Constrained Path Planning using Quadratic Programming

Franco Fusco, Olivier Kermorgant, Philippe Martinet

Constrained Path Planning using Quadratic Programming

Franco Fusco[1], Olivier Kermorgant[1] and Philippe Martinet[1,2]

***Abstract*— Sampling-based planning algorithms have been extensively exploited to solve a wide variety of problems. In recent years, many efforts have been dedicated to extend these tools to solve problems involving constraints, such as geometric loop-closure, which lead the valid Configuration Space (CS) to collapse to a lower-dimensional manifold.**

One proposed solution considers an approximation of the constrained Configuration Space that is obtained by relaxing constraints up to a desired tolerance. The resulting set has then non-zero measure, allowing to exploit classical planning algorithms to search for a path connecting two given states. When constraints involve kinematic loops in the system, relaxation generally bears to undesired contact forces, which need to be compensated during execution by a proper control action.

We propose a new tool that exploits relaxation to plan in presence of constraints. Local motions inside the approximated manifold are found as the result of an iterative scheme that uses Quadratic Optimization to proceed towards a new sample without falling outside the relaxed region. By properly guiding the exploration, paths are found with smaller relaxation factors and the need of a dedicated controller to compensate errors is reduced. We complete the analysis by showing the feasibility of the approach with experiments on a real platform.

I. INTRODUCTION

Planning a motion via exact techniques for a robot with many degrees of freedom (dof) is hard and often impractical. Sampling-based planners overcome computational issues by exploring only a finite subset of the Configuration Space (CS) and checking for collisions at the considered states, rather than attempting to obtain a full representation of the C-obstacle regions. In particular, algorithms like Probabilistic Road Maps (PRM) [1] and Rapidly-exploring Random Trees (RRT) [2] approximate the valid CS of a system by means of a graph whose nodes correspond to configurations and edges to motions between pairs of samples. To expand the graph, random configurations can be obtained by drawing each component independently from a given random distribution, while local motions are often created using linear interpolation.

These techniques can be applied to a number of contexts, but when dealing with tasks like sliding an object on a surface or carrying a load with two arms their performances become unsatisfactory. The reason is that in presence of such category of constraints the CS degenerates to a manifold with zero-measure. The probability of randomly sampling a valid configuration belonging to an n-dimensional subset using uniform distributions is directly related to the measure of the set itself, being null in case of lower-dimensional manifolds. Furthermore, linear interpolation fails to connect valid samples, since it produces straight paths that likely fall outside the valid CS.

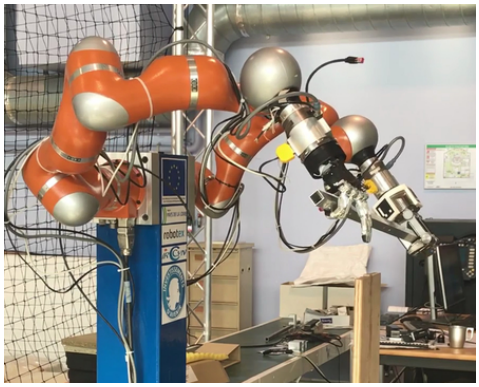


Fig. 1. The experimental platform at LS2N: a dual-arm robot featuring two Kuka LWR manipulators with a Schunk SDH (left hand) and Schunk SVH (right hand).

Different strategies can be found in the literature to tackle the problem and extend randomized techniques to the constrained setup, to generate both feasible samples [3] and valid motions. A projection approach is used by the *Constrained Bi-Directional Rapildy-exploring Random Tree* (CBiRRT) [4] which exploits the gradient descent algorithm to push invalid configurations onto the manifold. Two samples are connected by alternately performing a linear step toward the goal and a projection of the resulting (infeasible) configuration back to the valid set. Other recent works [5], [6] rely on the construction of an *Atlas* whose charts consist in local linearizations of the constraint manifold. A planner is then required to explore the charts rather than the manifold, using linear interpolation while moving inside a single chart.

The concept of *relaxation* [7], [8] consists in allowing a small constraint violation during planning. As a result, the valid CS is approximated by a full-dimensional set, which can then be explored by standard sampling-based planners. The technique was exploited to plan motions for compliant closed-loop systems, for whom infeasible samples lead to undesired internal forces. These efforts are compensated by a proper control action which steers the configuration back to the constraint manifold.

Since relaxation highly relies on system's compliance and on the control action, it might be infeasible in many scenarios involving rigid robots. These systems would need paths with lower constraint violation directly from the planning step, which can be achieved only by reducing the considered relaxation factor. However, by doing so the topology of the valid

[1]Centrale Nantes, Laboratoire des Sciences du Numérique de Nantes LS2N, France franco.fusco/olivier.kermorgant@ls2n.fr

[2]Inria Sophia Antipolis, France philippe.martinet@inria.fr

Configuration Space changes to a set of extremely narrow passages and the planning time increases significantly.

We propose a new approach based on relaxation which allows to find, in a shorter amount of time, paths characterized by lower constraint violation. The objective is to reduce the necessity of a control action during execution. We support the feasibility of the approach by including real experiments on a dual-arm system that carries an object while avoiding obstacles.

The rest of the paper is organized as follows. In Section II constraints are formally introduced and the relaxed Configuration Space is defined, while the planning algorithm is detailed in Section III. Real experiments involving geometric loop-closure constraints were performed to test the planner, and are discussed in Section IV. We report our conclusions and future work in Section V.

II. CONSTRAINTS AND RELAXATION

In this Section constraints are introduced, distinguishing between two classes, and a formal definition of the relaxed manifold is presented. We then detail the formulation of loop-closure constraints exploited in real experiments.

The Configuration Space $\mathcal{C}$ is assumed to be an n-dimensional subset of $\mathbb{R}^n$, *i.e.*, $\mathcal{C} \subset \mathbb{R}^n$, n corresponding to the dof of the system. The configuration vector, denoted as $\mathbf{q} = \begin{bmatrix} q_1 & \cdots & q_n \end{bmatrix}^T$, is assumed to have bounded coordinates $q_i \in [q_{i,\min}\,,\, q_{i,\max}]$ $(i = 1, \cdots, n)$.

Constraints are distinguished in two classes: (1) differentiable and (2) non-differentiable. The first category is represented by a set of n_1 functions in the form $C_i : \mathcal{C} \to \mathbb{R}$, for whom the gradient vector is assumed to be well defined and available in analytic form. They define the constraint manifold as the set $\mathcal{M} = \{\mathbf{q} \in \mathcal{C} : C_i(\mathbf{q}) = 0, \forall i\}$. Examples from this category are kinematic loop-closure or orientation constraints on the end-effector.

The second class of constraints is instead described by n_2 functions $D_i : \mathcal{C} \to \{0, 1\}$. A configuration $\mathbf{q}$ satisfies a constraint in this form if $D_i(\mathbf{q}) = 1$. This formulation can be used, *e.g.*, to describe the collision state of a robot, and in general those constraints that do not cause the CS to collapse to lower-dimensional sets.

Relaxation is introduced by considering n_1 constants $\varepsilon_i > 0$, each one quantifying the maximum allowed violation of the corresponding differentiable constraint. This bears to a natural definition of the relaxed Configuration Space as:

$$\mathcal{C}_\mathrm{R} = \{\mathbf{q} \in \mathcal{C} : |C_i(\mathbf{q})| \leq \varepsilon_i \cap D_{i'}(\mathbf{q}) = 1 \; \forall i, i'\} \quad (1)$$

As functions C_i provide a numerical value for constraints violation, we adopt in the rest of this work the terminology *error* to refer to the value of constraints at a given configuration $\mathbf{q}$. We therefore introduce the vector $\mathbf{e} = \begin{bmatrix} C_1(\mathbf{q}) & \cdots & C_{n_1}(\mathbf{q}) \end{bmatrix}^T$ to group all constraints violations and introduce the Jacobian matrix $\mathbf{J}_\mathrm{e} = \frac{\partial \mathbf{e}}{\partial \mathbf{q}}$, whose i-th row is given by the gradient (as a row vector) of C_i. In order to explore $\mathcal{C}_\mathrm{R}$, the Jacobian is used to locally approximate $\mathcal{M}$, and motions are performed with the objective of minimizing the norm of $\mathbf{e}$.

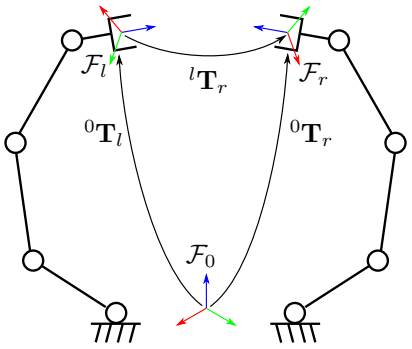


Fig. 2. Reference frames and transformations in a dual-arm system.

We conclude this section by showing a possible choice of constraint functions to describe loop-closure for a dual-arm system rigidly grasping an object with both end-effectors. Fig. 2 shows a sketch of a dual-arm system, wherein three frames are considered: $\mathcal{F}_0$ (world reference frame), $\mathcal{F}_l$ (left end-effector frame) and $\mathcal{F}_r$ (right end-effector frame). We assume the full configuration vector to be partitioned as $\mathbf{q} = \begin{bmatrix} \mathbf{q}_l^T & \mathbf{q}_r^T \end{bmatrix}^T$.

Rigid grasp of an object constrains relative translation and rotation of frames $\mathcal{F}_l$ and $\mathcal{F}_r$, that is, the transformation matrix ${}^l\mathbf{T}_r$ needs to be constant. Let $\mathbf{t}^\star$ be the desired translation and ${}^l\mathbf{R}_r^\star$ the desired rotation matrix in ${}^l\mathbf{T}_r$. It is then possible to define the constraint error as:

$$\mathbf{e} \doteq \begin{bmatrix} \mathbf{t}_l + {}^0\mathbf{R}_l\, \mathbf{t}^\star - \mathbf{t}_r \\ \theta \mathbf{u} \end{bmatrix} \quad (2)$$

$\mathbf{t}_l$ and $\mathbf{t}_r$ being the position of the end-effectors in world frame, and $\theta\mathbf{u}$ the angle-axis decomposition of the rotation matrix ${}^0\mathbf{R}_r^T\, {}^0\mathbf{R}_l\, {}^l\mathbf{R}_r^\star$.

Finally, the Jacobian writes:

$$\mathbf{J}_\mathrm{e} = \begin{bmatrix} \mathbf{M}\mathbf{J}_l & -\mathbf{J}_r^{(v)} \\ \mathbf{B}\, {}^0\mathbf{R}_r^T\, \mathbf{J}_l^{(\omega)} & -\mathbf{B}\, {}^0\mathbf{R}_r^T\, \mathbf{J}_r^{(\omega)} \end{bmatrix} \quad (3)$$

where: $\mathbf{J}_l$ is the Jacobian matrix of the left end-effector, $\mathbf{J}_r^{(v)}$ the right arm Jacobian of linear velocity, $\mathbf{J}_l^{(\omega)}$, $\mathbf{J}_r^{(\omega)}$ are the left and right Jacobians of angular velocity. Matrices $\mathbf{M}$ and $\mathbf{B}$ are given by:

$$\mathbf{M} = \begin{bmatrix} \mathbf{I}_3 & -\left[{}^0\mathbf{R}_l\, \mathbf{t}^\star\right]_\times \end{bmatrix} \quad (4)$$

$$\mathbf{B} = \mathbf{I}_3 - \frac{\theta}{2}\left[\mathbf{u}\right]_\times + \left(1 - \frac{\operatorname{sinc}\theta}{\operatorname{sinc}^2\frac{\theta}{2}}\right)\left[\mathbf{u}\right]_\times^2 \quad (5)$$

$\mathbf{I}_3$ representing the 3-by-3 identity matrix and $[\cdot]_\times$ the skew-matrix operator. The derivation of the Jacobian is detailed in the Appendix.

III. PLANNING WITH CONSTRAINTS

In this section the relaxation-based planner is presented. The core of the algorithm, presented in Section III-A, is a local router that tries to reach a given configuration starting from an initial sample. It produces a discrete set

of configurations, each one being obtained after solving a Quadratic Optimization. In Section III-B we show how the connection algorithm is integrated in a Bi-directional RRT to explore the manifold and connect the start configuration to the goal one. We finally propose some post-processing techniques to refine a path in Section III-C.

A. Local Router

The idea behind the local routing algorithm is that the motion toward a desired configuration can be generated by optimizing both the current distance to the local goal and the error vector $\mathbf{e}$. The two objectives will be in general in contrast, since to approach the goal a system will often be required to leave the constraint manifold. Quadratic Programming (QP) is then exploited to find a compromise that allows to shorten the distance towards the goal without excessively violating the constraints.

To enforce also non-differentiable constraints such as collision checking, we require the optimization to produce a sample that is not far away from the current configuration. If at the new point $\mathbf{q}_{new}$ the condition $D_i(\mathbf{q}_{new}) = 1$ is satisfied for every non-differentiable constraint, a new step can be performed using QP. Thus, the local motion is generated and validated iteratively by repeating the two phases.

In the sequel, we firstly show how a single optimization is formulated and solved and then we detail the full iterative scheme.

1) Quadratic Programming: For the purposes of this work, a QP optimization problem is modeled as follows:

$$\mathbf{q}^{(j+1)} = \underset{\mathbf{q}^{(j+1)} \in \mathcal{C}}{\arg\min} \left\| \mathbf{Q}^{(j)}\, \mathbf{q}^{(j+1)} - \mathbf{v}^{(j)} \right\|^2 \tag{6}$$

$$\text{subject to:} \quad \mathbf{A}^{(j)}\, \mathbf{q}^{(j+1)} \leq \mathbf{b}^{(j)} \tag{7}$$

The optimization variable, $\mathbf{q}^{(j+1)}$, corresponds to the new configuration that will be appended to the local motion. Since the router will produce a number of intermediate samples, we use the index j to denote the last generated configuration. The objective is modeled via $\mathbf{Q}^{(j)} \in \mathbb{R}^{m \times n}$ and $\mathbf{v}^{(j)} \in \mathbb{R}^m$. The superscript (j) is used to underline that these quantities are constant during the optimization, fully determined by the value of $\mathbf{q}^{(j)}$. The optimization is subject to the set of linear inequalities given by (7) ($\mathbf{A}^{(j)}$ and $\mathbf{b}^{(j)}$ being of dimension $p \times n$ and p respectively).

In a first instance, the minimization of $\left\|\mathbf{q}^{(j+1)} - \mathbf{q}_g\right\|^2 + \left\|\alpha\, \mathbf{e}_{\text{lin}}^{(j+1)}\right\|^2$ could be considered, $\mathbf{q}_g$ being the local goal configuration, α a diagonal weighting matrix and $\mathbf{e}_{\text{lin}}^{(j+1)}$ the linearized error corresponding to $\mathbf{q}^{(j+1)}$, given by:

$$\mathbf{e}_{\text{lin}}^{(j+1)} = \mathbf{e}^{(j)} + \mathbf{J}_{\text{e}}^{(j)} \left(\mathbf{q}^{(j+1)} - \mathbf{q}^{(j)} \right) \tag{8}$$

This formulation however presents a drawback: in early steps the distance to $\mathbf{q}_g$ will be higher, and it will reduce after each iteration. The relative weight between the two terms in the objective would then be non-uniform during the iterative procedure, resulting in generally higher violations at the beginning of a local motion. A better behavior can be obtained by minimizing in the objective the distance to an intermediate point $\mathbf{q}_g^{(j)}$, obtained using linear interpolation from $\mathbf{q}^{(j)}$ to $\mathbf{q}_g$, *i.e.*, $\mathbf{q}_g^{(j)} = \mathbf{q}^{(j)} + \delta \left(\mathbf{q}_g - \mathbf{q}^{(j)} \right)$, for a given scalar $\delta \in (0, 1]$ selected to ensure homogeneous relative weight of the two objectives.

The objective is then defined, in a single iteration, by the following values:

$$\mathbf{Q}^{(j)} = \begin{bmatrix} \mathbf{I}_n \\ \alpha\, \mathbf{J}_{\text{e}}^{(j)} \end{bmatrix} \qquad \mathbf{v}^{(j)} = \begin{bmatrix} \mathbf{q}_g^{(j)} \\ \alpha \left(\mathbf{J}_{\text{e}}^{(j)}\, \mathbf{q}^{(j)} - \mathbf{e}^{(j)} \right) \end{bmatrix} \tag{9}$$

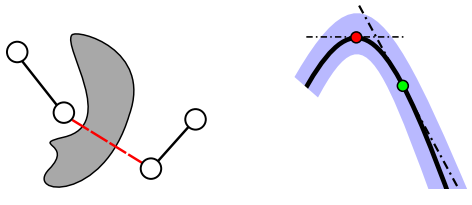
To complete the definition of a QP instance, the set of inequalities (7) is now detailed. As we assume each coordinate of the configuration vector to be bounded, as mentioned in Section II, a first set of inequalities is exploited to enforce these limits:

$$\mathbf{q}_{min} \leq \mathbf{q}^{(j+1)} \leq \mathbf{q}_{max} \tag{10}$$

A further set of inequalities imposes a maximum local displacement on each coordinate during a single iteration:

$$\mathbf{q}^{(j)} - \beta^k \Delta\mathbf{q} \leq \mathbf{q}^{(j+1)} \leq \mathbf{q}^{(j)} + \beta^k \Delta\mathbf{q} \tag{11}$$

the entries of vector $\Delta\mathbf{q}$ being positive constants and introducing $\beta \in (0, 1)$ and $k \in \mathbb{N}$. These inequalities play a double role in the optimization: on one hand, they prevent the distance between subsequent configurations from being too large. This is fundamental in order to consistently check non-differentiable constraints. Considering collision as an example, small obstacles could be wrongly overtaken if the path is too sparse, as depicted in Fig. 3(a). On the other hand, the inequalities can be used to reduce the impact of linearization introduced in (8). Depending on the manifold, a first-order approximation may not well reflect the topology of the valid set, as exeplified in Fig. 3(b). Thus, a step guided by the error Jacobian might lead to a sample that is indeed outside the relaxed CS. Whenever this happens, it is possible to shrink the allowed displacement by increasing k and to repeat the optimization step.



(a) (b)

Fig. 3. Pictorial justification of inequalities (11). In (a) an obstacle (in gray) is not detected due to a large step, and two configurations are wrongly connected. In (b) a manifold is represented as the bold black line, and it is linearized around two points. Due to the high curvature at the red point, a new configuration could easily fall outside the relaxed region (in light blue) even with a small step, while around the green point the linear approximation is valid in a larger area.

2) Iterative Scheme: The motion connecting two configurations $\mathbf{q}_s$ and $\mathbf{q}_g$ is found by iteratively applying the procedure illustrated above to produce new intermediate samples. Algorithm 1 contains the pseudo-code of the local router. At the beginning of an iteration, the QP instance is initialized by computing $\mathbf{Q}^{(j)}$, $\mathbf{v}^{(j)}$ and $\mathbf{A}^{(j)}$. An inner cycle tries to generate a new sample $\mathbf{q}^{(j+1)}$ for increasing values of the parameter k. If the new configuration falls outside the relaxed CS, the optimization is repeated using a smaller maximum step size.

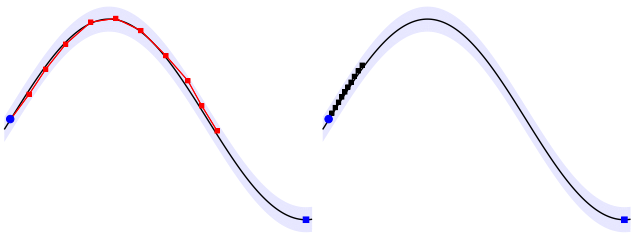
Algorithm 1 QP-based Motion Validator (QPMove)

QPMove($\mathbf{q}_s$, $\mathbf{q}_g$) :
$\mathbf{q}^{(0)} \leftarrow \mathbf{q}_s$
$\sigma \leftarrow \{\ \}$
$d^{(0)} \leftarrow \|\mathbf{q}_a - \mathbf{q}_b\|$
for j=0 **to** $j_{\max}$ **do**
  $k \leftarrow 0$
  $\mathbf{Q}^{(j)}$, $\mathbf{v}^{(j)}$, $\mathbf{A}^{(j)}$ $\leftarrow$ INIT_QP($\mathbf{q}^{(j)}$, $\mathbf{q}_b$)
  do
    if $k \geq k_{\max}$ **then**
      return "stop", σ
    end if
    $\mathbf{b}^{(j)} \leftarrow$ GET_QP_B_VECTOR($\mathbf{q}^{(j)}$, k)
    $\mathbf{q}^{(j+1)} \leftarrow$ SOLVE_QP($\mathbf{Q}^{(j)}$, $\mathbf{v}^{(j)}$, $\mathbf{A}^{(j)}$, $\mathbf{b}^{(j)}$)
    $\mathbf{e}^{(j+1)} \leftarrow$ EVALUATE_ERROR($\mathbf{q}^{(j+1)}$)
    $k \leftarrow k + 1$
  while not($-\boldsymbol{\varepsilon} \leq \mathbf{e}^{(j+1)} \leq \boldsymbol{\varepsilon}$)
  if $\exists i : D_i\left(\mathbf{q}^{(j+1)}\right) = 0$ **then**
    return "stop", σ
  end if
  $\sigma \leftarrow \sigma \cup \left\{\mathbf{q}^{(j+1)}\right\}$
  $d^{(j+1)} \leftarrow \left\|\mathbf{q}^{(j+1)} - \mathbf{q}_b\right\|$
  if $d^{(j+1)} \leq d_{min}$ **then**
    return "success", σ
  end if
  if $d^{(j+1)} \leq d^{(j)}$ **and** $d^{(j)} - d^{(j+1)} \leq \Delta d^{(-)}$ **then**
    return "stop", σ
  end if
  if $d^{(j+1)} \geq d^{(j)}$ **and** $d^{(j+1)} - d^{(j)} \geq \Delta d^{(+)}$ **then**
    return "stop", σ
  end if
end for
return "stop", σ

As soon as a new configuration lying in the relaxed region is found, non-differentiable constraints are evaluated at the sample. If any of them is violated the routine must be interrupted and the local path is returned, otherwise $\mathbf{q}^{(j+1)}$ can be appended to the local path σ.

Four criteria are finally used to check if the local router should stop:

1) If the current distance $d^{(j+1)}$ to the goal configuration is smaller than a given threshold, the connection attempt is considered valid.
2) If an iteration produced a very small improvement in terms of distance $d^{(j)}$ to the target, the search is stopped, as a stationary point might have been reached.
3) As not only the distance to the goal is optimized, but also the linearized error, a new configuration might be located farther from the goal than the previous sample. Such configurations are rejected if the increment in distance exceeds the limit.
4) A limit of iterations is set as well, to prevent spending too much time on the connection of two samples.



(a) (b)

Fig. 4. Motions generated in $\mathcal{C}_{\mathrm{R}}$ after 10 iterations by QPMove (a) and a projection approach (b). Start configurations are represented as blue circles, while the goals as blue squares. The constrained Configuration Space is a sinusoidal wave. Linear interpolation cannot produce any valid sample, while our strategy is able to proceed toward the goal. The projection approach moves slowly due to the fact that the goal is found almost orthogonally to the constraint.

Fig. 4 contains an example of motion computed using the proposed router. Starting from the given sample, linear interpolation cannot produce any feasible motion toward the goal configuration due to the small relaxation. Instead, using the *QPMove* function it is possible to move inside $\mathcal{C}_{\mathrm{R}}$ by keeping $\mathbf{e}$ sufficiently small (Fig. 4(a)). A comparison can be made with respect to classical projection approaches, *e.g.*, CBiRRT. These planners usually perform motions by firstly stepping toward the goal and then re-projecting the sample on the manifold. In the case of Fig. 4(b) the goal is found in a direction that is almost perpendicular to the constraint curve at the start configuration. This is an unlucky condition for a projection approach, since a linearly interpolated sample is reprojected almost at the same initial position of the previous configuration. Despite being more expensive in terms of computation time, our approach usually explores $\mathcal{C}_{\mathrm{R}}$ faster than a projection technique, not blocking in these harder configurations.

B. Sampling-based Planner

The connection routine presented above can be easily inserted in classical Sampling-based planning algorithms. We decided during tests to use a bidirectional RRT [9], replacing the *extend* function with our custom router. In its standard version, the bidirectional RRT firstly picks a random configuration and then selects the nearest neighbor from the current tree to attempt a connection. The motion is however limited to a maximum length, which may prevent from reaching the random sample. In this study, it was considered a more aggressive approach, firstly proposed in [10], that does not limit the length of an extension step.

Regarding state sampling, we exploit the simplest uniform sampler that can extract any point from the Configuration Space of the system with equal probability. This is justified by the fact that the connection algorithm can produce new motions even when heading toward infeasible samples. In these cases, it will simply stop when the current branch cannot be further extended. However, since the motions are always constrained on the manifold, the new branches that are added to the trees could cover the relaxed CS non

uniformly. In the future, it will be worth investigating the use of better sampling techniques.

C. Post Processing

It is well-known that randomized planners usually find rather irregular paths which would result in jerky trajectories during execution. To smooth and shorten the solution path, a number of shortcuts is therefore attempted. The operation works by selecting two nodes at random from the final path and trying to connect them. We again exploit QPMove for this purpose, and the new local motion, if successfully found, replaces the existing one if it reduces the total length of the overall path.

Finally, as we handle constraints using a relaxation approach their violation will not be completely nullified along local paths. A further post-processing phase is thus performed to refine a solution by better enforcing the constraints at each intermediate sample. This operation is carried out by solving a number of QP instances formulated in a similar way as the ones considered before. The main difference is that regarding the objective the local goal is set to be the initial path sample. This allows to optimize the error without drastically change the path. It was noted, however, that in some cases this operation might invalidate one or more samples. As an example, if a path point was initially located really near to an obstacle, after some iterations the path could be moved in such a way that collision is not satisfied anymore. To deal with such problem, we try to re-optimize the initial path point by slightly changing the position of the local goal in the opposite direction in which collision or other non-differentiable constraints were invalid.

IV. EXPERIMENTAL RESULTS

We present in this Section some experimental results obtained using the proposed planner. The algorithm was tested to find motions for a dual-arm system featuring two Kuka LWR4 arms (Fig. 1), having 14 dof in total. The robot is required to move an object grasped with both hands from an initial configuration to a given one while avoiding obstacles. The bi-manual grasp adds six constraints to the system, due to the presence of a closed kinematic loop in the structure, which is described via (2) and (3).

The *Open Motion Planning Library* [11] was used to implement planning-related capabilities, while ROS [12] and *MoveIt!* [13] are used to send and execute planning requests. Obstacles are assumed to be static and with known pose, as perception of the environment is out of the scope of this study.

Different simulations were run to experimentally tune the parameters of QPMove. Regarding the choice of β, values ranging between 0.7 and 0.85 gave the best results. It was noted that high values of constraint weights in the objective increase the planning time, as less importance is given to the exploration of $\mathcal{C}_{\mathrm{R}}$. Motions get often stuck due to very small improvements or by reaching the maximum number of iterations. Concerning the choice of the relaxation factor, rather tolerant violations were allowed during the planning phase (around $1\,\mathrm{cm}$ for translation and 1° for rotation). Although these values would result inappropriate during execution, by properly tuning the weights for error minimization it is possible to achieve better results in practice. In addition, while attempting to smooth and simplify the path by shortcut operations higher weights can be used. As a result violations can be reduced even without the final error refinement operation to be less than $1\,\mathrm{mm}$ and $1\,\mathrm{mrad}$ in average.

In Fig. 5 some frames taken from a real experiment are shown. The robot is required to lift a metallic bar while avoiding some obstacles, two of them forming a narrow passage. Along the planned path the average violation is of $0.27\,\mathrm{mm}$ and 0.05° without running the final refinement process (only shortcuts were performed). By repeatedly attempt to solve the query, planning times were recorded. The average runtime is $4.168\,\mathrm{s}$ (excluding the post-processing phase), with a minimum of $0.931\,\mathrm{s}$. Standard relaxation techniques were tested for a comparison, and no feasible plan was found within $60\,\mathrm{s}$, using relaxation factors of $100\,\mathrm{mm}$ and 17.2°.

An example of error evolution during actual execution is shown in Fig. 6. It can be noted that violations increase significantly with respect to the values obtained after the planning phase, raising to a maximum of $5\,\mathrm{mm}$ in translation and 0.6° in rotation. This results from two factors: the absence of a control loop that regulates the error during the actual motion and a rough estimation of the dynamic parameters of the end-effectors. Depending on the final application these values may not be acceptable, and thus a regulating control action would be required to compensate them.

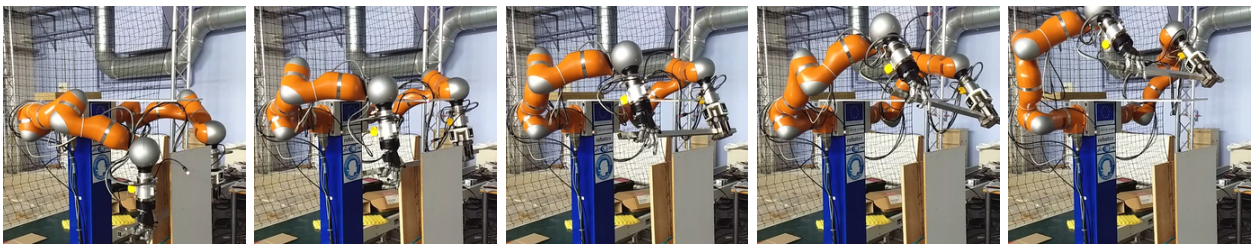


Fig. 5. Execution of a planned path, from initial configuration (left) to the final one (right). The full execution is included in the attached video.

V. CONCLUSIONS

In this work we proposed a new approach that exploits Quadratic Programming to reduce constraints violations in a relaxation-based context. Exploiting the Jacobian matrix of constraints functions allows to formulate the problem of moving toward a random sample as a local optimization of the distance to the configuration and the error associated with differentiable constraints.

As a result, paths of higher quality can be found in a shorter time, reducing the need of control actions at the execution level. The feasibility of our algorithm is confirmed by successful experiments on a real platform.

A drawback of this approach is the presence of a number of parameters which need to be tuned experimentally, and whose value can affect significantly the planning time.

A naive sampling technique was exploited to generate random configurations in the full Configuration Space of the robot. A better sampling approach able to uniformly cover the constraint manifold might lead to improvements in planning time and quality of the resulting path, and should therefore be investigated and integrated in the proposed approach.

APPENDIX

We show in the sequel the derivation of (3) starting from error's definition. The translation component is easily identified by considering that as $\mathbf{t}^\star$ is constant, it holds:

$$\dot{\mathbf{t}}_l + \dot{\overbrace{{}^0\mathbf{R}_l\,\mathbf{t}^\star}} - \dot{\mathbf{t}}_r = \mathbf{v}_l + \boldsymbol{\omega}_l \times \left({}^0\mathbf{R}_l\,\mathbf{t}^\star\right) - \mathbf{v}_r \qquad (12)$$

By definition, the linear/angular velocities $\mathbf{v}_l$, $\mathbf{v}_r$ and $\boldsymbol{\omega}_l$ are linked with joint vector's derivatives by robot's Jacobian matrices $\mathbf{J}_l$ and $\mathbf{J}_r$, thus leading to the first part of the error Jacobian.

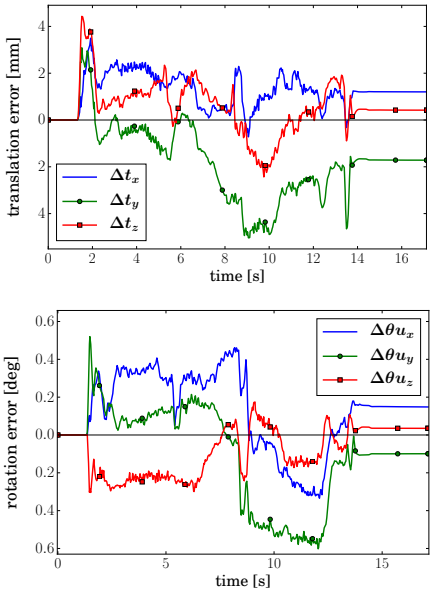


Fig. 6. Evolution of constraint errors during real execution. Translation and rotation components are reported respectively in the top and bottom pictures.

Regarding rotation, one needs to evaluate $\dot{\overbrace{\theta\mathbf{u}}}$. The derivative of the angle-axis representation of a rotation matrix ${}^a\mathbf{R}_b$ is in general given by $\mathbf{B}\,{}^a\boldsymbol{\omega}_{b/a}$, with ${}^a\boldsymbol{\omega}_{b/a}$ representing the angular velocity vector of the generic frame b with respect to a (expressed in frame a). The form of matrix $\mathbf{B}$ was already given in (5). In the considered case the rotation matrix for which the decomposition is evaluated is ${}^0\mathbf{R}_r^T\,{}^0\mathbf{R}_l\,{}^l\mathbf{R}_r^\star$. It is possible to consider a virtual frame $\mathcal{F}_{r^\star}$ rigidly attached to the origin of the left end-effector, characterized by the rotation ${}^l\mathbf{R}_{r^\star} = {}^l\mathbf{R}_r^\star$. In this sense ${}^0\mathbf{R}_r^T\,{}^0\mathbf{R}_l\,{}^l\mathbf{R}_r^\star$ represents the rotation of the virtual frame $\mathcal{F}_{r^\star}$ with respect to $\mathcal{F}_r$, and therefore:

$$\dot{\overbrace{\theta\mathbf{u}}} = \mathbf{B}\,{}^r\boldsymbol{\omega}_{r^\star/r} = \mathbf{B}\,{}^0\mathbf{R}_r^T\left(\boldsymbol{\omega}_{r^\star} - \boldsymbol{\omega}_r\right) \qquad (13)$$

In addition, as $\mathcal{F}_{r^\star}$ and $\mathcal{F}_l$ are rigidly linked to each other, $\boldsymbol{\omega}_{r^\star} = \boldsymbol{\omega}_l$. Substituting $\boldsymbol{\omega}_l = \mathbf{J}_l^{(\omega)}\dot{\mathbf{q}}_l$ and $\boldsymbol{\omega}_r = \mathbf{J}_r^{(\omega)}\dot{\mathbf{q}}_r$ bears to the final relation.

ACKNOWLEDGMENT

This work was carried out in the framework of the PROMPT project, a project funded by RFI Atlanstic 2020. Parts of the equipment used here were funded by the project ROBOTEX, reference ANR-10-EQPX-44-01.

REFERENCES

[1] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[2] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planing," Deptartment of Computer Science, Iowa State University, Tech. Rep., 1998.

[3] J. Cortes and T. Simeon, "Sampling-based motion planning under kinematic loop-closure constraints," in *Algorithmic Foundations of Robotics VI*. Springer, 2004, pp. 75–90.

[4] D. Berenson, S. S. Srinivasa, D. Ferguson, and J. J. Kuffner, "Manipulation planning on constraint manifolds," in *IEEE Int. Conf. on Robotics and Automation*. IEEE, 2009, pp. 625–632.

[5] L. Jaillet and J. M. Porta, "Path planning under kinematic constraints by rapidly exploring manifolds," *IEEE Trans. on Robotics*, vol. 29, no. 1, pp. 105–117, 2013.

[6] C. Voss, M. Moll, and L. E. Kavraki, "Atlas+ x: Sampling-based planners on constraint manifolds," Rice University, Tech. Rep., 2017.

[7] M. Bonilla, E. Farnioli, L. Pallottino, and A. Bicchi, "Sample-based motion planning for soft robot manipulators under task constraints," in *IEEE Int. Conf. on Robotics and Automation*. IEEE, 2015, pp. 2522–2527.

[8] M. Bonilla, L. Pallottino, and A. Bicchi, "Noninteracting constrained motion planning and control for robot manipulators," in *IEEE Int. Conf. on Robotics and Automation*. IEEE, 2017, pp. 4038–4043.

[9] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *IEEE Int. Conf. on Robotics and Automation*, vol. 2. IEEE, 2000.

[10] S. M. LaValle and J. J. Kuffner Jr, "Rapidly-exploring random trees: Progress and prospects," 2000.

[11] I. A. Sucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012, http://ompl.kavrakilab.org (hit on 2017-06-30).

[12] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2, 2009, p. 5.

[13] I. A. Sucan and S. Chitta. Moveit! [Online]. Available: http://moveit.ros.org