Improving Relaxation-based Constrained Path Planning via Quadratic Programming

Franco Fusco, Olivier Kermorgant, Philippe Martinet

Improving Relaxation-based Constrained Path Planning via Quadratic Programming

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

[1] Centrale Nantes, Laboratoire des Sciences du Numérique de Nantes LS2N, France
[2] Inria Sophia Antipolis, France

Abstract. Many robotics tasks involve a set of constraints that limit the valid configurations the system can assume. Some of these constraints, such as loop-closure or orientation constraints to name some, can be described by a set of implicit functions which cause the valid Configuration Space of the robot to collapse to a lower-dimensional manifold. Sampling-based planners, which have been extensively studied in the last two decades, need some adaptations to work in this context.
A proposed approach, known as *relaxation*, introduces constraint violation tolerances, thus approximating the manifold with a non-zero measure set. The problem can then be solved using classical approaches from the randomized planning literature. The relaxation needs however to be sufficiently high to allow planners to work in a reasonable amount of time, and violations are counterbalanced by controllers during actual motion.
We present in this paper a new component for relaxation-based path planning under differentiable constraints. It exploits Quadratic Optimization to simultaneously move towards new samples and keep close to the constraint manifold. By properly guiding the exploration, both running time and constraint violation are substantially reduced.

1 Introduction

Sampling-based planning techniques have been successfully exploited to solve a number of problems involving a wide variety of systems, such as mobile robots and manipulators. They rely on the construction of a graph, either in the form of networks [7] or trees [10], trying to approximate the valid Configuration Space (CS) of a system. Nodes, corresponding to configurations, are generated randomly and validated in a further step by checking for collision. In many situations the sampling process and the construction of edges – which represent motions between pairs of configurations – require simple operations. Random samples can be obtained by drawing each component independently from a given random distribution, while local motions are often created using linear interpolation.

However, many robotics tasks impose a number of constraints on the system. As an example, a domestic robot carrying a tray loaded with objects should ensure that during its motion the platter remains horizontal. These constraints cause the valid configuration space to reduce to a lower-dimensional manifold implicitly defined by constraint equations. This introduces many challenges in

the planning problem, mainly due to the fact that classical samplers and local routers cannot be exploited any more.

In order to deal with this added complexity, new tools have been developed to generate and connect samples inside the constraint manifold. Early studies focused on closed kinematic chains [17] and used the Gradient Descent algorithm to enforce loop-closure on random invalid samples. Cortes *et al.* introduced the *Random-Loop Generator* [4], a sampling technique designed to produce valid samples for closed-loop mechanisms.

Planning under task-space constraints was investigated in [13], which introduces *Tangent Space Sampling* and *First-Order Retraction* in order to sample feasible joint configurations. The *CBiRRT* (Constrained Bi-directional Rapidly-exploring Random Tree) planner [1] uses the Jacobian pseudo-inverse in a similar way to the Tangent Space Sampling in order to project an infeasible sample on the valid manifold defined by end-effector pose constraints.

Further methods have been designed to explore an approximation of the constraint manifold, based on high-dimensional continuation. *AtlasRRT* [5] focuses on the joint construction of Atlases and of a bi-directional RRT to approximate and explore the constraint manifold. A similar strategy is exploited in [8], defining a set of tangent spaces that locally approximate the manifold. A generalized framework based on Atlases is proposed in [16]. It extends several existing randomized planners to the exploration of constrained manifolds.

Other recent works [2,3] use the concept of *relaxation*, consisting in allowing a small constraint violation during planning. This technique has been mainly exploited to plan motions for compliant systems, using a control action during trajectory execution to steer robot's state close to the constraint manifold. The planning phase is therefore solved via standard techniques from the sampling based domain, since the valid CS is no longer a zero-measure set.

Such approach requires a trade-off between the quality of a planned path and planning time. In fact, if the allowed constraint violation is too small, the topology of the valid Configuration Space changes to a set of extremely narrow passages and the planning time increases significantly. Furthermore, the technique is highly dependent on system's compliance and on controller's ability to reconfigure a robot in feasible states during the motion. Thus, many practical scenarios involving rigid robots cannot exploit relaxation, since they would require a higher quality path directly from the planning step.

In this paper we propose a new approach inspired by relaxation techniques that allows to generate in a short amount of time paths with lower constraint violation. The algorithm uses Quadratic Programming (QP) to locally perform motions in the relaxed CS, with the objective of keeping configurations close to the original manifold while extending to a random sample. We exploit the Jacobian matrix of the constraints to locally linearize the manifold and guide the exploration towards valid states. Thanks to smaller violation tolerances, the necessity of a control action during execution can be reduced, allowing a broader range of robots to exploit these techniques.

The remainder of this paper is organized as follows: differentiable constraints and the concept of relaxed Configuration Space are introduced in Section 2.1, while in Section 2.2 the technique used to perform local motions inside the relaxed CS is detailed. The router is integrated in a randomized planner, which is presented in Section 2.3. Experiments have been conducted in simulation considering different setups, to demonstrate the generality of our approach. We show in Section 3 that the technique allows to rapidly find paths featuring small constraint violation.

2 Planning Algorithm

In this section we present the proposed planner based on QP to enforce a set of constraints while planning. In order to connect two configurations, the local router is asked to accomplish two tasks concurrently: drive the system toward a given sample and keep the error associated with constraints as small as possible.

We formulate such problem as a Sequential Quadratic Optimization, wherein each step aims at changing the current configuration to a sample that is nearer to the target one, while enforcing the constraints. In addition, the displacement of each coordinate of the configuration vector is bounded. This allows to obtain a discrete set of intermediate configurations, and to consistently check for collisions along the path.

We introduce the constraints in Section 2.1, while the local path planner is detailed in Section 2.2. The connection routine is integrated in a complete Sampling-Based Planner based on the RRT-Connect algorithm [9], which is presented in Section 2.3. It attempts to generate new samples and to connect them to the existing graph using the local planner. Post-processing operations can finally be performed to enhance the quality of the resulting path, if any is found.

2.1 Differentiable Constraints

We consider the case of an n-dimensional Configuration Space $\mathcal{C} \subset \mathbb{R}^n$, the configuration vector being denoted as $\mathbf{q} = \begin{bmatrix} q_1 & \cdots & q_n \end{bmatrix}^T$. Each coordinate is assumed to be bounded in a range $[q_{i,\min}\ ,\ q_{i,\max}]$. A set of n_c constraints is also considered, each one being described by a differentiable function $C_i : \mathcal{C} \to \mathbb{R}$ $(i = 1, \cdots, n_\mathrm{c})$. Altogether, they define the constrained Configuration Space as:

$$\mathcal{C}_\mathrm{C} = \{\mathbf{q} \in \mathcal{C} : C_i(\mathbf{q}) = 0 \quad \forall i\} \tag{1}$$

To approximate the valid set defined by all constraints we introduce n_c constants $\varepsilon_i > 0$, which quantify the maximum allowed violation of each constraint at a given configuration $\mathbf{q}$. These constants need to be set manually by the user depending on the required quality of the planned path. We then define the relaxed Configuration Space $\mathcal{C}_\mathrm{R}$:

$$\mathcal{C}_\mathrm{R} = \{\mathbf{q} \in \mathcal{C} : -\varepsilon_i \leq C_i(\mathbf{q}) \leq \varepsilon_i \quad \forall i\} \tag{2}$$

In order to shorten the notation in following sections, we stack all constraints in the n_c-dimensional vector $\mathbf{e} = \left[C_1(\mathbf{q}) \cdots C_{n_c}(\mathbf{q})\right]^T$. We finally recall that the constraints are assumed to be differentiable. Under this assumption, we introduce the Jacobian matrix $\mathbf{J}_e = \frac{\partial \mathbf{e}}{\partial \mathbf{q}}$, whose i-th row is given by the gradient of the constraint C_i.

The original planning problem is then formulated as finding a discrete sequence of points in $\mathcal{C}_R$ which approximate a valid path in $\mathcal{C}_C$. The exploration, as detailed in the following section, is guided by the use of $\mathbf{J}_e$ to locally approximate the manifold implicitly defined by the constraints.

2.2 Local Motions using Quadratic Programming

In order to evaluate a local path between two configurations, we solve sequentially a number of QP problems. Our iterative algorithm exploits a single step to perform a short motion towards the goal while keeping $\mathbf{e}$ as close to zero as possible. The problem is formalized as follows: two samples $\mathbf{q}_a \in \mathcal{C}_R$ (initial configuration) and $\mathbf{q}_b \in \mathcal{C}$ (final state) are considered. During the process, we denote with $\mathbf{q}^{(j)}$ the configuration obtained after the j-th iteration (such that $\mathbf{q}^{(0)} = \mathbf{q}_a$). We then select a proper value of $\mathbf{q}^{(j+1)}$, *i.e.*, the next way-point of the local path from $\mathbf{q}_a$ to $\mathbf{q}_b$, by solving the quadratic optimization problem

$$\mathbf{q}^{(j+1)} = \underset{\mathbf{x} \in \mathcal{C}}{\arg\min} \left\| \mathbf{Q}^{(j)}\, \mathbf{x} - \mathbf{v}^{(j)} \right\|^2 \quad \left(\mathbf{Q}^{(j)} \in \mathbb{R}^{m \times n},\ \mathbf{v}^{(j)} \in \mathbb{R}^m \right) \tag{3}$$

subject to a set of p linear inequalities in the form:

$$\mathbf{A}^{(j)}\, \mathbf{x} \leq \mathbf{b}^{(j)} \quad \left(\mathbf{A}^{(j)} \in \mathbb{R}^{p \times n},\ \mathbf{b}^{(j)} \in \mathbb{R}^p \right) \tag{4}$$

We use the superscript (j) to underline that $\mathbf{Q}$, $\mathbf{v}$, $\mathbf{A}$ and $\mathbf{b}$ are evaluated using only values coming from the previous iteration, thus being constant quantities during the $(j+1)$-th step.

In the sequel, we firstly show how to derive the expression of matrices contained in (3) and (4), and afterwards the iterative scheme is detailed.

Objective and Linear Inequalities. Since the final goal of the iterative scheme is to reach $\mathbf{q}_b$ – or at least to move as close as possible to it – the considered objective function should contain a term that reaches its minimum in correspondence of the given configuration. In addition, a further contribution should be considered to enforce the constraints. A candidate function satisfying both requirements could be:

$$f\left(\mathbf{q}^{(j+1)}\right) = \left\| \mathbf{q}^{(j+1)} - \mathbf{q}_b \right\|^2 + \left\| \alpha\, \mathbf{e}^{(j+1)} \right\|^2 \tag{5}$$

wherein the constant parameter $\alpha \in \mathbb{R}^{n_c \times n_c}$ is a positive definite diagonal matrix used as a weighting factor to modulate the "priority" associated to the second

task. The term $\mathbf{e}^{(j+1)}$, corresponding to constraints violation evaluated at $\mathbf{q}^{(j+1)}$, is however generally non-linear in the configuration vector, and does not fit the quadratic formulation of (3). However, the function can be linearized around $\mathbf{q}^{(i)}$, using the Jacobian matrix $\mathbf{J}_\mathrm{e}$ introduced in Section 2.1:

$$\mathbf{e}^{(j+1)} - \mathbf{e}^{(j)} \simeq \mathbf{J}_\mathrm{e}^{(j)} \left(\mathbf{q}^{(j+1)} - \mathbf{q}^{(j)} \right) \tag{6}$$

After injecting the linearized error in (5), the objective can be re-written in a matrix form compatible with (3), thus obtaining:

$$\mathbf{Q}^{(j)} = \begin{bmatrix} \mathbf{I}_n \\ \alpha\, \mathbf{J}_\mathrm{e}^{(j)} \end{bmatrix} \qquad \mathbf{v}^{(j)} = \begin{bmatrix} \mathbf{q}_b \\ \alpha \left(\mathbf{J}_\mathrm{e}^{(j)}\, \mathbf{q}^{(j)} - \mathbf{e}^{(j)} \right) \end{bmatrix} \tag{7}$$

Regarding the set of linear inequalities, we choose to reduce the search interval according to two factors. The first one imposes upper and lower bounds to each component of the configuration vector:

$$\mathbf{q}_{min} \leq \mathbf{q}^{(j+1)} \leq \mathbf{q}_{max} \tag{8}$$

The second set of inequalities that is considered limits the local motion of each coordinate to a symmetric interval centered in $\mathbf{q}^{(j)}$:

$$\mathbf{q}^{(j)} - \beta^k \Delta\mathbf{q} \leq \mathbf{q}^{(j+1)} \leq \mathbf{q}^{(j)} + \beta^k \Delta\mathbf{q} \quad (\beta \in (0,1) \text{ and } k \in \mathbb{N}) \tag{9}$$

In this relation, the entries of $\Delta\mathbf{q} \in \mathbb{R}^n$, all being positive, correspond to the allowed step of each coordinate, while β^k is used to tune the size of the allowed interval. This constraint is justified by two reasons. On one hand the sequential optimization relies on the linearized error dynamics. The approximation must be kept consistent, and therefore the configurations $\mathbf{q}^{(i)}$ and $\mathbf{q}^{(i+1)}$ should not differ too much. If the new configuration is too far away from the initial one, the constraint error could exceed the tolerance, even if the linearized one is null, as depicted in Fig. 1. The use of the coefficient β^k allows to resize the step size during the optimization, and its use is detailed in next section. On the other hand, in many practical planning problems some non-differentiable constraints could be considered as well. They should be verified at each iteration, and a large step size could bear the system to "jump" over small invalid regions. As an example, in our approach we perform discrete collision checking, by verifying the validity of a configuration at each iteration. With the set of inequalities (9) we try to avoid situations wherein collisions with small obstacles are not detected.

Sequential Optimization The overall local motion routine, called *QPMove*, is reported in algorithm 1. It performs a Sequential Optimization by solving at each step a QP instance as formulated in the previous section.

An iteration starts with the evaluation of the matrices defining the objective and the inequalities. Then, the inner cycle (between lines 8 and 16) is executed to solve the current QP instance. The obtained solution is tested by checking

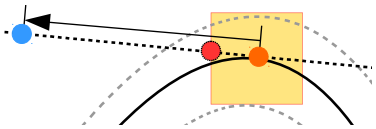

Fig. 1. From an initial configuration (orange) lying on the constraint – the black continuous line – the optimization would move the system to an invalid configuration (in blue). If the motion is limited to the small yellow rectangle, there are higher chances to still fall inside the relaxed region (surrounded by the dashed gray lines).

constraints violation at the new configuration. This is a fundamental step: since constraints are linearized during the procedure, a resulting sample will only ensure an approximation of the error to be optimized. On the other hand, a component of the actual error could fall outside its valid range $[-\varepsilon_i, \varepsilon_i]$. To better enforce bounding constraints when such situation occurs, k is incremented by one unit and the optimization step is repeated. This shrinks the valid range of motion, possibly leading $\mathbf{q}^{(j+1)}$ to lie inside $\mathcal{C}_{\mathrm{R}}$. Nonetheless, the procedure is limited up to a maximum value of k, in order to prevent the algorithm from spending too much time on some critical samples.

Algorithm 1 QP-based Motion Validator (QPMove)

QPMove($\mathbf{q}_a$, $\mathbf{q}_b$) :
$\mathbf{q}^{(0)} \leftarrow \mathbf{q}_a$
$f^{(0)} \leftarrow +\infty$
$f^{(1)} \leftarrow +\infty$
for j=0 **to** $j_{\max}$ **do**
  $k \leftarrow 0$
  $\mathbf{Q}^{(j)}$, $\mathbf{v}^{(j)}$, $\mathbf{A}^{(j)}$ $\leftarrow$ INIT_QP_ITERATION($\mathbf{q}^{(j)}$, $\mathbf{q}_b$)
  do
    if $k > k_{\max}$ **then**
      return "failure", $\mathbf{q}(j)$
    end if
    $\mathbf{b}^{(j)} \leftarrow$ GET_QP_B_VECTOR($\mathbf{q}^{(j)}$, k)
    $\mathbf{q}^{(j+1)}$, $f^{(j+1)} \leftarrow$ SOLVE_QP_INSTANCE($\mathbf{Q}^{(j)}$, $\mathbf{v}^{(j)}$, $\mathbf{A}^{(j)}$, $\mathbf{b}^{(j)}$)
    $\mathbf{e}^{(j+1)} \leftarrow$ EVALUATE_ERROR($\mathbf{q}^{(j+1)}$)
    $k \leftarrow k + 1$
  while not$(-\boldsymbol{\varepsilon} \leq \mathbf{e}^{(j+1)} \leq \boldsymbol{\varepsilon})$
  if IN_COLLISION($\mathbf{q}^{(j+1)}$) **then**
    return "failure", $\mathbf{q}(j)$
  end if
  if $f^{(j+1)} \leq f_{min}$ **then**
    return "success", $\mathbf{q}_b$
  end if
  if $f^{(j+1)} \leq f^{(j)}$ **and** $f^{(j+1)} \geq f^{(j)} - \Delta f^{(-)}$ **then**
    return "failure", $\mathbf{q}(j+1)$
  end if
  $\mathbf{q}^{(j)} \leftarrow \mathbf{q}^{(j+1)}$
  $f^{(j)} \leftarrow f^{(j+1)}$
end for
return "failure", $\mathbf{q}(j_{\max})$

The remaining part of the main loop is instead used to check the progresses done between two iterations. As briefly mentioned in the previous section, we propose to handle non-differentiable constraints – in particular, collision detection – by checking them at each new sample. Thus, after having solved the current QP instance, the validity of the sample is tested. If any violation is detected, the algorithm stops and returns the last validated sample.

A second criterion verifies instead if the objective value $f^{(j+1)}$ has become small enough, and in case the algorithm is stopped since the goal configuration has been reached with the error being sufficiently small.

A known problem of quadratic minimization is the existence of local minima. To detect stationary points, the algorithm computes the difference between the objective after subsequent iterations: if the improvement is below a given threshold $\Delta f^{(-)}$, the algorithm returns with the status "failure". It must be noted that due to the linearization the objective could get worse between successive iterations, and thus local minima are checked only if the objective is improving.

Since a proper tuning of the objective thresholds could be hard in practice, a maximum number of iterations is exploited as the last strategy to prevent the routine to waste too much computation time.

The proposed motion component requires more computation efforts and longer run time compared to the simpler technique of linear interpolation, usually exploited in the relaxation context. Nonetheless, its ability of following constraint manifold's curvature proved to effectively counter-balance the drawbacks. Fig. 2 shows some comparative examples in a 2-Dimensional Configuration Space. In the first case (Fig. 2(a)) the valid set defined by the constraint is a circle. Even with a high relaxation factor, linear interpolation would fail to connect the two shown samples, since it would try to follow a straight line path that is incompatible with the curved constraint. In practical scenarios, the relaxation would be way smaller, making it even harder to connect configurations. The example depicted in Fig. 2(b) shows another feature of QPMove: even when attempting to reach an invalid sample, a valid motion can still be performed inside $\mathcal{C}_{\mathrm{R}}$. We also show in Fig. 2(c) the motion that would be produced by a projection approach like the one exploited in the CBiRRT planner. To reach the goal more samples are necessary, with a more irregular spacing between adjacent samples.

2.3 QPlan

The component QPMove described in the previous section was used as part of a complete path-planner named *QPlan*. The algorithm exploits a bidirectional RRT [9] to explore the constraint manifold, and runs post-processing techniques to enhance the quality of a path. We detail in the following how the typical components of a randomized planner were integrated.

Sampling. A State Sampler produces new configurations which could potentially become leaves of the trees being expanded by the planner. When moving on a manifold, samples should satisfy the constraints imposed on the system.

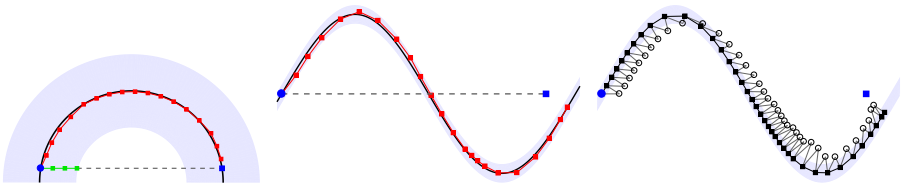

Fig. 2. Motions generated in $\mathcal{C}_{\mathrm{R}}$ by QPMove (red), linear interpolation (green) and a projection approach (black). Start configurations are represented as blue circles, while the goals as blue squares. In (a), the Constrained Configuration Space is represented by a circular arc, while in (b) and (c) by a sinusoidal wave. Linear interpolation can only produce a short path in the first case thanks to a large relaxation, while no new samples can be generated in the sinusoidal region. QPMove (b) produces less samples, which are better distributed than in the case of a projection approach (c).

However, QPMove does not necessarily need valid samples to extend one tree. In fact, an infeasible goal could be passed as $\mathbf{q}_b$. The algorithm would not converge to the given configuration, but it could still produce valid motions toward new points in the Constraint Manifold. The advantage of this choice is a faster sampler, since the simple uniform sampling technique can be used.

Trees Extension. The extension step of a bidirectional RRT works by selecting a random configuration and its nearest neighbor from the current tree. A motion is then attempted from the latter to the random sample, but limiting its length to a maximum value.

We rather adopt the greedy version of this algorithm, sometimes referred to as *RRT-ConnConn* [11]. This variant tries to extend a branch until either an invalid state is reached or the connection is successfully performed.

Since QPlan is able to find a discrete set of way-points, it could be useful to insert all intermediate configurations in the tree. However, this could bear to an over-populated tree. Therefore, only one generated configuration out of N is inserted in the tree, and only if its distance from the previously added sample is higher than a given threshold.

Post-processing. As we handle constraints using a relaxation approach, their violation will not be completely nullified along local paths. Post-processing is used to refine an obtained solution by enforcing the constraints at each intermediate sample. If a path is found by the planner, the sequence of joint way-points is re-built, and a local QP optimization is run on each sample. The procedure is similar to the one exploited in QPMove, with two differences: rather than setting inequalities that enforce space bounds, a much smaller range is considered. In practice, inequalities (8) are replaced by:

$$\mathbf{q}_{raw} - \mathbf{dq} \leq \mathbf{q}^{(j+1)} \leq \mathbf{q}_{raw} + \mathbf{dq} \tag{10}$$

where $\mathbf{q}_{raw}$ is the path configuration before optimization, and $\mathbf{dq}$ is a vector containing small values allowing the error to be minimized without moving too far away from $\mathbf{q}_{raw}$. In addition, we set as goal configuration the raw initial sample itself. It further constrain the system to only locally change its state.

In addition, a certain number of short-cutting attempts is performed, in order to reduce path's length and simplify the solution.

3 Simulation Results

We present in this section results obtained from simulations on several different setups with the implemented algorithm. The planner has been implemented using the components provided by OMPL, the *Open Motion Planning Library* [15], and integrated in the ROS [12] framework in the form of a MoveIt! [14] planning plugin.

During all experiments, the selection of a proper value of the parameter β was done by trial and error. A value between 0.7 and 0.85 gave the best results in practice. Relaxation constants ε_i were set to ensure a reasonably small violation of constraints. Their numerical value is reported later for each experiment.

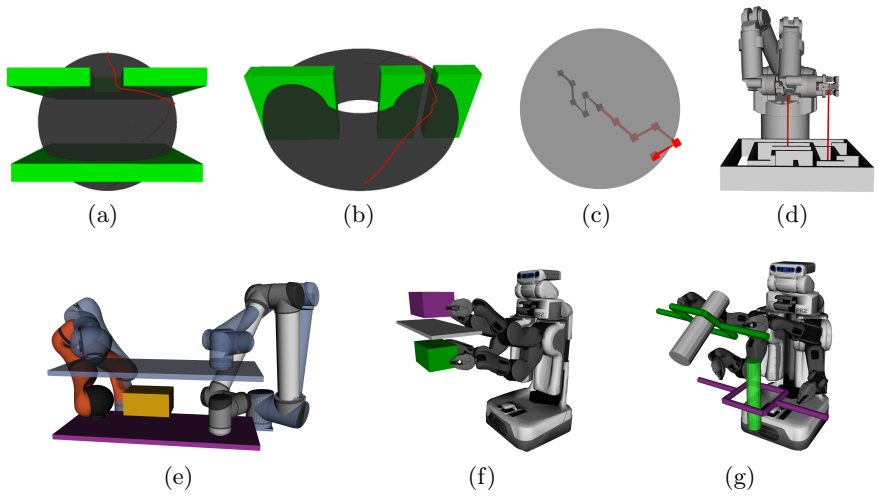

Fig. 3. Different tests with QPlan: a 3D point moving respectively (a) on a sphere and (b) on a torus, (c) a five-links chain whose tip is constrained to a sphere, (d) a Barrett arm solving a maze, (e) a UR10 and a Kuka LWR4 moving a plate, (f) PR2 moving a box, (g) PR2 displacing an object with a hole.

A first set of tests was performed considering a 3-Dimensional point (x, y, z) constrained to the surface of a sphere (Fig. 3(a)) and of a torus (Fig. 3(b)),

in presence of obstacles forming narrow passages. The constraints are written in the two setups as $C(x, y, z) = x^2 + y^2 + z^2 - R^2$ and $C(x, y, z) = \left(\sqrt{x^2 + y^2} - r_1\right)^2 + z^2 - r_2^2$, R being the radius of the sphere and r_1, r_2 the radii defining the torus. We compared the performances of QPlan with a classical relaxation technique, considering different tolerances. Although the algorithm presented in [2] uses RTT* [6], we implemented it using the RRT-Connect algorithm for a meaningful comparison of run times. We exploited a sampler that uniformly generates samples over the manifold in order to reproduce the proposed setup. As reported in Table 1, our approach is considerably faster than a simple relaxation technique when the allowed tolerance becomes small[3].

Some tests were run on a more complex test-case, which had been proposed in [16]. A five-links kinematic chain rooted at the origin is considered (see Fig. 3(c)). Each link is parametrized by a 3D point, for a total number of 15 degrees of freedom. Five constraints are set to fix the distance between pairs of adjacent joints, while a further constraint requires chain's tip to move on a spherical surface. An additional constraint can be set to fix the vertical coordinate of the first point. While QPlan can effectively find paths for this system, standard relaxation techniques require either a higher tolerance or longer execution time. The results shown in Table 2 were obtained considering all links having a length of 0.2 m, and the constraint sphere a radius of 0.6 m. The tolerances have been fixed to $0.005\,\mathrm{m}^2$ for the five distance constraints, to $0.025\,\mathrm{m}^2$ for the tip and 1 mm for the vertical constraint. It is also worth to note that the run-time is at the same order of magnitude of the Atlas-based planners, according to the results given in [16].

Table 1. Average planning time in seconds over 100 runs of sphere and torus setups. We used $R = 1\,\mathrm{m}$, $r_1 = 1\,\mathrm{m}$, $r_2 = 0.5\,\mathrm{m}$. The relaxation factor ε is given in m^2.

Algorithm	ε	Sphere	Torus
Relaxation	10^{-2}	0.214	0.477
	10^{-3}	8.785	14.632
QPlan	10^{-2}	0.104	0.027
	10^{-3}	0.120	0.218

Table 2. Results with the five-links chain, under 6 or 7 constraints. The actual run-time of the first algorithm is unknown, since no plans were found before a time limit of 10 s.

Algorithm	$n_c = 6$	$n_c = 7$
Relaxation	> 10	> 10
QPlan	0.080	0.087

Other tested scenarios involve a Barrett arm solving a maze (Fig. 3(d)) and some dual arm systems cooperatively displacing an object (Figures 3(e), 3(f) and 3(g)). We do not report running times obtained with standard relaxation

[3] A relaxation factor $\varepsilon = 10^{-3}\,\mathrm{m}^2$ on a sphere with unitary radius corresponds to constrain the points to lay at most 0.5 mm from the surface. The same factor gives a maximum distance of 1mm from a torus having $r_1 = 1\,\mathrm{m}$, $r_2 = 0.5\,\mathrm{m}$.

techniques, since they require a much higher planning time for equal relaxation factors.

During the maze-solving test three constraints are applied to the end-effector so as to keep the stick grasped by the arm vertical and in contact with labyrinth's floor. This scenario is particularly challenging for the planner, since many obstacles are encountered during planning. As explained in Section 2.3, we do not use a specific Sampler to generate new configurations. This might lead the robot to attempt many motions toward configurations that are completely unreachable either due to constraints or to obstacles. Using a more involved sampling technique may instead improve the performances. Another factor that greatly influences the planning time is the choice of the relaxation constants. In a first instance, we considered very strict tolerances on both stick position and orientation: 1 cm of error for the altitude and 0.05 rad for roll and pitch constraints. The chances to find a solution in a short amount of time are quite low, as shown in Table 3. However, further tests were conducted by allowing orientation constraints to be violated with at most 0.2 rad, and optimizing them in the post-processing phase. With this more tolerant setup, the planning time is slightly reduced.

In the dual-arm setups, relative translation and rotation of the end-effectors are forbidden in order to maintain a fixed relative transformation between tip frames. The challenge here comes from the high number of both Degrees of Freedom and constraints.

Table 3. Planning time of the tests involving different manipulators.

Scene	Avg.	Min.	Max.
Barrett maze (0.05 rad)	4.034	0.669	13.828
Barrett maze (0.2 rad)	3.446	0.426	13.162
UR10 + LWR4	0.819	0.084	4.085
PR2 box	4.338	1.036	10.395
PR2 cylinders	0.866	0.202	2.322

4 Conclusions

We have presented a new approach based on Quadratic Programming that can effectively generate paths while dealing with a set of constraints. Our approach takes advantage of relaxation to enlarge the range of valid configurations while planning and of analytic description of constraints to guide local motions. As a drawback, the proposed algorithm features many parameters which need to be tuned in order to guarantee good performances. The relaxation factor plays a relevant role: if constraint violations are not too strict, the search can proceed faster. Nonetheless, with an increased tolerance robots would need higher control action to re-project the samples back to the manifold.

Finally, in our contribution we focused only on the planning step, verifying that paths generated by the algorithm can be found quickly and with better

constraint enforcement. As a future line of work, experiments with a real robot should be performed to confirm the validity of the proposed approach.

References

1. Dimitry Berenson, Siddhartha S. Srinivasa, Dave Ferguson, and James J. Kuffner. Manipulation planning on constraint manifolds. In *IEEE Int. Conf. on Robotics and Automation*. IEEE, 2009.
2. Manuel Bonilla, Edoardo Farnioli, Lucia Pallottino, and Antonio Bicchi. Sample-based motion planning for soft robot manipulators under task constraints. In *IEEE Int. Conf. on Robotics and Automation*, 2015.
3. Manuel Bonilla, Lucia Pallottino, and Antonio Bicchi. Noninteracting constrained motion planning and control for robot manipulators. In *IEEE Int. Conf. on Robotics and Automation*. IEEE, 2017.
4. Juan Cortes and Thierry Simeon. Sampling-based motion planning under kinematic loop-closure constraints. In *Algorithmic Foundations of Robotics VI*. Springer, 2004.
5. Léonard Jaillet and Josep M Porta. Path planning under kinematic constraints by rapidly exploring manifolds. *IEEE Trans. on Robotics*, 2013.
6. Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. Journal of Robotics Research*, 2011.
7. Lydia E. Kavraki, Petr Svestka, J-C Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation*, 1996.
8. Beobkyoon Kim, Terry Taewoong Um, Chansu Suh, and Frank C. Park. Tangent bundle rrt: A randomized algorithm for constrained motion planning. *Robotica*, 2016.
9. James J. Kuffner and Steven M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE Int. Conf. on Robotics and Automation*. IEEE, 2000.
10. Steven M. LaValle. Rapidly-exploring random trees: A new tool for path planing. Technical report, Deptartment of Computer Science, Iowa State University, 1998.
11. Steven M. LaValle and James J. Kuffner Jr. Rapidly-exploring random trees: Progress and prospects, 2000.
12. Maorgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, 2009.
13. Mike Stilman. Task constrained motion planning in robot joint space. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*. IEEE, 2007.
14. Ioan A. Sucan and Sachin Chitta. Moveit! `http://moveit.ros.org`. Accessed: 2017-06-30.
15. Ioan A. Sucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robot. Autom. Mag.*, 2012. `http://ompl.kavrakilab.org` Accessed: 2017-06-30.
16. Caleb Voss, Mark Moll, and Lydia E. Kavraki. Atlas+ x: Sampling-based planners on constraint manifolds. Technical report, Rice University, 2017.
17. Jeffery Howard Yakey, Steven M. LaValle, and Lydia E. Kavraki. Randomized path planning for linkages with closed kinematic chains. *IEEE Trans. on Robotics and Automation*, 2001.