From Discrete Mission Schedule to Continuous Implicit Trajectory using Optimal Tasks Warping

François Keith[1,4], Nicolas Mansard[2], Sylvain Miossec[3], and Abderrahmane Kheddar[1,4]

[1]CNRS-UM2 LIRMM, Montpellier, France
[2]CNRS-LAAS, Toulouse, France
[3]PRISME-Univ. d'Orléans, Bourges, France
[4]CNRS-AIST JRL, UMI3218/CRT, Tsukuba, Japan
{*keith, kheddar*}*@lirmm.fr, nmansard@laas.fr, sylvain.miossec@bourges.univ-orleans.fr*

Abstract

This paper presents a generic solution to apply a mission described by a sequence of tasks on a robot while accounting for its physical constraints, without computing explicitly a reference trajectory. A naive solution to this problem would be to schedule the execution of the tasks sequentially, avoiding concurrency. This solution does not exploit fully the robot capabilities such as redundancy and have poor performance in terms of execution time or energy. Our contribution is to determine the time-optimal realization of the mission taking into account robotic constraints that may be as complex as collision avoidance. Our approach achieves more than a simple scheduling; its originality lies in maintaining the task approach in the formulated optimization of the task sequencing problem. This theory is exemplified through a complete experiment on the real HRP-2 robot.

Introduction

A robot is designed to perform missions in various application contexts. When the environment is well or partially structured most missions can be hierarchically decomposed into a set of tasks (i.e. generic sensory-motor functions) which has to be mapped into robot execution. Numerous works have been proposed to compute such a sequence of tasks from a given mission and a set of causal paradigms (Dechter 2003; Ghallab, Nau, and Traverso 2004). However, they generally produce a symbolic plan, where the only numerical precisions lie on the scheduled time data. Moreover, constraints have to be expressed under a symbolic expression. Its robotic application into the real world requires the time sequence to be refined, typically through an applicative path planner (LaValle 2006), that will compute the trajectories to be followed by the robot (Lamare and Ghallab 1998). Yet, the meaning of the symbolic plan is lost in the global trajectory. Such low-level methods lack of robustness to environment changes or uncertainties. Consequently, the remaining trajectory may have to be recomputed several times while the mission is being achieved. Moreover, it is difficult (and often specifically hard coded) to enhance the trajectory with symbolic data, that would help re-computing only part of the plan (Py and Ingrand 2004) or distort locally the trajectory after small environment changes (Quinlan and Khatib 1993).

Rather than using a trajectory planner between the temporal reasoning and its real robotic execution, we propose to use a sensory-motor control approach based on task components. The task function (Samson, Le Borgne, and Espiau 1991) is an elegant approach to produce intuitively sensor-based robot objectives. Based on the redundancy of the system, the approach can be extended to consider a hierarchical set of tasks (Siciliano and Slotine 1991). Hierarchy of tasks are becoming popular to build complex behavior for very redundant robot such as humanoids (Mansard and Chaumette 2007; Sentis and Khatib 2006).

A task (*i.e.* a *task function*) can be directly linked to the symbols on which the task temporal network is reasoning. Mission decomposition is thus executable directly using the sensory-motor mapping of the task function. However, exclusive task sequencing on the robot produces generally jerky suboptimal movements which may look to humans as monotonous automated motions. This paper focuses on finding a solution to produce automatically an optimal plane/schedule that makes use of the redundancy by enabling task concurrency. It seems difficult to use temporal networks to produce a scheduling with task overlapping **when the tasks concurrency is restricted by physical limitations** of the robot (for example obstacles or balance of a biped robot), since the constraints are not in a discrete form. On the other hand, semi-infinite optimization (Miossec, Yokoi, and Kheddar 2006) is known to generate low level trajectories, while accounting for such constraints, but with insufficient robustness to environment uncertainties.

In this paper, we propose to rely on task for both the symbolic reasoning and control of the robot. In between, we propose to use semi-infinite optimization to refine the symbolic schedule and account for system constraints. Given a sequence of tasks to achieve a mission, our solution returns for each task the optimal times at which it is activated and inactivated and the optimal parameters for the task execution. The originality of our approach lies in keeping the task component in the formulation of this problem, which can roughly translate to optimizing tasks overlapping by manipulating tasks, i.e. the controllers as *variables* of the optimization problem.

Generic Task Sequencing

Task function formalism and Stack of Tasks

Defining the motion of the robot in terms of task simply consists in choosing several control laws to be applied on a subpart of the robot degrees of freedom (DOF).

A task is defined by a vector $\mathbf{e}$ (typically, the error between a signal $\mathbf{s}$ and its desired value, $\mathbf{e} = \mathbf{s} - \mathbf{s}^*$). The Jacobian of the task is noted $\mathbf{J} = \frac{\partial e}{\partial q}$, where $\mathbf{q}$ is the robot configuration vector. In the following, we consider that the robot input control is the joint velocity $\dot{\mathbf{q}}$. The equation of motion is thus reduced to the kinematics:

$$\dot{\mathbf{e}} = \mathbf{J}\dot{\mathbf{q}} \tag{1}$$

Considering a reference behavior $\dot{\mathbf{e}}^*$ to be applied in the task space, the control law to be applied on the robot whole body is given by the least-square solution:

$$\dot{\mathbf{q}} = \mathbf{J}^+\dot{\mathbf{e}}^* + \mathbf{P}\mathbf{z} \tag{2}$$

where $\mathbf{J}^+$ is the least-square inverse (Ben-Israel and Greville 2003) of $\mathbf{J}$, $\mathbf{P} = \mathbf{I} - \mathbf{J}^+\mathbf{J}$ is the null-space of $\mathbf{J}$ and $\mathbf{z}$ is any secondary criterion that will be applied without disturbing the main task thanks to the projection into $\mathbf{P}$[1]. A typical requested behavior is the regulation of the error, which can be obtained through an exponential decrease by setting:

$$\dot{\mathbf{e}}^* = -\lambda\mathbf{e} \tag{3}$$

As mentioned earlier, (2) enables to compose a complex behavior from a set of tasks (Siciliano and Slotine 1991; Baerlocher and Boulic 2004; Sentis and Khatib 2006): $\mathbf{z}$ can be used to fulfil a secondary task, *without* disturbing the main task having priority. This nice decoupling can be extended recursively to a set of n tasks, each new task being fulfilled without disturbing the previous ones:

$$\dot{\mathbf{q}}_\mathbf{i} = \dot{\mathbf{q}}_{\mathbf{i-1}} + (\mathbf{J_i}\mathbf{P}^\mathbf{A}_{\mathbf{i-1}})^+(\dot{\mathbf{e}}_\mathbf{i} - \mathbf{J_i}\dot{\mathbf{q}}_{\mathbf{i-1}}),\ \ i = 1 \dots n \tag{4}$$

where $\dot{\mathbf{q}}_0 = \mathbf{0}$, $\mathbf{P}^\mathbf{A}_\mathbf{i}$ is the projector onto the null-space of the augmented Jacobian $\mathbf{J}^\mathbf{A}_\mathbf{i} = (\mathbf{J_1}, \dots \mathbf{J_i})$ and $\widetilde{\mathbf{J}}_\mathbf{i} = \mathbf{J_i}\mathbf{P}^\mathbf{A}_{\mathbf{i-1}}$ is the limited Jacobian of the task i. The robot joint velocity realizing all the tasks in the stack is $\dot{\mathbf{q}} = \dot{\mathbf{q}}_\mathbf{n}$. A similar recursive formulation can be obtained to compute the $\mathbf{P}^\mathbf{A}_\mathbf{i}$ (Baerlocher and Boulic 2004). The complete implementation of this approach is proposed in (Mansard and Chaumette 2007) under the name *Stack of Tasks* (SoT). The structure enables to easily add or remove a task, or to swap the priority order between two tasks, during the control. Constraints (such as joints limit) can be taken into account. The continuity of the control law is preserved even at the instant of change.

Gain handling

The simple attractor presented in (3) has the advantage to introduce a nice exponential decrease. However, it can be penalizing, since $\dot{\mathbf{q}}$ is directly proportional to $\mathbf{e}$ (3) . At the beginning of the task, $\|\mathbf{e}\|$ reaches its higher value (strong acceleration), while at the end of the task, $\|\mathbf{e}\|$ decreases slowly (slow convergence).

[1]Eq. (2) is the least-square solution when $\mathbf{z} = \mathbf{0}$

A very classical 'trick' when regulating a task is to rather use an adaptive gain $\lambda = \lambda(\mathbf{e}(t))$ that depends on the norm of the error of the task. To keep the nice property of the attraction, the gain only adapts with the error, and not directly with the time. We choose the following function:

$$\lambda(\mathbf{e}) = (\lambda^F - \lambda^I)\exp\left(\frac{-\|\mathbf{e}\|\beta}{\lambda^F - \lambda^I}\right) + \lambda^I \tag{5}$$

with λ^I the gain at infinity, λ^F the gain at regulation and β the slope at regulation. Besides, since the gain near convergence λ^F has to be higher than the gain at the entry λ^I, we have: $\lambda^I \leq \lambda^F$.

Sequence of tasks

A task sequence is a finite set of tasks sorted by order of regulation, and eventually linked to each other. Any pair of tasks can be either independent (i.e. they can be achieved in parallel), or constrained (i.e. one may have to wait for another one to be achieved, in order to make sense or to be doable).

The sequence can be formulated into a classical temporal network scheduling, starting at t^0 and ending at t^{End}. Both values are finite and the sequence does not loop. Besides, we may 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 SoT. For a given task i, this interval is noted $[t_i^I, t_i^F]$: the task enters in the SoT at t_i^I and is removed at t_i^F. These instants are defined with respect to the beginning of the sequence at t^0. However, they do not indicate the achievement level of the task: t_i^F may apply before the task regulation. Let's ϵ_i be the tolerance on the task regulation: a task is considered as regulated when $\|\mathbf{e_i}(t)\| \leq \epsilon_i$. The regulation time t_i^R is defined by $\|\mathbf{e_i}(t_i^R)\| = \epsilon_i$.

Two types of tasks can be distinguished: the main tasks and the optional tasks. The main tasks describe the basic trajectory of the robot and have to be regulated before being removed of the SoT. The optional tasks are aimed at locally modifying the trajectory of the robot by creating a attractor and do not need to be regulated.

A task sequence is characterized by a set of time-constraints binding the schedules of two tasks $\mathbf{e_i}$ and $\mathbf{e_j}$. They can be defined as follows[2]: $\mathbf{e_i}$ must begin or end once $\mathbf{e_j}$ has begun, has ended or has been regulated. We use the graphical representation given by Fig. 1 and the following notation to describe the sets of pairs of tasks $\mathbf{e_i}$ and $\mathbf{e_j}$ that undergo these dependencies ($\mathbf{e_i}$ is the direct predecessor of

[2]contrary to Allen Logic, that only considers the start and end points of the time interval, here is also considered the regulation time t^R

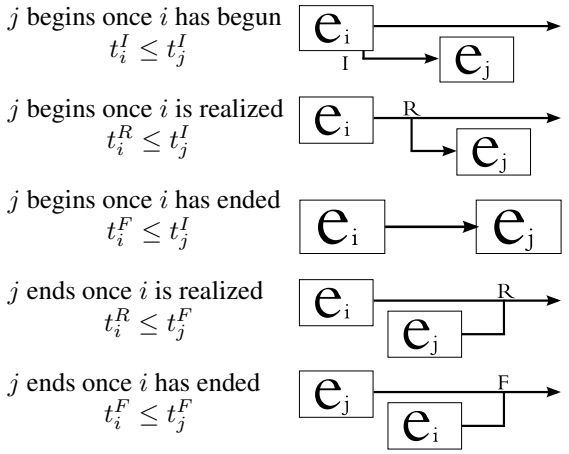

Figure 1: Five time-dependency relations are considered.

$\mathbf{e_j}$) :

$$S_{I,I} = \{(\mathbf{e_i}, \mathbf{e_j}) \mid t_i^I \leq t_j^I\} \quad (6a)$$

$$S_{R,I} = \{(\mathbf{e_i}, \mathbf{e_j}) \mid t_i^R \leq t_j^I\} \quad (6b)$$

$$S_{R,F} = \{(\mathbf{e_i}, \mathbf{e_j}) \mid t_i^R \leq t_j^F\} \quad (6c)$$

$$S_{F,I} = \{(\mathbf{e_i}, \mathbf{e_j}) \mid t_i^F \leq t_j^I\} \quad (6d)$$

$$S_{F,F} = \{(\mathbf{e_i}, \mathbf{e_j}) \mid t_i^F \leq t_j^F\} \quad (6e)$$

For example, the robot has first to grasp an object and maintain the force closure on it ($\mathbf{e_A}$) before moving it ($\mathbf{e_B}$). The task ($\mathbf{e_B}$) can only start once the task ($\mathbf{e_A}$) has been regulated, and must end before the task ($\mathbf{e_A}$).

Continuous optimization of sequence of tasks

Given a set of hypothesis described using (6), we now propose a generic solution to automatically compute an optimal set of task-behavior parameters and their sequencing plan to be executed by the SoT.

General problem formulation

An optimization problem is composed of a criterion to minimize, and of a set of equality and inequality constraints that must be satisfied. Our chosen criterion is to minimize the regulation duration of the mission.

The variables of our problem are for each task: (i) the time of its entry, (ii) the time of its removal (from the SoT), and (iii) the gains $(\lambda^I, \lambda^F, \beta)$ which describe the task execution behavior.

The general optimization problem is written as follows:

$$\min_{\mathbf{x}} t^{\text{End}} \quad (7a)$$

$$\text{subject to } \dot{\mathbf{q}} = SoT_{\mathbf{x}}(\mathbf{q}, t) \quad (7b)$$

$$\text{seq}(\mathbf{q}) < 0 \quad (7c)$$

$$\phi(\mathbf{q}) < 0 \quad (7d)$$

$$\forall i, t_i^F \leq t^{\text{End}} \quad (7e)$$

The vector $\mathbf{x}$ gathers the optimization variables of each task and t^{End}, the duration of the mission, $\mathbf{x} = [t_1^I, t_1^F, \lambda_1^I, \lambda_1^F, \beta_1, \ldots, t_n^I, t_n^F, \lambda_n^I, \lambda_n^F, \beta_n, t^{\text{End}}]$. $\text{seq}(\mathbf{q})$ and $\phi(\mathbf{q})$ are respectively the sequencing and the robotic constraints.

The optimization criterion t^{End} is computed indirectly. An equivalent explicit definition could be given by $t^{\text{End}} = \max_i(t_i^F)$. However this constraint is not smooth. Giving only (7b), the problem is smooth and properly defined: at the optimal solution, t^{End} will be equal to the maximum termination time of all tasks' t_i^F. Vector $\mathbf{q}$ is in fact a vector of functions of time, hence constraints $\phi(\mathbf{q})$ are semi-infinite, *i.e.* taking place for all the values of the continuous variable $t \in [t^0, t^{\text{End}}]$.

It can be shown that (7) defines a continuous optimization problem. However, it cannot be solved directly because of the semi-infinite nature of the constraints. Therefore we expanded the semi-infinite constraint into a discrete form.

Constraints

Parameters $\mathbf{x}$ must satisfy both the sequencing and the robotic time-constraints enumerated hereafter:

Tasks constraints, noted seq$(\mathbf{q})$ gather the task sequence conditions of (6) and the following constraints:
For each task i:

Time coherence	$0 \leq t_i^I < t_i^F \leq t^{\text{End}}$	(8a)
Termination condition	$\|s_i^* - s_i(t_i^F)\| < \epsilon_i$	(8b)
Gain consistency	$\lambda_i^I \leq \lambda_i^F$	(8c)

The constraints (6a), (6d), (6e), (8a) and (8c) are linear. On the contrary, the constraint (8b) is impossible to compute directly using $\mathbf{x}$, and is determined from a *simulation* of the execution. Care has to be taken while resolving the condition described by (6b) and (6c). Indeed, discretizing t^R to the closest simulation step will produce discontinuities which may disturb the optimization process. A rather fastidious solution to this continuity problem would be to determine this point by interpolation. Another solution is to reformulate them: we rather evaluate the regulation of the task i when the task j begins. The constraint (6b) and (6c) becomes respectively:

$$\forall (i,j) \in S_{R,I}, \|s_i^* - s_i(t_j^I)\| \leq \epsilon_i \quad (9)$$

$$\forall (i,j) \in S_{R,F}, \|s_i^* - s_i(t_j^F)\| \leq \epsilon_i \quad (10)$$

Robot constraints : $\phi(\mathbf{q})$ Those constraints are mainly due to hardware intrinsic limitations of the robot. For now, we realize only a kinematic simulation of the task sequence. Thus, dynamic constraints (such as torque limits, stability ...) are not taken into account. The remaining constraints are as follows:

Joint limits	$\mathbf{q}_{\min} \leq \mathbf{q} \leq \mathbf{q}_{\max}$	(11a)
Velocity limits	$\dot{\mathbf{q}}_{\min} \leq \dot{\mathbf{q}} \leq \dot{\mathbf{q}}_{\max}$	(11b)
Collision avoidance	$0 \leq d_{ij}$	(11c)

$\mathbf{q}_{\min}$, $\mathbf{q}_{\max}$, $\dot{\mathbf{q}}_{\min}$, $\dot{\mathbf{q}}_{\max}$ are respectively the lower and upper joint limits and the lower and upper velocity limits. d_{ij} is the distance between objects i and j. Object designate those found in the mission's environments and each link of the robotic system. Hence, both collision with the environment and self-collision of the robot have to be evaluated.

All of those constraints are semi-infinite: the following section presents how they have been tackled.

Technical aspects of the optimization resolution

Semi-infinite constraints In a first approach, we tried to discretize the semi-infinite constraints on the basis of the simulation steps grid. However, since the number of the grid sample points changes in function of t^{End}, the number of constraints is variable. Subsequently a classical optimization solver can not handle them.

Let c be the evaluation value of a given constraint: $(\forall t \in [t^0, t^{\text{End}}], c(t) < 0)$. We considered associating only one value to the constraint, c_V, that is computed as follows: If the constraint is always satisfied, then c_V is the higher value of $c(t)$. Otherwise, it is the sum of all the violations found at each time step. Considering that the time step can change (e.g. when adding an interpolation point), we choose to weight the added value by the time step δt.

Constraint by task Each task appears only once in the sequence, but a same action can be associated to many tasks. Associating the constraints $\phi(\mathbf{q})$ to the whole simulation can thus raise an issue: a violated constraint can not be linked to the responsible task. In order to compensate this problem, we consider n_T additional sets of constraint $\phi(\mathbf{q})$, noted $\phi_i(\mathbf{q}), i \in [1 \dots n_T]$, (with n_T the number of tasks in the sequence). Each set $\phi_i(\mathbf{q})$ is computed only when the task i is in the SoT.

Scaling Since the constraints are not homogeneous (times, angles, velocities, distances), they have to be normalized based on the constraint values obtained while executing the sequence corresponding to the initial set of parameters $\mathbf{x}_0$. This simple scaling improves significantly the convergence of the optimization.

Absolute versus relative timing

In this parameterization, the tasks are described with an absolute time. As it is, decreasing t_i^I for a task i will not have any direct effect on t_i^F: we have also to decrease t_i^F then decrease t^{End}: it is thus necessary to propagate the reduction for all the following tasks. To avoid this, another parameterization consists in describing the SoT entry time of a given task with respect (i.e. relatively) to the previous one. We introduce a relative timing: each task is now described by two delays (instead of the absolute times t^I and t^F), namely:

1. dt^I: is the delay which occurs between (i) the maximum time of entry or of end of the preceding tasks, and (ii) the SoT entry time of the task in question.
2. dt^F: is the delay between the SoT entry and the removal times of the task in question.

These two delays fulfil the following equations:

$$t_i^I = \max\left(\max_{(j,i)\in S_{I,I}}\{t_j^I\}, \max_{(j,i)\in S_{F,I}}\{t_j^F\}\right) + dt^I \quad (12)$$

$$t_i^F = t_i^I + dt_i^F \quad (13)$$

Subsequently, the new parameter vector is noted : $\mathbf{x}' = [dt_1^I, dt_1^F, \lambda_1^I, \lambda_1^F, \beta_1, \dots, dt_n^I, dt_n^F, \lambda_n^I, \lambda_n^F, \beta_n, t^{\text{End}}]$.

If the task sequence is only a chain of tasks realized one after the other, we directly have $\mathbf{x}' = f(\mathbf{x})$, with f a linear function, and $t^{\text{End}} = \sum_i dt_i^I + \sum_i dt_i^F$

Considering this new set of parameters, the formulation of the optimization problem changes: the tasks constraints $\text{seq}(\mathbf{q})$ are modified. We consider the following tasks on the delay:

$$\forall i, 0 \leq dt_i^I \quad (14)$$

$$\forall i, 0 < dt_i^F \quad (15)$$

These constraints replace the previous constraints (6a), (6d) and (8a). Each task's start depends on its predecessors in the sequence. The other constraints (6e), (8b), (8c), (9) and (10) remain unchanged.

Implementation

Optimization

At each optimization step, the solver chooses a new set of parameters $\mathbf{x}$. It then computes the constraints. Constraints (6e) and (8c), (14) and (15), can be evaluated directly. As stated previously, the other constraints can not be directly computed (since they do not write in an analytical formulation). They are thus evaluated using a complete simulation of their execution. The chosen value of the current optimization variable vector $\mathbf{x}$ is transmitted by the optimization solver to the simulation engine. The simulation returns the evaluation of the constraints and the optimization solver computes a new step vector $\mathbf{x}$, until convergence.

The optimization solver is chosen to be the SQP algorithm from the MATLAB optimization toolbox. The complexity of the optimization does not depend on the number of possible solutions for a set of tasks, but on the number of parameters. Our solver has a linear complexity in the number of parameters $\mathcal{O}(P)$.

Simulation

In section *Generic Task Sequencing*, we presented the computation of desired joint velocities for a hierarchy of tasks, as (7). The simulation is basically a numerical integration of this equation (we used an explicit Euler integration method with a fixed step $\Delta t = 0.005$sec). The starting t_i^I and ending t_i^F times of tasks are continuous variables that are not aligned with the grid. Those instants are important since they correspond to a change in the SoT and thus a change in the control. If postponing the change of control to the next

time step (like on a real system) we will not have a continuous problem (hence potentially raising the same problem described in section). To solve this problem, the entry time t_a for a given task is added as an integration point during the time step $[t, t + \Delta t]$, resulting in the separation of the time step into the two smaller time steps $[t, t_a]$ and $[t_a, t + \Delta t]$.

Initialization
$[t_1^I, t_1^F, \ldots, t_n^I, t_n^F] = \texttt{computeTimes}\,(\mathbf{x})$
$t_{\text{Sim}}^{\text{End}} = \max_i \left(t_i^F\right)$
$t = 0$
while $(t < \max\left(t^{End}, t_{Sim}^{End}\right))$ **do**
  $\Delta t' = \texttt{findTimeStep(t)}$
  `handleStackOfTasks` (t)
  `updateConstraints()`
  $t = t + \Delta t'$
end

Algorithm 1: Tasks sequencing simulation

The algorithm 1 describes the simulation. The function `computeTimes` computes the absolute times using the relative times. The function `findTimeStep` computes the required time step for the Euler integration: the initial Δt, or a smaller one if needed, due to the need of splitting this interval in two. The function `handleStackOfTasks` computes the velocity of the robot induced by to the tasks execution and integrates it, altogether with any other simulated objects or processes, to obtain the new positions.

The simulation engine runs under the AMELIF framework (Evrard et al. 2008), an interactive dynamic simulator for virtual avatars which includes collision detection and task handling according to the SoT formalism. The execution for both simulation and real-robot control is performed by a generic control framework based on (Mansard and Chaumette 2007).

Experiment

Temporal network

The sequence of tasks (Fig. 2) describes a robot taking out a can from the fridge. The corresponding tasks are:

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

This is a complex mission that can not be split into smaller sequences. Indeed, the sequence is centered on the fridge: the grasping part does not make sense if the fridge is closed.

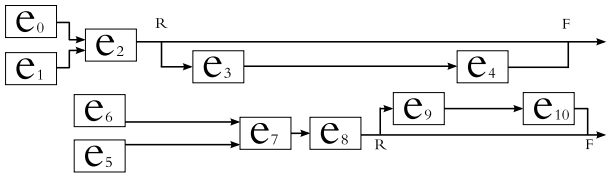

Figure 2: Sequence describing the HRP-2 taking the can in the fridge

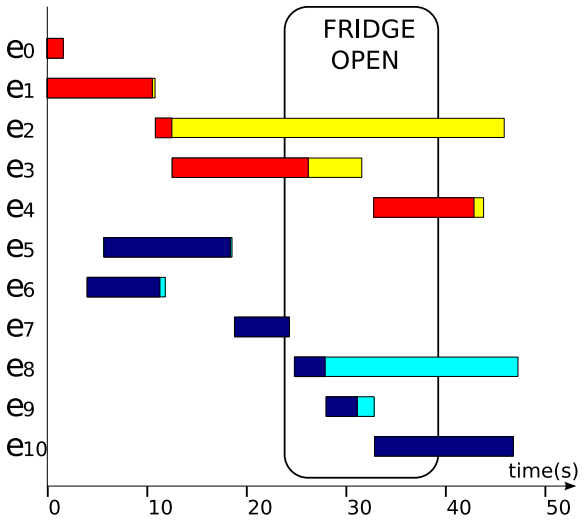

Figure 3: Results of the optimization of the sequence of task: when the task is added in the SoT, its error is first regulated (this corresponds to the dark part (red or dark blue) of the block). From t_i^R, the error is nearly null and the task is kept in the SoT (light part (yellow or cyan) of the block) until t_i^F.

Instead of adding an explicit timing conditions between the tasks to ensure that this will never occur, we choose to consider as constraint the collision between the left arm and the door, in order to test task overlapping.

The constraints considered for this problem are thus sequencing and robotic constraints (joint position and velocity limits), and non-colliding with the fridge.

Results of the optimization

We ran the optimization on a 3GHz desktop PC running under Windows OS. No specific effort of software optimization has been made. The sequence found is described on Fig. 3.

Each task is described by two periods: the dark one is the achievement period $[t_i^I, t_i^R]$, the bright one is the SoT presence period $[t_i^I, t_i^F]$.

The overlaps between the tasks of the left and the right arm appear clearly: the left arm starts to move before the fridge is open. It then starts to move toward the can pose even if the fridge is not completely open. And finally, the right arm starts to close the fridge before the left arm has completely left the fridge area. The whole task sequence lasts 47sec. Without these two overlaps, the robot will move to and grasp the can ($\mathbf{e_7}$) only after the fridge is fully opened

($\mathbf{e_3}$) and it will close the fridge ($\mathbf{e_4}$) only after the can is completely taken out ($\mathbf{e_{10}}$); consequently the total mission would have taken at least 71sec.

For this optimization (66 parameters and 120 constraints), one simulation takes around 1secs. The whole optimization process required 7 hours, mainly due to a first simplistic optimization scheme (in particular, gradients are estimated by finite differences). Reducing this cost is one of our main concerns.

Experiment on the real robot

The SoT formalism allows to directly apply the optimized task sequence as a control law using the same task definition. The task sequence is executed by the robot HRP-2. It is a full-size humanoid robot with 30 actuated DOF. For the experiment, we only used the upper part of the robot (the legs are fixed). Only the described tasks are used to compute the control law (which means that no additional care is taken for ensuring the constraints). For the tasks requiring a 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. However, we did not use a visual feedback from the robot: the configuration of the robot, the fridge and the can were fixed and were identical in the simulation and in the real execution.

The robot manages to grasp the can without colliding any obstacle or joint limits, and respecting the given velocity limits. The obtained execution is plotted on Fig. 4. Thanks to the optimized gain, the convergence of the error of the tasks that require a good precision (grasping the fridge handler and the can) is achieved quickly. On the opposite, the intermediary tasks (before grasping the handler and the can, and while leaving the fridge), which have been introduced only to reduce the collision with the non convex part of the environment, are not fully achieved. This is part of the optimization decision, in order to reduce the execution time. Snapshots of the execution are given in Fig. 5. See http://www.laas.fr/~nmansard/keithfridge.avi.

Discussion and future work

The experimentation on the real robot has been made under the strong hypothesis that the simulation matched precisely the real execution (e.g. the position of all objects in the universe are similar, the robot follows the expected behavior). Unfortunately, it is most likely that some unexpected modifications appear, that will prevent the good execution of the sequence.

We will work on this issue by supervising the execution of the optimized plan during the real experiment. In particular, we plan to use the error of each task at critical times, namely the time of tasks' entry and removal obtained from the optimization as references during the real experimentation (instead of times). It is likely that it's possible to adapt the tasks' behavior in order to adjust the sequence in case of small perturbations. If substencial corrections are required, it is likely that we need to operate at the level of the task planing and subsequently to run a new optimization.

Besides, we need to take into account the dynamic aspects and constraints of the simulation, and we will had more perception tasks such as visual interaction.

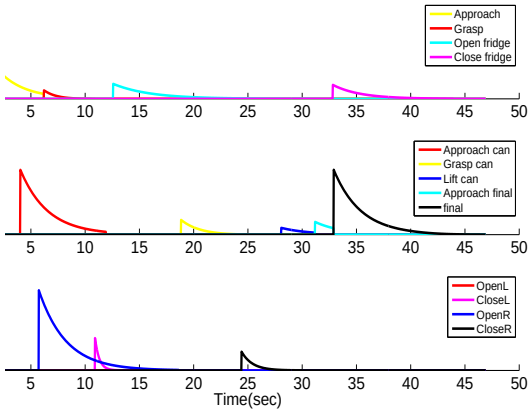

Figure 4: Real experiments: decrease of the errors when the optimized task scheduling is applied on the real robot: (top) right arm tasks (middle) left arm tasks (bottom) gripper tasks. The concurrency beween the tasks is clearly visible.

Conclusion

We devise a method which allows to optimize both the behavior and the overlapping scheduling of a sequence of tasks composing a robotic mission. The solution derives from an optimization formulation of the tasks scheduling keeping the formalism built on the top of a task-function based control. This allows to include the robot limitations as well as collision avoidance as constraints. Our method is exemplified through a complete simulation of a complex mission, where we demonstrated an improvement in the smoothness of the generated motion. For the time being, our method still needs a predefined ordered sequence. As a future work we will increase the autonomy by determining automatically the ordered sequence and compute all the necessary subtasks from definitions of actions/objects associations.

References

Baerlocher, P., and Boulic, R. 2004. An inverse kinematic architecture enforcing an arbitrary number of strict priority levels. *The Visual Computer* 6(20):402–417.

Ben-Israel, A., and Greville, T. 2003. *Generalized inverses: theory and applications*. CMS Books in Mathematics. Springer, 2nd edition.

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann. chapter 12, Temporal Constraint Network.

Evrard, P.; Keith, F.; Chardonnet, J.-R.; and Kheddar, A. 2008. Framework for haptic interaction with virtual avatars. In *Robot and Human Interactive Communication (RO-MAN'08)*.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. Morgan Kauffmann Publishers.

Lamare, B., and Ghallab, M. 1998. Integrating a temporal planner with a path planner for a mobile robot. In *Proc. AIPS Workshop on Integrating planning, scheduling and execution in dynamic and uncertain environments*, 144 –151.

Figure 5: Hrp-2 grasping a can in the fridge.

LaValle, S. 2006. *Planning Algorithms*. Cambridge Univ. Press.

Mansard, N., and Chaumette, F. 2007. Task sequencing for sensor-based control. *IEEE Trans. on Robotics* 23(1):60–72.

Miossec, S.; Yokoi, K.; and Kheddar, A. 2006. 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*.

Py, F., and Ingrand, F. 2004. Dependable exec. control for auton. robots. In *IEEE/RSJ Int. Conf. Intelligent Rob. Sys. (IROS'04)*.

Quinlan, S., and Khatib, O. 1993. Elastic bands: Connecting path planning and robot control. In *IEEE Int. Conf. Robot. Autom. (ICRA'93)*, volume 2, 802–807.

Samson, C.; Le Borgne, M.; and Espiau, B. 1991. *Robot Control: the Task Function Approach*. Clarendon Press, Oxford, UK.

Sentis, L., and Khatib, O. 2006. A whole-body control framework for humanoids operating in human environments. In *IEEE Int. Conf. Robot. Autom. (ICRA'06)*, 2641–2648.

Siciliano, B., and Slotine, J.-J. 1991. A general framework for managing multiple tasks in highly redundant robotic systems. In *IEEE Int. Conf. on Advanced Robotics (ICAR'91)*.