Optimisation du séquencement de tâches avec lissage du mouvement dans la réalisation de missions autonomes ou collaboratives d'un humanoïde virtuel ou robotique

François Keith

To cite this version:

HAL Id: tel-00798831

https://theses.hal.science/tel-00798831

Submitted on 10 Mar 2013

UNIVERSITY MONTPELLIER 2
SCIENCES ET TECHNIQUES DU LANGUEDOC

THESIS
to obtain the title of

PHD OF UNIVERSITY MONTPELLIER 2
Discipline : SYAM – Systèmes Automatiques et Microélectroniques
Graduate School : I2S – Information, Structures, Systèmes

presented by

François KEITH

Teams : LABORATOIRE D'INFORMATIQUE, DE ROBOTIQUE
ET DE MICROÉLECTRONIQUE DE MONTPELLIER
CNRS-AIST JRL, UMI3218/CRT (TSUKUBA, JAPON)

Title :

Optimization of motion overlapping for task sequencing

Defended December 10, 2010

JURY

Christine	CHEVALLEREAU	Reviewers
Micheal	BEETZ	
André	CROSNIER	Examinators
Pierre-Brice	WIEBER	
Abderrahmane	KHEDDAR	Director
Nicolas	MANSARD	Co-director

Remerciements

Avant tout, je voudrais remercier Abderrahmane Kheddar et Nicolas Mansard, mes directeur et co-directeur de thèse, pour la qualité de l'encadrement dont j'ai bénéficié. J'ai toujours trouvé une oreille attentive à mes questions, et je les remercie pour tous les conseils, leçons et explications qu'ils m'ont donnés, et pour les discussions dont je ressortais plus motivés que jamais. Merci à Abderrahmane Kheddar pour m'avoir donné la chance d'étudier au Japon en stage de fin d'études et pour m'avoir proposé de poursuivre en thèse. Merci à Nicolas Mansard pour son enthousiasme et sa ténacité contagieuses, et pour m'avoir appris à "ne rien lâcher". Plus que tout, je les remercie vivement pour toute la confiance qu'ils m'ont accordée.

Je souhaiterais aussi remercier les membres du jury, pour avoir accepté d'examiner ce travail : merci à Christine Chevelereau et à Michael Beetz pour leurs rapports sur le manuscrit final, et merci à André Crosnier et à Pierre-Brice Wieber pour leurs retours à mi-thèse et en fin de thèse.

Je pense que cette thèse n'aurait pas été une aussi bonne expérience dans un laboratoire autre que le JRL, et je réalise la chance que j'ai eue de pouvoir travailler dans une ambiance aussi motivante et agréable. Pour cela, il me faut remercier également Kazuhito Yokoi, Eiichi Yoshida, ainsi que Tokiko Uga, pour son aide plus que précieuse dans les démarches du quotidien. Merci à tous les membres du JRL, qui étaient autant des collègues que des amis, et dont l'apport aussi bien sur un plan professionnel que personnel m'a permis de traverser ces trois années de thèse.

Merci à Paul, Amélie et Sovan pour avoir été à l'origine de cette thèse, en me poussant à faire mon stage de fin d'études au Japon. Leur soutien, leur écoute et leur amitié depuis l'école jusqu'à ma soutenance m'ont été très précieux, pour ne pas dire indispensable, pendant ces trois années et demie.

Merci à Olivier Stasse pour tous ses conseils, qui m'ont permis de grandement progresser pendant ces 3 ans, d'un point de vue technique comme d'un point de vue humain. Merci à Sylvain Miossec et à Pierre-Brice Wieber pour leurs apports et leurs explications sur l'optimisation et l'étude de la continuité, ainsi que pour leur pédagogie et leur commun soucis du détail.

Merci à Claire pour ses nombreux conseils et anecdotes sur la fin de thèse et l'après-thèse qui légitimaient toute pause café, ainsi que pour son aide sur la vision. Merci à Sébastien pour son éternelle bonne humeur et pour le réseau. Merci à Adrien et Jean-Rémy, mes premiers voisins, qui m'ont fait profiter de leur expérience du Japon avant de partir (trop vite) vers d'autres horizons. Merci à Toréa, compagnon de promo, pour avoir partagé les mêmes périodes de stress que moi, mais surtout pour nos nombreuses références communes. Merci à Anthony, Wael, Mehdi, Karim, Andrei, Nicolas P., Antoine, Pierre, Francois B. et Benjamin. Merci à Marie Avril pour le travail remarquable qu'elle a réalisé durant son stage.

Du côté de Toulouse, je souhaiterais remercier Jean-Paul Laumond pour m'avoir accueilli au LAAS pendant 6 mois, ainsi que toute l'équipe Gepetto. En particulier, je remercie le trio Sovan, Layale, Thomas, mais aussi Séverin, qui a grandement facilité mon séjour à Toulouse. Merci au personnel administratif du LIRMM qui m'a toujours été d'une aide précieuse dans la gestion des formalités à distance.

Merci à tous ceux qui ont relu mon manuscrit et m'ont aidé à l'améliorer : Claire, Paul, Pierre-Brice, Sébastien Lengagne, Sébastien Druon et Sovan. Merci à Nicolas pour les nombreuses répétitions de la présentation de thèse.

Il me faut aussi remercier ceux qui ont rendu mon expatriation plus facile, voire m'en ont fait profiter. Merci à tous ceux qui m'ont fait découvrir le Japon : mes collègues du JRL bien sûr, mais aussi Diane, Audrey, François M., Armanda, Mitsuharu et Arisumi-san. Merci aussi à mes amis restés en France qui, malgré la distance, ont toujours gardé le contact et m'ont accueilli les bras ouverts lors de mes brefs retours au pays. Un énorme merci à ceux qui ont franchi cette distance pour venir me voir.

Enfin, je souhaite remercier ma famille qui a toujours été présente et aimante pendant toutes les années qui ont précédé ma thèse, et qui encore une fois m'a soutenu, encouragé et attendu pendant ces trois ans. Je ne serais pas allé bien loin sans toute l'affection que j'ai reçue, et je dédie ce mémoire à mes frères, mes parents et mes grands-mères.

Contents

Introduction

The context of this thesis is humanoid robotics. This type of platform brings new robotic research areas (bridging to cognitive systems, neuroscience, walking, etc.) but also leads to reconsider classical robotic fundamentals (such as planning, control or reasoning). Indeed, together with the new possibilities offered, additional constraints and problems appear when tackling anthropomorphism, balance, dynamics, under-actuation or physical human-huma-noid interaction. In addition to these fundamental researches, recent efforts are also dedicated to identify potential applications.

One of the particularities of humanoid robots is redundancy. The capabilities offered by redundancy can be illustrated by a small example, comparing two robots, a single arm and a bi-manual (e.g. humanoid) moving robot platform, waitering in a café. To conclude their meal, two customers order coffees to the robots. Their mission consists roughly in preparing the coffees and bringing them to the place where the customers are waiting to be served. Once the robot has poured the coffees in the cups, it puts each of them in their respective saucer, and put each saucer (with the filled cup on it) on the plate, takes carefully the plate and goes back to the customers.

The realization of this scenario for the single arm robot is straightforward: it has to prepare one coffee after the other. Optimizing this sequence (in the sense of the duration of the mission) consists simply in realizing each task as fast as possible.

The execution of the same sequence by a bi-manual (humanoid) robot would not be considered as optimal, since the motion capabilities of two arms robot are not exploited properly. The first straightforward consideration consists in using both arms to prepare the two coffees. The second one would consist in scheduling the tasks in a way to perform these tasks as best as possible in parallel. The optimized sequence for the humanoid robot would be to take one cup in each arm, put them under the coffee machine, fill them with coffee, place the two cups on their respective saucers, place the two saucers on the tray and take the platter to the customers.

Finally, the only optimization left consists in realizing the mission as fast as possible, without spilling the coffees or breaking the cups (e.g. by impacting them strongly on the saucers). Also, since the customers are human beings, additional constraints on motion style and speed, safety and cognitive behaviors should be considered.

This rather quite simple example illustrates classical steps for achieving a robotic mission:

1. the planning, that creates a task plan defining the dependencies between tasks,
2. the scheduling, that defines a time schedule (starting and ending time) for each task,
3. the execution, that realizes the task plan.

This work focuses on the items (2) and (3), and tries to solve the following general problem: "Given a (humanoid) robot and a mission preliminarily decomposed into a task sequence, how to optimize the motion that results from task achievements and take advantage of the robot capabilities?"

This thesis proposes tools to solve this problem and is organized as follows:

First, Chapter 1 states the problem with a more technical terminology and relates existing work in task planning and scheduling in robotics. It presents the task-function formalism (a tool that will be used all along this thesis) and details how it can bridge the planning directly to low level control. More importantly, the frame, the hypothesis and the application contexts of this work are defined.

The Chapter 2 focuses on the stack-of-tasks mechanism, that enables to realize a set of tasks organized into a hierarchy. In order to realize a sequence of tasks, it is necessary to consider operations such as the insertion and the removal of tasks in the stack. These elementary operations appear to introduce discontinuities in the control output. Ensuring the smoothness of the control output during this operations is a problem that was merely addressed in previous works that make use of the stack of tasks. Several methods to correct the discontinuities are presented and discussed, and a solution is proposed.

Then, we propose an optimization-based formalism to realize task scheduling by allowing task overlapping in Chapter 3. This method aims at finding an optimal execution of the sequence of tasks using the stack of tasks and provides a solution to the tasks scheduler (the upper level). This solution not only consists in finding the right timings for inserting or removing tasks, but also suggests a behavior regulation for each task, while ensuring that the motion verifies the constraints imposed by the sequence, the robot or the environment. Advantages and flaws of this method are discussed.

These two chapters are the main novel contributions. They are then enhanced by two main experiments involving a HRP-2 humanoid robot.

The first one is described in Chapter 4, together with the tools and frameworks used. It corresponds to an experimental validation of the methods proposed to realize a smooth and optimized motion.

The second one focuses on the flexibility offered by the task-based approach and proposes to use it to tune an adaptive behavior for the task. It is indeed necessary to deal with uncertainties of the environment and correct the sequence during its realization, but it is also important to personalize robot behavior so as to conform human expectations. Therefore, in Chapter 5, the task tuning is used as an approach to modify the *way* a mission is realized in order to satisfy the constraints of the environment or the preferences of a human operator. An illustrative example is proposed and detailed.

Finally, a summary of our contribution and a list of future works are proposed in the concluding chapter.

Chapter 1

Problem statement

Having a robot realize the tasks it has been designed for can be achieved by different methods, depending on the nature of the context, the application and the complexity of the tasks. For ad-hoc and complex tasks that can hardly be further decomposed, the method usually adopted is to directly compute the trajectory that will be tracked by the robot. For extreme motions, such trajectories can be computed using an optimization-based method, as for the kicking motions and the throwing motions presented in [66, 55], or by copying and adapting an observed trajectory, as for the “aizu-bandaisan odori" dance that has been reproduced by the HRP-2 humanoid robot based on the performance of a human grand master [70].

For other contexts (industry, more or less structured environments, etc.), the approach generally preferred to realize a complex mission is to decompose it into three steps: planning, scheduling and execution of a task sequence.

The goal of this thesis is to use this three-step method to realize a robot mission, and to use the task as the thread guide between the steps. This chapter presents the foundations of the choices made to build our approach.

1.1 Three-step method

The three-step method is not an approach specific to robotics. In this section, we introduce the general definition of the task planning, the task scheduling and the reactive mission execution, and the classical methods used to realize them.

1.1.1 Task planning

In its general meaning, task *planning* [32] consists in building the appropriate sequence of tasks to realize a given mission. In other words, it consists in choosing in a given pool of doable tasks (called know-how) the adequate set of tasks and ordering them so as to achieve the given mission. In such planning, the task is defined as a symbol that can be translated into an action (or a set of actions) to be concretely realized. The tasks can be sorted in two categories:

- The *Primitive* tasks (or Operator), that can be executed directly.
- The *Non-primitive* tasks (or Method), more complex, that have to be decomposed into a set of subtasks so as to be executed.

A primitive task is usually defined by a set of preconditions and an effect, e.g. using STRIPS operators [30]. The non-primitive tasks are defined by a set of preconditions and a set of subtasks, that can be either organized according to a totally ordered sequence (i.e. all the dependencies between tasks are defined), or only partially ordered (i.e. only some of the dependencies between tasks are defined). The Fig. 1.1 shows a toy example of decomposition of a mission into a task sequence. The method “Prepare table" is totally ordered: the task order between its two subtasks is fixed. On the opposite, the method “Place cutlery" is partially ordered: the order of realization of the three tasks is left free and will only be decided in the task scheduling phase, or even during the execution by the agent (human or robot).

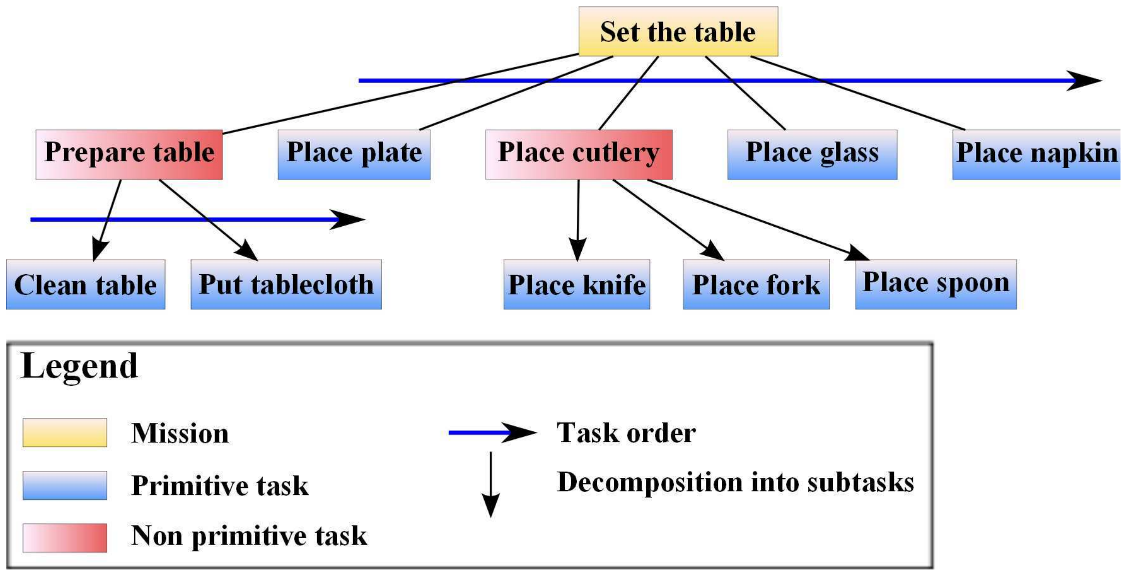


Figure 1.1: Decomposition of a toy example mission into a task plan.

Different methods can be used to define a task plan. A large majority of planners work directly with primitive tasks. They find the appropriate sequence of primitive tasks to realize a mission, given an initial state and a final state expressed as literals, and a database of primitive tasks.

More powerful, the Hierarchical Task Networks (HTN) –or their simplified version, Simple Task Networks (STN)– work with non-primitive tasks: they realize a descending approach from high level tasks down to low level ones (primitive tasks). Besides, they enable considering more constraints on the tasks compared to basic planners. Namely, post-conditions and conditions on the realization of tasks are taken into account.

The pros and cons of these methods are compared in [85]. Basic planners (such as STRIPS) do not work with time of resources. As a result, some tasks, that cannot be expressed using only formal logic (typically the maintain of a state or the respect of deadlines) cannot be formulated with this formalism. As for the HTN planners, the main issue is that they require an exhaustive definition of the know-how. A non-primitive task can be decomposed into different sets of primitive tasks, and the determination of all these combinations can be daunting. If this phase has been not be realized correctly (e.g. if a task has been omitted by the designer), the HTN planner will not be able to realize the task plan for the mission, whereas a common planner would have succeed.

When the mission is detailed enough, i.e. when the goals are well defined and the context (environment, role of external agents. . .) is known and does not present any uncertainties, it is possible to produce the sequence of tasks for the entire mission. Otherwise, only part of the task plan can be built off-line and the remaining part has to be built on-line according to the evolution of the mission.

1.1.2 Scheduling

The planning determines a doable task sequence to realize a mission, but is only concerned with the logical links between the tasks. To realize the sequence, it is necessary to disambiguate it (i.e. to determine an order of realization for partially order sub-sequences) and define the position of each task on a given time line.

1.1.2.1 Definition

In its general meaning, task *scheduling* [2, 3] is the step which comes next to task planning. It consists in determining the adequate timing for each task (or *job*) (start time, completion period and sometimes safety period) to realize the task sequence while fulfilling the constraints of availability of the resources and the temporal constraints. It often aims at minimizing the total duration of the mission, but other objectives can also be considered, such as the minimization of the cost or the respect of the deadlines (estimated by the maximal delay or the sum of the delays). Multi-objectives optimization problems can also be envisaged.

Usually, a scheduling problem is approached in two steps: the first step is a sequencing phase, and the second step is a scheduling phase (strictly speaking) [3]. The purpose of the sequencing phase is to remove the eventual ambiguities remaining after the planning (typically when the sequence is partially ordered) and make a choice on the task ordering.

One could object that the task ordering could be defined using only task scheduling. Yet, the difference between the two steps is that whereas task scheduling *implies* an order between the tasks (“Considering the given schedule, this task starts before the other one"), tasks sequencing *imposes* an order (“This task *has to* start before the other one").

Unlike planning, the scheduling phase takes into account the availability of the required resources (such as machines, computer memory, human power, raw materials, money…). Also, in the general meaning, the resources are characterized by a set of given properties: for example, they can be either renewable (they are available again at the end of the task) or non-renewable (they are consumed by the task); disjunctive (only one task can access it at a time) or conjunctive (only a limited number of tasks can access it simultaneously); preemptive (they can be interrupted and reintroduced later) or not.

A task is defined by a set of temporal data, as illustrated on Table 1.1 and Fig. 1.2. The task schedule has to fulfill two types of temporal constraints. Besides the causal dependencies between tasks imposed by the planner (often corresponding to precedence constraints), temporal constraints with respect to a specific date in the schedule (such as deadline) can be added.

r_i	**r**elease date (time at which the task is available for processing)
d_i	**d**ue date (time at which the task has to be realized)
st_i	**s**tarting **t**ime
ft_i	**f**inish **t**ime
p_i	**p**rocessing time (the period of realization of the task)
C_i	**C**ompletion time ($C_i = st_i + p_i$, if there is no interruption)
R_i	Interval of admissible **r**ank (the set of non forbidden positions for a task in an admissible sequence)

Table 1.1: Temporal data defining a task i.

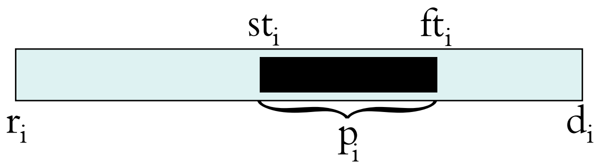


Figure 1.2: Basic definition of a task in scheduling.

A scheduling problem is usually formulated using the notation $\alpha|\beta|\gamma$, introduced in [34], where α defines the machine environment (number of machines, parallel or not), β defines the properties of the tasks (time of realization, precedence, preemption, release and due dates), and γ defines the objective of the optimization (minimization of the completion time, the lateness, the tardiness…). For example, $1|\text{prec}|C$ corresponds to the minimization of the completion time on a single machine subject to precedence constraints.

1.1.2.2 Simplification

The complexity of a scheduling problem grows drastically with the number of tasks considered for a given plan. A method reducing the complexity of scheduling problems consists in sorting the tasks by specific groups [59]. These groups decrease the number of possible combinations when organizing the tasks with respect to each others. As depicted on Fig. 1.3, three types of group of tasks are considered:

- Autonomous Tasks Sets (ATS), that are independent in term of sequencing, but they often contain too many tasks to allow a quick sequencing.
- Equal Rank Sets (EQS), that enable a quick sequencing of the tasks (they are interchangeable), but are coupled.
- Overlapped Rank-Intervals Sets (ORIS). Two tasks belong to the same set if there is a task in the set (possibly one of them) whose rank includes the rank of the two tasks.

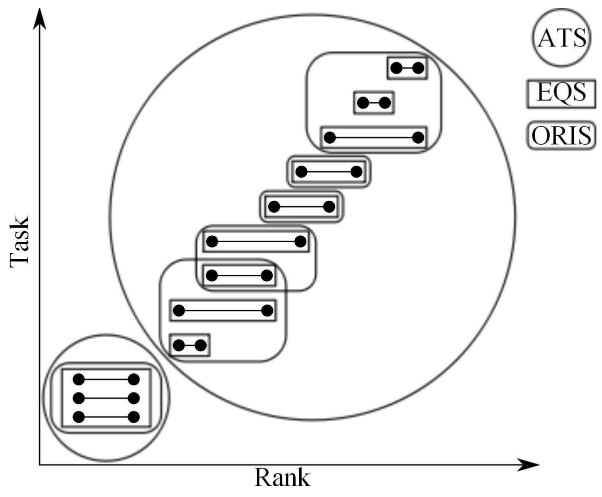


Figure 1.3: Simplification of the sequence by division into group of tasks.

1.1.3 Execution and reparation

The execution phase corresponds to the realization of the task sequence using the computed parameters. Because of the unexpected problems or events that may occur during the execution (access to a resource delayed, time of task regulation under estimated, difference between the expected and real environments, failure of an action...), it is most likely that the plan will have to be repaired, i.e. adapted in response to situation changes and execution results.

The Continuous Planning and Execution Framework (CPEF) [68] proposes a complex structure to handle the generation, execution and repair of task plan involving both human and robotic agents. The system is based on a planner manager that controls constantly the execution of the mission and repairs it when necessary.

Conservative repairs are realized to correct these failures, i.e. repairs that minimize the number of changes of the original task plan, so as to ensure a continuity in the plan. In this purpose, the HTN structure of the task planner is utilized in order to repair at the lowest level possible. The research of the failed task is realized from low level up to top level: when the node is found, the planner tries to repair it, and if this try is fruitless, the reparation process is run on the parent node.

The particularity of this planner is that it distinguishes direct and indirect execution. A *direct* execution is an execution *controlled* by the system. This is typically the case with machines, for which all the information are known (state, success of the mission...). On the opposite, an *indirect* execution is an execution only *observed* by the system. This qualifies for example actions realized by human operators, for which the informations may not be directly accessible, imprecise or delayed.

An interesting point to note at this stage is the central role of the task in the three steps: it can be seen as the guiding thread bridging the task plan to its optimal execution.

1.2 Adaptation to robotics

This section presents how some of the techniques presented below have been adapted and specified in order to be applied to robotics. Especially, the role of the task in each of the steps is highlighted.

1.2.1 Generic definition of a robotic task

In section 1.1.1, we stated that a *primitive task* can be defined as an order directly executable. We refine this definition by sorting them into two categories:

- Action tasks, that operate on the sensors (i.e. uses the sensors resources) but leave the posture of the robot unchanged (i.e. they do not use the robot's degrees-of-freedom (dof)). Such actions can be: “Take a picture", “Turn on/off the micro", “Introduce yourself", “Activate the camera"...

- Motion tasks, that operate on the motors and modify the state of the robot. This definition includes position tasks (“Turn your left foot in 45 degrees", “Go to the position $\mathbf{p}$", “Place your gripper at the position $\mathbf{p}$") but also actions depending on passive sensors (such as “Close the gripper until the force sensed exceeds n Newton").

The former category concerns any command that defines a *unique* displacement of whole or part of the robot. Such commands also use the sensory resources and sensors activities may even require motion (active perception); which means that the two categories of tasks are not necessarily exclusive. A typical task uses sensory and actuation resources; it defines not only the final state to reach/achieve, but also how to do it (in term of trajectory, speed. . .).

To put it in a generic way, these *primitive motion tasks* define a control law that can be written as follows:

$$\mathbf{A}\dot{\mathbf{q}} = \dot{\mathbf{x}} \tag{1.1}$$

where $\dot{\mathbf{q}}$ is the joint velocity vector, $\dot{\mathbf{x}}$ is the desired motion in the corresponding operational space and $\mathbf{A}$ is the inverse mapping from the joint space to operational space.

Note that the tracking of a trajectory can be directly written under this form, in which case the desired motion is the trajectory ($\dot{\mathbf{x}}(t) = \dot{\mathbf{q}}^*(t)$) and the transformation matrix is the identity:

$$\dot{\mathbf{q}} = \dot{\mathbf{q}}^*(t) \tag{1.2}$$

Two other approaches fitting this definition, the task function and the control-based approach, will be introduced later.

From now on, by *task*, we always mean primitive task (since this will be extensively used in this thesis, which is in fact the task function control component), otherwise we make clear what the task meaning is.

1.2.2 Planning

Some robotic planners are generic and follow the structures introduced earlier (classic planners and HTN). Based on a database of actions doable by the robot, they are able to build a task sequence for a given mission.

For example, Ixtex [31] is a planner based on a Partial Order Causal Link system (POCL), i.e. it can produce partially ordered plans and uses the principle of least commitment during the search (it produces plans that are also partially instantiated in order to postpone as late as possible a choice that is not mandatory). On the other hand, Icarus is a reactive symbolic planner [48, 73] based on a goal-indexed HTN. At each cycle, Icarus recognizes the environment, analyses the situations and defines the actions to realize. In this purpose, it uses two databases: a conceptual knowledge database that recognizes the situations, and a skill knowledge database, that details the operations doable by the robot.

The advantage of these planners is that the resulting task sequence still contains the causal dependency conditions between the tasks, usually defined by formal logic, which makes easier eventual plan reparations [46].

Though, when it comes to humanoid robots, the preference is often given to more specific planners [50]. Some of them are based on Rapidly-exploring Random Trees (RRT) [51], which are trees whose nodes are configurations (the root node being the initial configuration) and whose edges are the paths doable by the robot linking these configurations. At each iteration, a random configuration is generated and the closest node in the tree is found. Then, a new configuration placed between these two configurations and reachable from the node is added to the tree.

Considering that a humanoid robot has to *walk* from a place to another, without colliding with the environment, the generation of the motion is often divided in several steps. First, a trajectory is computed for a bounding volume containing the robot. Next, a feasible kinematic trajectory of the robot is deduced from it, and defines the walk pattern and the motion of the upper body. Finally this trajectory is validated and refined by a dynamical simulation [92].

Whereas the previous planner considers the objects present in the surrounding environment as obstacles that must be avoided, the contact planner presented in [24] enables defining motions where the robot takes advantage of the surrounding environment by leaning on it. The resulting plan is a sequence of statically stable postures the robot has to reach: the tasks can be considered as whole-body configurations organized sequentially.

1.2.3 Scheduling

1.2.3.1 Scheduling of actions

Scheduling problems are mainly applied for scenarios where the robot has to act autonomously, namely if it can hardly be remotely controlled by a human operator. This is typically the case with satellites [60], that have to realize some motions (correction of trajectory) and some actions (taking pictures, sending or receiving data from Earth) during specific time windows, while managing wisely their resources such as fuel or battery.

The handling of coupled tasks [82] is another class of regular scheduling problem. A coupled task is composed of two subtasks a_i and b_i separated by an incompressible delay L_i (cf. Fig. 1.4). Realizing tasks during this inactivity period enhances the schedule by removing dead times.

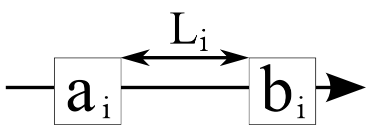


Figure 1.4: Schematic representation of a coupled task.

This problem has been applied on the control of torpedo [84]. An acquisition task can be defined as a coupled task: it sends an echo in the water (a_i), wait (during L_i) and receives it (b_i). Additionally, the torpedo has to perform data-processing tasks that analyze the received echo. The goal of the scheduling is to use the waiting time to achieve data-processing tasks, or to emit other echoes (on other frequencies). Coupled tasks also appear in problems of production and stock handling in workstations. In this context, the robots act as transporters that convey materials between the stations and the machines [10]. In this context, a coupled task can be defined as follows: a_i represents the time necessary to bring the material to the machine, L_i denotes the delay of treatment of the material by the machine, and b_i the time necessary to bring the object to another station. The goal of the scheduling is to reduce the production time.

1.2.3.2 Scheduling of motion

One could notice that in the examples of scheduling problem presented below, the scheduling is always realized between action tasks, or between action and motion tasks but never between motion tasks. This should be interrelated with the fact that so far (in this document) motion tasks correspond either to trajectories to track or to postures to reach, defining hence a whole-body motion. Besides, the order of motion tasks is usually fixed in the sequence: task scheduling is thus limited to reducing the delay between the tasks.

This approach is an important limitation to the optimization of motions, particularly in the case of redundant robots (such as humanoids) that can realize simultaneously different tasks. At this point, one may easily think on robotic sensors and actuators (degrees of freedom) as being the intrinsic resources for a robot to achieve a given mission, and consider that the tasks should not be applied on the whole body but only on part of it.

1.2.4 Execution and reparation

The reparation processes usually used in robotics are more specific and simpler than the CPFE framework mentioned earlier. Two types of corrections are usually considered: adaptations, i.e. on-the-fly modifications of the task sequence, and reparation of the task sequence (which corresponds to a reconstruction of the task plan).

For example, in order to be reactive and flexible to the variations of the environment, the controller of the Rhino robot [4] constantly verifies during the mission than the given task plan is adequate, and correct it if necessary. Using a library of routine plans, it first computes an appropriate schedule by making assumptions on the environment, then refines and eventually corrects this schedule during the execution by verifying these assumptions. This verification is realized continuously during the execution, and the adequacy (no unnecessary tasks in the sequence) and the coherency (no conflicting tasks) of the task schedule are ensured.

More recently, the Ixtex-Exec system [53, 54, 31] proposes a method to realize reactive planning and execution, and eventually plan repair. This system is layered in three parts: the planning (Ixtex), the temporal executive (Texec) and the execution module (Exec). The temporal executive handles the scheduling, by deciding the starting time of each task and eventually their stopping time (if the tasks are pre-emptive, else there is no control on them) according to the state of the system (position of the robot and available resources). The temporal executive is based on two constraint satisfaction problems: the first one is a Simple Temporal Network (STN) [19], in charge of verifying the determining the scheduling of the tasks (starting time duration), and the second one binds symbolic and numeric variable.

In the frame of humanoid robotics, lots of work have been realized to generate reactive motions for the robots. Especially, the generation and the tracking of a walk pattern is one of the main concern. We already mentioned that planning trajectories for humanoid robots is handled by specific planners. The execution of such a motion may also need to be corrected and adapted during the execution, either to enhance the stability of the default walking motion [90], or to take into account the variations caused by the walk to correct the walk pattern in order to better realize the mission, such as the tracking of a target [23].

Other works focus on the adaptation of the robot's motions to the changes in the environment. In [63], the humanoid robot realizes the grasping of a ball while walking. Using visual feedback, the robot is able to track the slow displacements of the ball and to correct the position of its gripper accordingly. Similarly, the realization of collaborative missions with an external agent, e.g. the manipulation of an object (Person-Object-Robot interaction), requires to correct the adapt the behavior of the robot according to the directions taken by the external agent. In [26], a force-based interaction system is used to determine the role and the actions of the robot (should it act as a slave, a master or should it adopt a mixed behavior?).

1.3 Binding Planning and Execution

Previous sections showed how a mission can be decomposed into a sequence of tasks, enhanced via task scheduling and finally executed by a robot. Yet, some of these techniques –even though functional– can raise an issue, because they transform the initial task sequence in order to define the motion to be executed by the robot. This conversion weakens the link between the planning and the execution, making difficult not to say impossible the consultation of the initial task plan during the execution.

In this section, we introduce two methods used to bridge this gap.

1.3.1 Gap between symbolic and numerical data

Defining the motion of the robot by an explicit or implicit trajectory is realized by converting the symbolic data used by the planner into numerical data. This transformation, if not memorized, loses knowledge in terms of semantic. While it is possible to guess what would be the trajectory associated to a set of tasks, it is most often difficult to determine what tasks led to a given trajectory (namely when the same trajectory can result from many dissimilar tasks). Using computed trajectories without any other information lacks of robustness to environment uncertainties and modifications: the original task plan is lost, which prevents any consultation during the execution. In fact, this gap appears during the transition between the planning and scheduling phases [85]: whereas the planning phase work with symbolic data, the scheduling phase works with numerical data, such as geometrical data, temporal data (time constraints, task duration) and resources.

This issue is consequent, insomuch that the workshop HYCAS[1], dedicated to this subject, has been recently created. To solve the issue raised by the gap between symbolic and

[1] International Workshop on Hybrid Control of Autonomous Systems

numerical data, a method consists in using the same element to define both the planning and the execution, making the link between them straightforward. Two approaches have been put forward in this context: the task-function approach and the control-law approach, detailed hereafter.

1.3.2 Task-function approach

The task-function approach presented in [38] offers an elegant way to bind the planning and the execution of a mission. On the one hand, it defines both the high level specification of the action expressed in formal logic causes (planning); on the other hand, it details the low level formulation allowing to compute the control (execution). Like the operational space approach [44], it has been introduced to simplify the control problem in robotics.

A task function, noted $\mathbf{e}$, is defined by three elements: a vector $\mathbf{e}$, a Jacobian $\mathbf{J}$, and a reference evolution of the task function $\dot{\mathbf{e}}^*$. Typically, the vector $\mathbf{e}$ corresponds to the error between a signal $\mathbf{s}$ and its desired value $\mathbf{s}^*$: $\mathbf{e} = \mathbf{s} - \mathbf{s}^*$. The Jacobian $\mathbf{J}$ binds the error and the vector of joint parameters $\mathbf{q}$, according to the equation $\mathbf{J} = \frac{\partial e}{\partial q}$. The reference behavior $\dot{\mathbf{e}}^*$ defines the way the error is handled. The control equation, that respects the formalism given in (1.1), writes as follows:

$$\mathbf{J}\dot{\mathbf{q}} = \dot{\mathbf{e}}^* \tag{1.3}$$

Most of the time, such commands control only part of the robot, giving the possibility to realize other tasks simultaneously when the robot presents enough redundancy. In this purpose, tasks are usually organized into a hierarchy, so that tasks of lower priority do not perturb the realization of higher priority tasks [83]. Tasks of high priority usually aim at preserving the security of the robot (balance, collision avoidance), while motion tasks have lower priority. Different methods to build such a hierarchy have been proposed in the literacy and are detailed in next chapter. The main difference between them is the trade-off they accept between the strict respect of the hierarchy and some other properties of the control law, such as the continuity. A strict hierarchy is realized by stacking the tasks one on top of each other, while a slacked hierarchy is realized by manually merging their contributions.

Recent work [73] gathers these two roles for the tasks, allowing the resulting mechanism to modify and construct reactively the task sequence plan during the execution.

1.3.3 Control-based approach

Another approach consists in working directly with control laws. This method is based on the Motion Description Language (MDL) [11] approach.

In [5, 65], an alternate way to realize the planning and execution of a mission is introduced. This method supposes that the set of possible control laws is known and that the transitions between them are smooth and known. This set can hence be formulated as a hybrid automaton where the states are the possible control laws, and the edges are the smooth transition functions between them (called interruption). Note that since it is directly expressed with control laws, this approach directly fits in the formalism given in (1.1).

The idea characterizing this approach is that a robot does not need to know everything about the surrounding environment, but only the part it can use (i.e. where it can go or act), in the same way as a train does not need to consider maritime roads. By adapting the analysis of the environment surrounding the robot to its specificities, it is possible to create a topological map detailing the places that the robot can reach using the motion defined in the set of possible control laws. The advantage of this hybrid topological map of the environment is that it associates both geometric data and the control laws.

Defining a motion comes down to finding the appropriate path in the automaton of reachable states, using the environment-based events (e.g. the approach of an obstacle) as interruptions enabling to pass from a control law to another.

Yet, the realization of task overlapping should not be realized at this level: defining the interruptions conditioning task overlapping is difficult, especially in situations where the interactions between the environment and the robot are limited (such as for a dance motion).

1.3.4 Influence of the task definition on the sequence composition

Using these two approaches, we illustrate that the *granularity* of the planning differs according to the chosen task definition. The mission is to pick an object placed on a table, and is decomposed into two motions: placing the gripper around the object and closing the gripper. In the considered environment, depicted on Fig. 1.6a, the robot can not directly reach the object since it is placed behind a large box: the only solution is to pass over the obstacle to reach the object. We consider that the best grip position chosen (manually or using an appropriate method [8]) is the same for the two approaches.

Control-based approach To define the appropriate curve trajectory that passes above the first object, we use the same task plan as the one presented in [65] (with simpler notations). The graph of possible motions is composed of three control laws (Fig. 1.5):

- $\dot{\mathbf{q}}_1$ attracts the gripper directly toward its desired posture,
- $\dot{\mathbf{q}}_2$ passes above the obstacle,
- $\dot{\mathbf{q}}_3$ realizes the grasping.

Three interruptions are considered:

- ξ_1, triggered when an obstacle is in the way,
- ξ_2, triggered when the obstacle has been bypassed,

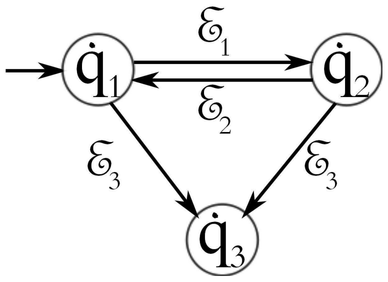

$$\begin{tikzcd}
\arrow[r] & \dot{q}_1 \arrow[rr, shift left, "\xi_1"] \arrow[dr, "\xi_3"'] & & \dot{q}_2 \arrow[ll, shift left, "\xi_2"] \arrow[dl, "\xi_3"] \\
& & \dot{q}_3 &
\end{tikzcd}$$

Figure 1.5: A hybrid automaton defining the motion of the arm of the robot.

- ξ_3, triggered when the goal is reached.

Hence, the sequence defining the mission is $(\dot{\mathbf{q}}_\mathbf{1}, \xi_1), (\dot{\mathbf{q}}_\mathbf{2}, \xi_2), (\dot{\mathbf{q}}_\mathbf{1}, \xi_3), \dot{\mathbf{q}}_\mathbf{3}$. The whole task sequence contains four primitive tasks (Fig. 1.6b) and the resulting trajectory is depicted on Fig. 1.6d.

Task-function approach The task defining the motion of the gripper aims at reducing the error between its current position and its final position. The classic definition of the control, $\dot{\mathbf{q}} = \mathbf{J}^+\dot{\mathbf{e}}^*$, ensures a maximal regulation of the error, and is thus not suitable. Indeed, this formalism makes the operational point travels a straight trajectory, causing in the present case a collision with the obstacle. A method to modify the trajectory consists in adding intermediary tasks in the sequence that act as temporary attractors or way-points. This method requires a preliminary correction of the task sequence, which is often specific to the environment or to the motion considered. For the picking motion, adding one attractor $\mathbf{e}_\mathbf{1}$ is enough. As a result, the task sequence contains three primitive tasks (Fig. 1.6c), and the final trajectory is depicted on Fig. 1.6e.

This example showed that the decomposition into primitive tasks realized by the planner depends on the task definition chosen. We could have chosen another task of higher priority that detects and locally avoids collisions. Yet such tasks are written in terms of inequalities that are only very recently being resolved.

In our developed solution, we made the choice to rely on the task function approach, which presents the advantage of keeping the task component and implicitly control only the parts (dof) of the robot that are necessary to achieve a given task or set of tasks. This other motivation is also that we adopted this formalism to control our humanoid robot HRP-2, and it proved to be elegant to handle complex collection of tasks such as grasping while walking, human-robot haptic joint action while walking, etc.

Subsequently, the robotic tasks scheduling phase is not limited to the simple reduction of the delay between the tasks anymore; our aim is to devise an approach that enables task overlapping.

1.4 Reinsertion of the scheduling

The example of the robot waiter given in introduction highlighted the main issue raised when realizing a sequence of tasks sequentially: the motion is suboptimal, or even jerky. Using this same example, we illustrate what should be the evolution of the task sequence during the planning phase and scheduler phase. The tasks realized by the robot are supposed not to conflict and are defined as follows (the indexes L (for left) and R (for right) indicate which arm realizes the motion):

$\mathbf{e_{1L-R}}$	Take the cup.
$\mathbf{e_{2L-R}}$	Put the cup in the coffee machine.
$\mathbf{e_{3L-R}}$	Fill it with coffee.
$\mathbf{e_{4L-R}}$	Place the cup on its saucer.
$\mathbf{e_{5L-R}}$	Put the saucer on the tray.
$\mathbf{e_{6}}$	Take the tray.

The partially ordered task plan corresponding to this mission is illustrated on Fig. 1.7. If the task scheduling phase is skipped, a naive task schedule would consist in realizing the tasks sequentially (Fig. 1.8). Else way, the scheduling phase should take advantage of the fact that the task sequence is only partially ordered to optimize it by realizing several tasks simultaneously. The resulting task schedule is illustrated on Fig. 1.9.

Hence, the purpose of the reintroduction of the scheduling step is to enhance the motion, typically by realizing task overlapping.

1.4.1 Unsuitability of the classic scheduling approaches

Considering that a robot can realize a hierarchy of tasks, the scheduling problem writes as follows: the jobs correspond to the robotic (primitive) tasks of the sequence and the resources are the degrees of freedom and the sensors of the robot. This type of scheduling cannot be solved by classical approaches for several reasons:

- The hierarchy of tasks introduces a non-linear dependency between the tasks and the resources.

- The time of realization of the tasks is variable (and tunable).

Graphical methods (such as Dijkstra's algorithm) are not conceivable either. The hierarchy between the tasks prevents using task functions as nodes of the graphs. Instead, it would be better to consider each possible hierarchy of tasks as nodes rather than simply the tasks, creating a graph whose size grows in a combinatorial way with the number of tasks considered. Moreover, it is impossible to plan the motion only using the graph and without considering the environment, since depending on the position of the robot and the actions to realize, some conflicts may appear.

1.4.2 Considered approach

The purpose of this work is to smooth a given task sequence as much as possible under the constraints related to task and resource dependencies, behavioral preferences, and variability dependencies. The entry of the problem is a doable sequence of tasks, where the dependencies between tasks and eventually sensors are set (the sequencing phase has been realized, together with the task planning). The sequence does not contain neither coupled tasks nor preemptive tasks, but both action and motion tasks are considered. Along with the optimization of the task schedule, an optimization of the motion is realized.

In the introduction, we mentioned that the last optimization (on the velocity) is realized after the scheduling. In practice, it is not possible to achieve successively the scheduling phase and the optimization phase: these two processes must be realized together. In the case where the scheduling phase is realized first, the final schedule may present dead times: it is sub-optimal. This issue is illustrated by Fig. 1.10. First, the task scheduling phase removes the dead times between the tasks, using the given properties of the task, especially its time of realization. Hence, each task belonging to the critical sequence of tasks is necessarily removed as soon as it is realized. Then, the optimization of the tasks finds the adequate parameterization for each task so as to realize them faster. Since the time of realization of each task is reduced, dead times are reintroduced in the previously optimized sequence.

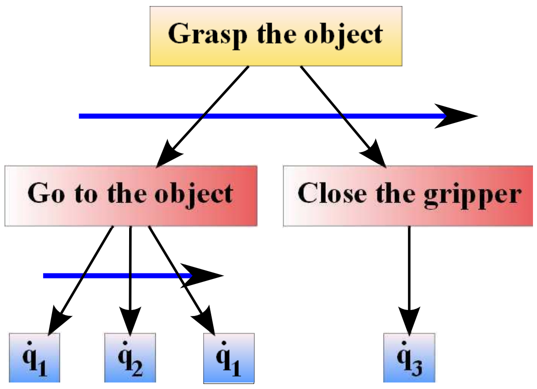
Similarly, it is not possible to start with the optimization phase and then to realize the scheduling phase either. Due to the task overlapping, the simultaneous realization of two tasks may violate conditions (such as joint velocity limits or stability) that neither of them violates if the tasks are not overlapped. Besides, we consider that only the critical sequence of tasks should be fully optimized, not all the tasks.

1.5 Conclusion

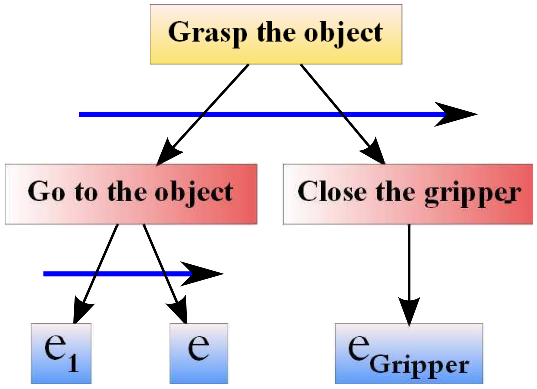
In this chapter, we briefly mentioned existing work in task planning, scheduling as used in general purpose applications and robotics. Thanks to the task function formalism introduced in robotics, it is possible to adapt the knowledge in scheduling of planed task sequence to robotics. In this thesis, we do not deal with the task planning and sequencing, which are rather considered as an input to what we are solving. We also choose to use the stack-of-tasks formalism as our basic controller, since it enables to organize a subset of task into a hierarchy (defining thus a priority for each of them) and to compute explicitly the control law that defines an implicit robot trajectory.

What is needed now is to find the schedule parameters that are the tasks timing and behavior (gain tuning). We first formulated the problem as an optimization, with a given criteria and constraints that will be described later in a dedicated chapter. After running the optimization problem we expected that non-smoothness in the task insertion and removal from the stack-of-stacks controller prohibits good convergence performance of the optimization solver. However, it appears that this problem has never been seriously considered before, even if it is fundamental because task insertion and removal on-the-fly are the basis for reactivity and task planning or sequencing adjustment to deal with unexpected situation and change in the on-line execution. Therefore, we decided to look more thoroughly toward this issue that is presented in the following chapter.

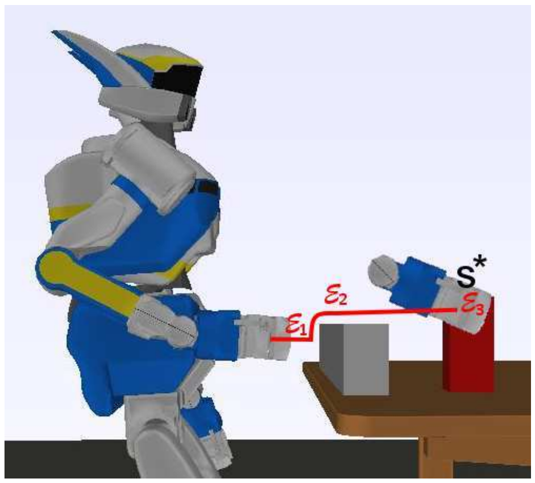
(a)



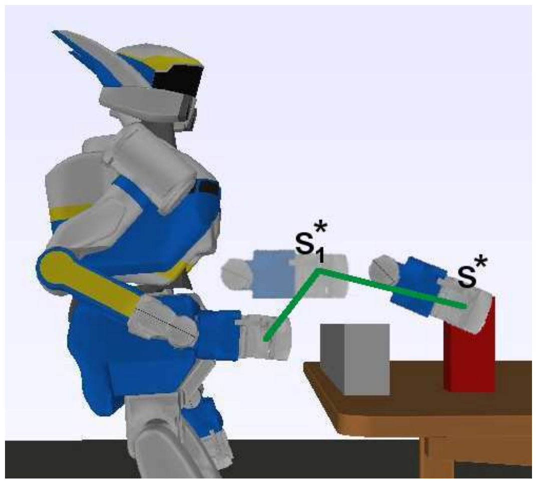
(b)



(c)



(d)



(e)

Figure 1.6: (a) Initial configuration of the environment. (b-c) Task plan for (b) the control-based approach and (c) the task function-based approach. (d) Schematic representation of the trajectory obtained with the control-based approach. (e) Schematic representation of the trajectory obtained with the task-function based approach.

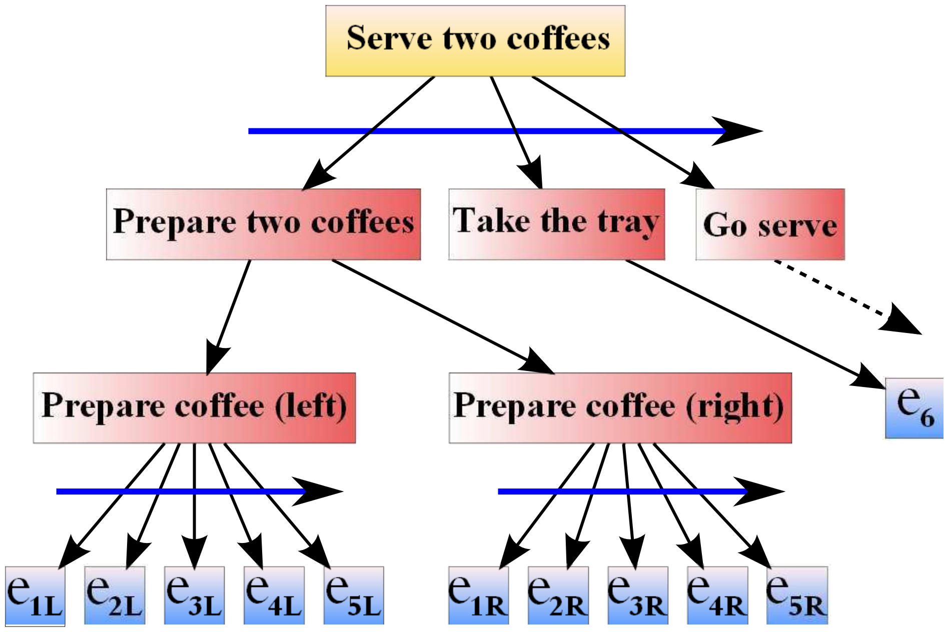


Figure 1.7: Partially ordered task sequence.

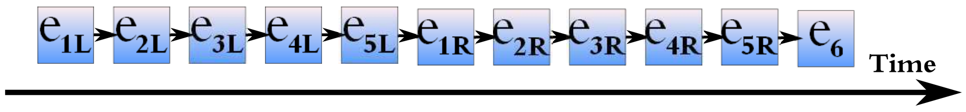

e_{1L} e_{2L} e_{3L} e_{4L} e_{5L} e_{1R} e_{2R} e_{3R} e_{4R} e_{5R} e_6
Time

Figure 1.8: Naive task scheduling.

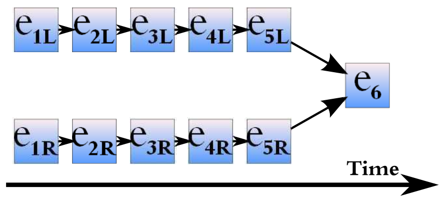

e_{1L} e_{2L} e_{3L} e_{4L} e_{5L}
e_6
e_{1R} e_{2R} e_{3R} e_{4R} e_{5R}
Time

Figure 1.9: Optimized task scheduling.

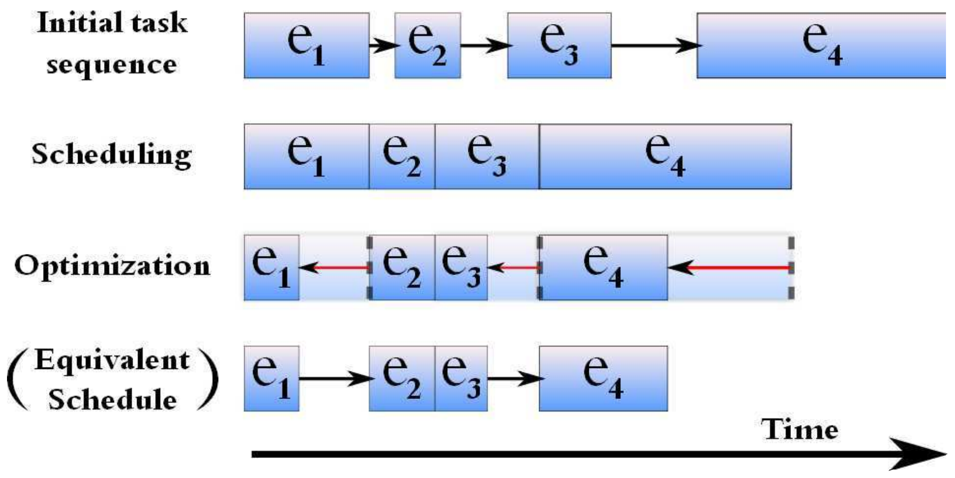


Figure 1.10: Partial optimization of the task sequence when realizing successively the scheduling and the optimization of the task. The scheduling phase removes the dead times between the tasks, then the optimization phase reduces the time of realization of some tasks. The final sequence, similarly to the first one, is sub-optimal and presents dead times.

Chapter 2

Continuity of the Stack of Tasks

The previous chapter presented the general ideas of this thesis and the main components that are needed to efficiently program a stack-of-tasks based robotic control. This chapter deals with the continuity properties of the stack of tasks. When a set of tasks of fixed number is ordered in a given priority and stacked at once, the outcome of the stack of tasks exhibits a continuous evolution in the most general case. We consider a set of operations that resemble what is generally performed on a computer data-structure stack: the insertion, the removal and the swap of tasks, that corresponds to a reordering of the priority. The influence of these operations on the continuity on the control is studied in this chapter. Different ways to formulate the stack of tasks and their continuity properties when realizing these operations are presented and analyzed.

This chapter has been realized in collaboration with Pierre-Brice Wieber, researcher at the INRIA, for the sections 2.2 and 2.4.

2.1 Introduction

2.1.1 Definition of the stack of tasks

As mentioned in the previous chapter, a task is defined by three elements: a vector $\mathbf{e}$, a Jacobian $\mathbf{J}$, and a reference evolution of the task function $\dot{\mathbf{e}}^*$. Typically, the vector $\mathbf{e}$ corresponds to the error between a signal $\mathbf{s}$ and its desired value $\mathbf{s}^*$: $\mathbf{e} = \mathbf{s} - \mathbf{s}^*$. We recall that:

$$\dot{\mathbf{e}} = \mathbf{J}\dot{\mathbf{q}} \tag{2.1}$$

The reference behavior $\dot{\mathbf{e}}^*$ defines the way the error is handled. For example, the regulation of the error can obey an exponential decrease by using

$$\dot{\mathbf{e}}^* = -\lambda \mathbf{e}, \lambda \in \mathbb{R}^+ \tag{2.2}$$

Finding the control value that gives the best regulation of a single task corresponds to solving the following minimization problem:

$$\min_{\dot{\mathbf{q}} \in \mathbb{R}^k} \|\mathbf{J}\dot{\mathbf{q}} - \dot{\mathbf{e}}^*\|^2 \tag{2.3}$$

This minimization problem can also be formulated as a least square solution:

$$\dot{\mathbf{q}} = \mathbf{J}^+\dot{\mathbf{e}}^* + \mathbf{P}\mathbf{z} \tag{2.4}$$

where $\mathbf{J}^+$ is the least squared inverse of $\mathbf{J}$, and $\mathbf{P} = \mathbf{I} - \mathbf{J}^+\mathbf{J}$ the projector into the null space of $\mathbf{J}$. While the term $\mathbf{J}^+\dot{\mathbf{e}}$ ensures the regulation of the task, the term $\mathbf{P}\mathbf{z}$ is an additional control that allows to regulate other tasks without modifying this one. If $\mathbf{z} = \mathbf{0}$, then $\dot{\mathbf{q}}$ corresponds to the least square solution.

This mechanism defines the stack of tasks. A hierarchy among the set of tasks is realized, such that each task (except the first one) is performed in the null space of higher priority ones. The resolution algorithm, detailed in [42], consists in solving for each task a minimization problem, that is

$$\min_{\dot{\mathbf{q}}_\mathbf{i} \in S_i} \|\mathbf{J}_\mathbf{i}\dot{\mathbf{q}}_\mathbf{i} - \dot{\mathbf{e}}_\mathbf{i}^*\|^2 \tag{2.5}$$

where S_i correspond to the null space left by the tasks $\mathbf{e_1}, \ldots, \mathbf{e_{i-1}}$:

$$\begin{cases} S_1 = \mathbb{R}^k \\ S_i = S_{i-1} \cap \text{null}(\mathbf{J_{i-1}}), 1 < i \leq n \end{cases} \tag{2.6}$$

Thus, the lower priority tasks do not affect the execution of higher priority ones. The control law for a task $\mathbf{e_i}$ is given in [83]

$$\begin{gathered} \dot{\mathbf{q}}_\mathbf{i} = \dot{\mathbf{q}}_\mathbf{i-1} + (\mathbf{J_i}\mathbf{P_{i-1}})^+(\dot{\mathbf{e}}_\mathbf{i}^* - \mathbf{J_i}\dot{\mathbf{q}}_\mathbf{i-1}) \\ \text{where } \mathbf{P_i} = \mathbf{I} - \begin{bmatrix} \mathbf{J_1} \\ \vdots \\ \mathbf{J_i} \end{bmatrix}^+ \begin{bmatrix} \mathbf{J_1} \\ \vdots \\ \mathbf{J_i} \end{bmatrix} \end{gathered} \tag{2.7}$$

For a stack of tasks containing n tasks, the sequence starts with $\dot{\mathbf{q}}_\mathbf{0} = \mathbf{0}$ and the control is $\dot{\mathbf{q}} = \dot{\mathbf{q}}_\mathbf{n}$; see [61] for more details.

This method is an idealistic implementation of the stack of tasks that ensures a continuous evolution of the control in nominal situations. Additional treatments and computations are needed in particular situations, e.g. kinematic singularities (due to infeasible posture) and algorithmic singularities (due to task incompatibility). In practice, using solely a pseudo-inverse is risky, and getting closer to singularities causes excessive control values. A simple 2D-robot, composed of a fixed waist, a chest and two arms, and that has to realize two incompatible position tasks for each of its arm (cf. Fig. 2.1), is used to illustrate this issue. The evolution of the control and the peaks reached when the robot is near the singularities are represented on Fig. 2.2.

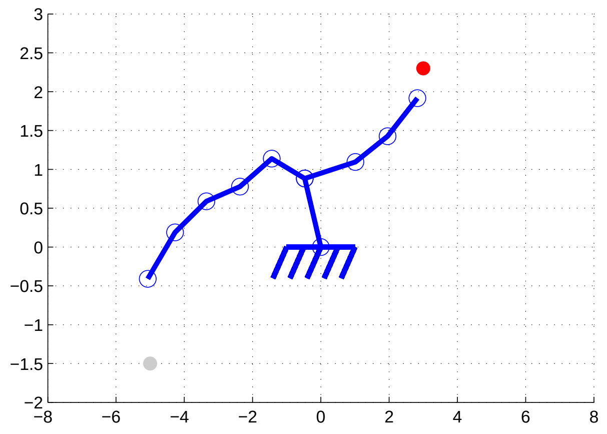


Figure 2.1: 2D planar robot simulation of two conflicting tasks.

A classic way to overcome singularities in this context is to rather use a damped inverse, defined by $\mathbf{M}^{\dagger} = (\mathbf{M} + \delta\mathbf{I})^{+}$, which limits the numerical value of the inverse matrix [69, 21]. With a damped pseudo-inverse, the equation (2.7) writes:

$$\dot{\mathbf{q}}_{\mathbf{i}} = \dot{\mathbf{q}}_{\mathbf{i-1}} + (\mathbf{J_i P_{i-1}})^{\dagger}(\dot{\mathbf{e}}_{\mathbf{i}}^{*} - \mathbf{J_i}\dot{\mathbf{q}}_{\mathbf{i-1}}) \tag{2.8}$$

This method preserves the continuous evolution of the control when crossing robot posture singularities, and achieves well in particular for *algorithmic* singularities [14], i.e. when the tasks start conflicting.

2.1.2 Event-based discontinuities

Assuming that for each task $\mathbf{e_i}$, the reference behaviour $\dot{\mathbf{e}}_{\mathbf{i}}^{*}$ is continuous and the Jacobian $\mathbf{J_i}$ has constant rank, this formulation leads to a continuous evolution of the control. The stack of tasks realizes thus a smooth control. Our aim is to ensure the continuity while realizing discrete operations such as inserting a new task (*push*), removing a task (*pull*), or pairwise task reordering, i.e. pairwise change of priority (*swap*) on-the-fly (i.e. during the execution).

Yet, the insertion and removal of a task are events liable to create a discontinuity. As an example, the additional control associated to the insertion of a task, even at the lowest priority (i.e. at the remaining null space of all the tasks present in the stack) is likely to cause a jump in the control output, since the error $\dot{\mathbf{e}}_{\mathbf{n}}$ associated to the task is most likely non null. For the removal case, ensuring that the error is regulated ($\dot{\mathbf{e}}_{\mathbf{n}} = \mathbf{0}$) prior to its removal is not a sufficient condition to ensure that $\dot{\mathbf{q}}_{\mathbf{n-1}} = \dot{\mathbf{q}}_{\mathbf{n}}$, due to the term of compensation generated by higher priority tasks (corresponding to the term $\mathbf{J_n}\dot{\mathbf{q}}_{\mathbf{n-1}}$).

Similarly, a swap operation can create a discontinuity, when the tasks swapped are conflicting. Considering two tasks $\mathbf{e_1}$ and $\mathbf{e_2}$, such that

$$(\mathbf{e_1}) \begin{cases} x = 1 \\ y = 1 \end{cases} \quad \text{and} \quad (\mathbf{e_2}) \begin{cases} x = 2 \\ z = 2 \end{cases} ,$$

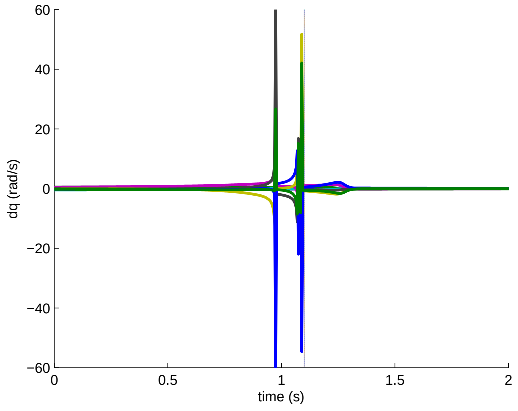


Figure 2.2: Evolution of the control using the standard method with the pseudo-inverse (2.7)*. The singularity causes discontinuities and arbitrary-large values in the control just before the discontinuity points.*

two control laws are possible: $\dot{\mathbf{q}}_{[\mathbf{1}|\mathbf{2}]} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix}$ and $\dot{\mathbf{q}}_{[\mathbf{2}|\mathbf{1}]} = \begin{bmatrix} 2 \\ 1 \\ 2 \end{bmatrix}$ where $\dot{\mathbf{q}}_{[\mathbf{A}|\mathbf{B}]}$ is the control law corresponding to the stack of tasks where $\mathbf{e}_\mathbf{A}$ has priority over $\mathbf{e}_\mathbf{B}$. Realizing an instantaneous swap results in a punctual discontinuity in the control shape.

In the following we first discuss a method which guarantees the continuity of the desired control $\dot{\mathbf{q}}$ with respect to the time t during these events, especially near singular cases. Three approaches are envisaged.

The first approach consists in considering that the *swap* is the basic operation. Hence, the *push* and *pull* operations for a given task are made respectively through propagating it from lowest to higher priority – until reaching its defined priority rank in the stack – (*push*), or reversely from its actual priority to lowest one (*pull*). Therefore, the continuity of the *push* and the *pull* operations inherits from the continuity properties of the elementary *swap* operation.

The second approach consists in considering that a *swap* operation can translate into a finite succession of *push* and *pull* operations, assuming that it is possible to push or pull a task at any priority. Hence, a method to realize the insertion and removal of a task at any priority is studied.

The third approach is based on the idea that the stack of tasks has to be considered as a seamless whole rather than by layers.

From now on, by *swap*, we always mean a change of order between two contiguous tasks in the stack.

2.2 Smooth control while swapping two tasks

In this section, different methods to swap the priority of two consecutive tasks $\mathbf{e_i}$ and $\mathbf{e_{i+1}}$ are presented. Especially, the continuity of the resulting control is studied. Each of these methods realizes operations that do not modify $S_{i+2} = S_i \cap \text{null}(\mathbf{J_i}) \cap \text{null}(\mathbf{J_{i+1}})$. By recursion, $S_{i+3}, \dots, S_n$ are also unchanged. Hence, only the method to compute the control of the tasks $\mathbf{e_i}$ and $\mathbf{e_{i+1}}$ is affected. The tasks $\mathbf{e_1}, \dots, \mathbf{e_{i-1}}$, having higher priority, are not affected by the proposed *swap*. Also, the lower priority tasks $\mathbf{e_{i+2}}, \dots, \mathbf{e_n}$ still obey (2.8) due to the invariance of their associated null space.

From now on, we consider that the stack of tasks only contains a pair of tasks $\mathbf{e_1}$, $\mathbf{e_2}$ on which a *swap* operation is realized. The 2D-robot previously introduced is used to illustrate the evolution of the control for each method.

2.2.1 Study of the continuity of the swap in the classic stack of tasks

When the two tasks conflict, the stack of tasks equation based on the damped pseudo-inverse (2.8) does not achieve a continuous and instantaneous *swap*. This discontinuity is shown in Fig. 2.3b: the swap, represented by a vertical line, occurs at $t = 1.1$s and causes a discontinuity in the control output. Realizing an instantaneous swap will result in a punctual discontinuity in the control shape.

A smoothing of the swap is required, and is realized during the time interval noted $[t_s^I, t_s^E]$.

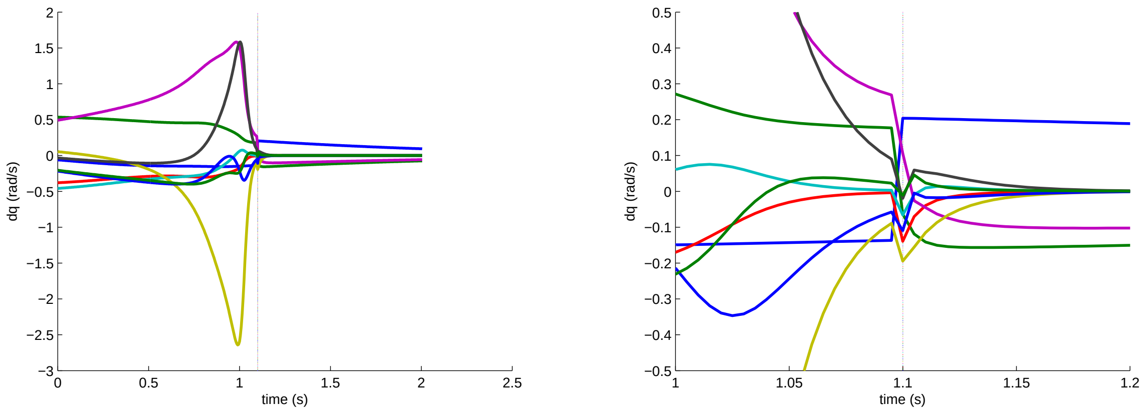


Figure 2.3: a) Evolution of the control using the damped inverse method (2.8)*. The singularities are damped so as to ensure a continuous evolution of the control except at the swap (t=1.1s). b) A closer view of the same simulation for* $t \in [1.0, 1.2]$.

2.2.2 The hierarchy of tasks as a limit of the weighing process: α-weighting

2.2.2.1 General formulation

In this method, the two tasks are temporarily placed at the same level and the swap is realized by modifying the weight of each task. During this period, the minimization problem writes:

$$\min_{\dot{\mathbf{q}}} \frac{1-\alpha}{2}\|\mathbf{J_1}\dot{\mathbf{q}} - \dot{\mathbf{e}}_{\mathbf{1}}^*\|^2 + \frac{\alpha}{2}\|\mathbf{J_2}\dot{\mathbf{q}} - \dot{\mathbf{e}}_{\mathbf{2}}^*\|^2 \tag{2.9}$$

where α is the swap coefficient depending on time[1], such that[2]:

$$\begin{cases} \alpha : [t_s^I, t_s^E] \rightarrow)0,1(\\ \lim\limits_{t\to t_s^I} \alpha(t) = 0^+ \\ \lim\limits_{t\to t_s^E} \alpha(t) = 1^- \end{cases}$$

The control law corresponding to (2.9) is defined by:

$$\begin{array}{l} \dot{\mathbf{q}} = \left(\mathbf{H}\begin{bmatrix} \mathbf{J_1} \\ \mathbf{J_2} \end{bmatrix}\right)^+ \left(\mathbf{H}\begin{bmatrix} \dot{\mathbf{e}}_{\mathbf{1}}^* \\ \dot{\mathbf{e}}_{\mathbf{2}}^* \end{bmatrix}\right) \\ \text{where } \mathbf{H} = \begin{bmatrix} (1-\alpha)\mathbf{I_1} & 0 \\ 0 & \alpha\mathbf{I_2} \end{bmatrix} \end{array} \tag{2.10}$$

This equation does not define the control law when α equals 0 (resp. 1), since the corresponding value would be $\dot{\mathbf{q}} = \mathbf{J_1^+}\dot{\mathbf{e}}_{\mathbf{1}}^*$ (resp. $\dot{\mathbf{q}} = \mathbf{J_2^+}\dot{\mathbf{e}}_{\mathbf{2}}^*$). Actually, at the limits, when α reaches 0^+, the system is equivalent to the classic model where the task $\mathbf{e_1}$ has higher priority over $\mathbf{e_2}$. When α reaches 1^-, the system is equivalent to the classic model where the task $\mathbf{e_2}$ has higher priority over $\mathbf{e_1}$.

In the case where the tasks are compatible, the control is continuous: during the swap period, it is a product of continuous functions (the matrix inversion is continuous since the rank is constant), and the continuity at the limits is ensured. The proof of the continuity at the limits is given by [86], and is based on the comparison of the solution of two minimization problems.

Proof Consider the matrices $\mathbf{A} \in \mathbb{R}^{m\times n}$ and $\mathbf{B} \in \mathbb{R}^{p\times n}$, such that $\text{rank}(B) = p$, and the vectors $\mathbf{a} \in \mathbb{R}^m$, $\mathbf{b} \in \mathbb{R}^p$ and $\mathbf{x} \in \mathbb{R}^n$.

[1] In order to simplify the notation, the dependency in t will be omitted in the next equations

[2] The notation $\lim x = a^+$ (resp a^-) means that x will reach a but will always be strictly superior (resp. inferior) to a

The first minimization problem is a Least Square Estimate problem with constraint, such as:

$$\begin{aligned} &\min_{\mathbf{x}} \|\mathbf{Bx}-\mathbf{b}\|^2 \\ &\text{subject to } \mathbf{Ax}=\mathbf{a} \end{aligned} \tag{2.11}$$

The global minimum is noted $\mathbf{x}^*$. The first order optimality condition at this point can be noted:

$$\begin{cases} \mathbf{Ax}^*-\mathbf{a}=\mathbf{0} \\ \mathbf{B^T}(\mathbf{Bx}^*-\mathbf{b})+\mathbf{A^T}\boldsymbol{\lambda}=\mathbf{0} \end{cases} \tag{2.12}$$

where $\boldsymbol{\lambda}$ represents the Lagrange multiplier. This system can be written as:

$$\begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{A} \\ \mathbf{0} & -\mathbf{I} & \mathbf{B} \\ \mathbf{A^T} & \mathbf{B^T} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{\lambda} \\ \mathbf{r_1} \\ \mathbf{x}^* \end{bmatrix} = \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{0} \end{bmatrix} \tag{2.13}$$

Then, consider the least square problem without constraints where one of the equation is balanced by the coefficient $\mu > 0$:

$$\min_{\mathbf{x}_\mu} \left\| \begin{pmatrix} \mu\mathbf{A} \\ \mathbf{B} \end{pmatrix} \mathbf{x}_\mu - \begin{pmatrix} \mu\mathbf{a} \\ \mathbf{b} \end{pmatrix} \right\|^2 \tag{2.14}$$

The global minimum is noted $\mathbf{x}_\mu^*$ and the first order optimality condition at this point is:

$$\begin{pmatrix} \mu\mathbf{A} \\ \mathbf{B} \end{pmatrix}^{\mathbf{T}} \left[\begin{pmatrix} \mu\mathbf{A} \\ \mathbf{B} \end{pmatrix} \mathbf{x}_\mu^* - \begin{pmatrix} \mu\mathbf{a} \\ \mathbf{b} \end{pmatrix} \right] = \mathbf{0} \tag{2.15}$$

This equation can be written as the following matrix product:

$$\begin{bmatrix} -\mu^{-2}\mathbf{I} & \mathbf{0} & \mathbf{A} \\ \mathbf{0} & -\mathbf{I} & \mathbf{B} \\ \mathbf{A^T} & \mathbf{B^T} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{r_1} \\ \mathbf{r_2} \\ \mathbf{x}_\mu^* \end{bmatrix} = \begin{bmatrix} \mathbf{a} \\ \mathbf{b} \\ \mathbf{0} \end{bmatrix} \tag{2.16}$$

When μ gets large, the systems (2.13) and (2.16) approach one another. As a result, the two minimization problems converge toward the same solution: $\lim_{\mu\to+\infty} \mathbf{x}_\mu^* = \mathbf{x}^*$, ensuring the continuity of $\mathbf{x}$ at the limits. ∎

2.2.2.2 Implementation

This method also faces the problem of lack of robustness near singular configurations due to the use of a pseudo-inverse. A naive adaptation for the equation (2.10) would be to damp the term relative to the Jacobians, that is:

$$\dot{\mathbf{q}} = \left(\mathbf{H} \begin{bmatrix} \mathbf{J_1} \\ \mathbf{J_2} \end{bmatrix} \right)^\dagger \left(\mathbf{H} \begin{bmatrix} \dot{\mathbf{e}}_\mathbf{1}^* \\ \dot{\mathbf{e}}_\mathbf{2}^* \end{bmatrix} \right) \tag{2.17}$$

but it still leads to a discontinuous evolution of the control even when the two tasks do not conflict as depicted by Fig. 2.4. This problem appears when the interpolation switching term α becomes negligible compared to the damping term δ characterizing the damped pseudo-inverse. As a result, the second task is shadowed and the control is closer to the one where there is only one task in the stack.

The minimization problem formulation helps explaining this observation.

$$\min_{\dot{\mathbf{q}}} \frac{1-\alpha}{2}\|\mathbf{J_1}\dot{\mathbf{q}} - \dot{\mathbf{e}}_\mathbf{1}^*\|^2 + \frac{\alpha}{2}\|\mathbf{J_2}\dot{\mathbf{q}} - \dot{\mathbf{e}}_\mathbf{2}^*\|^2 + \delta\|\dot{\mathbf{q}}\|^2 \tag{2.18}$$

where $\delta\|\dot{\mathbf{q}}\|^2$ is the term qualifying the damped pseudo-inverse. Suppose that $\alpha \ll \delta \ll 1$, the minimization problem can be formulated as:

$$\begin{aligned} \min_{\dot{\mathbf{q}}} \quad & (\beta_1\|\mathbf{J_1}\dot{\mathbf{q}} - \dot{\mathbf{e}}_\mathbf{1}^*\|^2 + \beta_2\|\dot{\mathbf{q}}\|^2 + \beta_3\|\mathbf{J_2}\dot{\mathbf{q}} - \dot{\mathbf{e}}_\mathbf{2}^*\|^2) \\ & \text{where } \beta_1 = \tfrac{1-\alpha}{2}, \beta_2 = \delta, \beta_3 = \tfrac{\alpha}{2} \\ & \text{hence } \beta_1 \gg \beta_2 \gg \beta_3 \end{aligned} \tag{2.19}$$

Intuitively, this corresponds to the control law associated to a stack containing three tasks: $\mathbf{e_1}$, $\mathbf{e}_\delta$ and $\mathbf{e_2}$, where $\mathbf{e}_\delta$ is the additional task induced by the damping, defined by a Jacobian $\mathbf{J}_\delta$ equal to the identity and a null error. Hence, the null space remaining to realize the task $\mathbf{e_2}$ is empty, preventing its regulation: the stack of tasks acts as if there was only one task, causing the discontinuity.

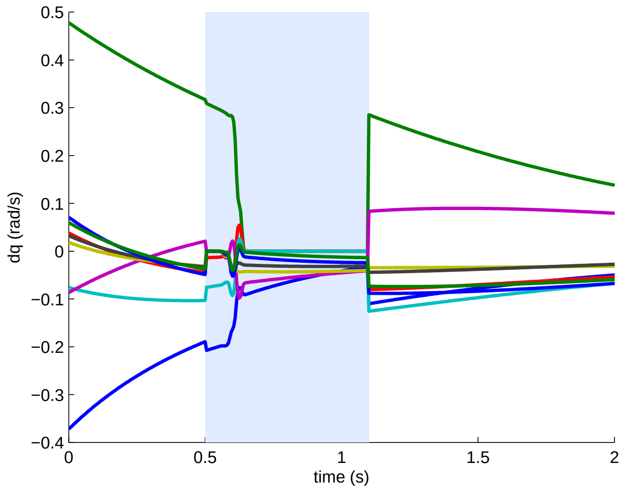


Figure 2.4: Discontinuous evolution of the control obtained with the α-weighting in a singularity-free configuration, using the naive damped inverse (method (2.17)*). The cyan zone represents the swap period.*

Note that a solution to avoid kinematic singularities consists in realizing the damping on the error, not on the Jacobians: instead of the real error ($\dot{\mathbf{e}}$), we consider the closest error that can be corrected ($\mathbf{J}\mathbf{J}^{\dagger}\dot{\mathbf{e}}$). Following this, the equation (2.10) becomes:

$$\dot{\mathbf{q}} = \left(\mathbf{H}\begin{bmatrix}\mathbf{J_1}\\ \mathbf{J_2}\end{bmatrix}\right)^{+}\left(\mathbf{H}\begin{bmatrix}\mathbf{J_1}\mathbf{J_1}^{\dagger}\dot{\mathbf{e}}_\mathbf{1}^{*}\\ \mathbf{J_2}\mathbf{J_2}^{\dagger}\dot{\mathbf{e}}_\mathbf{2}^{*}\end{bmatrix}\right) \tag{2.20}$$

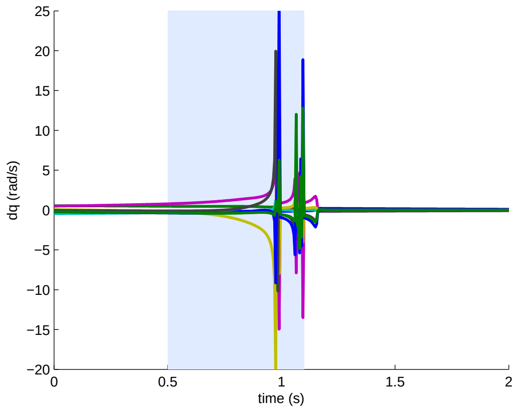


Figure 2.5: In the case of singularity, the control obtained with the α-weighting correctly damped (method (2.20)*) is discontinuous .*

2.2.2.3 Handling singularities

When the matrix $[\mathbf{J_1},\mathbf{J_2}]$ presents an algorithmic singularity (i.e. when their is a rank loss: $\mathbf{rank}([\mathbf{J_1},\mathbf{J_2}]) < \mathbf{rank}(\mathbf{J_1}) + \mathbf{rank}(\mathbf{J_2})$), the method (2.20) exhibits many discontinuities as depicted by Fig. 2.5. These discontinuities are due to the operation $\left(\mathbf{H}\begin{bmatrix}\mathbf{J_1}\\ \mathbf{J_2}\end{bmatrix}\right)^{+}\mathbf{H}$.

An ad-hoc solution would be to consider an evolutive damped inverse, such that the damping coefficient δ depends on the swap coefficient α. The minimization problem (2.18) becomes then:

$$\min_{\dot{\mathbf{q}}} \frac{1-\alpha}{2}\|\mathbf{J_1}\dot{\mathbf{q}} - \dot{\mathbf{e}}_\mathbf{1}^{*}\|^2 + \frac{\alpha}{2}\|\mathbf{J_2}\dot{\mathbf{q}} - \dot{\mathbf{e}}_\mathbf{2}^{*}\|^2 + \delta(\alpha)\|\dot{\mathbf{q}}\|^2 \tag{2.21}$$

This function $\delta(\alpha)$ must fulfil the following properties:

$$\begin{cases} \delta :]0,1[\rightarrow]0,1[\\ \lim\limits_{\alpha\to 0^+} \delta(\alpha) = \alpha \\ \lim\limits_{\alpha\to 1^-} \delta(\alpha) = 1-\alpha \end{cases} \tag{2.22}$$

In order to have an idea about the required behavior of $\delta(\alpha)$, we simulated the system (2.18) with different values of $\delta \in [10^{-15}, 10^{-1}]$. This system allows to analyze the behavior around $\alpha = 0$ and $\alpha = 1$. For $\alpha \leq 0$ (resp $1 \leq \alpha$), the control corresponds to a hierarchy of tasks where $\mathbf{e_1}$ (resp $\mathbf{e_2}$) has the priority.

$$\begin{cases} \text{if } \alpha \leq 0 & \dot{\mathbf{q}} = \dot{\mathbf{q}}_\mathbf{1} + (\mathbf{J_2 P_1})^\dagger(\dot{\mathbf{e}}_\mathbf{2}^* - \mathbf{J_2}\dot{\mathbf{q}}_\mathbf{1}) \\ & \text{where } \dot{\mathbf{q}}_\mathbf{1} = \mathbf{J_1}^\dagger \dot{\mathbf{e}}_\mathbf{1}^* \\ \text{if } 0 < \alpha < 1 & \dot{\mathbf{q}} = [\beta \mathbf{J_1^T J_1} + \alpha \mathbf{J_2^T J_2} + \delta \mathbf{I}]^{-1}[\beta \mathbf{J_1^T J_1 J_1^\dagger} \dot{\mathbf{e}}_\mathbf{1}^* + \alpha \mathbf{J_2^T J_2 J_2^\dagger} \dot{\mathbf{e}}_\mathbf{2}^*] \\ & \text{where } \beta = 1 - \alpha \\ \text{if } 1 \leq \alpha & \dot{\mathbf{q}} = \dot{\mathbf{q}}_\mathbf{2} + (\mathbf{J_1 P_1})^\dagger(\dot{\mathbf{e}}_\mathbf{1}^* - \mathbf{J_1}\dot{\mathbf{q}}_\mathbf{2}) \\ & \text{where } \dot{\mathbf{q}}_\mathbf{2} = \mathbf{J_2}^\dagger \dot{\mathbf{e}}_\mathbf{2}^* \end{cases} \tag{2.23}$$

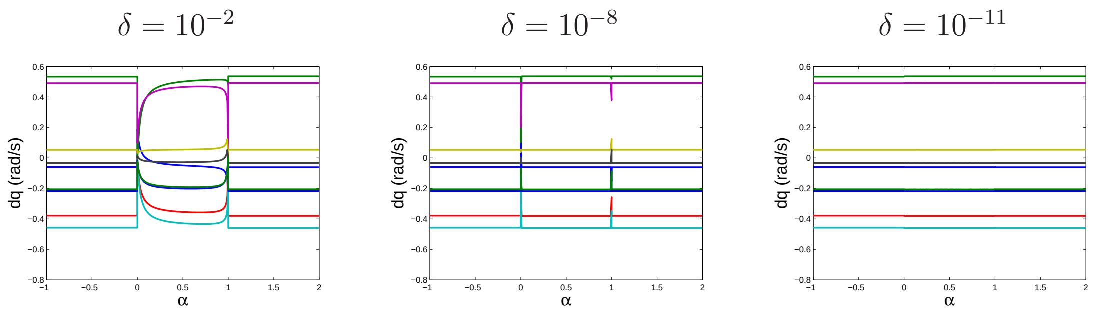


Figure 2.6: Simulation of the system (2.23) *when the matrices* $\mathbf{J_1}$, $\mathbf{J_2}$ *and* $[\mathbf{J_1}; \mathbf{J_2}]$ *are not singular, with different values of* δ.

In the singularity-free case (Fig. 2.6), a small value of δ (inferior to 10^{-8}) is required in order to ensure the continuity. The discontinuities observed in Fig. 2.6a at $t = 0^+$ (and $t = 1^-$) correspond to the case where the damped inverse interferes with the weighting process: when $\alpha \ll \delta$ (or $(1-\alpha) \ll \delta$), the damping acts as a virtual task $\mathbf{e}_\delta$ that has priority over the second task, which is not realized any more. The system acts as if there was only one task in the stack.

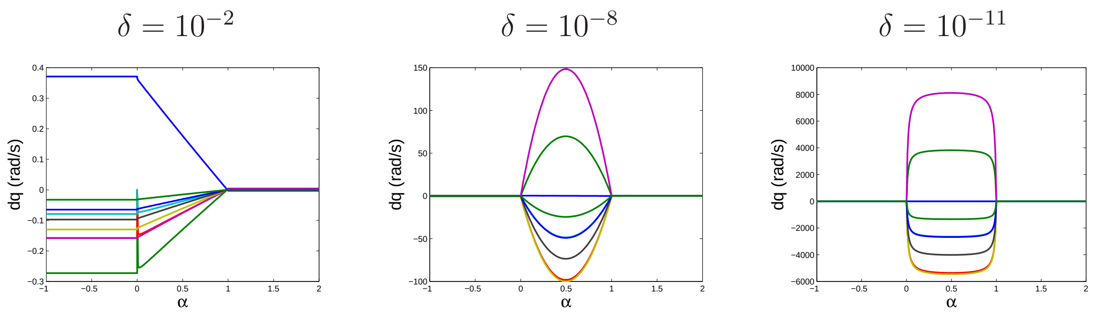


Figure 2.7: Simulation of the system (2.23) *when the matrix* $[\mathbf{J_1}; \mathbf{J_2}]$ *is singular, with different values of* δ.

In the singular case (shown on Fig. 2.7), the damping δ determines the maximum value reached by $\dot{\mathbf{q}}$. Besides, it is important to note that near $t = 0^+$ and $t = 1^-$, the smaller δ is, the higher the slope will be.

An appropriate function δ has hence to fulfil the following conditions:

1. When α is close to 0 or 1, the damping value $\delta(\alpha)$ has to be small, in order to ensure the continuity in the singularity-free case.
2. Else, the damping factor has to be high enough to limit the control values.

Using the previous results, it can be assumed that a function that equals 10^{-11} near the limits and 10^{-8} elsewhere satisfies these conditions. Yet, since low values of δ cause important slopes in the control function in the singular case, an ad-hoc function $\delta(\alpha)$ is hardly conceivable. As an example, we consider the following function:

$$\delta(x) = \frac{(\delta_{\max} - \delta_{\min})}{2} (\tanh(d(x - 0.5 + c)) \; - (\tanh(d(x - 0.5 - c)))) + \delta_{\min} \qquad (2.24)$$

where $\delta_{\min}$, $\delta_{\max}$, c and d are parameters setting the min and max values, the slope and the width of the function $\delta(x)$. We choose $c = 0.475, d = 100, \delta_{\min} = 10^{-11}$ and $\delta_{\max} = 10^{-8}$. The function shape is shown on Fig. 2.8a. The results obtained are illustrated on Fig. 2.8b and Fig. 2.8c and invalidate this method: the control obtained presents considerable variations not applicable on a real robot.

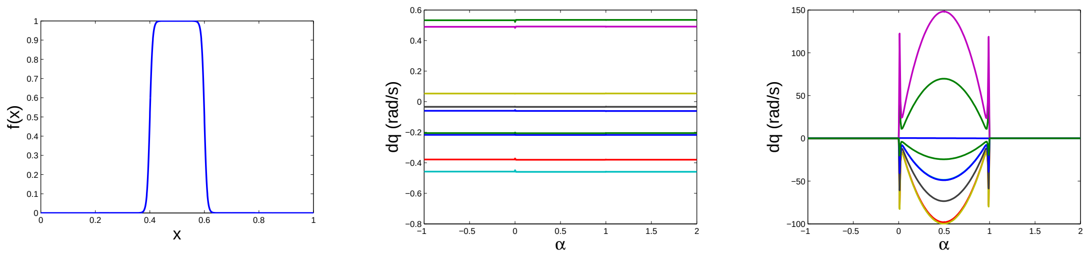


Figure 2.8: a) Evolution of the function $\delta(\alpha)$ given by (2.24) b-c) Evolution of the control during a swap using the evolutive damping method b) on two compatible tasks and c) on two incompatible tasks.

2.2.3 Linear interpolation

This method consists in realizing a linear interpolation of the two control laws $\dot{\mathbf{q}}_{[\mathbf{1}|\mathbf{2}]}$ and $\dot{\mathbf{q}}_{[\mathbf{2}|\mathbf{1}]}$

$$\dot{\mathbf{q}} = \alpha \dot{\mathbf{q}}_{[\mathbf{2}|\mathbf{1}]} + (1 - \alpha) \dot{\mathbf{q}}_{[\mathbf{1}|\mathbf{2}]} \qquad (2.25)$$

where $\dot{\mathbf{q}}_{[\mathbf{A}|\mathbf{B}]}$ is the control law corresponding to the stack of tasks where $\mathbf{e_A}$ has priority on $\mathbf{e_B}$ and α is the swap function, which is a smooth function of time.

Although it is obvious that the continuity is ensured, this method presents two major flaws:

1. There is no guarantee that the robot motion is feasible, since the control is obtained by interpolation.

2. The computation time is increased, since the control has to be computed twice for these two levels of the stack of tasks. Especially, two extra computation of the pseudo inverse $(\mathbf{JP})^\dagger$ are required. It is important to note that even in the case of a stack of tasks containing n tasks, only these two levels must be computed twice, and not the whole stack.

The Fig. 2.9 shows that the evolution of the control is continuous in singular cases.

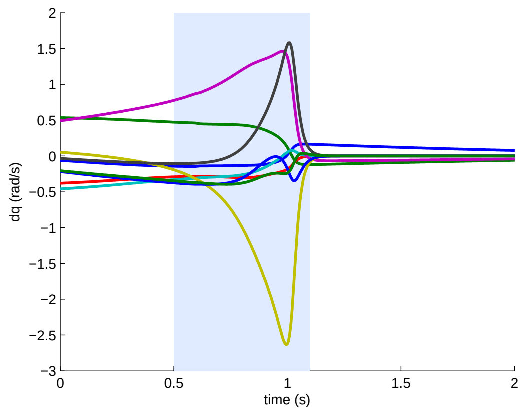


Figure 2.9: Continuous control during a swap using a linear interpolation in a singular case: the discontinuity observable on Fig. 2.3 has been smoothed.

2.2.4 Modeling the insertion/removal at the end of the stack as a swap

A hierarchy of task is usually ended by a virtual task that defines a default value in the null space of all the other tasks. Since the null space left is empty, a task placed after this virtual task has no effect on the control: it is virtually inactive. Inserting a task comes down to adding it after the virtual task and exchanging their priority. The removal of a task is the symmetric operation of the insertion: removing smoothly a task is thus realized with the same process.

This virtual task appears explicitly when computing the control law using a minimization process, but is implicit when using the pseudo-inverse approach. In this latter case, a solution to realize the smoothing is to explicitly weight the additional term corresponding to the inserted/removed task by the smoothing coefficient λ^{Ins} in the equation of the control law.

$$\dot{\mathbf{q}}_{\mathbf{n+1}} = \dot{\mathbf{q}}_{\mathbf{n}} + \lambda^{Ins}(\mathbf{J}_{\mathbf{n+1}}\mathbf{P}_{\mathbf{n}})^{+}(\dot{\mathbf{e}}^{*}_{\mathbf{n+1}} - \mathbf{J}_{\mathbf{n+1}}\dot{\mathbf{q}}_{\mathbf{n}}) \tag{2.26}$$

$\lambda^{Ins}(t)$ is defined by an appropriate smooth function of time t and fulfils the following conditions:

$$\begin{cases} \lambda^{Ins} : \mathbb{R}^+ \to [0,1] \\ \lambda^{Ins}(t_n^I) & = 0 \\ \lambda^{Ins}(t_n^F) & = 0 \\ \lambda^{Ins}(t) & = 1, \forall t \in [t_n^I + \Delta t^{Ins}, t_n^F - \Delta t^{Ins}] \end{cases}$$

where t_n^I and t_n^F are respectively the insertion and removal time of the task $\mathbf{e_n}$ from/in the stack, and Δt^{Ins} is the insertion delay during which the gain λ^{Ins} grows continuously between 0 to 1. With the classic control law, the gain λ^{Ins} is constant and equal to 1.

The slope of the control function $\dot{\mathbf{q}}(t)$ during the insertion and the removal of the task $\mathbf{e_n}$ is also determined by the gain. The gain defined in (2.27) allows to modify the slope via the coefficient d (Fig. 2.10). The insertion time is then fixed: $\Delta t^{Ins} = \frac{\pi}{d}$. However, it is important to notice that the steeper the slope is, the faster the insertion/removal is.

$$\begin{cases} \text{if} & t \leq t^I, & \lambda^{Ins}(t) = 0 \\ \text{if} & t^I \leq t \leq t^I + \Delta t^{Ins}, & \lambda^{Ins}(t) = \frac{1}{2} - \frac{1}{2}\cos(d(t - t^I)) \\ \text{if} & t^I + \Delta t^{Ins} \leq t \leq t^F - \Delta t^{Ins}, & \lambda^{Ins}(t) = 1 \\ \text{if} & t^F - \Delta t^{Ins} \leq t \leq t^F, & \lambda^{Ins}(t) = \frac{1}{2} - \frac{1}{2}\cos(d(t^F - t)) \\ \text{if} & t^F \leq t, & \lambda^{Ins}(t) = 0 \end{cases} \tag{2.27}$$

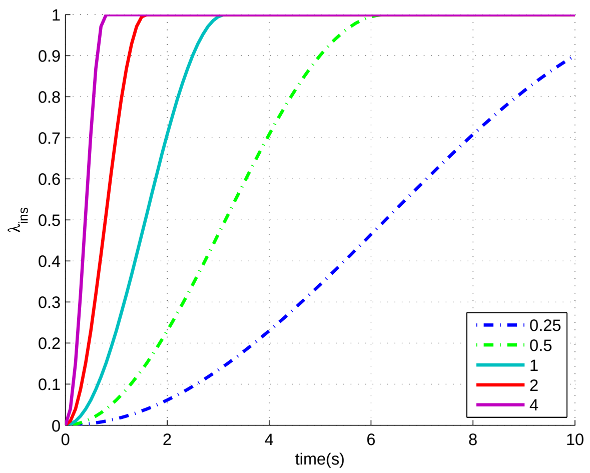
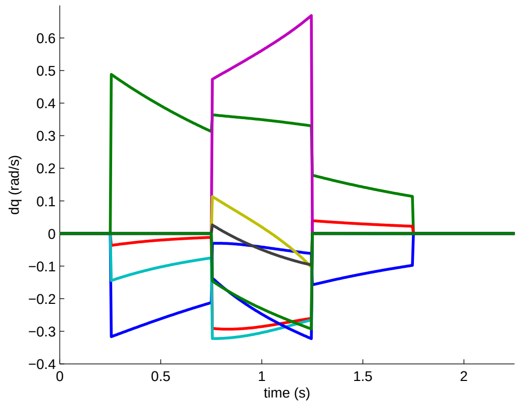
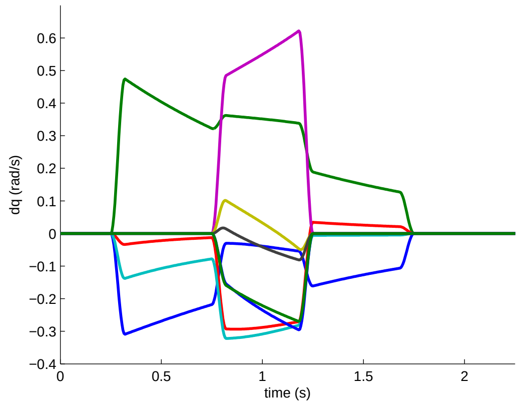


Figure 2.10: Aspect of the gain $\alpha(t)$ detailed in Eq. (2.27) for various values of d.

The Fig. 2.11a and the Fig. 2.11b illustrate the evolution of the control while inserting and removing two tasks by order of priority. The first task is inserted at 0.25s and removed at 1.75s, and the second one is inserted meanwhile at 0.75s, and then removed at 1.25s. The Fig. 2.11a illustrates the discontinuous evolution of the classic control law. The Fig. 2.11b represents the evolution of $\dot{\mathbf{q}}$ obtained with this smooth control law.



a)



b)

Figure 2.11: Evolution of $\dot{\mathbf{q}}$ while inserting and removing the tasks by order of priority with a) The classic control function b) The smooth control function.

2.2.5 Modification of the stack-of-tasks scheme

This section presented different methods to realize a swap between two consecutive tasks. The swap of two distant tasks must be brought down to a sequence of swaps of tasks with successive priority. Satisfying this condition does not ensure a both continuous and immediate control, especially due to the possible presence of algorithmic singularity. Realizing a blending operation (2.25) is thus a solution allowing to avoid discontinuities, but this operation is not instantaneous. This method introduces two delays: an insertion delay Δt^{Ins} and a swap delay Δt^{Swap}. The swap between the tasks $\mathbf{e_k}$ and $\mathbf{e_{k+1}}$ can still be instantaneous ($\Delta t^{Swap} = 0$) iff the tasks $\mathbf{e_k}, \ldots, \mathbf{e_n}$ are compatible in the null space P_{k-1}, i.e. if the tasks do not conflict; otherwise an interpolation must be realized, and Δt^{Swap} is non null.

Considering that an insertion can be assimilated to a swap process and that a task cannot be part of two swapping processes simultaneously, it is not possible to swap with a task being inserted or removed. Since some transitions are not instantaneous anymore, the task operation scheme changes from Fig. 2.12 to Fig. 2.13 (for the sake of readability and clarity, some transitions, that only correspond to a role swap between $\mathbf{e_1}$ and $\mathbf{e_2}$, are omitted). The transition period required for the insertion and removal of a task are illustrated by the addition of states. $\mathbf{e_A}(\text{Ins})$ (resp. $\mathbf{e_A}(\text{Rem})$) indicates the insertion (resp. the removal) of task $\mathbf{e_A}$ being in process. These states are quit once the insertion gain has reached the desired value (1 for the insertion, 0 for the removal).

This method ensures a continuous evolution of the control output of the stack of tasks during discrete events, but limits them to manipulations between neighbour tasks. As a consequence, a safe but time consuming method to insert a task at any level of the stack is to insert it smoothly at the end of the stack using an insertion gain, and then realize a succession of smooth swaps to put it at the desired priority.

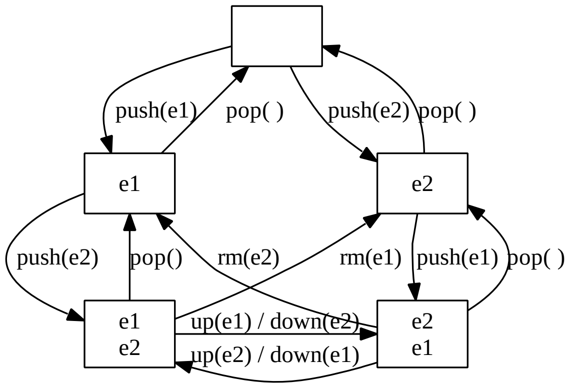


Figure 2.12: Automaton of possible operations and reachable states of the classic stack of tasks.

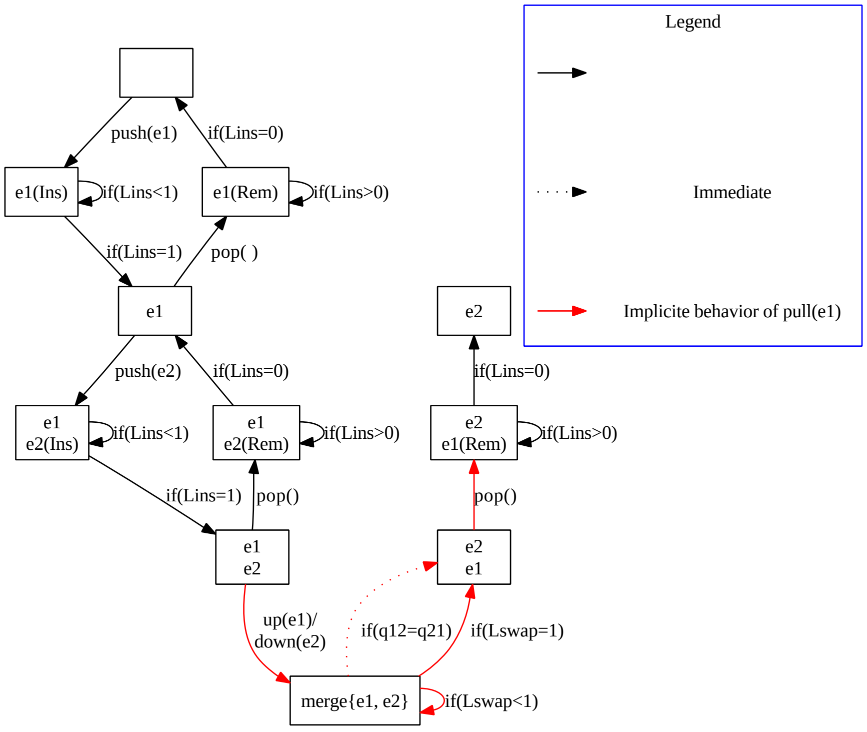


Figure 2.13: Partial automaton of reachable states of the smooth stack of task.

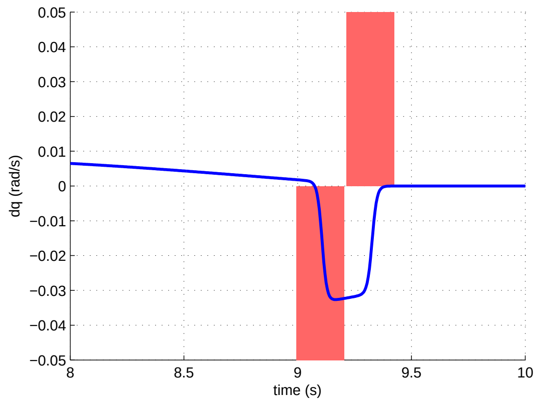
2.2.6 From a discontinuity to another

For a given schedule of events, the modification of the tasks scheme ensures a continuous evolution of $\dot{q}$ with respect to t. Each of the event operation *push*, *pull*, and *swap* (correspon-ding respectively to the insertion, removal or swap of tasks) can be realized while ensuring the continuity of the control. Now, we focus on the influence of a modification in the se-quence of events on the control function shape. Due to a small change of the sequence of events (such as an event being delayed due to some perturbation), the realization of another operation can be required in order to maintain the continuity of the control with respect to the time t. As a result, the control shape, that will obey this new sequence of tasks, can accuse huge differences.

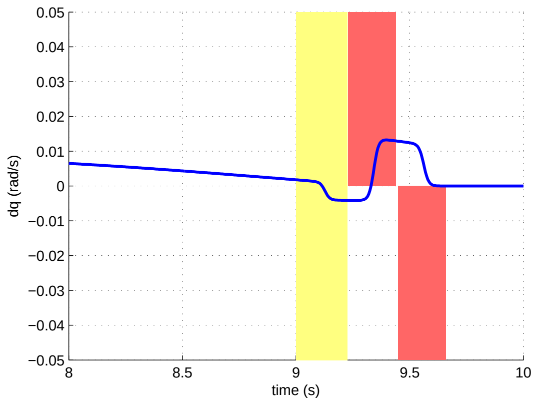
This issue can be highlighted by a simple system with two incompatible tasks $\mathbf{e_1}$ and $\mathbf{e_2}$, where $\mathbf{e_1}$ has higher priority. We study the behavior of the stack of tasks when the two tasks are to be removed almost at the same time. If $\mathbf{e_2}$ is removed first ($t_2^F < t_1^F$), there is no problem since it is the ideal case where only the task of lower priority is removed. If $\mathbf{e_1}$ has to be removed first, then since only the lowest priority task can be removed of the stack, a prior swapping operation between $\mathbf{e_1}$ and $\mathbf{e_2}$ is required. The effective removal time of $\mathbf{e_1}$ will thus be $t_1^F + \Delta t^{Swap}$.

To illustrate the discontinuous evolution of $\dot{\mathbf{q}}$ with respect to t_2^F, we simulate the regulation of two position tasks by the 2D-robot previously introduced. These two tasks share the waist joint and are to be removed at the same time. The Fig. 2.14 presents the two different evolutions of the control function depending on the task removed first.

Hence, this new stack of tasks scheme is likely to modify the schedule of the events, by adding extra delays in the original task plan. This corresponds to another discontinuity in the function which associates a control shape to a given sequence of discrete events ($f : \{\text{events}\} \to (t \to \dot{\mathbf{q}}(t))$). On the opposite, the original stack of tasks ensures the invariance of the sequence of events, yet without guaranteeing a continuous control.



a) $t_2^F < t_1^F$



b) $t_2^F > t_1^F$

Figure 2.14: Evolution of the control $\dot{\mathbf{q}}$ depending on t_2^F during the removal phase. In a) no swap is realized and in b) a swap operation is required. The light zone represents the swap period. The upper dark zone represents the removal period of the task $\mathbf{e_1}$, the lower dark zone represents the removal period of the task $\mathbf{e_2}$.

2.3 Insertion/Removal at any level of the stack

In order to reduce the delay required to place a task at a desired priority, we define a way to insert directly a task anywhere in the stack and to remove any task in the stack, while preventing the low priority tasks to create discontinuities.

Using an insertion gain is a solution that only works for the last task of the stack: introducing a task at higher priority is likely to cause a modification of the control corresponding to lower priority tasks. The simple case of a stack containing only one task $\mathbf{e_A}$ in which the task $\mathbf{e_B}$ is inserted at top priority is relevant: $\lim\limits_{\substack{t<t_B^I \\ t\to t_B^I}} \dot{\mathbf{q}}(t) = \mathbf{J_A^+}\dot{\mathbf{e}}_\mathbf{A}^*$, while $\lim\limits_{\substack{t>t_B^I \\ t\to t_B^I}} \dot{\mathbf{q}}(t) = (\mathbf{J_A P_B})^+\dot{\mathbf{e}}_\mathbf{A}^*$: the continuity of $\dot{\mathbf{q}}$ is ensured only if $\mathbf{e_A}$ and $\mathbf{e_B}$ are not conflicting.

The method proposed is based on the management of the joints used by the each task. While inserting/removing a task of rank p, none of the lower priority tasks can use the joints used by a task of higher priority. A task which has no more joints left to use is temporarily removed of the stack.

2.3.1 Weighted pseudo-inverse approach

In order to realize a partial insertion/removal of a task, we use a weighted pseudo inverse [22] defined when $\mathbf{A}$ is full-row-rank by:

$$\mathbf{A}^{\#\mathbf{w}} = \mathbf{W}\mathbf{A}^\mathbf{T}(\mathbf{A}\mathbf{W}\mathbf{A}^\mathbf{T})^{-1}$$

The matrix $\mathbf{W} = diag(\mathbf{w})$ is a diagonal gain matrix modifying the influence of each joint regarding the other. Thus it is possible to inhibit some joints, as long as there is still one active. In order to ensure a continuous evolution of $\mathbf{W}$, $\mathbf{w}$ is chosen to be a C^1 function: $\mathbb{R}^+ \to [0,1]^k$, where k is the number of joints.
The control law becomes thus

$$\dot{\mathbf{q}}_\mathbf{i} = \dot{\mathbf{q}}_\mathbf{i-1} + (\mathbf{J_i P_{i-1}})^{\#\mathbf{w_i}}(\dot{\mathbf{e}}_\mathbf{i}^* - \mathbf{J_i}\dot{\mathbf{q}}_\mathbf{i-1}), \tag{2.28}$$

where $\mathbf{w_i}$ corresponds to the weight associated to each joint.

2.3.2 Insertion/removal mechanism

Let $\mathbf{e_i}$ be a task. We define the corresponding sets:

- a_i the used joints, corresponding to the non-null rows in the Jacobian,
- a_i^F the forbidden joints, corresponding to those used by higher priority tasks,

$$a_i^F = \cup_{k<i} a_k \tag{2.29}$$

- a_i^B the booked joints, which are not used by any higher priority tasks, (such as $\forall k < i \Rightarrow a_k^F \cap a_i^B = \emptyset$),

$$a_i^B = a_i \cap \neg a_i^F \tag{2.30}$$

The algorithm is as follows: if $a_i^B = \emptyset$, the task must be temporarily removed, else only the forbidden joints are inhibited. The insertion and removal of a task is done using an insertion gain as defined before. However, in the frame of this method, it is meaningful to define an insertion gain for any task in the stack, since the disturbances due to lower priority tasks are avoided by the system of joint reservation. Thus, the insertion gain associated to the task i is noted λ_i^{Ins}. The partial removal is realized by decreasing the gain associated to forbidden joints $\mathbf{w_i}|a_i^F$, while maintaining (or eventually increasing) the others at 1.

2.3.3 Illustrative example

As proved in the Annex A, this method ensures a continuous evolution of the control output while inserting or removing tasks at any level of the stack of tasks. For practical purposes, a delay is required to realize the preliminary deactivation of lower priority tasks. To illustrate this method, we consider that the two arms 2D-robot has to realize three tasks as depicted in Fig. 2.15. The tasks considered are:

- $\mathbf{e_1}$, a position task of the left hand (cyan),
- $\mathbf{e_2}$, a position task for the left elbow and (green) and
- $\mathbf{e_3}$, a position task for the right hand (red).

All of them may use the waist joint and $\mathbf{e_1}$ can use all the joints used by $\mathbf{e_2}$. $\mathbf{e_1}$ has higher priority over both $\mathbf{e_2}$ and $\mathbf{e_3}$, and $\mathbf{e_2}$ has higher priority over $\mathbf{e_3}$. The tasks are inserted in inverse priority order.

The Fig. 2.16 illustrates the discontinuous behavior of the control when using the classical stack of tasks with insertion gains. The Fig. 2.17a shows the evolution of the control and Fig. 2.17b represents the evolution of the gains of each tasks and of the vector $\mathbf{w_3}$ characterizing the weighting of $\mathbf{e_3}$.

The following interesting events can be observed. Before the insertion of $\mathbf{e_2}$ (t_2^I= 0.5s), $\mathbf{e_3}$ is the only task in the stack. Since $\mathbf{e_2}$ and $\mathbf{e_3}$ both use the waist joint, the task $\mathbf{e_3}$ must release it before t_2^I. Thus, during a safety delay preceding t_2^I, $\mathbf{e_3}$ is partially removed, and this causes a higher solicitation of the other joints of the right arm (in red and cyan in Fig. 2.17a). This partial removal of $\mathbf{e_3}$ is illustrated in Fig. 2.17b, where only part of the vector $\mathbf{w_3}$ decreases to 0, while its insertion gain λ_3 remains equal to 1.

At the insertion of $\mathbf{e_1}$ (t_1^I= 1s), $\mathbf{e_1}$ and $\mathbf{e_2}$ are already in the stack. Again, $\mathbf{e_3}$ will be only partially removed, since the right arm is free. However, since the task $\mathbf{e_1}$ uses all the joints required by $\mathbf{e_2}$, the task $\mathbf{e_2}$ must be temporarily deactivated before t_1^I. The deactivation of $\mathbf{e_2}$ is illustrated in Fig. 2.17b, where the insertion gain λ_2 is temporarily set to 0 around t_1^I.

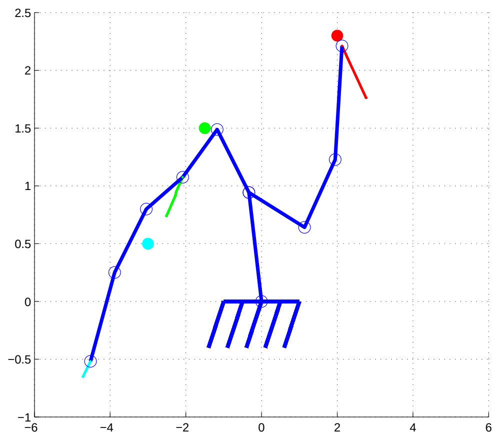


Figure 2.15: The 3 position tasks the 2D-robot has to regulate.

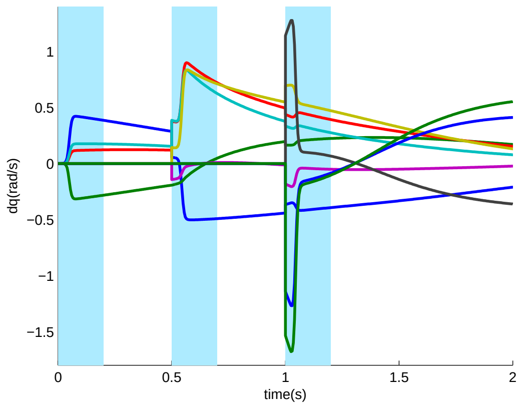


Figure 2.16: Evolution of the control while realizing the three tasks, using the classic control law with insertion gains. Each blue zone indicates the insertion period of a task (time during which its gain smoothly increases to 1).

2.3.4 Flaws of the method

This example illustrates that weighted pseudo-inverse *push* and *pull* suffer critical flaws: while inhibiting some joints for a task, the remaining ones are solicited instantly and hence can reach high values. Besides, locking some joints increases the likelihood of reaching

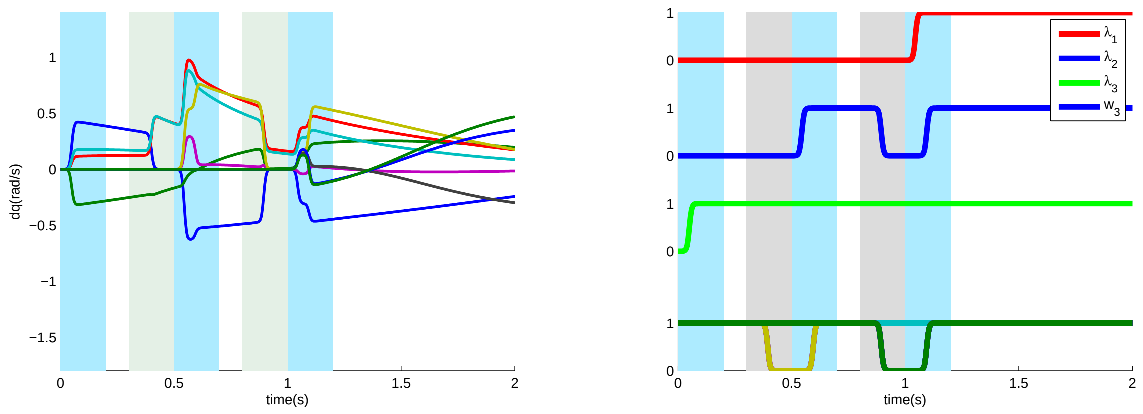


Figure 2.17: a) Evolution of the control using the weighted pseudo- inverse, b) Evolution of the gains. Each blue zone indicates the insertion period of a task (time during which it is activated and its gain smoothly increases to 1). Each grey zone indicates the safety delay preceding the removal. The three first curves represent the evolution of the insertion gain of the three tasks $\lambda_1, \lambda_2, \lambda_3$. The last one represents the evolution of the weighting vector $\mathbf{w_3}$.

kinematics singularities (the goal can become out of reach). And finally, the deactivation – even if it is partial– of some tasks is not conceivable (for example, the release of an on-going stability or grasping task is certainly not acceptable). As a result, this method, as described earlier, is generally not applicable on complex systems.

This section also illustrates why direct removal or insertion of a task at any priority level causes discontinuities in the control; in our opinion, an approach allowing such operations can hardly be conceived. Therefore, using a *swap* propagation appears to be the more appropriate solution, in spite of its cost in the length of the transition period.

2.4 Solving the stack of tasks using a single minimization algorithm

This section presents other techniques used to realize a hierarchy of tasks, by formulating the process as a single integrated minimization problem. The common advantage of these methods is to make the realization of discrete events easier, since it can be done numerically. Yet, the common flaw is that the hierarchy is less strict: the realization of lower priority tasks can disturb the realization of higher priority ones.

2.4.1 Cascade of α-weightings

This method, presented in [20], consists in an application of the α-weighting on the whole stack. The control results from the solution of the following problem:

$$\lim_{\alpha \to 0} \min_{\dot{\mathbf{q}}} \left(\sum_{i=1}^{n} \left(\|\mathbf{J_i}\dot{\mathbf{q}} - \dot{\mathbf{e}}_{\mathbf{i}}^{*}\|^2 \alpha^{(i-1)} \right) + \|\dot{\mathbf{q}}\|^2 \alpha^n \right) \tag{2.31}$$

Two approaches are relatively similar to this one. The same formulation is used in [78], at the difference that the coefficient are set manually. This method favours the flexibility of the control to the respect of a strict hierarchy.

$$\min_{\dot{\mathbf{q}}} \left(\sum_{i=1}^{n} \left(\|\mathbf{J_i}\dot{\mathbf{q}} - \dot{\mathbf{e}}_{\mathbf{i}}^{*}\|^2 \alpha_i \right) \right) \tag{2.32}$$

The following formulation is used in [40] to define the influence of each constraint on the control by modifying the coefficient matrix $\mathbf{Q}$.

$$\begin{array}{ll} & \min_{\mathbf{x}} \left(\mathbf{x}^T \mathbf{Q} \mathbf{x} \right) \\ st. & \mathbf{J_i}\dot{\mathbf{q}} = \dot{\mathbf{e}}_{\mathbf{i}}^{*} + \boldsymbol{\gamma_i} \\ where & \mathbf{x} = [\dot{\mathbf{q}}; \boldsymbol{\gamma_1}; \dots; \boldsymbol{\gamma_n}] \\ and & \mathbf{Q} = \text{diag}([\boldsymbol{\alpha_0}; \dots; \boldsymbol{\alpha_n}]) \end{array} \tag{2.33}$$

We use the definition (2.31) and approximate the solution using a small positive value for α. The swap of two neighbour tasks $\mathbf{e_k}$ and $\mathbf{e_{k+1}}$ consists in continuously changing their coefficient from α^{k-1} down to α^k and from α^k to α^{k-1} respectively.

The minimization problem was solved using the QP solver provided in the Matlab optimization toolbox. The system simulated is the one presented in Section 2.2: two position tasks are considered for the two arms $\mathbf{e_1}$ and $\mathbf{e_2}$, and the last task defines a default value in the null space of the two tasks.

The Fig. 2.18 represents the evolution of the control for two tasks that do not conflict. For small values of α (Fig. 2.18c), the evolution of the control is close to the expected behavior: the control is independent of the priority of the tasks. However, as the value of α increases, the hierarchy is not respected any more, and the evolution of the control depends on the order of the tasks in the stack. This appears in Fig. 2.18b and more clearly in Fig. 2.18a, where the swap operation becomes visible.

When the two tasks are conflicting, the last task realizes the damping and limits the control values obtained. The smaller α is, the higher the authorized control values near the singularities will be. Though, if α is too small, the numerical values associated to the lower priority tasks will be too small to be relevant in the minimization problem: they are not considered anymore. In Fig. 2.19c, the discontinuous behavior that appears after the swap ($t > 1.2$s) is due to the fact that the second task is only taken into account punctually.

Hence, the choice of α is based on a compromise: if it is too small, the low priority tasks will not be taken into account any more, which causes discontinuities in the evolution of the

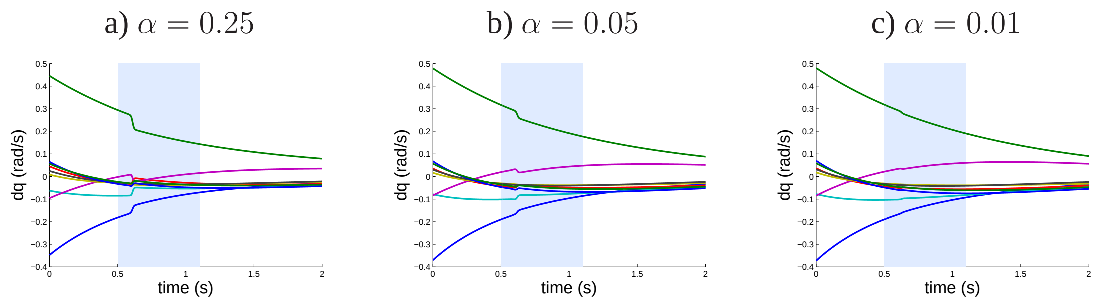


Figure 2.18: Evolution of the control for two tasks that do not conflict using the α-weighting with different values of α.

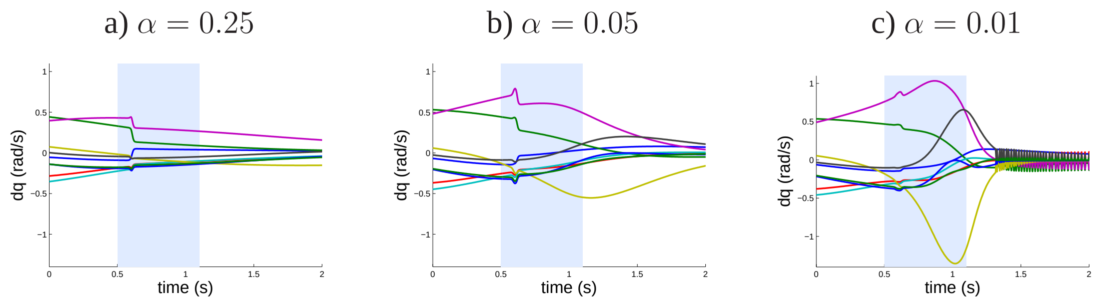


Figure 2.19: Evolution of the control for two tasks that conflict using the α-weighting with different values of α.

control; at the opposite, if it is too high, the hierarchy of tasks may not be strictly guaranteed.

2.4.2 “Single task" formulation $\mathcal{J}\dot{\mathbf{q}} - \mathcal{E}$

An elegant approach would consist in solving the entire stack of tasks with a unique minimization problem, that is:

$$\min_{\dot{\mathbf{q}}} \|\mathcal{J}\dot{\mathbf{q}} - \mathcal{E}\|^2 \tag{2.34}$$

This method would make the add of constraints to the control easier. Whereas these constraints must be respected for each of the n minimization problems of the formulation (2.5), they need to be considered only once with this approach. Then, a solution to avoid excessive control values would be to add bounds constraints on $\dot{\mathbf{q}}$.

Yet, finding the adequate matrices $\mathcal{J}$ and $\mathcal{E}$ is not trivial. Indeed, one could consider the set

$$\mathcal{J} = \begin{bmatrix} \mathbf{J_1} \\ \vdots \\ \mathbf{J_n P_{n-1}} \\ \mathbf{P_n} \end{bmatrix} \quad \mathcal{E} = \begin{bmatrix} \dot{\mathbf{e}}_\mathbf{1}^* \\ \vdots \\ \dot{\mathbf{e}}_\mathbf{n}^* \\ \mathbf{0} \end{bmatrix} \tag{2.35}$$

where the last "task" of the stack is characterized by an identity Jacobian and by an error equal to $\mathbf{0}$, and is applied in the null space left by the n tasks. Its purpose is to ensure a null value for the non-controlled axes.

However, this formulation is similar to that of equation (2.36), which is an incomplete formulation of the problem (2.7), where the additional error induced by previous tasks is not taken into account. This formulation is not satisfying, since it delays the adaptation of lower priority tasks to the modifications of configuration induced by higher priority tasks.

$$\mathbf{\dot{q}_i} = \mathbf{\dot{q}_{i-1}} + (\mathbf{J_i P_{i-1}})^{+} \mathbf{\dot{e}_i^*} \tag{2.36}$$

Based on this, the set corresponding to (2.7) appears to be:

$$\mathcal{J} = \begin{bmatrix} \mathbf{J_1} \\ \vdots \\ \mathbf{J_n P_{n-1}} \\ \mathbf{P_n} \end{bmatrix} \quad \mathcal{E} = \begin{bmatrix} \mathbf{\dot{e}_1} \\ \vdots \\ \mathbf{\dot{e}_n} - \mathbf{J_n \dot{q}^*_{n-1}} \\ \mathbf{\dot{q}} \end{bmatrix} \tag{2.37}$$

where $\mathbf{\dot{q}_i^*}|_{\mathbf{i}\in[\mathbf{1},\mathbf{n-1}]}$ are estimations of each of the $\mathbf{\dot{q}_i}$. It is important to notice that each term $\mathbf{\dot{q}_i^*}$ must be computed independently from this minimization problem. For example, they may be computed beforehand using the classic damped control law.

Indeed, the current minimization problem prevents using each $\mathbf{\dot{q}_i}|_{\mathbf{i}\in[\mathbf{1},\mathbf{j-1}]}$ while computing the control associated to priority $\mathbf{j}$. A brute-force solution to this would be to consider the control value of each layer $\mathbf{\dot{q}_1}, \ldots, \mathbf{\dot{q}_n}$ as a parameter of the optimization, but it highlights the problem of disrespect of the hierarchical structure of the stack. In order to illustrate this issue, let's consider the optimization problem (2.38). For the sake of clarity and without loss of generality, only two tasks are considered and the last task used to set the control at $\mathbf{0}$ is also omitted.

$$\min_{\mathbf{\dot{q}_1}, \mathbf{\dot{q}_2}} \left\| \begin{bmatrix} \mathbf{J_1} & 0 \\ \mathbf{J_2(I - P_1)} & \mathbf{J_2 P_1} \end{bmatrix} \begin{bmatrix} \mathbf{\dot{q}_1} \\ \mathbf{\dot{q}_2} \end{bmatrix} - \begin{bmatrix} \mathbf{\dot{e}_1^*} \\ \mathbf{\dot{e}_2^*} \end{bmatrix} \right\|^2 \tag{2.38}$$

Although this system seems similar to (2.7), it does *not* respect the hierarchical structure of the stack of tasks: the minimization of the second line may cause a modification of $\mathbf{\dot{q}_1}$, which should depend only on the first equation. Thus, using a fixed value for each of the $\mathbf{\dot{q}_i}|_{\mathbf{i}\in[\mathbf{1},\mathbf{n}]}$ appears to be the only way to prevent interdependencies between the levels of the stack of tasks.

2.5 Conclusion

In this chapter, we investigated several potential solutions to the problem of smooth insertion, removal or swapping of a given task at any priority in the stack. It appears that it is difficult to perform such operations *a priori*, i.e. through an instant explicit rewriting of the governing equations, because this generally always results in discontinuous control output. Smoothing *a posteriori*, i.e. filtering the control output, is not an option since there is no guarantee on the feasibility of the motion. In the general case, swapping a pair of tasks of any priority can be translated by a fixed number of task insertions and removals, therefore, we thought that solving task removal and insertion at any priority level will solve the problem: we explained why this was not possible.

Our solution consists in realizing insertions and removals by a succession of smooth swap operations between adjacent pairs of tasks. Hence, to insert a task at a given priority, it is necessary to insert it the end of the stack and move it to the desired priority by realizing a succession of smooth swaps with the adjacent higher priority task. For a removal, the reverse process is done. We provided methods to ensure a smooth evolution of the control during the discrete events: the swap of priority of adjacent pairs of tasks is realized using a method based on a linear interpolation and the removal and the insertion of the lowest priority task are smoothed by the use of an insertion gain. We also presented other approaches and discussed their pros and cons.

Even if it respect the continuity during discrete events, the method proposed presents also some flaws, namely the delay of realization of the events. Hence, performing stack basic operations (insertion/removal) at any priority straightforwardly remains an open challenge.

The following chapter will make use of these results to optimize the scheduling of a set of tasks by using an optimization formulation of the problem.

Chapter 3

Task overlapping using optimization

While previous chapter described how the stack-of-tasks mechanism enables the realization of a given set of tasks, this chapter focuses on achieving both scheduling and task behavioral tuning by allowing *task overlapping* for a given sequence of tasks, i.e. a set of tasks whose order of insertion and removal are constrained. Overlapping tasks consists in adjusting the overall sequence in the stack of tasks so as to realize the tasks simultaneously (when possible) rather than sequentially, thus producing a smoother motion and exploiting redundancy.

As a symbol, the task is used to describe the task plan (component of the plan reasoning), and as a task function, it is used to execute this plan (component of the low-level control). This chapter presents an intermediary computation module inserted between these two phases, which realizes task overlapping using an optimization process. Its purpose is to find the optimal parameters for each task of the task plan, i.e. the best task schedule and the best way to realize them, while fulfilling the constraints imposed both by the sequence (task ordering) and by the environment (intrinsic limitations of the robot –generally also described as tasks–, collision avoidance, etc.).

This chapter has been realized in collaboration with Sylvain Miossec, lecturer at the PRISME institute, for the section 3.3

3.1 Introduction

3.1.1 Illustrative example

Consider the following mission given to a humanoid robot: “Go bring that object located on that table and bring it back to your local position". This mission can be decomposed into a sequence of tasks achieved sequentially, i.e. one after the other, represented on Table 3.1.

This plan is sound and certainly safe. However, realizing the tasks one after the other in

- $\mathbf{e_1}$ “Go to the table";
- $\mathbf{e_2}$ “Grasp the specified object", which can be split into two other sequential tasks:
  - $\mathbf{e_{2a}}$ “Place the gripper around the object" (the exact position of the gripper can be the outcome of a grasp planner [8]);
  - $\mathbf{e_{2b}}$ “Close the gripper" (with the appropriate computed force closure);
- $\mathbf{e_3}$ “Go to your starting point".

Table 3.1: Decomposition of the mission “Go bring that object located on that table and bring it back to your local position".

an exclusive way results in automated-looking motions that can even be jerky. The purpose of overlapping tasks is not only to enhance the performance efficiency, e.g. in terms of speed for the same tasks’ behavior, but also to provide smooth-looking transitions in the motions of the robot and exploit nicely its redundancy capabilities.

Two overlays are possible for the previous sequence: the gripper positioning task $\mathbf{e_{2a}}$ can start before the end of the walking task $\mathbf{e_1}$ and the task $\mathbf{e_{2b}}$ can start before the end of $\mathbf{e_{2a}}$. However, the realization of these overlays cannot be done arbitrarily since the tasks are coupled.

For the first overlap, it is not desirable to have the robot arrive nearby the table, stop walking, and, only then, start the positioning motion toward the object. It would be better to start the gripper positioning task at some point nearby the table while the robot is still walking (which is possible and more smooth-looking). By inserting this positioning task together with the walking task in the stack, the robot will start moving its gripper while walking. If the object is far, the robot will move with a stretched arm, which is certainly bad and not the optimal way to walk. It is hence better to find the right timing to stack (insert) the gripper positioning task so that it occurs right before the walking task ends (as a human would do). Moreover, it is necessary to ensure that the motion of the arm does not perturb the walking pattern (e.g. by modifying its balance).

In the second task overlap, a new constraint on the gripper has to be explicitly taken into account: the gripper should not be completely closed when reaching the object. This constraint is satisfied implicitly by enforcing a strong precedence condition (exclusive task sequence). The difficulty when realizing task overlapping lies on properly taking into account such constraints explicitly.

This example highlights some of the difficulties in achieving an efficient task overlapping; it is not a straightforward process. It is necessary to check that the obtained sequence is still valid for each possible overlapping. While the exclusive realization of two tasks is very likely to fulfill all the constraints that are inherent to the robot capabilities and its surrounding environment, their (partly) simultaneous realization may violate them.

3.1.2 From planning to execution

When the environment is entirely or partially structured, most of the robotic missions can be decomposed into a sequence of primitive actions (or operations) that can be executed by the robot. The building of the sequence is usually realized by a symbolic planner [5]. Starting from this symbolic sequence, the two following approaches are possible.

3.1.2.1 The trajectory approach

In this approach, a trajectory is computed from the symbolic plan, then the symbolic data are converted into numerical data in order to give to the robot a doable command. This loss of symbolic data, in favor of numeric ones, makes it difficult to adapt to changes that may occur in the environment. As a consequence, local or global re-computation of the trajectory may be required; and this can be time consuming [53, 5].

3.1.2.2 The task-based approach

This second approach takes advantage of the fact that the task (in the sense of task function [79]) is the common component to all the steps, from plan reasoning to execution. In [73], the task is used to define both the high level specifications of the action expressed in formal logic causes (planning) and the low level formulation allowing to compute the control (execution). As a result, the author was able to propose a mechanism to reactively modify and select the task sequence plan during the execution.

For its flexibility, we chose to rely on the task-based method.

3.1.3 Task sequencing

The realization of task overlapping requires preliminary adaptations of the task sequence. It is indeed necessary to release some timing constraints of the sequence (typically, precedence constraints) by replacing them by other constraints (such as collision avoidance, or dependency on the regulation of a task). Verifying the satisfaction of these constraints cannot be done during the symbolic planning, since these constraints are not written into a symbolic form (they are numeric).

In order to realize this operation during the planning, the use of hybrid planners, that considers both numerical and symbolic data, is tempting. In [35], the symbolic planner is bound to a motion planner, which is in charge of the geometric data. Hence, a doable geometrical path is given, but it cannot be linked back to the tasks definition. In [57], the planner handles the geometrical data by using flow-tubes: each action has a fixed duration and is associated to a function which, given a set of start configurations, returns the set of all the reachable configurations. This approach is not compatible with the stack-of-tasks mechanism, since the considered tasks are independent.

Therefore, we come to the conclusion that the overlapping cannot be performed during the planning phase whether using a symbolic planner or a hybrid planner. Yet, similarly to the planning process, the overlapping has to be done off-line, before the execution. Subsequently, an intermediate step is added between the creation of the symbolic task plan and its execution. This step is formulated as an optimization process that takes into account numerical constraints while realizing the task overlapping. The global scheme is illustrated by Fig. 3.1.

In the following, we discuss how we formulated the optimization problem. Using the definition of a sequence of tasks and considering the stack-of-tasks controller, we define how to formulate the tasks overlapping as an optimization problem, and how solve it. Finally some open problems together with rooms of possible improvements of our method are mentioned.

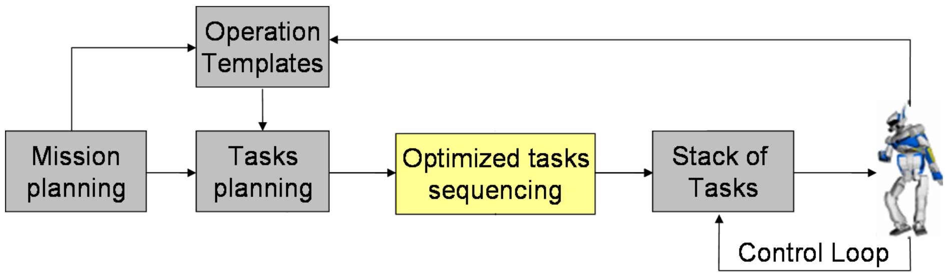


Figure 3.1: From mission planning to execution with the task overlapping module

3.2 Sequence of tasks

A sequence of tasks is a finite set of primitive tasks sorted by order of realization and eventually linked to each other. Any pair of tasks can be either independent (i.e. they can be achieved in parallel if possible) or constrained by time constraints (precedence constraints or simultaneity constraints) or constrained by more generic conditions (e.g. a task must wait for another one to be achieved before it can start). In the following, a sequence of tasks is described as a classical temporal network scheduling. First the temporal parameters describing a task are defined; then the temporal dependencies between tasks are specified.

3.2.1 Definition of a task in a temporal network

A task describes an action the robot has to realize. Since several primitive tasks can define a same action, we consider for the sake of clarity but without loss of generality that each task appears only once in the sequence.

The *position* of a task in the sequence is defined by the time interval during which it is maintained in the stack-of-tasks controller. For a given task $\mathbf{e_i}$, this interval is noted $[t_i^I, t_i^F]$: the task enters in the stack of tasks at t_i^I and is removed at t_i^F. Yet, these instants do not indicate the achievement level of the task: t_i^F may happen before the task regulation is completed. Let ϵ_i be the tolerance on the task regulation completion: a task is considered as regulated when $\|\mathbf{e_i}(t)\| \leq \epsilon_i$. The regulation time t_i^R is the time of the first regulation of the task, defined by

$$t_i^R = \left(\min_t \|\mathbf{e_i}(t)\| < \epsilon_i \right) \tag{3.1}$$

The task can be split into two phases: a regulation phase, during the time interval $[t_i^I, t_i^R]$, where the task error *decreases* toward the given value ϵ_i, and a maintaining phase (or holding phase), during the time interval $[t_i^R, t_i^F]$, where the task error *is kept* below ϵ_i (cf. Fig. 3.2). The error regulation remains active and acts if any perturbation (that may bring the error to increase above ϵ_i again) occurs during the time interval $[t_i^I, t_i^F]$.

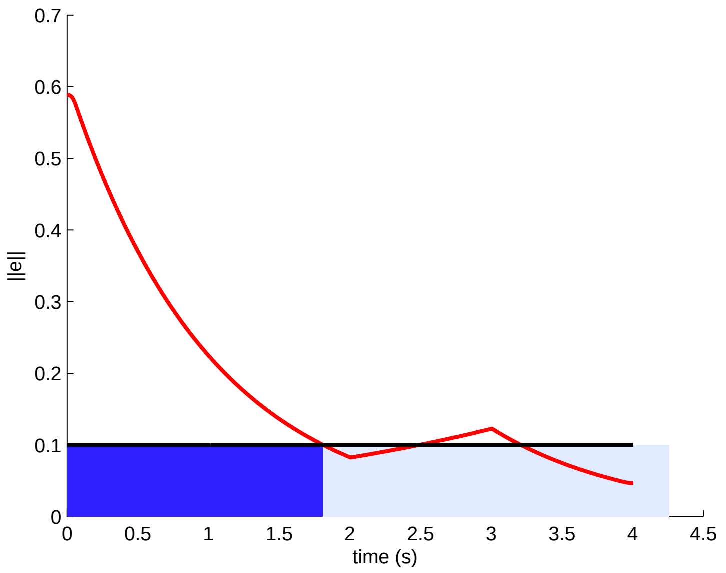


Figure 3.2: The evolution of the task error defines two phases: the dark one is the regulation phase $[t_i^I, t_i^R]$, the bright one is the maintaining/holding phase $[t_i^R, t_i^F]$. Before t=2s, the task is alone in the stack, and the evolution of the error follows an exponential decrease. Between t=2s and t=3s, a higher priority task makes the error of the current task increase and exceed the tolerance ϵ_i again. At t=3s, the higher priority task is removed, and the error is regulated again.

3.2.2 Time dependencies

A task sequence starts at t^0 and ends at t^{End}. Both values are finite and the sequence does not loop. The sequence is characterized by a set of time constraints binding the schedules of two tasks $\mathbf{e_i}$ and $\mathbf{e_j}$. They are defined as follows[1]: $\mathbf{e_i}$ must begin or end once $\mathbf{e_j}$ has begun, has ended or has been regulated. We use the graphical representation given by Fig. 3.3 and the following notation to describe the sets of pairs of tasks $\mathbf{e_i}$ and $\mathbf{e_j}$ that undergo these

[1]Contrary to Allen Logic, that only considers the start and end points of the time interval, here we additionally consider the regulation time t^R.

dependencies ($\mathbf{e_i}$ is the direct predecessor of $\mathbf{e_j}$):

$$S_{I,I} = \{(\mathbf{e_i}, \mathbf{e_j}) \mid t_i^I \leq t_j^I\} \tag{3.2a}$$
$$S_{R,I} = \{(\mathbf{e_i}, \mathbf{e_j}) \mid t_i^R \leq t_j^I\} \tag{3.2b}$$
$$S_{F,I} = \{(\mathbf{e_i}, \mathbf{e_j}) \mid t_i^F \leq t_j^I\} \tag{3.2c}$$
$$S_{R,F} = \{(\mathbf{e_i}, \mathbf{e_j}) \mid t_i^R \leq t_j^F\} \tag{3.2d}$$
$$S_{F,F} = \{(\mathbf{e_i}, \mathbf{e_j}) \mid t_i^F \leq t_j^F\} \tag{3.2e}$$

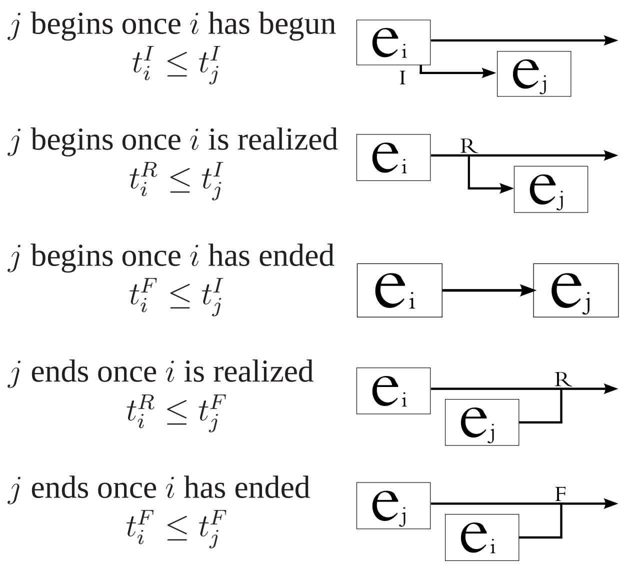


Figure 3.3: Five time-dependency relations are considered.

A pick-and-place action can be described as follows: once the robot has grasped the object, it has to maintain the force closure on the object while moving it, and can release it (i.e. open the gripper) only after the task of displacement has ended. This sequence contains only two tasks, the grasping task $\mathbf{e_g}$ and the moving task $\mathbf{e_m}$, and is characterized by two time constraints: $t_g^R \leq t_m^I$ and $t_m^F \leq t_g^F$. The Fig. 3.4 illustrates this sequence of tasks.

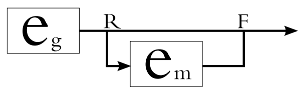


Figure 3.4: A task is realized during the maintaining period of another one.

3.3 Optimization of a given sequence of tasks

The purpose of the optimization process is to compute a better task schedule by taking advantage of the robot capabilities to overlap tasks (or even reorder them), in order to realize a smooth-looking motion.

Particularly, we want to use the stack-of-tasks formalism to realize several tasks simultaneously rather than executing them sequentially, one after the other.

Quantifying numerically the task overlay is not possible. The criterion chosen for the optimization process is the reduction of the duration of the whole sequence. Indeed, this reduction is done either by operating at the task behavioral level (by decreasing the duration of the critical tasks, in other words tuning gains), or by operating at the sequence level (by realizing tasks overlapping). Other criteria, such as the reduction of the energy or the jerk minimization, could be used similarly but they do not necessarily force an overlap of tasks.

In the same way as a symbolic planner cannot work with numeric data, a numerical optimization process cannot work with symbols. Hence the tasks are converted into proto-symbols [38]: their definition is completed with numerical values, that will be the base of the optimization. Two characteristics of the tasks are parameterized: their position in the sequence and their behavior (i.e. the way the tasks are regulated).

3.3.1 General problem formulation

The general optimization problem is written as follows:

$$\min_{\mathbf{x}} t^{\text{End}} \tag{3.3a}$$

$$\text{subject to } \dot{\mathbf{q}} = SoT_{\mathbf{x}}(\mathbf{q}, t) \tag{3.3b}$$

$$\text{seq}(\mathbf{x}) \leq 0 \tag{3.3c}$$

$$\phi(\mathbf{q}, \dot{\mathbf{q}}) \leq 0 \tag{3.3d}$$

- t^{End} is the duration of the entire mission.
- $\mathbf{x}$ is the vector of parameters for the optimization.
- $\mathbf{q}$ and $\dot{\mathbf{q}}$ are respectively the position and velocity of the system, both depending upon time.
- $\text{seq}(\mathbf{x})$ and $\phi(\mathbf{q}, \dot{\mathbf{q}})$ represent respectively the tasks constraints and the robot constraints. They are defined in section 3.3.3.

3.3.2 Parameters of the optimization

The vector of parameters $\mathbf{x}$ contains both the temporal data (time scheduling) that define the sequence and the gains that define the way the tasks are realized (task's behavior).

Task's timing Some of the times used to define the sequence can be used as parameters of the optimization: the time of insertion t^I, the time of removal t^F and the time of end of the simulation t^{End} can be tuned by the optimization process. On the opposite, the time of regulation t^R is not explicitly considered since it depends on the evolution of the error of the task. These timings can either be defined as absolute times (i.e. defined with respect to

$t^0 = 0$), or relative times (i.e. defined with respect to another time in the schedule).

Using absolute times comes down to directly use as parameters the insertion time t^I and the removal time t^F for each task. Since all times are defined with respect to t^0, moving a task forward or backward in the sequence will not have any influence neither on the other tasks nor on the time of end of the simulation: it is therefore necessary to add a propagation mechanism.

As an illustration, consider two tasks, $\mathbf{e_1}$ and $\mathbf{e_2}$, realized sequentially, as depicted on Fig. 3.5. We want to reduce the time of end of the entire simulation t^F, by modifying only the timing of the tasks and using the fact that the first task could be started sooner (the lower constraint on t_1^I is not saturated yet and t_1^I can still be decreased). Decreasing only t_1^I will extend the duration of the first task (unnecessarily), without changing t^F. In order to decrease t^F without violating the schedule constraints, it is necessary to move both tasks backward in the sequence, i.e. to decrease the times t_1^I, t_1^F, t_2^I and t_2^F.

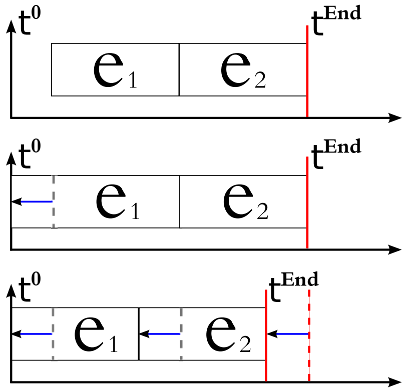


Figure 3.5: (Up) Initial task schedule. The first task can be started earlier. (Middle) Only the start time of the first task is modified: the duration of the simulation does not change. (Down) Propagation of the lag: the duration of the simulation is reduced.

The purpose of using relative times is to implicitly achieve this propagation. Each task is now described by two delays, namely:

1. Δt^I: the delay which occurs between (i) the maximum time of entry or end of the previous tasks, and (ii) the entry time of the task in question.

$$\Delta t^I = t_i^I - \max\left(\max_{(j,i)\in S_{I,I}}\{t_j^I\}, \max_{(j,i)\in S_{F,I}}\{t_j^F\}\right) \tag{3.4}$$

2. Δt^F: the delay between the entry and the removal times of the task in question.

$$\Delta t_i^F = t_i^F - t_i^I \tag{3.5}$$

The time duration of the task sequence t^{End} is such that $t^{\text{End}} = \max_i(t_i^F)$. To transform the equality into a $\mathcal{C}^1$ constraint, the optimization criterion t^{End} is computed indirectly, by adding it to the parameters of the optimization and restraining it by adding the following constraints:

$$\forall \mathbf{e_i}, t_i^F \leq t^{\text{End}}. \tag{3.6}$$

At the optimal solution, t^{End} will be equal to the maximum termination time of all tasks.

Task behavior In order to modify the way the tasks are regulated, we also parameterize the reference behavior $\dot{\mathbf{e}}^*$. The simple attractor presented in (2.2) ($\dot{\mathbf{e}}^* = -\lambda\mathbf{e}, \lambda \in \mathbb{R}^+$) introduces a direct dependency between $\dot{\mathbf{q}}$ and $\|\mathbf{e}\|$. The associated control follows a monotonous exponential decrease that can be penalizing for two reasons. First, it reaches its higher value at the insertion of the task, when $\|\mathbf{e}\|$ is maximal (fast acceleration), and second, it makes the task converge slowly (near the objective, the error and the control are small). The parameter λ only enables to avoid excessive value for the control $\dot{\mathbf{q}}$ at the insertion, but cannot correct the slow convergence issue.

A way to correct it is to rather use an adaptive gain $\lambda = \lambda(\mathbf{e})$ that depends on the norm of the error of the task. We therefore choose the following function:

$$\lambda(\mathbf{e}) = (\lambda^F - \lambda^I)\exp\left(-\|\mathbf{e}\|\beta\right) + \lambda^I \tag{3.7}$$

with λ^I the gain at infinity, $\leq$ the gain at regulation (such as $\lambda^I \leq \lambda^F$) and β the slope at regulation. The obtained gain gives a non monotonous evolution to the control law (cf. Fig. 3.6).

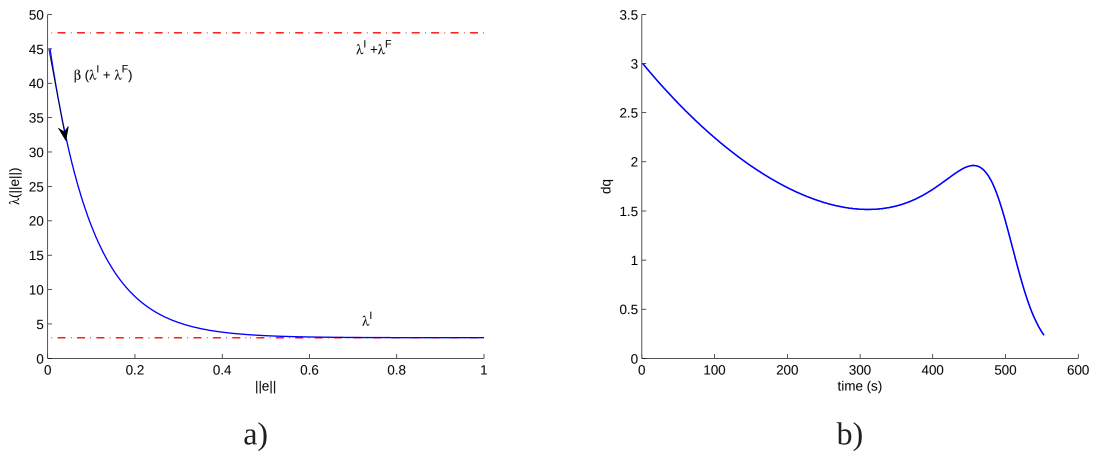


Figure 3.6: Typical evolution of a) the gain $\lambda(\|\mathbf{e}\|)$ and b) the joint velocity $\dot{\mathbf{q}}$ when using an adaptive gain. The evolution of the joint velocity is not monotonous and shows two inflexion points.

To sum up, the variables of our problem are:

1. the termination time of the entire mission (simulation);
2. the time of entry t^I and the time of removal t^F for each task;

3. the gains $(\lambda^I, \lambda^F, \beta)$ describing the execution behavior for each task.

Two sets of variables are thus conceivable for $\mathbf{x}$: the one that uses absolute times, $\mathbf{x_{1,A}}$, or the one that uses relative times, $\mathbf{x_{1,R}}$.

$$\mathbf{x_{1,A}} = [t_1^I, t_1^F, \lambda_1^I, \lambda_1^F, \beta_1, \ldots, t_n^I, t_n^F, \lambda_n^I, \lambda_n^F, \beta_n, t^{\text{End}}] \tag{3.8}$$

$$\mathbf{x_{1,R}} = [\Delta t_1^I, \Delta t_1^F, \lambda_1^I, \lambda_1^F, \beta_1, \ldots, \Delta t_n^I, \Delta t_n^F, \lambda_n^I, \lambda_n^F, \beta_n, t^{\text{End}}] \tag{3.9}$$

Note that if the task sequence is a chain of exclusive tasks, we directly have $\mathbf{x_{1,A}} = f(\mathbf{x_{1,R}})$, with f a linear function, and $t^{\text{End}} = \sum_i (\Delta t_i^I + \Delta t_i^F)$

3.3.3 Definition of the constraints of the optimization problem

The task sequence must satisfy both the sequencing and the robotic time-constraints enumerated hereafter:

Tasks constraints, noted seq$(\mathbf{x})$ gather the task sequence conditions (3.2) and the following constraints.
For each task $\mathbf{e_i}$:

Time coherence	$0 \leq t_i^I$	(3.10a)
	$t_i^I < t_i^F$	(3.10b)
	$t_i^F \leq t^{\text{End}}$	(3.10c)
Termination condition	$\lVert\mathbf{e_i}(t_i^F)\rVert < \epsilon_i$	(3.10d)
Gain consistency	$\lambda_i^I \leq \lambda_i^F$	(3.10e)

The tasks constraints can be sorted in two categories (cf. Table 3.2). Some are directly computable using the vector of parameters $\mathbf{x}$ (they are even linear when using the absolute set of parameters). Those linked to the regulation of a task (either the termination condition or the time constraints between tasks) can only be determined by a *simulation* of the execution (the value found is the value in the ideal case).

Particularly, the evaluation of constraints on time dependency (3.2b) and (3.2d) is not straightforward, since the time t^R is not directly computed. t^R could be approximated by monitoring the first regulation time of the task, but this method is not continuous, and the exact computation by interpolation is a rather fastidious and time consuming approach. Instead, a better option is to evaluate the regulation of the task $\mathbf{e_i}$ at these times. The constraints (3.2b) and (3.2d) become respectively:

$$\forall(i,j) \in S_{R,I}, \lVert\mathbf{e_i}(t_j^I)\rVert \leq \epsilon_i \tag{3.11a}$$

$$\forall(i,j) \in S_{R,F}, \lVert\mathbf{e_i}(t_j^F)\rVert \leq \epsilon_i \tag{3.11b}$$

Constraints computed directly using $\mathbf{x}$		Constraints computed by simulation	
$t_i^I \leq t_j^I$	(3.2a)	$t_i^R \leq t_j^I$	(3.2b)
$t_i^F \leq t_j^I$	(3.2c)	$t_i^R \leq t_j^F$	(3.2d)
$t_i^I \leq t_j^F$	(3.2e)	$\|\mathbf{e_i}(t_i^R)\| \leq \epsilon_i$	(3.10d)
$0 \leq t_i^I$	(3.10a)	$\|\mathbf{e_i}(t_j^I)\| \leq \epsilon_i$	(3.11a)
$t_i^I < t_i^F$	(3.10b)	$\|\mathbf{e_i}(t_j^F)\| \leq \epsilon_i$	(3.11b)
$t_i^F \leq t^{\text{End}}$	(3.10c)		
$\lambda_i^I \leq \lambda_i^F$	(3.10e)		

Table 3.2: Two categories of tasks constraints.

Using the relative time parameterization $\mathbf{x_{1,R}}$, the time constraints (3.2a), (3.2c) and (3.10a) can be replaced by the following constraint on the delay:

$$\forall i, 0 \leq \Delta t_i^I \tag{3.12}$$

And the time constraint (3.10b) can be replaced by

$$\forall i, 0 < \Delta t_i^F \tag{3.13}$$

Robot constraints, noted $\phi(\mathbf{q}, \dot{\mathbf{q}})$ These constraints are mainly due to intrinsic limitations of the robot:

Joint limits	$\mathbf{q}_{\min} \leq \mathbf{q} \leq \mathbf{q}_{\max}$	(3.14a)
Velocity limits	$\dot{\mathbf{q}}_{\min} \leq \dot{\mathbf{q}} \leq \dot{\mathbf{q}}_{\max}$	(3.14b)
Collision avoidance	$0 \leq d_{ij}$	(3.14c)

$\mathbf{q}_{\min}$, $\mathbf{q}_{\max}$, $\dot{\mathbf{q}}_{\min}$, $\dot{\mathbf{q}}_{\max}$ are respectively the lower and upper joint limits and the lower and upper velocity limits. d_{ij} corresponds to the distance constraints between two objects i and j in the environment. This formulation covers both collisions avoidance with the environment and self-collision avoidance. These constraints must hold for the entire simulation: $\mathbf{q}, \dot{\mathbf{q}}$ and d_{ij} are vectors of functions of time. The constraints $\phi(\mathbf{q}, \dot{\mathbf{q}})$ are indeed semi-infinite. The next section presents how they have been tackled.

3.4 Handling Semi-Infinite Constraints

3.4.1 Definition

The optimization problem that we formulated has the following form:

$$\begin{array}{ll} & \min\limits_{\mathbf{x}} f(\mathbf{x}) \\ \text{subject to} & g_j(\mathbf{x}, t) \leq 0 \\ \textit{where} & \mathbf{x} \in \mathbb{R}^n, t \in T \;\; (T = [t^0, t^{\text{End}}] \text{ being an infinite set}) \end{array} \tag{3.15}$$

This problem, noted $P[T]$, is a semi-infinite optimization problem (or SIP) [75]: it includes a finite number of parameters, but has infinitely many constraints. The semi-infinite dimension of the problem is due to the time-dependency: each constraint g must be satisfied all along the interval of time i.e. all along the motion trajectory, $\forall t$. In our case study, this period is always bounded and is noted $[t_g^I, t_g^E], ((t_g^I, t_g^E) \in \mathbb{R}^2)$.

$$\forall t \in [t_g^I, t_g^E], g(\mathbf{x}, t) \leq 0 \tag{3.16}$$

The following section presents briefly some of the classic methods used to solve a SIP, i.e. handle semi-infinite constraints in an optimization problem.

3.4.2 Specific methods

A method usually adopted to handle SIP is to use a discrete approach: each constraint is evaluated only a finite number of times according to a given grid, that fixes the time interval and the frequency of the evaluations. Since t_g^I and t_g^E are bounded (each of them can be constant value or a bounded variable changing during the optimization process), the discretization of this interval returns a finite number of grid's elements.

3.4.2.1 Multiple optimization process

To find the finest grid, an approach based on successive runs of the optimization process is proposed in [36, 74]: each run considers only a partial grid T_i to which corresponds an approximate solution $\tilde{\mathbf{x}}_i$. The sequence of grids converges toward a grid T^* such that $\lim\limits_{i \to +\infty} P[T_i] = P[T^*]$. This method is efficient if the solution of a run can be used to enhance the convergence properties of the next ones, namely by using it as a new start point.

This approach cannot be applied in our case study, since there is no a priori knowledge of the constraints that need to be active. The only remaining approach would be to consider a grid of increasing granularity for each run ($T_i \subset T_{i+1}$). However, realizing an optimization with a too large granularity presents the risk to miss the time when the constraint is violated. Hence, the resulting approximate solution will be unusable for a grid of smaller granularity, making this approach inappropriate.

3.4.2.2 Single optimization process

At the opposite, realizing a single optimization process enhances the simplicity and the computation time, but may result in a less precise satisfaction of the constraints [66]. Using a relatively high resolution grid, e.g. considering the constraints at every time step of the simulation, reduces the risks of missing constraints' violations, but this also increases drastically the number of constraints.

Besides, using a time-fixed grid (i.e. a grid whose elements are separated by a constant delay) is not suitable when the bounds depend on parameters that vary during the optimization process. Indeed, this results in a variation of the number of constraints from an iteration to another. Typically, one could consider that the grid is defined on the interval of time $[0, t^{\text{End}}]$. Since t^{End} varies during the optimization process, the number of constraints will vary. For example, with a grid that realizes an evaluation every second, the number of evaluation will decrease from 21 when $t^{\text{End}} = 20$s to 6 when $t^{\text{End}} = 5$s.

A classical optimization solver (such as CFSQP or IPOPT) requires that the number of constraints remains the same during the entire optimization process. As a result, using a time-fixed grid is problematic.

3.4.2.3 Analysis by interval

The purpose of this method is to avoid the problem inherent to the discretization approach. Indeed, choosing a fine grid results in increasing the optimization variables and subsequently the optimization computation time. More importantly, whatever the discretization grid and the efficiency of related solver are, there is in theory no guarantee that the constraints are satisfied within a pair of neighboring grid samples. Rather than increasing the granularity of the grid or adding a safety margin during the evaluation of the constraint (i.e. replacing $g \leq 0$ by $g+\epsilon \leq 0, \epsilon > 0$), interval-analysis based methods work on the interval of definition of the variables [67, 56]. Evaluating a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ on a whole interval defines a set of reachable values such that:

$$f([\underline{x}, \overline{x}]) = \{f(x) | x \in [\underline{x}, \overline{x}]\} \tag{3.17}$$

Hence, ensuring that the constraints are always satisfied comes down to modifying the interval of definition of the parameters so as to make the violations certainly tractable.

This approach is appropriate when the expression of the trajectory is explicit, e.g. when trajectory is defined using B-splines [56]. In our case, the trajectory is computed implicitly: it is the result of a numerical integration of the control given by the stack of tasks. Thus this method is not applicable straightforwardly to our problem.

3.4.3 Defining a constraint handler for Semi-Infinite Constraints

The method we adopted is based on a time-fixed grid: the evaluations are realized at fixed time step. Yet, to avoid the issue of the varying number of constraints, a post-treatment is realized so as to always return a constant number of evaluations, regardless of the number of evaluations realized by the grid.

Once the requirements defined, several methods are presented and compared, sorted in two categories: the first one gathers the evaluations realized on the whole interval as a single value while the second one returns a set of values.

3.4.3.1 Requirements

Our solvers require $\mathcal{C}^0$, preferably $\mathcal{C}^1$ constraints with respect to the variation of $\mathbf{x}$. Continuity with respect to the modifications of the gain is straightforward. The variations with respect to the times of insertion t^I and removal t^F are particularly important, since they correspond to discrete events that may introduce strong variations in the control computation.

As mentioned earlier, the evaluation of the constraints is realized through a simulation, at each time step and at each interpolation point. We investigated the following methods:

3.4.3.2 Single constraint

Maximum constraint value $g_{\max}$ Taking the maximum value (signed max distance of the constraint to 0) is the simplest way to associate a single value to a set of constraints of unknown size. This method has two drawbacks: (i) the max function is only $\mathcal{C}^0$ (the gradient of the maximum is not continuous, since the maximum can jump between two local maxima) and (ii) only the global maximum can be observed. Hence, none of the local maxima will be taken into account, although they may also reflect a constraint violation. During the optimization process, the constraint is likely to switch abruptly from a maximum to another, and this switching may influence the convergence property of the solver, especially if the switching occurs for maxima in opposite directions.

$$g_{\max} = \max_{t \in [t_g^I, t_g^E]} g(\mathbf{x}, t) \tag{3.18}$$

It is necessary to take into account the value of the constraints at interpolation times in order to avoid discontinuities. This issue can be highlighted using a single task with a constant gain and whose insertion is not smoothed. The resulting control decreases monotonously: the maximal value is reached at the insertion of the task, which may not happen at a time aligned with the grid. Thus, if the value of the constraint at the interpolation points is not taken into account, the value considered will be the one observed at the next step of the grid, which is inferior to the value reached at the insertion. The Fig. 3.7 shows the gap between the two values obtained with and without considering the interpolation points. In the former case, $g_{\max} = 1$, while in the latter case $g_{\max} = 0.915$.

As a result, when the interpolation points are not taken into account, the constraint of maximal velocity has a serrated and periodic evolution with respect to the time of insertion of the task (the period is the span of the grid). The correct value is only obtained when the time of insertion is aligned with the time of evaluation. The Fig. 3.8 shows the evolution of the constraint of maximal velocity for a task whose time of insertion varies. The interpolation points are not taken into account and the span of the grid is fixed to 5 ms.

Sum of the violation $g_{\Sigma,k}$ In order to implicitly take into account the multiple constraint violations (for a single set of constraints), we propose to evaluate the constraints in a single value, $g_{\Sigma,k}$, defined as follows: if the constraint is always satisfied, then it is the higher value of $g(t)$, so $\max(g(t))$. Otherwise, it is the area of violation, i.e. the area covered by the curve $t \to \max(0, g(t))$ (cf. Fig. 3.9). This method is also $\mathcal{C}^0$: if the constraint always has a negative value, it behaves like the previous method. Also, since this method behaves as a $\max$ operation when the constraint is not violated, it is mandatory to take into account the constraint value at interpolation points (for the same reason).

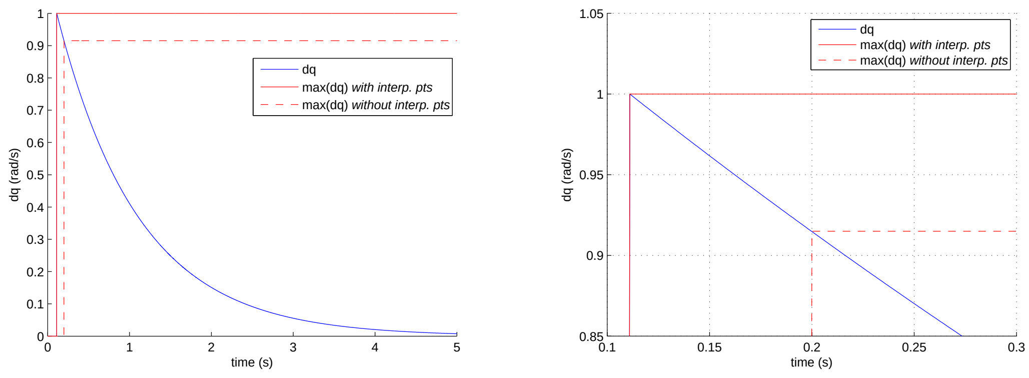


Figure 3.7: a) Evolution of the control for a simple task (with a constant gain), and the associated constraint $g_{\max}$ (with a grid span of 0.2s). The red plain (resp. dashed) curve represents the evolution of the constraint $g_{\max}$ with (resp. without) taking into account interpolation points. b) A closer view of the same simulation for $t \in [0.1, 0.3]$.

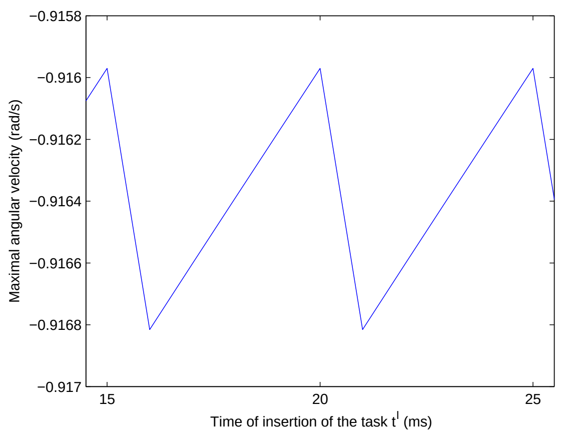


Figure 3.8: Discontinuous evolution of the constraint of maximal velocity with respect to the time of insertion of the task when the interpolation points are not taken into account. The grid span is fixed to 5ms.

In the following, the penalty associated to a violation is weighted by a user-defined coefficient k, in order to observe the influence it has on the convergence properties.

$$g_{\Sigma,k}(t+\delta t) = \begin{cases} \text{if } \; g(t) < 0, & \max(g(t), g_{\Sigma,k}(t)) \\ \text{else}, & g_{\Sigma,k}(t) + k\delta t \max(g(t), 0) \end{cases} \tag{3.19}$$

where δt is the integration step.

3.4.3.3 Evaluation on a family of subintervals

The two previous solutions only considered one observation of the constraint violation for the entire time interval. Instead of a single evaluation for the interval $[t_g^I, t_g^E]$, a set of time inter-

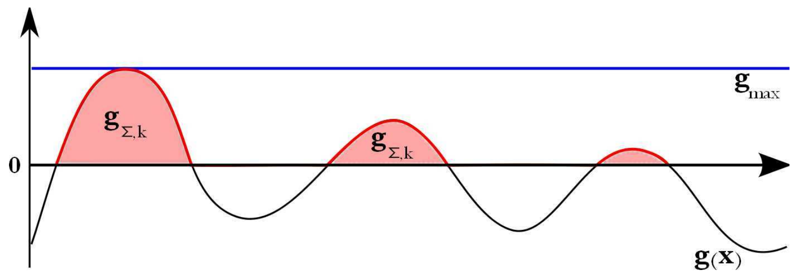


Figure 3.9: Definition of the area of violation $g_{\Sigma,k}$. In this case the total value correspond to the sum of the three red areas.

vals $T_{1\cdots n}$ such as $[t_g^I, t_g^E] \subseteq \bigcup_{i\in[1,n]} T_i$ could be used, and the constraints could be evaluated on each interval T_i using the maximum or the sum of the constraint violation as described above. Yet, as noticed earlier, defining intervals by fixed times can be a problem (e.g. the time interval may not be evaluated any more if the simulation ended first). Therefore, we decided to define the time interval relatively to discrete events.

Constraint by task $g_{\max/\mathbf{e}}$ Associating the robot constraints $\phi(\mathbf{q}, \dot{\mathbf{q}})$ to the whole simulation can raise an issue: a violated constraint cannot be easily linked to the "responsible" task. This is even more difficult considering that there is task overlapping, and that several tasks can realize a same action. For example, if there is a violation of the constraint of maximal joint velocity for a joint of the left arm, the best conclusion possible is that one of the tasks involving the left arm (or several of them) has violated the constraint, but it is impossible to know which one(s) precisely.

In order to compensate this problem, we consider $n_\mathbf{e}$ additional sets of constraint $\phi(\mathbf{q}, \dot{\mathbf{q}})$, noted $\phi_i(\mathbf{q}, \dot{\mathbf{q}}), i \in [1 \ldots n_\mathbf{e}]$, (with $n_\mathbf{e}$ the number of tasks in the sequence). Each set $\phi_i(\mathbf{q}, \dot{\mathbf{q}})$ is computed only when the task $\mathbf{e_i}$ is in the stack (during the time interval $[t_i^I, t_i^F]$, cf. Fig. 3.10). The optimization problem contains thus $n_\mathbf{e}+1$ sets of semi-infinite constraints: one per task and one covering the whole sequence, so as to take into account the constraints when the stack of tasks is empty.

Constraint by set of tasks This approach is similar to the previous one, except that an evaluation is associated to each ordered combination of tasks liable to describe the state of the stack-of-tasks controller during the simulation. If the number of tasks is important and no extra information is given on their respective priority, this method will be inappropriate since the number of cases to consider will be combinatorial, and an important part of the envisaged combinations will not appear during the simulation.

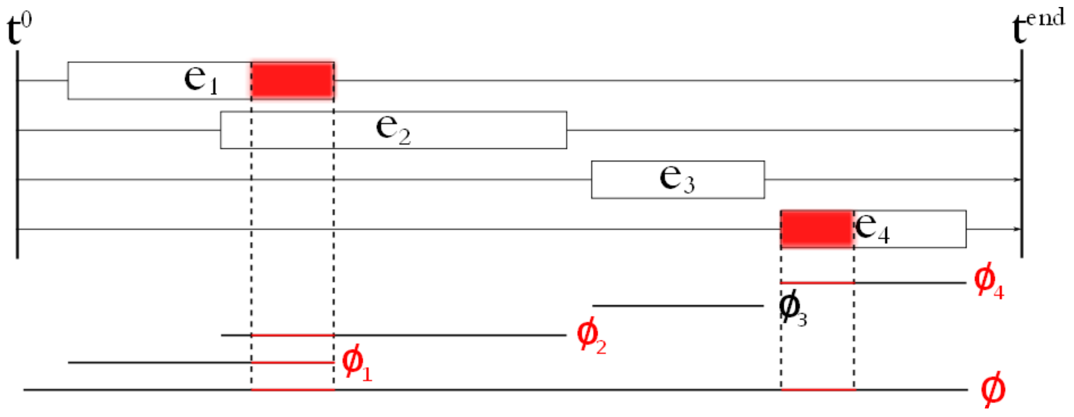


Figure 3.10: Assigning an additional set of robot constraints $\phi_i(\mathbf{q}, \dot{\mathbf{q}})$ to each task enables to track more easily the task responsible of the violation of a constraint than using solely a constraint set for the whole motion. Red zones of $\mathbf{e_1}$ and $\mathbf{e_4}$ represent constraint violations during the task. These violations appear on the additional set of robot constraints ϕ_1, ϕ_2 and ϕ_4 and make the research of the responsible task easier. In case of task overlapping, a set of responsible tasks is found.

3.4.4 Comparison of the constraint handlers

3.4.4.1 Comparison of methods $g_{\max}$ and $g_{\Sigma,k}$

These two methods are compared using a function that exhibits several maxima. Let $f_\mathbf{x}$ be a third order polynomial function, $f_\mathbf{x}(t) : t \to \sum_{i=0}^{3} \mathbf{x}_i t^i$ and $F(\mathbf{x}) = \int_0^1 f_\mathbf{x}(t)dt$ be its primitive. The criterion is the maximization of the area. The considered optimization problem has a semi-infinite form:

$$\begin{aligned} &\max_{a_i} \int_0^1 f_\mathbf{x}(t)dt \\ &\text{sc. } \forall t \in [0,1], f_\mathbf{x}(t) - 1 \leq 0 \end{aligned} \tag{3.20}$$

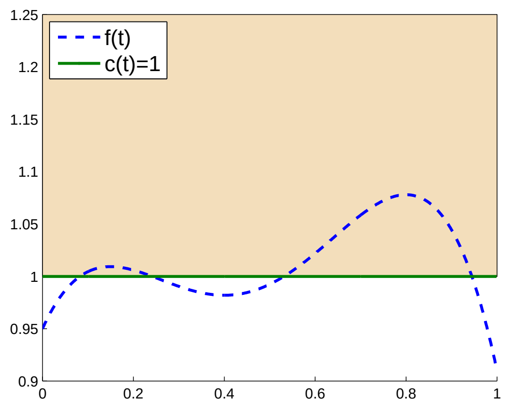


Figure 3.11: Graphic representation of the optimization problem. The criterion is to maximize the area of the zone under the blue curve. The orange zone represents the forbidden zone.

The Fig. 3.11 illustrates the optimization problem. The solution is the set $\mathbf{x}^* = [1, 0, 0, 0]$, that defines a constant function $f_{\mathbf{x}^*}(t) = 1$, such that $F(\mathbf{x}^*) = 1$.

The optimization is performed using the SQP solver of Matlab. The gradient of the criterion and the gradient of the constraint are estimated by finite differences.

Constraint type	Iterations	Time spent	$F(\tilde{\mathbf{x}}) - F(\mathbf{x}^*)$
$g_{\max}$	65	22 s 44	$3.299.10^{-3}$
$g_{\Sigma,\frac{1}{4\delta t}}$	305	1 min 25 s 00	$0.275.10^{-3}$
$g_{\Sigma,\frac{1}{2\delta t}}$	61	21 s 03	$8.800.10^{-3}$
$g_{\Sigma,\frac{1}{\delta t}}$	362	1 min 57 s 09	$0.618.10^{-3}$
$g_{\Sigma,\frac{2}{\delta t}}$	119	31 s 90	$8.682.10^{-3}$
$g_{\Sigma,\frac{4}{\delta t}}$	163	37 s 47	$4.075.10^{-3}$
$g_{\Sigma,\frac{5}{\delta t}}$	81	23 s 14	$8.741.10^{-3}$
$g_{\Sigma,\frac{10}{\delta t}}$	107	24 s 37	$8.740.10^{-3}$
$g_{\Sigma,\frac{50}{\delta t}}$	21	24 s 90	$8.739.10^{-3}$

Table 3.3: Comparison of convergence results using the methods $g_{\max}$ and $g_{\Sigma,k}$.

Table 3.3 compares the results of the optimization considering different type of constraints. The optimization process stops without finding the optimal solution with the constraint $g_{\max}$; which was expected. Unfortunately, the method $g_{\Sigma,k}$ does not seem to converge to the optimal solution neither, whatever is the value of the parameter k. Results of this test suggest that for small values of k, the approximation found was better than the one found with $g_{\max}$, but the convergence towards a solution is too slow to be applicable on more complex case studies (such as ours).

This is the reason why we finally adopted $g_{\max}$ as the evaluation method in solving our problem.

3.4.4.2 Comparison of methods using the simple $\max$ and the $\max$ by intervals

These two methods are compared using a sequence realizing a back and forth movement of an arm, represented by a chain of tasks (Fig. 3.12). In this chain, the tasks $\mathbf{e_{1a}}$, $\mathbf{e_{2a}}$ and $\mathbf{e_{3a}}$ (resp. $\mathbf{e_{1b}}$, $\mathbf{e_{2b}}$ and $\mathbf{e_{3b}}$) have identical definitions of the error and the Jacobian. The only constraints considered are the joint velocity bounds (the tasks are defined so that joint position bounds hold and collisions are avoided).

As mentioned earlier, using only a single evaluation of the maximum constraint on the whole sequence may cause convergence difficulties. Considering more constraints gives a more precise overview of the evolution of the control and of the constraints, providing a better result, as shown in Table 3.4.

In the following, the Semi-Infinite Constraints will be handled with the method $g_{\max/\mathbf{e}}$: a set of the considered Semi-Infinite Constraints is associated to each task and evaluated only

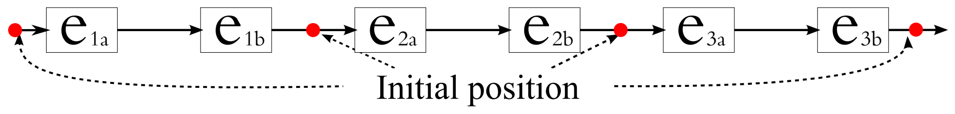


Figure 3.12: Sequence corresponding to a back-and-forth motion.

Constraint type	Constraints number	Optimization duration	iterations	$t^{\text{End}}(s)$
$g_{\max}$	103	1 h 21 min 25 s	45	34.20
$g_{\max/\mathbf{e}}$	583	1 h 38 min 10 s	65	22.44

Table 3.4: Comparison of optimization convergence during a back-and-forth motion, considering either a single value for the constraint ($g_{\max}$) or a constraint for each task ($g_{\max/\mathbf{e}}$).

when the task is active. The resulting constraint corresponds to the maximal value for the time interval $[t_i^I, t_i^F]$.

3.5 Optimization via simulation

3.5.1 Solver used

The optimization problem considered here is a non-linear constrained parametric problem, with continuous criteria and constraints, whose gradients are not provided (formulating the gradient of the constraints functions with respect to parameters appears to be particularly difficult). Even if the computation of the control law is smooth, this problem is only $\mathcal{C}^0$ due to the method chosen to handle the semi-infinite constraints (based on the $\max$ operator).

Depending on the properties, the type, and the data available of an optimization problem, some solvers are more suitable comparing to others. To only cite a few:

- Ipopt[2] [89] and Loqo [87] both solve constrained optimization problems with twice continuously differentiable ($\mathcal{C}^2$) criterion and constraints. The gradient must be provided but the Hessian can be approximated.

- The MATLAB optimization toolbox [15] and CFSQP[3] [52] propose Sequential Quadratic Programming (SQP) algorithms. Both assume that the criterion and the constraints are smooth, and may realize an approximation of the gradients if they are not provided.

- Solvopt[4] [47] handles non-linear non-smooth problems.

Considering that the optimization problem is only $\mathcal{C}^0$, Solvopt seems to be the only suitable solver among the list we described. Yet, despite the potential derivative discontinuities

[2]**I**nterior **P**oint **opt**imizer
[3]C code for Feasible Sequential Quadratic Programming
[4]**Solv**er for local **opt**imization problems

of some constraints, the solver CFSQP converges towards satisfying solutions that are better than those found by Solvopt. Since SQP solvers give better performance when the gradients of the constraints and the criteria are given, they are approximated by finite difference, even if this approximation is time consuming.

Hence, the solver used is CFSQP.

3.5.2 Dialog solver - simulator

At each optimization step, the solver chooses a new set of parameters $\mathbf{x}$. The corresponding constraints are then evaluated. As stated in Section 3.3.3, only few of them can be evaluated directly, since they correspond to a linear function of the vector of parameters $\mathbf{x}$. In order to evaluate the other ones (especially the constraints relative to the robot), it is required to realize a complete simulation of the task sequence. As a result, the chosen value for the current optimization variable vector $\mathbf{x}$ is transmitted by the optimization solver to the simulation engine, which returns the evaluation of the constraints. The optimization solver then computes a new step vector $\mathbf{x}$, until convergence.

3.5.3 Simulation

The computation of the control for a given hierarchy of tasks is detailed in Chapter 2. The simulation is basically a numerical integration of this equation, using an explicit Euler integration method with a fixed step. The entry and exit times of each task t_i^I and t_i^F are continuous variables that are not aligned with the grid. These instants correspond to discrete events that create a modification in the control. Aligning these instants to the next time step will create discontinuities in the control evolution with respect to events. To solve this problem, the contentious time t_a is added as an integration point during the time interval $[t, t+\delta t]$, splitting it into the two smaller ones $[t, t_a]$ and $[t_a, t+\delta t]$.

```
**Initialization**
$[t_1^I, t_1^F, \ldots, t_n^I, t_n^F, t^{\text{End}}] = \texttt{computeTimes}(\mathbf{x})$
$t_{\text{Sim}}^{\text{End}} = \max_i \left(t_i^F\right)$
$t = 0$
**while** $(t \leq \max(t^{End}, t_{Sim}^{End}))$ **do**
    $\dot{\mathbf{q}}(t) = \texttt{SoT}(\mathbf{q}(t), t)$
    $\delta t' = \texttt{findTimeStep}(t, \delta t)$
    $g_j(\mathbf{x}, t) = \texttt{computeConstraints}(t, \delta t')$
    $(t, \mathbf{q}(t+\delta t')) = \texttt{integrate}(t, \delta t', \mathbf{q}(t), \dot{\mathbf{q}}(t))$
**end**
```

Algorithm 1: Tasks sequencing simulation.

The algorithm 1 describes the simulation. The functions used are:

- `computeTimes` computes the absolute times using the relative times.

- `SoT` computes the control induced by the tasks execution.
- `computeConstraints` evaluates the constraints.
- `findTimeStep` computes the required time step for the Euler integration: δt, or a smaller one if needed, due to the need of splitting this interval in two.
- `integrate` updates the simulated objects or processes.

3.5.4 Enhancements of the optimization process

To reduce the time taken by the optimization process, the two following methods were used:

3.5.4.1 Scaling

The problem includes constraints that are not homogeneous (times, angles, velocities, distances…) and do not work on the same scale. As a result, they do not have the same influence during the optimization process. In order to reduce these scale differences, a normalization based on the value of the constraints obtained by the execution the initial set of parameters $\mathbf{x}_0$ is realized. The effects of this scaling operation is studied for the four next sequences:

1. A sequence containing only one task;
2. A sequence where a task has to be realized during the maintaining period of another (cf. Fig. 3.13a);
3. A sequence realizing simultaneously three conflicting positioning tasks (cf. Fig. 3.13b). $\mathbf{e_H}$ defines the position of the head, $\mathbf{e_L}$ the position of the left arm, $\mathbf{e_R}$ the position of the right arm. The three tasks share the waist joint. The position of each task with respect to the other ones is fixed arbitrarily;
4. The back-and-forth motion illustrated on Fig. 3.12.

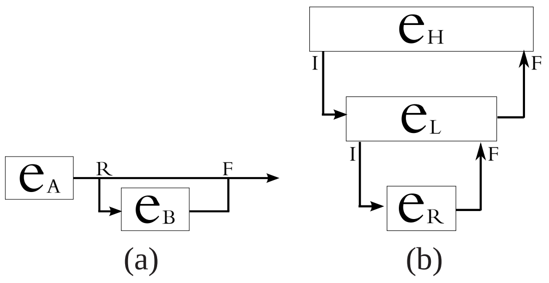


Figure 3.13: a. Sequence where one task has to be achieved during the maintaining period $[t^R, t^F]$ *of another one (left). b. Sequence where three tasks are realized simultaneously (right).*

The results are shown on Table 3.5. For each sequence, an optimization is run only on the timing (the gains of each task are fixed) and another one is run on both the timing and gains of each task. To compare the results, the first element considered is the result of the optimization (namely the time of end of the entire simulation t^{End}), and the second is the number of iterations, i.e. the convergence speed. For optimizations realized only on the timing of the tasks, the result is generally the same (only the fourth sequence is a little less optimal), but it is obtained in fewer iterations. For optimizations realized on both the timing and the gain of the tasks, the solution found is better, but the number of iterations realized is bigger. Hence, generally, the optimization process converges either faster or towards a better solution when using the scaling operation.

Sequence number	Gains	Scaling	Optimization duration	Number of iterations	$t^{\text{End}}(s)$
1	Fixed	off	4 min 37 s	30	43
		on	2 min 34 s	**17**	43
1	Optimized	off	5 min 02 s	**39**	9.76
		on	5 min 23 s	42	9.76
2	Fixed	off	10 min 46 s	54	36.31
		on	10 min 09 s	**44**	36.31
2	Optimized	off	15 min 39 s	43	22.23
		on	14 min 27 s	44	**18.73**
3	Fixed	off	17 min 38 s	71	14.03
		on	16 min 48 s	**61**	14.03
3	Optimized	off	1 h 04 min 56 s	23	102.67
		on	1 h 54 min 25 s	79	**32.44**
4	Fixed	off	1 h 02 min 46 s	49	77.00
		on	56 min 09 s	**43**	77.05
4	Optimized	off	1 h 06 min 43 s	33	33.73
		on	1 h 38 min 10 s	65	**22.44**

Table 3.5: Comparison of optimization final state with and without scaling the constraints.

3.5.4.2 Group of tasks

The complexity of an optimization on the entire sequence of tasks increases more than linearly with the number of tasks, thus it is beneficial to split the whole sequence into smaller ones. This cut can not be realized everywhere: due to the possible overlay between tasks, it is only possible to split a sequence in two if a task of one sub-sequence can never be realized at the same time than a task of the other sub-sequence.

The sequence represented on Fig. 3.14 is composed of three groups realized sequentially. Each of these groups contains two tasks realized simultaneously, due to the time constraints.

The Table 3.6 shows the results obtained while realizing the optimization on the entire task sequence and while realizing three separate runs on each group. The comparison is now first realized on the result of the optimization, and second on the computation time, since

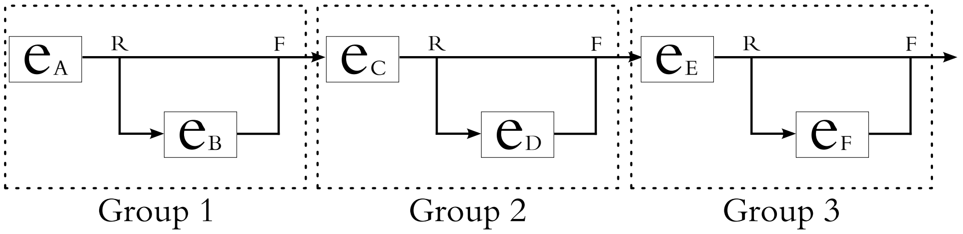


Figure 3.14: A sequence split into three groups.

	Optimization duration	Number of iterations	t^{End} (s)
Group 1	20 min 47 s	75	15.90
Group 2	24 min 51 s	105	12.69
Group 3	22 min 58 s	70	18.08
Total	1 h 08 min 36 s	/	46.67
Full sequence	2 h 24 min 21 s	59	49.42

Table 3.6: Comparison between an optimization realized on an entire sequence and three optimizations run on the corresponding subsequences.

the number of iterations cannot be used to compare the two systems. Even if the whole duration of the sequence has been reduced, the most notable improvement is the diminution in computation time. Indeed, since the number of tasks in the sequence is reduced, the time taken by the evaluation of the gradient by finite difference is also reduced.

3.6 Discussion

This formulation of the optimization problem raises two main issues, both related to the continuity of the task sequence, and are worth to be noted:

3.6.1 *A posteriori* evaluation of the constraints

The first issue is that the mechanisms of insertion and removal only depend on the time: the condition of regulation of a task is evaluated *a posteriori*. In other words, the task may be removed too soon (nothing proves that a task had been regulated at t_i^F), causing a distortion of the whole task sequence (it is the case of the pick-and-place sequence, when the picking task is not achieved correctly). This kind of incoherent schedule may be requested by the solver, e.g. during the line search, and the criterion and the constraints computed may not be meaningful.

In order to remove a task, two conditions (at least) must be satisfied: (i) the task has been regulated ($t \geq t^R$), and (ii) the removal time has been reached or exceeded ($t \geq t^F$). These two conditions are not similar: (i) expresses the fact that the removal time of a task is independent of its regulation time, and (ii) recalls that a task should not necessarily be removed as soon as the task is regulated, and may be maintained in the stack.

3.6.2 Discontinuity due to discrete events

The second issue is due to the discrete events. Ensuring the continuity of a task sequence with respect to the time for a given set of parameters $\mathbf{x}$ is doable at low cost: the continuity of the control is ensured by using a damped inverse in case of ill-conditioning, while the continuity during discrete events can be ensured by using the adequate smoothing operation, as shown in previous chapter.

Yet, as shown in section 2.2.6, this does not ensures the continuity with respect to the optimization parameters. To sum up, $f_{\mathbf{x}} : t \rightarrow (f_{\mathbf{x}}(t) = \dot{\mathbf{q}}(t))$ is continuous for a given $\mathbf{x}$, while $f : \mathbf{x} \rightarrow (f_{\mathbf{x}} : t \rightarrow \dot{\mathbf{q}}(t))$ is not. Indeed, the smoothing operation is likely to create a discontinuity of the control with respect to these events. Depending on whether a smoothing operation has to be realized or not, the shape of the control law may accuse strong differences (cf. Fig. 2.14).

The safest way to ensure that these events will not create any discontinuity is to impose the order of the events. Imposing only the priority between tasks is not enough, since it is the order of events that makes a swap of priority of tasks required, and this order is conditioned by the position of the tasks. This approach is safe, but too restrictive: for compatible tasks, the swap is instantaneous (no smoothing operation is required), hence whatever the respective positions of each task, the control law will remain continuous. As a result, determining which sequence of task give the smaller ending time for the sequence is a problem that cannot be done directly using the proposed optimization problem.

3.7 Conclusion

In this chapter we presented a method to perform task overlapping by formulating it as an optimization problem. Task overlapping aims at reducing the duration of a sequence and smoothing the robot motion, by finding the best timing and gains for each task. The optimization is solved using Sequential Quadratic Programming algorithm, and the handling of the semi-infinite constraints (inherent to simulations depending upon time) has been defined. The flaws of this method have been mentioned: first, the consequent computation time of the optimization process prevents the use of this method in real-time, and second, this method optimizes the sequence while respecting the initial order of events, although reordering them can lead to a better solution. There are rooms of investigations that will be discussed in the thesis concluding section.

Next chapter presents the tools used to realize the optimization, especially the simulation engine used to simulate the environment for a given set of parameters and compute the constraints. It completes the results of this chapter by studying the effects of the smoothing of the control law on the optimization process, and presents an experimental application of this method on the humanoid robot HRP-2.

Chapter 4

Simulations and Experimentations

The previous chapter closes our theoretical contributions. In this chapter, we gather the simulation and experimentations that have been achieved for larger scale scenarios in order to assess and validate the theoretical results. We also detail the technical parts of the software developments.

First, we describe briefly the simulation tools used, namely the inverse kinematic framework (StackOfTasks) and the dynamic simulator (AMELIF), and detail the contributions realized in both of them in the frame of this thesis. Then, we study the influence of the control law smoothing on the convergence properties of the optimization that leads to task overlapping process. Finally, a real scenario is experimented with the HRP-2 humanoid robot. We took an existing scenario (take a can from a fridge) that was previously addressed in a classical sequencing way and enhanced it using our proposed method.

4.1 Implementation of the inverse kinematic

The control framework used to implement the stack-of-tasks mechanism described in Chapter 2 is related in [64]. In the following, it is named SoT as an acronym for StackOfTasks, so as to make a distinction between the software framework and the theoretical stack-of-tasks mechanism.

4.1.1 Presentation of the framework

The software is organized by entities and signals, similarly to the mechanism of boxes and links in Simulink [17]. Each output signal is parameterized by a set of dependencies –input signals– that must be known or preliminarily computed in order to compute the value of the output signal at each time step. The SoT is structured as an oriented-graph. Thanks to this structure it is possible to update only the necessary signals that are induced from a request of a given (other) signal (i.e. those on which the requested signal depends on) instead of updating the entire system, thus reducing the computation cost.

Each "task" entity defines a signal error $\mathbf{e}$ that corresponds to the difference between a signal $\mathbf{s}$ and its desired value $\mathbf{s}^*$ (see Fig. 4.1). This error is used to compute the desired behavior $\dot{\mathbf{e}}^*$, but can also be used by other entities, for example to compute the value of the adaptive gain $\lambda(\mathbf{e})$.

The Fig. 4.1 is a simplified representation of the entities corresponding to a task and an adaptive gain. Both of them are defined by a set of methods and a set of input and output signals. The dependencies between the entities correspond to the links between these signals. An output signal can be linked to several input signals, but an input signal can only be linked to one output signal. For example, the "task" entity defines, computes and shares with the other entities the task error $\mathbf{e}$ as an output signal; it can then be used as an input signal by the "AdaptiveGain" entity.

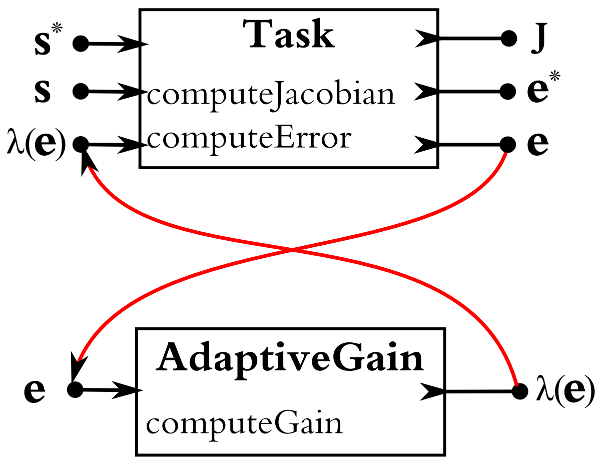


Figure 4.1: Simplified representation of the SoT: two entities, a task and an adaptive gain, communicate via the system of input and output signals, plugged together.

The description of the operations that are required to update a signal is a good way to illustrate the mechanism of the SoT. For example, the update of the desired behavior $\dot{\mathbf{e}}^* = \mathbf{e}\lambda(\mathbf{e})$ is decomposed into the set of following actions:

Update $\dot{\mathbf{e}}^* = \mathbf{e}\lambda(\mathbf{e})$
- $\rightarrow$ Update $\mathbf{e}$
  - $\rightarrow$ Update s
  - $\rightarrow$ Update s^*
  - $\rightarrow$ Execute *computeError*
- $\rightarrow$ Update $\lambda(\mathbf{e})$
  - $\rightarrow$ Update $\mathbf{e}$ (which does nothing this time, since it is already updated)
  - $\rightarrow$ Execute *computeGain*

This framework is implemented in C++, but the manipulation of the entities can be achieved using a scripting interface. This interface allows the user to load the libraries that define the entities, to create and destroy instances of these entities, to plug or unplug signals and to manually access their parameter values (read and write). A CORBA server is also provided to wrap the script commands and communicate with the sensors and the robot we are using for the experiments.

4.1.2 Implementation of the smooth control

The control law described in Chapter 2 is implemented within the SoT framework. We briefly address hereafter its implementation, along with the validation on a real-size example.

4.1.2.1 Characteristics

As a reminder, the characteristics and rules of the smooth control law are as follows:

- The control law is computed with the damped-inverse approach presented in (2.8), so as to be robust to singularities:

$$\mathbf{\dot{q}_i} = \mathbf{\dot{q}_{i-1}} + (\mathbf{J_i P_{i-1}})^{\dagger}(\mathbf{\dot{e}_i^*} - \mathbf{J_i \dot{q}_{i-1}})$$

- A task can only be inserted and removed at the end of the stack. This process is realized using an insertion gain λ^{Ins} (2.26):

$$\mathbf{\dot{q}_{n+1}} = \mathbf{\dot{q}_n} + \lambda^{Ins}(\mathbf{J_{n+1} P_n})^{\dagger}(\mathbf{\dot{e}_{n+1}^*} - \mathbf{J_{n+1} \dot{q}_n})$$

- The swap of priority between two tasks is made only through successive swaps between two adjacent tasks (in the SoT). It can be instantaneous if the two tasks do not conflict (i.e. $\mathbf{\dot{q}_{[A|B]}} = \mathbf{\dot{q}_{[B|A]}}$), either way it is necessary to smooth the swap by realizing a linear interpolation between the two control laws $\mathbf{\dot{q}_{[A|B]}}$ and $\mathbf{\dot{q}_{[B|A]}}$ (2.25):

$$\mathbf{\dot{q}} = \alpha \mathbf{\dot{q}_{[A|B]}} + (1-\alpha)\mathbf{\dot{q}_{[A|B]}}$$

In practice, we realize at most one discrete operation by task and by time step (e.g. a task cannot be swapped twice during the same time step). As a result, moving a task from the top to the end of a stack containing n tasks (including the task moved) lasts $(n-1)$ time steps.

The function α used to realize the smoothing grows continuously between 0 to 1 and is defined by:

$$\alpha(t) = \frac{1}{2} - \frac{1}{2}\cos\Big(d(t - t_\alpha^I)\Big) \tag{4.1}$$

This function is only defined on the interval $[t_\alpha^I, t_\alpha^I + \frac{\pi}{d}]$. t_α^I corresponds to the start of the smoothing period, and d is a user-defined value that fixes the duration of the transition $\Delta t = \frac{\pi}{d}$.

4.1.2.2 Example

The following example compares the evolution of the classic control law and the smoothed control law for a sequence of tasks realized on the HRP-2 robot. In this sequence, three positioning tasks are inserted and removed by order of insertion (the first task inserted is the first removed). The three tasks considered and their respective timings are:

Task	Description	Timing
$\mathbf{e_L}$	"Move the left gripper"	[0.0s, 2.0s]
$\mathbf{e_R}$	"Move the right gripper"	[0.5s, 2.5s]
$\mathbf{e_G}$	"Open the right gripper"	[1.0s, 3.0s]

The tasks $\mathbf{e_L}$ and $\mathbf{e_R}$ are coupled (they share the waist joint), but the task $\mathbf{e_G}$ is perfectly decoupled from the two of them (i.e. there is no dof in common).

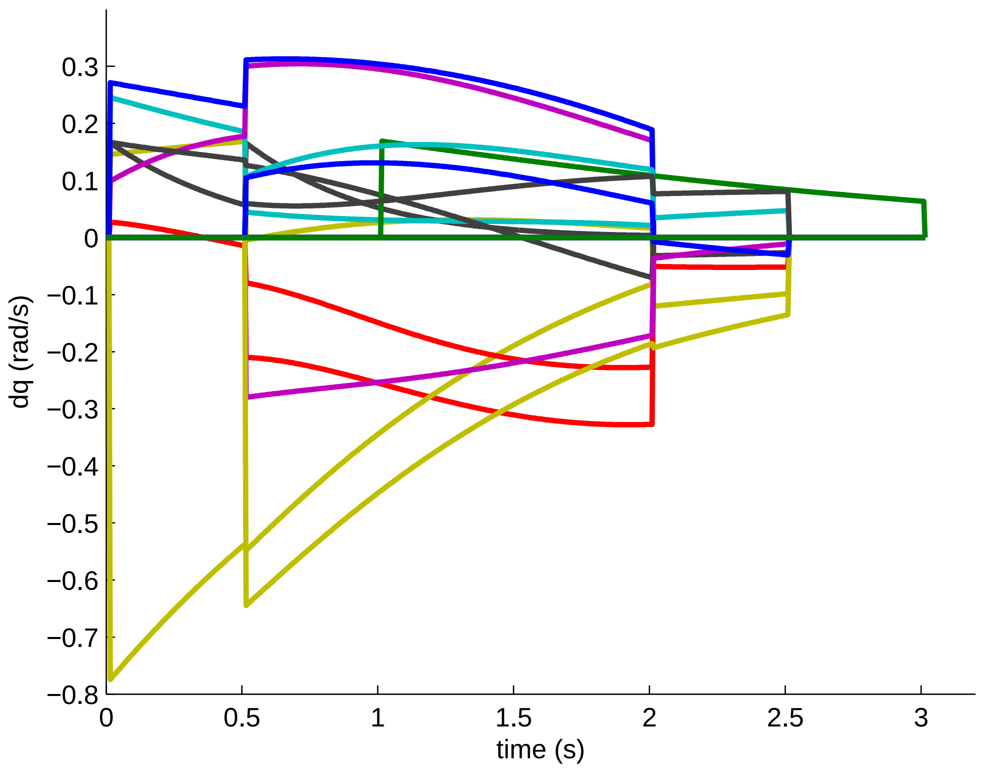
The Fig. 4.2 represents the evolution of the control without smoothing. The evolution of the control is smooth (thanks to the damped inverse), except at each insertion or removal, that creates a discontinuity in the control. The Fig. 4.3 represents the evolution of the control when each task is inserted and removed smoothly. Yellow areas represent the insertion period of the three tasks. Since each task is inserted at the end of the stack, this insertion process starts immediately, and the duration of this process is the same for the three tasks (0.8s). Similarly, the removal process of the task $\mathbf{e_G}$ starts as soon as the order is given (at t_G^F=3.0s), since it is the only task in the stack at that moment (hence it is at the end of the stack).

At the opposite, the removal processes of the tasks $\mathbf{e_R}$ and $\mathbf{e_L}$ cannot start immediately. First, they have to be moved to the end of the stack. For the task $\mathbf{e_G}$, this process is decomposed into two steps: first, a swap between the tasks $\mathbf{e_G}$ and $\mathbf{e_R}$, and second, a swap between $\mathbf{e_G}$ and $\mathbf{e_L}$. The first swap cannot be realized instantaneously, since the two positioning tasks are conflicting: a linear interpolation has to be realized during the period [2.0s, 2.08s] (represented by the green area). At the opposite, the second swap, between $\mathbf{e_R}$ and $\mathbf{e_G}$, can be realized instantaneously.

It is important to notice that because of the smoothing processes, the actual removal time of each task is more or less delayed. The removal of the task $\mathbf{e_G}$ is delayed of 0.08s (the minimal delay possible, conditioned by d), the removal of the task $\mathbf{e_L}$ is delayed of 0.0805s (the additional five milliseconds correspond to the instantaneous swap with $\mathbf{e_G}$, that lasts one time step) and the removal of the task $\mathbf{e_R}$ is delayed of 0.165s, because of the swapping processes.

4.1.2.3 Issues of the smoothing process

This additional delay taken by the transition process can become critical when there are too many tasks in the stack. For example, the delay taken to replace the task of highest priority by a new one (not yet inserted in the stack), is in the worst case equal to $2n\Delta t$, where n is the number of tasks currently in the stack of tasks. Indeed, first, the top-priority task must be put at the end of the stack, which costs $(n-1)$ swap operations, then it can be smoothly



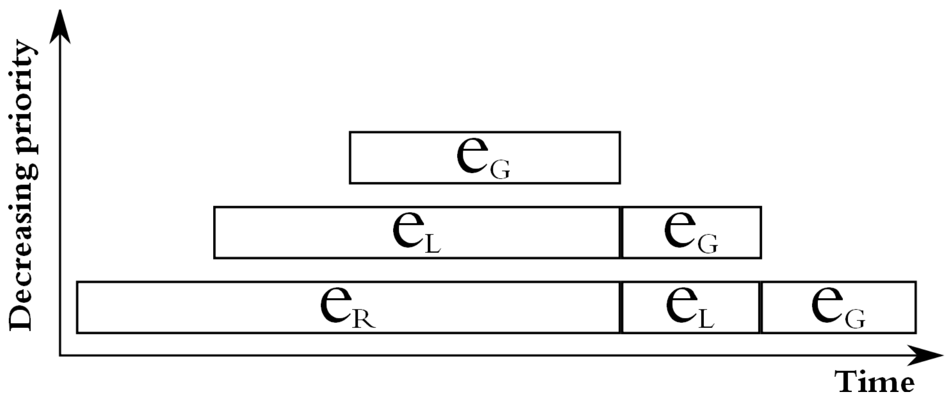
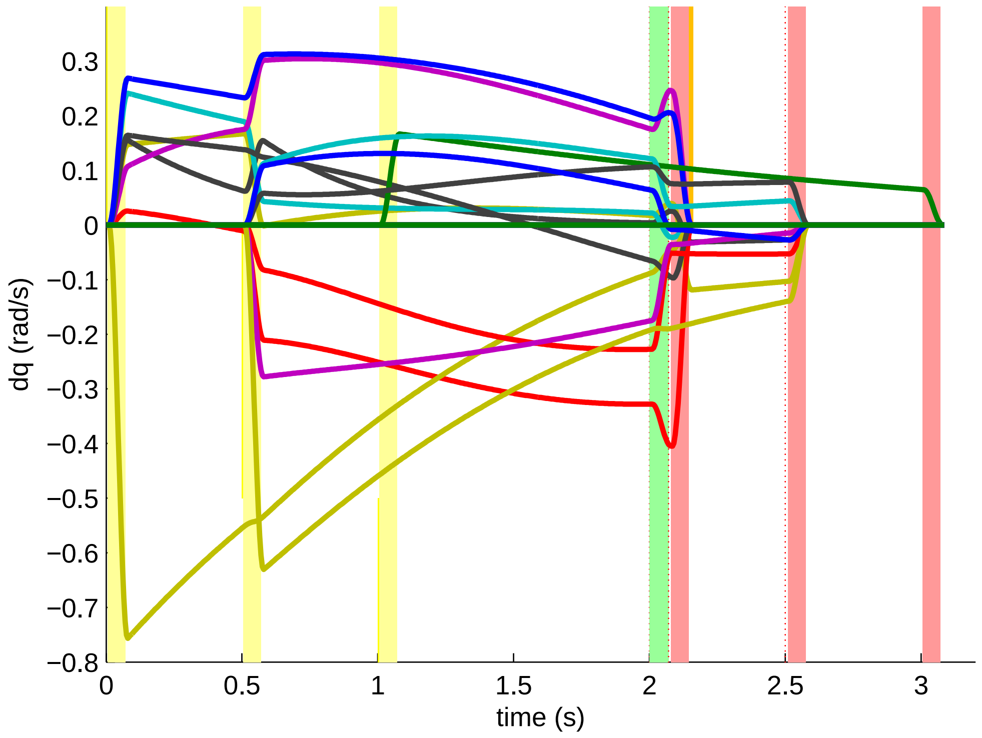


Figure 4.2: Evolution of the classic control law (without the smoothing process). The upper graph represents the evolution of the control law $\dot{q}$*. The lower graph represents the state of the task during the simulation.*

removed, and only then the new task can be inserted and placed to the top of the stack, which also costs $(n-1)$ swap operations.

As a result, smoothing the control law may cost a loss of reactivity. Nevertheless, in some configurations, the user will prefer to have important variations in the control rather than spending too much time to ensure that it is smooth. This is the case when the humanoid robot HRP-2 has to track a posture-based trajectory computed by the contact planner [24].



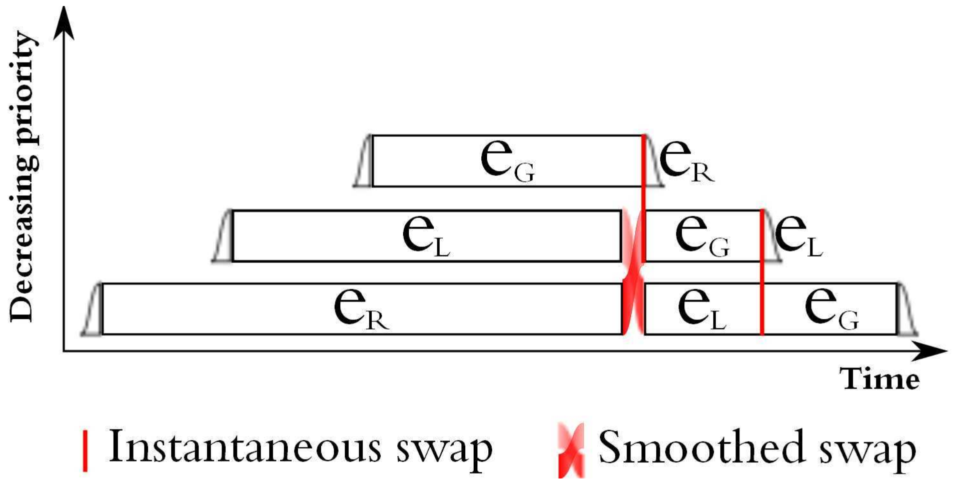


Figure 4.3: Evolution of the smooth control law. The upper graph represents the evolution of the control law $\dot{\mathbf{q}}$. Yellow areas represent insertion periods. Red areas represent removal periods. The green area represents the interval of time during which the swap between $\mathbf{e_R}$ and $\mathbf{e_L}$ is realized. The dotted lines represent the times where a swap is requested. The lower graph represents the state of the task during the simulation.

To maintain the equilibrium of the robot, a stabilizer that can be considered as a high priority task, is used. Yet, this stabilizer –conceived to close the loop of the preview control walking approach of the humanoid robot– should not be activated when the robot leaned on objects of the environment with the upper body (i.e. when additional contacts occur on the body of the robot). As a result, during the entire simulation of an acyclic motion, the stabilizing task is continuously switched 'on' and 'off' and 'on' again according to the posture of the robot and its contact configuration with the environment. This is typically a case where the reactivity is preferred to the smoothness of the trajectory.

The second issue, already mentioned, is the continuity of the trajectory. Indeed, the control law is now guaranteed to be continuous with respect to the time variable, whatever the insertion and removal times are. However, when considering the trajectory of the robot as a function of the insertion time, we have, mathematically speaking:

$$\mathcal{F} : t^{I,F} \rightarrow (q : t \rightarrow q(t)) \tag{4.2}$$

with $t^{I,F} = (t_1^I, t_1^F, \cdots, t_n^I, t_n^F) \in \mathcal{R}^{2n}$ the finite set of insertion and removal times. For each set of continuous parameters, a different trajectory (as a continuous $\mathcal{C}^1$ function of time) is obtained. Then, it might happen than the mapping $\mathcal{F}$ is not continuous. In particular, when the insertion time of two tasks swaps, passing from $t_i^I < t_j^I$ to $t_j^I < t_i^I$, then the trajectory mapping $\mathcal{F}$ is discontinuous. This is due to the additional delay mentioned above: when $t_i^I < t_j^I$, the task $\mathbf{e_j}$ is delayed due to the insertion of the task i, while, suddenly, when $t_j^I < t_i^I$, it is the task $\mathbf{e_i}$ that is delayed.

This second issue appeared to be problematic in the optimization process (i.e. to the solver). To our best knowledge, it does not call into question the on-line controller. This problem will be discussed again later in this chapter.

4.2 AMELIF, a simulator for haptic and dynamic rendering

The SoT framework allows a user to compute the control law corresponding to the regulation of a given set of tasks organized into a hierarchy for a given configuration of the robot and the environment. In order to perform dynamic simulations introducing the robot, the SoT has been integrated to the dynamic framework AMELIF.

This section is a brief overview of the structure of the AMELIF framework. Its basic components are detailed, and a demonstrative simulation is presented.

4.2.1 Presentation

AMELIF [27, 26] is an integrative framework that proposes an API for the representation and simulation of virtual scenes including articulated bodies. It was devised to realize interactive scenario studies with haptic feedback while providing an interface enabling fast and general prototyping of humanoids (avatars or robots). AMELIF is entirely developed in C++ and has been successfully tested under the Linux and Windows operating systems. The architecture of the framework is based on a core library, upon which different modules are built with minimal dependencies among them, hence allowing a neophyte user to utilize only the libraries s/he is interested in while requiring a minimal knowledge of the interfaces.

In the following, the core components are briefly introduced, and my main contributions are indicated by a star (*).

4.2.2 Basic libraries

The core library provides the generic tools liable to be used by many modules:

- Mathematical tools: optimized implementation of small vectors and matrices (3D), matrix abstract layer for external matrix libraries (such as VNL[1] and Eigen[2]);
- Communication and manipulation tools: containers, console, observers (listeners and publishers), XML parsers.
- Multi-threading: thread handling, memory sharing systems for multi-threaded contexts, chronometer.
- Basic external device handler: keyboard and mouse handlers.

The state module provides the interfaces of the components that describe a virtual environment. The simulated objects are articulated and their limbs are linked by joints of different types (revolute, prismatic or spherical). Each simulated object is defined by its physical (mass, inertia, deformable or not) and geometrical properties (shape, color, rigid or deformable) and its position in the environment (position, velocity, acceleration). A specific XML parser enables to build the environment starting from an XML file. Each of the classes is provided with a default implementation that can be inherited and adapted.

It is the main module for the simulation of a virtual environment: most of the information pass through the bodies and the other modules use it to read or modify the state of each object.

The collision detection module* enables to detect and display the collisions between the objects of the simulated environment and compute the distance between them. For this module, an abstract interface is provided so as to wrap external collision query libraries. For now, the default external library used is PQP (Proximity Query Package [49]), and the integration of the STP-BV library (Sphere Torus Patches Bounding Volumes [7]) is ongoing. STP-BV creates convex bounding volumes around objects, thus ensuring that the evolution of the distance between the bounding volumes is continuous. The collisions are accessible either by direct query on the collision detection module or by a system of observers.

A system of groups allows the user to limit the collision detection on specific parts of the environment. Hence, objects known to never collide (e.g. if they are too far from each others)

[1]VNL is the numerics library contained in the collection of C++ libraries VXL `http://public.kitware.com/vxl/doc/development/books/core/book_6.html`

[2]A C++ template library for linear algebra `http://eigen.tuxfamily.org/`

can be distributed in separated groups, so as to avoid useless computations. Also, a group can contain only one articulated object, in which case it is associated to the detection of its self-collisions. Flags define the set of possible queries (collision detection, interpenetration, distance…) and can be associated to a specific group or pair of bodies.

The dynamics module* proposes an interface and a default implementation to realize a physical simulation of a virtual scene composed from objects and articulated bodies. The internal structure of this module reproduces the structure of the environment and completes the definition of the articulated objects with specific informations. Besides, the dynamic module is defined as a listener of the collision module, and is itself a publisher of interactions, that are collisions augmented by the corresponding interaction forces.

The default implementation is based on the two-steps method presented in [13, 12]:

1. Computation of the free dynamic (without contacts), that takes into account the gravity force field and the external forces (such as those due to haptic interactions), with the Featherstone algorithm [29]:
$$\ddot{\mathbf{q}}_{\mathbf{free}} = \mathbf{A}(\mathbf{q})^{-1}\left(\boldsymbol{\Gamma}(\mathbf{q}) - \mathbf{b}\left(\mathbf{q}, \dot{\mathbf{q}}\right) - \mathbf{g}(\mathbf{q})\right) \tag{4.3}$$
2. Computation of the contact and impact forces, using constraint-based methods so as to avoid interpenetration of objects $\mathbf{0} \leq \mathbf{f_c} \perp \mathbf{a_c} \geq \mathbf{0}$ and computation of the additional acceleration:
$$\ddot{\mathbf{q}}_{\mathbf{c}} = \mathbf{A}(\mathbf{q})^{-1}\mathbf{J}_c^T\mathbf{f}_c \tag{4.4}$$

The computed acceleration is then integrated with a numerical method (e.g. simple Euler integration), and the state of each object of the state module is updated.

The control module* wraps the SoT framework. It enables defining any object in AMELIF as an entity and using its attributes as signals in the SoT framework. For example, once a multi-articulated object is defined as an entity, its position, velocity and acceleration are computed by AMELIF (they are output signals) while the control is computed by the stack of tasks and read by AMELIF (as an input signal).

When running a kinematic simulation, the control is directly used as the velocity of the system. In a dynamic simulation, the control can be considered either directly as the torque given to the joints or as a way to define the desired position. In the former case, the control (aka the torque $\boldsymbol{\Gamma}$) is taken into account in the dynamic computation by adding the acceleration.
$$\ddot{\mathbf{q}} = \mathbf{A}^{-1}\boldsymbol{\Gamma} \tag{4.5}$$
In the latter case, the control is used to define the desired position $\mathbf{q}^*$ (the desired velocity is null $\dot{\mathbf{q}}^* = \mathbf{0}$). Finally, the corresponding acceleration is computed using a PD-controller.
$$\ddot{\mathbf{q}} = \mathbf{A}^{-1}\left(K_p(\mathbf{q}^* - \mathbf{q}) + K_v(\mathbf{0} - \dot{\mathbf{q}})\right) \tag{4.6}$$

The communication with the SoT framework is realized directly via the script mechanism.

The display module is based on the well established OpenGL graphics library and provides additional tools to display the graphical scene. A particular tool is the class called “Fixture”, that can be attached to objects in order to display extra information such as contacts (forces, friction cones, normal) or bounding volumes. Besides, it is possible to use it to wrap the display routines, which can be of use with the Haptic Library (HL) of OpenHaptics, that adds haptics to OpenGL rendered geometry (the computation of the force rendered is directly done by the library). This module is not mandatory since one could limit its use of AMELIF to the algorithmic part, and use AMELIF only via the console.

4.2.3 Execution and simulation

Using these libraries, it is now possible to create and simulate a robotic mission or tasks in a virtual environment.

Main Programs define the simulation. They are also built as dynamic libraries except that they do not define an API (hence they can not be used by other libraries). These applications respect the same 3-steps structure:

1. An initialization phase, during which the environment is created: definition of the simulated universe, definition of the collision groups, selection of the bodies that are dynamically simulated... Also, a warming-up phase can be realized, to prepare the dynamical simulation.

2. The main loop, called repeatedly. The time given as an input of the method is the time spent since the previous call of the function.

3. A termination phase that realizes the destruction of the elements created and ends the application.

Core application Two displayers are available to realize the *execution* of the main programs: the first one only contains the minimal set to display a virtual environment: a console and a simple viewer based on OpenGL, while the second one is based on wxWidget and provides a more user friendly interface: it enables the dynamic load and execution of main programs.

The Fig. 4.4 summarizes the relationship between the different modules described above.

4.2.4 Demonstrative scenario

In order to demonstrate the capabilities of the framework and gather the different work realized in the team, a demonstrative scenario has been designed and implemented.

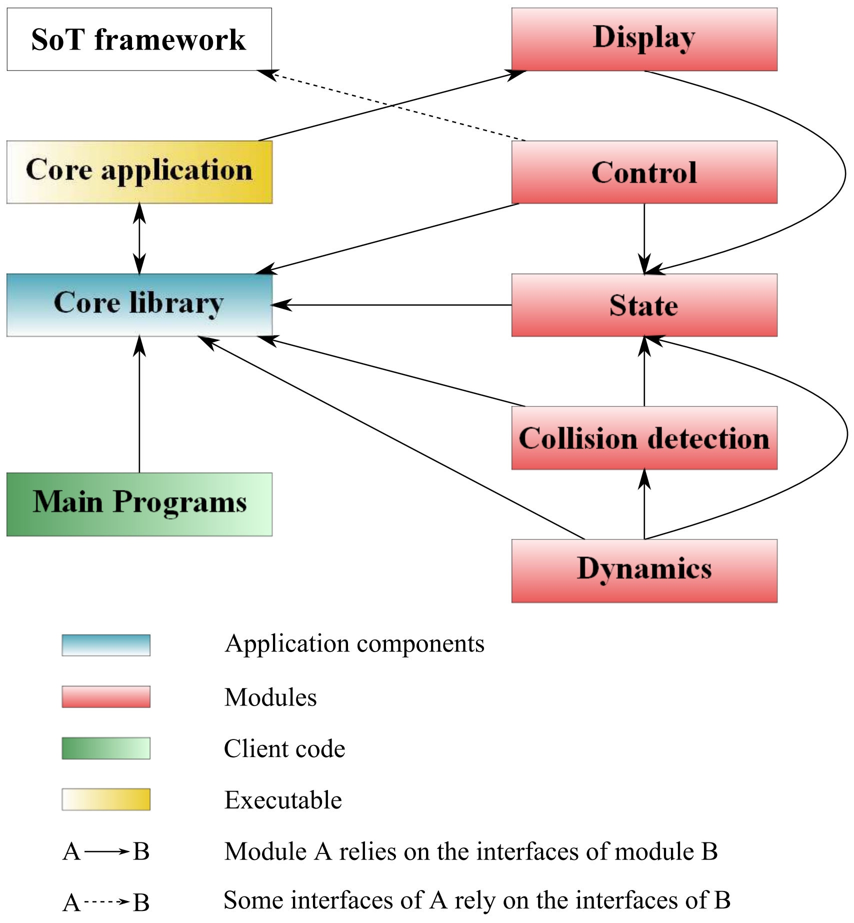


Figure 4.4: Modules composing the core of the AMELIF framework and their dependencies.

4.2.4.1 Additional libraries used

The following scenario integrates the work realized in two other libraries developed using the AMELIF formalism: a posture generator and a haptic device handler.

Posture Generation Configuring manually the posture of a virtual avatar can be burdensome for the user, especially when constraints such as equilibrium, joint limits and position constraints (e.g. contacts with the environment) for some bodies have to be taken into account. Hence, this module relies on a posture generator based on an optimization process, that researches a posture respecting the user-defined constraints while being as close as possible to the reference posture given. Details on this posture generator can be found in [9].

Haptic interaction In the proposed scenario, haptic interaction occurs during the collaborative manipulation of an object by a human operator and a virtual human avatar. It is based on the idea that the role of each partner is not limited to a static role of slave or master, but switches continuously between a leader role and a follower role. In this purpose, a homotopy-based model has been designed to state the role of each partner [28]. Depending on the intentions expressed by the human operator via the object (does the human operator apply forces on the object or is he passive?) and the constraints of the avatar (is the avatar near an auto-collision or a singularity?), the avatar complies with the trajectory implied by the user or takes the lead and tracks the trajectory it has computed based on its planner. The collision avoidance constraint used for the homotopy is based on the distance between the elbow and the chest.

4.2.4.2 Resulting simulation

The resulting simulation is split into two phases.

First, the user defines the reference position (here it corresponds to the half sitting posture given in Fig. 4.5a) and places the contact points he wants the avatar to fulfill (the two feet are on the ground and the right hand is holding the object). Once the user launches the program, a posture satisfying these conditions is processed during the initialization phase of the main program. It then switches to the main loop in an idle mode, and allows the user to check the results. The posture found is represented on Fig. 4.5b and corresponds to the initial posture of the human avatar for the dynamic simulation.

When the user decides to switch to the dynamic and haptic simulation mode (by pressing a key on the keyboard), the human avatar then focuses on the object Fig. 4.5c. The user can then start manipulating the object collaboratively with the avatar Using, by means of an haptic probe such as the PHANTOM Omni® or a the PHANTOM Premium 1.5® (Fig. 4.5d).

4.3 Optimization of a task sequence

In this section, we focus on the work realized in Chapter 3 on task overlapping, and present the integrative experiments with a fully integrated scenario. Namely, the optimizer inside the AMELIF simulator, using the SoT and the smooth control law presented before. We first present quickly some experiments on simple sequences, to validate the use of the smoothing inside the optimizer, and highlight the limitations due to some issues we mentioned previously, especially the discontinuities due to the modification in the order of discrete events. Then, we present a complete experiment on a full-size sequence, and its application to the real HRP-2 robot.

4.3.1 Smoothing and optimization

The used optimizer (CFSQP) gives better results when the criteria and the constraints are continuous. Therefore, the smooth control law should enhance the convergence of the optimization. Typically, the constraint on the joint velocity $\dot{\mathbf{q}}_{\min} \leq \dot{\mathbf{q}} \leq \dot{\mathbf{q}}_{\max}$ is smoothed. However, we have also seen that because of the discrete events, the smoothing process may introduce discontinuities in the evolution of the trajectory-based constraints with respect to the variations of the set of parameters $\mathbf{x}$.

The purpose of the following tests is to analyze the effects of the smoothing on the optimization process. In this frame, five tasks sequences are considered:

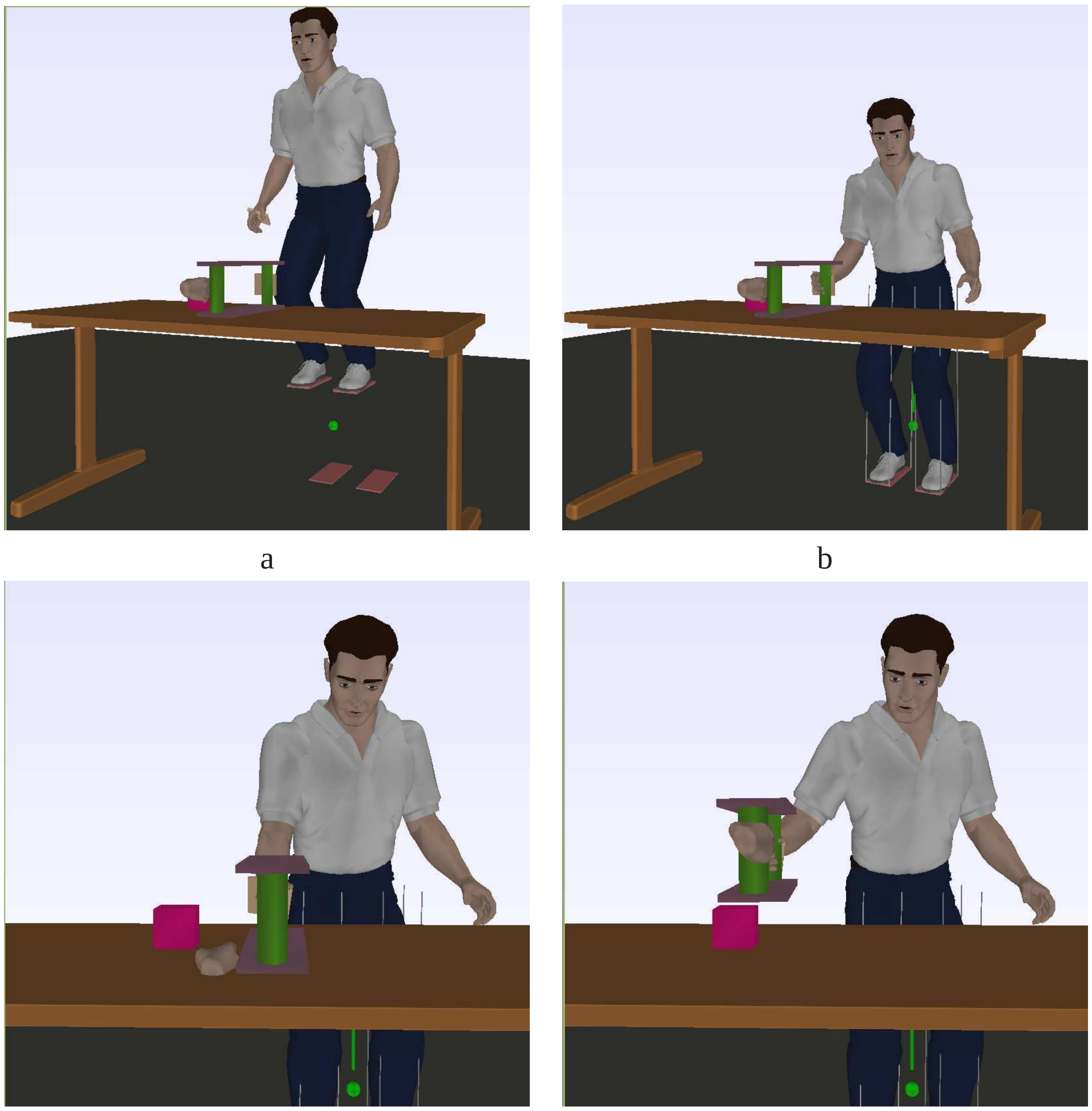


Figure 4.5: First, the human avatar is positioned so as to fit the initial contact conditions (a - b). Then the user can realize a collaborative manipulation of an object with the avatar (c - d).

1. A sequence containing only one task.
2. A sequence with two decoupled tasks $\mathbf{e_1}$ and $\mathbf{e_2}$.
3. A sequence with two decoupled tasks $\mathbf{e_1}$ and $\mathbf{e_2}$, such as $[t_2^I, t_2^F] \subseteq [t_1^I, t_1^F]$
4. A sequence with two coupled tasks $\mathbf{e_1}$ and $\mathbf{e_2}$.
5. A sequence with two coupled tasks $\mathbf{e_1}$ and $\mathbf{e_2}$, such as $[t_2^I, t_2^F] \subseteq [t_1^I, t_1^F]$

The constraints considered for the optimization problem are the task constraints (time coherence, termination condition, gain consistency) and the joint velocity limits (which are the only constraints we considered for the robot).

The smoothing is characterized by the parameter d: for low values of d, the process is slow but the derivative of the control, $\ddot{q}$, presents small variations; for high values of d, the swap is fast but $\ddot{q}$ presents large variations. In the following experiments, different values of d are tested. Immediate transitions (without smoothing) are noted "Immediate". The Fig. 4.6 represents the evolution of the control and the transition period for several values of d.

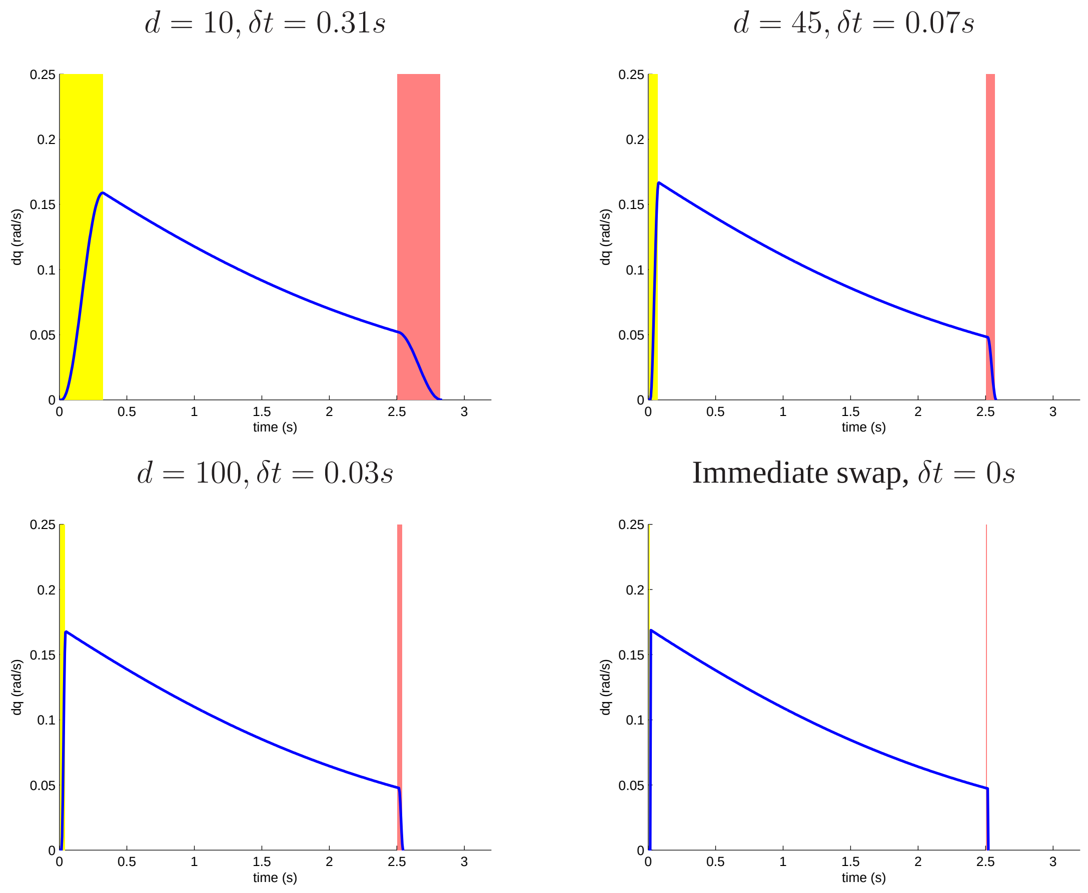


Figure 4.6: Evolution of the control during the insertion and removal of a single task (realized on one dof) for several values of the parameter d. The yellow area represents the insertion period. The red area represents the removal period.

First, we analyze the cases where the semi-infinite constraints on the joint velocity do not play a role, i.e. when only the timing is modified.

For simulations where the tasks are decoupled (Table 4.1(down) and Table 4.2(down)), the smoothing has only a light influence on the number of iterations realized. This can be explained by the fact that in this case, the delays added by the smoothing process are constant (one at the insertion and one at the removal for each task, but no swap) and do not play a role in the optimization. As a result, the optimization process only focuses on the minimization

Optimization of the gain	Smoothing (value of d)	Number of iterations	t^F
Yes	Immediate	39	3.457
	100	50	3.469
	45	37	3.474
	10	33	3.536
No	Immediate	24	4.354
	100	24	4.372
	45	24	4.391
	10	22	4.513

Table 4.1: Results of the optimization for the sequence 1 (one task).

Optimization of the gain	Smoothing (value of d)	Number of iterations	t^F
Yes	Immediate	246	3.733
	100	169	3.814
	45	75	4.382
	10	53	3.965
No	Immediate	37	12.186
	100	38	12.204
	45	37	12.223
	10	37	12.345

Table 4.2: Results of the optimization for the sequence 2 (two tasks not coupled, temporally independent).

Optimization of the gain	Smoothing (value of d)	Number of iterations	t^F
Yes	Immediate	198	3.815
	100	244	3.796
	45	201	3.864
	10	42	3.965
No	Immediate	41	12.186
	100	42	12.204
	45	46	12.224
	10	37	12.345

Table 4.3: Results of the optimization for the sequence 3 (two tasks not coupled, temporally dependent).

of the time of realization of the task(s), and starts the task with the longest duration as soon as possible.

Yet, when the tasks are coupled, the convergence of the process is disturbed by the dis-

Optimization of the gain	Smoothing (value of d)	Number of iterations	t^F
Yes	Immediate	214	3.273
	100	251	3.204
	45	196	3.196
	10	435	3.171
No	Immediate	38	12.070
	100	44	12.147
	45	62	12.111
	10	45	12.247

Table 4.4: Results of the optimization for the sequence 4 (two tasks coupled, temporally independent).

Optimization of the gain	Smoothing (value of d)	Number of iterations	t^F
Yes	Immediate	95	3.296
	100	90	3.294
	45	78	3.352
	10	152	3.358
No	Immediate	51	12.832
	100	49	13.177
	45	40	13.350
	10	35	14.580

Table 4.5: Results of the optimization for the sequence 5 (two tasks coupled and temporally dependent).

continuities with respect to the events. Indeed, in the experience 4, where the order of these events is left free, the smoothing process makes the optimization process converge slower than if the swap was instantaneous. At the opposite, in the experience 5, where the order of these events is fixed, the swap enhances the convergence performances of the optimization (cf. Table 4.5(down)). Besides, the effect of the smoothing is not monotonous with the variation of d: it is not possible to state if the slowness of convergence is directly linked with the value of d.

Now, we focus on the cases where the gains are optimized: the semi-infinite constraint on the velocity is activated during the optimization.

For simulations where the tasks are not decoupled, smoothing the control law enhances the convergence properties of the algorithm (cf. Table 4.1(up) and Table 4.2(up)). This result corresponds to the behavior expected: the solver works better with smooth constraints. Yet, when the tasks are coupled, the smoothing may not be advantageous anymore (cf. Tables 4.4(up), 4.5(up)). In order to know if this problem is due to the discontinuities with respect to the timing of the tasks, a sixth sequence has been realized with stronger constraints on the time schedule: the time constraints are now $[t_2^I, t_2^F] \subseteq [t_1^I + 4, t_1^F - 4]$. The purpose of this limitation is to prevent the optimization process from producing a schedule where the swap would appear. The results are illustrated on Table 4.6. As expected, when the optimization is only realized on the timing, the smoothing process has no influence. Though, when the gains are considered, the number of iterations is at the same level between the case with and without smoothing process (except for d =45), with an advantage for the immediate swap.

Aside from this analysis of the influence of the smoothing, it is important to note the important difference between the case where the sequence contains coupled tasks and the temporal constraints are given (Table 4.5) and the case where they are left free (Table 4.4).

Since working with coupled tasks is unavoidable –at least in the frame of this work–, this example shows the importance of the sequencing phase: leaving the determination of the task sequence up to the scheduling phase (i.e. to the optimization process) substantially penalizes the optimization process.

Optimization of the gain	Smoothing (value of d)	Number of iterations	t^F
Yes	Immediate	96	10.328
	100	103	10.129
	45	168	10.137
	10	101	10.754
No	Immediate	28	21.312
	100	28	21.330
	45	28	21.349
	10	28	21.471

Table 4.6: Results of the optimization for the sequence 6 (two tasks coupled and temporally dependent with important time margin).

4.3.2 Simulation of can grasping

The following scenario introduces a robot taking a can out of a fridge, and has been tested on the real humanoid robot HRP-2. This mission is decomposed as a task sequence, and optimized by realizing task overlapping.

4.3.2.1 Description of the task sequence

The mission is decomposed as a sequence of tasks, illustrated on Fig. 4.7. The corresponding tasks are:

Tasks of the right arm		Tasks of the left arm	
$\mathbf{e_0}$	Open the gripper,	$\mathbf{e_5}$	Open the gripper,
$\mathbf{e_1}$	Move the gripper to the fridge's handle,	$\mathbf{e_6}$	Move the gripper in the fridge area,
$\mathbf{e_2}$	Close the gripper,	$\mathbf{e_7}$	Move the gripper to the can,
$\mathbf{e_3}$	Open the fridge,	$\mathbf{e_8}$	Close the gripper,
$\mathbf{e_4}$	Close the fridge	$\mathbf{e_9}$	Lift the can,
		$\mathbf{e_{10}}$	Remove the can out of the fridge,

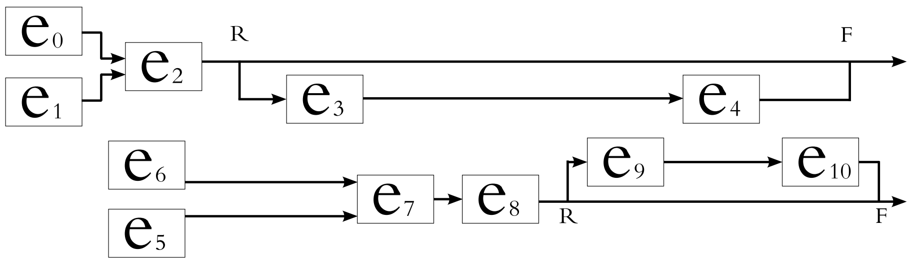


Figure 4.7: Sequence describing the HRP-2 taking the can in the fridge.

The task $\mathbf{e_6}$ is an intermediary task introduced as a way point, in order to ensure a lateral approach of the arm towards the can. The associated tolerance on task regulation ϵ_6 is large so as to avoid a null velocity of the arm at the regulation of the task.

This is a complex mission that can not be split into smaller sequences. Indeed, the sequence is centered on the fridge: the can-grasping part does not make sense if the fridge is closed and similarly, it is not possible to close the fridge while the left arm is still in it. Instead of adding explicit timing conditions between the tasks to ensure that this will never occur (such as $t_3^R \leq t_7^F$ and $t_{10}^R \leq t_4^F$), we choose to consider as limiting constraint the collision between the left arm and the door, in order to allow task overlapping.

The constraints considered for this problem are thus sequencing and robotic constraints (joint position and velocity limits), and collision avoidance with the fridge. As a safety measure, we reduce the joint velocity limits to 25% of their normal value.

4.3.2.2 Simulation: results of the optimization

The optimization of the task sequence has for parameters the position of each task in the schedule (i.e. the entry time t^I and the removal time t^F) and the behavior of each task, define by their gain function. The sequence found is described on Fig. 4.8. Each task is described by two periods: the dark one is the regulation period $[t_i^I, t_i^R]$, the bright one is the maintain period $[t_i^R, t_i^F]$.

The two overlaps between the tasks of the left and the right arm appear clearly. First, the left arm starts moving before the fridge is open and then aims at the can pose even if the fridge is not completely open. Second, the right arm starts closing the fridge before the left arm has completely left the fridge area. The whole task sequence lasts 47s. Without these

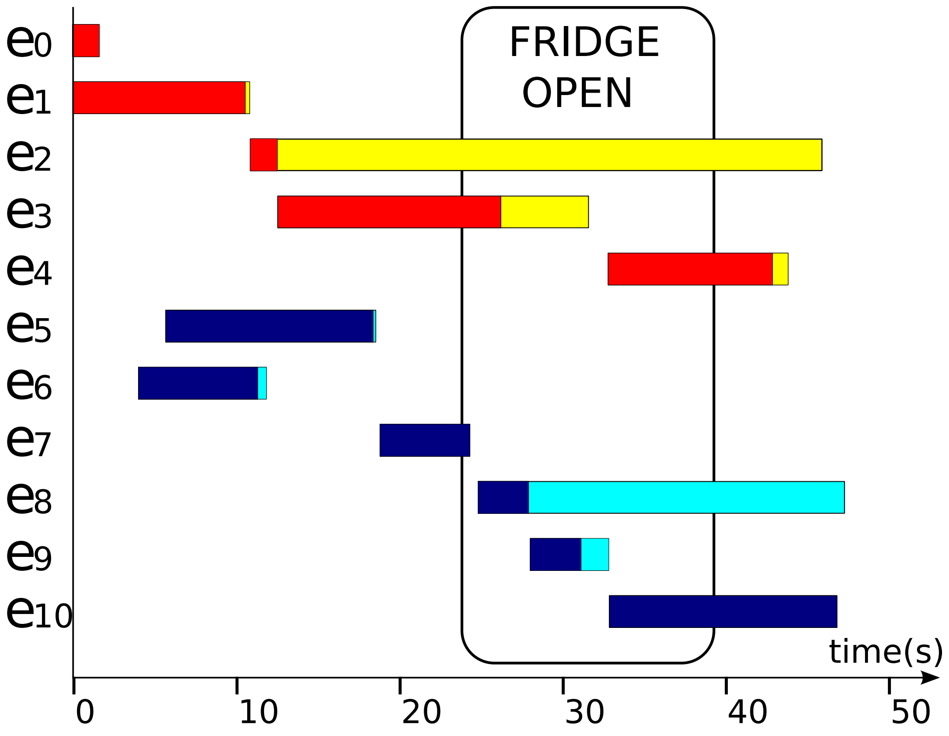


Figure 4.8: Results of the optimization of the sequence of task: when the task is added in the stack, its error is first regulated (this is the dark part (red or dark blue) of the block). From t_i^R, the error is nearly null and the task is kept in the stack (light part (yellow or cyan) of the block) until t_i^F.

two overlaps, the robot would have moved to the can ($\mathbf{e_7}$) only after the fridge was fully opened ($\mathbf{e_3}$) and it would have closed the fridge ($\mathbf{e_4}$) only after the can was completely taken out ($\mathbf{e_{10}}$). As a result, the total mission would have taken at least 71s.

4.3.2.3 Experiment on the real robot

The task sequence is experimented on the upper body of the HRP-2 humanoid robot; its properties and design characteristics are thoroughly described in [41]. The HRP-2 is a 42-degrees-of-freedom humanoid robot built by Kawada Industries (a Japanese company). In this scenario, only the described tasks are used to compute the control law. In other words, no additional care is taken for enforcing the constraints such as joint limits: their respect –or violation– is only function of the tasks of the sequence. For the tasks that require haptic interaction (i.e. opening and closing the fridge) the force sensor of the robot is used to close the loop and compensate for position uncertainties.

The robot manages to grasp the can without colliding with any obstacle and without reaching joint limits, while fulfilling the remaining robotic constraints on velocity limits during its motions. The obtained execution is plotted on Fig. 4.9. Thanks to the optimized gain, the convergence of the error of the tasks that require a good precision (grasping the fridge handle and the can) is achieved as quickly as allowed by joint velocity limits. Snapshots of the execution are given in Fig. 4.10, while the full video is available on `http://staff.aist.go.jp/francois.keith/video.html`.

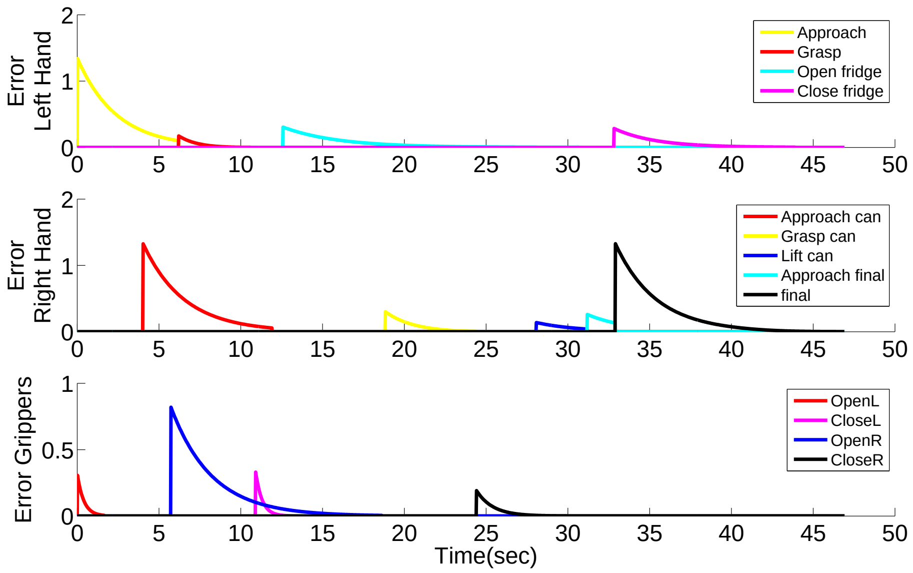


Figure 4.9: Experiment on HRP-2: errors diminish when optimized task scheduling is applied: (top) right arm tasks (middle) left arm tasks (bottom) gripper tasks. The concurrency between the tasks is clearly visible.

4.4 Conclusion

This chapter described the two frameworks used to implement and assess the theoretical development of our work. Particularly, the implementation of the smooth control law in the inverse kinematic framework SoT and the creation of the bridge between the SoT and the dynamic simulator AMELIF have been detailed. More importantly, this chapter highlighted the pros (smooth control) and cons (additional delays) of the smoothing process, and introduces the compromise the user has to make between the smoothness of the control law and the reactivity of the system unless computations can be made faster in the control loop (which is not the case for the moment). As a result, depending on the properties of the task sequence considered, the smoothing process can favor the convergence properties of the optimizer, or in the contrary penalize them. Note that smoothing generally induces a delay (phase filter).

In the next chapter, we see another facet of the task function, which enables the adaptation of a default behavior, typically returned by the optimization process to fulfill another goal: human preferences in human-robot collocated space and co-existence.

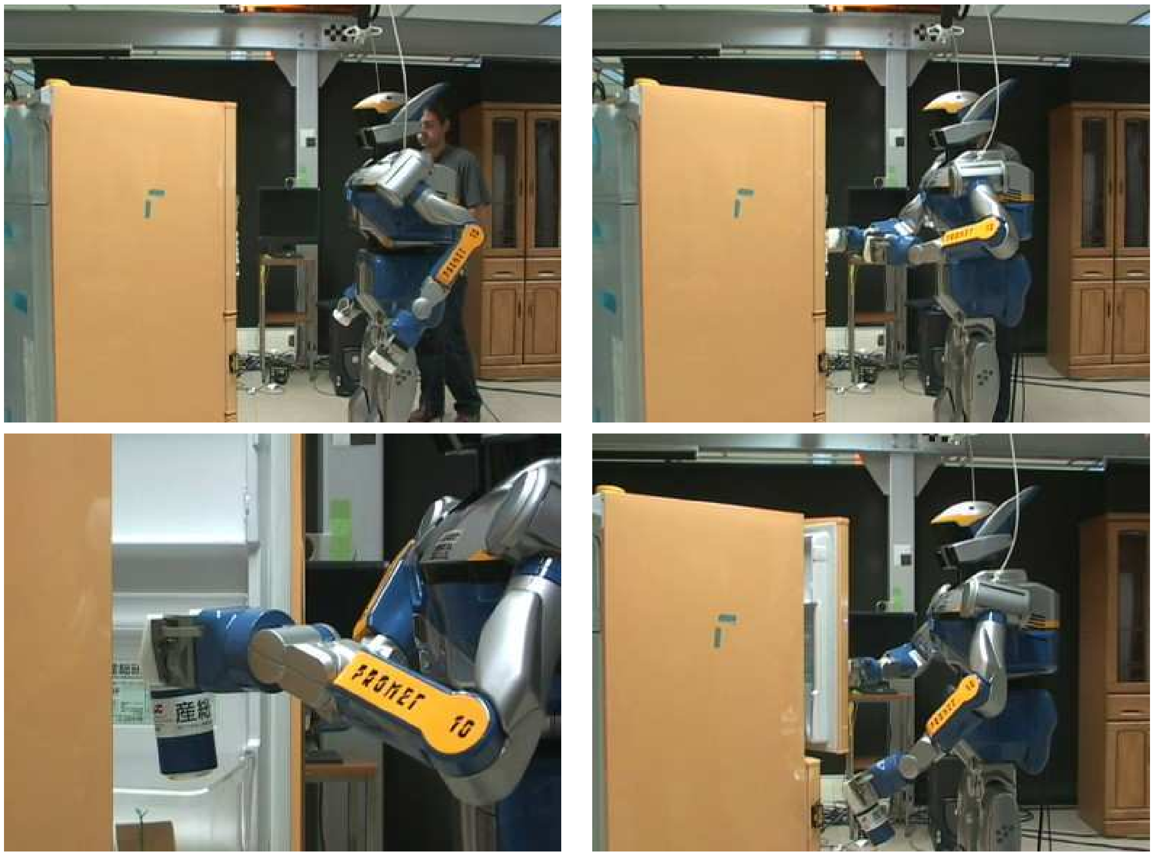


Figure 4.10: Snapshots from HRP-2 grasping a can in the fridge.

Chapter 5

Adaptive control

The optimization process detailed in Chapter 3 refines a task schedule where everything is predictable since depending only on the motions of the robot and on an idealistic representation of the environment and its potential changes. In practice, it is most likely that the robot needs to adapt its planned schedule and subsequent motions in response to situation changes and execution results. The task function approach, by design, offers such flexibility in the execution because the trajectory is implicitly generated from the interaction between the sensory error and the desired motion. In other words, the objectives of the tasks are continuously adapted to fit the variations of the environment. Still, such adaptations are valid within some bounds and locality constraints; the task function does not deal with important variations. Even if local, light variations may necessitate an adjustment of the scheduled plan that can be done by simple propagation unless the variations are important and subsequently, necessitate a rescheduling. This kind of adaptation modifies the goal of the tasks: i.e. the "what" (should be done?).

Rather, this chapter focuses on another type of adaptation, which consists in achieving and pursuing the same goal, but in a different way. This category of modification is about the "how" (should it be done?). The idea presented in this chapter is rather an extension of the stack-of-task formalism to the possibility to personalize the motion of the robot by using an appropriate parameterization of the gain and the timing of each task.

As an example, the task sequence introduced in the previous chapter is altered so as to take the presence of a human operator into account and to adopt a human-like behavior in a collaborative scenario.

This whole chapter has been realized in collaboration with Marie AVRIL, student at the ENSC (École Nationale Supérieure de Cognitique) of Bordeaux.

5.1 Adaptive control and task formalism

Consider a pick-and-place motion of, e.g., a glass. The way the robot plan for the grasping posture, the closure, and the manipulation motion considerably depends on the type/shape of the glass in question: a crystal glass should be manipulated carefully (by gently grasping it and avoiding impacts when putting it down), whereas a plastic glass can be manipulated faster and with less care (i.e. more tolerance on the impact's intensity). Even though these two actions are defined with the same objective, the way they are realized varies with such different considerations. Similarly, a user may specify constraints that will affect the resulting motion: for example the constraints “act as fast as possible”, “act as a human would do” or “use the minimum amount of energy” certainly results in different motions or even impose different constraints.

In the following, we present some of the advantages gained by such kind of adaptation; namely, the diversification of the behavior for a given know-how enabling a better human-robot interaction and cohabitation.

5.1.1 Enrichments and diversification of know-how

In computer graphics, this distinction between the action (called *verb*) and the way to realize it (called *adverb*) [76] is used to enlarge the set of possible actions doable by an avatar (its know-how) at low cost, by defining how to realize the known actions in a different way. Similarly, *adapting* the tasks contained in the know-how of a robot is an easy way to increase its capacities, cheaper than the definition of new tasks.

Another method used in computer graphics to improve the set of possible actions doable by an avatar is to realize a blending between these actions [91, 72]. This is what is realized during the swap of two incompatible tasks. Yet, this method has to be applied with extra care, since the blending of tasks may not fulfill some constraints (such as stability, auto-collision avoidance, etc.). The blending of two position tasks, one setting the right arm behind the waist and the other one before it, highlights this problem.

5.1.2 Human's expectation and preference

When a human operator and a robot share the same space, it is necessary that the robot adapts its behavior and plans –provided by the automated planner for a given mission– according to human expectations and preferences.

Such considerations are particularly useful to achieve collaborative tasks (where the human operator has an active role) and can even improve the performance of the human-robot teaming [80]. The robot must be able to adapt its motion according to the physical attitude of the human but also based on the human expression (stress in the voice, fatigue, etc.).

Note that there are two kinds of adjustments that are expected from a robot when acting in a collocated space with a human: (i) general rules of robot behavior that can be seen both as constraints and as robotic ‘manners’ and ‘know-how’ in the presence of a human user, and (ii) robot tuning to human preferences, that are acquired from long-term observations and consists in the robot ability to personalize.

As an example of the general considerations due to the ‘manners’ and ‘know-how’ of the robot, the task plan and task function controller must integrate additional constraints so that the trajectory generated for the robot fulfills not only the classical constraints but also human expectations. For example, the visibility criterion [1]: the robot should not surprise the human operator by appearing suddenly from behind an obstacle and should adopt a path visible by the human operator. This assessment was refined by a study on the influence of the trajectory followed by the robot on the feelings of a human operator [18]. In this experiment, the robot acted as a helper for the human subject: the robot brings an object to a human user sitting on a chair. The robot was programmed to take different trajectories: it started in front of the user and then followed either a straight trajectory (frontal approach) or a curve trajectory (approach by the left or right side). The main result of this study is that the frontal approach makes the human user uncomfortable, or even seems threatening.

The task function approach has many benefits but also some limitations, namely when trajectories are generated implicitly from the error between a target and a robot control point, the resulting trajectory in the operational is likely a strait line. One way to take into account the previously cited expectation is to add way-point tasks (i.e. tasks that deviate the robot from its initial trajectory, whose regulation is not mandatory).

In our previous example, the robot can come from right or left. This choice can be induced by the task or the environment configuration, but it may also be imposed by the human user preferences. Some users would prefer the robot to come from the left, other from the right. For a set of tasks where different orderings lead to the same results, the preference can be given to the user taste. The robot can store this additional knowledge (human preference), which is acquired from observations or interactions. This thesis does not deal with these aspects for the moment.

In the same way that an appropriate parameterization of the times and the gain functions can lead to an optimized realization of a task sequence (in the sense of the duration of the mission), we believe that the personalization of a given sequence of tasks can be realized at the task level by adapting these same parameters. As an illustration of this idea, we propose to adapt a given sequence of tasks realized autonomously by the robot in order to take into account a human operator. To have an idea of the natural behavior expected from the robot, a preliminary study is realized between two human subjects. This idea actually suggests that the task approach components in planning, scheduling and execution can be kept even in situations where the robot evolves in collocated space or in direct interaction with a human operator and subsequently definitely constitutes a powerful component by which a cognitive robotic architecture can build.

5.2 Preliminary experiment

Imagine a person (customer) leaving a shop with two heavy bags, one in each hand. The shop is not equipped with automatic doors. Another person (the shop bellboy), standing near the exit door, notices that the customer is coming and opens the door for him. If he has well timed the door opening motion, the client will pass the door without having to slow down. Otherwise, in the case the bellboy was slow and delayed the door opening motion, the client will slow down or even stop, waiting for the door to be opened to go out. In the case the door opens too early, it is very likely that the client accelerates his natural walking motion, so as not to abuse of the bellboy's time and kindness.

Between the two persons, there was no direct communication: the bellboy estimated the adequate timing considering the trajectory of the customer and adapted his task behavior accordingly.

We aim at showing that a similar issue can be programmed keeping the stack-of-task formalism. We took a simpler case study, which is however based on the same idea: the first step is the observation of invariants and possible parameterization of the task-case study; the second phase consists in integrating this knowledge to our control method.

5.2.1 Script

The studied scenario implies two persons that will coordinate their actions for a given scenario. The first person (the *client*) wants to take an object placed in a cupboard, and the second one (the *bellboy*) has to open a cupboard so that the client can take the object. For the purpose of observation, two human operators execute this scenario. This preliminary experiment aims at determining what can be considered as a natural behavior for the bellboy (role played later by the robot) and at finding on which criteria the bellboy schedules the cupboard opening motion (i.e. the timing and behavior).

5.2.1.1 Experimental setup

During the entire experiment, the bellboy stands next to the cupboard and has the handle of the cupboard within reach of his right hand. We did not impose any constraint on the posture (including body position/orientation) the bellboy should take; we only explained what the task goal was and made few white trials so that the bellboy takes the most comfortable posture to open the cupboard. Nevertheless, he has to operate the cupboard's handle with his right hand. Besides, once the cupboard handle is grasped, the bellboy is not allowed to release it before the end of the task; i.e. not before the object is taken out from the cupboard.

The initial position of the client is chosen among three possible starting points, all placed at four meters of the cupboard. The client is asked to walk toward the cupboard with three different subjective velocities (slowly, normally, or quickly) during the experiment. At the beginning of each simulation, the bellboy sees the initial position of the client, but is unaware of the velocity by which he is asked to perform the task. The Fig. 5.1 represents the initial experimental setup.

The study was run with 19 subjects (15 men, 4 women), each of them played both roles (client and bellboy). In the first set of experiments, the client always started in front of the cupboard (the only starting point considered was S_1), and adopted one of the three velocities: the instruction on what velocity to choose was written on a small board handed to the client in the back of the bellboy (who was hence not aware of it). In the second set of experiment (12 subjects), two more starting points were considered in order to see whether a lateral approach modifies the behavior of the bellboy. For this configuration, we run 9 experiments (each of the three velocities was tested for each of the three starting points). Furthermore, some particular configurations were added, during which the behavior of one of the subjects was imposed, in order to observe the reaction of the partner. They are detailed later.

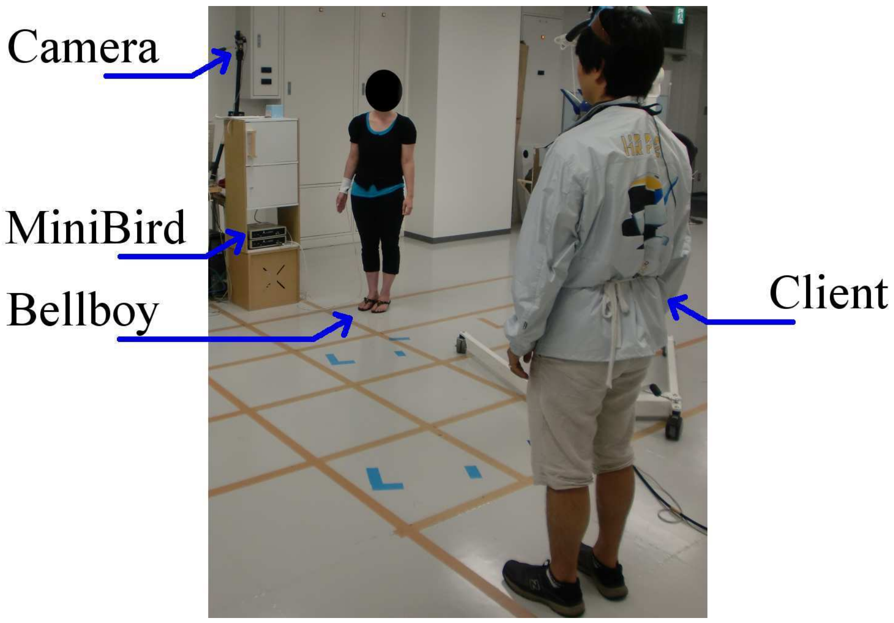


Figure 5.1: Experimental setup (initial position): the client starts far away from the cupboard while the bellboy is at a given posture of her/his choice and make sure that the cupboard's handle is within reach at a comfortable motion and grabbing posture. An accelerometer is placed on the head of the client.

5.2.1.2 Measures of the motions

Considering the motion of each subject (the bellboy does not walk –feet are at a fixed position–, while the client walks), two different types of measure were considered:

Measure of the client motion The client's body motion is tracked by a visual tracking system, which determines his position in space. The camera is placed on the cupboard when the client starts from S_1 or S_3. When he starts from S_2, this configuration is not suitable, since the bellboy occludes the view; the camera has to be moved to the right of the cupboard. To make the tracking easier, the client wears a rectangular blue plate placed on the chest (size:

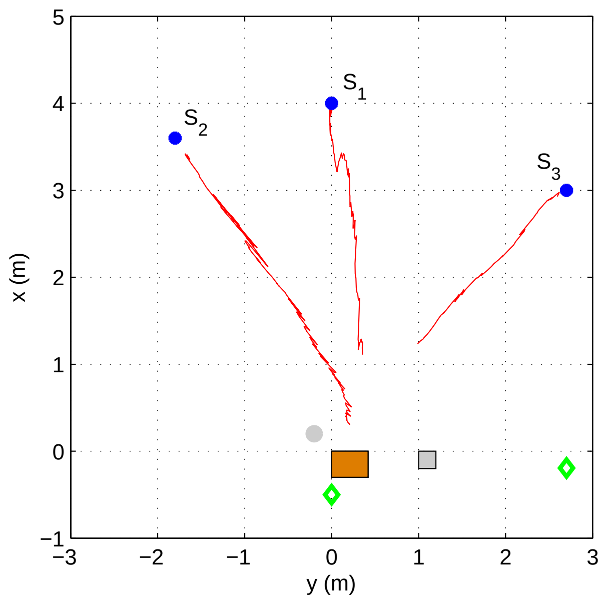


Figure 5.2: The environment of simulation. The three blue points S_1, S_2 and S_3 correspond to the starting points of the client. The gray point represents the position of the bellboy. The green diamonds represent the two possible positions for the camera. The orange and gray rectangles represent the cupboard and the table respectively.

45cm $\times$ 31cm) to be recognized and visually tracked. To confirm the trajectory obtained by the tracking system, an accelerometer, that records the acceleration with a high accuracy, is placed on the head of the client. It is linked to a computer by a 4.2-meters-long cable (the computer was placed at midway so as not to limit the motion of the client).

Measure of the bellboy motion Considering that the bellboy moves mainly his right arm, only the motion of his wrist is recorded. Two Ascension miniBirds[1] (electromagnetic sensor of position in space) are attached around his right wrist, and the position considered is the median of the two positions captured.

5.2.1.3 Questionnaire

At the end of the experiments, a simple questionnaire was given to the subjects:

1. Were you bothered by the sensors (miniBird for the bellboy, accelerometers and tracker for the client) ?

2. When playing the role of the client, did you feel the need to adapt your behavior to the one of the bellboy?

[1]`http://www.ascension-tech.com/realtime/RTminiBIRD500_800.php`

5.2.2 Results

On the 137 experiments realized, 101 were usable. The principal causes of failure during the data analysis were tracking errors (the tracker lost the subject, 8 cases), unusable miniBird results (the task schedule was not definable, 7 cases), and unusable tracking results (the trajectory of the subject was wrong, 6 cases).

The Fig. 5.2 represents the trajectories estimated by the visual tracking system for a same subject coming from the three possible starting points. The Fig. 5.3 illustrates the evolution of the distance between the client and the cupboard and the speed of the client along the axis client/cupboard. The data of the tracking system appeared to be sufficient and precise enough to define the dependencies between the motions of the two subjects. The acceleration was subsequently not needed to correct the visual tracking.

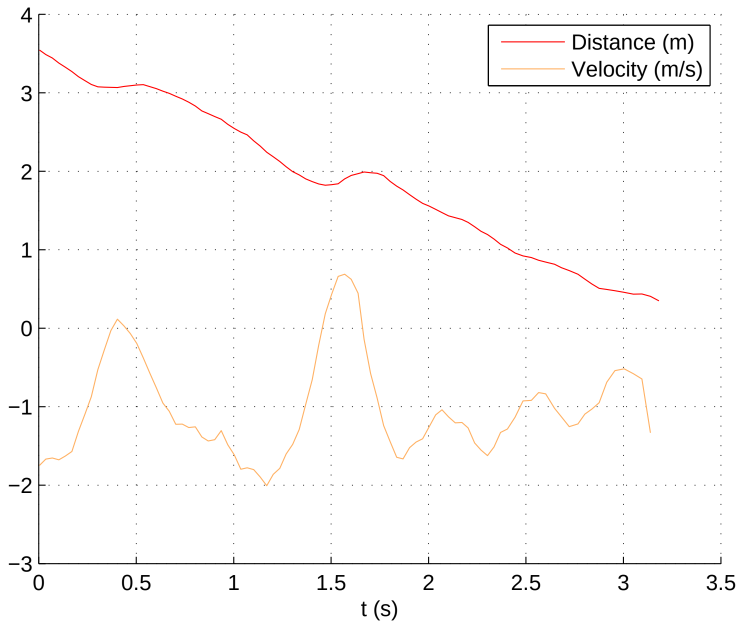


Figure 5.3: Evolution of the distance between the client and the cupboard and the velocity of the client projected on the client–cupboard axis.

In this section, we first detail the invariants in the motion observed for the bellboys, we then explain how they adapted their motion to the behavior (different velocities and initial start) of the client. In the following, by *distance*, we always mean distance between the client and the cupboard. Also, by *velocity*, we always mean the absolute velocity of the client projected on the client–cupboard axis.

5.2.2.1 Invariant in the bellboy motion

The typical evolution of the position of the hand of the bellboy is illustrated on Fig. 5.4. The frame, centered on the minibird, is the one used on Fig. 5.2: the x axis is oriented toward the client, the y axis is oriented towards the cupboard, and the z axis is oriented towards the ground. The three stable phases along the z axis, that indicates the height of the hand, enable determining each step of the motion. In the initial state ($t \in$[0s, 3.2s]), the hand is along the body, then, during the cupboard manipulation, it is attached to the handle of the cupboard ($t \in$[3.5s, 6.4s]), and in the final state, it is along the body again ($t \geq$ 6.8s). The transition periods can be decomposed based on the back-and-forth motions noticeable on the x and y axes: at $t = 3.2$s, the bellboy starts moving his hand toward the handle of the cupboard, then he grasps it around $t = 3.5$s (the height is stable but the hand still moves) and start opening the cupboard at $t = 4.0$s. The closure of the fridge follows a reversed evolution: at $t = 6$s, the bellboys closes the cupboard, then releases the handle at $t = 6.4$s and finally goes to its final position at $t = 6.8$s.

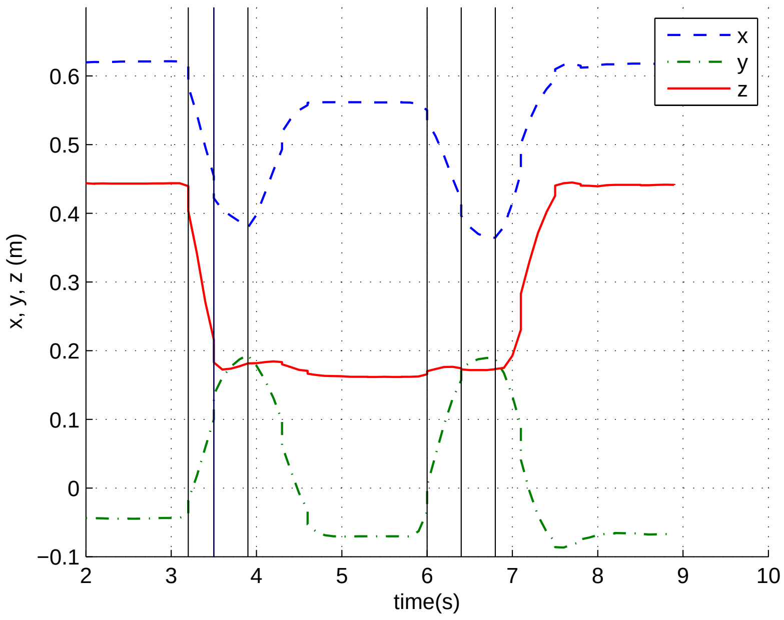


Figure 5.4: Evolution of the position of the hand in the frame of the miniBird. Each vertical line represents a change of trajectory that can be associated to an event. At $t = 3.2$s, the hand goes to the handle; at $t = 3.5$s, the handle is grasped; at $t = 4.0$s, the cupboard is opened; at $t = 6.0$s, the cupboard is closed; at $t = 6.4$s, the handle is released; at $t = 6.8$s, the hand goes to its final position.

One invariant observed in the behavior of the bellboys is that none of them paused during the opening or the closing of the cupboard. A possible behavior (classically programmed in robotics) would have been to first go for the handle and grab it, wait steadily for the client to come closer and then open the cupboard.

5.2.2.2 Adaptation of the bellboy behavior in function of the client motion

Two types of behaviors were observed:

Dependency in the client's trajectory Most of the bellboys adapted their behavior in function of the trajectory of the client. Especially, the trigger of the action of the bellboy is determined by a couple of elements: the distance between the client and the cupboard and his walking velocity. We note d^I the distance and v_h^I the velocity of the client when the bellboy starts moving (at t^I). By expressing this distance as a function of the velocity, as depicted in Fig. 5.5, three different behaviors can be distinguished:

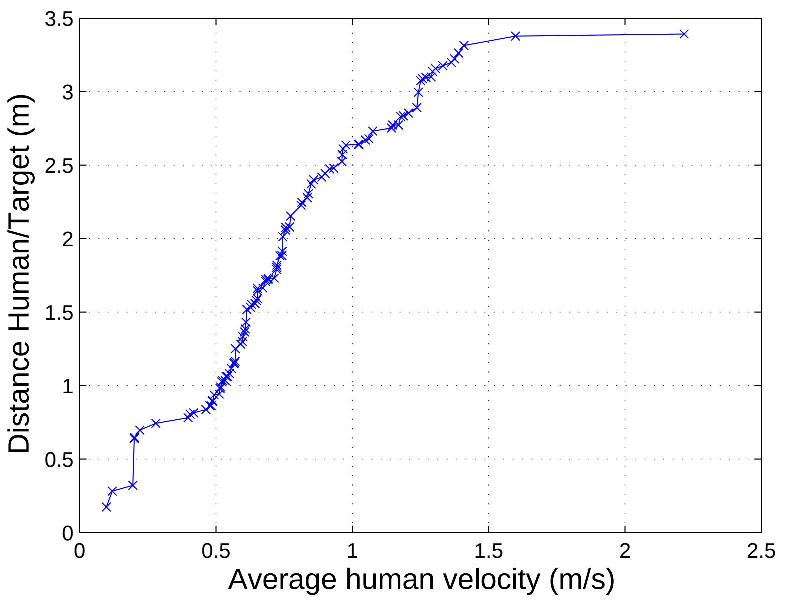


Figure 5.5: Distance between the client and the cupboard target d^I when the bellboy starts his sequence, with respect to the velocity v_h^I.

- When the client walks too fast ($v_h^I \geq$1.4m/s), the bellboy starts moving his arm as soon as possible (the distance is inferior to 4 meters due to the reaction time of the client). Besides, the bellboy has to speed up in order not to be late.
- When he has a slow speed (0.4$\leq v_h^I \leq$1.0m/s) or a regular speed (1.0$\leq v_h^I \leq$1.4m/s), the bellboy adapts its starting time accordingly: the faster the client goes, the sooner the bellboy begins its motion. However, the bellboy keeps opening the cupboard at the same speed.
- Finally, for an excessively slow motion, the action of the bellboy is triggered by the distance of the operator toward the cupboard, regardless of the velocity (found to be around 0.75m).

Dependency in time One of the bellboys adopted a particular attitude: s/he started at the same time as the client, but achieved the motion regardless of the client's trajectory.

This behavior can be considered as correct, since the client certified he was not perturbed by this attitude. Besides, opening the cupboard sooner does not penalize the environment: Maintaining a cupboard wide open a long time does not pose any problem, whereas doing the same thing with a fridge or an oven would cause a loss of energy. Also, it does not require any extra effort to maintain the cupboard open, because there is no mechanical door-closer.

5.2.2.3 Study of singular cases

Some particular configurations were arbitrarily inserted during the sessions. In each of these, only the subject concerned by the singular behavior was aware of the change in the original plan.

Zigzag trajectory of the client For every test of this kind, the bellboy still adapted his starting time depending on the velocity of the client, but he seemed to realize his motion faster, enabling him to start later than in the average case (i.e. when the client is closer).

Pause during client's walk In this configuration, the client was asked to stop walking during the opening phase of the cupboard (after the bellboy started moving and before he finished the opening of the cupboard). For a pause lasting less than 1s (2 cases), the bellboys did not modify their motions; else way (5 cases), he also paused. Besides, when this pause occurred before the bellboy had grasped the handle, several of them started lowering their arms.

Simultaneous start In this case, the bellboy had to start simultaneously with the client. Whereas the client kept the same behavior, the bellboy slowed down his motion so as to compensate the early start and finish opening the cupboard later.

5.2.2.4 Answers to questionnaire

Most of the time, the clients did not feel the need to adapt their motion to the bellboy's one. Two of them explained that during the experiment in which they walked quickly, the bellboy was not fast enough. Also, the accelerometer disturbed two of the subjects, especially the fragile-looking cable, that make them watch their motion so as not to break it.

5.3 Experiments with the HRP-2 robot

Using these results, we implemented this behavior on the robot. The purpose is to modify a task sequence already implemented so that it seems close to what is likely expected by a human user of the robot. In the following, we detail the default motion and the method used to integrate the presence of the human subject, validate them and finally run some new experiments.

5.3.1 Default task sequence

To play the role of the bellboy, the robot has to open and close the cupboard. The corresponding task sequence, represented on Fig. 5.6, is split into seven tasks:

$\mathbf{e_{1a}}$	Lift the right gripper (way-point task)
$\mathbf{e_1}$	Move the right gripper to the cupboard handle
$\mathbf{e_2}$	Close the gripper
$\mathbf{e_3}$	Open the cupboard door with the right arm
$\mathbf{e_4}$	Close the cupboard door
$\mathbf{e_5}$	Open the gripper
$\mathbf{e_6}$	Move the arm back to its initial position

The task $\mathbf{e_{1a}}$ is a way-point task ensuring that the gripper approaches horizontally the handle, and not from below, which would result in a collision with the support of the handle.

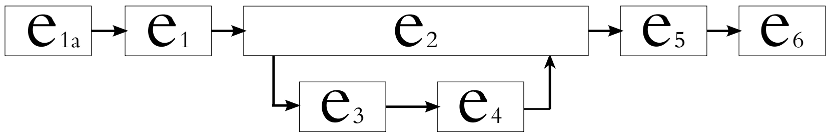


Figure 5.6: Task sequence of the fridge opening.

The observations made in the previous section lead to the following conclusions:

1. Only the tasks inducing arm motion are dependent on the behavior of the client, the gripper tasks ($\mathbf{e_2}$ and $\mathbf{e_5}$) are realized in constant time.

2. There is no dead time between the tasks: each task is removed immediately after regulation. In other words, the maintaining period is zero (0s): the regulation time t^R is equal to the removal time t^F. The opening task $\mathbf{e_3}$ is the only exception, because it must be maintained long enough to allow the client to take the object in the cupboard.

In order to make the robot react to the actions of the client, a vision task is added at high priority. This task acts both as a command (it sets the position of the head) and as a sensor (it determines the position of the client).

5.3.2 Adaptation of the tasks

The characteristics of the given default task sequence are known. Especially, for each task $\mathbf{e_i}$, the theoretical times of insertion t_i^I, regulation t_i^R and removal t_i^F are known. Thus, the time taken Δt_{op} by the robot to open the cupboard starting from its initial position is:

$$\Delta t_{op} = (t_{1a}^F - t_{1a}^I) + (t_1^F - t_1^I) + (t_2^F - t_2^I) + (t_1^R - t_1^I) \tag{5.1}$$

In order to take the trajectory of the client into account, it is necessary to adapt the behavior (i.e. the gains) and consequently the timing of each task. However, the objectives of the tasks remain unchanged. The two approaches considered to adapt the sequence to the presence of the human agent are those observed during the experiments between two human subjects:

Minimal adaptation In this configuration, the robot starts opening the cupboard as soon as the human operator starts walking and realizes normally the default sequence, regardless of the client motion. The only adaptation of the task sequence is the start time of the closure of the cupboard $\mathbf{e_4}$. This experiment can be interpreted as a qualitative experiment: its purpose is to see the interest of the adaptive approach.

Smooth adaptation In this scenario, the robot adapts both the timing and the behavior of the tasks in function of the behavior of the human operator. As indicated by the preliminary results, two adaptations are required:

First, the starting time of the sequence must be determined in function of the client's approach. To this purpose, the relation between the distance d^I and the velocity v_h^I observed during the experiment between human agents (illustrated on Fig. 5.5) is simplified as an affine function:

$$d^I(v_h^I) = \Delta t_{op} v_h^I + d_0$$

d_0 is the distance separating the client and the cupboard when he stops ($d_0 =$0.75m).
v_h^I is the client velocity when the bellboy starts acting.

As a result, the robot has to start as soon as:

$$d(t) \leq d^I(v_h(t))$$

$d(t)$ is the distance at the time t, and $v_h(t)$ is the client velocity at the time t.

The maximal velocity $v_{h,\max}^I$ the client can reach without forcing the robot to adapt its reference behavior is:

$$v_{h,\max}^I = \frac{d_{\max} - d_0}{\Delta t_{op}} \tag{5.2}$$

$d_{\max}$ is the initial distance between separating the client and the cupboard ($d_{\max} = 4$m).

As depicted in Fig. 5.7, three possible behaviors are expected:

- When the human walks too fast ($v_h^I \geq v_{h,\max}^I$), the robot has to start immediately.
- When he has a slow speed (0.4$\leq v_h^I \leq$1.0m/s) or a regular speed (1.0$\leq v_h^I \leq$1.4m/s), the start distance is determined by $d(t) = \Delta t_{op} v_h^I + d_0$
- Finally, for an excessively slow motion, the robot starts moving when the human is at the distance d_0 of the cupboard.

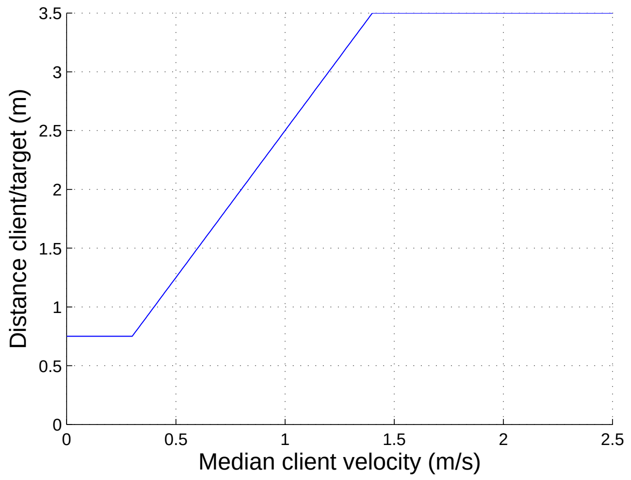


Figure 5.7: A simplified relation between the velocity and the distance. Δt_{op} *=2.5s.*

The second adaptation aims at making the robot react to changes of speed of the client. It consists in adapting the gain function as follows:

$$\lambda_h(\mathbf{e}) = \lambda(\mathbf{e})\frac{v_h(t)}{v_h^I} \tag{5.3}$$

If the client keeps walking at the velocity v_h^I during the whole motion, then the robot will follow its default behavior. However, if this velocity is above the limit velocity $v_{h,\max}^I$, the robot will be late. In this case, the default behavior cannot be realized as such, and has to be accelerated. To solve this problem, the equation (5.3) is modified in order to consider the reference velocity $\tilde{v}_h^I$, bounded to the interval [0, $v_{h,\max}^I$] rather than the actual velocity at the insertion v_h^I. The purpose of this limitation is to force an increase of the gain (i.e. an acceleration of the motion) when the client has an excessive velocity.

$$\lambda_h(\mathbf{e}) = \lambda(\mathbf{e})\frac{v_h(t)}{\tilde{v}_h^I} \tag{5.4}$$

The equation (5.4) introduces a direct dependency in the velocity of the client. This produces sudden changes of attitude, especially when the client stops walking. To smooth this behavior, we consider the median velocity of the client during the previous n steps. This method slows down the reactions of the robot, which allows the robot not to stop its motion for too brief stops of the human operator.

In practice, we choose $n = 100$ for time steps of 5 milliseconds. The median was hence computed considering the values obtained for half a second, i.e. during the interval of time $)t - 0.5, t]$. As a result, the client has to stay still for 0.5 second for the robot to completely stop.

$$\lambda_h = \frac{1}{n}\sum_{i=0..n-1}\frac{v_h(t-ndt)}{\tilde{v}_h^I} \tag{5.5}$$

5.3.3 Simulation

The reference trajectory corresponds to the optimal motion obtained using the method proposed in Chapter 3, considering joint velocity limits at 75% of their normal value. The purpose of this limitation is to keep a breathing space to realize faster motions (if the client walks too fast), while obtaining a motion fast enough to enable the human-robot interaction in normal case.

Though, as depicted on Fig. 5.8, the whole opening motion (tasks $\mathbf{e_1}$ to $\mathbf{e_3}$) lasts 4.62 seconds. Considering that a human walking at regular speed covers the four-meters distance between the starting point and the cupboard in less than 4 seconds, it is not possible for the robot to realize the entire opening subsequence in time. Thus, only the task $\mathbf{e_3}$, realized in 1.2s, was considered during the experiments and, as a safety measure, the gains where decreased in order to realize this motion in 2 seconds. The initial configuration of the robot changes accordingly: the robot starts with the gripper already on the handle.

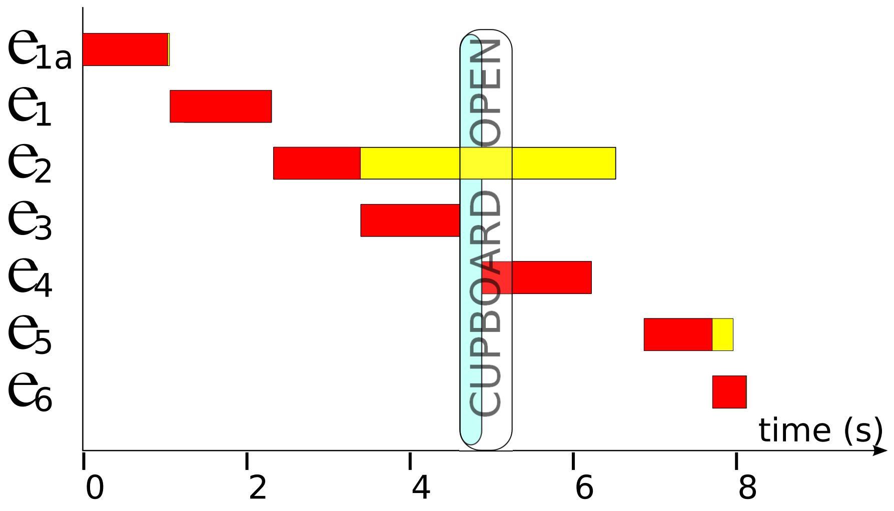


Figure 5.8: Reference behavior of the robot. Dark area corresponds to regulation period; light areas correspond to maintaining periods. The light blue zone represents an incompressible delay.

Considering that the robot opens the cupboard in 2s, the client can walk at most at $v^I_{h,\max}$ =1.625m/s without requiring an adaptation of the robot (according to the eq. (5.2)). The Fig. 5.9 represents the evolution of the norm of the control law in function of the velocity of the client, for constant velocities varying between 1 and 5m/s. Two behaviors are noticeable: for a velocity inferior to $v^I_{h,\max}$, only the timing of the task changes while the behavior remains untouched. At the opposite, for a velocity superior to $v^I_{h,\max}$, the sequence is started as soon as possible (as soon as the robot has understood the situation), and the motion is accelerated.

Using this reference parameterization, the two adaptive methods are tested on four trajectories. The first adaptive method, M_1, corresponds to the minimal adaptation, the second one, M_2, corresponds to the smooth adaptation. In three of the configurations tested, the

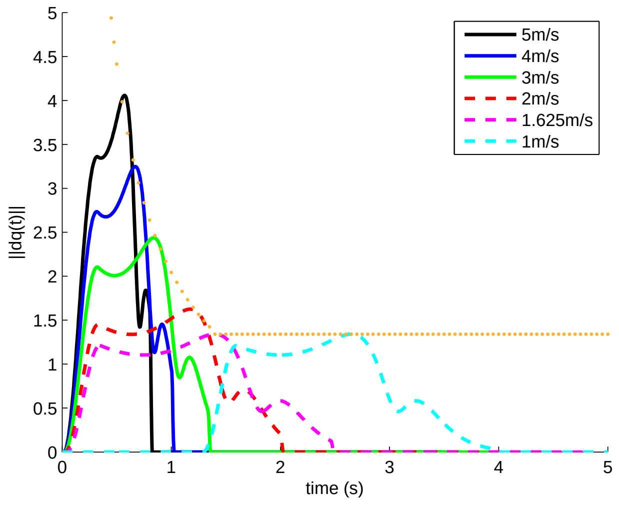


Figure 5.9: Evolution of the norm of the control law $\|q\|$ with respect to the time for several values of v_h^I. The red curve represents the evolution of $\|q\|$ for the limit velocity ($v_{h,\max}^I = 1.625$m/s): the motion realized is the default one and it is started at $t = 0$. When the velocity of the client is inferior to this limit, the motion realized by the robot is the same, but it starts later (cf. the pink curve, for $v_h^I = 1$m/s). When his velocity is superior to the limit, the default motion cannot be realized as such: it has to be accelerated, which appears as a compression of the curve on the graph. The evolution of the maximal value of the norm is estimated by the orange dotted line. For $t < 1.4$s, it is inversely proportional to the velocity, otherwise it is constant.

client walks at constant velocity, which can be slow (1m/s), medium (2m/s) or high (3m/s). In the fourth case, the client walks at the maximal velocity (1.625m/s), pauses, waits one second and starts again. In the following tests, we analyze the success of the mission depending on the method chosen and the velocity of the client. A mission is successful if the robot managed to open the cupboard before the client is at minimal distance of the robot. In this case, the negative delay represents the advance the bellboy has relatively to the client. Otherwise, a positive value indicates the delay between the bellboy and the client.

Method	Client velocity	t^I (seconds)	$d(t^I)$ (meters)	t^F (seconds)	$d(t^F)$ (meters)	delay (seconds)	Success
M_1	Slow	0.010	4	2.01	3.130	-1.290	Yes
	Casual				0.750	0.335	No
	Fast				0.750	0.875	No
	Stop				1.563	-1.040	Yes
M_2	Slow	1.290	2.714	3.290	0.760	-0.010	Yes
	Casual	0.025	3.959	1.665	0.750	0.035	No
	Fast	0.020	3.953	1.170	0.750	0.080	No
	Stop	0.030	3.958	2.985	0.781	-0.020	Yes

Table 5.1: Results of the adaptation of the gain using the two proposed methods.

The Table 5.1 gathers the results of each simulation. In the slow case, both methods succeed in satisfying the constraint. This result was predictable, since the behavior adopted by both of them is the default one. The only difference between the two methods is the timing: M_1 starts the sequence immediately, while M_2 starts as soon as the limit distance is reached. Similarly, when the human stops, both methods work. The method M_1 works since the default behavior works, even without the additional delay due to the pause of the client. Using the adaptive method M_2, the robot also pauses (cf. Fig. 5.10b), and succeeds in finishing in time.

Yet, for both methods, the deadline constraint imposed by the client is violated when the client walked faster than the maximal authorized velocity. In this case, the time dependent method introduces delays up to 0.87 seconds, which is quite important. The delay presented in the second method is smaller (less that 0.1 second), hence we believe it may not be noticed by the client during the experiments.

5.4 Conclusion

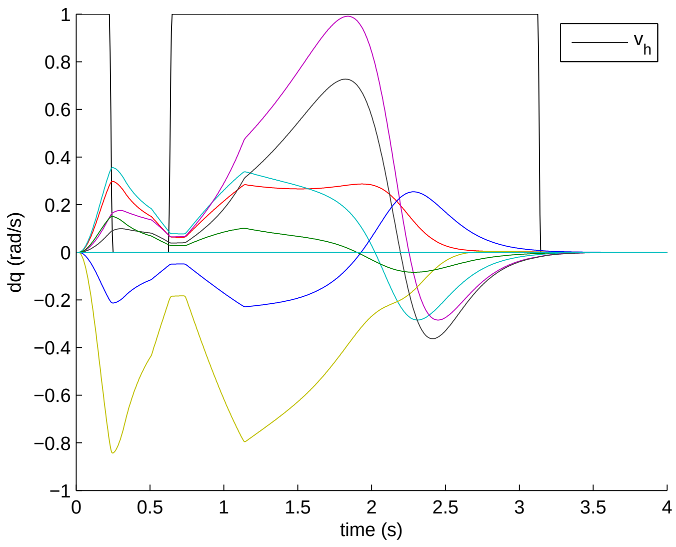
This chapter highlights two main important issues:

First, in the provision of extending this work to human-robot interaction or cohabitation, it is important to investigate whether what the tools and methods presented in previous chapter are still viable or not. The good news is that the task function and the stack-of-task can be maintained as a component of the architecture, but their scheduling must consider other constraints. Some parameters can even be set not from the optimization process, but from another database of robotic 'know-how' and 'manners' and also from human preference that can be observed by the robot or given through interaction (instructions from the human).

Second, these human expectations and preferences when obtained from observation can be heavy to monitor. We conducted a pilot simple task-case study to try finding the way human expectations can be programmed on the robot. The idea is to track a similar scenario occurring between two humans, process the data seeking for invariant and try to find scheduling criterion and relationships. It appears that such an approach can be demanding in time but can provide interesting knowledge that can be integrated by allowing the behavior of the task to be adaptable in function of the sensory observation. We showed in our example how this can be made, but still, although the methodology is clear, its extension to more general cases is certainly questionable.

Experiments on the HRP-2 are ongoing to assess real user's opinion.

This chapter ends our contributions. The following chapter concludes this thesis by summing up the work presented together with some perspectives.



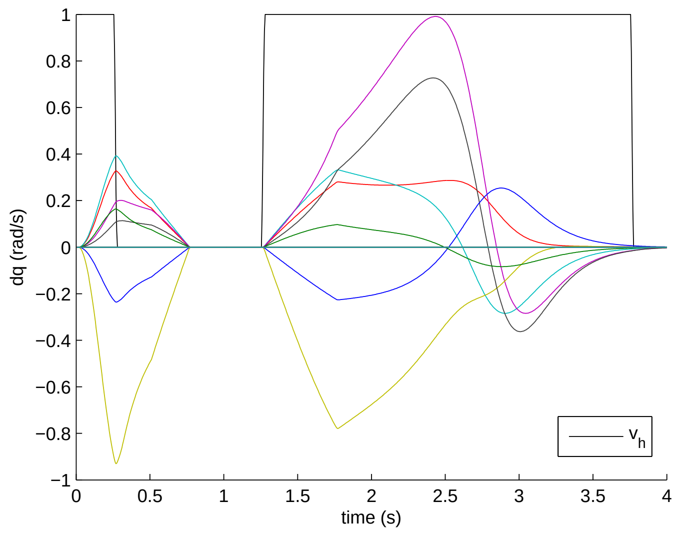


Figure 5.10: Evolution of the control when the client stop walking. The black curve represents the velocity of the target v_h. In a), the pause lasts 0.4s; the robot does not stop its motion, but slows down. In b), the pause lasts 1s and the robot stops and restart its motion smoothly.

Conclusion and Perspectives

There is a considerable amount of work and methods in robotics on mission and motion planning. However, it appears that a general agreed approach is to structure a robotic mission into three steps. The first step is the task planning, which decomposes the mission into a sequence of tasks. It appears more plausible, for several obvious reasons, that these tasks are predefined or learned and selected from a database describing the capabilities of the robotic agent. The second step is a merge between the task sequencing per se and the task scheduling: while the sequencing determines the order in which the tasks are realized, the scheduling associates a timing for each of them, accounting for all the physical constraints of the robot. The third step is the robotic execution of the task sequence.

This approach is adapted to robotics, but also to more general planning/scheduling problems that are encountered in wider applications. Yet, in robotics and particularly in humanoid robotics, the scheduling phase is usually skipped, because classical schedulers are not conceived to handle the case where the motion is defined not by one, but by a combination of several tasks organized into a hierarchy.

In this thesis, we proposed to really see the scheduling as the missing stage between planning and execution, bridging thus the gap between symbol (the plan) and numeric (the execution). This bridge comes naturally when considering the task-function formalism. We then proposed a method to optimize not only the scheduling but also the behavior of task sequences on the basis of a task-function based controller and a classical task planner. This optimization takes in input only symbols (the order of the sequence), but outputs the whole necessary numerical parameters to specify the controller, accounting for the physical constraint of the execution. Moreover, we have shown that it is even possible to define the task behavior in this step to take into account user preferences. Optimizing a task sequence is not limited to the acceleration of the entire motion while keeping the proposed ordered sequence from the planner: it is first necessary to operate on the task sequence and take advantage of the redundancy of the robot to realize smooth task overlaps when possible. The originality of our proposed method lies in maintaining the task component in the entire reasoning, and using it as a bridge between planning, scheduling and execution.

Contributions

The main advantage of high redundant robots (such as humanoid robots) is their capacity to realize several tasks simultaneously. Among the existing methods enabling task overlapping, we chose to rely on the stack-of-tasks mechanism, so as to strictly respect the priorities between the tasks. For a given set of tasks organized into a hierarchy, classical multi-function task execution based on the stack of tasks ensures a continuous evolution of the control, even in singular cases.

However, discontinuities in the control law are noted when discrete operations peculiar to a stack –in the computer science sense– are made. Namely, these operations are the insertion, removal and the swap of priority between tasks. These abrupt changes in the control output can be problematic (not to say dangerous). Consequently, we defined methods to ensure a smooth evolution of the control during these events. The smoothing processes restrict the sphere of operation of the events: a task can only be removed from and inserted at the end of the stack, and the swap should only be realized between neighboring tasks and requires a smoothing operation if the tasks conflict. As a result, a smooth evolution of the control law can be ensured in any situation. Yet, the delay introduced by the smoothing processes (namely the swap propagation process) is a compromise and even a drawback that one must account for.

The second main contribution is the definition of an optimization formulation for the problem of task sequence overlapping. This phase takes place between the planning of the task sequence and its execution, and gathers the scheduling phase and the improvement of the task behavior. Both the timing and the gains are optimized in order to improve the initial task sequence while fulfilling the constraints imposed by both the sequence, the intrinsic robot limitations, and the environment. We provided a complete implementation of the proposed approach that serve both an advanced interactive simulator (AMELIF) and a robot architecture. The optimization process is based on a dialog solver/simulator: the constraints cannot be computed directly, and can only be evaluated by simulating the task sequence for each set of parameters given by the solver. Therefore, a reliable simulator is an important component of the optimization process. Similarly, it is impossible to estimate the gradient of the constraints, which are rather evaluated by finite difference.

This method was exemplified on a real scenario with the HRP-2 humanoid robot. Some temporal constraints, preventing task overlapping were replaced by collision-avoidance constraints. This shows that a proper definition of the constraints is important. Then the task sequence has been optimized using the proposed method: it enhanced the timing of each task, by suppressing the dead times (i.e. removing the tasks as soon as regulated), but also accelerated the resulting motion (again, at the task level) in order to meet the joint velocity limits that were imposed.

Yet, this method faces two issues liable to prevent the solver from finding the optimal solution. The first one is the *a posteriori* evaluation of the constraints: each set of parameters is directly played by the simulator, without any verification of the coherency of the resulting motion. The resulting constraints may hence lack of physical sense and jeopardize the optimization process. The second one is due to the discontinuity of the evolution of the constraints. Indeed, the chosen solver, CFSQP, is conceived to work with continuous and derivable criterion and constraints. Yet, the smoothing of the control law may penalize the optimization process, because of the discontinuities due to modifications in the order of events.

Finally, we introduced a new perspective in the usage of the task-function approach, consisting in personalizing (to a human user) a task sequence, i.e. adapting it to the constraints of the situation, typically the requirements of a human operator. This property is exemplified by an example on the HRP-2, where the robot motion was modeled to adapt or rather adjust and auto-tune on the basis of natural behavior observed on the human user.

Perspectives

In the short term, the possible enhancements of this work are mainly functional. The straightforward ones would consist in extending the current optimization problem, by considering different type of tasks (such as tasks that maintain a state during a whole period, or perception tasks), simple modification of the definition of the time of realization and the associate constraint – or different types of behavior. Also, the influence of way-point tasks could be part of the optimization process. These tasks are used to ‘deviate’ from a straight line some trajectories of operational points by locally attracting them. By considering the tolerance on the task regulation as a parameter, the trajectory followed by the operational point can be controlled.

At the medium term, it would be necessary to reformulate or complete part of the optimization problem. Typically, it is necessary to enhance the convergence properties of the optimization problem and reduce the computational time of the optimization, especially in order to handle more complex sequences, for which the estimation of the gradient by finite difference is too time consuming.

The sequences optimized in this thesis were realized on the upper part of a static humanoid robot. Realizing tasks overlapping while walking requires the add of constraints such as the stability criteria and the motor constraints, whose formulation is similar to the robot constraints already considered. Though, considering the example of a grasping task that should start before the end of the walk, one can figure that these constraints may not be sufficient to express the correct behavior: a robot walking with its arms stretched may satisfy the constraints on the motion, but is not natural. This natural-looking condition may be formulated by the minimization of the torques during the whole motion, together with the minimization of the time spent. In other words, multiple objective optimization problems should be considered.

For the long term, some improvements are also possible in this optimization process. The first one intervenes upstream and consists in adapting the sequencing phase in order to handle priorities between tasks. The second one consists in linking it to the surrounding context by binding the task function approach to the control-law approach.

Enabling task overlapping makes critical the choice of the respective priority of a set of tasks, because this hierarchy between the tasks defines the computed motion: a bad choice can lead to sub-optimal motions, or worse, cause the failure of the given plan. In this thesis, when the sequence was only partially ordered, the sequencing phase was realized together with the scheduling phase. Hence, in the same way as the task schedule implies an order between the tasks rather than deciding it, it implies a hierarchy between the tasks, by associating to the first inserted task the higher priority, and to the last one the lower priority. This approach is not appropriate (in the frame of the optimization), especially when the control law is smoothed. To prevent the discontinuities due to the swap of the order of the events and ensure the convergence of the optimization process, the (rough) solution adopted consisted in deciding once and for all of the task order (together with their priority). Instead, it would be best to reintroduce the task sequencing phase and to extend it in order to take account of the priorities between the tasks. This comes down to adding the swap operation to the set of basic operations considered during the sequencing phase (for now there is only the task insertion and the task removal).

During the optimization of the can grasping mission, the main enhancement of the schedule was made possible by replacing the temporal constraints conditioning the motions of the arms by collision-avoidance constraints. This example shows that a way to enable the optimization consists in finding the temporal constraints that can be relaxed and replacing them by other constraints (e.g. depending on the environment). Defining manually this replacement may be burdensome, and it would best to find these openings automatically. Yet, this approach –consisting in defining the task sequence with respect to events rather than exclusively by a schedule–, is the approach realized by the control-law method mentioned in the first chapter. Indeed, it is based on a graph of control laws, whose edges correspond to *smooth* transitions between them. Defining such a graph is now possible with task-function approach, ensured to be smooth thanks to the method defined earlier. As a result, binding the task-function and the control-law approaches may be a good way to enable the optimization process.

Appendix A

Demonstration of the continuity of the weighted pseudo inverse method

Since the insertion and the removal of a task are very similar mechanisms, we will only detail the proof of the continuity during the removal of a task.

Consider a stack of tasks containing n tasks ($\mathbf{e_i}$ has priority on $\mathbf{e_{i+1}}$). We prove that, regardless of the priority of the task removed $\mathbf{e_{p,p\in[1\cdots n]}}$, the control law $\mathbf{\dot{q}_i}$ associated to any task $\mathbf{e_{i,i\in[1\cdots n]}}$ in the stack is continuous.

To this purpose, we prove by recurrence the following proposition:

$$\begin{array}{ll} (\mathcal{P}_i): & \forall p \in [1\cdots n], \mathbf{\dot{q}_i} \text{ is continuous} \\ \text{where} & \mathbf{e_p} \text{ is the task removed,} \\ \text{and} & \mathbf{\dot{q}_i} = (\mathbf{J_i P_{i-1}})^{\#\mathbf{w_i}}(\lambda_i^{Ins}(\mathbf{\dot{e}_i^*} - \mathbf{J_i \dot{q}_{i-1}})) \end{array}$$

A.1 Hypotheses

$\mathcal{H}$ **A.1.1** *When a task* $\mathbf{e_i}$ *is fully removed at* t_i^F*, its insertion gain at* t_i^F *is 0:*

$$\lim_{t \to t_i^{F-}} \lambda_i^{Ins}(t) = 0$$

$\mathcal{H}$ **A.1.2** *When a task is partially removed at* t^-*, then*

$$\begin{cases} \lim_{t\to t^-} \mathbf{w_i}(t)|_{a_i^F} \to 0 \\ \lim_{t\to t^-} \mathbf{w_i}(t)|_{\neg a_i^F} \to 1 \end{cases}$$

A.2 Proof

To simplify the notations, we consider that the level of a task is constant. In other words, considering the stack of tasks containing $\mathbf{e_1}|\mathbf{e_2}$, the removal of $\mathbf{e_1}$ will result in the stack of task $\mathbf{0}|\mathbf{e_2}$ noted $\mathbf{e_2}$. We distinguish different cases:

The task $\mathbf{e_i}$ has priority upon $\mathbf{e_p}$: $i < p$

Since the task $\mathbf{e_p}$ has no influence on higher priority tasks, $\mathbf{\dot{q}_i}$ is continuous: $(\mathcal{P}_i)$ is true.

The task removed is the task $\mathbf{e_i}$: $i = p$

Using $\mathcal{H}$[A.1.1], we have

$$\lim_{t \to t_i^{F-}} \mathbf{\dot{q}_i}(t) = 0 \text{ and } \lim_{t \to t_i^{F+}} \mathbf{\dot{q}_i}(t) = 0$$

The function is continuous: the proposition $(\mathcal{P}_i)$ is verified.

The task $\mathbf{e_p}$ has priority upon $\mathbf{e_i}$: $i > p$

We suppose $(\mathcal{P}_{i-1})$ verified. Thus, the continuity of $(\mathbf{\dot{e}_i^*} - \mathbf{J_i\dot{q}_{i-1}})$ is straightforward. We study the continuity of $(\mathbf{J_iP_{i-1}})^{\#\mathbf{w_i}}\lambda_i^{Ins}$.

Again, 3 cases are considered:
1. $(\mathcal{P}_{i,1}) : a_i = a_i^B$: the task does not need to be removed
2. $(\mathcal{P}_{i,2}) : a_i \cap a_i^B = \emptyset$: the task must be completely removed
3. $(\mathcal{P}_{i,3}) : a_i \cap a_i^F \neq \emptyset$: the task must be removed partially

The task $\mathbf{e_i}$ remains untouched: $a_i = a_i^B$

λ_i^{Ins} is a continuous function, $\mathbf{w_i} = diag[1 \ldots 1]$ and $\mathbf{W_i} = \mathbf{I}$.

$$(\mathbf{J_iP_{i-1}})^{\#\mathbf{w_i}} = (\mathbf{J_iP_{i-1}})^{\#\mathbf{I}}$$

Yet

$$\begin{aligned}
\mathbf{J_iP_{i-1}} &= \mathbf{J_i}\left(\mathbf{I} - \begin{bmatrix} \mathbf{J_1} \\ \vdots \\ \mathbf{J_{i-1}} \end{bmatrix}^+ \begin{bmatrix} \mathbf{J_1} \\ \vdots \\ \mathbf{J_{i-1}} \end{bmatrix}\right) \\
&= \mathbf{J_i} - \mathbf{J_i}\begin{bmatrix} \mathbf{J_1} \\ \vdots \\ \mathbf{J_{i-1}} \end{bmatrix}^+ \begin{bmatrix} \mathbf{J_1} \\ \vdots \\ \mathbf{J_{i-1}} \end{bmatrix} \\
\mathbf{J_iP_{i-1}} &= \mathbf{J_i}, \text{ since } \forall j < i, a_i \cap a_j = \emptyset
\end{aligned}$$

Thus

$$\begin{aligned}(\mathbf{J_iP_{i-1}})^{\#\mathbf{w_i}} &= (\mathbf{J_i})^{\#\mathbf{W_i}}\\ &= (\mathbf{J_i})^{\#\mathbf{I}}\\ &= \mathbf{J_i^T}(\mathbf{J_iJ_i^T})^{-1}\end{aligned}$$

$$\lim_{t\to t_i^{F-}}(\mathbf{J_iP_{i-1}})^{\#\mathbf{w_i}} = \mathbf{J_i^T}(\mathbf{J_iJ_i^T})^{-1}$$

For the same reason, we have:

$$\lim_{t\to t_i^{F+}}(\mathbf{J_iP_{i-1}})^{\#\mathbf{w_i}} = \mathbf{J_i^T}(\mathbf{J_iJ_i^T})^{-1}$$

$(\mathcal{P}_{i,1})$ is verified.

The task $\mathrm{e}_\mathbf{i}$ is removed: $a_i \cap a_i^B = \emptyset$

Considering $\mathcal{H}$[A.1.1], we have:

$$\lim_{t\to t_1^{F+}}\lambda_i^{Ins}(t) = 0 = \lim_{t\to t_1^{F-}}\lambda_i^{Ins}(t)$$

Thus

$$\lim_{t\to t_1^{F+}}\mathbf{\dot{q}_i}(t) = 0 = \lim_{t\to t_1^{F-}}\mathbf{\dot{q}_i}(t)$$

When the task is completely removed, $\mathbf{\dot{q}_i}$ is continuous at t_i^F.

$(\mathcal{P}_{i,2})$ is verified.

The task $\mathrm{e}_\mathbf{i}$ is partially removed: $a_i \cap a_i^F \neq \emptyset$

$$\begin{aligned}(\mathbf{J_iP_{i-1}})^{\#\mathbf{w_i}} &= \mathbf{W_i}(\mathbf{J_iP_{i-1}})^T(\mathbf{J_iP_{i-1}W_i}(\mathbf{J_iP_i})^T)^{-1}\\ &= \mathbf{W_iP_{i-1}^TJ_i^T}(\mathbf{J_iP_{i-1}W_iP_{i-1}^TJ_i^T})^{-1}\end{aligned}$$

Yet

$$\begin{aligned}\mathbf{P_{i-1}W_i} &= \left(\mathbf{I} - \begin{bmatrix}\mathbf{J_1}\\ \vdots\\ \mathbf{J_{i-1}}\end{bmatrix}^+ \begin{bmatrix}\mathbf{J_1}\\ \vdots\\ \mathbf{J_{i-1}}\end{bmatrix}\right)\mathbf{W_i}\\ &= \mathbf{W_i} - \begin{bmatrix}\mathbf{J_1}\\ \vdots\\ \mathbf{J_{i-1}}\end{bmatrix}^+ \begin{bmatrix}\mathbf{J_1}\\ \vdots\\ \mathbf{J_{i-1}}\end{bmatrix}\mathbf{W_i}\end{aligned}$$

Using $\mathcal{H}$[A.1.2]

$$\lim_{t\to t_i^{F-}}\mathbf{P_{i-1}W_i}[t] = \lim_{t\to t_i^{F-}}\mathbf{W_i}[t]$$

Likewise

$$\lim_{t \to t_1^{F-}} \mathbf{W_i P_{i-1}^T}[t] = \lim_{t \to t_i^{F-}} \mathbf{W_i}[t]$$

Thus

$$\lim_{t \to t_i^{F-}} (\mathbf{J_i P_{i-1}})^{\#\mathbf{w_i}}[t] = \lim_{t \to t_i^{F-}} \mathbf{W_i J_i^T}(\mathbf{J_i W_i J_i^T})^{-1}[t]$$

When $t \to t_p^{F+}$

$$\mathbf{P_{i-1} W_i} = \mathbf{W_i} - \begin{bmatrix} \mathbf{J_1} \\ \vdots \\ \mathbf{J_{p-1}} \\ \mathbf{J_{p+1}} \\ \vdots \\ \mathbf{J_i} \end{bmatrix}^+ \begin{bmatrix} \mathbf{J_1} \\ \vdots \\ \mathbf{J_{p-1}} \\ \mathbf{J_{p+1}} \\ \vdots \\ \mathbf{J_i} \end{bmatrix} \mathbf{W_i}$$

Using $\mathcal{H}$[A.1.2], we have:

$$\mathbf{P_{i-1} W_i} = \mathbf{W_i P_{i-1}^T} = \mathbf{W_i}$$

$$(\mathbf{J_i P_{i-1}})^{\#\mathbf{w_i}} = \mathbf{W_i J_i^T}(\mathbf{J_i W_i J_i^T})^{-1}$$

$$\lim_{t \to t_i^{F-}} (\mathbf{J_i P_{i-1}})^{\#\mathbf{w_i}} = \lim_{t \to t_i^{F+}} (\mathbf{J_i P_{i-1}})^{\#\mathbf{w_i}}$$

$(\mathcal{P}_{i,3})$ is verified.

Conclusion

The proposition $(\mathcal{P}_i)$ is verified in the case where the removed task $\mathbf{e_p}$ has priority upon $\mathbf{e_i}$.

General conclusion

As a conclusion, whatever the priority of the task removed is, the control law associated to the other tasks in the stack is continuous at the time of removal t_p^F.

Hence, $\dot{\mathbf{q}}$ is continuous on $\mathbb{R}$.

Appendix B

Dynamic motions using equality and inequality constraints

The work presented in this appendix is a shortening of the paper [77]. It introduces the work realized by Layale Saab, from the LAAS (Laboratoire d'Analyse et d'Architecture des Système) at Toulouse, on which I collaborate.

The objective of this work is to propose a solution to generate full-dynamic motions for the humanoid robot, accounting for various kind of constraints such as dynamic balance or joint limits. In a first time, a unification of task-based control schemes was proposed, in inverse kinematics or inverse dynamics. Based of this unification, the hierarchy of tasks based on a cascade of quadratic programs that was developed in LAAS for inverse kinematics only, is applied for inverse dynamics. We applied this solution to generate in simulation whole-body motions for a humanoid robot in unilateral contact with the ground, while ensuring the dynamic balance on a not horizontal surface.

The classical formulations for the stack-of-task computations are based on pseudo inversions (as done all over this thesis) [58, 83, 43]. However, it is a well-recognized fact that, beyond this computationnal formulation, the inverse-kinematic formulation can be written as a quadratic problem where the goal is to find a norm-minimizer under linear equality and inequality constraints in configuration space. It was recently proposed [42] a specific way to consider such quadratic problems that enables to find an approximate solution based on successive prioritization even when inequality constraints forbid an exact solution.

Whole-body motion is naturally concerned with dynamics and contact forces. A dynamical formulation is necessary and task dynamics writes as linear equalities whose unknowns are the generalized torques. Several approaches consider prioritization techniques within a dynamic formulation. This is particularly the case for the works by Khatib and colleagues [81, 71, 45]. Recently, various techniques have been developed for taking into account exactly unilateral constraints due for instance to joint limits or multiple contacts: [62] proposed an original scheme to compute a generic control law from a hierarchic set of both unilateral constraints and bilateral tasks and [16] solves dynamic and static quadratic problems for multi-contact.

The work presented below exploits the quadratic nature of the dynamic formulation including unilateral contact constraints in order to take into account equality and inequality constraints in a way similar to the one proposed at kinematic level in [42].

B.1 Dynamic inverse using QP-based stack of tasks

B.1.1 Task function approach

A task is defined by the task space $\mathbf{e}$, the reference to be applied in this task space $\dot{\mathbf{e}}^*$, and by $\mathbf{G}$, the differential link between the task space and the robot actuators:

$$\dot{\mathbf{e}} + \boldsymbol{\mu} = \mathbf{G}\mathbf{u} \tag{B.1}$$

where $\boldsymbol{\mu}$ is the drift of the task. The reference $\dot{\mathbf{e}}^*$ is in the tangent space to $\mathbf{e}$, and the differential link $\mathbf{G}$ maps the elements of the tangent subspace of the robot configuration to the tangent space to $\mathbf{e}$.

This is the same equation as in Chapter 2, but using specific notations to make the link with the dynamics. In inverse kinematics, the control input $\mathbf{u}$ is simply the robot joint velocity $\mathbf{u} = \dot{\mathbf{q}}$. The differential link is the task Jacobian $\mathbf{G} = \mathbf{J}$. In that case, the drift $\boldsymbol{\mu}$ is null.

The differential link $\mathbf{G}$ gives the direct link between the actuator $\mathbf{u}$ and the feedback $\mathbf{e}$: from a given robot motion, $\mathbf{G}$ gives the reaction in the task space. To compute a specific robot control $\mathbf{u}$ that performs the reference $\dot{\mathbf{e}}^*$, any numerical inverse of $\mathbf{G}$ can be used. The generic control law is then

$$\mathbf{u} = \mathbf{G}^{\#}(\dot{\mathbf{e}}^* + \boldsymbol{\mu}) + \mathbf{P}\mathbf{u_2} \tag{B.2}$$

where $.^{\#}$ is a matrix-inverse operator, $\mathbf{P}$ is the projection and $\mathbf{u_2}$ is any secondary objective to be considered without disturbing the main objective.

Once more, this is the same equation as in the thesis. For kinematics inverse, the matrix inverse operator is most of the time the pseudo inverse [6, 33]. The control law is then:

$$\dot{\mathbf{q}}^* = \mathbf{J}^{+}\dot{\mathbf{e}}^* + \mathbf{P}\dot{\mathbf{q}}_2 \tag{B.3}$$

The template equation (B.2) can be similarly used to compute the control law $\mathbf{u_2}$ accounting for the secondary task:

$$\dot{\mathbf{e}}_\mathbf{2} - \mathbf{J_2}\dot{\mathbf{q}}^* = \mathbf{J_2}\mathbf{P}\dot{\mathbf{q}}_2 \tag{B.4}$$

The differential link is the projected Jacobian $\mathbf{G} = \mathbf{J_2}\mathbf{P}$, and the drift is $\boldsymbol{\mu} = -\mathbf{J_2}\dot{\mathbf{q}}^*$. The same inversion can then be directly applied to obtain the stack-of-tasks recurrence equation.

B.1.2 Inverse dynamics

In dynamics, the input of the system is the robot motor torques $\mathbf{u} = \boldsymbol{\tau}$. The state of the robot is the pair $(\mathbf{q}, \dot{\mathbf{q}})$, and the task space comprehends both the position and velocity. The reference $\dot{\mathbf{e}}^*$, homogeneous to an acceleration, is denoted $\ddot{\mathbf{e}}$. Contrarily to the kinematic case, the map to the control input can not be built directly, but is obtained from two stages: first the dynamic equation of the system, typically:

$$\mathbf{A}\ddot{\mathbf{q}} + \mathbf{b} = \boldsymbol{\tau} \tag{B.5}$$

where $\mathbf{A}$ is the generalized inertia matrix and $\mathbf{b}$ is the dynamical drift. The acceleration $\ddot{\mathbf{q}}$ is linked to the task space by the task Jacobian:

$$\mathbf{J}\ddot{\mathbf{q}} + \dot{\mathbf{J}}\dot{\mathbf{q}} = \ddot{\mathbf{e}} \tag{B.6}$$

Multiplying (B.5) by $\mathbf{J}\mathbf{A}^{-1}$, the differential link between $\boldsymbol{\tau}$ and $\ddot{\mathbf{e}}$ is obtained:

$$\ddot{\mathbf{e}} + \dot{\mathbf{J}}\dot{\mathbf{q}} + \mathbf{J}\mathbf{A}^{-1}\mathbf{b} = \mathbf{J}\mathbf{A}^{-1}\boldsymbol{\tau} \tag{B.7}$$

This corresponds to the template (B.2) with $\mathbf{G} = \mathbf{J}\mathbf{A}^{-1}$ and $\boldsymbol{\mu} = \dot{\mathbf{J}}\dot{\mathbf{q}} + \mathbf{J}\mathbf{A}^{-1}\mathbf{b}$.

The dynamic-inverse control law is then directly obtained by inverting $\mathbf{G}$. To fit with the dynamic of the system, it has been proposed [44] to use the weighted pseudo inverse, with $\mathbf{A}$ as weights of the right-hand side [22]:

$$\boldsymbol{\tau}^* = (\mathbf{J}\mathbf{A}^{-1})^{\#\mathbf{A}}(\ddot{\mathbf{e}} + \dot{\mathbf{J}}\dot{\mathbf{q}} + \mathbf{J}\mathbf{A}^{-1}\mathbf{b}) + \mathbf{P}\boldsymbol{\tau}_{\mathbf{2}} \tag{B.8}$$

As in the kinematic case, $\mathbf{P}$ represents the redundancy of the system with respect to the task $\mathbf{e}$. $\boldsymbol{\tau}_{\mathbf{2}}$ can be used to perform a second task $\mathbf{e}_{\mathbf{2}}$, similarly as before.

B.1.3 Inequalities in the loop

We have seen that both kinematics- and dynamics-based problems can be put under a same shape (B.2). Now, since this common shape can be solved using a cascade of quadratic programs accounting for both equalities and inequalities [42], this means that the cascade of QP can be also applied in dynamics. Since quadratic solvers are able to handle indifferently equalities and inequalities, it is then possible to have both inequalities and equalities in the task definition. The reference part is then rewritten:

$$\underline{\dot{\mathbf{e}}}^* \leq \dot{\mathbf{e}} \leq \overline{\dot{\mathbf{e}}}^* \tag{B.9}$$

with $\underline{\dot{\mathbf{e}}}^* = \overline{\dot{\mathbf{e}}}^*$ in the case of equalities, and $\underline{\dot{\mathbf{e}}}^* = -\infty$ or $\overline{\dot{\mathbf{e}}}^* = +\infty$ to handle single-bounded constraints. When considering a single task, the inversion (B.2) corresponds to the optimal solution of the problem:

$$\min_{\mathbf{u}} \|\mathbf{G}\mathbf{u} - \dot{\mathbf{e}}^* - \boldsymbol{\mu}\|^2 \tag{B.10}$$

It is straightforward to introduce inequality constraints into a quadratic program. However, this would introduce also a de-facto hierarchy between the inequalities part and the equalities.

It was then proposed [42, 37] to rely on slack variables. The quadratic program for both equalities and inequalities is then written:

$$\begin{aligned} &\min_{\mathbf{u},w} \|w\|^2 \\ s.t.\quad &\underline{\dot{\mathbf{e}}}^* \leq \mathbf{G}\mathbf{u} - \boldsymbol{\mu} + w \leq \overline{\dot{\mathbf{e}}}^* \end{aligned} \tag{B.11}$$

with $\underline{\dot{\mathbf{e}}}^* = \overline{\dot{\mathbf{e}}}^*$ for the equality parts of the task. The effect of the slack variable is to relax the parts of the task that are not feasible, and therefore it ensures that the task is fulfilled at the best (in the sense of the norm of the rest).

When the first task is solved, it was proposed [42] to use the optimal slack denoted w^* to introduce a secondary task in the hierarchy. After resolution of the first quadratic program, a secondary task is solved by

$$\begin{aligned} &\min_{\mathbf{u},w_2} ||w_2||^2 \\ s.t.\quad &\underline{\dot{\mathbf{e}}}^* \leq \mathbf{G}\mathbf{u} - \boldsymbol{\mu} + w^* \leq \overline{\dot{\mathbf{e}}}^* \\ &\underline{\dot{\mathbf{e}}}_{\mathbf{2}}^* \leq \mathbf{G_2}\mathbf{u} - \boldsymbol{\mu_2} + w_2 \leq \overline{\dot{\mathbf{e}}}_{\mathbf{2}}^* \end{aligned} \tag{B.12}$$

In this secondary program, the first slack w^* is not variable anymore. Indeed, the first task has now priority, and should be solved at least as accurately as done by the first program. On the other hand, the second task has no priority upon the first. If the two tasks are not compatible, the second task will be relaxed, and then less accurately executed, due to its slack variable. Similarly, the second slack can be used to introduce a third task, and iteratively, any number of tasks.

Based on the generic writing (B.2), the cascade of QP above can be applied directly for both kinematics and dynamics.

B.2 Experiments

B.2.1 Task-set for dynamic inverse

The first experiments already realized are simply based on the regulation of the posture under constraint of permanent contact. The first task is then basically the regulation of the posture of the robot. The task space is equal to the actuated-joint space, and the desired acceleration in this space is a proportional derivative to a given reference position at null velocity:

$$\ddot{\mathbf{q}}^* = -\lambda_P(\mathbf{q} - \mathbf{q}^*) - \lambda_D\dot{\mathbf{q}} \tag{B.13}$$

In that case, the Jacobian $\mathbf{J}$ is simply the selection matrix $\mathbf{S}$ that separates the actuated links from the free-floating non-actuated degrees of freedom.

The second task $\mathbf{e_{bal}}$ ensures an immediate balance control, by preventing the contact points to take off the ground. The task space is the space of the forces normal to the contact, and the task is to prevent them to reach 0:

$$f_\perp = \mathbf{S}_\perp \mathbf{J_c}^{\#T}(\mathbf{S}^T\boldsymbol{\tau} - \mathbf{b}) \leq \epsilon \tag{B.14}$$

where $f_\perp$ are the normal components of the contact forces f_c, $\mathbf{S}_\perp$ is the selection matrix of the corresponding lines of the contact Jacobian $\mathbf{J_c}$. The right-hand side parameter ϵ is user defined, to ensure a security margin: if chosen close to 0, the robot can perform more dynamic movements, but that are less robust in case of perturbation. We have used $\epsilon = 10N$ in the experiments.

It is possible to show that this last constraint is equivalent to the well-known ZMP constraint [88, 39] when all the contact points are planar and horizontal.

B.2.2 Results

The experiments have been performed in simulation using the dynamic simulator AMELIF. The control law has been integrated in the control framework SoT, using the dedicated inequalities solver developed for inverse kinematics [25]. We reproduced a well known experiment of physiology: the subject is asked to follow an oscillatory reference with the legs. When the oscillations frequency or amplitude increase, the required acceleration increases, until the natural contact constraint is saturated. An opposite oscillation then naturally appears on the chest to counteract the oscillation of the legs, and preserve the constraint. When put on a force sensor, the subject ZMP was shown to present saturation at the maximum of the amplitude.

The robot is put on one leg. An oscillatory acceleration is given has a reference, that requires the whole body to remain static, except for one joint of the support leg. The amplitude of the acceleration is then increased until saturation of the support constraint. The experiment is summed up on Fig. B.1 and Fig. B.2. The robot configuration at the maximum of the oscillation is shown on Fig. B.1. The robot is bended on its left, with the hip roll axis moving. The balance-constraint saturation comes both from the bending (the center of mass of the robot is far from the support polygon), and from the acceleration required to inverse the velocity and come back to the rest position. The Fig. B.2a gives the normal forces at the four corners of the foot during the motion. The minimal acceptable force is set to 10N. Around iteration 1000, the force corresponding to the front right of the foot comes to saturation. This correspond to the time of maximal acceleration. The Fig. B.2b gives the acceleration of the hip joint (roll) and chest joint (yaw). The chest joint is required not to move. However, when the contact constraint comes to saturation, this part of the task becomes infeasible. Therefore, the chest is used to compensate for the motion of the hip, and prevent the foot to move off the ground. when the acceleration of the hip decreases, the contact constraint leaves the saturation area, and the chest comes back to a 0 acceleration.

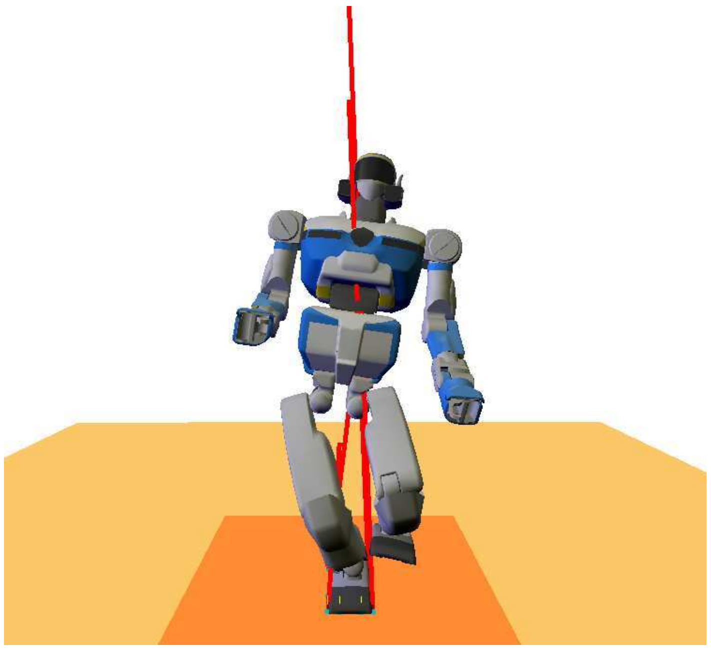
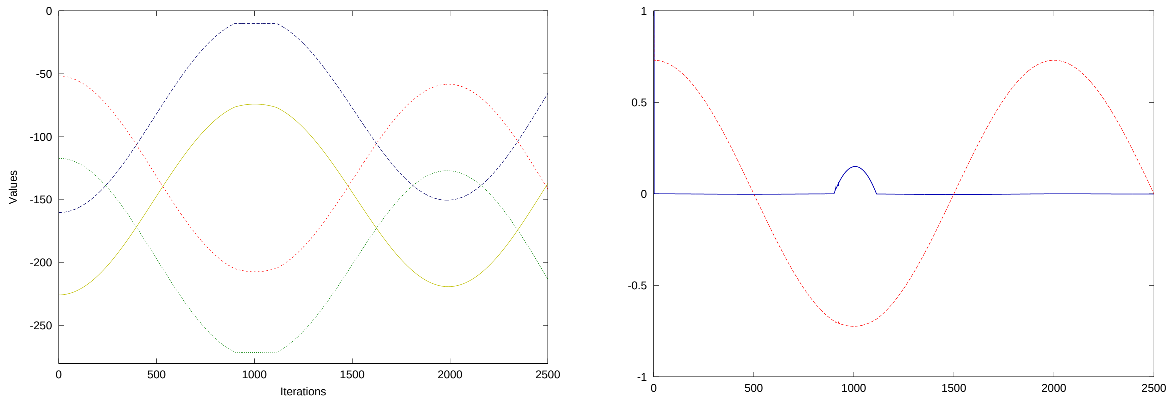


Figure B.1: Position of the robot at iteration 1000, when the maximal acceleration is reached. The red strikes correspond to the forces applies by the ground on the robot. The front forces are smaller, with the front-right force nearly null.



(a) (b)

Figure B.2: (a) Normal forces to the ground during the motion. (b) Acceleration of the hip roll joint (red) and chest yaw joint (blue) during the motion.

Appendix C

Résumé

C.1 Définition du problème

Parmi les différentes méthodes permettant à un robot de réaliser une mission, la plus générale consiste à décomposer la mission en trois étapes : planification, ordonnancement et exécution. Le but de cette thèse est d'utiliser cette décomposition en considérant la tâche comme fil conducteur entre les trois étapes.

Approche en trois étapes

La décomposition d'une mission en trois étapes n'est pas une méthode spécifique à la robotique, on y distingue :

- La phase de planification qui, à partir d'un ensemble de compétences, construit un plan de tâches symboliques et précise l'ordre logique des tâches. Ce plan peut être partiellement ordonné (une partie seulement des dépendances entre tâches est défini) ou au contraire totalement ordonné.

- La phase d'ordonnancement, dont le rôle est de définir le timing de chaque tâche. Ce timing doit respecter l'ordre entre les tâches et prendre en compte la disponibilité des ressources qu'elles utilisent. En général, on peut y distinguer deux phases : la phase de séquencement, dont le rôle est d'enlever les ambiguïtés restant dans le plan de tâches en fixant l'ordre des tâches, et la phase d'ordonnancement (à proprement parler), qui détermine le timing des tâches.

- La phase d'exécution, qui consiste à réaliser le plan de tâches, et à s'adapter voire à réparer le plan de tâches en cas de changements ou de problèmes par rapport au plan de tâches d'origine.

Adaptation à la robotique

En robotique, la tâche peut être définie comme un ordre directement exécutable par le robot, en opérant sur les moteurs et/ou les capteurs. On distingue les tâches liées aux actions, qui utilisent les capteurs du robot mais ne modifient pas sa posture, des tâches de mouvements, qui agissent sur les moteurs et modifient la posture du robot. En utilisant cette définition de la tâche, il est possible d'utiliser les méthodes classiques de planification, d'ordonnancement et d'exécution ou de définir d'autres algorithmes spécifiques à la robotique qui seront plus efficaces.

Lier la planification à l'exécution

Pour exécuter un plan de tâches, il est nécessaire de passer des données symboliques fournies par la planification à des données numériques. La perte des ces symboles cause une perte de connaissances, qui peut diminuer la robustesse du plan par rapport aux incertitudes de l'environnement. Afin de limiter cette perte d'informations, une méthode consiste à travailler sur le même élément de la planification à l'exécution. Deux solutions sont envisagées :

- L'utilisation de la fonction de tâche permet à la fois de définir des données haut niveau en utilisant la logique formelle (planification) mais aussi des données bas niveau permettant de calculer la commande (exécution). Ces tâches relient directement l'espace de travail du robot à l'espace de la tâche, et ne définissent une commande que sur une partie du robot. Il est alors possible de réaliser plusieurs tâches simultanément en tirant profit de la redondance du robot, et de les organiser selon une hiérarchie, en les classant par ordre de priorité.

- L'approche basée commande consiste à définir un graphe dont les nœuds sont l'ensemble des commandes réalisables par le robot, et dont les arcs sont les fonctions de transition entre elles. Cette méthode permet de créer une carte topologique de la mission : en fonction de l'environnement, une partie seulement du graphe est accessible. Cette approche est cependant moins adaptée à la superposition des tâches.

Réintroduction de la phase d'ordonnancement

La réalisation séquentielle des tâches d'un plan de tâches partiellement ordonné peut conduire à un mouvement saccadé et sous optimal. Le but de la réinsertion de la phase d'ordonnancement est de profiter de la redondance du robot pour réaliser plusieurs tâches simultanément.

Les approches classiques d'ordonnancement ne sont plus adaptées, dans la mesure où d'une part la hiérarchie entre les tâches introduit une dépendance non linéaire entre les tâches et les ressources (qui sont les degrés de liberté du robot et ses capteurs) et d'autre part le temps de réalisation des tâches devient variables.

L'approche considérée consiste à introduire une phase d'optimisation entre les phases de planification et d'exécution. A partir d'un plan de tâche donné, elle tend à améliorer le mouvement du robot en opérant à la fois sur le timing et la vitesse de réalisation des tâches.

C.2 Continuité de la pile des tâches

Introduction

Une tâche est définie par trois éléments : une erreur à réguler, une jacobienne liant l'espace de la tâche à l'espace articulaire du robot et un comportement de référence. Comme une tâche ne définit une commande que sur une partie du robot, il est possible de réaliser plusieurs tâches simultanément. En les organisant selon une hiérarchie, il est possible d'empêcher que les tâches de plus faible priorité ne modifient la commande donnée par les tâches de plus forte priorité. Pour cela, chaque tâche est réalisé dans l'espace nul laissé par les tâches de plus forte priorité. Cette méthode permet de définir une commande continue, sauf à l'approche des configurations singulières, où la commande peut atteindre des valeurs excessives à cause de l'inversion de la jacobienne. La solution classique consiste à introduire un coefficient d'amortissement dans les équations afin de limiter ces valeurs.

Ainsi, pour une hiérarchie de tâches fixée, la commande est continue. La réalisation d'une séquence de tâches nécessite de réaliser des opérations discrètes sur la pile, telles que l'insertion, le retrait ou l'échange de priorité entre deux tâches. Ces événements sont susceptibles de créer des discontinuités dans la commande. Trois approches sont proposées pour lisser ces discontinuités. La première consiste à formaliser toutes les opérations comme des suites d'échanges de priorité entre tâches voisines. La seconde, à l'inverse, consiste à tout formaliser comme des insertions ou retraits de tâches à n'importe quel niveau dans la pile. Enfin, la dernière méthode consiste à considérer la pile de tâches comme un tout plutôt que par niveaux.

Lissage du contrôle durant l'échange de deux tâches

On considère que l'échange de tâches n'est réalisé qu'entre tâches voisines. Pendant l'échange, le comportement des autres tâches n'est pas perturbé : les tâches de plus forte priorité ne sont pas affectées (par définition), et l'espace nul des tâches de plus faible priorité reste constant. Si les deux tâches sont compatibles dans l'espace nul, alors la hiérarchie entre les tâches n'a aucune influence et l'échange peut être instantané. A l'inverse, s'il y a conflit, alors la commande va présenter une discontinuité à l'instant de l'échange : une phase de lissage est donc nécessaire.

Une première méthode consiste à considérer la pile de tâche comme une limite de la pondération entre les tâches. Les deux tâches sont placées au même niveau et la priorité entre les deux tâches et déterminée par leurs poids respectifs. Lorsque la contribution d'une tâche est négligeable par rapport à l'autre, le système est équivalent à une hiérarchie classique entre les tâches. Cette méthode permet de réaliser un échange continu, mais uniquement si le facteur d'amortissement n'est pas pris en compte. En effet, l'introduction d'un facteur d'amortissement dans les équations introduit une discontinuité aux limites, i.e. lors des transitions avec la pile de tâches classique. Cette approche n'est de ce fait pas continue au voisinage des singularités.

Une autre méthode consiste à calculer deux fois la commande (pour le cas où la tâche 1 est prioritaire sur la tâche 2 et inversement) puis à évaluer la commande finale en réalisant une interpolation linéaire de ces deux commandes. Elle présente deux défauts. D'une part, la commande calculée ne peut pas être écrite sous la forme d'un problème d'optimisation, et de ce fait il n'est pas garanti que le mouvement soit faisable. D'autre part, la commande doit être calculée deux fois (mais uniquement pour ces deux niveaux et ce, quel que soit le nombre de tâches dans la pile). Malgré ces défauts, cette méthode est la seule qui assure une commande continue.

La hiérarchie de tâches est usuellement terminée par une tâche définissant le comportement par défaut pour l'ensemble des articulations. Toute tâche de plus faible priorité est donc réalisée dans un espace nul, et n'est pas donc pas prise en compte. Ainsi, il est possible de considérer l'insertion et le retrait d'une tâche comme un échange de priorité avec cette dernière tâche.

La modélisation des événements discrets par des échanges de priorité entre tâches modifie le schéma de la pile de tâches : afin d'assurer la continuité de la commande, il est nécessaire de décomposer toute action discrète en une succession d'échanges de priorité entre tâches voisines. Cette méthode assure donc la continuité de la commande pour une séquence de tâches donnée, mais introduit dans le même temps des latences lors de la réalisation d'événements discrets, qui peuvent être pénalisantes. De plus, elle présente des discontinuités par rapport à l'ordre des événements : en effet, une modification dans l'ordre des événements peut nécessiter l'ajout de phases de lissage supplémentaires pour assurer la continuité de la commande par rapport au temps. De ce fait la commande associée à cette nouvelle séquence de tâches peut présenter des différences par rapport à celle de la séquence de départ.

Insertion/retrait de tâches à n'importe quel niveau de la pile

Cette seconde méthode consiste à considérer l'insertion et le retrait de tâche comme événements de base, afin de limiter le temps nécessaire pour placer une tâche à priorité donné. Le problème des discontinuités dues au changement d'espace nul est résolu en empêchant les tâches de faible priorité d'utiliser les articulations déjà utilisées par les tâches de forte priorité. Pour cela, on utilise une pseudo-inverse pondérée à la place d'une pseudo inverse amortie, ce qui permet de contrôler l'influence de chacune des articulations, à condition qu'au moins une articulation soit toujours active.

Le mécanisme d'insertion et de retrait est le suivant : lorsqu'une opération sur une tâche est réalisée, toute tâche de priorité moindre ne peut utiliser que les articulations qui ne sont utilisées par aucune des tâches de plus forte priorité. Si toutes les articulations peuvent être utilisées, alors la tâche est réalisée normalement. Si seulement une partie de ces articulations est disponible, alors la tâche n'est réalisée qu'avec celles-là. Enfin, si aucune articulation n'est disponible, la tâche est temporairement désactivée.

Cette méthode ne peut pas être appliquée sur des systèmes complexes pour plusieurs raisons : d'une part, priver une tâche revient à limiter son espace de travail, ce qui augmente les risques de singularité et peut aussi augmenter la contribution des articulations libres, et d'autre part, la désactivation, même partielle, de certaines tâches peut ne pas être souhaitable.

Calcul de la commande avec un unique problème de minimisation

Calculer la commande à l'aide d'un seul problème de minimisation présente l'avantage de faciliter la réalisation d'événements discrets, mais a pour désavantage une relâche au niveau de la hiérarchie entre les tâches. Deux méthodes sont étudiées :

Une première méthode consiste à définir la priorité de chacune des tâches par son poids, pour l'ensemble des tâches (et non seulement deux tâches comme précédemment). L'inconvénient de cette méthode est numérique et concerne le choix du poids des tâches. S'il est trop petit, alors les tâches peuvent ne plus être prises en compte, ce qui crée des discontinuités dans la commande. A l'inverse, s'il est trop grand, alors la hiérarchie entre les tâches n'est plus respectée.

Une seconde méthode consiste à formuler le problème comme la résolution d'une tâche unique. Cette méthode permettrait d'ajouter plus facilement des contraintes sur la commande, mais la détermination de cette tâche unique est difficile : soit elle relâche la hiérarchie entre les tâches, soit elle retarde l'adaptation des tâches de plus faible priorité aux modifications liées aux tâches de plus forte priorité.

Conclusion

Ce chapitre traite du problème des discontinuités liées à la réalisation d'événements discrets. La méthode proposée consiste à formaliser tous ces événements sous la forme d'une suite d'échanges de priorités entre tâches voisines. Cette échange est réalisé grâce à une interpolation linéaire entre les deux commandes possibles, et bien qu'elle assure la continuité de la commande, elle présente aussi comme inconvénient le délai de réalisation des événements.

C.3 Superposition des tâches par optimisation

Ce chapitre définit comme paramétrer à la fois le timing et le comportement des tâches en superposant des tâches, c'est-à-dire en réalisant les tâches simultanément plutôt que séquenciellement, de façon à produire un mouvement plus fluide.

Introduction

A partir du plan symbolique, il est possible de définir une trajectoire implicite à l'aide la fonction de tâches. Il est aussi possible de définir de façon explicite la trajectoire que doit suivre le robot, mais cette méthode rend difficile l'adaptation de la séquence durant son exécution. Afin de pouvoir superposer les tâches, il est nécessaire de modifier la séquence initiale en remplaçant certaines contraintes temporelles par d'autres contraintes liées à la réalisation d'une tâche ou à l'environnement. Ces contraintes ne peuvent pas être mises sous forme symbolique, donc cette opération ne peut pas être gérée durant la phase de planification. De plus, il est nécessaire de vérifier avant l'exécution que la séquence est réalisable. On introduit donc entre les phases de planification et d'exécution une phase d'optimisation qui réalise la superposition des tâches tout en vérifiant les contraintes.

Séquence de tâches

Une séquence de tâches est un ensemble fini de tâches classées par ordre de réalisation et liées les unes aux autres. On considère qu'une tâche n'apparait qu'une seule fois dans la séquence, et on lui associe trois variables temporelles : un temps d'insertion et un temps de retrait de la pile, tout deux contrôlables, ainsi que le temps de régulation de la tâche, au bout duquel la norme de l'erreur de la tâche est inférieure à une valeur donnée. Ce troisième temps n'est pas contrôlé, car c'est une conséquence de la réalisation de la séquence de tâches. Ces temps permettent d'évaluer les contraintes temporelles définies par la séquence de tâches et les contraintes de régulation : chaque tâche doit être régulée avant d'être retirée de la pile.

Optimisation d'une séquence de tâches

Le but de la phase d'optimisation est d'améliorer le plan de tâches en tirant profit de la redondance du robot afin de réaliser un mouvement fluide. Comme il est impossible d'évaluer numériquement la superposition des tâches, on minimise à la place le temps de fin de la séquence, ce qui permet de réduire les temps morts et de forcer la superposition des tâches.

Les paramètres de l'optimisation sont les tâches, qui sont des données symboliques. Afin de pouvoir réaliser une optimisation numérique, il est nécessaire de les convertir en proto-symboles, en leur associant des données numériques. Ainsi, les paramètres de l'optimisation sont les paramètres temporels (temps d'entrée et de sortie de la pile) et le comportement de référence (défini par un gain adaptatif) de chaque tâche.

Les contraintes du problème regroupent les contraintes temporelles liées à la séquence (ordre entre les tâches et régulation des tâches) et les contraintes liées à l'environnement (limites physiques du robot, évitement des collisions. . .). Une partie seulement de ces contraintes peuvent être directement calculées à partir du vecteur de paramètres, les autres ne peuvent être déterminées que par une simulation de la séquence de tâches.

Gestion des contraintes semi-infinies

Le problème à optimiser est un problème semi-infini, c'est-à-dire qu'il a un nombre fini de paramètres, mais un nombre infini de contraintes. Ce caractère semi-infini est dû au temps, dans la mesure où les contraintes doivent être vérifiées à tout moment de la simulation.

Les méthodes classiques de gestion des contraintes semi-infinies, comme l'analyse par intervalles ou l'utilisation d'une séquence de problèmes d'optimisation, ne peuvent pas être appliquées à notre problème. La méthode que nous utilisons est basée sur une l'évaluation discrète des contraintes : les contraintes sont évaluées à intervalles de temps fixe, mais aussi lors de la réalisation d'événement discret. Afin d'avoir une évolution continue des contraintes et un nombre constant de contraintes durant l'optimisation, on considère que la valeur d'une contrainte semi-infinie correspond à la valeur maximale obtenue lors de l'exécution de la séquence. De plus, de façon à plus facilement relier la violation d'une contrainte à la tâche responsable et à améliorier la convergence de l'algorithme, un jeu de contraintes semi-infinie est associé à chacune des tâches, et n'est évalué que lorsque cette tâche est active.

Optimisation via simulation

Le problème à optimiser est continu et sa dérivée est continue par morceaux. Le solveur utilisé est CFSQP, qui, bien que prévu pour résoudre des problèmes de classe $\mathcal{C}^1$, retourne pour notre problème de meilleurs résultats que SolvOpt, un solveur qui résout des problèmes ni linéaires ou continus. La plupart des contraintes ne peuvent pas être calculées directement de façon analytique : la seule façon de les évaluer est de réaliser une simulation de la séquence de tâches. Il y a de fait un dialogue permanent entre le solveur qui choisit le vecteur de paramètres et le simulateur qui évalue les contraintes correspondantes. Deux méthodes sont proposées pour améliorer le processus d'optimisation : une mise à l'échelle des contraintes et la décomposition de la séquence en plus petites séquences indépendantes.

Discussion

La formulation de ce problème d'optimisation présente deux problèmes liés à la continuité de la séquence de tâche. Le premier problème est l'évaluation *a posteriori* de la contrainte de régulation : l'insertion et le retrait des tâches n'est déterminé que par le timing, ce qui peut conduire à des incohérences dans la séquence de tâches si les tâches ne sont pas régulées avant d'être retirées et donner des valeurs de contraintes incohérantes. Le second problème est lié à la discontinuité de la commande par rapport à l'ordre des événements. Une solution sûre mais trop restrictive serait de fixer l'ordre des événements (c'est-à-dire de travailler avec un plan de tâches totalement ordonné). De ce fait, il n'est pas possible de déterminer quelle séquence de tâche a le petit temps de réalisation avec cet algorithme.

C.4 Simulations et expérimentations

Implémentation de la cinématique inverse

Le calcul de la commande pour une hiérarchie de tâches donnée est réalisé par un framework qui se base sur un mécanisme d'entités et de signaux similaire àu mécanisme de simulink. Chaque entité est définie par un ensemble de dépendances (signaux d'entrées) et de méthodes permettant de calculer les signaux de sortie. Grâce à cette structure de graphe orientée, il est possible de limiter les calculs réalisés au strict minimum. La commande continue basée sur l'échange de priorité entre tâches voisines est implémentée et testée dans ce framework, et le problème du délai additionnel est illustré. En pratique, ce délai peut causer un manque de réactivité de la part du robot, auquel un utilisateur pourra préférer une discontinuité ponctuelle.

AMELIF, outil pour la simulation dynamique avec rendu haptique

Le logiciel AMELIF réalise la simulation dynamique d'avatars articulés virtuels avec retour haptique, et a été conçu de façon à permettre un prototypage rapide. Il est composé d'un cœur qui propose un ensemble d'outils génériques (multi-threading, outils mathématiques) autour duquel s'articule différentes librairies qui regroupent les algorithmes par thème : définition de l'environnement et des robots, calcul des collisions, de la dynamique, de la commande et affichage de l'environnement. A partir de ces librairies, il est possible de définir et de simuler dynamiquement le mouvement d'un robot dans un environnement virtuel. Un scénario illustratif est proposé, dans lesquels d'autres librairies plus spécifiques ont été utilisées : une librairie de gestion de l'haptique et une liée à la génération de posture.

Optimisation d'une séquence de tâches

Cette partie détaille les différents tests d'optimisation de séquence de tâches réalisés. Dans un premier temps, l'influence du lissage sur le processus d'optimisation est testé. Ces expériences montrent que le lissage n'a d'influence que lorsque les tâches sont couplées : le processus d'optimisation est pénalisé par les discontinuités par rapport à l'ordre des événements. Au delà de l'influence du lissage, ces expérience montrent l'importance de la phase de séquencement : laisser la détermination de l'ordre des événements à la phase d'ordonnancement (c'est-à-dire à la phase d'optimisation) pénalise le processus d'optimisation.

L'algorithme d'optimisation est ensuite testé sur un scénario existant dans laquelle le robot HRP-2 saisit une boisson à l'intérieur d'un frigo. Afin de permettre la superposition des tâches, certaines contraintes temporelles sont remplacées par des contraintes d'évitement de collisions. La séquence obtenue est plus rapide que la séquence de tâche initiale, et est jouée sur le robot HRP-2 réel.

C.5 Contrôle adaptatif

Le processus d'optimisation qui a été défini permet d'améliorer une séquence dans laquelle le robot est le seul agent. En pratique, il est probable que le robot ait à adapter son comportement aux imprévus de l'exécution. L'avantage de la fonction de tâche est d'être flexible en travaillant directement sur les informations données par les capteurs, ce qui permet par exemple de corriger l'objectif de la tâche en ligne. Dans ce chapitre, l'adaptation recherchée concerne la manière de réaliser la tâche (le “comment”) plutôt que le but (le “quoi”) : le mouvement du robot est personnalisé en utilisant les timings et les gains adéquats.

Contrôle adaptatif et formalisme de tâche

Distinguer le but de la tâche de la façon dont elle est réalisée permet de définir différentes façons de réaliser une même tâche. Cette distinction présente l'avantage d'agrandir le spectre des actions réalisables par un robot à peu de frais, mais est aussi nécessaire, car elle facilite l'adaptation du comportement du robot aux préférences et aux attentes de l'utilisateur humain.

De la même façon qu'une séquence peut être optimisée en déterminant le jeu de paramètres optimal, nous pensons que la personnalisation d'une séquence de tâches peut aussi se faire au niveau de la tâche. Nous illustrons cette idée par la personnalisation d'une tâche collaborative entre un humain et un robot.

Expérience préliminaire

Dans cette expérience, une tâche collaborative est réalisée entre deux acteurs, un client et un portier (rôle du robot). Le client doit saisir un objet placé dans une armoire située à distance et le portier doit l'aider en ouvrant cette armoire à temps, c'est-à-dire de façon à ce que le client n'ait pas à modifier son allure et puisse prendre l'objet au vol. Cette adaptation se fait sans communication directe entre les deux acteurs. Le but de l'expérience est de montrer qu'il est possible de définir un mouvement adaptatif à l'aide du formalisme de tâche.

Dans un premier temps, cette expérience est réalisée entre deux sujets humains, afin de définir les caractéristiques du mouvement à réaliser par le robot. Pour l'ensemble des participants, le mouvement réalisé par le portier est continu (il ne fait pas de pauses entre les étapes). Deux comportements ont été observés. La plupart des participants a adapté son mouvement en fonction de la position et de la vitesse du client : lorsque le client approche à vitesse réduite ou normale, le portier réalise son mouvement "par défaut" (c'est-à-dire à une vitesse fixée de son choix), et ne modifie que son timing. Par contre, si le client est trop rapide, alors le portier est obligé d'accélérer aussi son mouvement pour être dans les temps. L'autre comportement observé consistait à ne pas adapter la vitesse de réalisation de la tâche mais à démarrer le mouvement au plus tôt (en même temps que le client) de façon à être dans les temps.

Expérience sur le robot HRP-2

Le mouvement par défaut est obtenu par optimisation de la séquence de tâches, et permet de connaitre le timing théorique de chacune des tâches ainsi que la durée totale de la séquence de tâche. Les deux comportements observés chez un portier humain ont été implémentés et testés : l'adaptation par rapport au temps consiste simplement à démarrer en même temps que l'humain et à réaliser le mouvement par défaut, tandis que l'adaptation par rapport au mouvement du client nécessite de modifier le timing et la vitesse de réalisation des tâches en fonction de la vitesse du client. Le timing des tâches est adapté en conséquence, et dans le second cas, on introduit une dépendance entre la fonction de gain et la vitesse moyenne du client sur les pas précédents.

En simulation, on observe que l'adaptation minimale est moins performante que l'adaptation par rapport au mouvement : bien que la conclusion globale (succès ou échec de la simulation) soit la même dans les deux cas, le timing est meilleur dans le second cas, dans la mesure où le retard du robot par rapport à l'humain est moindre.

Conclusion

Dans cette thèse, nous proposons de réintroduire la phase d'ordonnancement entre les phases de planification et d'exécution, comblant ainsi l'écart entre les données symboliques (planification) et les données numériques (exécution). Afin de réduire cet écart, on utilise la fonction de tâche comme fil conducteur durant les trois phases de planification, ordonnancement et exécution.

Contributions

La première contribution est la définition d'une méthode permettant d'assurer la continuité de la pile de tâches lors de la réalisation d'événements discrets. Cette méthode consiste à formaliser tous les événements discrets comme des suites d'échanges de priorité entre tâches voisines. Le principal défaut de cette méthode est le délai lié au processus de lissage.

La seconde contribution est la formulation d'un problème d'optimisation permettant de réaliser la superposition des tâches. A partir d'un plan de tâches donné par la planification, le processus d'optimisation règle le timing et la vitesse de réalisation de chaque tâche de façon à améliorer le plan de tâches initial tout en respectant les contraintes physiques (liées à l'environnement) et temporelles (liées à la séquence de tâche). Ce processus d'optimisation est testé sur une séquence de tâches réalisées par la HRP-2. Il présente toutefois deux faiblesses : l'estimation *a posteriori* des contraintes et les discontinuités par rapport aux événements discrets.

Enfin, nous avons proposé d'utiliser la fonction de tâche pour personnaliser une séquence de tâches, c'est-à-dire l'adapter aux contraintes de l'environnement, typiquement les préférences d'un utilisateur humain. Un exemple a été réalisé sur le robot HRP-2

Perspectives

Ce travail peut être prolongé de plusieurs façons :

- A court terme, il est possible d'étendre le problème d'optimisation en considérant d'autres types de tâches (tâches de perception, points de passage...).
- A moyen terme, il serait nécessaire de reformuler partiellement ou totalement le problème d'optimisation, de façon à améliorer les temps de calculs, mais aussi à gérer des tâches plus complexes, telles que la marche du robot.
- Enfin, à long terme, deux prolongations sont envisageables : en définissant en amont une méthode de séquencement des tâches qui précéderait la méthode d'ordonnancement, et en tirant parti de l'environnement, de façon similaire à l'approche basée directement sur la commande.

Bibliography

[1] R. Alami, A. Clodic, V. Montreuil, E. A. Sisbot, and R. Chatila. Task planning for human-robot interaction. In *sOc-EUSAI '05: Proceedings of the 2005 joint conference on Smart objects and ambient intelligence*, pages 81–85, New York, NY, USA, 2005. ACM.

[2] K. R. Baker. *Introduction to sequencing and scheduling*. John Wiley & Sons, New-York, 1974.

[3] K. R. Baker and D. Trietsch. *Principles of Sequencing and Scheduling*. Wiley Publishing, 2009.

[4] M. Beetz, T. Arbuckle, T. Belker, A. B. Cremers, D. Schulz, M. Bennewitz, W. Burgard, D. Hähnel, D. Fox, and H. Grosskreutz. Integrated plan-based control of autonomous robots in human environments. *IEEE Intelligent Systems*, 16(5):56–65, 2001.

[5] C. Belta, A. Bicchi, M. Egerstedt, E. Frazzoli, and E. Klavins. Symboling planning and control of robot motion. *IEEE Robotics & Automation Magazine*, 14:61–70, March 2007.

[6] A. Ben-Israel and T. Greville. *Generalized inverses: theory and applications*. CMS Books in Mathematics. Springer, 2nd edition, 2003.

[7] M. Benallegue, A. Escande, S. Miossec, and A. Kheddar. Fast C1 proximity queries using support mapping of sphere-torus-patches bounding volumes. In *ICRA'09: Proceedings of the 2009 IEEE international conference on Robotics and Automation*, pages 3429–3434, Piscataway, NJ, USA, 2009. IEEE Press.

[8] D. Berenson, R. Diankov, K. Nishiwaki, S. Kagami, and J. Kuffner. Grasp planning in complex scenes. In *IEEE-RAS International Conference on Humanoid Robots (Humanoids07)*, December 2007.

[9] K. Bouyarmane and A. Kheddar. Static multi-contact inverse prolem for multiple humanoid robots and manipulated objects. In *IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, Decembre 2010.

[10] N. Brauner, G. Finke, V. Lehoux-Lebacque, C. Potts, and J. Whitehead. Scheduling of coupled tasks and one-machine no-wait robotic cells. *Computers and Operations Research*, 36(2):301–307, 2009.

[11] R.W. Brockett. On the computer control of movement. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, 1988.

[12] J.-R. Chardonnet, F. Keith, A. Kheddar, K. Yokoi, and F. Pierrot. Interactive dynamic simulator for humanoid with haptic feedback. *ROMANSY*, 2008.

[13] J.-R. Chardonnet, S. Miossec, A. Kheddar, H. Arisumi, H. Hirukawa, F. Pierrot, and K. Yokoi. Dynamic simulator for humanoids using constraint-based method with static friction. In *IEEE International Conference on Robotics and Biomimetics (ROBIO '06)*, pages 1366 – 1371, December 2006.

[14] S. Chiaverini. Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators. *IEEE Trans. on Robotics and Automation (ITRA)*, 13(3):398–410, June 1997.

[15] T. Coleman, M. A. Branch, and A. Grace. *Optimization Toolbox For Use with MATLAB User's Guide Version 2*. The MathWorks, Inc, `http://www.mathworks.com/access/helpdesk_r13/help/pdf_doc/optim/optim_tb.pdf`, Septembre 2003.

[16] C. Collette, A. Micaelli, C. Andriot, and P. Lemerle. Dynamic balance control of humanoids for multiple grasps and non coplanar frictional contacts. *Humanoids*, 2007.

[17] J. B. Dabney and T. L. Harman. *Mastering SIMULINK*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1997.

[18] K. Dautenhahn, M. Walters, S. Woods, K. L. Koay, C. L. Nehaniv, E. A. Sisbot, R. Alami, and T. Siméon. How may I serve you? A robot companion approaching a seated person in a helping context. In *Proceeding of the 1st ACM SIGCHI/SIGART conference on Humanrobot interaction (HRI)*, pages 172–179. ACM Press, 2006.

[19] R. Dechter, I. Meiri, and J. Pearl. Temporal constraint networks. *Artificial Intelligence*, 49(1-3):61–95, 1991.

[20] W. Decré, R. Smits, H. Bruyninckx, and J. De Schutter. Extending iTaSC to support inequality constraints and non-instantaneous task specification. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1875–1882, Piscataway, USA, May 2009.

[21] A. Deo and I. Walker. Robot subtask performance with singularity robustness using optimal damped least squares. In *IEEE Int. Conf. on Robotics and Automation (ICRA'92)*, pages 434–441, Nice, France, May 1992.

[22] K. Doty, C. Melchiorri, and C. Bonivento. A theory of generalized inverses applied to robotics. *Int. J. Robotics Research*, 12(1):1–19, December 1993.

[23] C. Dune, A. Herdt, O. Stasse, P.-B. Wieber, K. Yokoi, and E. Yoshida. Visual servoing of dynamic walking motion by ignoring the sway motion. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, October 2010.

[24] A. Escande and A. Kheddar. Contact planning for acyclic motion with task constraints and experiment on hrp-2 humanoid. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 416–417, Piscataway, NJ, USA, 2009. IEEE Press.

[25] A. Escande, N. Mansard, and P-B. Wieber. Fast resolution of hierarchized inverse kinematics with inequality constraints. In *IEEE Int. Conf. on Robotics and Automation (ICRA'10)*, Anchorage, USA, Mai 2010.

[26] P. Evrard. *Contrôle d'humanoïdes pour réaliser des tâches haptiques en coopération avec un opérateur humain (thesis in english)*. PhD thesis, Université Montpellier II, December 2009.

[27] P. Evrard, F. Keith, J.-R. Chardonnet, and A. Kheddar. Framework for haptic interaction with virtual avatars. In *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2008.

[28] P. Evrard and A. Kheddar. Homotopy-based controller for physical human-robot interaction. In *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, volume 18, pages 1–6, October 2009.

[29] R. Featherstone. Robot dynamics: Equations and algorithms. In *IEEE Int. Conf. Robotics and Automation*, pages 826–834, 2000.

[30] R. E. Fikes, P. E. Hart, and N. J. Nilsson. *Learning and executing generalized robot plans*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[31] M. Gallien, F. Ingrand, and S. Lemai-Chenevier. Robot actions planning and execution control for autonomous exploration rovers. In *14th International Conference on Automated Planning & Scheduling Workshop*, 2005.

[32] M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kauffmann Publishers, 2004.

[33] G. Golub and C. Van Loan. *Matrix computations*. John Hopkins University Press, 1996.

[34] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 5:287 – 326, 1979.

[35] J. Guitton and J.-L. Fargues. Taking into account geometric constraints for task-oriented motion planning. *Workshop on Bridging the Gap Between Task and Motion Planning of International Conference on Automated Planning and Scheduling (ICAPS)*, 19:26–33, 2009.

[36] R. Hettich. An implementation of a discretization method for semi-infinite programming. *Math. Program.*, 34(3):354–361, April 1986.

[37] A. Hofmann, M. Popovic, and H. Herr. Exploiting angular momentum to enhance bipedal center-of-mass control. In *IEEE Int. Conf. on Robotics and Automation (ICRA'09)*, 2009.

[38] T. Inamura, H. Tanie, and Y. Nakamura. Proto-symbol development and manipulation in the geometry of stochastic model for motion generation and recognition. *Proceedings of the Annual Conference of the Institute of Systems, Control and Information Engineers*, 48:53–56, September 2004.

[39] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa. Biped walking pattern generation by using preview control of zero-moment point. In *IEEE Int. Conf. on Robotics and Automation (ICRA'03)*, pages 1620–1626, Taipei, Taiwan, September 2003.

[40] F. Kanehiro, W. Suleiman, K. Miura, M. Morisawa, and E. Yoshida. Feasible pattern generation method for humanoid robots. In *IEEE-RAS Int. Conf. on Humanoid Robots (Humanoids)*, volume 9, pages 542–548, December 2009.

[41] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi, and T. Isozumi. Humanoid robot HRP-2. In 2, editor, *In IEEE/RSJ Int. Conf. on Robots and Intelligent Systems*, pages 1083–1090, 2004.

[42] O. Kanoun, F. Lamiraux, P.-B. Wieber, F. Kanehiro, E. Yoshida, and J.-P. Laumond. Prioritizing linear equality and inequality systems: application to local motion planning for redundant robots. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 724–729, Kobe, Japan, May 2009. IEEE Press.

[43] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *Int. Journal of Robotics Research*, 5(1):90–98, Spring 1986.

[44] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *International Journal of Robotics Research*, 3(1):43–53, February 1987.

[45] O. Khatib, L. Sentis, and J. Park. A unified framework for whole-body humanoid robot control with multiple constraints and contacts. In *European Robotics Symposium*, pages 303–312, 2008.

[46] H. Kress-Gazit, G. E. Fainekos, and G. J. Pappas. Temporal logic-based reactive mission and motion planning. *IEEE Trans. on Robotics (ITRO)*, 2009.

[47] A. Kuntsevich and F. Kappel. *Solvopt: The Solver for Local Nonlinear Optimization Problems*. Institute for Mathematics Karl-Franzens University of Graz, `http://www.kfunigraz.ac.at/imawww/kuntsevich/solvopt/ps/manual.pdf`, June 1997.

[48] P. Langley and D. Choi. A unified cognitive architecture for physical agents. In *AAAI'06: proceedings of the 21st national conference on Artificial intelligence*, pages 1469–1474. AAAI Press, 2006.

[49] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. Technical report, Department of Computer Science University of North Carolina, 1999.

[50] J.-P. Laumond. Kineo cam: a success story of motion planning algorithms. *Robotics & Automation Magazine, IEEE*, 13(2):90–93, 2006.

[51] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98–11, Dept. of Computer Science, Iowa State University, 1998.

[52] C. Lawrence, J. L. Zhou, and A. L. Tits. User's guide for CFSQP version 2.5: A C code for solving (large scale) constrained nonlinear (minimax) optimization problems, generating iterates satisfying all inequality constraints. Technical report, Institute for Systems Research TR-94-16r1, 1997.

[53] S. Lemai-Chenevier. *IXTET-EXEC: planning, plan repair and execution control with time and resource management*. PhD thesis, Institut National Polytechnique de Toulouse, June 2004.

[54] S. Lemai-Chenevier and F. Ingrand. Interleaving temporal planning and execution in robotics domains. In *AAAI'04: Proceedings of the 19th national conference on Artifical intelligence*, pages 617–622. AAAI Press, 2004.

[55] S. Lengagne, P. Mathieu, A. Kheddar, and E. Yoshida. Generation of dynamic motions under continuous constraints: Efficient computation using B-splines and taylor polynomials. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[56] S. Lengagne, N. Ramdani, and P. Fraisse. A new method for generating safe motions for humanoid robots. In *IEEE-RAS Internationnal conference on Humanoid robots*, pages 105–110, 2008.

[57] H. X. Li and B. C. Williams. Generative planning for hybrid systems based on flow tubes. *International Conference on Automated Planning and Scheduling (ICAPS)*, 18:206–213, September 2008.

[58] A. Liégeois. Automatic supervisory control of the configuration and behavior of multi-body mechanisms. *IEEE Trans. on Systems, Man and Cybernetics*, 7(12):868–871, December 1977.

[59] P. Lopez, M.-L. Levy, and B. Pradin. Characterisation by decomposition in scheduling. *Comput. Ind.*, 36(1-2):113–116, 1998.

[60] C. Mancel and P. Lopez. Complex optimization problems in space systems. In *Int. Conf. on Automated Planning and Scheduling (ICAPS)*, volume 13, Italy, March (26) 2003.

[61] N. Mansard and F. Chaumette. Task sequencing for sensor-based control. *IEEE Trans. on Robotics*, 23(1):60–72, February 2007.

[62] N. Mansard and O. Khatib. Continuous control law from unilateral constraints. In *IEEE Int. Conf. on Robotics and Automation, ICRA'08*, pages 3359–3364, Las Passadenas, USA, May 2008.

[63] N. Mansard, O. Stasse, F. Chaumette, and K. Yokoi. Visually-guided grasping while walking on a humanoid robot. In *IEEE Int. Conf. on Robotics and Automation (ICRA'07)*, pages 3041–3047, Roma, Italia, April 2007.

[64] N. Mansard, O. Stasse, P. Evrard, and A. Kheddar. A versatile generalized inverted kinematics implementation for collaborative working humanoid robots: The stack of tasks. *International Conference on Advanced Robotics (ICAR)*, 14(119):1–6, June 2009.

[65] P. Martin and M. Egerstedt. Motion description language-based topological maps for robot navigation. *Communications in Information and Systems*, 8(2):171–184, 2008.

[66] S. Miossec, K. Yokoi, and A. Kheddar. Development of a software for motion optimization of robots - application to the kick motion of the HRP-2 robot. In *IEEE International Conference on Robotics and Biomimetics*, December 2006.

[67] R. E. Moore. *Interval Analysis*. Prentice-Hall, 1966.

[68] Karen L. Myers. CPEF : A continuous planning and execution framework. *AI Magazine*, 20(4):63–69, 1999.

[69] Y. Nakamura and H. Hanafusa. Inverse kinematics solutions with singularity robustness for robot manipulator control. *Trans. ASME Journal of Dynamic System, Measures and Control*, 108:163–171, September 1986.

[70] S. Nakaoka, A. Nakazawa, F. Kanehiro, K. Kaneko, M. Morisawa, and K. Ikeuchi. Task model of lower body motion for a biped humanoid robot to imitate human dances. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2005.

[71] J. Park. *Control Strategies for Robots in Contact*. PhD thesis, Stanford University, California, USA, March 2006.

[72] Sang Il Park, Hyun Joon Shin, Tae Hoon Kim, and Sung Yong Shin. On-line motion blending for real-time locomotion generation. *Comput. Animat. Virtual Worlds*, 15(3-4):125–138, 2004.

[73] R. Philippsen, N. Nejati, and L. Sentis. Bridging the gap between semantic planning and continuous control for mobile manipulation using a graph-based world representation. *1st InternationalWorkshop on Hybrid Control of Autonomous Systems (Hycas)*, July 2009.

[74] R. Reemtsen. Semi-Infinite Programming: Discretization methods, SIP. 1998.

[75] R. Reemtsen and J.-J. RÃijckmann. *Semi-Infinite Programming*. Kluwer Academic, 1998.

[76] C. Rose, M. F. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications*, 18:32–40, 1998.

[77] L. Saab, N. Mansard, F. Keith, J-Y. Fourquet, and P. Soueres. Generation of dynamic motion for anthropomorphic systems under prioritized equality and inequality constraints. In *Submitted to the IEEE Int. Conf. on Robotics and Automation (ICRA'11)*, Shangai, China, Mai 2011.

[78] J. Salini, S. Barthélemy, and P. Bidaud. LQP-based controller design for humanoid whole-body motion. *Advances in Robot Kinematics*, 3:177–184, 2010.

[79] C. Samson, M. Le Borgne, and B. Espiau. *Robot Control: the Task Function Approach*. Clarendon Press, Oxford, United Kingdom, 1991.

[80] M. Scheutz, P. Schermerhorn, and J. Kramer. The utility of affect expression in natural language interactions in joint human-robot tasks. In *HRI '06: Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 226–233, New York, NY, USA, 2006. ACM.

[81] L. Sentis and O. Khatib. Control of free-floating humanoid robots through task prioritization. In *IEEE Int. Conf. on Robotics and Automation (ICRA'05)*, pages 1718–1723, Barcelona, Spain, April 2005.

[82] R. D. Shapiro. Scheduling coupled tasks. *Naval Research Logistics Quarterly*, 27:489âĂŞ–498, 1980.

[83] B. Siciliano and J-J. Slotine. A general framework for managing multiple tasks in highly redundant robotic systems. In *IEEE Int. Conf. on Advanced Robotics (ICAR)*, volume 2, pages 1211 – 1216, Pisa, Italy, June 1991.

[84] G. Simonin, R. Giroudeau, and J.-C. König. 2-cover definition for a coupled-tasks scheduling problem. *International Conference on Theoretical and Mathematical Foundations of Computer Science : Extended matching problem for a coupled-tasks scheduling problem, ORLANDO, Floride*, june 2009.

[85] D. E. Smith, J. Frank, and A. K. Jónsson. Bridging the gap between planning and scheduling. *Knowledge Engineering Review*, 15(1):47–83, 2000.

[86] C. Van Loan. On the method of weighting for equality constrained least squares problems. Technical report, Cornell University, Ithaca, NY, USA, march 1984.

[87] R. J. Vanderbei. Loqo user's manual - version 4.05. 2000.

[88] M. Vukobratović and D. Juricić. Contribution to the synthesis of biped gait. *IEEE Trans Biomed Eng.*, 16(1):1–6, January 1969.

[89] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, March 2006.

[90] P.-B. Wieber and C. Chevallereau. Online adaptation of reference trajectories for the control of walking systems. *Robotics and Autonomous Systems*, 54(7):559–566, 2006.

[91] A. Witkin and Z. Popović. Motion warping. Computer Graphics (Proc. SIGGRAPH '95):105–108, 1995.

[92] E. Yoshida, I. Belousov, C. Esteves, and J.-P. Laumond. Humanoid motion planning for dynamic tasks. In *In International Conference on Intelligent Robotics and Systems*, pages 1–6. IEEE, 2005.

Contributions

[93] J.-R. Chardonnet, F. Keith, A. Kheddar, K. Yokoi, and F. Pierrot. Interactive dynamic simulator for humanoid with haptic feedback. *ROMANSY*, 2008.

[94] J.-R. Chardonnet, F. Keith, S. Miossec, A. Kheddar, and F. Pierrot. Simulation dynamique interactive pour corps poly-articulés. *3ème Journées Nationales de la Robotique Humanoïde (JNRH)*, 3, May (13-14) 2008.

[95] P. Evrard, F. Keith, J.-R. Chardonnet, and A. Kheddar. Framework for haptic interaction with virtual avatars. In *IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, 2008.

[96] F. Keith, P. Evrard, J.-R. Chardonnet, S. Miossec, and A. Kheddar. Haptic interaction with virtual avatars. In *Proceedings of the EuroHaptics*, pages 630–639, Madrid, June (9-13) 2008.

[97] F. Keith, N. Mansard, and A. Kheddar. Task-formalism based method: optimization and adaptation of a tasks schedule. *28th Annual Conference of the Robotics Society of Japan (RSJ)*, 28, September (22-24) 2010.

[98] F. Keith, N. Mansard, S. Miossec, and A. Kheddar. From discrete mission schedule to continuous implicit trajectory using optimal time warping. *Proceedings of International Conference on Automated Planning and Scheduling (ICAPS)*, 19:366 – 369, September 2009.

[99] F. Keith, N. Mansard, S. Miossec, and A. Kheddar. Optimisation de séquence de tâches avec lissage des mouvements. *4ème Journées Nationales de la Robotique Humanoïde (JNRH)*, 4, mai 2009.

[100] F. Keith, N. Mansard, S. Miossec, and A. Kheddar. Optimization of tasks warping and scheduling for smooth sequencing of robotic actions. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, october 2009.

[101] F. Keith, N. Mansard, S. Miossec, and A. Kheddar. Optimized time-warping tasks scheduling for smooth sequencing. *9th International Symposium on Robot Control (Syroco)*, pages 265–270, September 2009.

[102] F. Keith, P.-B. Wieber, N. Mansard, and A. Kheddar. Analysis of the discontinuities in the task priority scheme when switching priorities. In *submitted to ICRA 2011*, 2011.

[103] L. Saab, N. Mansard, F. Keith, J.-Y. Fourquet, and P. Soueres. Generation of dynamic motion for anthropomorphic systems under prioritized equality and inequality constraints. In *submitted to ICRA 2011*, 2011.

TITRE : Optimisation de séquence de tâches avec lissage du mouvement dans la réalisation de missions autonomes ou collaboratives d'un humanoïde virtuel ou robotique

RÉSUMÉ : La réalisation d'une mission robotique se décompose usuellement en trois étapes : la planification, i.e. le choix des taches à réaliser, le séquencement, i.e. la détermination du timing et de l'ordre de réalisation des tâches, et finalement l'exécution du plan de tâches. Pour les systèmes redondants tels que les robots humanoïdes, la tâche (dans le sens de fonction de tâche) détermine une commande sur une partie du robot, permettant ainsi la réalisation simultanée de plusieurs tâches à l'aide d'un formalisme de pile de tâches. Cependant, les mécanismes d'ordonnancement classiques ne gèrent pas les cas où le mouvement est déterminé par un ensemble de tâches hiérarchisé : pour ces robots, la phase d'ordonnancement est éludée et l'exécution se base directement sur la plan de tâches donné par le planificateur. Le but de cette thèse est de réintroduire cette phase d'ordonnancement, tout en maintenant le rôle central de la tâche. Dans un premier temps, la continuité de la commande fournie par la pile de tâches est étudiée. En particulier, nous mettons en évidence les discontinuités accompagnant la réalisation d'événements discrets (à savoir l'insertion, le retrait et l'échange de priorité de tâches), puis proposons et comparons plusieurs méthodes de lissage. Ensuite, nous présentons une méthode permettant d'optimiser une séquence de tâches donnée en modifiant le timing et la paramétrisation des tâches, tout en respectant les contraintes liées à l'environnement. Enfin, une nouvelle utilisation de la flexibilité de la fonction de tâche consistant à adapter une séquence de tâches aux préférences d'un utilisateur humain est illustrée. Ces résultats sont illustrés sur un robot humanoïde.

MOTS-CLEFS : Robotique, Optimisation, Ordonnancement, Superposition de tâches, Personnalisation de tâches.

TITLE : Optimization of motion overlapping for task sequencing

ABSTRACT : A general agreed approach on mission and motion planning consists in splitting it into three steps: decomposing the mission into a sequence of tasks (task planning), determining the order of realization and the timing of the tasks (task scheduling) and finally executing the task sequence. This approach maintains the task component in the entire reasoning, using it as a bridge between planning, scheduling and execution. In the sense of task function, a task defines a control law on part of the robot. Hence, for highly redundant systems such as humanoid robots, it is possible to realize several tasks simultaneously using a stack-of-tasks formalism. Though, classical schedulers do not handle the case where the motion is specified not by one, but by a combination of tasks organized into a hierarchy. As a result, the scheduling phase is usually skipped. This thesis aims at reintroducing the scheduling phase, while maintaining the central role of the task. First, the stack-of-tasks formalism is recalled and the continuity of the control law is studied. Particularly, we show that discreet operations (insertion, removal and swap of priority between tasks) create discontinuities in the control. We then present and discuss smoothing methods. Second, we present a task-overlapping based method to optimize not only the scheduling but also the behavior of the tasks of a given sequence, while accounting for the physical constraints of the execution. Finally, we introduce a new perspective in the usage of the task-function approach to personalize a task sequence and take into account user preferences. These results are experimented on the humanoid robot platform HRP-2.

KEYWORDS : Robotic, Optimization, Scheduling, Task overlapping, Task personalization.

DISCIPLINE : Génie Informatique, Automatique et Traitement du Signal

Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier
UMR 5506 CNRS/Université de Montpellier 2
161 rue Ada - 34392 Montpellier Cedex 5 - FRANCE