Optimal Time-Jerk Trajectory Generation for Robot Manipulators

Jose Avalos and Oscar E. Ramos
Department of Electrical Engineering, Universidad de Ingenieria y Tecnologia - UTEC, Lima, Peru
e-mail: {jose.avalos, oramos}@utec.edu.pe

***Abstract*—This work presents a method to obtain minimum-time smooth motion trajectories for robot manipulators. The strategy is to design the trajectory as a combination of cubic splines for position, velocity and acceleration using seven degree polynomials to ensure that the initial and final conditions are set to zero. The spline is obtained with an optimization based in L-BFGS-B (Limited-memory Broyden–Fletcher–Goldfarb–Shanno) with weighing between the accumulated time or jerk, which are part of the cost equation to let the user generate the trajectory according to the specific task. The proposed method is implemented in a Sawyer robot and, under the premise of precisely passing through the required knots, the spline motion is compared with functions from the manufacturer that directly provide tools for kinematic inversion to show the advantage of our approach.**

I. Introduction

Optimal trajectory planning, which consists in finding an optimal motion law based on a geometric path and satisfying some predefined requirements, is an unsolved problem in robotics. The inputs for a trajectory planning algorithm are usually the geometric path and the kinematic constraints, while the output is the trajectory of the joints expressed as a time sequence of position, velocity and acceleration. In industry, the desired positions of the robot are typically recorded by the user in a mechanical way, so that the task consists in repeating this routine assuming there is no event that will change the pre-configured motion path. Nonetheless, for example in packing tasks, minimal variations in the object position will cause the robot to stop, and a manual reconfiguration will be needed. The problem to use feedback in this case is the generation of the proper path since many industrial robots are redundant and their kinematics has infinite solutions.

To solve that, a basic method includes senddirectly the reference to the manufacturer controller, however, heavily relies on the internal robot controller and does not allow the user to control the execution time or the high-level vibration generated by the jerk, which is the derivative of the acceleration. To solve these problems, an approach based on finding an optimal trajectory can be applied. This approach can be used to minimize jerk and time, to decrease joint position errors, to generate less vibration, to prevent large oscillations, among other objectives. To achieve an efficient trajectory plan, the most important part lies on the interpolation function that is used to obtain the trajectory. Thus, the efficiency of trajectory planning algorithms is limited not only by the required time but also by the physical limitations of the robot to ensure that it achieves the expected results from the algorithm.

978-1-5386-8464-1/18/$31.00

Fig. 1: Sawyer robot at UTEC moving a cup and avoiding a bottle using optimal trajectory generation.

There exist different approaches to determine an optimal trajectory. Gasparetto [1] applied quintic B-spline interpolation to generate smooth joint trajectories: their proposed method allows to impose kinematic constraints, expressed as upper bounds on the absolute values of velocity, acceleration and jerk of the robot joint. Kucuk [2] developed three criteria for serial robots based on cubic splines with some added restrictions to avoid impact between robot joints. In [3], [4], [5], an objective function composed of two terms, where one is proportional to the execution time and the other is proportional to the jerk, is minimized to obtain the optimal trajectory.

This work presents a framework that achieves optimal trajectory using the minimum time-jerk value as an optimization criterion. The paper proposes a motion routine for a comparison between motion purely generated by direct kinematics and motion generated with using the optimal trajectory. The proposed framework was tested using a Sawyer robot as Fig.1 shows. The remainder of this paper is organized as follows: section II explains the methods to achieve optimal motion; and section III shows the experimental results of both methods.

II. Methods

For optimal trajectory, it is crucial to properly define the constraints before solving the cost equation. In this section, we first define the necessary mechanical information for the process, we then describe the cost equation, and finally we explain the mathematical optimizer as the initial point.

A. Sawyer Robot

The Sawyer robot was built by Rethink Robotics to develop advanced robotic application for industrial tasks that involve cooperation with people. At the mechanical level, Sawyer is a redundant cooperative robot with seven degrees of freedom. Joint limits for the position and velocity of the robot are provided by the robot manufacturer and are summarized in Table I. In robot control those values must always be avoided since they will generate problems involving undesired or even unpredictable behavior. For Sawyer, if values exceeding these limits are sent to the robot by a time no longer than 20 ms, they will be ignored by the internal robot controller. However, if the duration is longer, the robot will be stopped by its internal security framework.

TABLE I: Joint limits for the Sawyer robot.

Joint	Upper limit [rad]	Lower limit [rad]	Max. Velocity (rad/s)	Max. ACC (rad/s^2)
J_0	3.0503	-3.0503	1.74	10.0
J_1	2.2736	-3.8095	1.328	8.0
J_2	3.0426	-3.0426	1.957	10.0
J_3	3.0439	-3.0439	1.957	10.0
J_4	2.9761	-2.9761	3.485	12.0
J_5	2.9761	-2.9761	3.485	12.0
J_6	3.14	-3.14	4.545	12.0

It must be pointed out that this work assumes as a starting point that a geometric path is available within the configuration limits of the robot.

Table II: Nomenclature

N	Number of robot joints
α	Weight of the term proportional to the execution time
k	Number of knots
h_i	Time interval between two consecutive knots
H_i	Accumulated time interval from the initial knot to the i-th knot
t_t	Total execution time of the trajectory
f	frequency for the robot time
$Q_j(t)$	Position of the j-th joint based on knots
V_j	Velocity constraint for the j-th joint
$q_j(t)$	Position of the j-th joint
$\dot{q}_j(t)$	Velocity of the j-th joint
$\ddot{q}_j(t)$	Acceleration of the j-th joint
$\dddot{q}_j(t)$	Jerk of the j-th joint

B. Trajectory planning in joint space

Due to the mechanical operation of any robot, time and jerk are inversely related. To achieve a task in minimum time implies that the jerk will increase, unless it is also limited at some point, and vice-versa. Based on the cost equations shown in [4], [5] we define two minimization objectives: (i) the *accumulated time*, defined as the total time multiplied by the number of joints; and (ii) the *accumulated jerk*, defined as the sum of the squared contribution of the jerks throughout time. Using both objectives and the nomenclature shown in Table II, the cost equation C can be written as:

$$C = N \sum_{i=1}^{k \quad 1} h_i + \sqrt{\sum_{i=1}^{N} \int_0^{t_t} [\dddot{\mathrm{q}}_\mathrm{i}(\mathrm{t})]^2 \, \mathrm{d}t} \tag{1}$$

where the first component shows the accumulated time, and the second component the accumulated jerk. Note that a robot with more joints will take more time in its total displacement due to the definition of the accumulated time, where there is a multiplication with the total number of joints N. Also, the values of jerk $\dddot{q}$ are squared since they can take on positive or negative values.

To send intermediate waypoints to the robot, we use knots, which are defined as a given robot configuration in time. They can be specified in the joint space or in the Cartesian space of some operational point such as the end effector, but we will use the former one. As Table II shows, the time between each knot will be denoted as h_i and its accumulated value as H_i; for k knots that is:

$$H_n = \sum_{i=1}^{n \quad 1} h_i, \quad \text{with} \quad H_1 = 0 \tag{2}$$

For some interpolation processes, the desired configuration and time (knots) are required. This time will be represent as $\vec{\boldsymbol{H}}^{\,t} = [H_1, H_2, H_3, ..., H_k]$. The control data sent to the robot needs to be updated with a certain frequency f, and therefore we need discrete values for the joint position, velocity and acceleration, each at sampling time n. For Sawyer, the recommended frequency is $f = 100$ Hz but a higher frequency would lead similar control results, for frequency highest than 500 hz, noise in motion express as vibration is observed. Using the previous definitions, the cost equation in (1) can also be written as:

$$C = NH_k + \sqrt{\sum_{j=0}^{(N \quad 1)} \sum_{n=1}^{w} (\dddot{\mathrm{q}}_\mathrm{j}[\mathrm{n}])^2} \tag{3}$$

where the time for the jerk has been explicitly written as discrete using the samples given by n for the entery value of $w = fH_k$.

C. Spline Definition

For Spline generation, we have to follow both some mechanical rules and some *good practices*. These include, for example, to get an initial and final null value for the velocity, the acceleration and the jerk. We proposed to use a cubic spline due to its low computational cost, but it only allows for the specification of one initial condition. Due to this restriction, we chose to begin the velocity at a null value, which is a type of spline called a *clamped cubic spline*. The spline that we used in this work can be fully specified given a position function $Q(t)$ defined on $[a, b]$, and a set of nodes such that $a = H_1 < H_2 < ... < H_k = b$. A cubic interpolation spline S for Q is a piecewise cubic polynomial S_j on $[H_j, H_{j+1}]$, $\forall j = 1, 2, ..., k-1$, and it can be mathematically defined by (4) where a_i, b_i, c_i, d_i are parameters that need to be found based on the knots.

Although the spline is generates for each joint, all of them follow the same time vector, and therefore, the same time constraints. The algorithm that was used is describe in Alg. 1 and it returns the coefficients for $S(t)$.

$$S(t) = \begin{cases} a_1 + b_1(t-H_1) + c_1(t-H_1)^2 + d_1(t-H_1)^3, & \text{if} \quad H_1 \leq t \leq H_2 \\ a_2 + b_2(t-H_2) + c_2(t-H_2)^2 + d_2(t-H_2)^3, & \text{if} \quad H_2 \leq t \leq H_3 \\ ... \\ a_{k\ \ 1} + b_{k\ \ 1}(t-H_{k\ \ 1}) + c_{k\ \ 1}(t-H_{k\ \ 1})^2 + d_{k\ \ 1}(t-H_{k\ \ 1})^3, & \text{if} \quad H_{k\ \ 1} \leq t \leq H_k \end{cases} \tag{4}$$

Algorithm 1: Clamped Cubic Spline

Input: $\vec{Q}$, $\vec{H}$

Output: $\vec{a}$, $\vec{b}$, $\vec{c}$, $\vec{d}$

Set $\vec{a} = \vec{Q}$

Compute $\vec{h}$ from $\vec{H}$

Compute the tridiagonal matrix A:

$$A = \begin{bmatrix} 2h_1 & h_1 & 0 & \dots & 0 \\ h_1 & 2(h_1+h_2) & h_2 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 2(h_{k\ \ 2}+h_{k\ \ 1}) & h_{k\ \ 1} \\ 0 & 0 & \dots & h_{k\ \ 1} & 2h_{k\ \ 1} \end{bmatrix}$$

Compute $\vec{c}$:

$$\begin{bmatrix} c_1 \\ c_2 \\ ... \\ c_{k\ \ 1} \end{bmatrix} = A^{\ \ 1} \begin{bmatrix} \frac{3}{h_1}(a_2-a_1) \\ \frac{3}{h_2}(a_3-a_2) - \frac{3}{h_1}(a_3-a_2) \\ ... \\ -\frac{3}{h_{k\ \ 1}}(a_k - a_{k\ \ 1}) \end{bmatrix}$$

Compute $\vec{b}$ and $\vec{d}$: $\forall i = 1, ..., k-1$

$$b_i = \frac{1}{h_i}(a_{i+1}-a_i) - \frac{h_i}{3}(c_{i+1}+2c_i)$$

$$d_i = \frac{1}{3h_i}(c_{i+1}-c_i)$$

To ensure a null initial and final velocity, acceleration and jerk, a seven degree-of-freedom equation is set. The coefficients of this polynomial are determined by the position, velocity and acceleration generated by the cubic spline at the first and last knots. These polynomials are added in the optimization process since the high grade in the equation could generate high jerks. It is also possible to change the whole cubic spline with a 7th grade spline, but the computational cost will drastically increase.

D. Time Optimization

For optimization, we begin assuming that the velocity constraint for all the joints, $V_j,\ \forall j = 0 : N-1$ must not be exceeded. We then set the difference between two consecutive position joints defined by the knots as

$$q_j^{i+1} - q_j^i = \Delta q_j^i, \quad \forall i = 1 : k-1$$

where the total number of knots is k. The ratio between this variation and the maximum velocity determines the minimum time required for the joint. As there are multiple joints, the maximum value of this group will represent the minimum effective time h_i^{min} that it takes the robot to move between two configurations:

$$h_i^{min} = \max_{j=0,...,N\ \ 1} \left\{ \frac{\Delta q_j^i}{V_j} \right\}. \tag{5}$$

These minimum values are then stored in $\boldsymbol{H}_{min}^t$, as (2), which will be optimized as introduced with the vector time in section II-C. If the motion between two consecutive knots requires more than 200 ms, we introduce *support knots* which are linear values between the two knots in order to get more points to interpolate:

$$\vec{\boldsymbol{H}}_{min}^{\ \ t} = \left[0, H_2^{min}, H_3^{min}, ..., H_k^{min}\right]$$

E. Method vector

Using vector $\vec{Q}_i$, which contains the configuration values from the knots, we can obtain the value of $\boldsymbol{H}_{min}^t$. For optimization, we propose to multiply this vector with $\vec{a}$ to obtain:

$$\vec{\boldsymbol{H}}_{opt}^{\ \ t} = \vec{\boldsymbol{a}}\ \vec{\boldsymbol{H}}_{min}^{\ \ t},$$

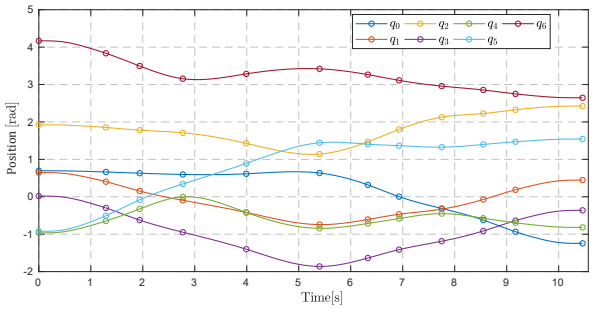
and we minimize the following cost equation:

$$\min_{\boldsymbol{a}} a\alpha N H_k^{min} + (1-\alpha)\sqrt{\sum_{j=0}^{N\ \ 1} \sum_{n=1}^{(afH_k^{min})} (\dddot{\mathrm{q}}_{\mathrm{j}}[\mathrm{n}])^2} \tag{6}$$

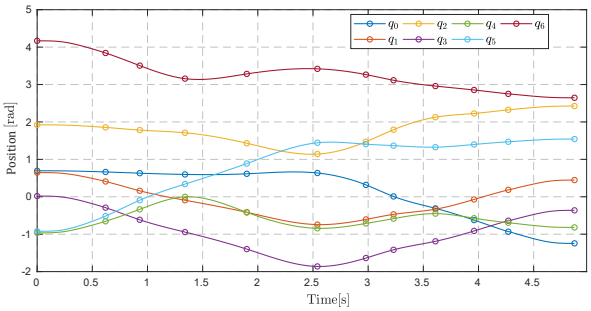
It should be noted that this problem has a bound restriction for any element of $\vec{a}$ that is inside $[1, \infty >$ that multiply the minimum time define in (5), optimize with some tolerance such as $1 \times 10^{\ \ 3}$. The value for $\alpha =< 0.1,\ 0.95 >$ must be explicitly introduced by the user and is used in the training stage. The cost that minimizes the cost function, will then be found using a BFGS optimizer from the numpy library for Python, since it provides better time results over other more classical optimization methods such as SLSQP and TNC, also available from the same library. The values of $\vec{a}$, $\vec{H}_{opt}^{\ \ t}$ are then obtained and introduced to the spline to calculate the jerk $\dddot{q}_j[n]$ at the n sample points. This process is iterative, and it is repeated until the optimizer finds the minimum value.

III. Results

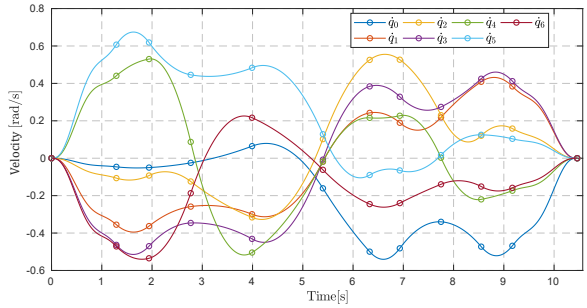
We performed experimental results using the real Sawyer robot available at UTEC. We will execute two approaches that will serve as comparison tests: the first one uses the trajectory optimization, and the second one uses the output of the kinematics directly. Both approaches will be applied to the task explained in Fig. 4. The joint positions were obtained



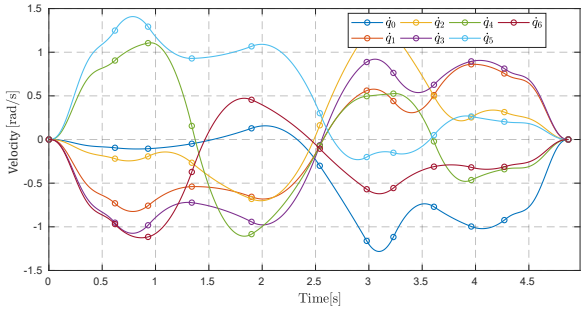
(a) Optimal position generated with α = 0.2



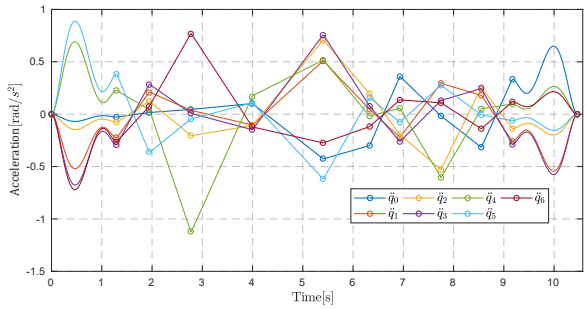
(b) Optimal position generated with α = 0.8



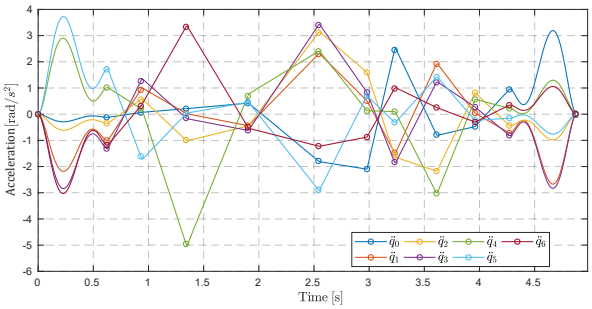
(c) Optimal velocity generated with α = 0.2



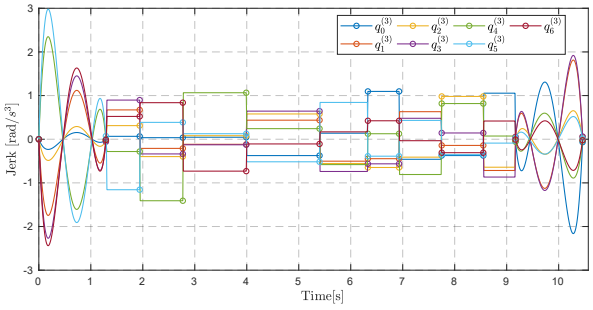
(d) Optimal velocity generated with α = 0.8



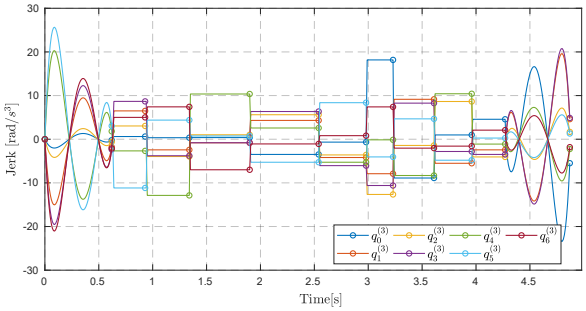
(e) Optimal acceleration generated with α = 0.2



(f) Optimal acceleration generated with α = 0.8



(g) Optimal jerk generated with α = 0.2

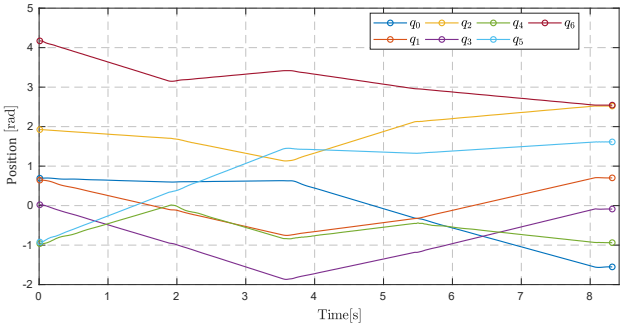


(h) Optimal jerk generated with α = 0.8

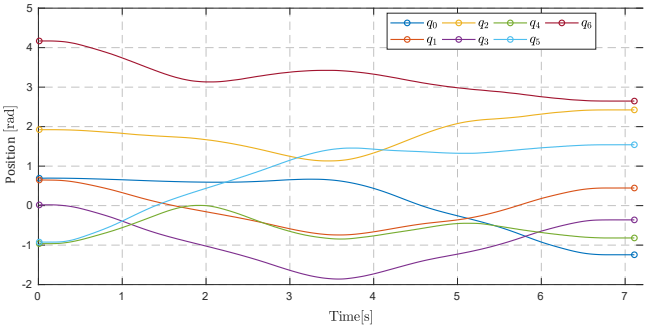
Fig. 2: Motion generated by optimal trajectory with α = 0.2 and α = 0.8.

using the inverse kinematics server provided by the Sawyer robot for specified poses. The joint trajectory control set to the robot requires the specification of the position, velocity and acceleration, all sent with a frequency of 100 Hz for these experiments.

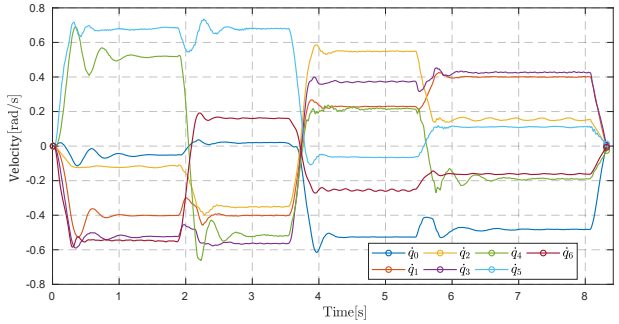
Following the cost equation introduced in (6), values generated with $\alpha = 0.2$ and 0.8 are shown in Fig. 2. Comparing the result, in the first case, the motion takes around 10.5 s. while the second took 5 second less, this demonstrates how the factor α has an influence over the behavior of the cost equation by accumulating time and jerk. About the peak values, for jerk with $\alpha = 0.8$ is around eight times more over the case. This



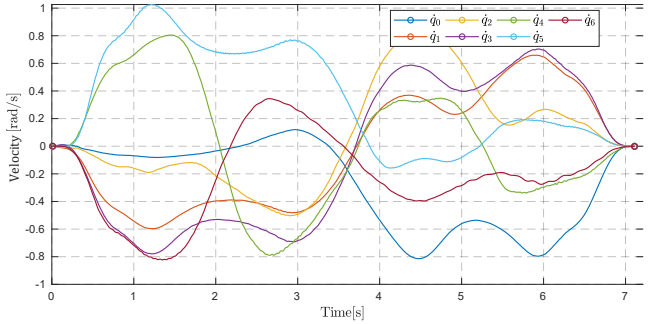
(a) Position record using direct kinematics.



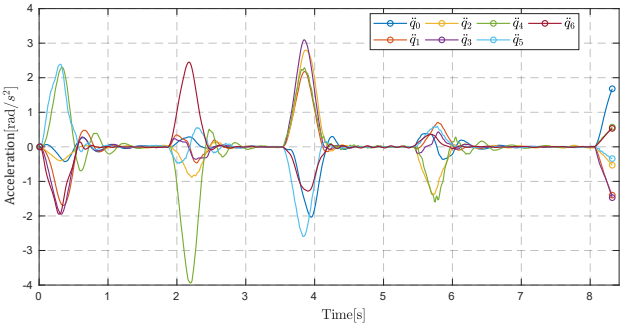
(b) Position record for optimal trajectory with α=0.5.



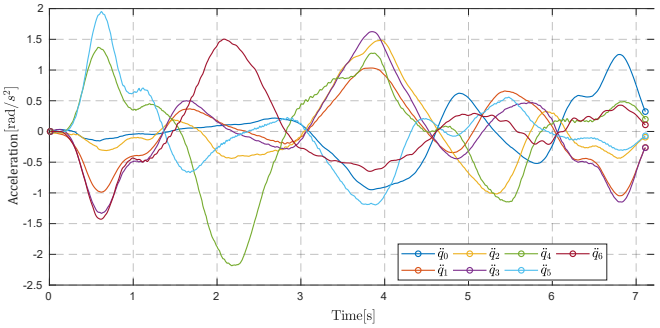
(c) Velocity record using direct kinematics.



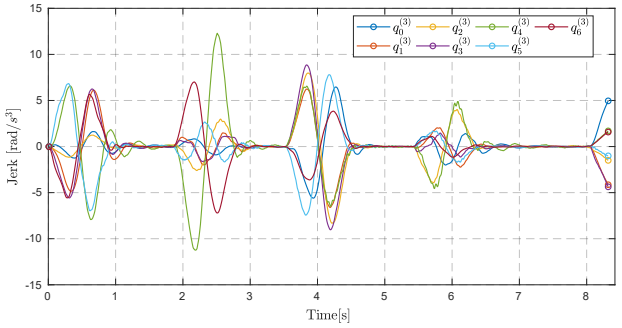
(d) Velocity record for optimal trajectory with α=0.5..



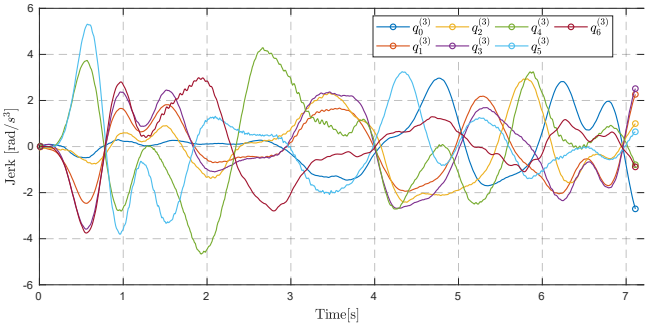
(e) Acceleration record using direct kinematics.



(f) Acceleration record for optimal trajectory with α=0.5.



(g) Jerk record using direct kinematics.



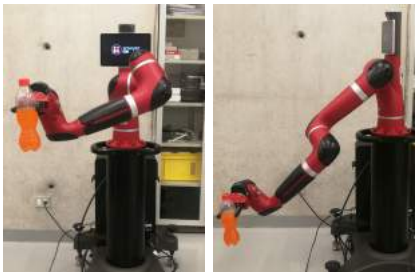
(h) Jerk record for optimal trajectory with α=0.5

Fig. 3: Comparison between direct kinematics and optimal trajectory with α=0.5.

effect is repeated for level of acceleration and velocity but in both cases the motions meets with the knots setting for the task. For different values of α we could messure a correlation between position and time. A summary showing the effect of different values of α is depicted in Fig. 5

When dealing with the robot, we read the information from a ROS topic which provides the joint position and velocity from the internal sensors. To obtain the acceleration and

(a) Position A (b) Position B (c) Position C



(d) Position D (e) Position E

Fig. 4: Proposed motion that executes the sequence A-B-C-D-E. A bottle with liquid provides a fast visual inspection about the vibration (jerk) in the robot motion.

jerk, the Euler method was used with a windowed filter to reduce noise. The optimal approach is compared with the direct application of kinematics with $\alpha = 0.5$ in Fig. 3. It is evident that the position is more continuous for the optimal motion. Although the time is shorter for optimization, its peak values are lower in velocity, acceleration and jerk. For position, the displacement in the first case is linear compare to the continuity in our propose method. About velocity, for each knot space, the manufacture controller try to set a constant velocity generating high acceleration rates between each knot, compare to more longer space variation in optimal trajectory generation. That could be demonstrate in an acceleration comparative, in the first case there are some section with values at 0 complemented with constantly high peak values which is two times the peak value in our method. Finally, vibration is related with jerk, both cases have near execution time, normally, we hope get related values for jerk, however, the jerk for direct kinematic show double values respect to the optimization trajectory method that prove our method reduce considerable the jerk and also vibration . A summary of the real value for time and jerk are presented in Fig. 6, where the direct kinematics offers just a point compared to the optimal approach which give a curve of possibilities according to the user application. In fact, if the user priority is reduce drastically the vibration in the task a $\alpha = 0.1$ is set but if we want to use the minimal time available $\alpha = 0.95$ is used.

IV. Conclusion

This paper shows that using optimal trajectory generation there is a compromise between the jerk and the time, which can be determined according to the objective of the application

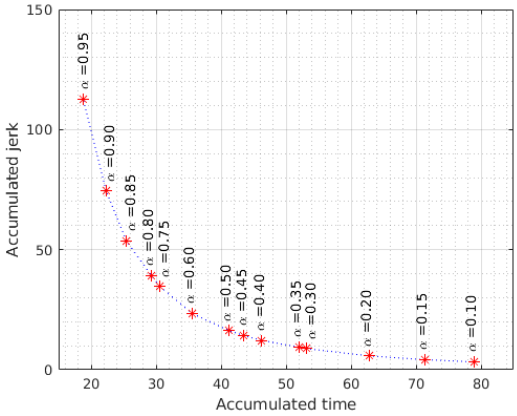


Fig. 5: Simulated behavior of α for accumulated time and jerk.

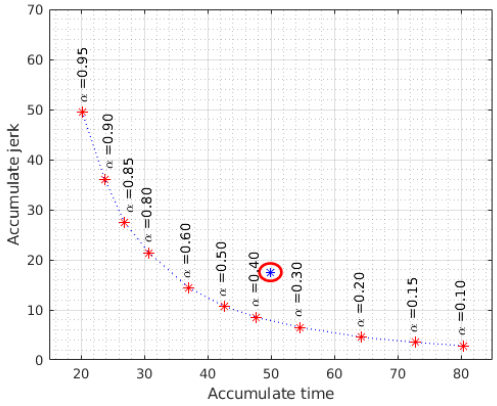


Fig. 6: Experimental behavior of α for accumulated time and jerk. The Blue point represents the direct kinematics case.

using a parameter in the equation cost implemented in robot Sawyer. The approach also allows for the selection of different smooth behaviors that are not achievable using only kinematics witch generate the motion with minimum value in time or jerk (related to vibration). Further work involves exploring other interpolation techniques, cost functions and introduce advance motion planning in order to develop real industrial task.

References

[1] A. Gasparetto and V. Zanotto, "Optimal trajectory planning for industrial robots," *Advances in Engineering Software*, vol. 41, no. 4, pp. 548–556, apr 2010.

[2] S. Kucuk, "Optimal trajectory generation algorithm for serial and parallel manipulators," *Robotics and Computer-Integrated Manufacturing*, vol. 48, pp. 219–232, dec 2017.

[3] T. Chettibi, H. Lehtihet, M. Haddad, and S. Hanchi, "Minimum cost trajectory planning for industrial robots," *European journal of mechanics. A, Solids*, vol. 23, no. 4, pp. 703–715, 2004.

[4] A. Gasparetto and V. Zanotto, "A technique for time-jerk optimal planning of robot trajectories," *Robotics and Computer-Integrated Manufacturing*, vol. 24, no. 3, pp. 415–426, jun 2008.

[5] J. Huang, P. Hu, K. Wu, and M. Zeng, "Optimal time-jerk trajectory planning for industrial robots," *Mechanism and Machine Theory*, vol. 121, pp. 530–544, mar 2018.