Optimized Time-Warping Tasks Scheduling for Smooth Sequencing
François Keith, Nicolas Mansard, Sylvain Miossec, Abderrahmane Kheddar

To cite this version:
Optimized Time-Warping Tasks Scheduling for Smooth Sequencing

François Keith∗,∗∗, Nicolas Mansard∗∗, Sylvain Miossec***, Abderrahmane Kheddar∗,∗∗∗

∗ CNRS-LIRMM, Montpellier, France (email: {keith,kheddar}@lirmm.fr)
** CNRS-LAAS, Toulouse, France (e-mail: nmansard@laas.fr)
*** PRISME-Univ. d’Orléans, Bourges, France (e-mail: sylvain.miossec@bourges.univ-orleans.fr)
**** CNRS-AIST JRL, UMI3218/CRT, Tsukuba, Japan

Abstract: This paper presents an optimal formulation for sequencing a set of robotic tasks. Tasks description and execution are based on the task formalism. A naive solution would be to arrange the execution of the tasks sequentially (potential redundancy not exploited, not optimal), or to set manually the timing and the behavior for each task (burdensome for the user, lack of autonomy, not optimal), or to arrange the tasks by priority and pill them up at once in the stack-of-tasks (high likelihood of conflicts, wrong or cumbersome execution). Our contribution is to determine the time-optimal realization of the mission taking into account robotic constraints as complex as collision avoidance; its originality lies in keeping the task formalism in the formulated optimization of the task sequencing problem. This theory is exemplified through a simulation on a 2D robot.

Keywords: Scheduling algorithms, Task-based control, Optimization, Robotics

1. 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. That is, missions undergo functional objective decomposition into a set of processes or operations that can be defined as templates. Each operation can be decomposed into a set of tasks (i.e. generic sensory-motor functions), and each task can be easily mapped into robot execution. The whole scheme may constitute an exploitable generic skill/behavior. Yet, various levels of decomposition can be achieved depending on the envisaged software/hardware implementation, additional environment constraints, the human-machine interface, etc. In the end, the robot is assigned with a sequence of tasks to realize a given mission. A typical example of such a mission decomposition is given in Fig. 1.

Numerous works have been proposed to obtain such a sequence of tasks from a given mission and a set of causal paradigms (Dechter (2003); Ghallab et al. (2004)). However, they generally produce a symbolic plan, where the only numerical precisions lie on the scheduled time data. 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 then often specifically hard coded) to enhance the numerical trajectory with symbolic data, that would help re-computing only part of the plan (Py and Ingrand (2004)) or distort locally the trajectory to apprehend small environment changes (Quinlan and Khatib (1993); Lamiraux et al. (2004)).

Rather than using a trajectory planner between the temporal reasoning and its robotic real execution, we suggest
using a sensory-motor control approach based on task components. The task function (Samson et al. (1991)) or the operational space formulation (Khatib (1987)) are elegant approaches to produce intuitively robot objectives. They also allow to address the control problem directly in the sensor space, improving robustness of the action execution against environment uncertainties and variability (Espiau et al. (1992); Chaumette and Hutchinson (2006)). They are trajectory free, which means that it is not necessary to explicitly compute all trajectories before the execution or during the execution, namely in response to environment changes. Moreover, since a same task space is valid for a large set of robots, a control scheme based on task formalism is certainly portable and easy to modify and to maintain. In addition, these methods include a kinematic or dynamic formulation that decouples the task space from the null-space (i.e. the joint space that let the task invariant) (Liégeois (1977); Hanafusa et al. (1981)). A secondary task can then be applied in the null-space, and, recursively, a hierarchic set of tasks (or stack-of-tasks) can be considered (Nakamura et al. (1987); Siciliano and Slotine (1991)). Hierarchy of tasks are becoming popular to build complex behavior for very redundant robot such as humanoids (Baerlocher and Boulic (2004); Bolder et al. (2007); Mansard et al. (2007); Sian et al. (2005); Sentis and Khatib (2006)). The formalism introduced in (Mansard and Chaumette (2007)) proved to be efficient in dealing with complex humanoid missions: the Stack of Tasks (SoT) is mainly a hierarchy of tasks driving the robot while ensuring locally a given set of constraints to be satisfied. We make use of this formalism (Section 2).

A task (i.e. a task function (Samson et al. (1991))) can be directly linked to the symbols on which the task temporal network is reasoning (e.g. reaching an object to be grasped is a task that requires the robot arm to be available, and whose post-condition is to have the gripper on the object – it is also directly described by a sensory-motor function applicable to the SoT).

Mission decomposition is thus executable directly by a SoT, which guarantees good robustness and avoid unnecessary trajectory (re)computation. However, exclusive task sequencing on the robot produces generally saccadic movements which may look to humans as monotonous automated motions. On the other hand, it is difficult for the temporal network to produce a scheduling with task overlapping when tasks’ concurrency is restricted by physical limitations of the robot (for example obstacles or dynamical constraints on a humanoid). Since the problem is not in a discret form –i.e. the resources (mainly redundancy in motion generation) are not discrete –, task scheduling techniques can not apply directly to our problem. On the contrary, semi-infinite optimization techniques (Lee et al. (2005), Miossec et al. (2006)) can be used to generate low level trajectories for the overall execution, providing the problem to be formulated this way. One of the drawbacks of such an approach is that the generated trajectory does not necessarily account for the robot controller and is not robust to environment uncertainties: we propose a method that encapsulates the tasks concept to solve this issue.

The overall architecture, including the optimized task sequencing, is illustrated in Fig. 2. Given a set of elementary tasks sequence to achieve a given mission, our solution re-
They can be defined as follow:

1. the force closure on it (\( F \)) for example, first we have to grasp an object and maintain the position.

\( \parallel e \parallel \) is considered as regulated when \( \parallel e \parallel \leq \lambda_0 \). However, they do not indicate the achievement level of the task, contrary to Allen Logic, that only considers the start and end points of the time interval, here is also considered the regulation time \( t^F \).

2.2 Gain handling

The simple attractor presented in (3) produces a nice exponential decrease. However, it also produces a strong acceleration at the beginning of the task, while at the end of the task, \( \parallel e \parallel \) decreases slowly (slow convergence). A very classical “trick” when regulating a task is to rather use an adaptive gain \( \lambda = \lambda(e(t)) \), for example:

\[
\lambda(e) = (\lambda^F - \lambda^I) \exp\left(-\frac{\|e\|}{\beta}\right) + \lambda^I
\]

with \( \lambda^I \) the gain at infinity, \( \lambda^F \) the gain at regulation (\( \lambda^I \leq \lambda^F \)) and \( \beta \) the slope at regulation.

2.3 Sequence of tasks

A task sequence is a finite set of 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), 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 \([t^i_F, t^i_I]\) 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. Thus we define the tolerance on the task regulation \( \epsilon_i \): a task is considered as regulated when \( \|e_i(t)\| \leq \epsilon_i \). The regulation time \( t^i_R \) is defined by \( \|e_i(t^i_R)\| = \epsilon_i \).

A task sequence is characterized by a set of time-constraints binding the schedules of two tasks \( e_i \) and \( e_j \). They can be defined as follow: \( e_i \) must begin or end once \( e_j \) has begun, has ended or has been regulated. A graphical representation of such time-constraints is given in Fig. 3.

For example, first we have to grasp an object and maintain the task \( e_A \) before we can move it \( e_B \). The task \( e_B \) can only start once the task \( e_A \) has been realized, and must end before the task \( e_A \).

2.3 Sequence of tasks

A task sequence is a finite set of 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), 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 \([t^i_F, t^i_I]\): 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. Thus we define the tolerance on the task regulation \( \epsilon_i \): a task is considered as regulated when \( \|e_i(t)\| \leq \epsilon_i \). The regulation time \( t^i_R \) is defined by \( \|e_i(t^i_R)\| = \epsilon_i \).

A task sequence is characterized by a set of time-constraints binding the schedules of two tasks \( e_i \) and \( e_j \). They can be defined as follow: \( e_i \) must begin or end once \( e_j \) has begun, has ended or has been regulated. A graphical representation of such time-constraints is given in Fig. 3.

For example, first we have to grasp an object and maintain the task \( e_A \) before we can move it \( e_B \). The task \( e_B \) can only start once the task \( e_A \) has been realized, and must end before the task \( e_A \).
the semi-infinite nature of the constraints. Therefore we expanded the semi-infinite constraint into a discret form.

3.2 Constraints

\( x \) must satisfy both the sequencing and the robotic time-constraints enumerated hereafter:

**Tasks constraints:** noted seq\( (q) \) and defined as follow:

- Time coherence for each task \( i \), that is:
  \[ \forall i, 0 \leq t_i^t < t_i^F \leq t_i^{\text{End}} \]  
  (8)

- Termination condition for each task \( i \), that is:
  \[ \forall i, \| s_i^t - s_i(t_i^F) \| < \epsilon_i \]  
  (9)

- Any task sequence condition for a given couple of tasks \( i \) and \( j \) described in (6)
- The control gain, namely:
  \[ \forall i, \lambda_i^t \leq \lambda_i^F \]  
  (10)

The constraints (6a), (6c), (6d) and (8) are linear. On the contrary, the constraint (9) is impossible to compute directly using \( x \), and is determined from a simulation of the execution. Care has to be taken while resolving the condition described by (6b). Indeed, discretizing \( t_i^R \) 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 re-formulate (6b) and evaluate the regulation of the task \( i \) when the task \( j \) begins. The constraint (6b) becomes:

\[ \forall (i,j) \in S_{R,I}, \| s_i^t - s_j(t_j^F) \| \leq \epsilon_i \]  
(11)

**Robot constraints:** \( \phi(q) \)

Those constraints are mainly due to hardware intrinsic limitations of the robot:

- Joint limits, given by
  \[ q_{\text{min}} \leq q \leq q_{\text{max}} \]  
  (12)

- Velocity limits, given by
  \[ \dot{q}_{\text{min}} \leq \dot{q} \leq \dot{q}_{\text{max}} \]  
  (13)

- Collision avoidance between a given pair of objects \( i \) and \( j \), given by
  \[ d_{ij} \geq 0 \]  
  (14)

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

3.3 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 \( t_r^{\text{End}} \), the number of constraints is variable. Subsequently a classical optimization solver can not handle it.

Let \( c \) be the evaluation value of a given constraint: \((\forall t \in [t_i^t, t_i^F], c(t) < 0)\). 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.

The reformulation \( c_v < 0 \) acts similarly to the semi-infinite constraint \( c(t) \).

**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 \( x_t \). This simple scaling improves significantly the convergence of the optimization.

3.4 Absolute versus relative timing

In this parameterization, the tasks and constraints are described with an absolute time. As it is, decreasing \( t_i^t \) for a task \( i \) will not have any direct effect on \( t_i^F \); we have also to decrease \( t_i^F \) then decrease \( t_i^{\text{End}} \); 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 \( t_i^t \) and \( t_i^F \)), namely:

(1) \( dt_i^t \): is the delay which occurs between (i) the maximum entry time and the end time of the preceding tasks, and (ii) the SoT entry time of the task in question.

(2) \( dt_i^F \): is the delay between the SoT entry time of the task in question and the removal times of the task in question.

These two delays fulfil the following equations:

\[ t_i^t = \max \left( \max_{(i,j) \in S_{R,I}} \{ t_j^t \}, \max_{(i,j) \in S_{R,I}} \{ t_j^R \}, \max_{(i,j) \in S_{R,I}} \{ t_j^f \} \right) + dt_i^t \]  
(15)

\[ t_i^F = t_i^t + dt_i^F \]  
(16)

Subsequently, the new parameter vector is noted:

\[ \chi' = [dt_1^t, dt_1^F, \lambda_1^t, \lambda_1^F, \beta_1, \ldots, dt_n^t, dt_n^F, \lambda_n^t, \lambda_n^F, \beta_n] \]

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

Considering this new set of parameters, the formulation of the optimization problem changes. The objective remains the same: minimize \( t_r^{\text{End}} \), while the constraints become:

- Constraints on the delay:
  \[ \forall i, 0 \leq dt_i^t \]  
  (17)

- Constraints of termination:
  \[ \forall i, \| s_i^t - s_i(t_i^F) \| \leq \epsilon \]  
  (19)
• Constraint on the task sequence
\[ \forall(i,j) \in S_{F,F}, t_i^F \leq t_j^F \quad (20) \]
The constraints (6b), (6c) and (8) are now replaced by the constraints (17) and (18). Each task’s start depends on its predecessors in the sequence. Because of the possible dependence on the regulation time, the absolute times \( t_i^F \), \( t_R^F \) and \( t_F^F \) can not be pre-computed, and are computed during the simulation.

4. IMPLEMENTATION

4.1 Presentation

At each optimization step, the solver chooses a new set of parameters \( x \). It then computes the constraints. Constraints (17) and (18) 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 \( x \) 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 \( x \), until convergence. The optimization solver is chosen to be the SQP algorithm from the MATLAB optimization toolbox; the simulation engine runs also under MATLAB.

4.2 Simulation

In Section 2, we presented the computation of the joint-velocities control law (7). The simulation is basically a numerical integration of this last equation (using an explicit Euler integration method with a fixed step \( \Delta t = 0.005 \text{sec} \)). The starting \( t_i^F \) and ending \( t_F^F \) 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 3.2). 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
\[
t_{\text{End}} = t_{\text{max}} = \sum dt_i^I + \sum dt_i^F
\]
\[
t = 0
\]
while \((t < t_{\text{End}})\) do
\[\left[ t_i^I, t_i^F, t_F^F, \ldots, t_i^I, t_i^F, t_F^F, \ldots, t_{\text{End}} \right] = \text{updateTimes}(t)\]
\[\Delta t' = \text{findTimeStep}(t)\]
\[
\text{handleStackOfTasks}(t) \\
\text{updateConstraints}() \\
t = t + \Delta t'
\]
end
return \( t_{\text{End}} \)

Algorithm 1: Tasks sequencing simulation

The function \text{findTimeStep} computes the required time step for the Euler integration: the initial \( \Delta t \), or a smaller one if needed, due to the need of splitting this interval

5. EXPERIMENT

5.1 Temporal network

The sequence of tasks (Fig. 4) describes a robot taking out an object placed behind a door. The corresponding tasks are:

• \( e_1 \): Move the right arm to the door
• \( e_2 \): Close the right gripper
• \( e_3 \): Open the door
• \( e_4 \): Move the left gripper to the object
• \( e_5 \): Close the left gripper
• \( e_6 \): Remove the object out
• \( e_7 \): Close the door
• \( e_8 \): Open the right gripper
• \( e_9 \): Move the right arm away

This is a complex mission that can not be split into smaller sequences. Indeed, the sequence is centered on the door: the grasping part does not make sense if it is closed. Instead of adding an explicit timing conditions between the tasks to ensure that this will never occur, we chose to consider as constraint the collision between the left arm and the door, in order to allow task overlapping (the condition \( t_i^I \leq t_i^F \) only postpones the entry time of the task without preventing overlapping). Similarly, we considered the collision constraint between the gripper and the handle or the object. The gripper closing tasks are used to grasp and hold objects, so these tasks have to be maintained as long as necessary.

The constraints considered for this problem are thus sequencing and robotic constraints (joint position and velocity limits).

5.2 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. (5).

The overlaps between the tasks of the left and the right arm appear clearly: the left arm starts to move before the door is open. It then starts to move toward the object pose even if the door is not completely open. And finally, the right arm starts to close the door before the left arm has completely left the door area. The whole task sequence lasts 46.9sec. Without these two overlaps, the robot would have taken 54sec.

Fig. 4. Sequence describing the robot taking the object in two. The function \text{handleStackOfTasks} computes the velocity of the robot induced by the tasks execution and integrates it, altogether with any other simulated objects or processes, to obtain the new positions.
6. 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. We will also focus on more complex scenarios, doable by a humanoid robot, using in particular visual interaction.

REFERENCES


