A simple path optimization method for motion planning

Mylène Campana, Florent Lamiraux, Jean-Paul Laumond

A simple path optimization method for motion planning

M. Campana[1,2], F. Lamiraux[1,2] and J.-P. Laumond[1,2]

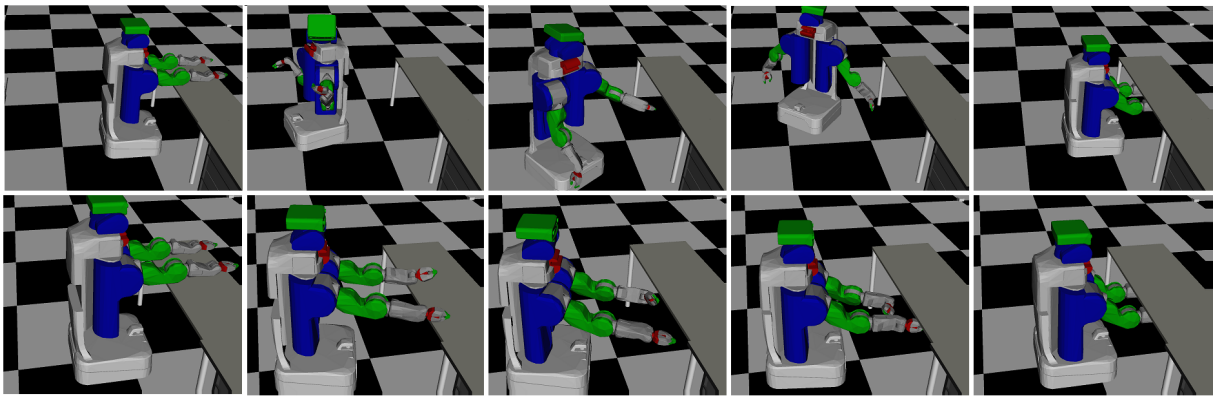

Fig. 1: On top, the initial path from Visibility-PRM planner for a PR2 robot in a kitchen environment. On bottom, the optimized path found with our algorithm. The cost of the initial path was 15.5, our optimizer reduced it to 3.0 in 28 iterations. Random shortcut has only downed it to 5.6 after 96 tries.

***Abstract*— Most algorithms in probabilistic sampling-based path planning compute collision-free paths made of straight line segments lying in the configuration space. Due to the randomness of sampling, the paths make detours that need to be optimized. The contribution of this paper is to propose a gradient-based algorithm that transforms a polygonal collision-free path into a shorter one, while both:**

- **requiring only collision checking, and not any time-consuming obstacle distance computation, and**
- **constraining only part of the configuration variables that may cause a collision, and not the entire configurations.**

Moreover, the algorithm is simple and requires few parameter tuning. Experimental results include navigation and manipulation tasks, e.g. a PR2 robot working in a kitchen environment and a sliding HRP2 robot.

I. INTRODUCTION AND PROBLEM STATEMENT

Motion planning for systems in cluttered environments has been adressed for more than thirty years [1], [2]. Most planners explore the system configuration space [3]. To compute a collision-free optimized motion, mainly three approches are used:

- Planning by path-optimization [4], [5] where obstacle avoidance is handled by constraints or cost using computation of the nearest obstacle distance. Most of these planners are using non-linear optimization [6] under constraints. Such planners provide close-to-optimality paths and have smaller time computation for easy problems, but they are mostly unable to solve narrow passage issues.
- Optimal random sampling [7] are also close to an optimal solution, but computation time is significantly higher than classical approaches. Moreover the roadmap is only valid for a given motion planning problem.
- Combining a classical random sampling based planner [8], [9] with a path or trajectory optimizer. Sampling based algorithms provide probabilitic completeness and remain effective even in high dimensional configuration spaces as required by humanoid inspired avatars.

For the last category of methods, random shortcut methods are still very popular [10]–[12] for optimizing the output of the path planning algorithm. However, random shortcut requires fine tuning of the termination condition (see Algorithm 2) and is not so efficient for long trajectories where only a small part needs to be optimized. On the other hand, numerical optimization methods like CHOMP [13] can be used. They have clear termination conditions, but collision avoidance is handled by inequality constraints sampled at many points along the trajectory. Those methods therefore require a pre-processing step of the robot (and/or environment) model in order to make it simpler: [13] covers PR2 bodies with spheres, while [14] needs to decompose objects into convex subsets. Moreover, in the applications we address, optimality is not desirable as such since the shortest path between two configurations in a cluttered environment usually contains contact with an obstacle.

*This work is partly supported by the project European Research Council (ERC Advanced Grant 340050 Actanthrope) and by the European project FP7 608849 EuRoc

[1]{mcampana, florent, jpl}@laas.fr
CNRS, LAAS, 7 avenue du colonel Roche, F–31400 Toulouse, France
[2]Univ. de Toulouse, LAAS, F–31400 Toulouse, France

For complex environments with numerous polyhedric-mesh obstacles, distance computation is much more time-consuming than collision-checking.

The idea of our method is to find a good trade-off between the simplicity of "blind" methods like shortcut algorithm, and the complexity of distance based optimization techniques. The idea is to use gradient-based information without evaluating distances between objects. Moreover the way we introduce constraints prevents the method to converge to contact motions.

Our algorithm does not pretend to compute a path which is optimal with respect to any criterion. Optimal paths rarely exist, and when they exist, their computation is most of the time out of the scope of numerical optimization algorithms [15]. The method we propose is then issued from a pragmatical compromise between generality, time computation efficiency and solution quality.

Related work is presented in Section II. Section III explains how the path-optimizer works, from the formulation of the problem to the implemented algorithm. Finally, we conclude on experimental results in Section IV and lay out future work perspectives.

II. RELATED WORK

In a similar framework to our approach, the CHOMP algorithm [13] optimizes an initial guess and tries to smooth it. It reduces a time invariant cost function using an efficient covariant hamiltonian gradient-descent. The cost is quantified by non-smooth parts (with high velocities) and an obstacle avoidance term, provided by the distance to the nearest obstacle for each iteration of the trajectory. But computing these nearest distances is quite long because all object pairs must be analysed at each time. Therefore the method starts to build offline a map of distances that will be called during the optimization at the requested time. Besides, meshes are preprocessed into overlapping spheres so that distances are computed faster in exchange of a geometry approximation. Finally, avoiding obstacles may lead to convergence issues when planning in narrow passages.

STOMP method [16], inspired by CHOMP, avoids to compute an explicit gradient for cost optimization using a stochastic analysis of local random samples. But as for CHOMP, the obstacle cost term requires a voxel map to perform its Euclidian Distance Transforms, and represents the robot bodies with overlapping spheres. Such technique provides lots of distance and penetration information but remains very time consuming and is not as precise as some distance computation techniques based on the problem meshes as Gilbert-Johnson-Keerthi [17].

Some optimization-based planners may not require an initial guess but some naive straight-line manually or randomly-sampled initialization as TrajOp [18]. Its trajectory is iteratively optimized with sequential convex optimization by minimizing at each step its square length, linear constraints and non-linear constraints considered as penalities. To compute the collision-constraints, nearest obstacle distances are calculated at each discrete time of the trajectory vector. This can be a burden for a high-dimensional robot or a complex environment as we propose to use, and may be compensated with a short path composed of only one or two waypoints.

Torque command laws can be constructed from repulsive potential and "internal" path deformation forces in the elastic strips optimization [19]. This optimizer is based on a spring formulation, looking for a path in the robot free-space defined by geometry-overlapping spheres, which radii are smaller than the nearest obstacle distance. For distance computations, robot is also approximated with spheres in order to apply hierarchical bouding sphere method [20]. As for CHOMP, the initial geometry of the problem is not respected and spheres allow to increase the speed of distance computations.

Some heuristics use random shortcuts on the initial guess combined with a trajectory re-building that returns $\mathscr{C}^1$ shortcuts made of parabolas and lines (bang-bang control) [11]. Other smoothing methods randomly sample the initial trajectory to interpolate with B-splines [21], even for SO_3 joints thanks to an exponential map. The returned trajectory is shorter and $\mathscr{C}^2$ except on a few points. The interpolation is used in two stages: a global interpolation to smooth the trajectory, and a local random sample shortcut choosing between linear (with $\mathscr{C}^1$ ends conditions) or spline interpolation, to decrease the trajectory length. However, the convergence speed and the result quality remain probabilistic, as for all shortcuts methods [10]. For example, if we consider long paths which require only some local optimization, random shortcut will fail to sample relevant configurations, whereas numerical optimization algorithms as our method will normally succeed.

III. PATH OPTIMIZATION

In this part, we describe our optimization problem: notations, cost function, constraints and finally the optimization algorithm.

A. Notations

Lets $\mathbf{C}$ denotes the configuration space of the robot of dimension $n_{\mathbf{C}}$. We want to optimize a polygonal path computed by a sample-based planner, given the initial configuration q_0 and the goal configuration q_{n+1}. n is the number of waypoints of the initial path, the vector of these waypoints $x_i \in \mathbf{C}^n$ will be modified in order to reduce the path cost at the i^{th} iteration of the algorithm.

B. Problem definition

We want to reduce the sum of length squares of each segment. The optimal solution without constraint for this cost minimizes the total length of the path, but it also is a quadratic function:

$$c(x) = \frac{1}{2}\sum_{k=1}^{n} \|q_{k+1} - q_k\|^2$$

Optimizing the problem remains to find $\operatorname{argmin} c(x)$, subject to constraints which will be defined in the following

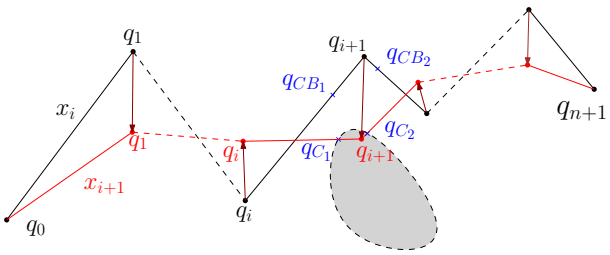

Fig. 2: Illustration of one iteration of our optimization. x_{i+1} appears to be in collision with the obstacle, border colliding configurations q_{C_1} and q_{C_2} are returned by the collision checker.

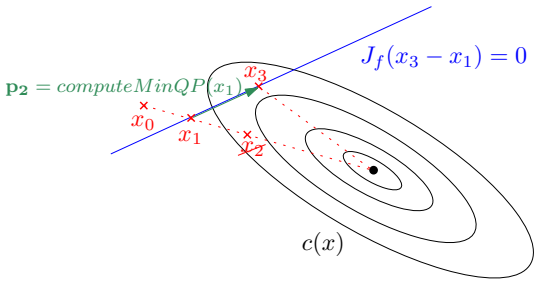

Fig. 3: Some iterations of the optimization with a representation of the quadratic cost (ellipsoids), its minimum (dot) and the constraints tangent-space (straight-line). x_2 has collisions that allow to compute constraints on x_1 and then the step minimizing the cost under the constraint leading to the new x_3 path. If x_3 is collision-free, our resolution is over.

sections. In the above norm, each robot degree of freedom is weighed based on the farthest point moved by the corresponding degree of freedom. We omit this fact in the following developments for clarity.

If H denotes the constant Hessian of the cost function, an iteration is described as follow:

$$\begin{aligned} \mathbf{p_{i,i+1}} &= H^{-1}\nabla c(x_i)^T \\ x_{i+1} &= x_i - \alpha_i \mathbf{p_{i,i+1}} \end{aligned} \tag{1}$$

where α_i is a real valued parameter. Taking $\alpha_i = 1$ yields the unconstrained minimal cost path (all waypoints aligned at equal distance from each other on the straight line between q_0 and q_{n+1}). As we checked before planning that this solution is in collision, we set $\alpha_i = \alpha$ where α is a parameter that will be precised in the algorithm section.

We iterate step (1) until path x_{i+1} is in collision. When a collision is detected, we introduce a constraint and perform a new iteration from x_i as explained in the next section.

C. Constraints

Let T be a positive real number such that each path x_i is a mapping from interval $[0,T]$ into $\mathbf{C}$: $x_i(0) = q_0$, $x_i(T) = q_{n+1}$ for all i. Let us denote by $t_{coll\,i}$ the abscissa of the first collision detected on path x_{i+1}. Thus in configuration $x_{i+1}(t_{coll\,i})$ a collision has been detected. Two cases are possible

1) the collision occured between two bodies of the robot: $\mathscr{B}_1$ and $\mathscr{B}_2$, or
2) the collision occured between a body of the robot $\mathscr{B}_1$ and the environment.

In the first case, we denote by $M \in SE(3)$ the relative transformation of $\mathscr{B}_2$ with respect to $\mathscr{B}_1$ when the robot is in configuration $x_i(t_{coll\,i})$. We add the constraint that for any iteration $j > i+1$, x_j of the optimization process, the relative position of $\mathscr{B}_2$ with respect to $\mathscr{B}_1$ is equal to $M \in SE(3)$ in configuration $x_j(t_{coll\,i})$. In the second case, the constraint is defined similarly, except that the constraint is on the position of $\mathscr{B}_1$ with respect to the environment.

We denote by F the mapping from $\mathbf{C}$ to $SE(3)$ such that the constraint is written:

$$F(x_i(t_{coll\,i})) = M \tag{2}$$

This constraint is non-linear in the state x_i, but we linearize it around x_i to keep the quadratic program formulation of the problem as explained in the next section.

D. Constrained quadratic program (QP)

The constraint on the position at $t_{coll\,i}$ is a function that depends only on two waypoints $q_{k\,i}$[1], $q_{k+1\,i}$, with $0 \leq k \leq n$. There exists $\beta \in [0,1]$ such that for any j:

$$x_j(t_{coll\,i}) = (1-\beta)q_{k\,j} + \beta q_{k+1\,j}$$

Therefore, the Jacobian J_F of constraint (2) with respect to x is

$$\begin{pmatrix} 0 & \cdots & (1-\beta)\frac{\partial F}{\partial q}(x_j(t_{coll\,i})) & \beta\frac{\partial F}{\partial q}(x_j(t_{coll\,i})) & \cdots & 0 \end{pmatrix}$$

Both F and $\frac{\partial F}{\partial q}$ are provided by our path planning framework [22] as differentiable functions.

The linearized constraint that future iteration should satisfy is the following:

$$J_F(x_j - x_i) = 0$$

where we recall that x_i is the latest collision-free iteration. We refer to [23] for solving constrained quadratic programs. Using the Singular Value Decomposition (SVD) of Eigen linear algebra library and a Cholesky LL^T orthogonal decomposition, this is straightforward to compute the step leading to x_j, the constrained QP minimum.

Figure 2 illustrates the case when x_{i+1} is in collision. Two strategies are possible for adding constraints when testing the path from q_0 to q_{n+1}: we can choose to return the first configuration in collision q_{C_1} with a discrete or continous collision checker. However, if we want to gather a maximum of colliding configurations from x_{i+1}, in order to compute more constraints in one iteration, we can compute with a discrete collision checker input and output collisions regarding each colliding obstacle.

Figure 2 is in that second case: q_{C_1} and q_{C_2} are both returned as colliding configurations. Thus, q_{CB_1} and q_{CB_1} represent the configuration for which constraints are computed and will be applied when calculating x_{i+2}. This method

[1] q_{ki} denotes waypoint k along path x_i.

could prevent from calculating more iterations to resolve collisions, but in pratice, we found that it is much heavier to compute these collisions and the numerous obtained constraints is alterating the path optimality. Therefore we choose to compute only first the collision using a discrete collision checker.

Figure 3 illustrates briefly our first iterations according to the quadratic cost and the constraints sub-space.

E. Algorithm

In this part we describe the path-optimizer algorithm 1 according to the previous step definitions. The main difficulty here is to handle the scalar α determining how much of the computed step $\mathbf{p}$ will be traveled through, and also to return a collision-free path. Typically, we chose $\alpha_{init} = 0.2$ to process small steps. $\alpha_{init} = 0.5$ is more encountered in the optimization literature, procuring larger steps but with more risks of collision. i_{max} represents the maximal number of iterations. The iteration counter is basically incremented each time a collision test occurs.

Algorithm 1 Description of the path optimization algorithm.

1: **procedure** OPTIMIZE(x_0)
2: $i \leftarrow 0 \quad \alpha \leftarrow \alpha_{init}$
3: **while** True **do**
4: $\mathbf{p} \leftarrow H^{-1}\nabla c(x)^T$
5: $x_{tmp} \leftarrow x - \alpha\mathbf{p}$
6: $i \leftarrow i+1$
7: **if** collision(x_{tmp}) **then** BREAK
8: $x \leftarrow x_{tmp}$
9: $\alpha \leftarrow \alpha_{max} - 0.8\,(\alpha_{max} - \alpha)$
10: **while** True **do**
11: addCollisionConstraints(J_f, x_{tmp})
12: $\mathbf{p} \leftarrow$ computeMinQP(x)
13: **if** isFullRank(J_f) **or** $i > i_{max}$ **then return** x
14: $x_{tmp} \leftarrow x - \mathbf{p}$
15: $i \leftarrow i+1$
16: **if not**(collision(x_{tmp})) **then return** x_{tmp}
17: $\alpha \leftarrow \alpha_{init}$
18: $x_{tmp} \leftarrow x - \alpha\mathbf{p}$
19: $i \leftarrow i+1$
20: **if not**(collision(x_{tmp})) **then**
21: **while** True **do**
22: $x \leftarrow x_{tmp}$
23: $\mathbf{p} \leftarrow$ computeMinQP(x)
24: $x_{tmp} \leftarrow x - \alpha\mathbf{p}$
25: $i \leftarrow i+1$
26: **if** collision(x_{tmp}) **then** BREAK
27: **if** $i > i_{max}$ **then return** x_{tmp}
28: $\alpha \leftarrow \alpha_{max} - 0.8\,(\alpha_{max} - \alpha)$
29: **end procedure**

The following part describes in details the algorithm 1. Lines 2 represents the initialization, the following loop (begining line 3) is the first unconstrained QP iterative resolution. The more iterations of the loop, the higher α becomes, toward α_{max} (line 9). Thus we are converging to the minimum of our unconstrained QP, until the loop breaks because of a detected collision (line 7).

The second main loop of our algorithm (line 10), manages new collision constraints at its begining (line 11) and computes the step leading to the minimum of the constrained QP (line 12). Since no relaxation of our constraints is considered, our problem may become full constrainted, equivalent to J_f being full rank (line 13). In that case, the last collision-free path will be returned.

Globally, each time a collision-constraint is added, we directly compute and test the solution of the QP under previously added constraints with an equivalent of $\alpha = 1$ since if this path is collision free, the algorithm is over (lines 14 and 16). Thus we do not waste time computing smaller iterations that would have lead to the same result, if no obstacle has been jumped at line 14.

If this path has collisions, we do not compute associated constraints because they may not be relevant since this path is very optimized and not similar to its previous iteration. Instead, we re-use the found optimal step $\mathbf{p}$ limited by α (line 18) and test it. If this path has collisions, we go back to the begining of the loop (line 10) and add constraints to J_f. In the other case, we continue the gradient-descent adapting our QP solution to the current path x (lines 23 and 24) until a collision is detected (line 26) or until the maximal number of iterations is reached (line 27).

At last, the algorithm parameters are that not numerous compared to other numerical optimizations based on distance computation: α_{init}, α_{max} and i_{max}.

IV. RESULTS

This part gathers optimization results performed on the planning software Humanoid Path Planner [22]. The initial trajectory is obtained with two kind of probabilistic planners: Visibility-PRM [24] and RRT-connect [25].

A. From 2D basic examples...

Figure 4 shows several iterations of our optimizer on 2D cases. Since in this special case, transformation constraints are equivalent to compulsory configurations to pass through, we can verify that our optimized path is applying all computed constraints. We can also analyse which collisions have lead to the backtracked constraints.

Besides, these examples give a better understanding of how the tuning of α has to balance lot of iterations and relevant collision-constraint addition. For example, if we alterate a lot the initial path with a large gradient step and compute the corresponding collisions, constraints will be chosen on a very-not optimal path and will not be pertinent with regard to the obstacles we wanted to avoid.

B. Toward 3D complex problems

We also experiment our algorithm on more complex robots, with transformation collision-constraints.

In the included video, five problems present the good properties and limits of our algorithm. On the 3-DOF arm,

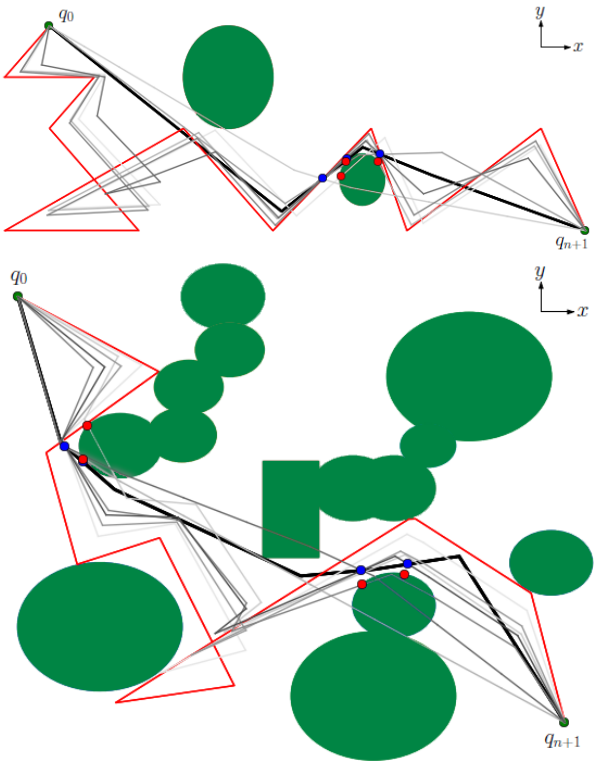

Fig. 4: Path-optimization results on a 2D-ponctual robot, moving around obstacles. Initial paths are in red, optimized ones in black. Grey paths represent intermediate iterations, red dots colliding configurations and blue dots the associated configurations where constraints are computed. Respectively on top and bottom, 8 and 13 iterations have occured.

optimization is efficient since constraints are computed for the end-effector in the global frame to slightly go along the obstacle. We even noticed that in practice, the smaller alpha, the nearer the robot is from the white cylinder.

For the following example with a 6-axis manipulator arm (Figure 5) in a cluttered environment, path reduction only manages to reduce some but not all detours. We explain this behaviour by a path over-constraining due to the important proximity of the robot to the obstacles during the whole path, and because there are not so many DOF to play on to easily avoid collisions and return a small-cost solution. We advance this explaination because on the two last high-DOF examples

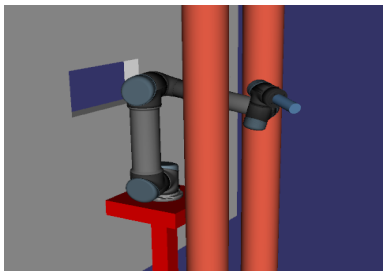

Fig. 5: 6-DOF robot arm on which our path-optimization has been tested.

Algorithm 2 Random shortcut as adapted from [10] Section 6.4.1. `steeringMethod` returns the linear interpolation between two configurations. $x_{|I}$ denotes path x restricted to interval I. *maxNbFailures* is a parameter that affects time of computation and quality of the result.

procedure RANDOMSHORTCUT(x)
  nbFailures $\leftarrow 0$
  while $nbFailures < maxNbFailures$ **do**
    $failure \leftarrow true$
    $T \leftarrow$ upper bound of x definition interval
    $t_1 < t_2 \leftarrow$ random numbers in $[0,T]$
    $lp0 \leftarrow$ `steeringMethod`$(x(0),x(t_1))$
    $lp1 \leftarrow$ `steeringMethod`$(x(t_1),x(t_2))$
    $lp2 \leftarrow$ `steeringMethod`$(x(t_2),x(T))$
    $newPath \leftarrow$ empty path defined on $[0,0]$
    if $lp0$ is collision-free **then**
      $newPath \leftarrow lp0$; $failure \leftarrow false$
    else
      $newPath \leftarrow x_{|[0,t_1]}$
    if $lp1$ is collision-free **then**
      $newPath \leftarrow$ `concatenate`$(newPath, lp1)$
      $failure \leftarrow false$
    else
      $newPath \leftarrow$ `concatenate`$(newPath, x_{|[t_1,t_2]})$
    if $lp2$ is collision-free **then**
      $newPath \leftarrow$ `concatenate`$(newPath, lp2)$
      $failure \leftarrow false$
    else
      $newPath \leftarrow$ `concatenate`$(newPath, x_{|[t_2,T]})$
    $x \leftarrow newPath$
    if $failure$ **then** $nbFailures \leftarrow nbFailures + 1$
  return x
end procedure

of the video (and Figures 1 and 6), results are far better in terms of cost and quality.

Let us consider the worked-out example of a mobile 40-DOF PR2 performing a manipulation task in a kitchen environment (Figure 1). The task is simple: the robot has just to move his hands from the top to the bottom of the table. First, a sampling based path planning algorithm computes a solution that includes large detours for the platform and useless moves of the arms and the head of the robot (see Figure 1 top). We want and manage to optimize it to a logically trajectory Our optimizer manages to reduce its cost and improves its quality (Figure 1 bottom) just adding constraints between the table and the robot's grippers. Thus, the robot just slightly moves backward its mobile platform and uses its arm DOF to avoid the table.

We also experiment our optimizer on an other high-DOF robot, a sliding HRP2 [26]. The robot simply has to pass its left arm from its front to its back. Once again, the planner offer us detours and activates non-usefull DOF such as the left arm and the translation on the ground (Figure 6 top). This example is here interesting to compare the optimized

path quality between random shortcut (Algorithm 2) and our method, because the first one will never completely *kill* the translation move of the robot and other useless DOF uses, unlike our optimized-path which mainly results in moving the left arm as expected (Figure 6 bottom). The removal of the translation is also a consequence of our cost ponderation as described in the previous section.

We are dependent on the shape and number of waypoints of the output x_0 of the planner, dependancy we tried to anihilate computing 30 runs of each problem and using the two cited planners. The results are presented in the table I.

The comparaison with random shortcut (Algorithm 2) is presented in Table I. In all cases, we have a smaller average computation time than random shortcut since it is computing many more iterations until satisfying its final criterion. Now regarding the cost, for high-DOF examples where we expect to beat random shortcut in terms of computation time of path cost as HRP2 and PR2, the challenge is succeeded. However, there are still scenarii where our algorithm needs some improvement, as the 6-DOF arm. We believe that, since high-mobility results are better, this limitation is linked to the rigidity of our collision-constraints. Adding a possibility to relax or cancel well-chosen constraints would improve this statement.

Moreover, the termination condition of random shortcut allows it to try shortening the path until 15 iterations of non-improvement are reached (corresponding to *maxNbFailures* in Algorithm 2). This is not equivalent to our iteration limitation i_{max}. But we admit that restrincting the number of random shortcut iterations is neither a acceptable criterion because of randomness.

V. CONCLUSIONS

We managed to settle a path optimization for high dimension articulated robots. Our algorithm uses standard numerical tools as collision checking, singular value decomposition and QP resolution. The algorithm correlates theses tools in a simple but effective way, playing on the scalar iteration parameter. Therefore, our method does not require distance computation, geometry pre-processing nor offline optimization. We demonstrate that the optimizer is time-competitive comparing to random shortcut and proposes better quality paths for high-DOF robots.

In a future work, we plan to implement user-defined constraints of the problem, simply modifying our collision-constraint jacobian, and also to improve matrix computations to increase the optimization speed and be more competitive. It would egally be interesting to study the possibility of relaxing our collision-constraints to locally improve our path iterations.

REFERENCES

[1] M. Brady, J. Hollerbach, T. Johnson, T. Lozano-Pérez, and M. T. Masson, *Robot Motion: Planning and Control*, 1983.

[2] J. Schwartz and M. Sharir, "On the piano movers problem ii: General techniques for computing topological properties of real algebraic manifolds," *Advances of Applied Mathematics*, vol. 4, no. 3, pp. 298–351, 1983.

[3] T. Lozano-Perez, "Spatial planning: A configuration space approach," *IEEE Transactions on Computers*, vol. 32, no. 2, pp. 108–120, 1983.

[4] C. Park, J. Pan, and D. Manocha, "ITOMP: incremental trajectory optimization for real-time replanning in dynamic environments," in *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling, ICAPS 2012, Atibaia, São Paulo, Brazil*, 2012.

[5] M. Garber and M. Lin, "Constraint-based motion planning using voronoi diagrams." in *Algorithmic Foundations of Robotics V, volume 7 of Springer Tracts in Advanced Robotics*, 2004, pp. 541–558.

[6] J. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, 2nd ed. Society for Industrial and Applied Mathematics, 2010.

[7] S. K. Sertac and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[8] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *Robotics and Automation, IEEE Transactions on*, vol. 12, no. 4, 1996.

[9] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, 2001, pp. 293–308.

[10] S. Sekhavat, P. Svestka, J.-P. Laumond, and M. Overmars, "Multi-level path planning for nonholonomic robots using semi-holonomic subsystems," *International Journal of Robotics Research*, vol. 17, no. 8, pp. 840–857, 1998.

[11] K. Hauser and V. Ng-Thow-Hing, "Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts," in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 2010, pp. 2493–2498.

[12] R. Guernane and N. Achour, "Generating optimized paths for motion planning," *Robotics and Autonomous Systems*, vol. 59, no. 10, pp. 789–800, 2011.

[13] M. Zucker, N. Ratliff, A. Dragan, M. Pivtoraiko, M. Klingensmith, C. Dellin, J. Bagnell, and S. Srinivasa, "CHOMP: covariant hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[14] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 28, no. 5, 2014.

[15] J.-P. Laumond, N. Mansard, and J. Lasserre, "Optimality in robot motion: Optimal versus optimized motion," *Communications of the ACM*, vol. 57, no. 9, pp. 82–89, 2014.

[16] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011, pp. 4569–4574.

[17] E. Gilbert, D. Johnson, and S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal of Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[18] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[19] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 1031–1052, 2002.

[20] S. Quilan, "Efficient distance computation between non-convex objects," in *Robotics and Automation. ICRA 1994. Proceedings of the IEEE International Conference on*, 1994, pp. 3324–3329.

[21] J. Pan, L. Zhang, and D. Manocha, "Collision-free and smooth trajectory computation in cluttered environments," *International Journal of Robotics Research*, vol. 31, no. 10, 2012.

[22] F. Lamiraux and J. Mirabel, "Humanoid path planner," http://projects.laas.fr/gepetto/index.php/Software/Hpp.

[23] J. Nocedal and S. Wright, *Numerical Optimization, Second Edition*. Springer New York, 2006.

[24] T. Siméon, J.-P. Laumond, and C. Nissoux, "Visibility based probabilistic roadmaps for motion planning," *Advanced Robotics Journal*, vol. 14, no. 6, 2000.

[25] J. J. Kuffner and S. M. Lavalle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, IEEE International Conference on*, vol. 2, 2000, pp. 995–1001.

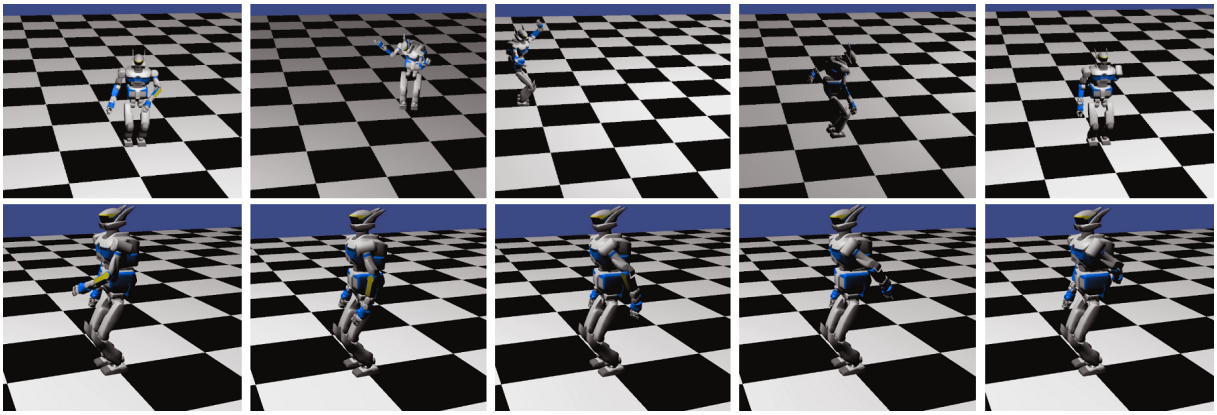

Fig. 6: Sliding HRP2 robot. On top, path returned by the sample-based planer, with large translations. On bottom, path optimized by our method, the robot does not move its feet anymore. The cost has been down from 13.4 to 1.38 in 30 iterations.

	Non-optimized cost	Optimized cost		Average time computation gain compared to random shortcut	Average number of iterations	
		Our optimizer	Random shortcut		Our optimizer	Random shortcut
2D-robot with obstacles (RRT)	23.6 ± 2.4	17.3 ± 1.7	15.5 ± 3.7	32%	14	108
3-DOF arm (RRT)	7.03 ± 1.23	5.18 ± 1.35	3.70 ± 1.08	36%	12	69
6-DOF arm (RRT)	13.1 ± 1.8	12.4 ± 2.0	6.83 ± 1.54	75%	30	106
PR2 in kitchen (PRM)	15.1 ± 2.2	5.75 ± 1.6	5.90 ± 1.04	50%	19	104
HRP2 (PRM)	9.45 ± 1.69	1.82 ± 0.28	3.42 ± 1.24	14%	14	52

TABLE I: Results for 30 runs of the presented examples with our path-optimization method (with parameters $i_{max} = 30$, $\alpha_{init} = 0.2$ and $\alpha_{max} = 0.9$). RRT and PRM stand for the used planner (resp. RRT-connect and Visibility-PRM). Average time computation gain column concerns only the optimization part of the process.

[26] S. Dalibard, A. E. Khoury, F. Lamiraux, M. Taïx, and J.-P. Laumond, "Small-space controllability of a walking humanoid robot," in *Humanoid Robots (Humanoids), 2011 11th IEEE-RAS International Conference on*, 2011, pp. 739–744.