solve task l (starting from $l=1$) for $i=N-1,\ldots,0$ and then move on to task $l+1$.

2. Solve all tasks for time step i (starting from $i=N-1$) and then move on to time step $i-1$.

We decided to use the first option. To understand the reason behind this choice we need to look at what happens after the backward pass, that is the forward pass. During the forward pass we simulate the system forward in time using the locally-optimal control policy; if needed, we gradually reduce the magnitude of the feedforward control term until the associated cost does not improve. This is equivalent to the line-search procedure used in Sequential Quadratic Programming for nonlinear optimization [Wright 1999]. During the backward pass of task $l+1$ we assume that the feedforward term of task l will be completely applied, while this may not be the case because of the potential reduction occurring in the forward pass. It could be then beneficial to perform the forward pass of task l before the backward pass of task $l+1$, so that we could account for this reduction. This is only possible if we select the first one of the two options mentioned above.

Following this choice, Algorithm 3 summarizes the backward pass. Its inputs are: the derivatives of the dynamics and the cost function $\partial_{\cdot} f, \{\partial_{\cdot} \phi_i\}_i$; the null-space projector, feedforward and feedback terms of the previous tasks $\{N_i, \bar{k}_i, \bar{K}_i\}_i$; the task regularization parameter λ; the nominal control sequence $\bar{U}$ and the Hessian regularization parameter μ (see Section 3.5.3). The central part of the algorithm (i.e. lines 6, 9, 10, 14, 15, 16) is identical to the PHQP algorithm presented in Section 3.3.

3.5.5 Forward Pass (Line Search)

As usual, the forward pass consists in a forward simulation of the system using the locally-optimal control policy computed in the backward pass. It is used as a line search, i.e. at the end of the forward simulation we check whether the cost of all tasks decreases in a lexicographic order, i.e. a decrease of the cost of task l must not lead to an increase of the cost of any task $j<l$. In practice, we allow for a small increase of the higher-priority costs to speed-up convergence: a step is validated if it leads to a decrease of the current cost that is *much larger* than the increase of the higher-priority costs. A user-defined parameter defines how much larger this improvement should be (we used 10^3 for all our tests).

If this is not the case we reduce the magnitude of the feedforward term of the control policy and repeat again. This is the control policy used during the forward

Algorithm 3 Regularized HDDP: Backward Pass

function BACKPASS($\partial_{\cdot} f, \{\partial_{\cdot}\phi_i, N_i, \bar{k}_i, \bar{K}_i\}_i, \lambda, \bar{U}, \mu$)
2: Initialize $V^{(l)}_{x,N}$, $V^{(l)}_{xx,N}$, $\hat{V}^{(l)}_{x,N}$, $\hat{V}^{(l)}_{xx,N}$
for $i = N-1$ to 0 **do**
4: ▷ Compute regularized control law
$(\hat{Q}^{(l)}_{x,i}, \hat{Q}^{(l)}_{u,i}, \hat{Q}^{(l)}_{xx,i}, \hat{Q}^{(l)}_{uu,i}, \hat{Q}^{(l)}_{xu,i}) \leftarrow$ (3.17)
6: $\bar{Q}_{uu} \leftarrow N_i(\hat{Q}^{(l)}_{uu,i} + \mu I)N_i$
if $\min(\text{eigenvalues}(\bar{Q}_{uu,i})) < 0$ **then**
8: Increase μ and repeat from line 2
$\hat{k}_i \leftarrow -\bar{Q}^{\dagger}_{uu}(\hat{Q}^{(l)}_{u,i} + \hat{Q}^{(l)}_{uu,i}\bar{k}_i)$
10: $\hat{K}_i \leftarrow \bar{Q}^{\dagger}_{uu}(\hat{Q}^{(l)}_{ux,i} - \hat{Q}^{(l)}_{uu,i}\bar{K}_i)$
$(\hat{V}^{(l)}_{x,i}, \hat{V}^{(l)}_{xx,i}) \leftarrow$ (3.18) with $\bar{k}_i + \hat{k}_i$ and $\bar{K}_i + \hat{K}_i$
12: ▷ Compute unregularized control law
$(Q^{(l)}_{x,i}, Q^{(l)}_{u,i}, Q^{(l)}_{xx,i}, Q^{(l)}_{uu,i}, Q^{(l)}_{xu,i}) \leftarrow$ (3.17)
14: $\bar{Q}_{uu} \leftarrow N_i Q^{(l)}_{uu,i} N_i$
$k_i \leftarrow -\bar{Q}^{\dagger}_{uu}(Q^{(l)}_{u,i} + Q^{(l)}_{uu,i}\bar{k}_i)$
16: $K_i \leftarrow \bar{Q}^{\dagger}_{uu}(Q^{(l)}_{ux,i} - Q^{(l)}_{uu,i}\bar{K}_i)$
$(V^{(l)}_{x,i}, V^{(l)}_{xx,i}) \leftarrow$ (3.18) with $\bar{k}_i + k_i$ and $\bar{K}_i + K_i$
return $\{\hat{k}_i, \hat{K}_i, K_i, \mu\}_i$

pass of task j:

$$\begin{aligned}\delta\hat{u}^{(l)}_i =& \delta\hat{u}^{(l-1)}_i + \nu^{(l)}\hat{k}^{(l)}_i - \hat{K}^{(l)}_i(x_i - \bar{x}_i) \\ & - \bar{K}^{(l-1)}_i(x_i - \bar{x}_i - \delta\hat{x}^{(l-1)}_i),\end{aligned}$$

where $\delta\hat{u}^{(0)}_i = 0$, $\nu^{(l)}$ is the line-search parameter of task l and $\delta\hat{x}^{(l-1)}$ is the state deviation corresponding to the control deviation $\delta\hat{u}^{(l-1)}$. The term $\delta\hat{x}^{(l-1)}$ for the feedback of the higher-priority tasks comes from the hybrid control law (3.19). In this way, the unregularized feedback gains of all higher-priority tasks help not modifying the associated unregularized costs. Note that, by doing so, if $\hat{k}^{(l)}_i$ and $\hat{K}^{(l)}_i$ are void we get exactly $\delta\hat{u}^{(l)}_i = \delta\hat{u}^{(l-1)}_i$. We always initialize $\nu^{(l)} = 1$ and we decrease it with an exponential update rule:

$$\nu^{(l)} := a^l \nu^{(l)},$$

where $a \in [0, 1]$ is a parameter of the algorithm and l is the current iteration number of the line-search procedure. If the line-search does not converge after a predefined number of iterations, we increase the regularization parameters $\mu^{(l)}$ and repeat the backward pass. The pseudo-code to compute one complete step of RHDDP is reported in Algorithm 4.

Algorithm 4 Regularized Hierarchical Differential Dynamic Programming: 1 Step

1: **function** RHDDP($x_0, \bar{U}, f(.), \{\bar{U}^{(l)}, c^{(l)}(.), \lambda^{(l)}, \mu^{(l)}\}_l$)
2: $N \leftarrow I, \bar{k} \leftarrow 0, \bar{K} \leftarrow 0, \delta\hat{u} \leftarrow 0, \delta\hat{x} \leftarrow 0, \bar{U}^{(l)} \leftarrow \bar{U}$
3: $\bar{X} \leftarrow$ FORDWARDDYNAMICS$(x_0, \bar{U}, \bar{K})$
4: **for** $l = 1$ to n_l **do**
5: $(\{\hat{k}_i, \hat{K}_i, K_i\}_i, \mu^{(l)}) \leftarrow$ BACKPASS$(\partial f, \{\partial \phi_i^{(l)}, N_i, \bar{k}_i, \bar{K}_i\}_i, \lambda^{(l)}, \bar{U}^{(l)}, \mu^{(l)})$
6: $(\delta\hat{u}_i, \delta\hat{x}_i, \nu) \leftarrow$ LINESEARCH$(\{\delta\hat{u}_i, \hat{k}_i, \hat{K}_i, \bar{K}_i, \delta\hat{x}_i\}_i)$
7: **if** $\nu = 0$ **then**
8: Increase $\mu^{(l)}$ and go back to line 5
9: **if** $\nu < \nu^{thr}$ **then**
10: Increase $\mu^{(l)}$
11: **else if** $\nu = 1$ **then**
12: Decrease $\mu^{(l)}$
13: **for** $i = 0$ to $N - 1$ **do**
14: $\bar{k}_i \leftarrow \delta\hat{u}_i + K_i\delta\hat{x}_i$
15: $\bar{K}_i \leftarrow \bar{K}_i + K_i$
16: $\bar{U}_i^{(l)} \leftarrow \bar{U}_i^{(l)} + \delta\hat{u}_i$
17: Initialize $\tilde{V}_{x,N}^{(l)}$
18: **for** $i = N - 1$ to 0 **do**
19: $(\tilde{Q}_{x,i}^{(l)}, \tilde{Q}_{u,i}^{(l)}) \leftarrow$ (3.17)
20: $(\tilde{V}_{x,i}^{(l)}) \leftarrow$ (3.18) with $\bar{k}_i$ and $\bar{K}_i$
21: $N_i \leftarrow N_i \mathcal{N}((Q_{uu,i}^{(l)}, \tilde{Q}_{u,i}^{(l)\top}, Q_{xu,i}) N_i)$
22: $\bar{U} \leftarrow \bar{U} + \delta\hat{u}$
23: $\bar{X} \leftarrow \bar{X} + \delta\hat{x}$
24: **return** $(\bar{U}, \bar{X}, \{\bar{U}^{(l)}, \mu^{(l)}\}_k)$

3.5.6 Improving the algorithm

Because of the regularization, all tasks try to reduce the control inputs while not affecting the higher-priority tasks. This means that each task is likely to undo the action taken by the lower-priority tasks at the previous iteration of the algorithm (assuming that this action increased the control inputs). This effect should not disturb the convergence because in the backward pass each task can "see" what has been done by the higher-priority tasks, and compensate for it if that does not affect their unregularized costs. However, whenever the line search of a task converges to a small value of ν (e.g. < 0.1), its control action is drastically reduced, and so this compensation is not completely applied. In practice, the result of this phenomenon is that the low-priority tasks converge very slowly.

To avoid this effect, we modify the regularization term in the cost functions. Rather than minimizing the norm of all the control inputs, each task only minimizes the norm of the control inputs computed by itself and the higher-priority tasks. This is exactly the content of the variable $\bar{U}^{(l)}$ in Algorithm 4, which is updated in line 16.

3.5.7 Algorithm Summary

We now summarize the proposed algorithm. Each iteration is composed of the following phases:

1. Problem approximation.
2. Local control computation, or backward pass.
3. Line search, or forward pass.
4. Computation of the null space

Convergence is tested at the end of each iteration. The convergence criteria consists in an absolute criterion and a relative one. We assume that the algorithm has converged if the cost is lower than the absolute tolerance value. Alternatively, the relative improvement between two successive iterations must be smaller than a relative tolerance value.

3.6 Simulations

This section presents several simulations on different robotic systems to validate the proposed algorithm (RHDDP) and compare it to the classic DDP algorithm and to the previous version of HDDP [Romano 2015]. In all tests we use weights to approximate strict priorities with the DDP algorithm and we compare this approximation with our method. The main result is that obtaining a sound hierarchy behavior with optimal control is nearly impossible with weighted DDP, while it is straightforward with RHDDP. The accompanying video collects all the results

of the simulations presented in this section, plus an additional simulation on the humanoid robot HRP-2.

Section 3.6.1 presents simulations with the PR2 robot that stress the difficulty of tuning the cost weights with DDP. Moreover it analyzes the convergence of the RHDDP algorithm. Section 3.6.2 (always with PR2) shows that without a hierarchy we cannot properly regularize the tasks: either regularization is too large and takes over the low-priority task, or regularization is not enough and so commands are too large. In Section 3.6.3 we use a simple cart-pole system to highlight again—but in a different way—the benefits of the hierarchy when regularizing the tasks. Finally, Section 3.6.4 presents a more complex set of simulations with the UR5 robot, which is asked to execute a sequence of tasks distributed in time. RHDDP allows us to trade-off accuracy and efficiency by using the regularization without compromising the hierarchy (i.e. low-priority tasks deteriorate first when regularization increases). In this test we also show that RHDDP is capable of faster convergence with respect to the old version of the algorithm (HDDP [Romano 2015]).

In all tests we used dynamics models for the robots, which take motor torques as control inputs and the number of samples along the temporal horizon is arbitrarily set to $N = 100$.

The tests have been executed on a computer with two processors Intel Xeon CPU E5-2620V2 (6 cores 2.10GHz) and 64GB of RAM. The tested algorithms were coded in Matlab (v2013b) but the simulation of the systems is computed thanks to the C++ dynamic engine MuJoCo [Todorov 2012].

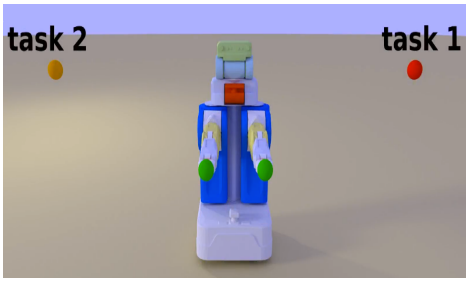
3.6.1 Test 1: PR2 - Final Cost

This simulation compared RHDDP and DDP in a grasping task with the PR2 robot (state dimension 36, control dimension 18). The goal was to perform the following four tasks (in order of priority):

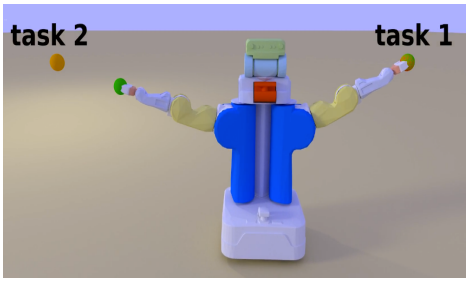
0. have zero joint velocity at final time $i = N$
1. grasp left (red) ball with left gripper (final cost only)
2. grasp right (orange) ball with right gripper (final cost only)
3. minimize 2-norm of control trajectory

In this test we did not need to use any regularization for the tasks because they used only a final cost—the robot was not asked to reach the goal as fast as possible. Results are summarized by Fig. 3.2 to 3.4 and Table 3.1.

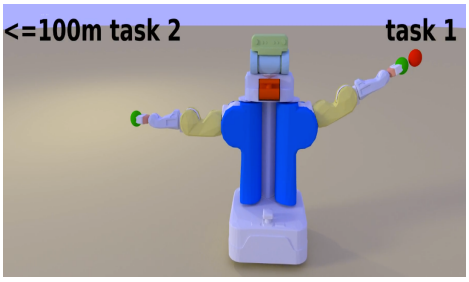
When using DDP we used the following weights to approximate strict priorities: $10^6, 10^3, 1, 10^{-6}$. First we set the two balls slightly too far away from each other to be reached at the same time (see Fig. 3.2a). In this scenario both RHDDP and DDP managed to reach the first ball and to minimize the distance to the second ball (see Fig. 3.2b). Then we moved the second ball 100 meters away from the robot. In this case DDP did not manage to reach the first ball (see Fig. 3.2c), while RHDDP



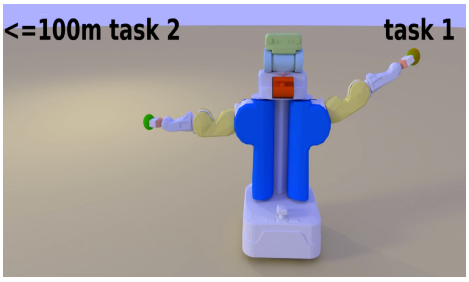
(a) Initial state.



(b) Final state with both algorithms.



(c) Final state with DDP and second ball 100 m away.



(d) Final state with RHDDP and second ball 100 m away.

Figure 3.2: Test 1: snapshots of the simulations when both balls are close to the robot (top row) and when one of the balls is far away (bottom row).

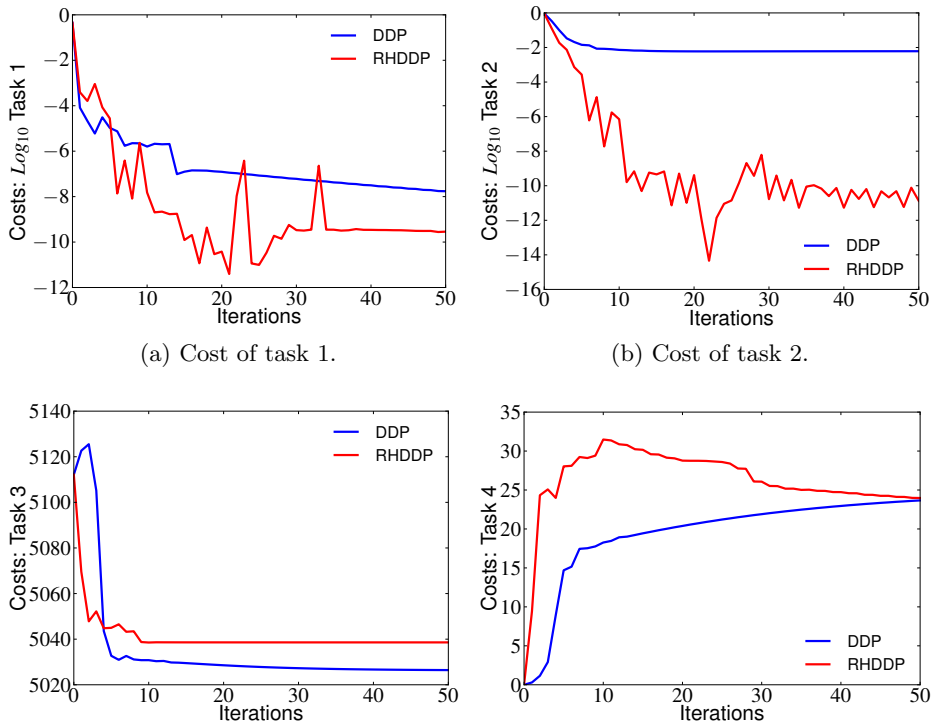
Table 3.1: Test 1: cost function values.

Algorithm	Scenario	Task 2	Task 3	Effort
DDP	Balls close	$1.23\ 10^{-7}$	$1.01\ 10^{-1}$	$3.51\ 10^{4}$
RHDDP	Balls close	$3.65\ 10^{-13}$	$1.12\ 10^{-1}$	$2.84\ 10^{4}$
DDP	Balls far	$6.03\ 10^{-3}$	$5.03\ 10^{3}$	$2.64\ 10^{4}$
RHDDP	Balls far	$2.39\ 10^{-10}$	$5.04\ 10^{3}$	$2.73\ 10^{4}$

did (see Fig. 3.2d). This clearly shows that weights are task dependent, and using strict priorities provides more robust behaviors. Table 3.1 shows the values of the cost functions with RHDDP and DDP.

3.6.1.1 Convergence of the Algorithm

Fig. 3.3 shows the value of the cost functions throughout the iterations of RHDDP and DDP. Fig. 3.4 instead shows the values taken by the regularization parameters $\mu^{(l)}$. With RHDDP the regularization quickly converged to its minimum value (10^{-8}) for the first two tasks, which allows a fast convergence to a good approximate solution. On the other hand, DDP convergence is delayed by the algorithm regularization, which never reaches its minimum value due to the artificial ill conditioning introduced by the weight scaling. Convergence of DDP is somehow sequential: low-priority tasks improve only after high-priority tasks have converged (e.g. see



(a) Cost of task 1.

(b) Cost of task 2.

(c) Cost of task 3.

(d) Cost of task 4.

Figure 3.3: Test 1: cost of the different tasks over the iterations of the algorithm when second ball is 100 m away from the robot.

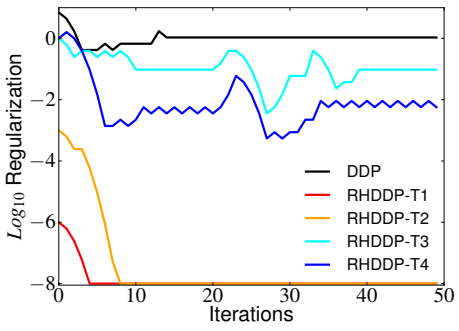
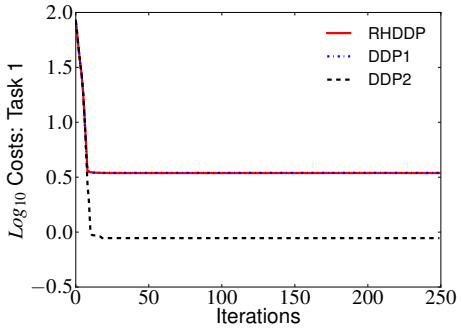


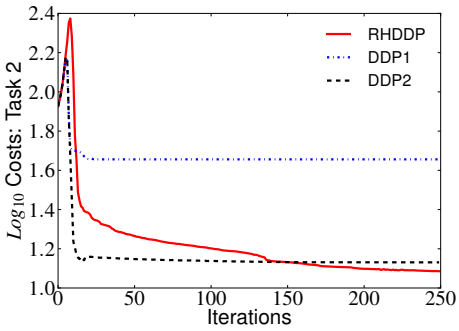
Figure 3.4: Test 1: value of the regularization parameters throughout the iterations of the RHDDP (one value for each task) and DDP algorithm.

Table 3.2: Test 2: cost function values.

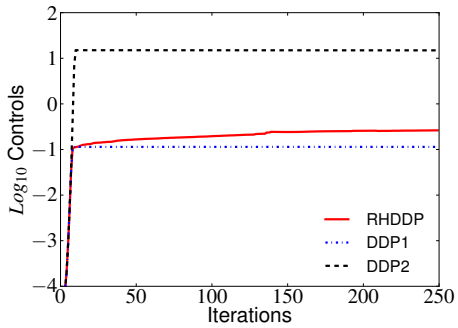
Algorithm	Regularization	Task 1	Task 2	Effort Max
DDP	10^{-5}	3.46	45.3	238
DDP	10^{-8}	0.881	13.5	3325
RHDDP	10^{-5}	3.45	12.2	319



(a) Cost of task 1.



(b) Cost of task 2.



(c) Cost of task 3.

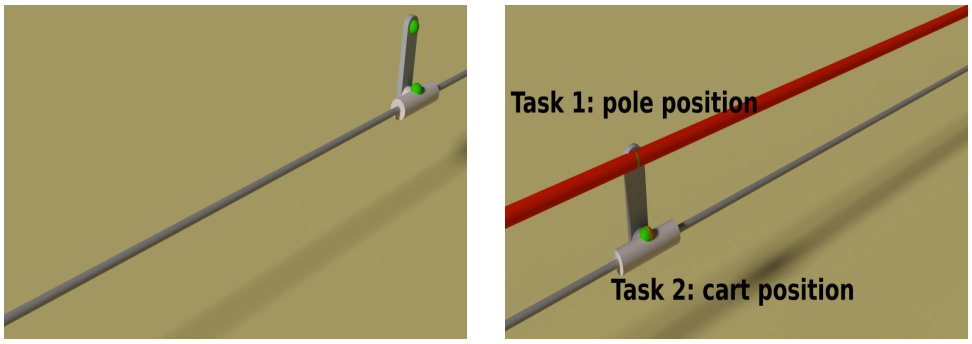
Figure 3.5: Test 2: cost of the different tasks over the iterations of the algorithm. In the legend, DDP1 is DDP with $w_r = 10^{-5}$, while DDP2 is DDP with $w_r = 10^{-8}$.

Fig. 3.3b and 3.3c). This effect slows down the convergence of the algorithm, which we study in more details in Test 4.

3.6.2 Test 2: PR2 - Integral Cost

This test is based on the same scenario of the previous test (with the second ball almost reachable). These are the tasks in order of priority:

1. grasp left (red) ball with left gripper (integral cost)
2. grasp right (orange) ball with right gripper (integral cost)
3. minimize 2-norm of control trajectory



(a) Initial state. (b) Goal state.

Figure 3.6: Test 3: initial and goal state of the cart-pole.

Since we are using integral costs, we need to regularize the tasks to avoid too large commands. For DDP we used $w_1 = 1$ (weight of the first task), $w_2 = 10^{-3}$ (weight of the second task) and we tried two different values of w_r (weight of the regularization, i.e. minimum-effort task). Results are summarized by Fig. 3.5 and Table 3.2. In brief, it was not possible to find a correct value of w_r for DDP, while it was immediate for RHDDP.

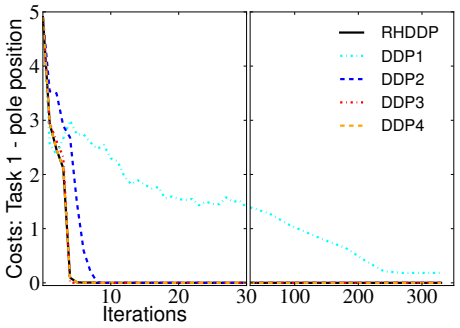
Using $w_r = 10^{-5}$ was enough to avoid too large control inputs, but resulted in a large cost for the second task. Using $w_r = 10^{-8}$ allowed us to improve the execution of the second task, but resulted in large control inputs. With RHDDP we used $w_r = 10^{-5}$ for the regularization—keeping in mind that regularization is here applied on each priority level—which was large enough to avoid large commands; at the same time, also the second task was executed at its best. This is thanks to the fact that RHDDP allows us to keep the same ratio between task and regularization weights, which is not the case for DDP. With DDP, when using $w_r = 10^{-5}$, the ratio between w_2 and w_r was only 10^{-2}, so the performance of the second task was negatively affected.

RHDDP and DDP with $w_r = 10^{-5}$ resulted in almost identical costs for task 1 and 3 (see Table 3.2), but RHDDP performed better at task 2. Increasing w_2 would improve task 2, but at the expense of task 1 and 3. Increasing w_1 and w_2 is equivalent to decreasing w_r and would worsen task 3. This suggests that RHDDP allowed us to get a behavior that we could not achieve by using DDP, no matter the values of w_1, w_2 and w_r.

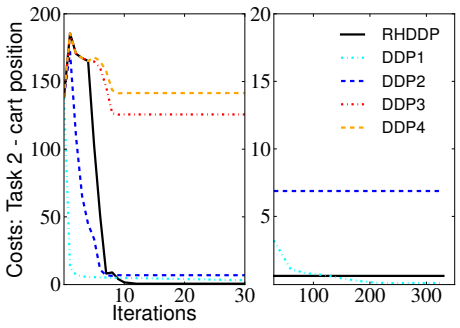
3.6.3 Test 3: Cart-Pole

In this simulation we tested RHDDP and DDP with a simple underactuated system: the cart-pole (state dimension 4, control dimension 1). In order of priority, the tasks to perform were:

1. keep the pole high (integral cost between $0.7N$ and N)
2. reach a goal position with the cart (integral cost between $0.7N$ and N)

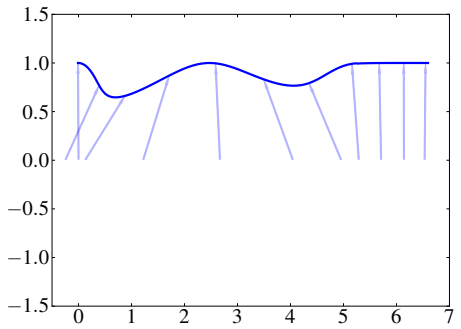


(a) Cost of task 1. After iteration 30 the scale of the x axis changes.

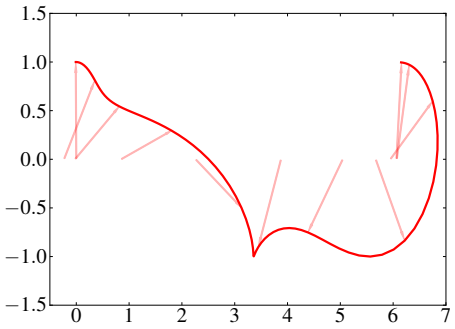


(b) Cost of task 2. After iteration 30 the scale of the x and y axes changes.

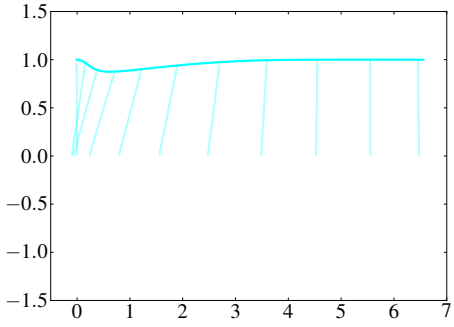
Figure 3.7: Test 3: value of the cost functions throughout the iterations of the algorithm. In the legends, DDP1 to DDP4 represent DDP with $w_2 = 1$ to 10^{-3}, respectively.



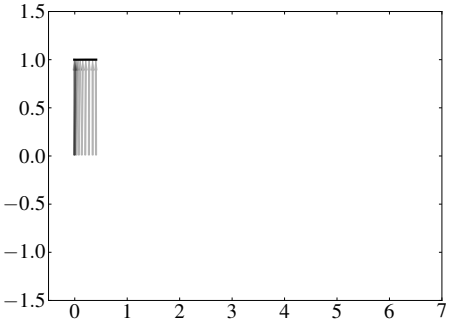
(a) RHDDP.



(b) DDP with $w_2 = 1$.



(c) DDP with $w_2 = 0.1$.



(d) DDP with $w_2 = 0.01$.

Figure 3.8: Test 3: trajectory of the cart-pole system obtained with the different algorithms. The cart start at position 0 and should move to position 6.

Table 3.3: Test 3: cost function values.

Algorithm	w_2	w_r	Task 1	Task 2	Effort
RHDDP	N.A.	0	$1\ 10^{-5}$	$5\ 10^{-5}$	$4.55\ 10^{2}$
RHDDP	N.A	10^{-2}	$3.65\ 10^{-4}$	$5.93\ 10^{-1}$	$1.60\ 10^{2}$
DDP	1	10^{-2}	$1.84\ 10^{-1}$	$6.66\ 10^{-2}$	$1.11\ 10^{2}$
DDP	10^{-1}	10^{-2}	$2.55\ 10^{-4}$	6.88	$1.60\ 10^{1}$
DDP	10^{-2}	10^{-2}	$3.77\ 10^{-6}$	$1.25\ 10^{2}$	$7.83\ 10^{-2}$
DDP	10^{-3}	10^{-2}	$1.25\ 10^{-7}$	$1.41\ 10^{2}$	$1.25\ 10^{-3}$

Table 3.4: Test 4: cost function values with the RHDDP algorithm.

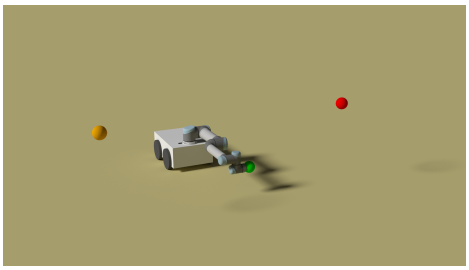
w_r	Task 1	Task 2	Task 3	Task 4	Task 5	Effort
$5\ 10^{-5}$	$1.4\ 10^{-8}$	$3.3\ 10^{-6}$	$4.4\ 10^{-6}$	$8.3\ 10^{-5}$	$1.1\ 10^{-4}$	$2.8\ 10^{-1}$
$5\ 10^{-4}$	$4.7\ 10^{-12}$	$7.5\ 10^{-7}$	$1.7\ 10^{-5}$	$1.0\ 10^{-2}$	$4.6\ 10^{-3}$	$3.6\ 10^{-1}$
$5\ 10^{-3}$	$2.2\ 10^{-9}$	$3.7\ 10^{-4}$	$4.6\ 10^{-3}$	$9.8\ 10^{-1}$	$2.8\ 10^{-1}$	$2.7\ 10^{-1}$
$5\ 10^{-2}$	$1.4\ 10^{-9}$	$6.5\ 10^{-2}$	$3.7\ 10^{-2}$	26.5	16.1	$8.0\ 10^{-2}$

The two tasks are compatible only if we allow the system to use large control inputs (see first line of Table 3.3). With DDP, we kept constant $w_1 = 1$ (weight of the first task) and $w_r = 10^{-2}$ (weight of the regularization), whereas we varied w_2 (weight of the second task). Using $w_2 = 10^{-1}$ or less, only task 1 was achieved (see corresponding lines in Table 3.3) because w_r was too large w.r.t. w_2. Using $w_2 = 1$ allowed the system to execute the second task, but it deteriorated the performance of the first task (see third line of Table 3.3). With RHDDP we used $w_r = 10^{-2}$ as well, but the hierarchy allowed for the execution of the second task while not deteriorating the first task (see second line of Table 3.3). Fig. 3.8 depicts some of the trajectories found by DDP and RHDDP.

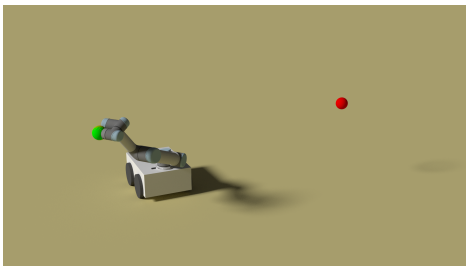
3.6.4 Test 4: UR5 - Sequential Tasks

In this test we used a mobile version of the UR5 robot (state dimension 16, control dimension 8) to reach two balls, one after the other. The two reaching tasks are compatible because they are executed one after the other. However, they become conflicting if we introduce regularization, which does not allow the system to use large control inputs. The first goal of this test is to show how, with RHDDP, increasing the regularization affects first the low-priority tasks. In order of priority, the tasks to perform were:

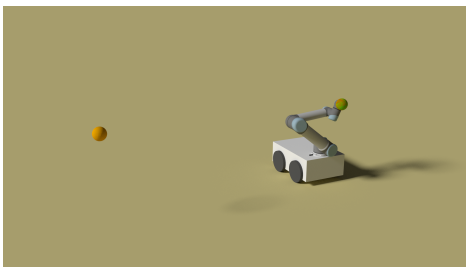
1. have zero velocity at time step N
2. reach second ball at time step N (see Fig. 3.9c)
3. have zero velocity for the gripper between $0.9\,N$ and N
4. reach first ball at time step $0.45\,N$ (see Fig. 3.9b)



(a) Initial state.



(b) Task 4: reach ball at time step $i = 0.45N$.



(c) Task 2: reach ball at time step $i = N$.

Figure 3.9: Test 4: snapshots of the simulation.

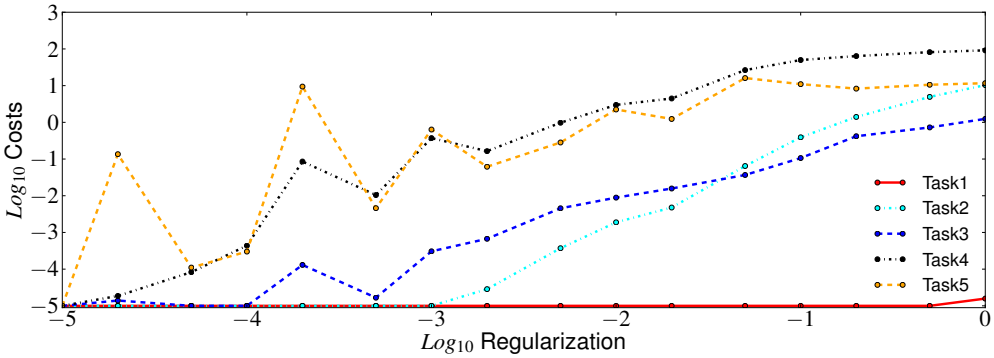
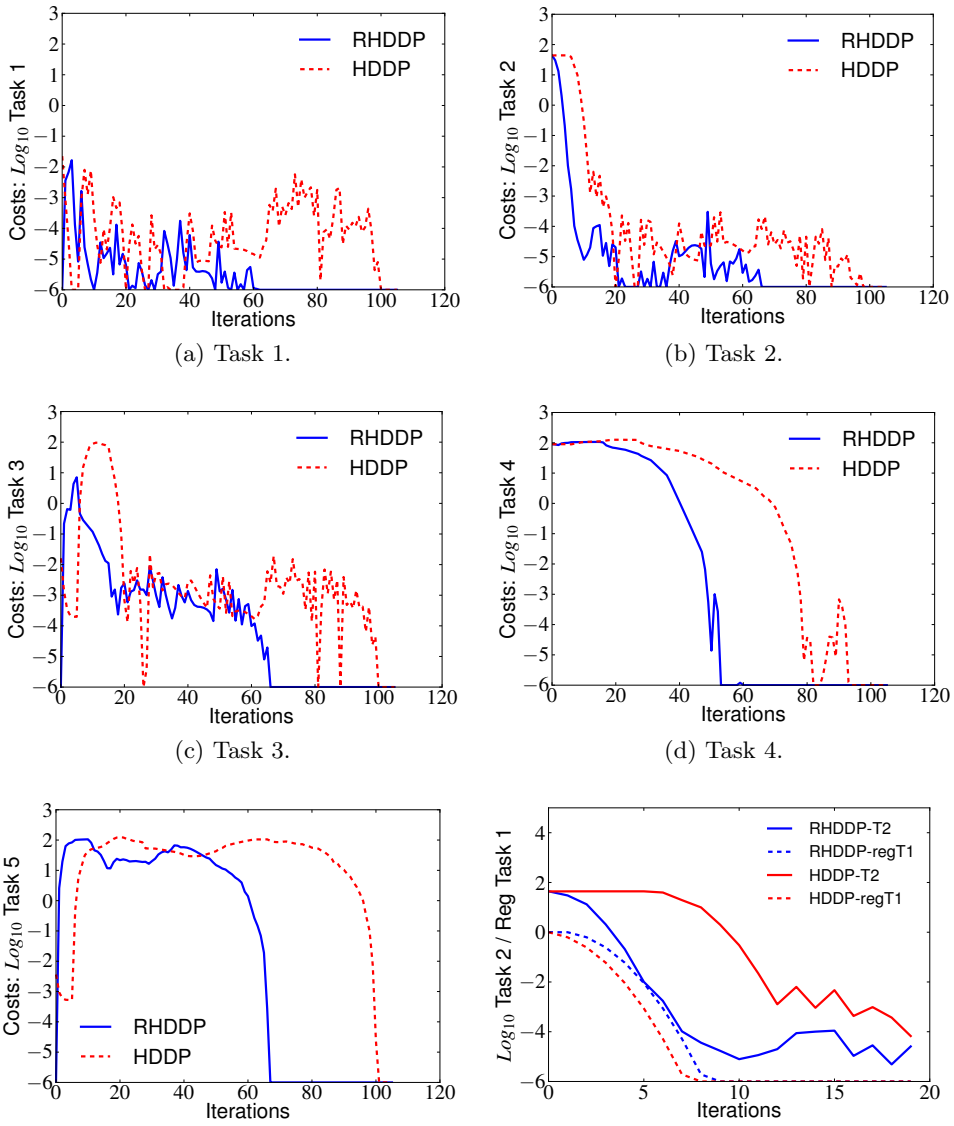


Figure 3.10: Test 4: values of the cost functions obtained by using different values of the regularization parameter w_r.



(a) Task 1.

(b) Task 2.

(c) Task 3.

(d) Task 4.

(e) Task 5.

(f) Cost of task 2 and algorithm regularization parameter of task 1.

Figure 3.11: Test 4: convergence of the cost functions using the algorithm presented in this chapter (RHDDP) and its previous version (HDDP [Romano 2015]).

5. have zero velocity for the gripper between $0.4\,N$ and $0.5\,N$

Table 3.4 and Fig. 3.10 summarize the results. We observed that w_r can be increased to force the system to use smaller control inputs, and it progressively deteriorates the low-priority tasks.

Table 3.5: Test 4: convergence of DDP/HDDP/RHDDP. Time is expressed in seconds.

μ_{init}	DDP0		DDP2		DDP3		HDDP		RHDDP	
	It.	Time	It.	Time	It.	Time	It.	Time	It.	Time
10^{-6}	7	2.8	9	3.8	65	28	245	299	157	197
10^{-4}	7	2.9	35	14.3	>500	>250	83	105	58	77
10^{-2}	8	3.4	259	116	>500	>250	98	130	58	77
10^{0}	12	5.3	393	179	>500	>250	100	129	63	82.3
10^{2}	14	5.8	>500	>250	>500	>250	113	141	89	111
10^{4}	16	6.7	>500	>250	>500	>250	116	146	72	90.8
10^{6}	21	9.0	>500	>250	>500	>250	118	148	70	88

3.6.4.1 Comparison with HDDP

We also solved this problem with the previous version of our algorithm (HDDP [Romano 2015]), but without using any task regularization because this was not allowed by HDDP. Fig. 3.11 shows that, thanks to the improvements proposed in this chapter, convergence is faster with RHDDP. Moreover, Fig. 3.11f shows that HDDP starts decreasing the cost of task 2 only after the algorithm regularization of task 1 has decreased. This is due to an incorrect handling of the algorithmic regularization in HDDP, which has been addressed in RHDDP. As a result, RHDDP is capable of decreasing the cost of task 2 while optimizing task 1.

3.6.4.2 Comparison with DDP

Finally, we solved the same problem also with DDP, trying three different sets of values for the task weights and different initial values of the algorithmic regularization parameter μ. Due to the lack of task regularization all the tasks are compatible, so the choice of the weights does not affect the final results, but it affects the convergence of the algorithm. Table 3.5 summarizes the results.

Let us define the weight distance Δw as the ratio between the weights of the neighbor tasks, e.g. w_1/w_2. When $\Delta w = 1$ (DDP0) the algorithm converged quickly regardless of the value of μ_{init}. With $\Delta w = 10^2$ (DDP2), the artificial ill-conditioning introduced by the weights slows down the convergence, especially for large values of μ_{init}. With $\Delta w = 10^3$ (DDP3) the ill-conditioning has an even stronger effect on the convergence, which is much slower for all values of μ_{init}. More in details, the reason why DDP2 and DDP3 converged faster for small values of μ_{init} lies in the line search. For instance, with DDP3 task 5 has a weight of 10^{-6}, so starting with $\mu_{init} = 10^{-2}$ the regularization has a much larger weight than task 5. The result is that, during the first steps, the other tasks converge to a local minimum (because compared to them, the regularization is not so large) and μ reduces until task 5 can be solved. At this point, the algorithm needs to reduce $c^{(5)}$ without worsening the other costs, e.g. for a step that reduces $c^{(5)}$ of 1, $c^{(1)}$ should increase no more than 10^{-12} (because task 1 has a weight of 10^6). This is almost

impossible because of the nonlinearity of the problem, so the algorithm can only take little steps to reduce the last tasks, which leads to a slow convergence. On the contrary, using a smaller value of μ_{init}, all tasks can converge during the first iterations. Since task 1 decreases, even if task 5 affects a bit task 1, it is tolerated as long as the weighted sum of the costs decreases.

RHDDP and HDDP seem more robust to variations of μ_{init}. The reason behind this is that the line search of (R)HDDP is not negatively affected by the scaling introduced by the weights. In our tests, (R)HDDP validates a step if it leads to an improvement of the current cost that is 10^3 times larger than the deterioration of the higher-priority costs (see Section 3.5.5). Moreover, as expected, RHDDP converges always in less iterations than HDDP.

Table 3.5 also shows that one iteration of DDP takes on average about 0.4 s, whereas one iteration of RHDDP takes about 1.3 s. This larger computation time is justified by the RHDDP algorithm, which has to perform a backward propagation and a line search for each task. DDP instead performs a single backward propagation and a single line search for all tasks at once.

3.7 Discussion

There are several benefits of the proposed approach with respect to non-hierarchical trajectory optimization. First, fixing priorities is usually much easier when setting the program than finding good weights. It also results in more robust behaviors (i.e. weights may need to be tuned again if the task is modified, as shown in Section 3.6.1). Second, RHDDP allows us to properly regularize the tasks: as shown in Section 3.6.2 and 3.6.3 we can arbitrarily increase regularization while still executing the secondary tasks and satisfying the priority constraints. Also compared to other hierarchical trajectory optimization methods [Tazaki 2014], our approach presents some benefits. In particular, it exploits the sparsity of the HOC problem and it properly handles the task regularization.

One limitation of our approach is that, in order to get a low computational cost for each iteration, we replaced some constraints with constraints that approximate the original one in a stricter way. This allows us to guarantee the satisfaction of the original constraints, at the expense of restricting the space of solutions explored by our algorithm. Let us illustrate this with a simple 1-dimensional dynamical system:

$$x_{i+1} = x_i + \delta t\, u_i$$

Suppose that the system starts at $x_0 = 0$ and its first task is to minimize the cost $(x_N - 5)^2$. If we regularize the task, the system will not reach exactly 5, but it will stop before, e.g. at $x_N = 4.7$. Now let us introduce a second task, of lower priority, which consists in minimizing the cost $(x_{N/2} - 10)^2$. Again, due to the task regularization, the system will not reach $x = 10$, but will stop before, e.g. at $x_{N/2} = 9$. Starting from $x_{N/2} = 9$, the best thing to do to preserve the same cost of task 1 (i.e. $(4.7 - 5)^2$) would be to reach $x_N = 5.3$. However, RHDDP is not able

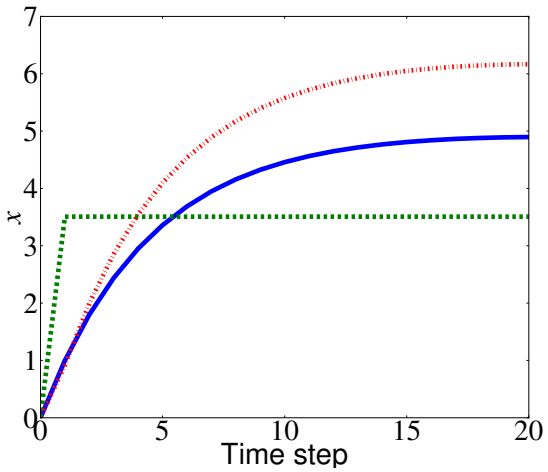


Figure 3.12: Example of trajectory redundancy: these three state sequences result in the same value of the cost function: $\sum_{i=0}^{N}(x_i - 5)^2$.

to do that, and will still reach $x_N = 4.7$, thus using larger control inputs to obtain the same cost. This limitation is due to the relaxation of the priority constraints that we proposed in Section 3.2.3. One way to overcome this issue would be to use inequalities to express the priority constraints as:

$$g^{(j)}(y) \leq g^{(j)}(\hat{y}^{(j)})$$

This would also preserve the convexity of the regularized HQP problem, but it would increase the computational cost of the algorithm due to the need of handling the inequalities (e.g. by using an interior-point method [Wright 1999]).

Nonetheless, our relaxation of the priority constraints leads the method to scale well with the number of tasks. In other words, regardless of the number of low-priority tasks, RHDDP would still achieve the first task similarly well. This is because we use a stricter version of the priority constraints, which is a sufficient (but not necessary) condition for the original constraints.

Let us now consider another example using the same simple dynamical system. Suppose that the first task consists in minimizing $\sum_{i=0}^{N}(x_i - 5)^2$. Given that the task is regularized, the resulting state sequence is depicted by the blue continuous line in Fig. 3.12, which results in a cost of task 1 of about 34.8. Clearly, there exist infinitely many other state sequences that result in the same cost of task 1, e.g. the other two lines in Fig. 3.12. However, RHDDP cannot exploit this redundancy to achieve secondary tasks because of the relaxation introduced in Section 3.4.2. Rather than constraining the total cost of the higher-priority tasks, RHDDP constrains their cost-to-go at each time step, which removes part of the redundancy in case of integral costs. To overcome this issue we should abandon dynamic programming and study another algorithm to exploit the sparse structure of the HOC problem. Even if the computational complexity may be similar to RHDDP, the resulting algorithm

would be harder to implement. DDP also directly provides the feedback gains for free, which makes closed-loop trajectory execution possible. This advantage would be lost with another approach.

3.8 Conclusions

This chapter presented a novel algorithm for motion generation of nonlinear dynamical systems. The motion is generated by performing a hierarchical minimization of a number of functions of the system state and control trajectories. Each function describes a task that the robot should achieve, and its position in the hierarchy defines its priority level w.r.t. the other tasks. This concept of *strict* priorities, as opposed to the *soft* priorities obtained by using a weighted sum, allows for an easier and more robust specification of the motion objectives. In particular, no weight tuning is necessary and numerical ill conditioning is avoided. Moreover, the proposed algorithm properly handles the issue of regularizing the tasks. Our tests clearly show the benefit of the hierarchy, which allows us to properly regularize the problem while preserving the strict priorities. This was not possible with soft priorities.

Another contribution of this chapter deals with the regularization of a hierarchy of quadratic functions (HQP), the optimization problem that is at the core of our algorithm. Even if the use of regularization is ubiquitous in robotics optimization problems, we are the first to define the regularized HQP problem and to show that it is nonconvex. We then proposed a convex relaxation of this nonconvex problem, which allows us to get an approximate solution in a single iteration. We also showed the importance of both the hierarchy and the regularization to generate feasible behavior with autonomous robots in simulation.

In most cases, MPC boils down to minimum time trajectories (through a integral cost) or trajectory tracking. In both cases, the cost function is formulated as an integral cost and regularization needs to be added to ensure smooth and limited controls. Thus, regularization is a key part for MPC. The special care taken here on regularization would allows to directly use RHDDP as MPC controller. With a proper implementation of the algorithm, we could consider a real application on HRP-2 as in [Koenemann 2015]. Even if the computation per time step (of the preview windows) is increased compare to a soft hierarchy, we might hope that the hierarchical constraints stabilize the system and therefore allow to reduce the preview window (i.e. the number of time steps) and the total computation time. Moreover, if we proceed to some adjustments (increasing the algorithmic regularization to avoid a linesearch), each task can easily be parallelized with a shift of one time step thus reducing the computation time of the algorithm from $o(NK)$ to $o(N+K)$.

An important question that still remains is warm-starting. While IREPA gave good results to initialize multiple shooting, it can be more complex for a single shooting algorithm as RHDDP. Dynamic consistency is enforced by the single shooting

scheme and therefore the results should be closer to a roll-out of the policy.

Another question is the pertinence of hierarchies when the algorithm is inserted in a cascade of controllers. While the hierarchy simplifies the construction of the cost function, it forces to have properly defined tasks. The hierarchy will make the robot follow the given task even if this will make it fall. Therefore, we need a strong confidence on the tasks given to the algorithm, e.g. on the trajectories given by the higher-level controller in a cascade. To construct proper trajectories with higher-level controller, we need to take into account the dynamic capabilities of the robot although the model used must stay simple. To do that, we need to construct methods that allow to reveal feasibility of the solution. In the next chapter, we will develop a method to construct a feasibility estimator allowing to plan a path for a biped robot on uneven environment without explicitly computing contacts.

Chapter 4

Pose Learning

In the two previous chapters, we have shown that the maturity of direct optimal control makes it an indisputable methodology for controlling a robot with an efficient and versatile manner. This is suitable for robots whose dynamics are neither too complex nor too fast. However, for legged robots, optimal control cannot (yet) be applied in a standalone manner. We will need a planner to guide the optimal control solver with an initial solution candidate. While we have shown in Chapter 2 a preliminary method to generate this warm start from data science, it is not yet mature for legged robots. In this chapter, we rather focus on the team's approach that we are collaboratively developing in *Gepetto*, based on the initial work of Tonneau [Tonneau 2015] and that we refer as *Loco3d*. As explained in Chapter 1, this approach is to divide the locomotion problem into a cascade of simpler problems, whose iterative resolution leads to a proper whole-body initial guess to warm start the optimal control solver. A bottleneck of this approach is the generation of "proxy" constraints, i.e. constraints to be handled at any level N of the cascade to make sure the level $N+1$ is feasible. In this chapter, we explore a generic method to approximate such a proxy, with an approach based on offline data generation and machine learning.

4.1 Introduction

In this chapter, we are considering a bipedal robot going through an uneven environment made from multiple plane surfaces (Fig. 4.1). This problem is non-convex and high-dimensional, and it is unlikely that a solution to the full problem can be found quickly. As explained in the first chapter, we believe that an efficient approach is to use a cascade of planners to find the solution. Intermediate plans are computed with low-dimensional models then use as initial guesses or guide trajectories for the lower-level planners.

4.1.1 Feasibility conditions

Now, each level of the cascade should take into account the feasibility constraints of the next levels. This is made difficult because each level typically works with a subset of the robot whole state and thus cannot directly express constraints of the other part of the robot state. However, it is important to check the feasibility (or at least a simple approximation of it), otherwise infeasible plans due to lack of constraints in the early cascade levels result in failures in the later levels, hence in

costly iterations in the cascade. In the current state of Loco3d, the first controller plans the path of the robot using the reachability and non-collision conditions already introduced in section 1.5.1.2. However, these criteria do not reveal feasibility with respect to the robot stability. In this chapter, we focus on the feasibility condition used in the highest controller of our cascade (see Section 1.5.1.1). This planner computes a guide trajectory for the main body of the robot. As an explicit computation of contacts is too costly at this stage, we rather generate the guide trajectory using a feasibility condition. The feasibility condition initially implemented in the cascade is based on the intuition of the human behind the planner: we name it the *Reachability Condition* [Tonneau 2015]. As explained in Section 1.5.1.2, the reachability condition is a constraint based on two volumes: one represents the minimum space the robot needs to avoid collisions (Fig. 4.2, red shapes), the other one represents the reachable spaces for each end-effector i.e. spaces where the end-effectors can create contacts with the environment (Fig. 4.2, green shapes). A main body placement is valid for the reachability condition if the environment is not colliding with the "minimum space" but colliding with the reachable spaces of at least N end-effectors (where N is predefined by the user). This approach is computationally efficient because it allows us to have a feasibility condition that can be verified with only few collision checkings. While efficient, this criterion also leads to infeasible trajectories and must be used jointly with ad hoc constraints in practice (e.g. at least one leg must be in contact). The objective of the work presented in this chapter is to establish an automatic method to design a similar condition based on offline sampling of the robot motion capabilities in random environments. We expect that this would make the approach more generic, less based on intuition but also more accurate as it would capture more adequately the robot capabilities.

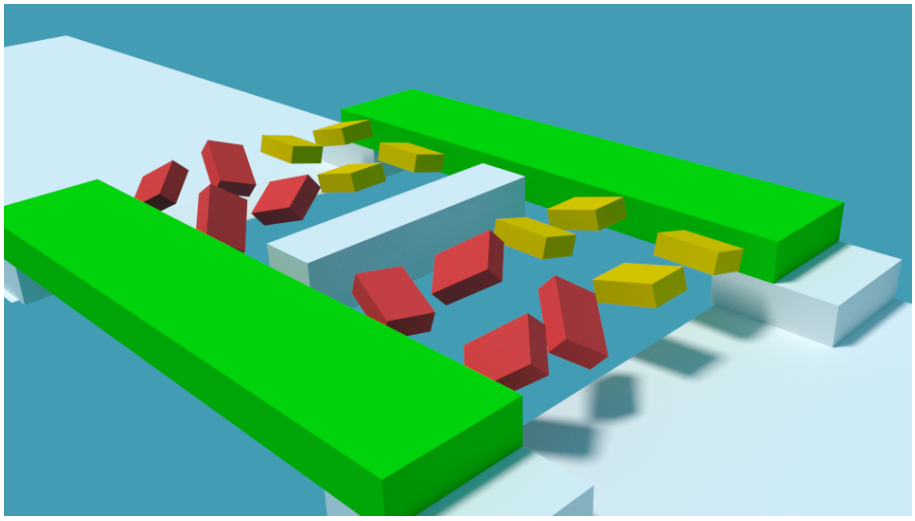


Figure 4.1: Uneven environment made from plane surfaces.

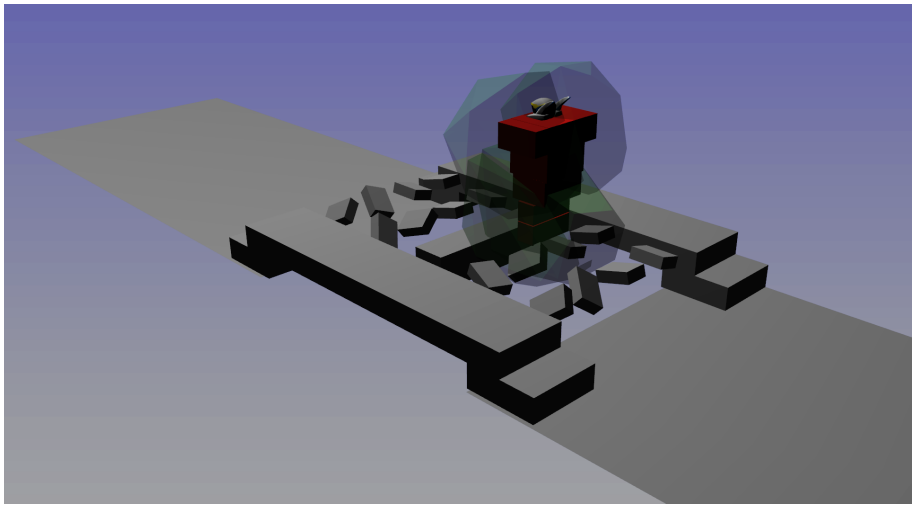


Figure 4.2: Reachability condition for HRP-2: the red shapes represent the minimum space to avoid collisions between the robot and the environment, the green shapes represent the spaces reachable by each end-effectors.

4.1.2 Stability for biped walkers

In the Loco3d cascade, a static sequence of key postures is first computed then the first elements of the dynamics are calculated using a preview controller on the centroidal dynamics of the robot. Let us first see what are the implications of the feasibility of this MPC on the planner that computes the sequence of key postures. To reduce the computation time and therefore allow to quickly start a movement and to adapt it to perturbations, preview controllers use small prediction windows (usually 2 steps). This window can be extended but the computation time will grow at least linearly (linearly for sparse algorithms like DDP, more otherwise) with the length of the prediction window (assuming that the time between each control is constant). Therefore, if we want the robot to quickly adapt its control if a perturbation occurs, we cannot directly compute the whole movement. To ensure that the robot will not fall, we need to ensure that the robot can reach a stable pose at the end of the prediction horizon [Wieber 2008]. Therefore, even if the environment is uneven, we need to have surfaces that allows stable position along the trajectory. The "distance" (in terms of steps) between two consecutive stable poses should be smaller than the length of the time window used. While Quadruped and multi-contact locomotion can have three or more contacts and many possible transitions, biped locomotion is intrinsically more constrained by the stability of the system. As a result, we need stronger guarantees that the movement can achieve stable poses. The feasibility condition currently used in our cascade is computationally efficient but suffers from several drawbacks: the approach does not consider the surface orientations so the surfaces can be too steep to generate a contact (due to the kinematics constraints of the limb); and there is no information on the stability

of the robot. For instance, the robot placement shown in Fig. 4.2 validate the reachability condition but it is unlikely that the robot would be able to generate a stable pose. The feasibility condition that will be constructed in this chapter has been aimed toward biped walking. The condition should render capabilities of the robot to generate stable poses.

4.1.3 State of the art

To rapidly search for stable poses in the environment, we will use an approach based on offline generation (for exploration) and machine learning (for encoding). Such approaches were already used to navigate in uneven terrain. Yang [Yang 2016] and Kang [Kang 2017] used learning techniques to construct reachability and occupancy estimators. However, these estimators do not take into account static or dynamic feasibility so their application on complex walk is limited to collision detection. Kalakrishnan [Kalakrishnan 2010] and Kolter [Kolter 2011] used demonstration from experts to recognize walkable terrains. These works were based on the quadruped *LittleDog* and therefore the stability criteria does not play such a key role. They rather analyze the terrain to choose a collision-free path and ensure non-slippage of the feet. Lin [Lin 2017] used *Support Vector Regression* to learn traversability over uneven terrains. However, they place themselves in the context of multicontact where stability is, once more, a lesser issue. Zucker [Zucker 2010] explicitly took into account stability of *LittleDog* by constructing a *pose cost*: for each set of contacts, a cost representing stability and non-collision is computed. Although a *terrain cost* was learned as in [Kalakrishnan 2010, Kolter 2011], the pose cost was explicitly calculated for each footstep and needed several steps of inverse kinematics. Therefore, computation was too expensive to analyze a large number of candidate poses.

In the next sections, we propose an original approach able to predict if a path is feasible. The prediction relies on a scoring estimation (built from data) that also makes it possible to quantify the difficulty of a path, hence to select a path among several possibilities. For instance, if the robot needs to go through the environment shown in Fig. 4.1, we would like the algorithm to analyze not only whether a path is possible, but also to be capable of selecting the most traversable one, i.e. the one using the green surfaces. The algorithm must be fast enough to analyze the whole environment while the results must reveal the capabilities of the robot.

4.2 Summary of the approach

The initial idea is that we already have existing tools to properly quantify the stability of the system, or the feasibility of a path in a 3D scene (e.g. using brute-force sampling and projections of contact configurations). However, even if more efficient, these methods are too slow to be called on the fly, in particular when replanning a path in response to new stimuli or new tasks.

We rather propose to sample stable configurations in random environments offline, to generate a "catalogue" of acceptable contact configurations. As the resulting dataset is too large for naive storage and access, we then propose to encode it using *Machine Learning* techniques, while trying to generalize across unexplored situations and despite marginal inconsistency due to the brute-force exploration. Once the encoding is achieved, the resulting knowledge corpus can be used to quickly predict and score stable situations, while computing a path in an environment map (typically using sampling-based motion planners) without having to explicitly compute the contact configurations. Such approach would allow to quickly search on an uneven terrain, the regions that the robot is likely capable of going through. The learned condition can be directly used instead of the heuristic-based reachability condition. We will also show that our approach provides a quality measure that can be used to select the best path among a set of feasible path, e.g. using $A*$ search The next section introduces several approaches that can be used to generate the data from which our stability estimator is learned. We show that the different approaches can lead to different results so data generation needs to be designed according to the application. Section 4.4 presents the model used to encode the dataset. Section 4.5 gathers the results of the learning as well as multiple applicative tests to show the pertinence of the methodology.

4.3 Data Generation

In this work, generating and encoding the data is done offline. Even if encoding the data seems a challenging task, our experience is that building the proper set of data is a key part that needs to be deeply thought. In this section we will explain more precisely which data we need to generate and how. In the following sections :

- $\mathcal{X} = \mathcal{X}_m \times \mathcal{X}_r \times \mathcal{X}_l$ where $\mathcal{X}_m$, $\mathcal{X}_r$ and $\mathcal{X}_l$ are respectively the sets of main-body placements (positions and orientations), right-foot placements and left-foot placements. Elements of $\mathcal{X}$ are denoted by $\boldsymbol{x}$.

- Ω is the set of parameters describing the environment and corresponds here to surface placements. Elements of Ω are denoted $\boldsymbol{\omega}$.

- $\mathcal{Q}$ is the configuration space of the robot, whose elements are $\boldsymbol{q} \in \mathcal{Q}$.

Any $\boldsymbol{x} \in \mathcal{X}$ can be obtained as the image of one or several $\boldsymbol{q} \in \mathcal{Q}$. The idea is to measure "how many" $\boldsymbol{q}$ are indeed mapped to each x, in each environment $\boldsymbol{\omega} \in \Omega$. Then at runtime, we can make a query knowing $\boldsymbol{\omega}$ only (i.e. marginalizing over $\boldsymbol{x}$ and $\boldsymbol{q}$) giving us an estimate of how likely a $\boldsymbol{q}$ (hence an $\boldsymbol{x}$) exists for this $\boldsymbol{\omega}$. Let us now develop the details of this idea.

4.3.1 What do we want to learn?

4.3.1.1 Occupancy Measure

We use occupancy measures to approximate our probability of finding stable poses [Carpentier 2017b]. Occupancy measures will give information about the size of the set of valid solutions. Carpentier defines the occupancy measure as follows:

$$\mu_o(\tilde{\boldsymbol{x}}) \triangleq \int_{\boldsymbol{q} \text{ s.t. } \gamma(\boldsymbol{q})=\tilde{\boldsymbol{x}}} \mathrm{d}\boldsymbol{q} = \int_{\mathcal{Q}} \mathbb{1}_{\gamma(\boldsymbol{q})=\tilde{\boldsymbol{x}}} \mathrm{d}\mu_{\mathcal{Q}} \tag{4.1}$$

where $\mathbb{1}_a$ is an indicator function (i.e. 1 is a is true, 0 otherwise), and in his case $\tilde{\boldsymbol{x}} \in \tilde{\mathcal{X}}$ represents the position of the center of mass in the end-effector frame. Occupancy measures quantify the "volume" of configurations $\boldsymbol{q}$ that are flattened (due to projection γ in space $\tilde{\mathcal{X}}$). From random samples, the occupancy measure will correspond to the local density of data points. Choosing an element with maximum occupancy measure will lead to a state with more possible configurations so more chance to get a valid one for the complete model.

4.3.1.2 Adding robot constraints

The indicator function $\mathbb{1}_\gamma$ is used to select the configurations to integrate. However they can also be used for other criteria like collisions with the environment, maximum torques in each joint or stability. For example, Carpentier uses the forward centroidal projection to construct its indicator function [Carpentier 2017b]. However, by sampling over the set of valid configurations $\mathcal{Q}$, he implicitly adds two other criteria on the configurations: no self-collision and joint positions between their bounds. In this work, we want to render the feasibility of generating a static pose, so a criterion on the stability of the robot is also added. For more clarity, we explicit all the criteria:

$$\mu_o(\boldsymbol{x}) \triangleq \int_{\mathcal{Q}} \mathbb{1}_{\gamma(\boldsymbol{q})=\boldsymbol{x}} \mathbb{1}_{\beta(\boldsymbol{q})>0} \mathbb{1}_{\boldsymbol{q_l} \leq \boldsymbol{q} \leq \boldsymbol{q_u}} \mathbb{1}_{\alpha(\boldsymbol{q})>0} \mathrm{d}\mu_{\mathcal{Q}} \tag{4.2}$$

where $\beta(\boldsymbol{q})$ is the vector of distances between each link (so $\beta(\boldsymbol{q}) > 0$ corresponds to no self-collision), $\boldsymbol{q_l}$ and $\boldsymbol{q_u}$ are respectively the lower and upper bounds on $\boldsymbol{q}$, and $\alpha(\boldsymbol{q})$ represents stability and is positive if the robot is stable. The stability criterion is computed using a linear (inner) approximation of the friction cone so the problem can be solved as a linear problem over the contact forces [Del Prete 2016b].

4.3.2 Sampling space

4.3.2.1 Sampling in $\mathcal{Q}$

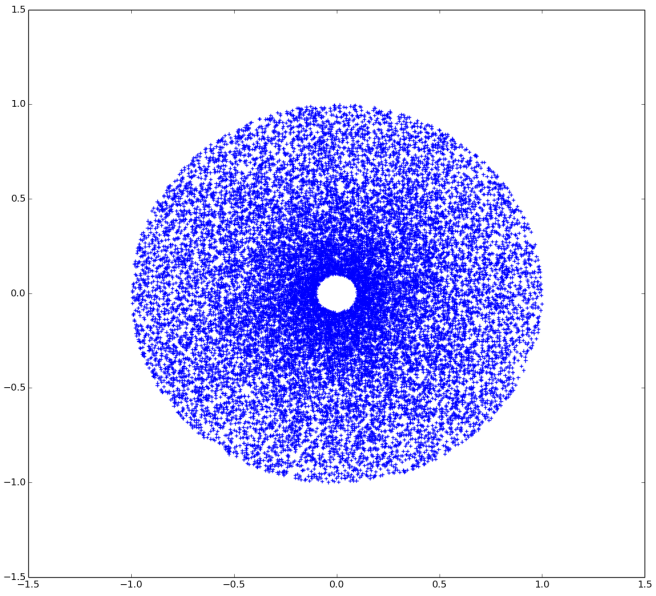
Several approaches can be used to generate the data. The idea of the occupancy measure is to have an image of the uniform distribution in $\mathcal{Q} \times \Omega$. However uniform sampling in Ω is not trivial to define, hence uniform distribution over $\mathcal{Q} \times \Omega$ seems

not achievable. A direct approximation is to uniformly sample $\mathcal{Q}$ (easy) and construct a $\boldsymbol{\omega} \in \Omega$ from $\boldsymbol{q} \in \mathcal{Q}$ afterwards. In [Carpentier 2017b], the relative position of the center of mass and the end-effectors is learned as a probability function. To generate the data, they generate random configurations using uniform probability on the position of each joint. This approach facilitates the data generation since the only computation needed to check if a data point is valid is the self-collision checking. Moreover it somehow reveals the system redundancies: if several configurations lead to the same position, the calculated probability will be higher.

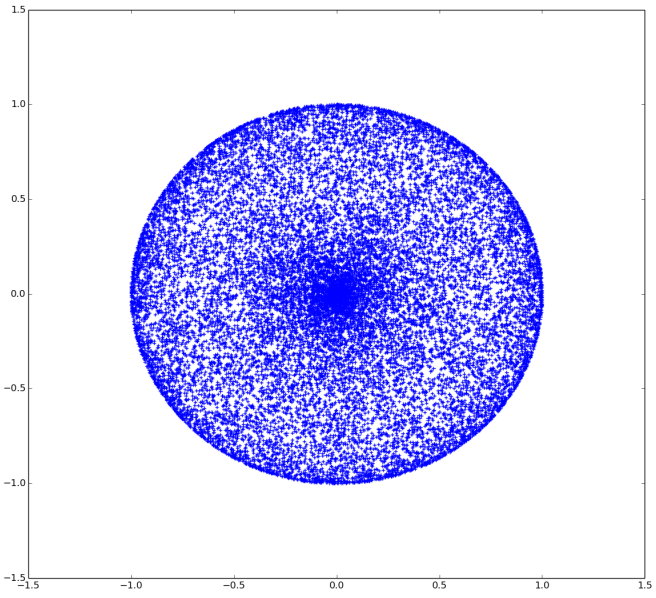
Effects of the forward kinematics on the distribution. Transforming a probability from the configuration space to the Cartesian space also leads to unwanted effects. For instance, if we generate this data for a system rotational and prismatic joints, even if there is no redundancy, transformation from a cylindrical space to a Cartesian space will lead to higher density so a higher occupancy measure near the center of rotation (Fig. 4.3a). Moreover, this behavior can drive the system toward singularities. For a double-pendulum system, all non-singular points can be reached with two configurations. However, if we generate the data using random configurations, we can see that we get higher densities near the singularities (Fig. 4.3b). If we use this measure, we will favor positions where a change in the configuration space will only result in small change in the Cartesian space. This behavior can be bad when considering constraints in the Cartesian space. In our case, a small change in the position of a contact surface would lead to a large modification of the configuration and therefore can more likely result in non-valid solutions (for instance because of a collision). This effect can be counterbalanced when there are redundancies. In [Carpentier 2017b], the relative position between the end-effectors and the CoM is considered as decoupled for each limb. Moreover, the CoM is a point so no relative orientation is taken into account. Each limb has 6 degrees of freedom, so if we consider the CoM as a fixed point in the root frame (and the torso and waist joints as fixed) then, for each position, occupation can be seen as the size of the set of possible root orientations. The occupation measures found by Carpentier show that in this case, redundancy drives the high occupation regions away from singularities (and joint bounds). For Carpentier's pattern generator, the contact points are already known and the lower controller is directly the whole-body controller so it is judicious to look for large whole-body solution sets. In our case, contact points are actually selected at the next step, so it is more relevant to maximize the set of possible contact points instead. Moreover, if we consider the root position/orientation as in the reachability-based planner, there is almost no redundancy left so the phenomenon explained earlier will have an important effect.

4.3.2.2 Sampling in $\mathcal{X}_m \times \Omega$

An intuitive approach is to directly sample constraints $(\boldsymbol{x}_m, \boldsymbol{\omega}) \in \mathcal{X}_m \times \Omega$ and use inverse kinematics to get a corresponding configuration. Let $\boldsymbol{q}_\gamma$ be the configuration obtained by applying inverse kinematics on a random configuration to generate

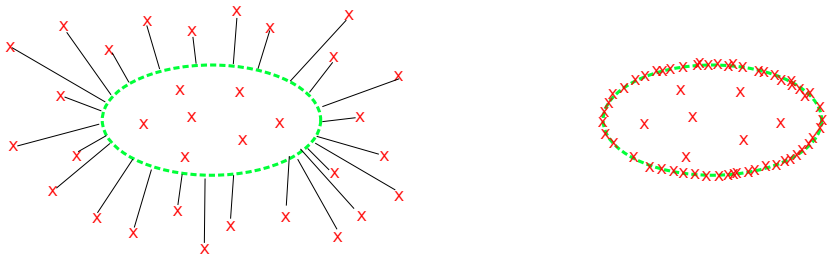


(a) rotational then prismatic joints ($\theta = [0, 2\pi]$, $l = [0.1, 1]m$)



(b) double pendulum ($l_1 = l_2 = 0.5m$, $\theta_1, \theta_2 \in [0, 2\pi]$)

Figure 4.3: Distribution from random configurations: each point represents the end-effector position after applying forward kinematics on a random configuration. Each configuration is generated from a uniform probability. The kinematic organization biases the Cartesian distribution and would push the resulting planner toward singular configurations.



(a) Data points generated from a uniform sampling.

(b) Data points after projection.

Figure 4.4: Deformation of the occupancy measure after a nonlinear projection. The dashed line represents the image set of a nonlinear projection. The projection biases the resulting sampling toward the borders.

contacts on $\boldsymbol{\omega}$.

$$\boldsymbol{q}_\gamma = \gamma_{\mu_\mathcal{Q}}^{-1}(\boldsymbol{x}_m, \boldsymbol{\omega}) \tag{4.3}$$

$$\mu_o(\boldsymbol{x}) \triangleq \int_\mathcal{Q} \mathbb{1}_{\exists \boldsymbol{q}_\gamma} \mathbb{1}_{\gamma(\boldsymbol{q}_\gamma)=\boldsymbol{x}} \mathbb{1}_{\beta(\boldsymbol{q}_\gamma)>0} \mathbb{1}_{\boldsymbol{q}_l \leq \boldsymbol{q}_\gamma \leq \boldsymbol{q}_u} \mathbb{1}_{\alpha(\boldsymbol{q}_\gamma)>0} \, \mathrm{d}\mu_\mathcal{Q} \tag{4.4}$$

where $\mathbb{1}_{\exists \boldsymbol{q}_\gamma}$ return 1 if the inverse kinematics converges and 0 otherwise.

Effects of the inverse kinematics on the distribution. Inverse kinematics can be seen as a nonlinear projection. Each point outside of the set (i.e. not in contact) is moved to the set. Therefore, if the projection can modify a variable of the occupancy measure, the projection of a uniform density will result in a non-uniform density. There will be a higher density on the borders because the points outside the set are projected to its border. Thus, it will drive the system to choose non-robust solutions (Fig. 4.4). If a variable is integrated to construct the occupancy measure, projection will not reveal the size of the set but only the probability of success of the projection. Moreover, if a criterion is checked, a smaller set can actually increase the probability to validate it. If the set of the projection is included in the valid set of the criteria, all points will be considered as valid and we will have a high density although the set is small. For instance, if we check stability of the robot when considering only the vertical position and the roll angle of the root and projecting the feet to a horizontal ground, we get the data repartition of Fig. 4.5. In that case, the higher the root is, the smaller the set of possible foot position is. However, the set is restrained to an area under the root so it will most likely results in a stable position (Fig. 4.6). This effect will also result in a non-smooth occupancy measure that an approximator will have difficulty to learn and therefore there is a good chance the approximation will overestimate the infeasible positions which are near these sharp boundaries.

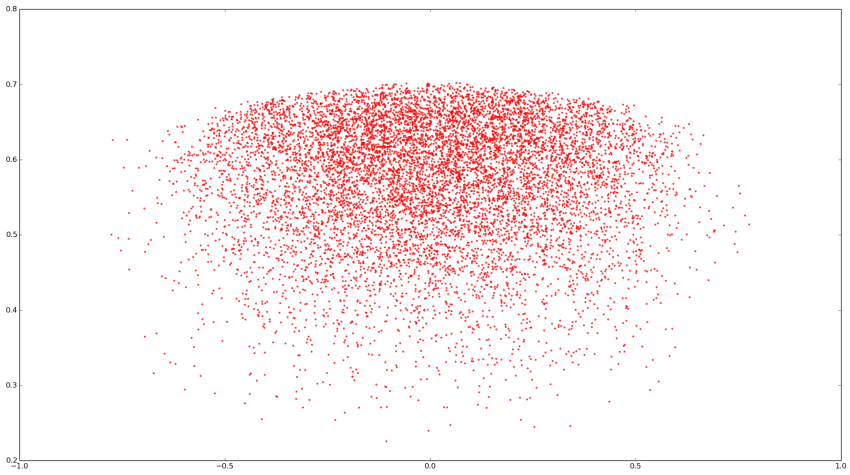
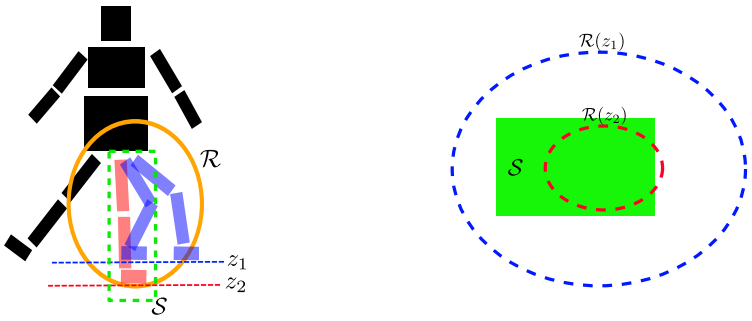


Figure 4.5: Data repartition after projecting the feet on a flat ground and rejecting non-valid configurations (i.e. unstable or in collision). The horizontal axis represents the roll angle of the main body θ_m and the vertical axis represents the distance between the main body and the ground z_m. Initially, (z_m, θ_m) were uniformly sampled. The resulting data repartition favors positions distant to the ground although the set of possible feet positions is very restrained. Once more, this bias would lead a resulting planner toward singular configurations (stretch legs).



(a) Robot configurations.

(b) Top view of the sets.

Figure 4.6: Left foot projection on different ground levels z_1 and z_2. $\mathcal{R}$ represents the reachability set and $\mathcal{S}$ the set of foot positions where the robot is stable (considering only a single contact).

4.3.2.3 Sampling in $\mathcal{X}$

Since inverse kinematics can bias the distribution, it will not be used to directly sample valid $\boldsymbol{x}$ points (i.e. by projecting it if not valid) but only as a criterion to know if a specific point is reachable or not:

$$\boldsymbol{q}_\gamma = \gamma_{\mu_q}^{-1}(\boldsymbol{x}) \tag{4.5}$$

$$\mu_o(\boldsymbol{x}) \triangleq \int_{\mathcal{Q}} \mathbb{1}_{\exists \boldsymbol{q}_\gamma} \mathbb{1}_{\gamma(\boldsymbol{q}_\gamma)=\boldsymbol{x}} \mathbb{1}_{\beta(\boldsymbol{q}_\gamma)>0} \mathbb{1}_{\boldsymbol{q_l} \leq \boldsymbol{q}_\gamma \leq \boldsymbol{q_u}} \mathbb{1}_{\alpha(\boldsymbol{q}_\gamma)>0} \, \mathrm{d}\mu_{\mathcal{Q}} \tag{4.6}$$

Then, we can marginalize over some variables to get the desired distribution. For instance, for a given main-body placement $\boldsymbol{x}_m$ and environment $\boldsymbol{\omega}$, we can get the "number" of feet position resulting in stable poses by marginalization:

$$\mu_o(\boldsymbol{x}_m, \boldsymbol{\omega}) \triangleq \int_{\mathcal{Q},\mathcal{X}_l,\mathcal{X}_r} \mathbb{1}_{\exists \boldsymbol{q}_\gamma} \mathbb{1}_{\gamma(\boldsymbol{q}_\gamma)=\boldsymbol{x}} \mathbb{1}_{\beta(\boldsymbol{q}_\gamma)>0} \mathbb{1}_{\boldsymbol{q_l} \leq \boldsymbol{q}_\gamma \leq \boldsymbol{q_u}} \mathbb{1}_{\alpha(\boldsymbol{q}_\gamma)>0} \, \mathrm{d}\mu_{\mathcal{Q}} \mathrm{d}\mu_{\mathcal{X}_l} \mathrm{d}\mu_{\mathcal{X}_r} \tag{4.7}$$

where $\mu_{\mathcal{X}_l}$ and $\mu_{\mathcal{X}_r}$ are uniform distribution on the contacts surfaces.

4.3.3 Implementation of the sampler

In this section we will define more precisely the variables of the problem and how they are considered. Following the reachability-based planner, we take into account the main body and want to determine the reachability and stability of the system on different surfaces. To do that, we need to compute the possible foot positions. Therefore, the variables considered here are: the orientation of the main body with respect to the acceleration axis (here we limit our study to stable pose so the acceleration axis is the gravitation axis); the orientation of the feet and the relative position of the feet and the main body. The variables of the upper body are considered as unknown and are uniformly sampled in the configuration space. If we directly consider all the geometric variables of the problem, the space to sample is a 14-dimension space, which is difficult to sample with a good resolution. Therefore, as a first approach, we will reduce the number of dimensions. Here, we consider only the rotation along the x-axis of the robot (backward-forward axis), for the main body and the feet. The rotations along the y-axis of the robot (right-left axis) are set to zero for the main body and the feet. Moreover, the orientation of the foot on the surface is left as unknown and is not sampled.

At the end, the sampled space is reduced to a 9-dimension space such as $\boldsymbol{x} = (\theta_m, \boldsymbol{x}_r, \theta_r, \boldsymbol{x}_l, \theta_l)$, where θ_m, θ_r and θ_r are the roll angle of respectively the main body, the right foot and the left foot; $\boldsymbol{x}_r$ and $\boldsymbol{x}_l$ are the relative position between the main body and the right/left foot.

4.3.3.1 Chosen frame

To easily take into account different position of the feet on surfaces and easily marginalized-out variables, the axis are aligned with the orientations of the surface

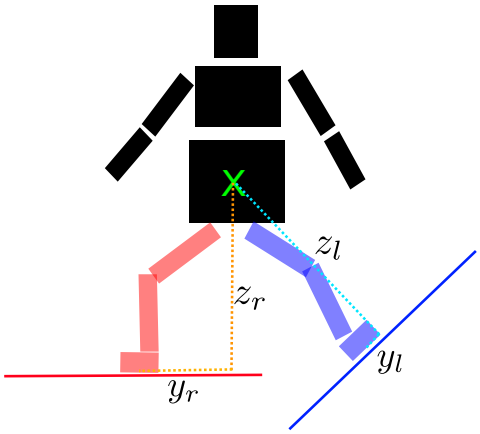


Figure 4.7: Frame used for the relative positions of the feet with respect to the main body.

(Fig. 4.7). $\boldsymbol{x}_i = x_i, y_i, z_i$ with $i = \{r, l\}$, where z_i is the distance between the surface and the main body and, (x_i, y_i) is the positions of the foot on this surface (the closest point to the main body is taken as origin).

4.3.3.2 From foot position to surfaces position

Until now, the generated data corresponds only to the possible state of the robot. At this stage, we would like to integrate the fact that we are not directly planning the foot position but we want to select the surfaces used for contacts. For now, the sizes of the surfaces are still unknown so we will only consider that the contact point needs to be close to the center of the surface. Thus, we assume that the distance between the contact point and the center of the surface respect a Gaussian distribution. Under this hypothesis, the generated data can be modified to correspond to the center of surface just by adding a Gaussian noise on the position of the feet. Since the chosen frames are aligned with the surfaces, the Gaussian noises can be directly added on (x_l, y_l) and (x_r, y_r).

4.4 Learning

At this step, we have a set of valid points. We can now evaluate a score for each state by computing the density of data points in its neighborhood. To estimate this density, we use *Kernel Density Estimator* with Gaussian kernels [Parzen 1962]. This estimator associates a Gaussian distribution to each data point. Then a score can be computed by summing the contribution of all the distributions. However, this computation becomes expensive when the number of data points increases. Let's take a numerical example to motivate the fact that directly applying classic estimation algorithm might not scale to the size of our problem. For Carpentier's

problem (3 dimensions, 10000 samples), the computation time for a single Gaussian is around $10ns$ so a score is computed in $100\mu s$. In our case, the dimension of the problem is higher so computing a single Gaussian distribution takes longer and we need more samples to cover the whole space. For instance, for a 9-dimension problem with 500000 samples, computing a score takes around $50ms$. In both cases, the computation is too long for their applications so the complexity of the estimator needs to be reduced. In the following, we first recall the basics in *Gaussian Mixtures* then we present how we design the learning scheme adequate for our context.

4.4.1 Gaussian Mixture Model

To reduce the complexity, we use a *Gaussian Mixture Model* (GMM) to approximate the KDE estimation. A Gaussian mixture distribution can be written as:

$$p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \tag{4.8}$$

with

$$\pi_k \in \mathbb{R}, \forall k \tag{4.9}$$

$$\sum_{k=1}^{K} \pi_k = 1 \tag{4.10}$$

where K is the number of components and $\mathcal{N}(x|\mu_k, \Sigma_k)$ is a Gaussian distribution of mean μ_k and covariance Σ_k. π_k is called the mixing coefficient. Therefore, instead of having N isotropic Gaussian distributions, where N is the number of samples, we will set K Gaussian distributions (with $K << N$) and optimize their parameters to fit to the density estimation. Gaussian Mixture Model as several advantages: the parameter estimation can be done with the *K-means* and *Expectation-Maximization* algorithms which can handle a high number of data points; marginal and conditional probabilities can easily be obtained from the joint probability; the resulting distribution can still be used as a sampler instead of a scoring system.

4.4.1.1 K-means clustering and Expectation-Maximization

In this section, we briefly recall the two algorithms used to fit the GMM to the generated data. The K-means algorithm is used to initialize the Expectation-Maximization (EM) algorithm. K-means algorithm partitions data points into K clusters by adjusting the cluster centers. Then, the EM algorithm optimizes the GMM parameters using the K clusters as an initial guess for the parameters of the K Gaussian distributions. The two algorithms are based on the same principle: they optimize their model using an iterative procedure in which each iteration can be divided into two phases, namely *expectation* and *maximization*. Each phase corresponds to an optimization with respect to a subset of the variables. For the K-

means algorithm, the expectation phase corresponds to assigning each data points to the closest cluster center. Then, the maximization phase optimize the cluster centers taking the data assignment as fixed. This operation corresponds to setting the cluster centers as the means of all the data points assigned to the cluster. The EM algorithm follows the same philosophy. However during the expectation phase, instead of assigning each point to a single component (i.e cluster for K-means, Gaussian distribution here), each point x_n are assigned to all components but weighted by their relative responsibilities r_{nk}:

$$r_{nk} = \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_j \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)}, \tag{4.11}$$

with $k = \{1..K\}$. Then, during the maximization phase the responsibilities are supposed fixed. The maximization phase maximize for each component the likelihood function $\ln p(X|\pi_k, \mu_k, \Sigma_k)$ with respect to the means, the covariances and the mixing coefficients:

$$\begin{aligned}
\mu_k &= \frac{1}{N_k} \sum_{n=1}^{N} r_{nk} x_n, \\
\Sigma_k &= \frac{1}{N_k} \sum_{n=1}^{N} r_{nk} (x_n - \mu_k)(x_n - \mu_k)^T, \\
\pi_k &= \frac{N_k}{N},
\end{aligned}$$

with

$$N_k = \sum_{n=1}^{N} r_{nk},$$

where N is the number of data points.

4.4.1.2 Conditional and marginal distributions

During the planning phase, some variables are known and fixed like the surface placements. Others are not considered yet and we only want to maximize the set of valid solutions. For instance, we can marginalized-out the main body placement and only score pairs of surfaces. To do that, we need to get the conditional and/or marginal probabilities of our distribution. GMM allows to easily derive the marginal and conditional distributions from the joint probability [Bishop 2007]. From the joint probability of the following GMM:

$$p(x_1, x_2) = \sum_k \pi_k \mathcal{N}(x_1, x_2|\mu_k, \Sigma_k),$$

where

$$\mu_k = \begin{bmatrix} \mu_{k1} \\ \mu_{k2} \end{bmatrix}, \quad \Sigma_k = \begin{bmatrix} \Sigma_{k11} & \Sigma_{k12} \\ \Sigma_{k21} & \Sigma_{k22} \end{bmatrix},$$

the marginal probability simply is:

$$p(x_1) = \sum_k \pi_k \mathcal{N}(x_1 | \mu_{k1}, \Sigma_{k11}), \tag{4.12}$$

and the conditional probability is:

$$p(x_1 | x_2) = \sum_k \pi_{k|2} \mathcal{N}(x_1 | \mu_{k1|2}, \Sigma_{k1|2}), \tag{4.13}$$

with

$$\begin{aligned} \pi_{k|2} &= \frac{\pi_k \mathcal{N}(x_2 | \mu_{k2}, \Sigma_{k22})}{\sum_k \mathcal{N}(x_2 | \mu_{k2}, \Sigma_{k22})}, \\ \mu_{k1|2} &= \mu_{k1} + \Sigma_{k12} \Sigma_{k22}^{-1} (x_2 - \mu_{k2}), \\ \Sigma_{k1|2} &= \Sigma_{k22} - \Sigma_{k12}^T \Sigma_{k11}^{-1} \Sigma_{k12}. \end{aligned}$$

Thus, instead of learning a marginal probability as in (4.7), we can learn the joint probability and marginalized it a posteriori.

4.4.2 Learning the data

The number of components K is a hyper-parameter that can be chosen by successive trials. The criterion used to select this number is a Monte Carlo estimate of the *Kullback-Leibler* (KL) divergence between the KDE taken as ground truth and the generated GMM [Hershey 2007]. This estimate is defined as:

$$D_{KL}(p_{KDE} || p_{GMM}) = \frac{1}{I} \sum_{i=1}^{I} \log \frac{p_{KDE}(x_i)}{p_{GMM}(x_i)}, \tag{4.14}$$

where x_i is a point sampled from the KDE distribution.

4.5 Results

In this section, we report the results obtained with the model of the humanoid robot HRP-2. We start by reporting the procedure used to generate the data points and show the results of the learning phase. Then we demonstrate on simple examples of environment that the data generated and the estimator learned allow to predict capabilities of the robot. The results are compared with the results obtained with the configuration-sampling strategy presented in Section 4.3.2.1. We also show that the estimator can be used to optimize the probability of generating a contact on a given environment. The last part shows a real application of the estimator in a

motion planner.

4.5.1 Implementation details

In the next sections, the contact planner used is *hpp-rbprm*, a part of the open software *Humanoid Path Planner* HPP [Mirabel 2016] (implemented in C++) specialized in contact planning. GMMs are learned and used with the python library *scikit-learn* while the KDE are used with a C++ implementation to speed up computations. All computations are done on an Intel Xeon(R) CPU E3-1240 v3 @ 3.40GHz (4 cores), 16GB RAM.

4.5.2 Offline phase

4.5.2.1 Generating the data

The data points are generated using HPP, a software developed in the *Gepetto* team. This software is built as a toolbox and implements all the basic functionalities for sampled-based motion planning. In particular it allows the user to easily define constraints on the movement to achieve. Points are generated using the following procedure:

- A point $\boldsymbol{x} = (\theta_m, \boldsymbol{x}_r, \theta_r, \boldsymbol{x}_l, \theta_l)$ is sampled from a uniform probability.
- A configuration q_0 is sampled from a uniform probability.
- The configuration q_0 is projected (using numerical optimization implemented in HPP) onto the space $\{q \in \mathcal{Q} | \gamma(\boldsymbol{q}) = \boldsymbol{x},\ \boldsymbol{q_l} \leq \boldsymbol{q} \leq \boldsymbol{q_u}\}$ using HPP.
- Self collision is checked with HPP again (this part uses the *Fast Collision Library* – FCL [Pan 2012]).
- The stability criterion is checked by solving a linear problem where the contact forces are optimized within cone constraints [Del Prete 2016b].

If the projection does not converge or results in a self-collision or in an unstable configuration, the point $\boldsymbol{x}$ is rejected and a new point is sampled. This operation is repeated until we get a sufficient number of samples. Data are generated over a large range of states. Following the procedure, we generated over 1 million valid points (success rate of 0.04%). All Datasets are available on `https://github.com/Mathieu-Geisert/StablePose-Dataset.git`). However, we will see in section 4.5.4.2 that a KDE with 200000 data points gives already good results. The generation of data took about one week on a desktop computer. However, this task can be easily reduced by parallelizing computation over several processors and/or by using prior knowledge on the valid set (reachability or stability) to reject invalid points without computing the inverse kinematics.

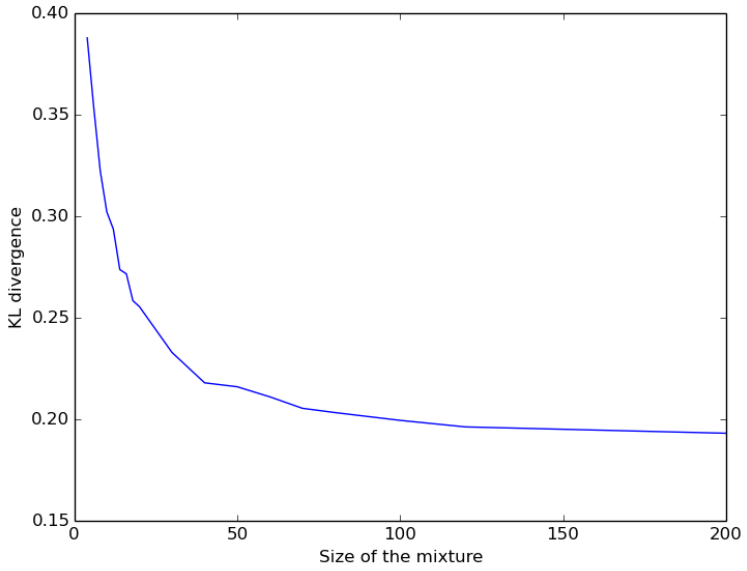


Figure 4.8: Evolution of the KL divergence with the number of components (4-200).

4.5.2.2 Selecting the number of GMM components

In this section, we select the number of components in the GMM by comparing the KL divergence with respect to the KDE taken as ground truth. To speed up computations, only a subset of 500000 samples is used. Fig. 4.8 shows the evolution of the estimated KL divergence (with respect to a KDE of one million data points) according to the number of components. After 80 components, the divergence is almost stable. Therefore, we choose this value for the tests presented in the next sections. Fitting this model to one million data points takes only 62 iteration of the EM algorithm but the whole process takes 28 minutes. GMMs and KDEs can easily be sampled regardless the number of components, however computing the probability associated to a sample depends linearly with this number. The transformation from a KDE of one million data points to an 80-component GMM allows to reduce the computation time of this probability from $167ms$ to $8\mu s$.

4.5.3 Stable poses on a simple environment

While it is quite easy to compare the learned approximation and the ground truth in small dimensions, it is rather difficult here to visualize the quality of the approximation. In this section, we test on simple but relevant examples of environment if the approximation obtained can indeed be used to predict capabilities of the robot. Environments are generated with only two square surfaces of 30 centimeters, ran-

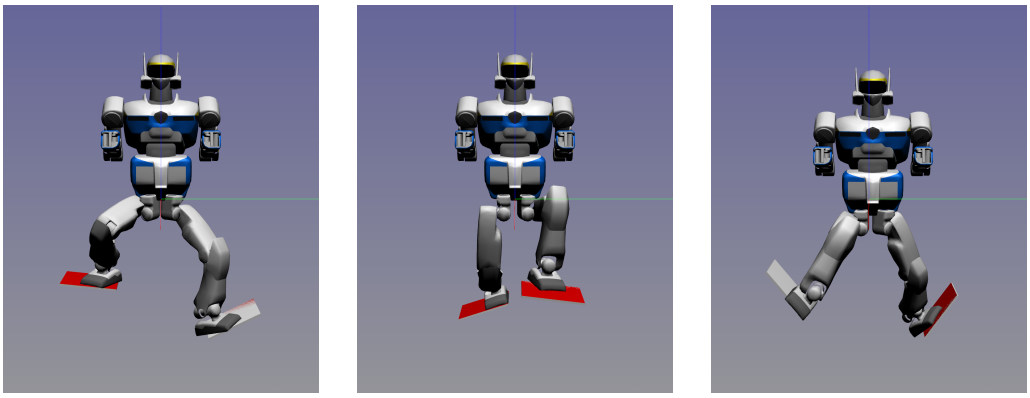


Figure 4.9: Stable poses in two-random-surface environments.

domly placed (see Fig. 4.9). The two surfaces are placed with the following rules:

$$\theta_{si} = (-1)^i U(-0.4, 1.2), \tag{4.15}$$

$$\Delta x = U(-0.4, 0.4), \quad \Delta y = U(0.1, 0.5), \quad \Delta z = U(-0.2, 0.2), \tag{4.16}$$

$$\boldsymbol{x}_{si} = (-1)^i \begin{bmatrix} \Delta x \\ \Delta y \\ \Delta z \end{bmatrix} \tag{4.17}$$

for $i = \{1, 2\}$. θ_{si} are the roll angles of the surfaces, $\boldsymbol{x}_{si}$ their positions and $U(a, b)$ is a value uniformly sampled between a and b. The purpose here is to see if we can use the information learned to predict and find stable poses on these simple environments. The results presented in the next sections use or are compared with a simple heuristic which place the main body as follows:

$$\begin{aligned} \boldsymbol{x}_m &= \frac{\boldsymbol{x}_{s1} + \boldsymbol{x}_{s2}}{2} + L\boldsymbol{z} \\ \theta_m &= 0 \end{aligned} \tag{4.18}$$

where $\boldsymbol{z}$ is a unit vertical vector and L is the parameter setting the vertical position of the main body. $\boldsymbol{x}_m$ and θ_m are respectively the position and roll angle of the main body.

4.5.4 Predicting stable poses

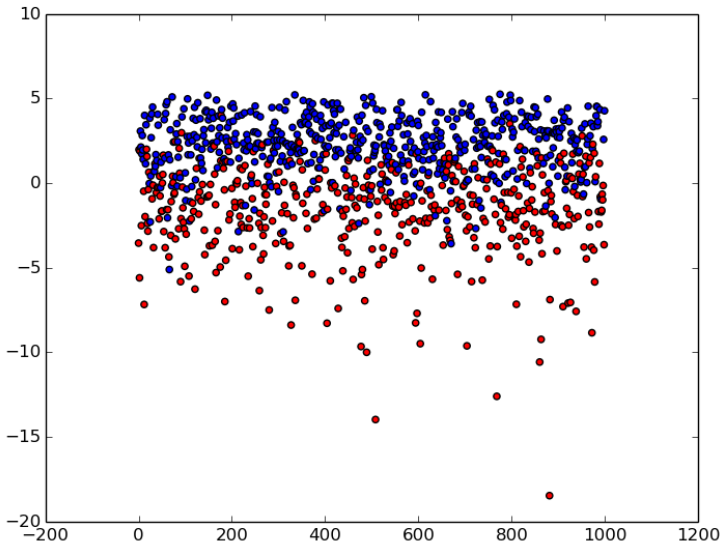
4.5.4.1 predicting the result of the contact planner

In this section we use learned information to predict if the robot is able to generate a stable contact knowing the main body position and the surface placements. Even if the contact planner is biased (due to the heuristics used to speed up computations), we use its results as ground truth. A random environment is generated and the main body is placed using the heuristic with $L = 0.45m$. Then we score the situation with

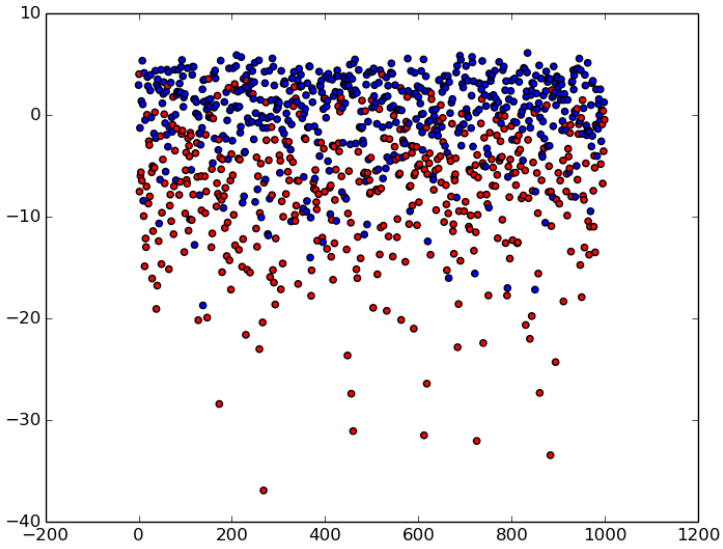
the GMM and ask the contact planner to find a stable pose. Fig. 4.10 shows the results. For comparison, we also show the results for the case where the learned data was generated from random configuration as in [Carpentier 2017a]. We can see that for both approaches, upon a certain score the planner mostly found stable poses. However, the results from random configurations are much more difficult to segregate. Fig. 4.11 shows the *Receiver Operating Characteristic* (ROC) diagrams for both strategies. It shows that sampling directly the state space gives a much better prediction rate (+5% up to +20%) for any false positive rate. Moreover, we believe the difference between the two strategies would be even bigger if L varies since the configuration-sampling strategy favor contacts close to the main body. Since the reachability condition does take into account neither surface orientations nor stability, its predictions can hardly be used. On the same tests, the true and false positive rates were respectively 0.56 and 0.46.

4.5.4.2 Set of foot positions

If we look back to the definition of the occupancy measure (4.7), with a given environment, the score does not correspond to the most stable main body position. It actually corresponds to the one maximizing to the number of foot positions resulting in stable poses. Therefore, if we randomly sample contact points on the surfaces, the success rate of finding a stable pose must correspond to the GMM scores (up to a regularization factor). We tested it on four environments and a large number of main body positions. For each main body position, the success rate is computed from 100 tests. Results are shown on Fig. 4.12. Fig. 4.13 and 4.15 show respectively the score given before the learning phase (i.e. generated by the KDE) and after (i.e. generated by the GMM). Fig. 4.16 show the score that would be obtained if we had sampled configurations instead of states. As expected, sampling configuration favor positions where the main body is close to the feet. For comparison, we also plot the results for a KDE (for the state-sampling strategy) with a larger dataset: 1000000 data points for Fig. 4.14 instead of 200000 data points for Fig. 4.13. Although the number of points seems small compared to the size of the space to explore, we can see that both distributions give similar results. This encourages us to push forward and to rely on the capacity of generalization of the method to apply it to larger space without exponentially increasing the size of the data set. Following these results, we tested the correspondence between the success rates and the GMM/KDE scores for the many random environments and main body positions. Fig. 4.17 shows a comparison of the results for the two strategies before any learning (i.e. scored from the KDEs). The results show that the configuration-sampling is unable to predict the success rate although the state-sampling strategy reveals a linear correlation between the two results. Fig. 4.18 shows the results for the state-sampling strategy after learning (i.e. scored with the GMM). Even if the variance of the results is quite high, the mean score follows the expected behaviors.



(a) state-sampling strategy



(b) configuration-sampling strategy

Figure 4.10: Stability prediction: the vertical axis represents the score given by the GMM, the horizontal axis is the test number (1 to 1000). The color represents the result of the contact planner: blue for stable poses - red for unstable ones. The distribution leads to better results (better separation of the data set) when using the state-sampling strategy.

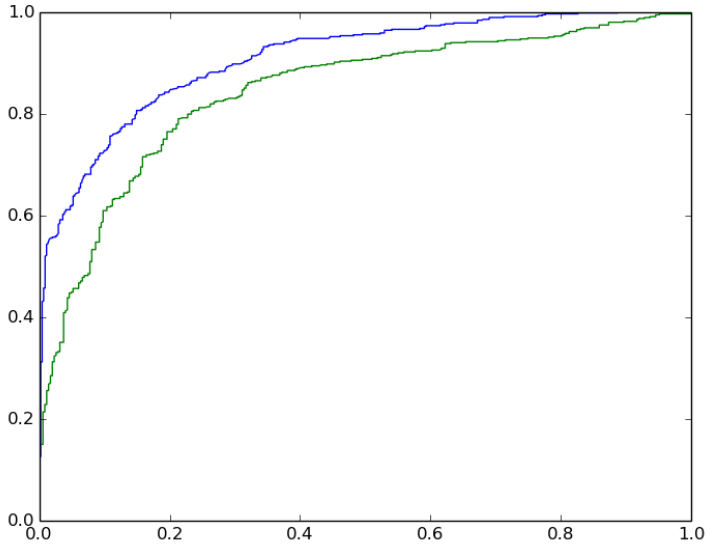


Figure 4.11: Stability prediction: ROC diagram. The horizontal axis represents the false positive rate and the vertical axis represents the true positive rate. The blue and green lines respectively correspond to the state-sampling and configuration-sampling strategies. Once more, the state-sampling strategy produces better results.

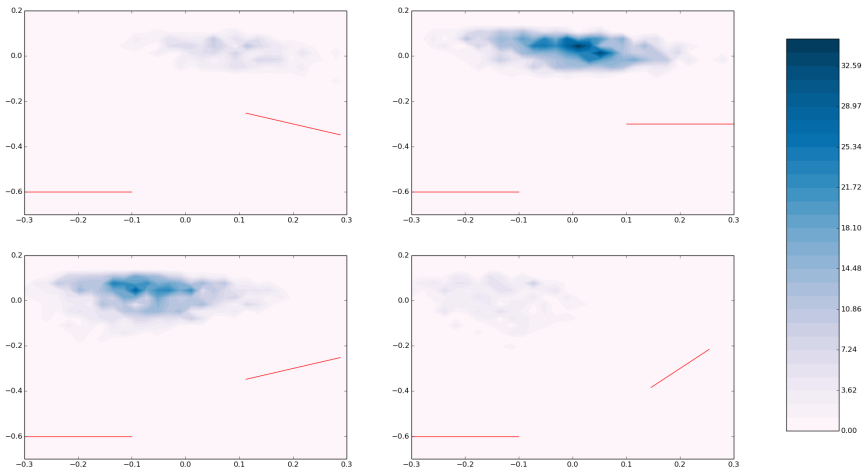


Figure 4.12: Main body positions: success rate obtained with random contact points on the surfaces. The red lines represent the contact surfaces.

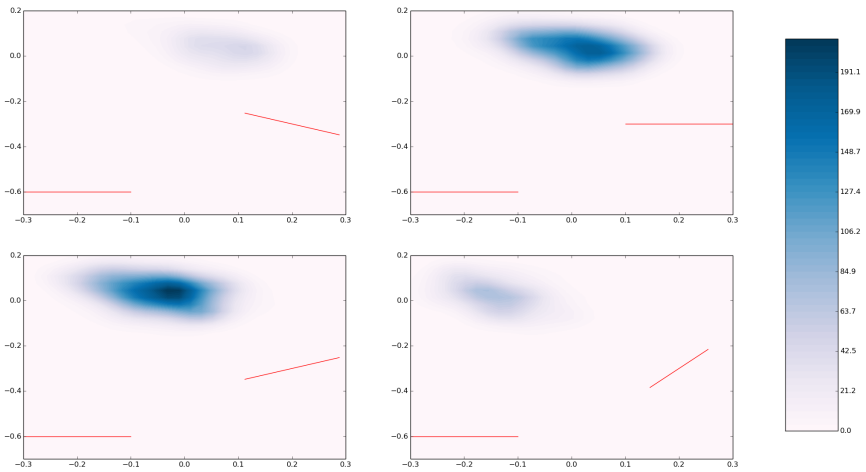


Figure 4.13: Main body positions: scores given before the learning phase (KDE 200000 data points).

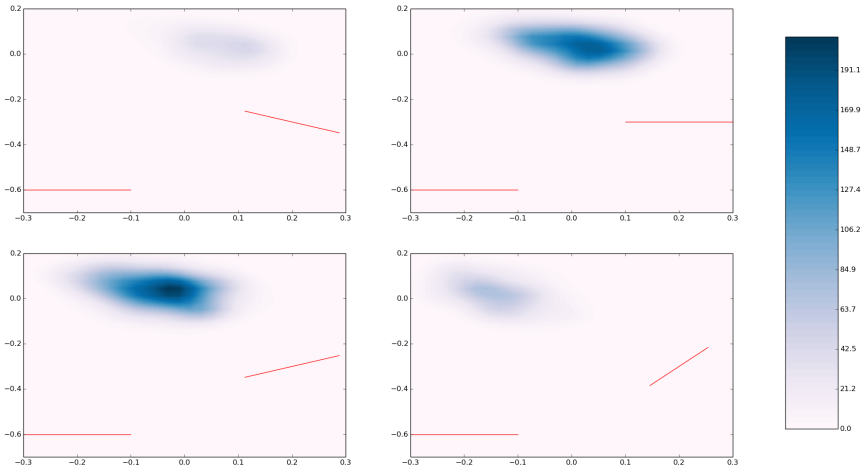


Figure 4.14: Main body positions: scores given before the learning phase (KDE 1000000 data points).

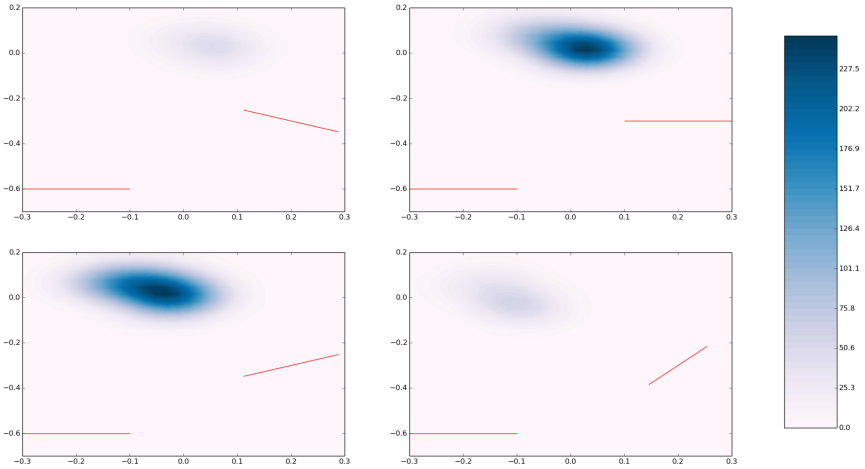


Figure 4.15: Main body positions: scores given after the learning phase, i.e. the GMM.

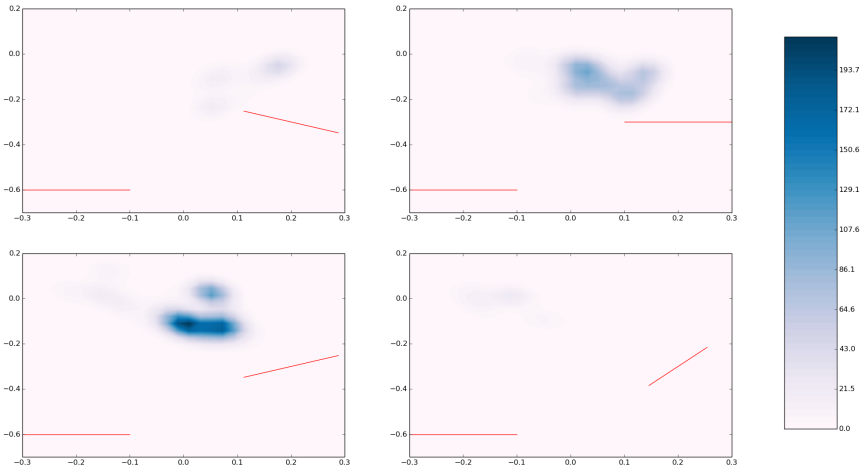


Figure 4.16: Main body positions: score given before the learning phase (KDE 200000 data points), with the random-configuration strategy.

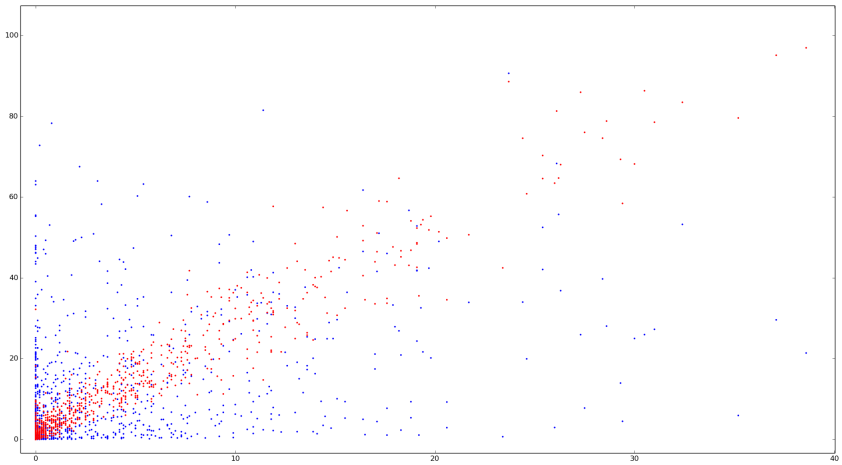


Figure 4.17: Set of foot positions: test on 1000 random environments and main body positions. The horizontal axis corresponds to the success rate with contact points uniformly sampled on the surfaces. The vertical axis corresponds to the KDE score. The red color represents scores from the state-sampling strategy and the blue represents the ones from the configuration-sampling strategy. Both KDE are generated from 200000 data points.

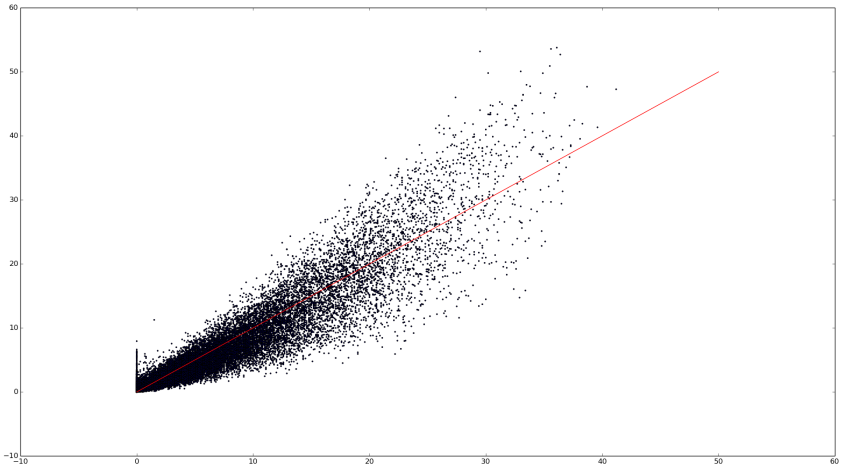


Figure 4.18: Set of foot positions: test on 10000 random environments and main body positions. The horizontal axis corresponds to the success rate with contact points uniformly sampled on the surfaces. The vertical axis corresponds to the GMM scores (with a scaling factor of 0.222).

4.5.5 Online query

We will develop here two situations where the results given by our learned model can be beneficial to find stable poses: selecting the main body position and selecting the contact point.

4.5.5.1 Choosing the contact points

Considering a given environment and a predefined main body position (i.e. Eq. (4.18) with $L = 0.45$), we want to generate (brute force and heuristics) a stable pose. The algorithm currently used in the reachability-based planner follows the following procedures:

1. The first foot defined in the model (here the left one) is projected to the surface. The initial configuration of the leg to project is selected from a set of random samples sorted with heuristics [Tonneau 2015]. If the projection fails, a new initial configuration is chosen from the set and the projection is repeated. If all samples fail, the surface is unreachable so the system is considered as unstable.

2. The same procedure is used to project the second foot (here the right one).

3. Then, the system stability is checked. If the system is stable the algorithm stops, otherwise a new initial configuration for the second leg is picked and step 2 and 3 are repeated.

4. If no stable pose was found for all the samples of the second leg, the system is considered as unstable.

Contacts are solved sequentially. Therefore, if the contact position of the first leg is misplaced, it can be impossible to find a contact position for the second leg resulting to a stable pose. Finding a better heuristic to choose the first contact position can seem a fake problem since when walking we already know one of the contact positions. However if we try to predict the stability after two steps, no contact position is defined yet. Moreover, if we are using jumps to move as in [Campana 2016] both contacts need to be chosen. We rather propose to use the GMM to select the best placement for the first foot. The procedure used is the same but instead of choosing the contact point for the first leg using simple heuristics, we discretize the surface and choose the contact point with the highest score given by the GMM. The results on 9400 random environments are shown in Tab. 4.1. They show that the contact position given by the GMM is not always the best one since the heuristic can find stable poses that were not found with the GMM. However, the learned model permits to increase the number of stable poses found of 11%.

4.5.5.2 Main body placement

In this section, we use the GMM to select the main body position. The best position is chosen by sampling positions over the vertical and the left-right axes and keeping

Table 4.1: Contact positions: stable positions over 9400 environments.

GMM \ Heuristic	stable	unstable	total
stable	4649	684	5333
unstable	149	3918	4067
total	4798	4602	9400

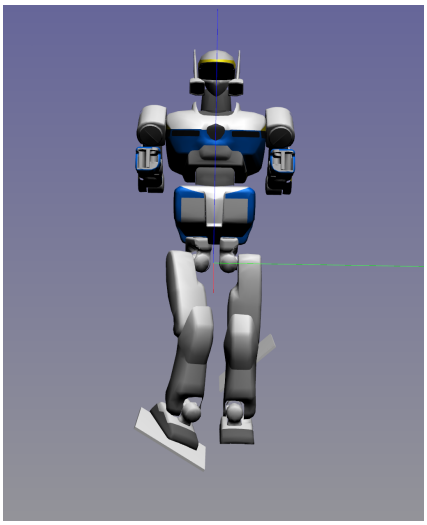
Table 4.2: Stable positions over 1000 environments with $L = 0.55m$.

GMM \ Heuristic	stable	unstable	total
stable	369	247	616
unstable	31	353	384
total	400	600	1000

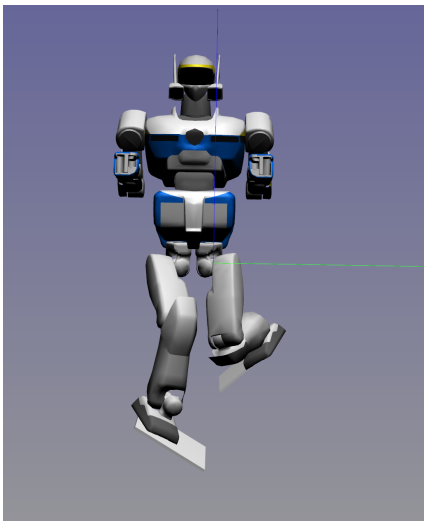
Table 4.3: Stable positions over 1000 environments with $L = 0.55m$. With only the situations where the reachability condition is valid.

	Heuristic	GMM
stable	223	380
unstable	275	221
rate	0.45	0.63

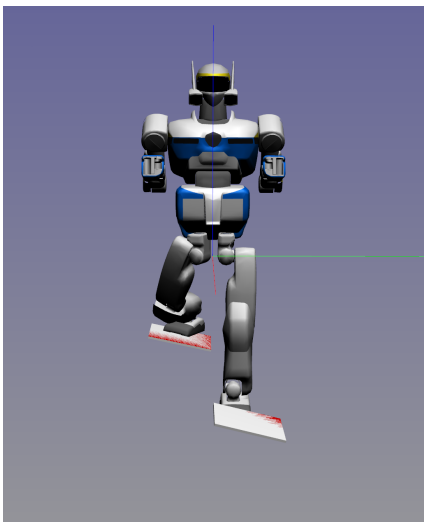
only the one with the highest score. The results are compared with the heuristic given in (4.18). In both cases, we project the feet to the surfaces with the same heuristics and check the stability of the system. In a first test, the value of L is chosen by selecting the highest marginal probability of the GMM on a flat ground, i.e. $L = 0.55m$. The results are summarized in Tab. 4.2. They show that the GMM can find good positions for the main body since the success rate grows from 40% to 61.6%. Some environments can be unreachable using the current heuristic, so to be fair, we also compare the success rates taking into account only the environments which validate the reachability condition. Tab. 4.3 shows that there is no real difference, the success rate still grows from 45% to 63%. Fig. 4.19 shows that sometimes the robot just needs a slight modification of its main body position to generate the contacts. In a second test, the vertical position L used for the heuristics is reduced to $L = 0.45m$ to increase the reachability of the feet. The vertical distance between the surfaces is also decreased ($\Delta z = U(-0.15, 0.15)$) to avoid generating unreachable surfaces for the heuristics. The results are shown in Tab. 4.4. The results are less impressive here but GMM still increases the success rate from 54.7% to 67.9%.



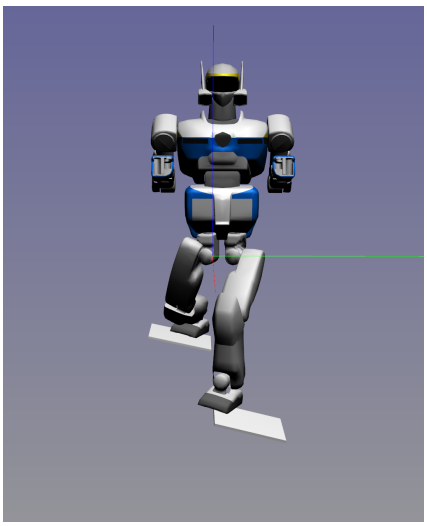
(a) Environment 1: heuristic



(b) Environment 1: GMM



(c) Environment 2: heuristic



(d) Environment 2: GMM

Figure 4.19: Stable pose found by the GMM but not by the heuristic.

Table 4.4: Stable positions over 1000 environments with $L = 0.45m$.

GMM \ Heuristic	stable	unstable	total
stable	530	149	679
unstable	17	304	321
total	547	453	1000

4.5.6 Integration to a path planner

In this section we test our approximator directly in a motion planner. The learned information can be used in several manners:

- to reject positions if its score is too low.
- to score position then use an $A*$ algorithm to select the best path.
- to sample points directly using the learned distribution.
- to optimize the score at each stage.
- or a mixed of some/all those approaches.

Moreover, we can use this tool at several steps of the planning:

- we can marginalized out the main body position/orientation to select pairs of surfaces that maximize the possible root positions.
- we can use it to select main body position that maximizes the possible foot positions.
- we can use it to select the best foot positions.

The integration of this work to the *Humanoid Path Planner* is still under process and will be reported in a future paper.

4.6 Conclusion

In this chapter, we used the machine learning framework to learn the capability of the robot to generate static poses. We showed that the data needs to be generated in such a way the sampling spaces and the projections do not bias the results. GMM is then used to reduce the complexity of the model. We exhibited versatility of the model: GMM can be used to score a pose but also as a random generator biased toward feasible poses; it can quickly compute marginal or conditional probability so can be used to solve various problems. Finally, we empirically demonstrated that the model learned can efficiently be used to predict the feasibility of static poses. Moreover, we are currently working to evaluate whether this model can be incorporated in a motion planner to improve the path of the robot and the selection of contact surfaces. The next step is to increase the number of variable to take into account rotations around the lateral axis. In addition, we could add criteria to take into account dynamic constraints to reach the static poses. This would be a first step toward integrating kinetic constraints to the reachability proxy. A direct way, that should not trigger the use of dynamics, is to sample robot state and velocity and validate them using an approximation of 0-step capturability such as the one introduced in [Del Prete 2017]. This work is a first step toward the construction of a general methodology to represent proxy constraints using machine learning.

A more challenging problem is open by the following stage of the cascade, where contact postures are computed. Here, a proxy should be defined to predict if the transition between two contacts is feasible. However, the space to sample is much larger and may involve more advanced machine learning methods. It would also be interesting to introduce, in the dataset, data coming from physical trial and to extend the learning procedure to online learning, in order to more properly adapt the planner exploration to the robot contact capabilities, imposed by the environment, in particular in outdoor scenarios.

ïż£

Conclusion

In Chapter 2, we showed the capabilities of *Direct Optimal Control* to generate complex behaviors. One advantage of this technique is that it does not need any analytical development and therefore can easily and quickly be adapted to different systems. Instead of relying on *human intelligence*, we rely on *algorithmic intelligence* to design efficient controllers. In Chapter 3, we extended this notion by developing a "smarter" algorithm. Even if optimal control permits to simplify the development of controllers, an important task that still relies on the designer is to adjust the parameters of the cost function. Therefore, we have developed a new algorithm capable of simplifying this task by solving the problem as a hierarchy. Instead of tuning the parameters associated with each subtask, the designer only needs to sort them by order of priority. Moreover, we showed that this new algorithm is able to generate useful behaviors that previous algorithms were incapable of. Another aspect of this document is the notion of memory. Even most genius humans strongly rely on knowledge learned from books or conferences, i.e. taught knowledge. Moreover, they improve because they are also able to explore new things and learn from those experiences. Therefore, storing and organizing knowledge is a key part of what we call *intelligence*. In Chapter 4, we made the robot learn its own capabilities. The robot generated a set of random samples revealing its kinematics capacity and stability, then efficiently stored the information to be able to quickly reuse it. We carefully designed the way the samples were picked and saved to reveal the size of the solution set. More specifically, in our case we wanted to be able to plan a path for the robot without explicitly computing each contact. Therefore, we generated samples revealing the "number" of feet positions. The acquired knowledge allows the robot to quickly analyze the difficulties of the terrain with respect to its own capabilities. Then it could use this analysis to select the best path for itself. The second part of Chapter 2 focused deeper into the concept of "intelligent" algorithms since the algorithm used was designed to explore and learn from its own experiences. It combines the capabilities of optimal control (to generate relevant solutions), sampling-based methods (to explore new areas) and machine learning (to accumulate knowledge). We showed that this algorithm is capable of generating more pertinent trajectories and in a faster way than optimal control or machine learning could do alone. In conclusion, machine learning has the capacity to quickly retrieve an approximate solution from previous experiences while optimal control can easily adapt it online to specific situations. The combination of those two approaches can solve complex problems that robotics is facing nowadays while staying generic. In terms of dynamic systems and tasks to accomplish. Machine learning and optimal control are now mature for real applications. As *Pablo Picasso* once said "Computers are useless. They can only give you answers". The challenge now is not to find the answer ("intelligent" algorithms do it for you), but to ask the correct question, i.e. to correctly formulate the problem and choose the right algorithms to

solve it. While *Boston Dynamics* (most probably) uses simple algorithms but a lot of manpower (and spare parts) to precisely tune each parameter, we rather study the topology of the problem and use the right set of powerful algorithms to solve it. *Loco3d* is a good example of such an approach, where the locomotion problem is studied and decoupled to solve each part of the problem with the appropriate tools. A direct application of the work presented in this thesis is the development of new controllers for UAVs. As we showed in Chapter 2, the algorithms used were capable of generating highly dynamic trajectories while ensuring the safety of the vehicle. Even if the computation time can appear very slow compared to the dynamics of quadrotors, a feedback controller can easily be obtained from the optimal solver and/or from the neural network. The main remaining tasks are the implementation of an efficient solver on the real hardware and the integration with the right set of tools to estimate the quadrotor state and its environment. The approach explored in this thesis, i.e. the combination of data-based and model-based approaches, is going to be studied more deeply thanks to the *Memory of Motion* (MEMMO) European project that just started in 2018. This project gathers 9 research partners including several industrial groups and aims at innovative applications: humanoid robots performing advanced locomotion and industrial tooling tasks in an aircraft assembly; advanced exoskeletons for paraplegic patients demonstrating dynamic walking; quadruped robots performing inspection tasks in a construction site. An interesting research direction that could be pushed further is an even more intimate intricacy of data-based and model-based approaches. In the IREPA algorithm, the optimal solver and the neural network are trained together but are built separately. Moreover, one only uses the output of the other. So could we build a more efficient structure to learn and/or to generate more precise results? For instance, could we use the neural network output as a cost function for the optimal solver? That way, the neural network could be used to automatically adapt the cost function along iteration and make the solver converge toward global minima. Also, could a neural network be improved by adding the robot model? We already have fast tools to compute the complete dynamics of the robot (e.g. *Pinocchio* [Mansard 2015]) so could we improve the results of a neural network by coupling the two of them? For instance, a neural network is unlikely able to approximate the space of rotations $SO(3)$ well. Thus, could we help it by inserting blocks that properly compute rotations? The more general question here is can we couple the knowledge of humans and machines?

Appendix A

Quadrotor based Systems

A.1 Costs weights and dynamic variables

Quadrotor (values are taken from measures on our own model): $m_q = 0.9[kg]$, distance between center of mass and rotor $d = 0.25[m]$, Inertia $J_q = diag(0.018, 0.018, 0.026)[kg.m^2]$, $C_f = 6.6 \times 10^{-5}$, $C_m = 1 \times 10^{-6}$, $V_{mot_i} \in [50, 300][rad.s^{-1}]$, $\dot{V}_{mot_i} \in [-314, 314][rad.s^{-1}]$. Quadrotor with pendulum : $m_p = 0.05[kg]$, $L = 4[m]$. Aerial manipulator : $m_q = 40[kg]$, $J_q = diag(10, 10, 20)[kg.m^2]$, distance rotor to center of mass $d = 1[m]$, $f_i \in [0, 200][N]$, *UR5* model taken from its official urdf file, $J_{mot} = 5.10^{-6}[kg.m^2]$, $K_{red} = 250$. Window : height $h = 2[m]$. Cost functions : $C_1 = 10^{-3}$, $C_2 = 10^{-2}$, $C_3 = 10^{-1}$, $C_4 = 10^{-2}$, $C_5 = C_7 = 10$, $C_6 = C_8 = 1$, $\rho = 2$

Appendix B

RHDDP

Hybrid Control Law

Here we prove the soundness of the hybrid control law (3.19). Recalling the definition given in Section 3.4.4, the hybrid control law $\tilde{U}_i^{(k)}$ must satisfy this equation:

$$c_i^{(l)}(x_i, \tilde{U}_i^{(l)}) = c_i^{(l)}(\hat{x}_i^{(l)}, \hat{U}_i^{(l)})$$

As usual, we can reformulate this as a constraint on the single control input $\tilde{u}_i^{(l)}$:

$$\tilde{\mathcal{V}}_i^{(l)}(x_i, \tilde{u}_i^{(l)}) = \tilde{\mathcal{V}}_i^{(l)}(\hat{x}_i^{(l)}, \hat{u}_i^{(l)})$$

Using the least-squares approximation of $\tilde{\mathcal{V}}$ and dropping the indexes i and l for the sake of simplicity we get:

$$\frac{1}{2} y^\top A^\top A y + h^\top y = \frac{1}{2} \hat{y}^\top A^\top A \hat{y} + h^\top \hat{y},$$

where $A = \begin{bmatrix} A_x & A_u \end{bmatrix}$, $y = (\delta x, \delta \tilde{u})$ and $\hat{y} = (\delta \hat{x}, \delta \hat{u})$. Let us perform a change of variable $y = \hat{y} + \Delta y$, where $\Delta y = (\Delta x, \Delta u)$:

$$\frac{1}{2} \Delta y^\top A^\top A \Delta y + \hat{y}^\top A^\top A \Delta y + a^\top A \Delta y = 0$$

Suppose now that $\Delta u = -K \Delta x$:

$$\left(\frac{1}{2} \Delta x^\top \begin{bmatrix} I & -K^\top \end{bmatrix} A^\top + \hat{y}^\top A^\top + a^\top \right) A \begin{bmatrix} I \\ -K \end{bmatrix} \Delta x = 0$$

A sufficient condition to satisfy this equation for any value of Δx is:

$$\begin{aligned} A \begin{bmatrix} I \\ -K \end{bmatrix} &= 0 \\ A_x - A_u K &= 0 \\ K &= A_u^\dagger A_x \\ K &= (A_u^\top A_u)^\dagger A_u^\top A_x \\ K &= Q_{uu}^\dagger Q_{ux} \end{aligned}$$

This gives us the hybrid control law (3.19):

$$\delta\tilde{u} = \delta\hat{u} - Q_{uu}^{\dagger} Q_{ux}(\delta x - \delta\hat{x})$$

Bibliography

[Al Borno 2012] Mazen Al Borno, Martin de Lasa and Aaron Hertzmann. *Trajectory Optimization for Full-Body Movements with Complex Contacts.* IEEE transactions on visualization and computer graphics, pages 1–11, dec 2012. (Cited in page 19.)

[Arleo 2013] G. Arleo, F. Caccavale, G. Muscio and F. Pierri. *Control of quadrotor aerial vehicles equipped with a robotic arm.* In Mediterranean Conference on Control Automation (MED), pages 1174–1180, Jun 2013. (Cited in page 35.)

[Baerlocher 1998] Paolo Baerlocher and R Boulic. *Task-priority formulations for the kinematic control of highly redundant articulated structures.* Intelligent Robots and Systems, no. 2, 1998. (Cited in page 16.)

[Baerlocher 2004] Paolo Baerlocher and Ronan Boulic. *An inverse kinematics architecture enforcing an arbitrary number of strict priority levels.* The Visual Computer, vol. 20, no. 6, jun 2004. (Cited in page 16.)

[Bellman 2015] RE Bellman and SE Dreyfus. Applied dynamic programming. Princeton university press, 2015. (Cited in page 100.)

[Benziane 2015] L. Benziane. *Attitude estimation and control of autonomous aerial vehicles.* PhD thesis, Jun 2015. (Cited in page 41.)

[Bhattacharya 2016] Subhrajit Bhattacharya. *Topological Motion Planning.* In IEEE International Conference on Robotics and Automation (ICRA Workshop 2016), 2016. (Cited in page 28.)

[Bishop 2007] Christopher M. Bishop. Pattern recognition and machine learning. Springer, New York, 2007. (Cited in page 136.)

[Boeuf 2015] Alexandre Boeuf, Juan Cortés, Rachid Alami and Thierry Siméon. *Enhancing sampling-based kinodynamic motion planning for quadrotors.* In IEEE/RSJ International Conference on Intelligent Robots and Systems, Hamburg, Germany, Sep 2015. (Cited in page 74.)

[Bouffard 2009] P. Bouffard and S. Waslander. *A Hybrid RandomizedNonlinear Programming Technique For Small Aerial Vehicle Trajectory Planning in 3D.* In IEEE/RJS International Conference on Intelligent Robots and Systems (IROS), 3rd Workshop on Planning, Perception and Navigation for Intelligent Vehicles (PPNIV), pages 63–68, Oct 2009. (Cited in pages 28, 39, and 44.)

[Brescianini 2013] D. Brescianini, M. Hehn and R. D'Andrea. *Quadrocopter pole acrobatics.* In IEEE/RJS International Conference on Intelligent Robots and Systems (IROS), pages 3472–3479, Nov 2013. (Cited in pages 35 and 45.)

[Campana 2016] Mylène Campana, Pierre Fernbach, Steve Tonneau, Michel Taïx and Jean-Paul Laumond. *Ballistic motion planning for jumping superheroes.* In Motion in Games Conference, Burlingame, CA, United States, Oct 2016. (Cited in page 147.)

[Carpentier 2016] Justin Carpentier, Steve Tonneau, Maximilien Naveau, Olivier Stasse and Nicolas Mansard. *A Versatile and Efficient Pattern Generator for Generalized Legged Locomotion.* Robotics and Automation (ICRA), 2016 IEEE International Conference on, 2016. (Cited in pages 13, 17, and 18.)

[Carpentier 2017a] Justin Carpentier, Rohan Budhiraja and Nicolas Mansard. *Learning Feasibility Constraints for Multi-contact Locomotion of Legged Robots.* In Robotics: Science and Systems, volume Porceedings of Robotics Science and Systems, page 9p., Cambridge, MA, United States, Jul 2017. (Cited in pages 17, 24, and 140.)

[Carpentier 2017b] Justin Carpentier and Nicolas Mansard. Multi-contact Locomotion of Legged Robots. Submitted to IEEE Transaction on Robotics, May 2017. (Cited in pages 127, 128, and 129.)

[Carpentier 2017c] Justin Carpentier, Andrea Prete, Steve Tonneau, Thomas Flayols, Alexis Mifsud, Kevin Giraud, Dinesh Atchuthan, Pierre Fernbach, Rohan Budhiraja, Justin Carpentier, Andrea Prete, Steve Tonneau, Thomas Flayols and Florent Forget. *Multi-contact Locomotion of Legged Robots in Complex Environments âĂŞ The Loco3D project.* In Robotics: Science and Systems Workshop, 2017. (Cited in pages 22 and 23.)

[Chaumette 2001] F. Chaumette and E. Marchand. *A redundancy-based iterative approach for avoiding joint limits: application to visual servoing.* IEEE Transactions on Robotics and Automation, vol. 17, no. 5, pages 719–730, Oct 2001. (Cited in page 16.)

[Chiaverini 1994] Stefano Chiaverini, Bruno Siciliano and Olav Egeland. *Review of the Damped Least-Squares Inverse Kinematics with Experiments on an Industrial Robot Manipulator.* Control Systems Technology, IEEE Transactions on, vol. 2, no. 2, pages 123–134, 1994. (Cited in page 90.)

[Dai 2014] Hongkai Dai, Andres Valenzuela and Russ Tedrake. *Whole-body Motion Planning with Simple Dynamics and Full Kinematics.* International Conference on Humanoid Robots, no. JANUARY 2014, pages 295–302, 2014. (Cited in page 18.)

[de Crousaz 2015] C. de Crousaz, F. Farshidian, M. Neunert and J. Buchli. *Unified Motion Control for Dynamic Quadrotor Maneuvers Demonstrated on Slung Load and Rotor Failure Tasks.* In IEEE International Conference on Robotics and Automation (ICRA), pages 2223–2229, May 2015. (Cited in pages 20, 35, and 61.)

[De Lasa 2009] Martin De Lasa and Aaron Hertzmann. *Prioritized optimization for task-space control*. In Intelligent Robots and Systems, IEEE/RSJ International Conference on, volume 3, pages 5755–5762. Ieee, oct 2009. (Cited in page 16.)

[Decré 2013] Wilm Decré, Herman Bruyninckx and Joris De Schutter. *Extending the iTaSC Constraint-based Robot Task Specification Framework to Time-Independent Trajectories and User-Configurable Task Horizons*. In Robotics and Automation (ICRA), IEEE International Conference on, pages 1933–1940, 2013. (Cited in page 90.)

[DeepMind 2017] DeepMind. *Producing flexible behaviours in simulated environments*, 2017. https://deepmind.com/blog/producing-flexible-behaviours-simulated-environments/. (Cited in page 29.)

[Deits 2014] Robin Deits and Russ Tedrake. *Footstep Planning on Uneven Terrain with Mixed-Integer Convex Optimization*. Groups.Csail.Mit.Edu, 2014. (Cited in page 26.)

[Del Prete 2014] Andrea Del Prete, Francesco Romano, Lorenzo Natale, Giorgio Metta, Giulio Sandini and Francesco Nori. *Prioritized Optimal Control*. In Robotics and Automation (ICRA), IEEE International Conference on, 2014. (Cited in pages 90, 91, and 103.)

[Del Prete 2015] Andrea Del Prete, Francesco Nori, Giorgio Metta and Lorenzo Natale. *Prioritized Motion-Force Control of Constrained Fully-Actuated Robots: "Task Space Inverse Dynamics"*. Robotics and Autonomous Systems, vol. 63, pages 150–157, 2015. (Cited in pages 20, 21, and 90.)

[Del Prete 2016a] Andrea Del Prete and Nicolas Mansard. *Robustness to Joint-Torque Tracking Errors in Task-Space Inverse Dynamics*. IEEE Transaction on Robotics, vol. 32, no. 5, pages 1091 – 1105, 2016. (Cited in page 24.)

[Del Prete 2016b] Andrea Del Prete, Steve Tonneau and Nicolas Mansard. *Fast Algorithms to Test Robust Static Equilibrium for Legged Robots*. In Robotics and Automation (ICRA), 2016 IEEE International Conference on, 2016. (Cited in pages 128 and 138.)

[Del Prete 2017] Andrea Del Prete, Steve Tonneau and Nicolas Mansard. Zero Step Capturability for Legged Robots in Multi Contact. working paper or preprint, Dec 2017. (Cited in page 150.)

[Diehl 2001] Moritz Diehl. *Real-Time Optimization for Large Scale Nonlinear Processes*. PhD thesis, 2001. (Cited in page 27.)

[Diehl 2005] M. Diehl, H.G. Bock, H. Diedam and P.-B. Wieber. *Fast Direct Multiple Shooting Algorithms for Optimal Robot Control*. In Fast Motions in

Biomechanics and Robotics, volume 340, pages 65–93, 2005. (Cited in page 37.)

[Diehl 2009] M. Diehl, H.J. Ferreau and N. Haverbeke. *Efficient Numerical Methods for Nonlinear MPC and Moving Horizon Estimation.* In Nonlinear Model Predictive Control, volume 384, pages 391–417, 2009. (Cited in page 49.)

[Dimitrov 2014] Dimitar Dimitrov, Pierre-Brice Wieber and Adrien Escande. *Multi-Objective Control of Robots.* Journal of the Robotics Society of Japan, vol. 32, no. 6, pages 512–518, 2014. (Cited in page 92.)

[Dimitrov 2015] Dimitar Dimitrov, Alexander Sherikov and Pierre-Brice Wieber. *Efficient resolution of potentially conflicting linear constraints in robotics.* IEEE Transaction on Robotics (under review), 2015. (Cited in pages 90 and 91.)

[Doncieux 2014] S. Doncieux and J.-B. Mouret. *Beyond black-box optimization: a review of selective pressures for evolutionary robotics.* Evolutionary Intelligence, vol. 7, no. 2, pages 71–93, 2014. (Cited in page 26.)

[El Khoury 2013] Antonio El Khoury. *Planning Optimal Motions for Anthropomorphic Systems.* Theses, Université Paul Sabatier - Toulouse III, Jun 2013. (Cited in page 28.)

[Escande 2010] Adrien Escande, Nicolas Mansard and Pierre-Brice Wieber. *Fast resolution of hierarchized inverse kinematics with inequality constraints.* In Robotics and Automation (ICRA), IEEE International Conference on, number 4, pages 3733–3738. IEEE, 2010. (Cited in pages 17, 90, and 92.)

[Escande 2014] Adrien Escande, Nicolas Mansard and Pierre-Brice Wieber. *Hierarchical Quadratic Programming: Fast Online Humanoid-Robot Motion Generation.* International Journal of Robotics Research, vol. 33, no. 7, pages 1006–1028, 2014. (Cited in pages 16, 91, 92, 93, and 95.)

[Faessler 2018] Matthias Faessler, Antonio Franchi and Davide Scaramuzza. *Differential Flatness of Quadrotor Dynamics Subject to Rotor Drag for Accurate, High-Speed Trajectory Tracking.* IEEE Robot. Autom. Lett., vol. 3, no. 2, pages 620–626, apr 2018. (Cited in page 14.)

[Faust 2013] A. Faust, I. Palunko, P. Cruz, R. Fierro and L. Tapia. *Learning Swing-free Trajectories for UAVs with a Suspended Load.* In IEEE International Conference on Robotics and Automation (ICRA), pages 4902–4909, May 2013. (Cited in page 35.)

[Faverjon 1987] Bernard Faverjon and Pierre Tournassoud. *The Mixed Approach for Motion Planning: Learning Global Strategies from a Local Planner.* In IJCAI, 1987. (Cited in page 16.)

[Featherstone 2007] R. Featherstone. Rigid body dynamics algorithms. 2007. (Cited in pages 43 and 44.)

[Garimella 2015] G. Garimella and M. Kobilarov. *Towards model-predictive control for aerial pick-and-place.* In IEEE International Conference on Robotics and Automation (ICRA), pages 4692–4697, May 2015. (Cited in pages 35 and 40.)

[Geisert 2016] Mathieu Geisert and Nicolas Mansard. *Trajectory Generation for Quadrotor Based Systems using Numerical Optimal Control.* In International Conference on Robotics and Automation (ICRA 2016), pages pp. 2958–2964, Stockholm, Sweden, May 2016. IEEE. (Cited in page 29.)

[Geoffroy 2014] Perle Geoffroy, Nicolas Mansard and M Raison. *From Inverse Kinematics to Optimal Control. ...* in Robot Kinematics, pages 1–8, 2014. (Cited in page 20.)

[Ghadiok 2011] V. Ghadiok, J. Goldin and W. Ren. *Autonomous indoor aerial gripping using a quadrotor.* In IEEE/RJS International Conference on Intelligent Robots and Systems (IROS), pages 4645–4651, Sep 2011. (Cited in page 35.)

[Gienger 2005] M. Gienger, H. Janssen and C. Goerick. *Task-oriented whole body motion for humanoid robots.* In 5th IEEE-RAS International Conference on Humanoid Robots, 2005., pages 238–244, Dec 2005. (Cited in page 16.)

[Grant 2014] Michael Grant and Stephen Boyd. *CVX: Matlab software for disciplined convex programming, version 2.1. http://cvxr.com/cvx*, 2014. (Cited in page 26.)

[Hamalainen 2015] Perttu Hamalainen, Joose Rajamaki and C. Karen Liu. *Online Control of Simulated Humanoids Using Particle Belief Propagation.* ACM Transactions on Graphics, pages 1–13, 2015. (Cited in page 26.)

[Herdt 2010a] a Herdt, N Perrin and Pierre-Brice Wieber. *Walking without thinking about it.* 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 190–195, oct 2010. (Cited in pages 18 and 23.)

[Herdt 2010b] Andrei Herdt, Holger Diedam, Pierre-brice Wieber, Dimitar Dimitrov, Katja Mombaur and Moritz Diehl. *Online Walking Motion Generation with Automatic Foot Step Placement.* Advanced Robotics, vol. 24, no. 5-6, 2010. (Cited in pages 13, 17, 19, 22, and 23.)

[Hershey 2007] J. R. Hershey and P. A. Olsen. *Approximating the Kullback Leibler Divergence Between Gaussian Mixture Models.* In 2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07, volume 4, pages IV–317–IV–320, April 2007. (Cited in page 137.)

[Herzog 2016] Alexander Herzog, Nicholas Rotella, Sean Mason, Felix Grimminger, Stefan Schaal and Ludovic Righetti. *Momentum control with hierarchical inverse dynamics on a torque-controlled humanoid.* Autonomous Robots, vol. 40, no. 3, pages 473–491, 2016. (Cited in page 92.)

[Hoffmann 2011] Christian Hoffmann, Christian Kirches, Andreas Potschka, Sebastian Sager and Leonard Wirsching. *Muscod User Manual*, 2011. `http://www.iwr.uni-heidelberg.de/groups/agbock/FILES/muscod_manual.pdf`. (Cited in page 13.)

[Homsi 2016] Saed Al Homsi, Alexander Sherikov, Dimitar Dimitrov and Pierre-Brice Wieber. *A hierarchical approach to minimum-time control of industrial robots.* In Robotics and Automation (ICRA), IEEE International Conference on, pages 2368–2374, 2016. (Cited in page 91.)

[Houska 2011] B. Houska, H.J. Ferreau and M. Diehl. *ACADO Toolkit – An Open Source Framework for Automatic Control and Dynamic Optimization.* In Optimal Control Applications and Methods, volume 32, pages 298–312, 2011. (Cited in page 45.)

[Isermann 1982] H. Isermann. *Linear lexicographic optimization.* OR Spektrum, vol. 4, no. 4, pages 223–228, dec 1982. (Cited in page 91.)

[Jacobson 1970] D Jacobson and D Mayne. Differential dynamic programming. Elsevier, New York, 1970. (Cited in page 102.)

[Jimenez-Cano 2013] A.E Jimenez-Cano, J. Martin, G. Heredia, A. Ollero and R. Cano. *Control of an aerial robot with multi-link arm for assembly tasks.* In IEEE International Conference on Robotics and Automation (ICRA), pages 4916–4921, May 2013. (Cited in page 35.)

[Kajita 2003] Shuuji Kajita and F. Kanehiro. *Biped walking pattern generation by using preview control of zero-moment point.* In Robotics and Automation (ICRA), 2003 IEEE International Conference on, 2003. (Cited in pages 12, 17, and 19.)

[Kajita 2010] S. Kajita, M. Morisawa, K. Miura, S. Nakaoka, K. Harada, K. Kaneko, F. Kanehiro and K. Yokoi. *Biped walking stabilization based on linear inverted pendulum tracking.* In 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, pages 4489–4496, Oct 2010. (Cited in page 18.)

[Kalakrishnan 2010] Mrinal Kalakrishnan, Jonas Buchli, Peter Pastor, Michael Mistry and Stefan Schaal. *Learning, planning, and control for quadruped locomotion over challenging terrain.* The International Journal of Robotics Research, vol. 30, no. 2, pages 236–258, nov 2010. (Cited in pages 22, 24, 25, and 126.)

[Kang 2017] Changgu Kang and Sung-Hee Lee. *Multi-Contact Locomotion Using a Contact Graph with Feasibility Predictors*. ACM Trans. Graph., vol. 36, no. 2, apr 2017. (Cited in page 126.)

[Kanoun 2011] Oussama Kanoun. *Kinematic Control of Redundant Manipulators: Generalizing the Task-Priority Framework to Inequality Task. ...* , IEEE Transactions on, pages 1–9, 2011. (Cited in page 16.)

[Khatib 1986] Oussama Khatib. *Real-Time Obstacle Avoidance for Manipulators and Mobile Robots*. The International Journal of Robotics Research, vol. 5, no. 1, 1986. (Cited in page 16.)

[Kingma 2014] Diederik Kingma and Jimmy Ba. *Adam: A method for stochastic optimization*. arXiv preprint arXiv:1412.6980, 2014. (Cited in page 76.)

[Kirches 2012] C. Kirches, H.G. Bock, J.P. Schlöder and S. Sager. *Complementary Condensing for the Direct Multiple Shooting Method*. In Modeling, Simulation, and Optimization of Complex Processes, pages 195–206, 2012. (Cited in page 39.)

[Kirk 1970] Donald E Kirk. Optimal control theory: an introduction. Courier Dover Publications, 1970. (Cited in page 99.)

[Koenemann 2015] Jonas Koenemann, Andrea Del Prete, Yuval Tassa, Emanuel Todorov, Olivier Stasse, Maren Bennewitz and Nicolas Mansard. *Whole-body Model-Predictive Control applied to the HRP-2 Humanoid*. In Intelligent Robots and Systems (IROS 2015), IEEE International Conference on, 2015. (Cited in pages 19, 89, and 122.)

[Kolter 2011] J. Zico Kolter and Andrew Y Ng. *The Stanford LittleDog: A learning and rapid replanning approach to quadruped locomotion*. The International Journal of Robotics Research, vol. 30, no. 2, pages 150–174, 2011. (Cited in pages 24, 25, and 126.)

[Lasserre 2001] Jean B. Lasserre. *Global Optimization with Polynomials and the Problem of Moments*. SIAM Journal on Optimization, vol. 11, pages 796–817, 2001. (Cited in page 26.)

[Liegeois 1977] A Liegeois. *Automatic supervisory control of the configuration and behavior of multibody mechanisms*. IEEE Transactions on Systems, Man, and Cybernetics, no. 12, pages 868–871, 1977. (Cited in page 16.)

[Lin 2017] Yu-Chi Lin and Dmitry Berenson. *Humanoid navigation in uneven terrain using learned estimates of traversability*. In 17th IEEE-RAS International Conference on Humanoid Robotics, Humanoids 2017, Birmingham, United Kingdom, November 15-17, 2017, pages 9–16, 2017. (Cited in page 126.)

[Mansard 2007] N. Mansard, O. Stasse, F. Chaumette and K. Yokoi. *Visually-Guided Grasping while Walking on a Humanoid Robot.* In Proceedings 2007 IEEE International Conference on Robotics and Automation, pages 3041–3047, April 2007. (Cited in page 16.)

[Mansard 2008] Nicolas Mansard and Oussama Khatib. *Continuous Control Law from Unilateral Constraints Application to Reactive Obstacle Avoidance in Operational Space.* In International Conference on Robotics and Automation, pages 3359–3364, 2008. (Cited in page 16.)

[Mansard 2009] Nicolas Mansard, Olivier Stasse, Paul Evrard and Abderrahmane Kheddar. *A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: the Stack of Tasks.* In ICAR'09: International Conference on Advanced Robotics, pages 1–6, Munich, Germany, Jun 2009. (Cited in page 17.)

[Mansard 2015] N. Mansard, J. Carpentier, F. Valenza*et al. Pinocchio : Dynamic computations using Spatial Algebra*, 2015. https://github.com/stack-of-tasks/pinocchio. (Cited in pages 43 and 154.)

[Mansard 2018] Nicolas Mansard, Andrea Del Prete, Mathieu Geisert, Steve Tonneau and Olivier Stasse. *Using a Memory of Motion to Efficiently Warm-Start a Nonlinear Predictive Controller.* In International Conference on Robotics and Automation (ICRA 2018), Brisbane, Australia, May 2018. IEEE. (Cited in pages 29, 73, and 75.)

[Mayne 1966] D Mayne. *A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems.* International Journal of Control, vol. 3, page 8595, 1966. (Cited in page 104.)

[Mellinger 2011] D. Mellinger and V. Kumar. *Minimum snap trajectory generation and control for quadrotors.* In 2011 IEEE International Conference on Robotics and Automation, pages 2520–2525, May 2011. (Cited in page 14.)

[Mirabel 2016] Joseph Mirabel, Steve Tonneau, Pierre Fernbach, Anna-Kaarina Seppälä, Mylène Campana, Nicolas Mansard and Florent Lamiraux. *HPP: a new software for constrained motion planning.* In International Conference on Intelligent Robots and Systems (IROS 2016), Daejeon, South Korea, Oct 2016. (Cited in page 137.)

[Mombaur 2005] Katja Mombaur, Richard Longman, Hans Georg Bock and Johannes Schlöder. *Open-loop stable running.* Robotica, vol. 23, no. 1, pages 21–33, jan 2005. (Cited in page 89.)

[Mordatch 2012] Igor Mordatch, Emanuel Todorov and Zoran Popović. *Discovery of complex behaviors through contact-invariant optimization.* ACM Transactions on Graphics, vol. 31, no. 4, pages 1–8, jul 2012. (Cited in page 27.)

[Mordatch 2015] Igor Mordatch and Emanuel Todorov Kendall Lowrey, Galen Andrew, Zoran Popovic. *Interactive Control of Diverse Complex Characters with Neural Networks.* Nips, pages 1–8, 2015. (Cited in page 29.)

[Morisawa 2005] Mituharu Morisawa, Shuuji Kajita, Kensuke Harada and Kiyoshi Fujiwara. *Emergency Stop Algorithm for Walking Humanoid Robots.* Proc. {IEEE}/{RSJ} Intl. Conf. Intell. Robots & Systems ({IROS}), vol. 2, pages 2109–2115, 2005. (Cited in page 17.)

[Naveau 2014] M Naveau, J Carpentier, S Barthelemy, O Stasse and P Soueres. *METAPOD - Template META-PrOgramming applied to Dynamics: CoP-CoM trajectories filtering.* In IEEE/RAS International Conference on Humanoid Robot (ICHR), 2014. (Cited in pages 13, 17, and 19.)

[Naveau 2016] Maximilien Naveau. *Advanced human inspired walking strategies for humanoid robots.* Theses, Université Paul Sabatier - Toulouse III, Sep 2016. (Cited in page 18.)

[Nishiwaki 2009a] K. Nishiwaki and S. Kagami. *Online Walking Control System For Humanoids With Short Cycle Pattern Generation.* The International Journal of Robotics Research, vol. 28, no. 6, pages 729–742, 2009. (Cited in page 17.)

[Nishiwaki 2009b] K. Nishiwaki and S. Kagami. *Online Walking Control System for Humanoids with Short Cycle Pattern Generation.* The International Journal of Robotics Research, vol. 28, no. 6, pages 729–742, may 2009. (Cited in page 19.)

[Nocedal 2006] J. Nocedal and S. J. Wright. Numerical optimization. Springer, 2nd edition, 2006. (Cited in page 40.)

[Orsag 2014] M. Orsag, C. Korpela, S. Bogdan and P. Oh. *Valve turning using a dual-arm aerial manipulator.* In International Conference on Unmanned Aircraft Systems (ICUAS)„ pages 836–841, May 2014. (Cited in page 35.)

[Orthey 2013] Andreas Orthey and Olivier Stasse. *Towards Reactive Whole-Body Motion Planning in Cluttered Environments by Precomputing Feasible Motion Spaces.* In Humanoid Robots, 2013 13th IEEE-RAS International Conference on, 2013. (Cited in page 23.)

[Palunko 2012] I. Palunko, R. Fierro and P. Cruz. *Trajectory generation for swing-free maneuvers of a quadrotor with suspended payload: A dynamic programming approach.* In IEEE International Conference on Robotics and Automation (ICRA), pages 2691–2697, May 2012. (Cited in page 35.)

[Palunko 2013] I. Palunko, A. Faust, P. Cruz, L. Tapia and R. Fierro. *A reinforcement learning approach towards autonomous suspended load manipulation*

using aerial robots. In IEEE International Conference on Robotics and Automation (ICRA), pages 4896–4901, May 2013. (Cited in page 35.)

[Pan 2012] Jia Pan, Sachin Chitta and Dinesh Manocha. *FCL: A general purpose library for collision and proximity queries.* In 2012 IEEE International Conference on Robotics and Automation, ICRA 2012, pages 3859–3866, 2012. (Cited in page 138.)

[Parzen 1962] Emanuel Parzen. *On Estimation of a Probability Density Function and Mode.* Ann. Math. Statist., vol. 33, no. 3, pages 1065–1076, 09 1962. (Cited in page 134.)

[Pellegrini 2017] Etienne Pellegrini. *Multiple-shooting differential dynamic programming with applications to spacecraft trajectory optimization*, 2017. (Cited in page 20.)

[Perrin 2012] Nicolas Perrin, Olivier Stasse, Léo Baudouin, Florent Lamiraux and Eiichi Yoshida. *Fast humanoid robot collision-free footstep planning using swept volume approximations.* IEEE Transactions on Robotics, vol. 28, no. 2, pages p.427–439, Mar 2012. (Cited in page 23.)

[Posa 2016] M. Posa, M. Tobenkin and R. Tedrake. *Stability Analysis and Control of Rigid-Body Systems With Impacts and Friction.* IEEE Transactions on Automatic Control, vol. 61, no. 6, pages 1423–1437, June 2016. (Cited in page 27.)

[Pratt 2006] Jerry Pratt, J. Carff, S. Drakunov and Ambarish Goswami. *Capture Point: A Step toward Humanoid Push Recovery.* 2006 6th IEEE-RAS International Conference on Humanoid Robots, 2006. (Cited in page 12.)

[Ramos 2014] Oscar E. Ramos, Nicolas Mansard, Olivier Stasse, Jean-bernard Hayet and Philippe Soueres. *Towards reactive vision-guided walking on rough terrain: an inverse-dynamics based approach.* International Journal of Humanoid Robotics, vol. 11, no. 2, page 1441004, 2014. (Cited in page 17.)

[Ritz 2011] R. Ritz, M. Hehn, S. Lupashin and R. D'Andrea. *Quadcopter performance benchmarking using optimal control.* In IEEE/RJS International Conference on Intelligent Robots and Systems (IROS), pages 5179–5186, Sep 2011. (Cited in page 49.)

[Romano 2015] Francesco Romano, Andrea Del Prete, Nicolas Mansard and Francesco Nori. *Prioritized Optimal Control: a Hierarchical Differential Dynamic Programming approach.* In Robotics and Automation (ICRA), IEEE International Conference on, 2015. (Cited in pages 20, 21, 90, 91, 97, 103, 108, 109, 117, and 118.)

[Saab 2013] Layale Saab, Oscar E. Ramos, Nicolas Mansard, Philippe Soueres and Jean-yves Fourquet. *Dynamic Whole-Body Motion Generation under Rigid*

Contacts and other Unilateral Constraints. IEEE Transactions on Robotics, vol. 29, no. 2, pages 346–362, 2013. (Cited in page 90.)

[Sentis 2004] Luis Sentis and Oussama Khatib. *Task-oriented control of humanoid robots through prioritization.* International Journal of Humanoid Robotics, pages 1–16, 2004. (Cited in page 16.)

[Sherikov 2014] Alexander Sherikov, Dimitar Dimitrov and Pierre-brice Wieber. *Whole body motion controller with long-term balance constraints.* In Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on, 2014. (Cited in page 18.)

[Sherikov 2016] Alexander Sherikov, Dimitar Dimitrov and Pierre-brice Wieber. *Safe navigation strategies for a biped robot walking in a crowd.* In IEEE-RAS International Conference on Humanoid Robots (Humanoids), volume 2, 2016. (Cited in pages 16 and 91.)

[Sian 2004] Neo Ee Sian, K. Yokoi, S. Kajita and K. Tanie. *A framework for remote execution of whole body motions for humanoid robots.* In 4th IEEE/RAS International Conference on Humanoid Robots, 2004., volume 2, pages 608–626, Nov 2004. (Cited in page 16.)

[Siciliano 1991] Bruno Siciliano and J. J. E. Slotine. *A general framework for managing multiple tasks in highly redundant robotic systems.* In Advanced Robotics, 'Robots in Unstructured Environments', 91 ICAR, Fifth International Conference on, pages 1211–1216. IEEE, 1991. (Cited in pages 16 and 90.)

[Sreenath 2013] K. Sreenath, N. Michael and V. Kumar. *Trajectory Generation and Control of a Quadrotor with a Cable-Suspended Load – A Differentially-Flat Hybrid System.* In IEEE International Conference on Robotics and Automation (ICRA), pages 4888–4895, May 2013. (Cited in page 35.)

[Stasse 2014] Olivier Stasse, Andreas Orthey, Francesco Morsillo, Mathieu Geisert, Nicolas Mansard, Maximilien Naveau and Christian Vassallo. *Airbus/future of aircraft factory HRP-2 as universal worker proof of concept.* In International Conference on Humanoid Robotics, Madrid, Spain, Nov 2014. (Cited in page 3.)

[Stasse 2017] Olivier Stasse, Thomas Flayols, Rohan Budhiraja, Kevin Giraud-Esclasse, Justin Carpentier, Andrea Del Prete, Philippe Souères, Nicolas Mansard, Florent Lamiraux, Jean-Paul Laumond, Luca Marchionni, Hilario Tome and Francesco Ferro. *TALOS: A new humanoid research platform targeted for industrial applications.* In International Conference on Humanoid Robotics, ICHR, Birmingham 2017, Birmingham, United Kingdom, Nov 2017. (Cited in page 4.)

[Tang 2014] S. Tang, K. Sreenath and V. Kumar. *Aggressive maneuvering of a quadrotor with a cable-suspended payload.* In Robotics: Science and Systems, Workshop on Women in Robotics, Jul 2014. (Cited in page 35.)

[Tassa 2012] Yuval Tassa, Tom Erez and Emanuel Todorov. *Synthesis and stabilization of complex behaviors through online trajectory optimization.* In Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on, pages 4906–4913, 2012. (Cited in pages 19, 20, 27, 37, 89, 102, 103, and 104.)

[Tassa 2014] Yuval Tassa, Nicolas Mansard and Emanuel Todorov. *Control-Limited Differential Dynamic Programming.* 2014. (Cited in page 20.)

[Tazaki 2014] Y Tazaki and T Suzuki. *Constraint-Based Prioritized Trajectory Planning for Multibody Systems.* IEEE Transactions on Robotics, vol. 30, no. 5, pages 1227–1234, 2014. (Cited in pages 91 and 119.)

[Tedrake 2009] Russ Tedrake. *Underactuated Robotics: Algorithms for Walking, Running, Swimming, Flying, and Manipulation.* Course Notes for MIT 6.832), 2009. Downloaded on 10/10/2017 from http://underactuated.mit.edu/. (Cited in page 9.)

[Thomas 2013] J. Thomas, J. Polin, K. Sreenath and V. Kumar. *Avian-Inspired Grasping for Quadrotor Micro UAVs.* In ASME International Design Engineering Technical Conference (IDETC), Aug 2013. (Cited in page 35.)

[Todorov 2005] Emanuel Todorov and Weiwei Li. *A generalized iterative LQG method for locally-optimal feedback control of constrained nonlinear stochastic systems.* In Proceedings of the 2005, American Control Conference, 2005. (Cited in page 104.)

[Todorov 2011] Emanuel Todorov. *A convex, smooth and invertible contact model for trajectory optimization.* In Robotics and Automation (ICRA), 2011 IEEE, 2011. (Cited in page 27.)

[Todorov 2012] Emanuel Todorov, Tom Erez and Yuval Tassa. *MuJoCo: A physics engine for model-based control.* In Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on, 2012. (Cited in page 109.)

[Todorov 2014] Emanuel Todorov. *Analytically-invertible dynamics with contacts and constraints: Theory and implementation in MuJoCo.* homes.cs.washington.edu, 2014. (Cited in page 21.)

[Tonneau 2015] S Tonneau, N Mansard, C Park, D Manocha, F Multon and J Pettr. *A Reachability-based planner for sequences of acyclic contacts in cluttered environments.* In ISRR, 2015. (Cited in pages 23, 24, 123, 124, and 145.)

[Tonneau 2016] Steve Tonneau, Andrea Del Prete, Julien Pettre, Chonhyon Park, Dinesh Manocha and Nicolas Mansard. *A fast and efficient acyclic contact planner for multiped robots.* International Journal of Robotics Research (under review), 2016. (Cited in pages 22, 23, and 25.)

[Von Stryk 1993] Dipl Math Oskar Von Stryk. *Numerical solution of optimal control problems by direct collocation.* In Optimal Control, pages 129–143. Springer, 1993. (Cited in page 89.)

[Wang 2012] Jing Wang, Islam Boussaada, Arben Cela, Hugues Mounier and Silviu-Iulian Niculescu. *Analysis and control of quadrotor via a Normal Form approach.* no. 206, 2012. (Cited in page 14.)

[Wieber 2002] Pierre-Brice Wieber. *On the stability of walking systems.* In Proceedings of the International Workshop on Humanoid and Human Friendly Robotics, pages 1–7, 2002. (Cited in page 12.)

[Wieber 2008] Pierre-Brice Wieber. *Viability and predictive control for safe locomotion.* Intelligent Robots and Systems, 2008. IROS 2008. . . . , 2008. (Cited in pages 13 and 125.)

[Wieber 2017] Pierre-Brice Wieber, Adrien Escande, Dimitar Dimitrov and Alexander Sherikov. *Geometric and numerical aspects of redundancy.* In Geometric and Numerical Foundations of Movements. JPL, 2017. (Cited in page 16.)

[Wright 1999] SJ Wright and J Nocedal. Numerical optimization. Springer Science, 1999. (Cited in pages 96, 105, and 120.)

[Xie 2017] Z. Xie, C. K. Liu and K. Hauser. *Differential dynamic programming with nonlinear constraints.* In 2017 IEEE International Conference on Robotics and Automation (ICRA), pages 695–702, May 2017. (Cited in page 20.)

[Yang 2016] Yiming Yang, Vladimir Ivan, Zhibin Li, Maurice Fallon and Sethu Vijayakumar. *iDRM: Humanoid Motion Planning with Real-Time End-Pose Selection in Complex Environments.* 2016. (Cited in page 126.)

[Zameroski 2008] D. Zameroski, G. Starr, J. Wood and R. Lumia. *Rapid swing-free transport of nonlinear payloads using dynamic programming.* In Journal of Dynamic Systems, Measurement and Control, volume 130, page 041001, 2008. (Cited in page 35.)

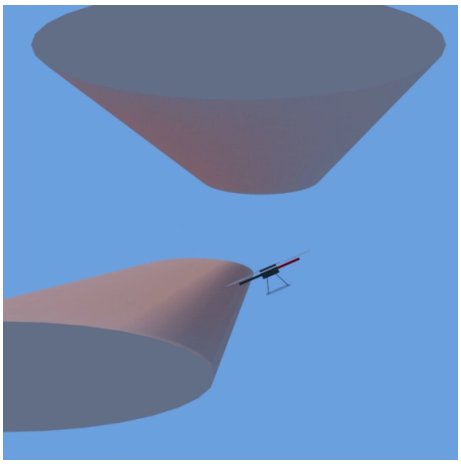
[Zaytsev 2015] Petr Zaytsev. *Using Controllability Of Simple Models To Generate Maximally Robust Walking-Robot Controllers.* PhD thesis, Cornell University, 2015. (Cited in page 22.)

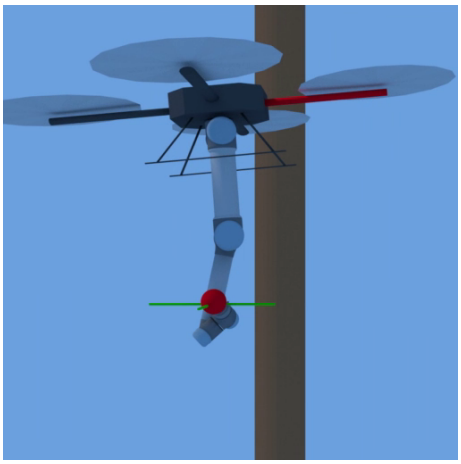
[Zhao 1994] Jianmin Zhao and Norman I. Badler. *Inverse Kinematics Positioning Using Nonlinear Programming for Highly Articulated Figures.* ACM Trans. Graph., vol. 13, no. 4, pages 313–336, oct 1994. (Cited in page 16.)

[Zucker 2010] M. Zucker, J. Andrew Bagnell, C. G. Atkeson and J. Kuffner. *An optimization approach to rough terrain locomotion.* In 2010 IEEE International Conference on Robotics and Automation, pages 3589–3595, May 2010. (Cited in pages 22, 24, and 126.)

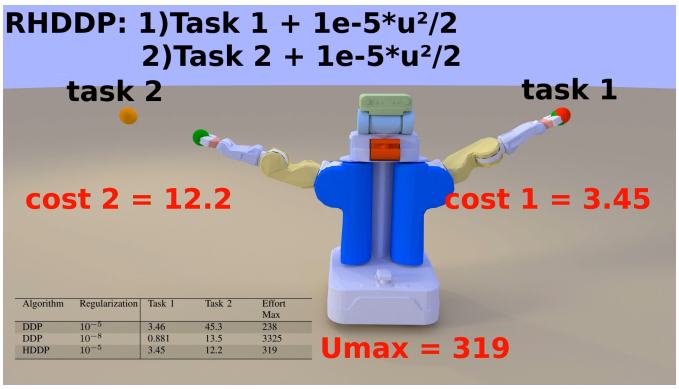
Résumé: Depuis plusieurs années, la robotique s'est fortement développée dans l'industrie et les transports. Ces robots se limitent bien souvent à des robots solidement fixés au sol ou à des robots mobiles mais à roues, ce qui limite grandement leur capacité à évoluer dans des environnements industriels complexes. Dans cette thèse, nous explorerons des méthodes permettant à des robots à mobilités plus avancées (robots aériens ou à pattes) d'évoluer et d'agir dans de tels environnements. En commençant par identifier les difficultés liées à la locomotion des robots humanoïdes, cette thèse montre comment le contrôle optimal et l'apprentissage automatique peuvent être developpés et utilisés de manière polyvalente et versatile pour effectuer des tâches complexes tel que la saisie, la dépose et la manipulation d'objet ou de charge, la réalisation de manoeuvres avec une dynamique élevée ou le déplacement en environnement encombré et/ou irrégulier. Cette thèse se divise ensuite en 3 parties distinctes.

La première partie s'intéresse à l'application du contrôle optimal dit "direct" (c'est-à-dire par résolution d'un problème d'optimisation statique, discrétisé, approximant le problème de contrôle initial) pour contrôler des robots aeriens. Nous montrons ici que le contrôle optimal direct permet de générer des trajectoires complexes et s'adapte facilement à differents systèmes dynamiques (véhicule aérien sans ou avec charge ballante, manipulateur aérien) ou tâches. Les capacités de l'algorithme sont alors utilisées pour construire un système de téléopération permettant à un opérateur de contrôler en toute sûreté et en temps réel, un véhicule aérien évoluant dans un environnement encombré. Parmi les limites connues de cette approche, il y a la facilité qu'a ce type d'algorithme de contrôle à être piégé dans des minima locaux ainsi qu'un temps de calcul relativement long. Nous montrons alors qu'un algorithme d'apprentissage automatique peut être couplé au contrôle optimal afin d'initialiser efficacement le problème et ainsi d'éviter les minimums locaux et d'améliorer les temps de calcul.

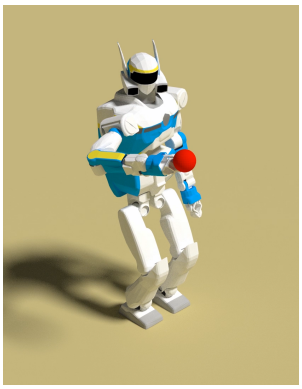




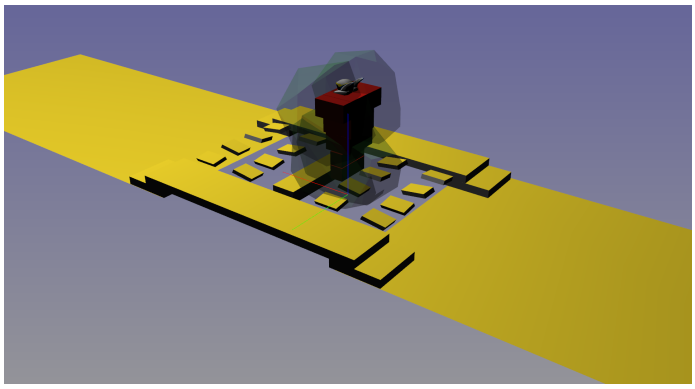
La seconde partie s'intéresse à la gestion de la redondance des robots. Les robots humanoïdes ont un grand nombre de degrés de liberté qui peuvent être utilisés pour résoudre simultanément plusieurs tâches. L'utilisation de "Hiérarchie de Tâches" est déjà courante sur les robots humanoïdes, mais se limite pour l'instant à des contrôleurs fondés sur une linéarisation instantanée. L'utilisation de hiérarchies pour l'optimisation de trajectoire a été peu étudiée. Cette partie s'intéresse donc à développer un nouveau type d'algorithme de contrôle optimal permettant de gèrer de manière efficace une hiérarchie de tâches mais aussi de gérer le compromis entre efficacité des tâches et puissance des commandes. Nous montrons alors que ce nouvel algorithme permet de faire émerger des comportements qui n'étaient pas possibles d'obtenir avec les algoritmes précédents et donc de correctement résoudre des hiérarchies avec des tâches séquentielles et/ou des tâches avec des fonctions de coût intégrales.

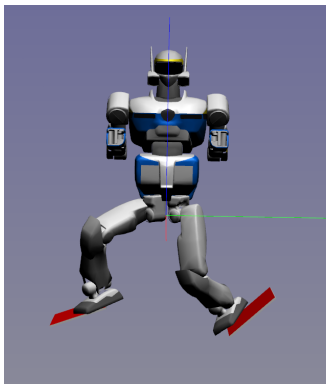


Algorithm	Regularization	Task 1	Task 2	Effort Max
DDP	10^{-5}	3.46	45.3	238
DDP	10^{-8}	0.881	13.5	3325
HDDP	10^{-5}	3.45	12.2	319

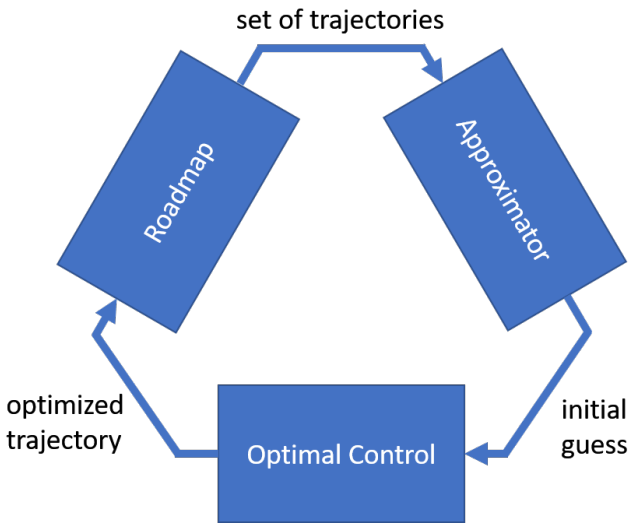


La troisième partie s'intéresse à la plannification de mouvement sur terrains irréguliers pour robots à pattes. Contrairement à la marche de quadrupèdes ou à la locomotion utilisant tous les membres, le critère de stabilité de la marche bipède contraint fortement les trajectoires possibles. Utiliser l'inverse cinématique pour vérifier la faisabilité d'un contact et calculer la stabilité du système sont des calculs coûteux qui ne peuvent pas directement être utilisés pour analyser tout l'espace. Nous utilisons ici les techniques d'apprentissage afin d'entrainer hors ligne un oracle permettant de prédire rapidement les capacités d'équilibre du robot. Cet oracle peut alors être utilisé pour analyser rapidement le terrain et sélectionner le chemin permettant de le franchir le plus facilement. On montre alors comment cet oracle peut être utilisé pour étendre les propriétés dites "d'atteignabilité" utilisés dans les plannificateurs de contact de l'équipe Gepetto.





Plus généralement, cette thèse montre que les approches basées modèles (contrôle optimal et plannification de mouvement) et les approches basées données (apprentissage automatique) peuvent être combinées afin de profiter des avantages de chacune des approches. L'utilisation de l'apprentissage automatique pour construire la solution initiale du contrôleur optimal permet gagner du temps de calcul par rapport à une approche uniquement basée modèles, mais permet aussi d'avoir de meilleurs résultats (optimalité et respect des contraintes) par rapport à une approche uniquement basée données. L'utilisation de l'apprentissage permet aussi de construire des modèles réduits des robots, et ainsi de gagner du temps de calcul avec les approches basées modèles, tout en gardant un indicateur sur la probabilité de faisabilité pour le modèle complet. De plus, l'utilisation des approches basées modèles peuvent être utilisées pour générer la base donnée utilisé par les approches basées données mais aussi les aider à apprendre plus facilement en les guidant. Au final, nous pouvons construire des algorithmes où ces deux differentes approches sont complètement entrelacées et ainsi obtenir des controleurs à la fois simples à mettre en place et rapides.



Résumé: Quelle sont les points communs entre un robot humanoïde et un quadrimoteur ? Et bien, pas grand-chose... Cette thèse s'intéresse donc au développement d'algorithmes permettant de contrôler un robot de manière dynamique tout en restant générique par rapport au model du robot et à la tâche que l'on cherche à résoudre. Le contrôle optimal numérique est pour cela un bon candidat. Cependant il souffre de plusieurs difficultés comme un nombre important de paramètres à ajuster et des temps de calcul relativement élevés. Cette thèse présente alors plusieurs améliorations permettant dâĂŹatténuer ces difficultés. D'un côté, l'ordonnancement des différentes tâches sous la forme d'une hiérarchie et sa résolution avec un algorithme adapté permet de réduire le nombre de paramètres à ajuster. D'un autre côté, l'utilisation de l'apprentissage automatique afin d'initialiser l'algorithme d'optimisation ou de générer un modèle simplifié du robot permet de fortement diminuer les temps de calcul.

Mots clés : Contrôle Optimal, Contrôle Hiérarchique, Apprentissage Automatique, Plannification de Mouvement, Robots Humanoïdes, Robots Aériens.

Abstract: What are the common characteristics of humanoid robots and quadrotors? Well, not many... Therefore, this document is focused on the development of algorithms allowing to dynamically control a robot while staying generic with respect to the model of the robot and the task that needs to be solved. Numerical optimal control is good candidate to achieve such objective. However, it suffers from several difficulties such as a high number of parameters to tune and a relatively important computation time. This document presents several ameliorations allowing to reduce these problems. On one hand, the tasks can be ordered according to a hierarchy and solved with an appropriate algorithm to lower the number of parameters to tune. On the other hand, machine learning can be used to initialize the optimization solver or to generate a simplified model of the robot, and therefore can be used to decrease the computation time.

Keywords: Optimal Control, Hierarchical Control, Machine Learning, Motion Planning, Humanoid Robots, Aerial Robots.
