# Trajectory Smoothing using Jerk Bounded Shortcuts for Service Manipulator Robots

Ran Zhao, Daniel Sidobre

# Trajectory Smoothing using Jerk Bounded Shortcuts for Service Manipulator Robots

Ran Zhao[1,2] and Daniel Sidobre[1,2]

***Abstract*— This paper aims to smooth jerky trajectories for high-DOF manipulators with Soft Motion [1] shortcuts which are bounded in velocity, acceleration and jerk. The algorithm presented here iteratively picks two points on the trajectory and attempts to replace the intermediate trajectory with a shorter and collision-free segment. The objective of this algorithm is to shorten the execution time of an input trajectory as much as possible while retaining the feasibility. Simulation and real-world experimental results on reaching tasks in human environments show that this technique can generate smooth and collision-free motions for a KUKA Light-Weight Robot.**

## I. INTRODUCTION

To achieve a large variety of tasks in interaction with human or human environments, autonomous robots must have the capability to quickly generate safe and natural-looking motions. Classical approaches, such as sample-based planners (e.g. RRT [2], PRM [3]) define the motion by collision-free paths, which are expected to follow by a robot. Despite the high speed, these approaches often deliver a path as piecewise polygonal lines, which constrain the motion to stop at each vertex. As a result, this motion is slow and unnatural. Thus, trajectory smoothing is always performed before execution so as to produce feasible motions.

Moreover, arm manipulators for human interaction should be intrinsically safe [4]. In a human interaction context, safety is directly linked with the velocity bound while comfort is linked with the acceleration and jerk bounds. This paper presents a fast and simple smoothing algorithm for generating dynamic trajectories from paths. This algorithm respects the velocity, acceleration and jerk bounds and avoids collision.

We apply a variant of a shortcutting heuristic, which is commonly used in robotics and animation. The heuristic randomly selects two points along the existing path, then constructs a segment between them in the configuration space, and checks the occurrence of collision. If it is collision-free, the segment replaces the subpath between the two points. Then third-degree polynomial functions are used to describe the trajectories. These functions bring about sufficient flexibility in terms of providing higher-order smoothness, i.e., computing trajectories that are $\mathcal{C}^2$. In the first stage, we suppose that a motion planner produces a polygonal collision-free path. From this polygonal path, we build a feasible trajectory composed of piecewise straight-lines that are bounded in jerk, acceleration and velocity. This trajectory corresponds to type V in the classification proposed by Kröger [5]. Then we smooth the trajectory by removing some unnecessary turns. As this operation modifies the initial free path, the new smoothed path must be checked for collision. The output of the algorithm is a smooth trajectory that respects the collision and kinematic motion bounds (velocity, acceleration and jerk), as shown in Fig. 1.

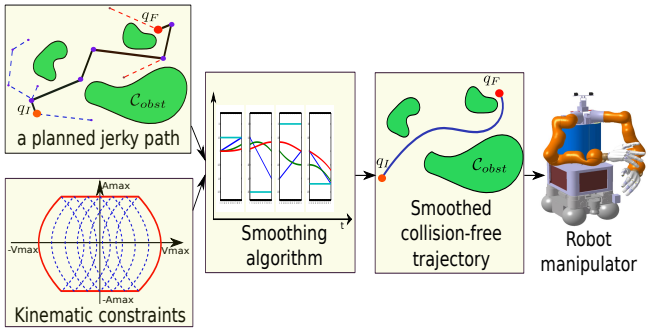



Fig. 1. Overview: Building a smooth trajectory that respects collision,velocity, acceleration and jerk constraints for robot manipulators

The main contributions of this work are listed below.

1) We propose a simple and fast algorithm that operates in the configuration/velocity/acceleration state space.
2) We make an analytical derivation of time-optimal, velocity-bounded, acceleration-bounded and jerk-bounded trajectories that interpolate between endpoints with specified velocity and acceleration. For a single joint, the time-optimal interpolant can be derived in the closed form. We interpolate multiple joints by detecting the joint with the longest execution time, and then interpolate the remaining joints by finding the jerk-bounded, acceleration-bounded and velocity-bounded interpolants with fixed duration.
3) We also present a method for fast collision checking of third-order polynomial trajectories based on a reported technique [6].

## II. RELATED WORK

High-quality motion generation is a topic of interest in the field of robotics, especially related to the domain of Human-Robot Interactions (HRI), which requires safe and human-like motions. A solution [1] was proposed to compute

*This work has been supported by the European Community's Seventh Framework Program FP7/2007-2013 "SAPHARI" under grant agreement no. 287513 and by the French National Research Agency project ANR-10-CORD-0025 "ICARO".

[1,2]R. Zhao and D. Sidbore are with CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France; and Univ de Toulouse; UPS, LAAS; F-31400 Toulouse, France. ran.zhao@laas.fr, daniel.sidobre@laas.fr

a time-optimal trajectory bounded in jerk, acceleration and velocity for an axis joining any pairs of states defined by a position, a velocity and an acceleration. For a point-to-point movement in a $n$-dimensional space, the time-optimal straight-line motion is obtained by projecting the bounds onto the line [7]. It should be noted that all these methods only build time evolution functions along the paths, and thus generate unnatural motions as outputs.

Many techniques have been proposed in literatures to generate smooth motions. Optimization approaches have long been pursued with the establishment of various methods. An optimal motion can be defined based on a cost function, including obstacle potential fields [8] [9] or physical criteria such as execution time, torque, or energy consumption [10]–[12] of the overall robot. Then the optimality criteria is minimized using iterative numerical techniques such as gradient descent. These techniques are typically computationally expensive owing to the high dimension of the optimization problem and the large number of bounds. Thus, these techniques are too slow for real-time use.

The shortcut technique is fast, easily implemented, and often produces high-quality paths quickly in practice. It uses heuristics to iteratively replace some portions of a path with shorter linear or parabolic segments [13] [14]. Then simple velocity and acceleration bounds were imposed over these parabolic segments [15]. A variant shortcut method was proposed by connecting two fixed points on two adjacent trajectories respectively [16]. The output trajectory deviates very slightly from the original one since the two points are close to the vertex of each trajectory. As a result, even the smoothed path is sometimes jerky and cannot satisfy the kinematic bounds.

In this paper we propose a solution to generate a trajectory from a jerky path. The objective of the method is to quickly build a smooth trajectory that respects collision, velocity, acceleration and jerk constraints. Meanwhile, our approach can be used by a large variety of robot manipulators.

## III. Notations and Representations

In this section, we introduce the notations and present the underlying Soft Motion representations used in computation of the trajectories.

### A. Notations

Let $\mathcal{C} = \mathbb{R}^n$ denote the $n$-dimensional configuration space, and let $\mathcal{C}_{free}$ denote the subset of configurations that are collision-free and respect joint limits. A trajectory $\mathcal{T}$ can be a direct function of time or the composition $\mathcal{P}(u(t))$ of a path $\mathcal{P}(u)$ and a function $u(t)$ describing the time evolution along this path. The background trajectory materials are summarized in the books from Biagiotti [17] and Kröger [5]. A trajectory $\mathcal{T}$ is then defined as:

$$\mathcal{T} : [t_I, t_F] \longrightarrow \mathbb{R}^n \tag{1}$$

$$t \longmapsto \mathcal{T}(t) \tag{2}$$

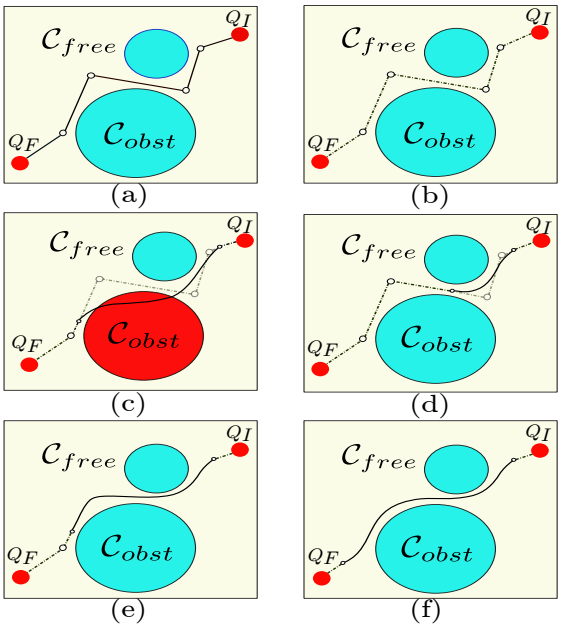



Fig. 2. Smooth algorithm. (a) A jerky path as a list of waypoints. (b) Converting into a trajectory that halts at each waypoints. (c) Performing a shortcut that fails in collision check. (d),(e) Two more successful shortcuts (f) The final trajectory.

where $\mathcal{T}(t) = Q(t) = ({}_1Q(t), {}_2Q(t), \cdots, {}_nQ(t))^T$ in joint space from the time interval $[t_I, t_F]$ to $\mathbb{R}^n$. $\mathcal{T}(t)$ is collision-free if all the configurations on $\mathcal{T}(t)$ lay in $\mathcal{C}_{free}$:

$$\mathcal{T}(t) \in \mathcal{C}_{free} \quad \forall t \in [t_I, t_F] \tag{3}$$

### B. Soft Motion Trajectories

We choose a particular series of $3^{rd}$ degree polynomial trajectories and name them as Soft Motion trajectories. A Soft Motion trajectory is a type V trajectory defined by Kröger, which satisfies:

$$\begin{aligned} V(t) \in \mathbb{R}^n \quad & |V(t)| \leq V_{max} \\ A(t) \in \mathbb{R}^n \quad & |A(t)| \leq A_{max} \\ Q(t) \in \mathbb{R}^n \quad & |J(t)| \leq J_{max} \end{aligned} \tag{4}$$

where $Q$, $V$, $A$ and $J$ represent the position, velocity, acceleration and jerk, respectively. $J_{max}$, $A_{max}$ and $V_{max}$ are the kinematic constraints. Therefore, at a discrete instant, the trajectory can transfer the motion states while not exceeding the given motion bounds. The continuity class of the trajectory is $\mathcal{C}^2$. $\mathcal{T}(t)$ is considered as feasible if it satisfies both Constraints (3) and (4). Our objective is to shorten the execution time of an input trajectory as much as possible while retaining the feasibility.

A trajectory is able to make the system transfer from the current state to a target state at any time instant. For the discussion of the next sections, a state of motion at an instant $t_i$ is denoted as $M_i = (Q_i, V_i, A_i)$. Once the trajectory is calculated, the function $M_t = getMotionState(\mathcal{T}, t)$ returns the motion state on trajectory $\mathcal{T}$ at time $t$.

## IV. SHORTCUTTING ALGORITHMS

Figure 2 illustrates the smoothing algorithm that performs four iterations of shortcutting on a polygonal path. Collision check fails during the first shortcut, and after three more attempts, a feasible trajectory is generated.

**Algorithm 1** Shortcutting algorithm

**Input:** a path as a list of waypoints, iteration count $N$
**Output:** a smoothed collision-free trajectory

```
Plan a time-optimal trajectory T_ptp that stops at each waypoints
Initialize the smooth trajectory T_smooth = T_ptp
for iteration = 0 to N do
    Pick t_s and t_e randomly from [0, t_F]
    M_s=getMotionState(T_smooth, t_s)=(Q_s, V_s, A_s)
    M_e=getMotionState(T_smooth, t_e)=(Q_e, V_e, A_e)
    T_sc = ComputeShortcutTraj(M_s, M_e)
    if T_sc → CollisionFree() then
        Replace T_{M_s M_e} by T_sc in T_smooth
    end if
end for
return T_smooth
```

### A. Generation of Time-optimal Phase Synchronized Trajectory

The first step is to convert a piecewise linear path to a time-optimal trajectory that stops at each waypoint. The trajectory along a straight-line path should be a phase-synchronized motion [18]. Phase synchronization refers to the synchronization in the position, velocity, acceleration and jerk spaces. In other words, with any given time instant, all variables must complete the same percentage of their trajectories. The phase synchronization in an $n$-dimensional space is defined as follows [19]:

$$\frac{{}_iQ(t) - {}_iQ(t_I)}{{}_jQ(t) - {}_jQ(t_I)} = \lambda \quad \forall i,j \in [1,n], t \in [t_I, t_F] \tag{5}$$

where $\lambda$ is a constant.

To achieve that, we compute the final time for each dimension. Considering the largest motion time, we readjust the other dimension motions to this time. Time adjusting is done by decreasing linearly $J_{max}$, $A_{max}$ and $V_{max}$. For each segment of the trajectory, one of the velocity, acceleration, or jerk functions of the $n$ initial joints is saturated, while the others are inside their validity domain. In other words, the motion consumes minimum time for one direction. At other directions, the motions are conditioned by the minimum one. Repeating this strategy for each straight segment, we build the time optimal trajectory $\mathcal{T}_{ptp}$ that stops at each waypoint.

### B. Soft Motion Interpolation

In the second step of this algorithm, we start the iterations. The point-to-point (straight-line) trajectory $\mathcal{T}_{ptp}$ obtained in the first step is feasible, but is not satisfactory because the velocity varies greatly at each waypoint, which stops the motion. These stops can be avoided by drawing shortcuts between random points on the trajectory.

Since each joint variable is assumed to be independent, the minimum execution time between $M_s$ and $M_e$ is determined by the slowest single-joint trajectory. Then the motion on the rest of the joints must be interpolated to this imposed time which is called $T_{imp}$. More precisely, $f(M_s, M_e, J_{max}, A_{max}, V_{max})$ is used to compute the time of the time-optimal interpolant between motion states $M_s$ and $M_e$ under the kinematic bounds $J_{max}$, $A_{max}$ and $V_{max}$. Thus,

$$T_{imp} = max(f(M_s^j, M_e^j, J_{max}^j, A_{max}^j, V_{max}^j)) \quad j \in [1,n] \tag{6}$$

The interpolation problem becomes one of building a motion with predefined time. We propose three methods by computing different parameters of the trajectory.

*1) Three-Segment Interpolants:* If we consider states $M_s$ and $M_e$ defined by a starting instant $t_s$ and an ending instant $t_e$, the starting and ending situations to be connected are: $(Q_s, V_s, A_s)$ and $(Q_e, V_e, A_e)$. An interesting solution to connect this portion of trajectories is to define a sequence of three trajectory segments with constant jerk that bring the motion from the initial situation to the final one within time $T_{imp}$. We choose three segments because we need a small number of segments and there is not always a solution with one or two segments.

The system to be solved is then defined by 13 constraints: the initial and final situations (6 constraints), the continuity in position velocity and acceleration for the two switching situations and time. Each segment of a trajectory is defined by four parameters and time. If we fix the three durations $T_1 = T_2 = T_3 = \frac{T_{imp}}{3}$, we obtain a system with 13 parameters where only the three jerks are unknown [16]. As the final control system is periodic with period $T$, the time $T_{imp}/3$ must be a multiple of the period $T$, and in this study, $T_{imp}$ is chosen to be a multiple of $3T$.

*2) Three-Segment Interpolants With Bounded Jerk:* The three-segment interpolants solves the problem of trajectory generation with fixed duration for each segment. However, it cannot be guaranteed that the computed jerk is always bounded. Here, we introduce a variant three-segment method with defined jerk.

Same as the three-segment method, the system is also defined by 13 constraints. With the variant method, however, we fix the jerks on the first and third segments as $|J_1| = |J_3|$, which have the value bounded within the kinematic constraints. Then, the unknown parameters in the system are $J_2$ and the three time durations. Thus we obtain a system of four equations with four parameters ($J_2$, $T_1$, $T_2$ and $T_3$):

$$A_e = J_3T_3 + A_2 \tag{7}$$

$$V_e = J_3\frac{T_3^2}{2} + A_2T_3 + V_2 \tag{8}$$

$$Q_e = J_3\frac{T_3^3}{6} + A_2\frac{T_3^2}{2} + V_2T_3 + Q_2 \tag{9}$$

$$T_{imp} = T_1 + T_2 + T_3 \tag{10}$$

where

$$
\begin{aligned}
A_2 =& J_2 T_2 + J_1 T_1 + A_s \\
V_2 =& J_2 \frac{T_2^2}{2} + (J_1 T_1 + A_s) T_2 + J_1 \frac{T_1^2}{2} + A_s T_1 + V_s \\
Q_2 =& J_2 \frac{T_2^3}{6} + (J_1 T_1 + A_s) \frac{T_2^2}{2} + (J_1 \frac{T_1^2}{2} + A_s T_1 + V_s) T_2 \\
&+ J_1 \frac{T_1^3}{6} + A_s \frac{T_1^2}{2} + V_s T_1 + Q_s
\end{aligned}
$$

To choose the values of jerks on each dimension, we resort to the velocities $V_s$ and $V_e$. The jerks are fixed by $J_1 = -J_3 = J_{max}$ when $V_s - V_e > 0$, and by $J_1 = -J_3 = -J_{max}$ when $V_s - V_e < 0$. If $V_s - V_e = 0$, we compare the values of $A_s$ and $A_e$ instead.

*3) Jerk-Bounded, Acceleration-Bounded, Velocity-Bounded Interpolants:* Now we derive the all-bounded trajectory given a fixed duration $T_{imp}$. The method in section IV-B.2 can directly bound the jerk, but have to readjust the jerk values by a predefined resolution to bound the velocity and acceleration. As we detect the longest execution time $T_{imp}$ by computing the time-optimal trajectory on each joint, the jerk is saturated and the acceleration and velocity may be saturated, depending on different cases. Thus, we can extend the duration of all joints (except the one with the longest duration) to $T_{imp}$ by unsaturated interpolants while maintaining the number of segments $\mathcal{N}_j$ on each joint. We name it a *Slowing Down Motion*. Algorithm 2 shows the pseudocode.

**Algorithm 2** All-Bounded Interpolants Generation

**Input:** Motion states: $M_s$, $M_e$; number of DOFs: $n$; Kinematic constraints
**Output:** All-Bounded Interpolants

```
Compute T_imp using Eq. 6
for j = 1 to n do
    Compute the time-optimal interpolants between M_s and M_e, then get the execution time T_j
    Get N_j and the execution time on each segment T_j^N
    if N_j = 0 then
        No motion on this joint, maintain the time-optimal interpolants
    else
        Enlarge T_j^N by T_j^N = T_j^N + (T_imp − T_j)/N_j
        Compute the new Jerk on each segment J_j^N
    end if
    Generate the interpolants with J_j^N and T_j^N
end for
```

### C. Trajectory Collision Checking

Collision checking is a basic operation in any robot motion planning and smoothing algorithm. This operation is commonly realized by discretizing the curve at a predefined constant resolution $\epsilon$ and statically testing each sampled configuration. However, this approach is inexact and cannot detect all collision that occurs. If $\epsilon$ is too small, the collision checker will be unnecessarily slow. On the other hand, choosing it too large might result in missing some obstacles.

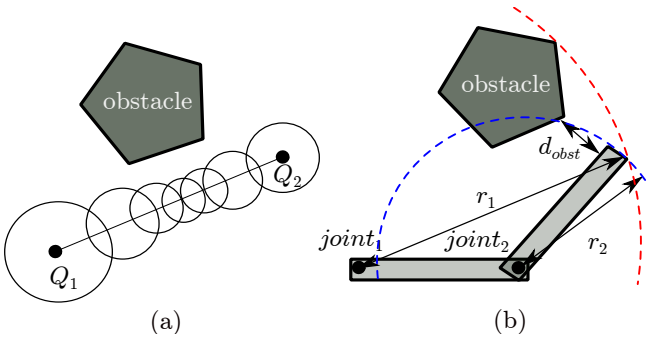



Fig. 3. (a) A collision-free $\mathcal{C}$-space path covered by free bubbles. (b) 2D robot manipulator showing the maximum distance $r_1$, $r_2$ and the minimum obstacle distance $d_{obst}$. The circle at the axis of joint 1 of radius $r_1$ (the red dashed line) contains the entire manipulator. The circle at joint 2 of radius $r_2$ (the blue dashed line) contains link 2.

In order to overcome this problem, we use an alternative local trajectory checking algorithm based on the divide and conquer algorithm [20] and the concept of bubbles of free configuration space, introduced in [6]. This algorithm recursively splits the path in two sub-paths, and then calculates bubbles of free space around a configuration and therefore can guarantee the collision-free status of a trajectory segment by overlapping these bubbles along each segment (Fig. 3(a)). The bubble $\mathcal{B}(Q)$ at the current configuration $Q$ is an upper bound computed using a distance $d_{obst}$, where $d_{obst}$ defines the minimum distance between the robot in configuration $Q$ and the obstacles. For the robots with $n$ revolute joints, the bubble will be diamond shaped [6]:

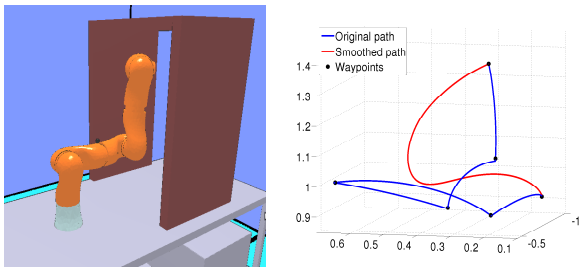
$$
\mathcal{B}(Q) = \left\{ X \in \mathcal{C} : \sum_{i=1}^{n} r_i |X_i - Q_i| \leq d_{obst} \right\} \tag{11}
$$

where $r_i$ is the radius of the cylinder that is centered along the axis of the $i$-th joint and contains all the subsequent links of the manipulator. Figure. 3(b) gives an example of a two-degree-of-freedom planar robot showing the maximum distance of each joint and the minimum obstacle distance $d_{obst}$. Algorithm 3 shows the overall pseudocode of exact collision check.

**Algorithm 3** Exact collision check

**Input:** A trajectory $\mathcal{T}$ in time interval $[t_s, t_e]$

```
Compute the free bubbles B(Q_s) and B(Q_e)
if OVERLAP(B(Q_s), B(Q_e)) then
    return collision-free
else
    Bisect the trajectory at (t_s + t_e)/2
    repeat
        Recurse on the two halves
    until collisionDetected or B(Q) ≤ threshold
    return collision or collision-free
end if
```




(a) (b)

Fig. 4. (a) A manipulator reaches under a shelf on a table from the zero position. (b) The blue curve depicts the original end effector path. The red curve depicts the smoothed path after 200 random shortcuts.

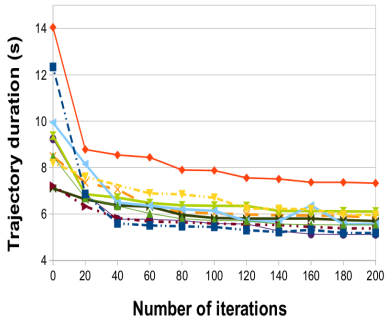



Fig. 5. The execution time of trajectories postproessed the smoothing algorithm for 10 different initial paths, on the task of Fig. 4.

Given a trajectory segment $\{\mathcal{T}(t)|t_s \leq t \leq t_e\}$, the algorithm computes the free bubbles $\mathcal{B}(Q_s)$ and $\mathcal{B}(Q_e)$ at $Q_s$ and $Q_e$, respectively. If they overlap, the algorithm terminates and reports a collision-free path. Otherwise, the segment is bisected at $\frac{t_s+t_e}{2}$ and the algorithm recurses on the two halves. We break the recursion when a collision is detected, or the largest segment, uncovered by bubbles, becomes smaller than the predefined threshold. As this method calculates a lower bound for the free bubble radius based on the minimum obstacle distance, the radius tends to get very small at a configuration with a low obstacle distance. Numerous distance and collision calculations are required in this situation, which slows down the collision check procedure. For real applications, as it is not desirable that the robot passes near the obstacles this trajectories can be discarded.

## V. Simulation and Experimental Results

### A. Simulation

Simulation was performed on a reaching task for a 7-DOF robot arm in human environment (Fig. 4). Totally 10 initial paths were generated with the same start and end configurations by a sample-based planner. Then these paths were converted into trajectories using both generation of phase-synchronized trajectory (see section IV-A) and our smooth algorithm. The maximum joint velocity is decided by the physical properties of the motor. The numerical values of the velocity constraint for each joint were cited from [21]. Thus, $V_{max}$ can be defined by a vector:

$$V_{max} = [v_1 \quad v_2 \quad v_3 \quad v_4 \quad v_5 \quad v_6 \quad v_7]^T$$
$$= [1.75 \quad 1.92 \quad 1.75 \quad 2.26 \quad 2.26 \quad 3.14 \quad 3.14]^T \, rad/s$$

Then, the kinematic motion bounds are defined in Table I. The numeric values are:

TABLE I

Robot motion is limited in jerk, acceleration, and velocity

| Jerk | Acceleration | Velocity |
|---|---|---|
| 5*$A_{max}$ $rad/s^3$ | 2.5*$V_{max}$ $rad/s^2$ | $V_{max}$ $rad/s$ |

$$A_{max} = [4.38 \quad 4.80 \quad 4.38 \quad 5.65 \quad 5.65 \quad 7.85 \quad 7.85]^T \, rad/s^2$$
$$J_{max} = [21.9 \quad 24.0 \quad 21.9 \quad 28.3 \quad 38.3 \quad 39.3 \quad 39.3]^T \, rad/s^3$$

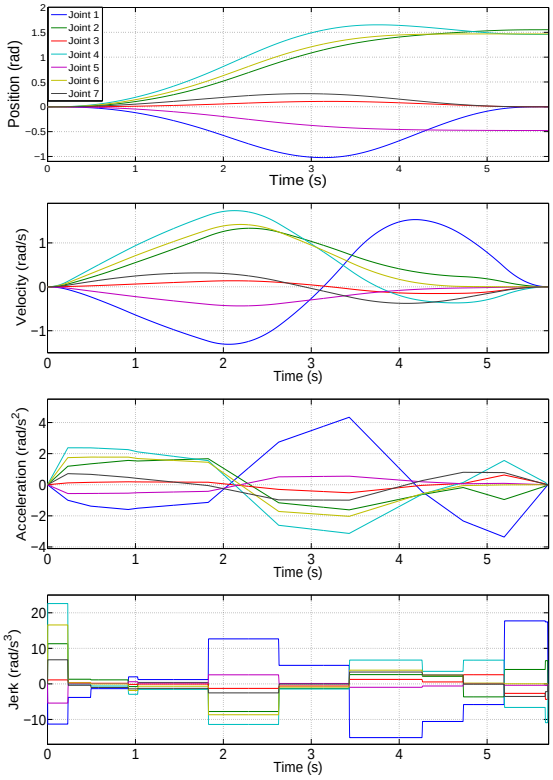



Fig. 6. The position, velocity, acceleration and jerk profile of the calculated trajectory in the robot reaching task

Figure 4 shows that the smoothed path of the end-effector during the reaching task is much shorter and more natural-looking compared to the unprocessed path directly given by the motion planner. Figure 5 illustrates that the execution time is largely reduced by 36.77% on average after 200 short-cutting iterations. Figure 6 illustrates the position, velocity, acceleration and jerk profile on each joint. All the trajectories variables were also checked successfully for the kinematic bounds.

The computation for these smooth trajectories consumes an average time of 4.8 s on an Intel Core$^{(TM)}$2 Quad CPU 2.66GHz machine. Because of our analytical construction, the time in construction of the shortcuts is negligible. Collision checking takes most of the smoothing time. As expected,

(a) (b)

Fig. 7. The planning setup of the ICARO industrial scenario. Left: a global view of the setup. Right: the start configuration.

the collision checker runs the most slowly when the robot passes under the shelf in this experiment.

### B. Robot Experiments

The smoothing algorithm was also applied to a real KUKA light-weight robot IV, which was controlled through the Fast Research Interface [22]. We use a real-world industrial scenario for evaluation. Figure 7 shows the planning setup. The control software was developed using Open Robots tools: GenoM3 [23]. The sampling time was fixed to 10 ms.

In the timing experiments, 50 iterations of shortcutting were finished with a computation time of 1.1 s and a reduced execution time of 2.8 s on average. Results demonstrate that our smoothing algorithm can generate natural-looking motions in clustered environment.

## VI. Conclusions and Future works

This paper presents a fast trajectory smoothing algorithm using third-degree polynomial functions for high-DOF robot manipulators. This algorithm uses a shortcutting heuristic to compute $\mathcal{C}^2$ trajectories that are bounded in velocity, acceleration and jerk. We also present a continuous collision detection algorithm along Soft Motion trajectories. The experimental results with a KUKA arm validate the effectiveness of this algorithm in cluttered human environments.

In future works, we will implement the algorithm on-line to smooth the trajectory during execution. It would also be interesting to consider the dynamic bounds. Moreover, we could attempt to shorten the computation time with faster collision detection algorithms.

## References


[1] Xavier Broquere, Daniel Sidobre, and Ignacio Herrera-Aguilar. Soft motion trajectory planner for service manipulator robot. *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*, pages 2808–2813, Sept. 2008.

[2] Steven M. LaValle and James J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[3] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, Aug 1996.

[4] A. Bicchi and G. Tonietti. Fast and soft arm tactics: Dealing with the safety-performance trade-off in robot arms design and control. *IEEE Robotics and Automation Magazine*, 11(2):22–33, 2004.

[5] T. Kröger. *On-Line Trajectory Generation in Robotic Systems*, volume 58 of *Springer Tracts in Advanced Robotics*. Springer, Berlin, Heidelberg, Germany, first edition, jan 2010.

[6] Sean Quinlan. Real-time modification of collision-free paths. Technical report, Stanford, CA, USA, 1995.

[7] Daniel Sidobre and Wuwei He. Online task space trajectory generation. In *Workshop on Robot Motion Planning Online, Reactive, and in Real-time*, 2012.

[8] Oliver Brock and Oussama Khatib. Elastic strips: A framework for motion generation in human environments. *The International Journal of Robotics Research*, 21(12):1031–1052, 2002.

[9] M. Toussaint, M. Gienger, and C. Goerick. Optimization of sequential attractor-based movement for compact behaviour generation. In *Humanoid Robots, 2007 7th IEEE-RAS International Conference on*, pages 122–129, Nov 2007.

[10] Z. Shiller and S. Dubowsky. Global time optimal motions of robotic manipulators in the presence of obstacles. In *Robotics and Automation, 1988. Proceedings., 1988 IEEE International Conference on*, pages 370–375 vol.1, Apr 1988.

[11] J.E. Bobrow. Optimal robot plant planning using the minimum-time criterion. *Robotics and Automation, IEEE Journal of*, 4(4):443–450, Aug 1988.

[12] Sbastien Lengagne, Joris Vaillant, Eiichi Yoshida, and Abderrahmane Kheddar. Generation of whole-body optimal dynamic multi-contact motions. *The International Journal of Robotics Research*, 32(9-10):1104–1119, 2013.

[13] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, Aug 1996.

[14] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann. Planning collision-free reaching motions for interactive object manipulation and grasping. In *Computer Graphics Forum*, volume 22 of *Comput. Graph. Forum (UK)*, pages 313–22. Blackwell Publishers for Eurographics Assoc, 2003. Robotics Res. Lab., Univ. of Southern California, Los Angeles, CA, USA.

[15] 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*, pages 2493–2498, May 2010.

[16] X. Broquère and D. Sidobre. From motion planning to trajectory control with bounded jerk for service manipulator robots. In *IEEE Int. Conf. Robot. And Autom.*, 2010.

[17] L. Biagiotti and C. Melchiorri. *Trajectory Planning for Automatic Machines and Robots*. Springer, November 2008.

[18] T. Kröger. Online trajectory generation: Straight-line trajectories. *IEEE Transactions on Robotic*, 27(5):1010–1016, 2011.

[19] A.Frisoli, C.Loconsole, R.Bartalucci, and M.Bergamasco. A new bounded jerk on-line trajectory planning for mimicking human movements in robot-aided neurorehabilitation. *Robotics and Autonomous Systems*, 2013.

[20] Fabian Schwarzer, Mitul Saha, and Jean-Claude Latombe. Exact collision checking of robot paths. In Jean-Daniel Boissonnat, Joel Burdick, Ken Goldberg, and Seth Hutchinson, editors, *Algorithmic Foundations of Robotics V*, volume 7 of *Springer Tracts in Advanced Robotics*, pages 25–41. Springer Berlin Heidelberg, 2004.

[21] KUKA Roboter GmbH. Lightweight Robot 4 Operating Instructions, 2008.

[22] G. Schreiber, A. Stemmer, and R. Bischoff. The fast research interface for the kuka lightweight robot. *In Proc. of the IEEE ICRA 2010 Workshop on ICRA 2010 Workshop on Innovative Robot Control Architectures for Demanding (Research) Applications - How to Modify and Enhance Commercial Controllers*, pages 15–21, 2010.

[23] Anthony Mallet, Cédric Pasteur, Matthieu Herrb, Séverin Lemaignan, and Félix Ingrand. Genom3: Building middleware-independent robotic components. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 4627–4632. IEEE, 2010.