Towards an Intuitive and Iterative 6D Virtual Guide Programming Framework for Assisted Human-Robot Comanipulation

Susana Restrepo, Gennaro Raiola, Joris Guerry, Evelyn d'Elia, Xavier Lamy, Daniel Sidobre

doi: xxx

Towards an Intuitive and Iterative 6D Virtual Guide Programming Framework for Assisted Human-Robot Comanipulation

Susana Sánchez Restrepo[1,2], Gennaro Raiola[1,3] *, Joris Guerry[4], Evelyn D'Elia[3], Xavier Lamy[1] and Daniel Sidobre[2]

1 *Interactive Robotics Laboratory (LRI), CEA-List, Gif-sur-Yvette, France*
2 *LAAS-CNRS, University of Toulouse, CNRS, UPS, Toulouse, France*
3 *Department of Advanced Robotics, Istituto Italiano di Tecnologia, Genoa, Italy*
4 *EDF R&D, Chatou, France*

SUMMARY
In human-robot co-manipulation, virtual guides are an important tool used to assist the human worker as they constrain the movement of the robot to improve the task accuracy and to avoid undesirable effects, such as collisions with the environment. Consequently, the physical effort and cognitive overload is reduced during accomplishment of co-manipulative tasks. However, construction of virtual guides often requires expert knowledge and modeling of the task, which restricts the usefulness of virtual guides to scenarios with fixed constraints. Moreover, few approaches have addressed the implementation of virtual guides enforcing orientation constraints, and when done, these approaches have treated translation and orientation separately, and consequently there is no synchronization of the translational and rotational motions. To overcome these challenges and enhance the programming flexibility of virtual guides, we present a new framework that allows the user to create **6D virtual guides** through *XSplines* which we define as a combination of Akima splines for the translation component, and spherical cubic interpolation of quaternions for the orientation component. For complex tasks, the user is able to initially define a 3D virtual guide and then use this assistance in translational motion to concentrate only on defining the orientations along the path. It is also possible for the user to modify a particular point or portion of a guide while being assisted by it. We demonstrate in an industrial scenario that these innovations provide an intuitive solution to extend the use of virtual guides to 6-DOF and increase the human worker's comfort during the programming phase of these guides in an assisted human-robot co-manipulation context.

1. Introduction

Using virtual guides reduces the user's physical effort and cognitive overload during the execution of a task with a collaborative robot.[1,2] This kind of assistance can be used in industry on manipulation tasks such as insertion, assembly, cutting, drilling and polishing. As a reminder, this assistance superposes a synthetic force onto forces perceived by the user through a haptic control interface on the robot. The applied force constrains the user's movements and consequently those of the controlled robot through a particular trajectory, a surface or limited volume, ensuring motion guidance during the task completion.

An example of virtual guide from everyday life is the ruler, which allows the user to draw very straight lines by constraining the movement of the pen tip along a 1-D trajectory on the 2-D paper.

* E-mail: gennaro.raiola@gmail.com

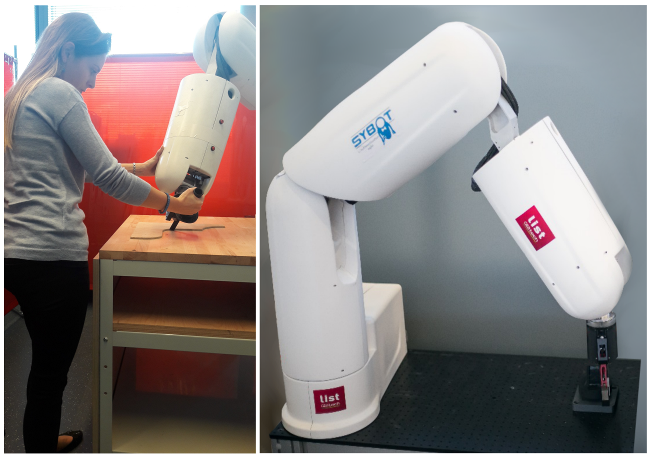


Fig. 1: (Left) Experimental setup for user study. (Right) ISybot 6-DOF Collaborative robot.

While using the kinesthetic programming technique[3] to create complex virtual guides in space and constrain both the Cartesian position and orientation of the tool, high levels of user concentration and effort may be required, which could lead to trajectory encoding errors.

Thus, we propose to **simplify the programming process by uncoupling Cartesian positions and orientations during the demonstrations**.

However, few approaches have addressed the implementation of virtual guides enforcing orientation constraints, and when done, they have treated translation and orientation separately so there is no synchronization of the translational and rotational motions. The main contribution of this paper is the novel modeling and construction of 6D virtual guides by introducing the concept of **XSplines** which are created through a combination of Akima Splines for the position and spherical cubic interpolation of quaternions for the orientation.

Another known challenge with virtual guide assistance is programming the virtual guides on the robot. At least one of the following requirements must be met:

- expert knowledge of the task,
- high technical expertise,
- modeling of the task,
- good programming skills.

Some of these requirements restrict the usefulness of virtual guides to scenarios with unchanging constraints. However, many industrial applications require multiple operations to perform a task which requires easy generation and modification of guides in order to adapt to different situations, such as:

- necessary changes on the work part (polish, sand, cut),
- low production volumes due high variability of the product,
- transporting an object to one of multiple possible positions or
- performing sub-tasks in a different order depending on the availability of tools.

For these reasons, we propose a **kinesthetic teaching framework** to program virtual guides in an **intuitive and flexible way** so they can be used by **non-robotics experts** and be **easily reprogrammed** when needed.

Despite its strengths, there are also some limitations when using kinesthetic teaching. The remaining inertia and articular friction of collaborative robots (cobots) – even after application of gravity compensation and inertia masking techniques – can disrupt the fluent execution of movements and hinder their manipulation. For example, it is difficult to impose a rectilinear trajectory because the robot tends to follow curved trajectories due to joint space friction. Also, it is sometimes difficult to perform slow and small displacements precisely. In order to keep the desired trajectory, the user must exert additional effort and concentration.

Moreover, most Programming by Demonstration (PbD) approaches using kinesthetic teaching need several demonstrations to encode a trajectory. This could be exhausting for the user and detract from the efficiency of using virtual guides.

Thus, we suggest that **the operator must be assisted during the programming phase**. To this aim, we propose to **use virtual guide programming in an iterative way, which allows for easier refinement and modification**.

Thanks to this novel iterative programming approach using **XSplines**, we are able to prove in a simulated scenario with a collaborative robot that the cognitive load of users is reduced and the user comfort is increased during the teaching process.

To summarize, the novel contributions of our work are:

- construction of orientation constraint using quaternions for spline interpolation;
- 6D virtual guide definition with separation of translation and orientation;
- iterative programming of virtual guides while assisting the user;
- experimental evaluation.

The rest of this work is structured as follows. In the next section, related work is discussed. In Section 3, the implementation of virtual guides is presented. Multiple ways to define the virtual guide constraints based on position, orientation, and pose are presented in Sections 4, 5 and 6. In Section 7 we present the iterative programming framework for 6D virtual guides. This new iterative approach is validated with a 6-DOF ISybot robot, and the results are presented in Section 8. Finally, this work is discussed and concluded in Section 9.

2. Related Work

2.1. Definition of virtual guides

Virtual guides are used to passively enforce virtual constraints on the movements of cobots, in order to assist the user during a collaborative task. Virtual guides were first introduced by Rosenberg[4] as *Virtual Fixtures.* The fundamental concept is that virtual fixtures can reduce mental workload, task time and errors during teleoperated manipulation tasks. After Rosenberg's initial work, the use of virtual fixtures has been extended to robotic surgery under the name of *active constraints*[5] and to industrial applications in the context of *Intelligent Assist Devices.*[6] Nowadays, virtual fixtures have been featured in several different works, but unfortunately "there is currently no definitive concept which unifies the field"[7] because of the different definitions, applications and implementation methods. Generally, virtual fixtures have been used in teleoperation,[89] or co-manipulation contexts.[1] The type of assistance offered by the virtual fixtures can vary among different definitions,[10] but in general, they are either used to guide the user along a task-specific pathway[11] or to limit the user to move the robot within a safe region.[12] In the first case, we refer to the virtual fixtures as *virtual guiding fixtures* or simply as *virtual guides.*

The particular implementation of virtual guides we use is based on the work presented by Joly,[13] where a passive virtual mechanism is connected to the robot end-effector by a spring-damper system in a teleoperation context. Virtual mechanisms have also been used in,[14] where they are called *proxies.* In addition, virtual guides have been implemented by using anisotropic admittances to attenuate the non-preferred user force components.[2] These methods require sensing external inputs, such as the force or the velocity applied by the user on the robot end-effector. This is not required in our control scheme.

2.2. Construction of virtual guides

There are many possible solutions to construct virtual guides. Usually, their creation method is strictly related to the goals of the final application. In general, virtual guides have often been limited to pre-defined geometric shapes[11] or combinations of shapes[15],[16] well-defined geometric models,[13] high level task models,[9] or shapes defined through vision based algorithms.[17] In a co-manipulation context, it is more natural to program virtual guides in the real workspace rather than in a simulated one. Therefore, Programming by Demonstration

(PbD)[15,18] appears as a promising strategy to program robots in a fast and simple way when the task is known by the user. During PbD, the operator can directly manipulate the robot end-effector to teach a desired movement.

In our previous work,[3] we proposed a framework for multiple probabilistic virtual guides where kinesthetic teaching and Gaussian Mixture Models (GMM) were used to implement virtual guiding fixtures. This probabilistic framework involves modeling a demonstrated set of guides with GMM and retrieving a generalized representation of the data set using Gaussian Mixture Regression (GMR). Unfortunately, with the mentioned methods, a compromise must be made between the number of demonstrations that demand time and effort and the level of information in the training data.

In the same vein, the authors of[19] treated the problem of incremental kinesthetic learning of motion primitives based on Hidden Markov Models (HMM). The main goal is to transfer human skills to a humanoid robot that will individually evolve later. This transfer is done using a combination of observational learning and kinesthetic teaching to obtain natural whole body motions. However, the motion refinement tube used in this work is different from the trajectory virtual guide teaching we target in our programming approach. Both approaches are similar on the concept of assistance during the teaching phase and the idea of iterative refinement. However, our approach seeks the simplest way to program a virtual guide while giving the whole responsibility of the gesture to the human worker.

2.3. Orientation constraints

2.3.1. Programming by demonstration literature. In PbD approaches, multivariate Gaussians are widely used to encode robot behaviors. Such approaches do not provide the ability to properly describe end-effector orientation, as the distance metric in the orientation space is not Euclidean. In,[20] the authors present an extension of common probabilistic imitation learning techniques to Riemannian manifolds. This work shows the importance of being able to represent end-effector orientations from user demonstrations, coupled with Cartesian positions. However, it does not address the creation of virtual guides. Thus, the work of[20] is useful to extend *probabilistic virtual guides*[3] to a more general framework using also orientation representation.

Other PbD approaches have focused on teaching also the required stiffness of the task. The authors of[21] present interfaces that allow a human operator to indicate compliance variations during task execution by physical interaction. The difference between this approach and ours is that the authors used an interface on a collaborative robot which represents a supplementary cost for the system. In our work, we present a simple framework that can be used without additional devices or sensors on a low-cost collaborative robot. Also, their work targets tasks where stiffness variation during the task is of major importance which makes it difficult to compare with our virtual guide programming framework. Moreover, their approach does not consider rotations.

2.3.2. Virtual guide literature. Due to the task-dependent nature of virtual guides, most experiments have tried to address the most general scenarios for their applications. Many of these applications mainly consisted of general tasks such as path following, targeting and object avoidance exercises in two dimensional environments[22,23] and in more complex 3D environments.[16,24] Fewer applications have addressed the implementation of virtual guides enforcing orientation constraints.[25,26] Traditional virtual guide implementations deal with rotation and translation separately in $\mathbb{R}^3$ space; the interconnection between them is not usually represented in the virtual guide design. The method presented in[25] uses *preferred direction virtual guides*, with an admittance control architecture, to constrain the user to follow a curve, surface or orientation. In the experimental section of this work, rotational and translational constraints are implemented separately. *Autonomous error compensation* was used in[27] to overcome human-related difficulties in simultaneously controlling the position and orientation of a 6-DOF robot under the vision-based *preferred direction virtual guides* presented in.[25] The system uses computer vision to generate a reference trajectory, and the virtual guide control algorithm then provides haptic feedback for implementing direct shared manipulation. The authors found that in a system with both position or orientation reference direction fixtures, only position or orientation would be effectively constrained at a time. They stated that this is due to the translational and rotational components of motion being decoupled from each other in such a way that the user, focusing on moving one, will not notice an error in the other. These works have addressed a different type of virtual guides enforcing orientation constraints than those used in this work. They also address translation and orientation separately. In the case of Dynamic Virtual Guides (DVG), the work of[28, 29] extended dynamic frictional constraints to enforce the position or the orientation of a tool. In contrast to our definition of virtual guides, DVGs are applied to environments which deform or move over time (e.g., soft tissue in the context of robot assisted surgery), and the constraints are not based on virtual mechanisms but on elasto-plastic friction models. However, the definition of the position constraints in $\mathbb{R}^3$ and orientation constraints in $\mathbb{SO}(3)$ are done independently, so there is no synchronization of the translational and rotational movements, which is one of the central contributions of our work.

In,[30] the structure of geometric and dynamic constraints of reference tasks is analyzed using screw theory. Virtual guides using screw theory unify rotational and translational constraints into one set. The spatial compliance/stiffness matrix synthesis for admittance and impedance controlled devices was also studied. In more recent work,[31] the authors studied the application of virtual guides in the deforming environment, and proposed a novel framework of DVG for admittance-type devices. This framework in the Euclidean Group $\mathbb{SE}(3)$ was proposed to enhance the surgical operation accuracy of admittance-type medical robotics in the deforming environment. This approach unites rotation and translation in a compact form.

These aforementioned approaches show the importance of coupling translation and orientation for virtual guide construction. However, they are based on dynamic virtual guides which are different from our non-dynamic definition based on virtual mechanisms. Nevertheless, our work is similar to,[31] in the choice of orientation representation and distance definition in $\mathbb{SE}(3)$.

2.4. Modification of virtual guides

The virtual guides obtained with the mentioned PbD approaches cannot be modified online. If the task changes, the operator must make a new set of demonstrations with the robot to obtain a new task representation. To overcome this lack of flexibility, in the two approaches presented by Rozo et al.[32] and Aarno et al.,[15] the robot is able to automatically adapt. In the first approach, this is done for tasks where initial and end points are more relevant than the trajectory itself. In the second, the fixtures are made flexible and adaptive by decomposing the trajectory into straight lines. The probability that the user is following a certain trajectory is estimated and used to automatically adjust the compliance of the virtual guide. Similarly to that approach, we previously explored in[3] an iterative method combining an incremental training and clustering of GMMs, but, given the probabilistic nature of GMMs, multiple complete demonstrations are still needed to correctly modify the guides.

From another point of view, the authors of[33] introduced the concept of *collaborative learning* to design ergonomic virtual guides for a tricycle cobot and adapt motion to changes in the environment. PbD is used to teach the cobot a path to follow. A dedicated GUI path editor is provided for offline definition and modification of guide paths. In the same manner, it was suggested in[34] that interaction in a PbD context could be improved by including a GUI in the programming loop to show the learned information. A relevant difference between the approaches of Boy et al.[33] and Mollard et al.[34] is that the second approach intends to optimize the learning of a task aimed to be automatically reproduced by the robot, while the first approach uses the operator not only as a part of the teaching phase but as a part of the task execution. Thus, motion guidance is not implemented in.[34] In our approach instead, we suggest assisting the user throughout the teaching process. At first, virtual guide assistance is created using PbD with only one demonstration or preprogrammed trajectory. Afterwards, the user is able to modify the generated guide while the assistance is active by changing a single point or section of it.

Another iterative method is proposed in.[35] In this work the authors propose to generate motion primitives through a variable stiffness impedance controller. The idea is to increase the stiffness after each demonstration, until the motion primitive is fully learned. In this way, the first primitive is used as a guide for the following demonstrations. Although the idea of assisting the human with an active guide during the demonstrations is similar to ours, the two frameworks differ on their final goal: in our work, we create virtual guides to assist the human operator during the task execution, instead, in,[35] the guide is used ultimately to teach the robot to autonomously perform the task. The authors of[36] also proposed a human-in-the-loop approach. However, there is no physical human-robot interaction since a haptic device is used to teleoperate the robot. The advantage of teleoperation is that it allows a human operator to program the robot by demonstration without being in the same physical location. This is particularly needed for robots working in hazardous environments. However, in the case of collaborative robots, using teleoperation adds more complexity and cost to the system. In addition, since the virtual guides are not programmed into the real robot workspace, it is possible to introduce position errors by the fact that the model may be misaligned with reality. Moreover, the required information about the environment is interpreted at the slave robot level, then transmitted to the master device, and finally presented to the user by haptic feedback. This makes the process unintuitive. As in,[35] this approach assists the user during the teaching process by modulating the robot compliance based on the given task requirements. However, the final goal is to teach a task that will be automatically reproduced by the robot, so using motion primitives makes more sense in this case. In our context, we want the user to be the master of the teaching process since our framework relies on a human's expert knowledge of the task. XSplines are therefore a more intuitive way for the teacher to encode exactly the desired trajectory. Moreover, our method can generate a guide with only one demonstration. If the task is complex, the user can teach the translation movements first, and then while being assisted by the robot, teach the orientations. The user can also refine the guide later without needing to perform full demonstrations of the trajectory.

Under this perspective, the work in,[37] is similar to our ideas since the virtual guides are generated through a penalized regression spline fitting algorithm which can be successively adapted online by recording new points. The main difference between this approach and ours is that our framework allows separation between rotations and translations which is useful to teach complex virtual guides in space. In this paper we show that decoupling the teaching phase into two programming steps (translation and then orientation) results in a more comfortable way for the worker to program the guide without loosing efficiency. The user is also able to modify only one point or portion in translation or orientation without modifying (and even while constraining) the other and generate a new virtual guide ensuring synchronization of both motion components. Finally, in,[38] a method was proposed to make the kinesthetic teaching easier by assisting the user during teaching using virtual tool dynamics.[39] However, the amount of assistance is gradually increased based on the accumulated demonstrations. Therefore, several demonstrations are still needed to refine the task before getting the correct assistance. Moreover, after each iteration, it is the robot who chooses an assistance and not the operator who decides where to refine the trajectory, which might be counter-intuitive to the user. One of the advantages of our method is that the operator is the master of the teaching and decides when and where a trajectory modification has to be done.

3. Virtual Mechanisms as Virtual Guides

To implement the virtual guides, we use the concept of virtual mechanisms presented by Joly.[13] In this section we summarize the definition of virtual mechanisms and formalize our implementation of virtual guides via these mechanisms. Our previous work presented this implementation for 3D virtual guides,[40] and we now present it for 6D virtual guides.

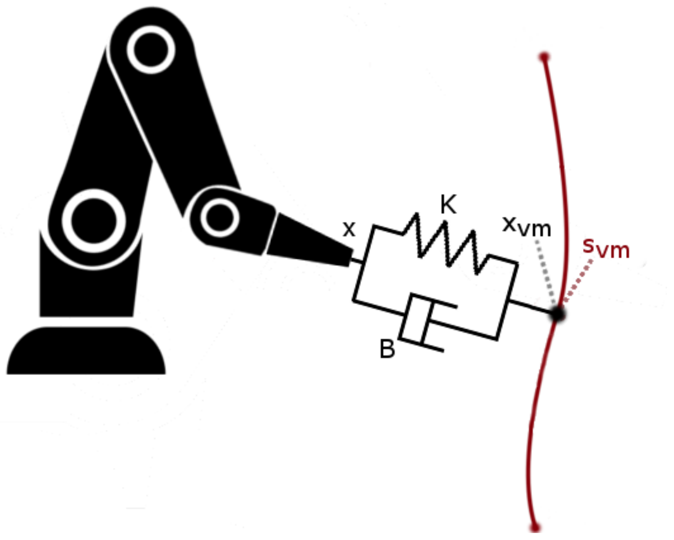


Fig. 2: Virtual mechanism representation. The red curve represents the possible configurations of the virtual mechanism in the Cartesian space X_{vm}, and because of the spring-damper system linking, it represents the allowed configurations of the robot end-effector X. The current position of the virtual mechanism is described by its parameterized space by the parameter $s_{vm} \in \mathbb{R}$.

Our implementation relays on the idea that the cobot end-effector is virtually connected to a virtual mechanism through a spring-damper system. We use this concept to constrain the movements of the cobot to a 6D path defined by position and orientation constraints obtained by kinesthetic teaching. The cobot end-effector and the virtual mechanism are coupled by a spring-damper system which corresponds to a *proportional-derivative* controller whose coupling gains are the stiffness K and the damping B, as shown in Figure 2. If the cobot end-effector moves, the virtual mechanism is pulled along the path in the direction of the movement; also, the virtual mechanism pulls the cobot end-effector towards the path, since the linking acts in both directions. The general effect is that the cobot end-effector can be moved easily along the constraining path, but not away from it.

Notation: The Cartesian pose and velocity of the virtual mechanism and the cobot end-effector are described by $\{X_{vm} \in \mathbb{SE}(3),\ \dot{X}_{vm} \in \mathbb{R}(6)\}$ and $\{X \in \mathbb{SE}(3),\ \dot{X} \in \mathbb{R}(6)\}$ respectively. The Cartesian poses are defined by the translational and rotational components of both virtual mechanism and cobot end-effector displacement as:

$$X_{vm} \triangleq \begin{Bmatrix} X_{vm,trans} \in \mathbb{R}^3 \\ X_{vm,rot} \in \mathbb{SO}(3) \end{Bmatrix}$$

$$X \triangleq \begin{Bmatrix} X_{trans} \in \mathbb{R}^3 \\ X_{rot} \in \mathbb{SO}(3) \end{Bmatrix}$$

To avoid singularity in the representation of rotations $(X_{vm,rot}, X_{rot})$, we use unit quaternions.[41] A quaternion q can be considered to be the association of a scalar $w \in \mathbb{R}$ and a vector $\mathbf{a} \in \mathbb{R}^3$:

$$q \triangleq \begin{bmatrix} w \\ \mathbf{a} \end{bmatrix}$$

The Cartesian velocities are defined by twists $\in \mathbb{R}(6)$ describing the instant movements of the robot end-effector relative to the robot base. We choose to reduce the twists in the center of the robot end-effector. The twists are then formed by the translational and rotational components of the time derivatives of both virtual mechanism and cobot end-effector displacements:

$$\dot{X}_{vm} \triangleq \begin{Bmatrix} v_{vm} \in \mathbb{R}^3 \\ \omega_{vm} \in \mathbb{R}^3 \end{Bmatrix}$$

$$\dot{X} \triangleq \begin{Bmatrix} v \in \mathbb{R}^3 \\ \omega \in \mathbb{R}^3 \end{Bmatrix}$$

The current position of the virtual mechanism is described by its parameterized space $s_{vm} \in \mathbb{R}$, and the evolution of the virtual mechanism is described by $\dot{s}_{vm} \in \mathbb{R}$.

The direct geometric and kinematic models of the virtual mechanism are defined by L_s and J_s respectively. The geometric model allows to determine the pose of the virtual mechanism X_{vm} according to the configuration of its links (in this case represented by s_{vm}), while the kinematic model allows to determine the velocity of the virtual mechanism $\dot{X}_{vm}$ according to the evolution of it (represented by $\dot{s}_{vm}$).

Geometric model:

$$X_{vm} = L_s(s_{vm}) \tag{1}$$

Kinematic model:

$$X_{vm} = f(s_{vm}) \tag{2}$$

$$\dot{X}_{vm} = J_s \dot{s}_{vm} \tag{3}$$

where $J_s[6 \times 1]$ is the virtual mechanism's Jacobian, as defined in.[42]

Force on the cobot end-effector: The force F_c applied by the spring-damper system on the cobot is given by:

$$F_c = K(X_{vm} - X) + B(\dot{X}_{vm} - \dot{X}) \tag{4}$$

Where $K \in \mathbb{R}(6 \times 6)$ and $B \in \mathbb{R}(6 \times 6)$ are diagonal matrix. $K_{trans} \in \mathbb{R}(3 \times 3)$ represents the stiffness gain in translation and is a symmetric definite positive matrix.

The notation $(X_{vm} - X)$ may be abusive for rotations. More adequate angular error computation can be used without restrictions.

$$X_{vm} - X \triangleq \begin{Bmatrix} X_{vm,trans} - X_{trans} \\ \delta(q_{vm}, q) \end{Bmatrix}$$

The Jacobian of the cobot is given by J $[6 \times n]$ where n represents the number of DOFs of the robot. We use the transposed Jacobian J^T to transform the forces into a torque reference for the controller.[42] The torque applied to the joints of the cobot is described by τ_c:

$$\tau_c = J^T F_c \tag{5}$$

Force on the virtual mechanism: the behavior of the virtual mechanism impedance is given by:

$$\tau_{vm} = K_s(s_{cons} - s_{vm}) + B_s(\dot{s}_{cons} - \dot{s}_{vm}) \tag{6}$$

Where $s_{cons} \in \mathbb{R}$ and $\dot{s}_{cons} \in \mathbb{R}$ represent the reference position and the velocity along the desired path, respectively. The gains $K_s \in \mathbb{R}$ and $B_s \in \mathbb{R}$ represent a stiffness-damping coupling and define the impedance of the virtual mechanism.
Since the virtual mechanism is ideal, the efforts applied on it are null. The equilibrium of the applied forces is given by:

$$J_s^T F_c = \tau_{vm} \tag{7}$$

Virtual boundary constraints: The purpose of "*virtual stops*" is to limit the permissible displacements of the tool. They can be defined in a complementary manner to a virtual mechanism, for example imposing limits on the travel of the links that constitute it. More generally, they can be described by surfaces delimiting an area of the working space in which the effector must remain confined. In our implementation of virtual guides, it is possible to specify the stiffness-damping coupling – K_s, B_s – between the virtual mechanism specification and a reference position s_{cons} along the desired path (see Figure 3). It is then possible to create virtual boundaries at the extremities of the path by applying to s_{cons} the following law:

- If $s_{vm} \in [0, s_{max}]$ then $s_{cons} \leftarrow s_{vm}$.
- If $s_{vm} > s_{max}$ then $s_{cons} \leftarrow s_{max}$.
- If $s_{vm} < 0$ then $s_{cons} \leftarrow 0$.

With these boundary constraints the user will feel a repulsive force (spring-damper effect) when he/she reaches the beginning or the end of the path.

Control law: using equations (3), (4), (6) and (7), we obtain:

$$J_s^T(K(X_{vm} - X) + B(J_s\dot{s}_{vm} - \dot{X})) = -B_s\dot{s}_{vm} + K_s(s_{cons} - s_{vm}) + B_s\dot{s}_{cons} \tag{8}$$

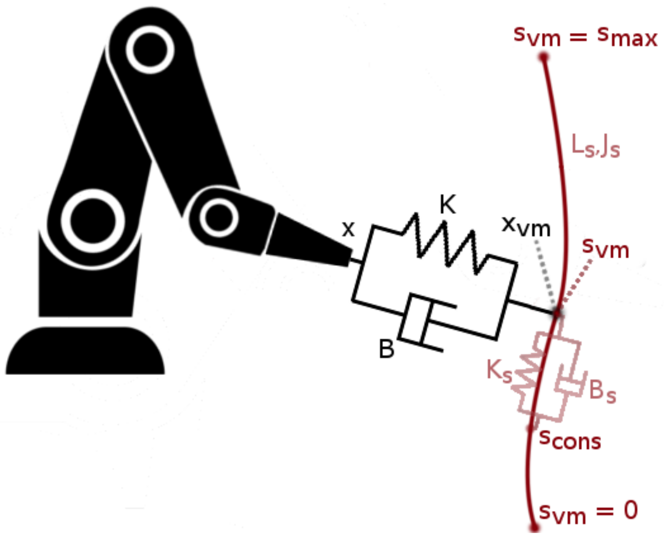


Fig. 3: Lateral view of the physical analogy of a virtual mechanism. The desired path is illustrated in red. The pose X of the cobotic system is linked to the pose X_{vm} of the virtual mechanism by a spring-damper system. The current position of the virtual mechanism is s_{vm}.

By solving equation (8) with respect to $\dot{s}_{vm}$, we obtain a first order dynamical system that expresses the evolution of the virtual mechanism:

$$\dot{s}_{vm} = (B_s + J_s^T B J_s)^{-1}(-J_s^T(K(X_{vm} - x) - B\dot{X})) + K_s(s_{cons} - s_{vm}) + B_s\dot{s}_{cons} \tag{9}$$

The matrix $J_s^T B J_s$ is symmetric, positive, definite because J_s has full column rank and B is symmetric, positive, definite.

s_{vm} can be determined at every instant by integrating the controller state equation (9) in real time. Then, since s_{vm} and $\dot{s}_{vm}$ are known, X_{vm} and $\dot{X}_{vm}$ can be computed based respectively on equations (2) and (3). Finally, the driving force can be computed using (4).

From equations (1), (3), (4), (5), (6) and (9) we obtain the control law scheme presented in Figure 4.

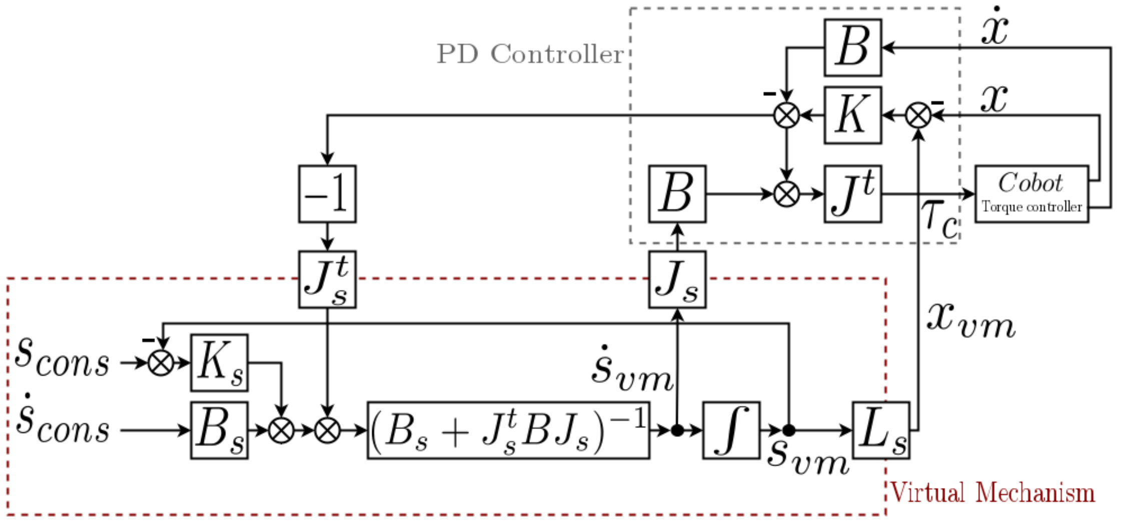


Fig. 4: Control law scheme of a 1-DOF virtual mechanism.

The gain specifications of the coupling – K and B – are independent from the virtual guide specification – L_s, K_s and B_s. Gain tuning is similar to the tuning of a Cartesian PD position loop. The higher the gains, the more the behavior of the robotic system will tend to the one defined by the virtual mechanism. In general, the spring K is chosen as stiff as possible.

The passivity of the virtual mechanism controller is proven by Joly,[13] by using the mechanical analogy of the system and studying the energy dissipation. Moreover, it was proven by Hogan[43] that the passivity of the system guarantees the stability of the controlled system when it interacts with any passive environment, including a human operator.

4. Virtual Guides as Position Constraints

In this section we address the virtual guide construction as a position constraint. We do not present any new contributions but recall from our previous work[40] the main ideas for 3D virtual guide construction since they will be needed in the following sections.

When the user manipulates the cobot and creates a displacement of the end-effector, the force applied by the cobot on the virtual mechanism is expressed by a new value of the phase s_{vm} by integrating the dynamical system defined in equation (9). We can use the direct geometric model L_s (1) and the kinematic model J_s (3) to compute the corresponding *pose* X_{vm} and *velocity* $\dot{X}_{vm}$ of the virtual mechanism for a given value of the phase $s_{vm} = s$. In this section we address the virtual guide construction as position constraints, thus we only focus on the translations of the cobot end-effector $X_{trans} \in \mathbb{R}^3$ (the orientation of the end-effector being fixed or free).

As presented in our previous work,[40] we can define the kinematics of the position constraint through PbD and Akima spline interpolation.[44] During the kinesthetic teaching, the user is able to show the cobot the desired trajectory by manually moving its end-effector. The Cartesian position X_{trans} of the cobot end-effector is recorded on user demand or continuously with a determined sampling time. In both cases, the position X_{trans} and a recording parameter t are stored as a list of points.

When the recording of the trajectory is done by user demand, the recording parameter starts at $t_0 = 0$ and increases monotonically along with the recorded points. When the recording is done continuously, the recording parameter corresponds to the recording time.

In order to create a virtual guide using the list of recorded points, we propose to use an interpolation function that passes exactly through these points so the demonstrated trajectory is encoded in a precise and smooth way. Using interpolation functions to describe the geometric and kinematic models of virtual guides presents the following advantages:

- only one demonstration without virtual guide assistance is needed,
- the recorded trajectory depends entirely on the demonstration and not on any automatic algorithm,
- virtual guides defined through interpolation functions can be easily modified.

In our framework, we reconstruct position constraints from the stored points by using a local cubic polynomial Akima interpolation. This method is a continuously differentiable sub-spline interpolation. It is built from piecewise third order polynomials, where only data from the next and previous two neighbor points is used to determine the coefficients of the interpolation polynomial. The slope of the curve is locally determined at each given point by the coordinates of five points centered on the studied point. This spline type creates a smooth curve between the recorded points and always passes directly through them. Some of the advantages of this interpolation method are:

- it yields to a smooth natural-looking curve,
- there is no need to solve large equation systems. It is therefore computationally very efficient,
- it reduces oscillatory effects,
- local changes do not affect the interpolation beyond neighbor points,
- Akima spline points are intuitive to use when modifying virtual guides.

Besides, the Akima spline interpolation method provides a curve of class C^1 at least, and as stated in,[45] to define virtual mechanisms it is possible to use any curve defined by a parametric function of class C^1. When it comes to interpolation, a compromise must be done between the smoothness of the curve and the robustness to local modifications. For example, for path planning it is better to guarantee C^2 continuity in order to ensure continuous acceleration. For virtual guide implementation we need at least C^1 continuity and for our particular use of virtual guides we need for sure to locally modify the splines.

Finally, the direct geometric model L_s of the virtual mechanism (see Eq.1) is defined by the Akima spline interpolation. Thus, the direct kinematic model J_s (see Eq.3) is defined by the spline's derivate function.

5. Virtual Guides as Orientation Constraints

In this section we address the virtual guide construction in $\mathbb{SO}(3)$, focusing on the orientation of the cobot end-effector, which is one of the main contributions of this work. The bases of quaternion interpolation are presented at the beginning of the section (see 5.1 and 5.2) to facilitate understanding of our contribution to orientation virtual guide construction (see 5.3).

As previously explained in section 4, during kinesthetic teaching the user is able to show the cobot the desired movements by manipulating its end-effector. The orientation X_{rot} of the cobot end-effector is recorded by user demand or continuously with a predetermined sampling time. In both cases, the orientation $X_{rot} \in \mathbb{SO}(3)$ and a recording parameter $t \in \mathbb{R}$ are stored as a list of points.

In order to create orientation constraints using the list of recorded points we propose to use an interpolation function that exactly matches the recorded orientations in order to encode the demonstrated movements in a precise and smooth way. We previously stated the advantages of using interpolation functions for construction of position constraints. The same advantages apply for orientation constraints.

Next, we present a possible representation of orientations based on unit quaternions and different methods to interpolate them: the first method is SLERP (spherical linear interpolation) which is equivalent to the linear interpolation between two points. Then, we will discuss SQUAD (spherical quadrangle interpolation), which builds upon SLERP to create an equivalent of a cubic interpolation. Finally, we present the method used in this work to generate the orientation constraints, which is based on a spline of SQUADs. This will be used to define the geometric and kinematic models of the virtual mechanism from Eq. (1) and (3).

5.1. Interpolation between two quaternions (SLERP)

SLERP[41,46] of quaternions generates constant motion along the geodesic linking between two unit quaternions. Here, we present the analogous quaternion formula of the Euclidean expression for linear interpolation: $x(t) = x_0 + t(x_1 - x_0)$. As the interpolation parameter t uniformly varies between 0 and 1, the values $Slerp(t)$ are required to uniformly vary along the circular arc from p to q, where p and q are unit quaternions. SLERP can be written in an exponential form[41] as:

$$Slerp(t;p,q) = p(p^{-1}q)^t. \tag{10}$$

An example of the application of (10) is shown in Figure 5.

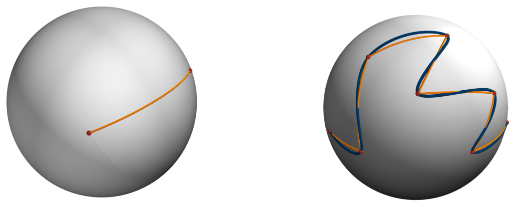


Fig. 5: (Left) Example of the SLERP interpolation between two unit quaternions on the sphere. (Right) Example of the SLERP (orange) and the SQUAD (blue) interpolations between eight unit quaternions on the sphere. We can see that in contrast to the SLERP curve, the SQUAD curve is smooth at the control points (red).

The first derivative of Eq. (10) along t[41] is given by:

$$\dot{Slerp}(t;p,q) = p(p^{-1}q)^t log(p^{-1}q). \tag{11}$$

Although p and $-p$ represent the same rotation, the values of $Slerp(t;q,p)$ and $Slerp(t;q,-p)$ are not the same. Indeed, geometrically the geodesic is a 4D circle, so it can be followed from q in two directions, arriving initially in p or in $-p$. This is a consequence of $\mathbb{SO}(3)$, because in this set a rotation about some direction of 2π returns to the same orientation of origin. The shortest path can be established simply by comparing the distance between p and q with the distance between $-p$ and q. As explained in,[47] it is customary to choose the sign σ on p so that $q \cdot (\sigma p) \geq 0$. In other words, the angle between q and σp is acute. This implementation choice avoids extra spinning caused by the interpolated rotations.

5.2. Interpolation over a series of quaternions (SQUAD)

In the set of unit quaternions, the SLERP interpolation curve of two quaternions is a geodesic. However, the simple juxtaposition of the successive geodesic interpolations joining a series of orientations presents some limitations (see Figure 5):

- the curve is not smooth at the control points,
- the angular velocity is not constant.
- the angular velocity is not continuous at the control points.

A reparametrization can easily ensure continuity across the entire interpolation, but fails to fix the lack of smoothness at the control points.[41] The smoothness is understood here as the continuity of at least the first derivative (C^1 continuity).

The spherical cubic interpolation called SQUAD[46] ensures the smoothness of the curve. It takes inspiration from Bezier curves, but involves spherical linear interpolations instead of simple linear interpolations. The evaluation of SQUAD uses an iteration of three SLERP similarly to the Casteljau algorithm[48] :

1. Imagine four unit quaternions p, a, b, and q as the ordered vertices of a quadrilateral.
2. Interpolate the quaternion c along the "edge" from p to q using $Slerp(t; p, q)$.
3. Interpolate the quaternion d along the "edge" from a to b using $Slerp(t; a, b)$.
4. Now interpolate the edge interpolations c and d to get the final result e.

The end result (final interpolation) is denoted SQUAD and is given by:

$$Squad(t; p, a, b, q) = Slerp(2t(1-t)); Slerp(t; p, q), Slerp(t; a, b)). \tag{12}$$

We can use (10) to obtain the exponential form of SQUAD:

$$Squad = Slerp(t; p, q)(Slerp(t; p, q)^{-1} Slerp(t; a, b))^{2t(1-t)} \tag{13}$$

The derivative of SQUAD in equation (13) is defined in[47] as:

$$\dot{Squad}(t; p, q, a, b) = \frac{d}{dt}[UW^{2t(1-t)}], \tag{14}$$

where

$$\begin{aligned} U(t) &= Slerp(t; p, q), \\ V(t) &= Slerp(t; a, b), \\ \dot{U}(t) &= U(t) log(p^{-1} q), \\ \dot{V}(t) &= V(t) log(a^{-1} b), \\ W(t) &= U(t)^{-1} V(t), \end{aligned}$$

and where $\dot{Squad}$ is not a unit quaternion.

It is also shown in[47] that the derivatives of SQUAD at the endpoints are:

$$\dot{Squad}(0;p,a,b,q) = p[log(p^{-1}q) + 2log(p^{-1}a)], \tag{15}$$

$$\dot{Squad}(1;p,a,b,q) = p[log(p^{-1}q) - 2log(q^{-1}b)]. \tag{16}$$

5.3. Orientation constraint construction through spherical cubic interpolation

Given a sequence of N unit quaternions $\{q_n\}_{n=0:N}$, we want to build an interpolation curve between those quaternions, subject to the following conditions:

- the spline must pass through the control points and
- the first derivatives are continuous at the control points.

To this aim, the idea is to chose intermediate quaternions a_n and b_n to allow control of the derivatives at the endpoints of the spline segments. More precisely, let $S_n(t) = Squad(t;q_n,a_n,b_{n+1},q_{n+1})$ be the spline segments. By definition of SQUAD, the last quaternion of a previous segment $n-1$ is equals to the first quaternion of the current segment n:

$$S_{n-1}(1) = q_n = S_n(0) \tag{17}$$

To obtain continuous derivatives at the endpoints we need to match the derivatives of two consecutive spline segments:

$$\dot{S}_{n-1}(1) = \dot{S}_n(0) \tag{18}$$

From Eq. (16) we can write:

$$\dot{S}_{n-1}(1) = q_n[log(q_{n-1}^{-1}q_n) - 2log(q_n^{-1}b_n)] \tag{19}$$

and

$$\dot{S}_n(0) = q_n[log(q_n^{-1}q_{n+1}) + 2log(q_n^{-1}a_n)] \tag{20}$$

The derivative continuity equation (18) provides one equation with the two unknowns a_n and b_n, so there is one degree of freedom. It is suggested in[41] and[47] to use an average T_n of "tangents", so $\dot{S}_{n-1}(1) = q_nT_n = \dot{S}_n(0)$, where:

$$T_n = \frac{log(q_n^{-1}q_{n+1}) + log(q_{n-1}^{-1}q_n)}{2} \tag{21}$$

With those two equations (18) and (21), a_n and b_n can be determined as follows:

$$a_n = b_n = q_n exp\left(-\frac{log(q_n^{-1}q_{n+1}) + log(q_n^{-1}q_{n-1})}{4}\right) \tag{22}$$

and $b_n = a_{n+1}$.

Thus, $S_n(t) = Squad(t;q_n,a_n,a_{n+1},q_{n+1})$.

The expression of SQUAD is not defined in the first and last interval since q_{n-1} appears in the expression for a_0 and q_{n+1} appears in the expression for a_n. Therefore, it is necessary to define bound values for a_0 and a_n. This choice could have an impact on the resulting interpolation curve continuity class and can be avoided during the implementation. We propose to add two points, before the beginning and after the end of the useful path, to ensure continuity on the desired interval.

For $n=-1$, we define the interpolation as:

$$q(t) = interpolation(t; q_{-1}, q_0) = Slerp(t; q_0, q_1) \tag{23}$$

and its derivative by:

$$\dot{q}(t) = S\dot{l}erp(t; q_0, q_1) \tag{24}$$

with t the interpolation parameter between two quaternions.

For $n = N+1$, we define the interpolation as:

$$q(t) = interpolation(t; q_{N-1}, q_N) = Slerp(t; q_{N-2}, q_{N-1}) \tag{25}$$

and its derivative by:

$$\dot{q}(t) = S\dot{l}erp(t; q_{N-2}, q_{N-1}) \tag{26}$$

where N represents the number of interpolation quaternions as if the quaternion value is equivalent to $q(s = s_{max})$.

5.3.1. Parameterization. Singularities can appear on the virtual mechanisms when the Jacobian J_s is not normalized. These singularities can disturb the interaction between the user and the virtual guide controller. For the position constraints, we proposed a solution based on the arc-length.[40] Similarly, since the list of orientations recorded by the cobot is obtained by user demonstrations, we can not guarantee the normality of J_s for the orientation constraints.

In order to guarantee a normalized Jacobian, it is desirable to evaluate the SQUAD at orientations based on the arc-length of the curve instead of the recording sampling time. For that matter, we propose to separate spacial and temporal aspects of the trajectory.

Let t be the time, s_θ be the arc-length quaternion curvilinear parameter and P_R the list containing the SQUAD orientation waypoints $q_{rot} \in \mathbb{SO}(3)$.

$$P_{R,i} = \{q_{rot,i}\}, \tag{27}$$

with $i = 0 : N-1$, where N is the number of orientation waypoints. When the waypoints are recorded manually, $t = i$.

The SQUAD curve parametrized with time t is defined by f_θ:

$$f_\theta : \left| \begin{array}{l} \mathbb{R} \longrightarrow \mathbb{SO}(3), \\ t \longmapsto P_R. \end{array} \right.$$

The transformation function from the the arc-length curvilinear to time parameterization is defined by g_θ:

$$g_\theta : \left| \begin{array}{l} \mathbb{R} \longrightarrow \mathbb{R}, \\ s_\theta \longmapsto t. \end{array} \right.$$

Where g_θ corresponds to a monotonic cubic interpolation function.[49] This kind of interpolation is optimal for the transformation function since the resulting curve does not present oscillations in the presence of outliers. In the context of rotations in $\mathbb{SO}(3)$, the natural metric is equal to the angle between two rotations. Specifically, given two rotation quaternions r and p, the product rp^{-1} is also a rotation by an angle $\theta \in [0, \pi]$ about some axis. We chose to use the metric measure of quaternions $d(r, p)$ as:

$$d(r,p) = \|\log(r^{-1}p)\| = \theta. \tag{28}$$

We approximate the computation of s_θ with:

$$s_{\theta,i+1} = s_{\theta,i} + d(q_i, q_{i+1}), \tag{29}$$

where $s_{\theta,0} = 0$ and $i = 0 : N-1$. The computation of $d(q_i, q_{i+1})$ is done using (28).

The result of this parameterization is an equal arc length quaternion curve subdivision, and therefore a normalized Jacobian J_s.

The SQUAD spline can now be defined as a composition of the initial curve parameterized with time f_θ and the space transformation function g_θ as:

$$f_\theta(t) = f_\theta(g_\theta(s_\theta)) = f_\theta \circ g_\theta(s_\theta). \tag{30}$$

The virtual guide is now described by the following data list of length N:

$$M_R(t_i, s_i) = \{t_i, s_i, q_{rot,i}\}_{i=0:N-1}. \tag{31}$$

5.3.2. Geometric and kinematic models. The above definitions of the interpolation function for quaternions can be applied to the list of M_R points where t values increase monotonically:

$$M_R(t_i) = \{t_i, x_{rot,i}\}_{i=0:N-1}, \tag{32}$$

where N represents the number of points.

We can now define the geometric model of the virtual mechanism L_s (1) as:

$$L_s : \left| \begin{array}{l} \mathbb{R} \longrightarrow \mathbb{SO}(3), \\ s_\theta \longmapsto Squad(s_\theta; \phi), \end{array} \right.$$

where ϕ represents the other function parameters needed to compute SQUAD.

Notice that the parameter s_{vm} of the virtual mechanism corresponds to the spline parameter s_θ. So we can then write:

$$X_{vm,rot} = Squad\left(g_\theta(s_{vm}), \phi\right). \tag{33}$$

The kinematic model of the virtual mechanism J_s (3) can be defined as the angular velocity which can be obtained using the SQUAD discrete derivative defined as:

$$\dot{Squad}(t;\phi) = \frac{Squad(t+\epsilon;\phi) - Squad(t-\epsilon;\phi)}{2\epsilon} \times \left(\frac{1}{s_{max} - s_{min}}\right), \tag{34}$$

In our implementation, a value of $\epsilon = 0.001$ gave good results.

The kinematic model of the virtual mechanism J_s can be defined using the instantaneous angular velocity[50] $\hat{\omega} \in \mathbb{R}^3$, by:

$$\begin{bmatrix} 0 \\ \hat{\omega} \end{bmatrix} = 2\dot{q} \cdot q^{-1}, \tag{35}$$

$$J_s = \hat{\omega} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}, \tag{36}$$

$$\dot{X_{vm,rot}} = J_s \dot{s_{vm}}. \tag{37}$$

6. Virtual Guides as 6D Constraints

In Section 4 and Section 5, we presented how to define and construct position and orientation constraints. However, robotic applications usually use the pose of the end-effector (or the calibrated tool). In this section, we present our new approach called *XSplines* based on virtual mechanisms and interpolation functions.

6.1. 6D virtual guides through XSplines

To enforce 6D virtual guides we use the virtual mechanism controller presented in Section 3.

However, we must first define the virtual guides and implement their geometric and kinematic models. To this aim, we developed the concept of *XSplines*.

XSplines are curves in $\mathbb{SE}(3)$, defined by both position interpolations in $\mathbb{R}^3$ and orientation interpolations in $\mathbb{SO}(3)$ of poses of a robot obtained through kinesthetic teaching. In other words, *XSplines* are a composition of SQUADS and Akima Splines which we will call *MDSplines* (i.e. multi-dimensional splines). Figure 6 shows an illustration of this concept. The advantage of *XSplines* is that they are parameterized in a way that allows the synchronization of the translation and orientation movements. Thus, we will propose a new parameterization that does the coupling and allows to keep a normalized Jacobian.

A pose $X \in \mathbb{SE}(3)$ is defined as:

$$X = \begin{Bmatrix} X_{trans} \in \mathbb{R}^3 \\ X_{rot} \in \mathbb{SO}(3) \end{Bmatrix} = \begin{Bmatrix} x \\ y \\ z \\ w \\ i \\ j \\ k \end{Bmatrix}. \tag{38}$$

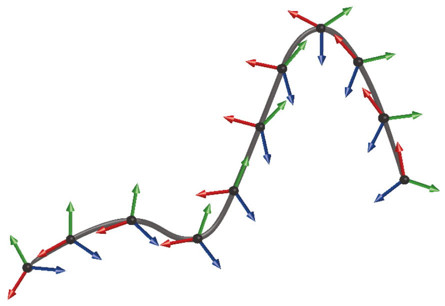


Fig. 6: Illustration of a *XSpline* where both translation and orientation movements are considered in a single 6D curve.

Parameterization: The space parameter s_x is defined to vary depending on both the translational and rotational components of a pose. This parameterization allows s_x to evolve along the curve even if the movement is done only on one of the components. This feature could be very useful when the movement is described by only a rotation along an axis without any translation. In this case, since translation and rotation are coupled via an *XSpline*, the curve parameter will continue to evolve.

This parameterization uses the previously defined parameterizations in $\mathbb{R}^3$ and $\mathbb{SO}(3)$. To this aim, we define a scaling factor L between the two parameters, whose value depends on the geometry of the tool used on the robot.

For two displacements e and u, we define the intermediate space parameter s_x by using the distance $d_x(e,u)$ between them, with:

$$d_x(e,u) = \sqrt{s^2 + L s_\theta^2}, \tag{39}$$

where:

- s: represents the Cartesian translation parameter defined as the arc-length of the curve, as presented in our previous work,[40]
- s_θ: represents the $\mathbb{SO}(3)$ rotation parameter defined in Section 5, and
- L: represents the scaling factor between both parameters.

We approximate the computation of s_x by:

$$s_{x,i+1} = s_{x,i} + d_x(x_i, x_{i+1}), \tag{40}$$

where $s_0 = 0$ and $i = 0 : N-1$. The computation of $d_x(x_i, x_{i+1})$ is done using (39). In our implementation, $L = 0.1$ (corresponding to a lever arm of $10cm$) gave good results.

This s_x parameterization corresponds to the previously defined space parameter of the virtual guides s_{vm} (see section 3 and figure 4):

$$s_{vm} = s_x \tag{41}$$

This new parameterization allows the synchronization of the translation and orientation movements even if the components have been recorded separately.

6.2. Geometric and kinematic models

The above definition of *XSplines* can be applied to a list of M_X poses where t values increase monotonically.

$$M_X(t_i) = \{t_i, x_i\}_{i=0:N-1}, \tag{42}$$

where N represents the number of poses.

We can define the geometric model of the virtual mechanism L_s (1) as:

$$L_s : \left| \begin{array}{l} \mathbb{R} \longrightarrow \mathbb{SE}(3), \\ s_x \longmapsto XSpline(s_x; \lambda), \end{array} \right.$$

where λ represents the other function parameters needed to compute $XSpline$: $\lambda = X_{trans,i}, X_{trans,i+1}, X_{rot,i}, a_i, a_{i+1}, X_{rot,i+1}$.

As explained before, the parameter s_{vm} of the virtual mechanism corresponds to the curve parameter s_x. Then:

$$X_{vm} = XSpline\left(s_{vm}, \lambda\right), \tag{43}$$

with:

$$XSpline(s_{vm}, \lambda) \triangleq \begin{Bmatrix} MDSpline(g(s_{vm})) \in \mathbb{R}^3 \\ Squad(g_\theta(s_{vm}), \lambda) \in \mathbb{SO}(3) \end{Bmatrix}. \tag{44}$$

The kinematic model of the virtual mechanism J_s (3) can be defined using:

$$J_s = \begin{bmatrix} MD\dot{S}pline(g(s_{vm})) \\ \hat{\omega} \end{bmatrix} = \begin{bmatrix} spl\dot{i}ne_x(g(s_{vm})) \\ spl\dot{i}ne_y(g(s_{vm})) \\ spl\dot{i}ne_z(g(s_{vm})) \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}, \tag{45}$$

and:

$$\dot{X}_{vm} = J_s \dot{s}_{vm}. \tag{46}$$

In sections 4, 5, and 6, **we presented how the geometric and kinematic models of virtual guides can be programmed through kinesthetic teaching and modeled through interpolation functions**. To construct position constraints we implemented multi-dimensional Akima Spline interpolations. To construct orientation constraints in $\mathbb{SO}(3)$ we used SQUAD interpolations. In both cases we proposed to separate the time and space components of the curves to parameterize them in a way that guarantees the Jacobian normality. **We also proposed a definition of 6D virtual guides through *XSplines***, based on both position interpolations – $AkimaSpline \in \mathbb{R}^3$ – and orientation interpolations – $Squad \in \mathbb{SO}(3)$ – of poses obtained through kinesthetic teaching. The advantage of *XSplines* is that they are parameterized in a way that allows the **synchronization of the translation and orientation movements**.

7. Iterative Virtual Guides Programming

When asking a user to perform the ideal path he has in mind several times, there may be many variations. These variations could exist due to the presence of friction and gravity forces that the user must compensate, for the poor repeatability inherent to human gestures, the variability of the task and, often, simply human concentration errors on a trajectory portion. For these reasons and the ones outlined in Section 2, we suggest that users program the virtual guides by iteratively modifying them while being assisted by the collaborative robot. In our previous work,[40] we presented a local guide refinement method. The refinement is done directly on the workspace by manually manipulating the cobot end-effector to show a new portion of the guide. During local refinement, the operator may be more focused than during the previous demonstration since he is only able to demonstrate the portion again and not the entire trajectory. The main advantage of this approach is that the worker is assisted throughout the iterative teaching phase and only one entire demonstration of the task is needed. An overview of this concept is introduced in this section to facilitate discussion of the iterative programming framework.

7.1. Scaled force control

In order to show the new portion of the trajectory while the virtual guide is active, the user needs to momentarily escape the guide. To allow this, we use the concepts of *soft virtual guides* and force scaling presented in[23] and[3] to allow the user to go off the path and locally modify the guide. When the user tries to go off the path, the guide controller's force fades proportionally with the distance between the guide's pose X_{vm} and the current pose of the

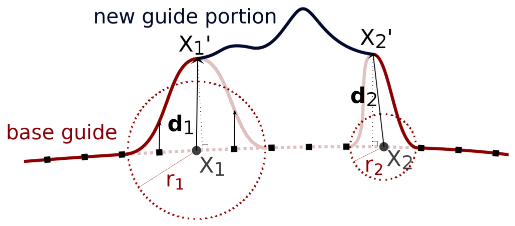


Fig. 7: Local refinement applied to the constraint points X_1 and X_2 lying on the base guide. $X_1^{'}$ and $X_2^{'}$ are the initial and final points of the new guide portion.

robot end-effector X. The user would feel an attractive force F when escaping or approaching the guide, up to a defined distance d_{max}.

$$F = \beta(Y) F_c, \tag{47}$$

To benefit computational efficiency, $\beta(Y)$ is defined as a 4th degree polynomial with $Y = \dfrac{\| X - X_{vm} \|}{d_{max}}$, and null when $Y > 1$:

```
**if** $Y <= 1$ **then**
    $\beta(Y) = y^4 - 2y^2 + 1,$
**else**
    $\beta(Y) = 0,$
**end if**
```

The parameter d_{max} can be tuned manually to modify the basin of attraction of the virtual guide.

Note: the notation $(X - X_{vm})$ may be abusive for rotations. More adequate angular error computation can be used without restrictions.

$$X - X_{vm} \triangleq \begin{Bmatrix} X_{trans} - X_{vm,trans} \\ \delta(q, q_{vm}) \end{Bmatrix}$$

7.2. Local guide refinement of position constraints

Initial and final positions of the partial demonstration do not always match a control point of the guide. To merge the new portion with the rest of the guide, we propose a method to modify the closest points on the base guide to match the first and last points on the new guide portion.

Let X_1 and X_2 be the constraint points of the current guide. $X_1^{'}$ and $X_2^{'}$ are the initial and final points of the new guide portion, respectively (see Fig.7). The constraint points are defined as the two base guide points which are closest to $X_1^{'}$ and $X_2^{'}$, respectively. We obtain X_1 and X_2 by calculating the distance from $X_1^{'}$ and $X_2^{'}$ to the base guide control points. Then we select the two control points of minimum distance. Displacement vectors $\overrightarrow{d_1}$ and $\overrightarrow{d_2}$ are determined between X_1 and $X_1^{'}$, and between X_2 and $X_2^{'}$. A radius of influence r must be defined for both constraints. We choose the radius proportional to the magnitude of the displacement vector.

$$\overrightarrow{d}_1 = \overrightarrow{X_1 X_1^{'}}; \qquad r_1 = \alpha_1 \parallel \overrightarrow{d}_1 \parallel,$$

$$\overrightarrow{d}_2 = \overrightarrow{X_2 X_2^{'}}; \qquad r_2 = \alpha_2 \parallel \overrightarrow{d}_2 \parallel.$$

The radii r_1 and r_2 allow a more intuitive control of the deformation. The only parameters to tune are α_1 and α_2, where a compromise must be made between the smoothness of the curve and the area of influence of the deformation, i.e, the neighbor points that will be deformed. We define a local deformation function F centered at the constraint points and decreasing to zero for points beyond the radius. In order to obtain a smooth displacement of the spline points, we determine a fourth degree polynomial as the deformation function $F(x)$, where $x = \dfrac{(s_{vm,i} - s_{vm,0})}{r}$. $\{s_{vm,i}\}_{i=0:N-1}$ represent the curvilinear abscissa parameter of the Akima spline, and N is equal to the number of interpolation points.

The deformation function $F(x)$ is then defined as:

$$F(x) = f(x)\,\overrightarrow{d}.$$

With :

$$f(x) = x^4 - 2x^2 + 1.$$

The deformation function $F(x)$ is applied to the X_1 and X_2 neighbor points within the radius of influence r_1 and r_2, in order to obtain the new control points that now include $X_1^{'}$ and $X_2^{'}$. These new guide control points are stored in a vector defined by the modified base guide control points before X_1 and after X_2, along with the new guide portion's control points. Finally, we perform a new Akima spline interpolation to define the new virtual guide.

7.3. Refinement of orientation components

The same approach presented for the refinement of position constraints can be applied to $\mathbb{SE}(3)$. Thus, the orientation components can also be refined locally or modified on a portion of the 6D guide. The modification is done on the orientation of the closest key point. However, it is also possible to add new key points if needed (e.g. if the closest point is still too far from the current point, in which case a threshold might be defined). This functionality can be used iteratively using the scaled force control.

7.4. Iterative programming of 6D virtual guides

During the execution of a complex task in space, it could be difficult to concentrate on both translation and orientation of the robot. For this reason, we propose to realize the trajectory programming in two phases to reduce the cognitive load on the user and enhance the programming experience by separating the translation and rotation programming.

In the first phase we record both translation and orientation. The second time we use the virtual mechanism's translation only in order to modify the orientation.

1. The user does a first demonstration of the trajectory (positions) using the cobot in gravity compensation mode.
2. A position constraint virtual guide is constructed and activated.
3. While being guided by the position constraint trajectory, the user does a demonstration of the tool's orientations.
4. A 6D virtual guide is constructed and activated.

A demonstration of this approach can be visualized using the following link:

- 6D virtual guides iterative programming.

In this section we **presented a new intuitive, iterative and assisted framework for programming 6D virtual guides**. We explained how virtual guides can be locally refined or modified by the user in both Cartesian position and orientation. During this iterative process, the user benefits from the virtual guide assistance. Also, the scaled force control allows to scape the guide to modify it. Finally, we presented an iterative approach to program 6D virtual guides in two phases. **This method aims to reduce the cognitive load of the user and enhance the programming experience by separating the translation and the rotation programming**. In our framework, the human operator masters the action plan and the cobot assists the human passively and under explicit demand.

8. Experimental Evaluation

A user study was designed in order to observe how novice users perceive the *Iterative Programming Mode* and to analyze the impact of our approach on a comanipulation programming task by comparing it to a classic programming mode used in the industry that we refer here as *One Shot*.

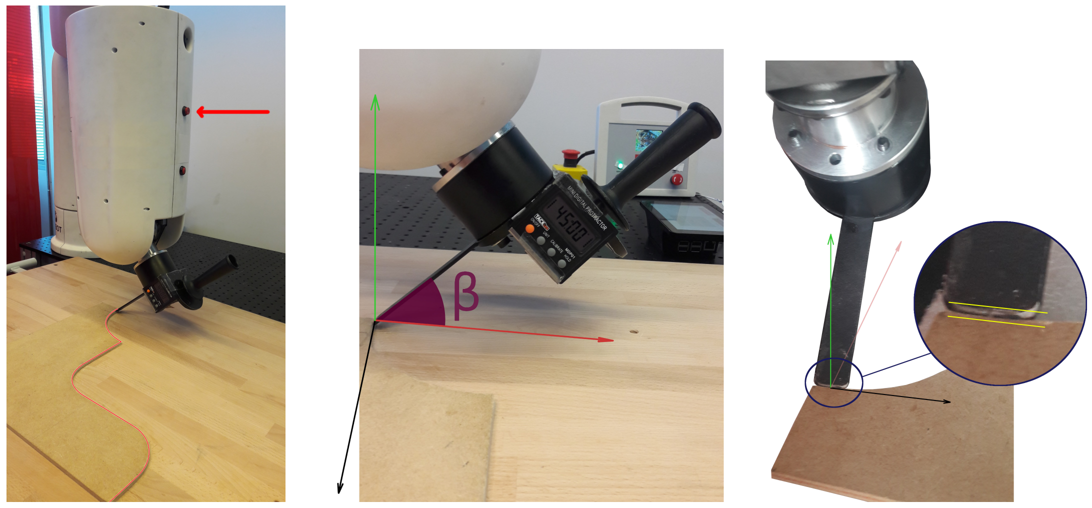
The following hypotheses were tested:

- H1: The Iterative Programming Mode reduces the *programming time* of the task,
- H2: it improves the *accuracy* of the results,
- H3: it is *intuitive and comfortable* to use,
- H4: it is perceived as *helpful* by the users,
- H5: and it reduces the user's *physical effort and cognitive overload* to program the task.
- H6: The co-manipulation task with the cobot is not perceived as stressful.

8.1. Task definition

The experience is conducted with the 6-DOF ISybot collaborative robot in Fig. 1 and a scraper tool. The general task consists of programming the cobot to follow the contour of a 2cm thick wood part respecting two conditions. As shown in Fig. 8, participants have to:

1. Keep the tooltip orientation angle at 45° with the vertical axis (Figure 8(b)).
2. Stay parallel to the tangent of the curve at each point of the trajectory (Figure 8(c)).



(a) (b) (c)

Fig. 8: *a)* Programming task setup. The path to follow is highlighted in red. The arrow indicates the *record button*. *a)* The orientation of the scraper tool, angle β, must be kept at 45° within the vertical plane. *b)* The scraper tool must be kept parallel to the tangent of the curve at each point of the path.

An inclinometer is placed on the tool in order to help the users to maintain the 45° angle instruction.

This task simulates the complexity of some manufacturing tasks such as: glue depositing and joint installation (automobile industry), grinding, and polishing.

To program the cobot, participants used the *Iterative Programming Mode* explained in Section 7. Users were able to record discrete key points of the trajectory using the *record button* – the upper button placed on the 4th axis of the cobot, indicated by the red arrow in

Figure 8(a). These points were then interpolated via *XSplines* and used to create *6D virtual guides* as explained in Section 6.

Participants were asked to program the task with a good accuracy/speed ratio. They were reminded that in order to draw a line at least two points are needed, to draw an arc at least 3 points are needed and that there should be a good compromise between number of points (precision) and the time recording them (speed).

A demonstration of the task can be visualized using the following links:

- 6D virtual guides iterative programming.
- 6D Virtual guides visualization using augmented reality.

To analyze the impact of our *Iterative Programming Mode*, we compare it to a classic programming mode used in the industry that we refer here as *One Shot*.

The *One Shot* programming mode allows only one demonstration with gravity compensation assistance.

To program the cobot with this mode, participants were also able to record discrete key points of the trajectory using the *record button*. They were given the same instructions regarding the accuracy/speed ratio for learning the task.

For the virtual guides assistance used in our *Iterative Programming Mode*, the controller gains were set as follows:

$K = [10000\ 10000\ 10000\ 3000\ 3000\ 3000]\ N/m$ and $N/m.rad$

$B = [150\ 150\ 150\ 30\ 30\ 30]\ N/m.s^{-1}$ and $N/m.rad.s^{-1}$

$K_s = [5000]\ N/m$ and $N/m.rad$

$B_s = [5]\ N/m.rad.s^{-1}$

The sampling time was set to 1 ms.

8.2. Protocol

We recruited 17 participants from our research laboratory (between 20 and 53 years old, 7 females). All participants were asked to program the cobot to follow the contour of the wood part, using both programming modes – One Shot and Iterative – resulting in two test conditions. The two test conditions were presented in a randomized order to avoid biasing the results towards the last mode tested. For each condition, the participants were asked to program the cobot 2 times in a row (Repetitions). In total, the participants performed $4 = 2 \times 2$ (Mode $\times$ Repetition) cobot programming tasks.

At the beginning of each condition, the programming mode was presented to the participants and the tested case was demonstrated to show the participants how to use the cobot and the programming interface. Then, they were able to familiarize themselves with the system and try the tested case on their own. After each repetition, the result was shown to the participants i.e., they could feel the virtual guide they created and compare the learned path with the real contour of the part thanks to the haptic feedback given by the cobot. Finally, when a condition was completed, the participants were asked to complete a post-condition survey – a Likert-scale survey with a rating from 1 (strong disagreement) to 7 (strong agreement).

8.3. Measurements

To validate our hypotheses, we made the following measurements:

1. Programming time measured between the beginning of the programming task and the last point saved, to validate H1 – The *Iterative Programming Mode* reduces the programming time of the task.

2. RMSE of the angle and RMSE of the distance between the participants' and the reference paths, to validate H2 – The *Iterative Programming Mode* improves the accuracy of the results.
3. Survey results, to validate H2, H3, H4, H5 and H6 – The *Iterative Programming Mode* improves the accuracy of the results, is intuitive and comfortable to use, is perceived as helpful by the users, reduces the user's physical effort and cognitive overload to program the task. *The co-manipulation task with the cobot* is not perceived as stressful.

8.3.1. Time. The programming time is measured in seconds and automatically recorded by the cobot for each *Programming Mode* and each *Repetition*. It was measured between the beginning of the programming task (one short click on upper button) and the last point saved (one long click on upper button). For the *Iterative Programming Mode*, the total programming time is the result of the addition of the two iterations.

8.3.2. Accuracy. To measure the accuracy of the programming task, two quantitative variables were taken into account:

- The RMSE of the angle
- The RMSE of the distance between the participants' inputted paths and the reference paths

Participants' paths were saved after each programming task execution using MATLAB through a RPC communication protocol with the cobot. The interpolation algorithm we used gives us a 0.5 mm spatial resolution.

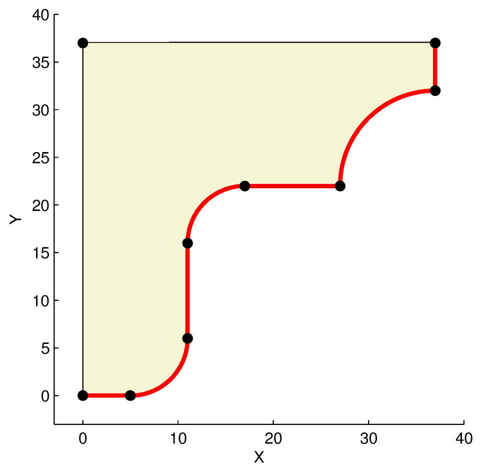


Fig. 9: Calibration points. The initial wood part measured 37x37x2 cm. It was then cut to shape the contour to follow.

We defined a parametric model of the wood part based on its exact measures. Before the experiment, we took nine calibration points with the tooltip of the cobot as shown in Figure 9. Using these nine recorded points, we computed the affine transformation to fit the cobot reference frame to the MALTAB reference frame[1]. We applied this unique transformation to every participant's path for both *Programming Modes* and both *Repetitions*. We can now compare the participant's recorded re-aligned path to its corresponding model portion as shown in Figure 10.

[1] This calibration method has the drawback of being highly dependent on the calibration of the tooltip and the mechanic flexibility of the cobot. However, these measures were used to do a comparison between paths obtained under the same conditions and using the same calibration process. Also, we used nine points for the calibration where three points could be enough.

For all interpolation points of the reference path, we defined normal vectors $\overrightarrow{n_j}$, tangent vectors $\overrightarrow{t_j}$ and the resulting vectors $\overrightarrow{s_j}$ of the cross product of those (respectively in green, black and magenta in Figure 10), thus resulting in a direct orthonormal basis.

For every interpolation point of a participant's path, we defined the vector $\overrightarrow{v_j}$ representing the orientation of the tooltip (in dark yellow). In our case, this vector corresponds to the third column of the rotation matrix representing the orientation quaternion at each point.

We define :

- $v_{tj} = \overrightarrow{t_j} \cdot \overrightarrow{v_j}$, the tangential component,
- $v_{nj} = \overrightarrow{n_j} \cdot \overrightarrow{v_j}$, the normal component.

2.1 - Angle: The required angle β_j was calculated as: $\beta_j = atan2d(v_{tj}, v_{nj})$. In other words, it corresponds to the angle between the scraper tool and the vertical vector of the cobot frame within the plane perpendicular to the trajectory. For this angle, we calculated the RMSE of the angle δ_{RMSE}, between the 45° instruction and β_j for each trajectory.

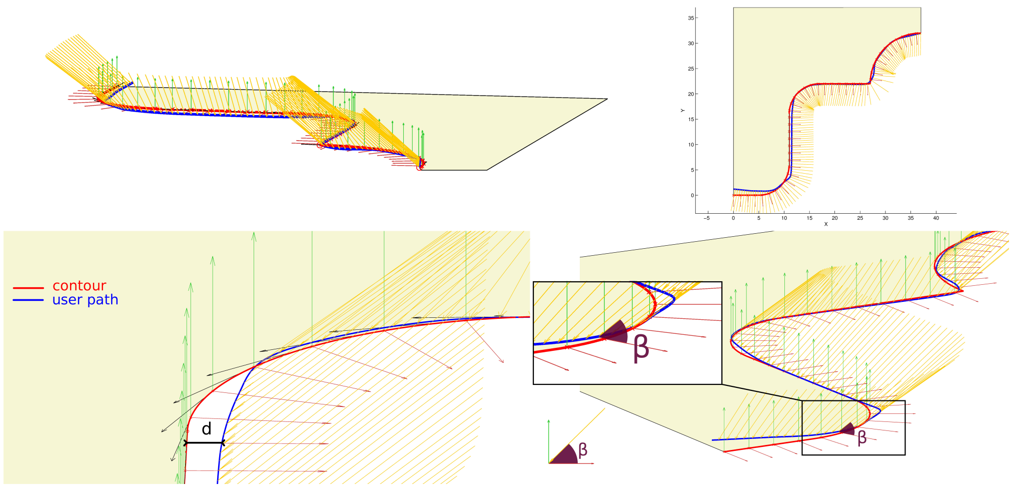


Fig. 10: Accuracy measures: d – distance between each recorded interpolation point and its closest corresponding point on the reference path – and β – angle between the scraper tool and the vertical vector of the cobot frame within the plane perpendicular to the trajectory. Reference paths are plotted in red and participant's paths are plotted in blue. The yellow arrows represent the tool orientation vectors, green arrows represent the normal vectors to the path, and the red arrows represent the normal vectors to the tangent of the path at each interpolation point. *Top-left figure*: Perspective view of the wood part. *Top-right figure*: Top view (x-y axis) of the representation of the followed path by participants. Distance values are on cm. *Bottom-left figure*: Perspective view. The measure d is shown. *Bottom-right figure*: Perspective view. The measure β is shown.

2.2 - Distance: As previously defined, the participant had also to precisely stick to the contour of the part. Thus, we measured the distance d between each recorded interpolation point and its closest corresponding point on the reference path as shown in Figure 10. We computed the RMSE of the distance d_{RMSE} for each trajectory.

Finally, we recorded the answers to the two post-condition surveys after each condition was performed by the participant.

8.4. Results

We performed independent repeated-measures ANOVA tests for 3 dependent variables:

1. Time,
2. RMSE of the angle (δ_{RMSE}), and
3. RMSE of the distance (d_{RMSE})

Each ANOVA test was performed for two factors (moreover, the participants were grouped by age, gender and previous experience with robots, but we found that these factors did not have any influence on the experiment results):

1. *Modes*
2. *Repetitions*

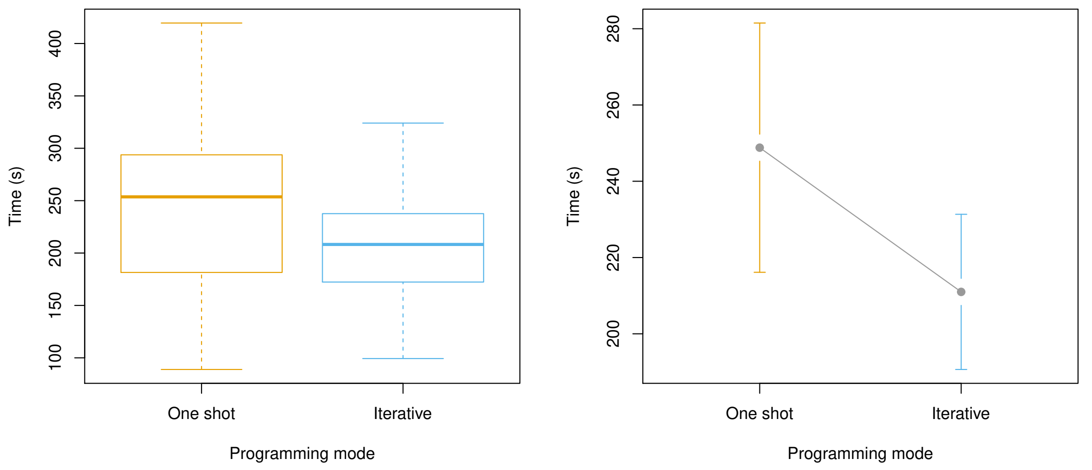
We performed independent repeated-measures ANOVA for each survey question on the *Mode*.

We set our significance level at $\alpha = 0.05$, which means there is at least 5% chance of having a difference between means of the studied variables. We consider a result strongly significant when p-value < 0.01 and poorly significant when p-value < 0.1. We use poorly significant results to show trends and give further analysis of data, however we do not draw conclusions based on these results. Post hoc pairwise comparisons were computed using non-pooled error terms, i.e., by computing separate paired-sample t tests: we used a sequentially acceptive step-up Benjamini[51] procedure, with an alpha level of .05.

8.4.1. Time results. We found a significant main effect of the *Programming Modes* method on the task execution time of the participants (p-value $< .01$). As we can see in Table I, participants were faster with the *Iterative Programming Mode* than with the *One Shot Programming Mode*. This indicates that the *Iterative Programming Mode*, using virtual guides, reduced the programming time and also allowed more stable performance since the standard deviation is 1.6 times smaller (see Fig. 11). This result validates hypothesis H1 –The *Iterative Programming Mode* reduces the programming time of the task.

Table I : Effect of the *Programming Mode* on the programming time.

Mode	$mean_{Time}$ (s)	σ_{Time} (s)
One Shot	248.08	93.67
Iterative	210.99	58.31



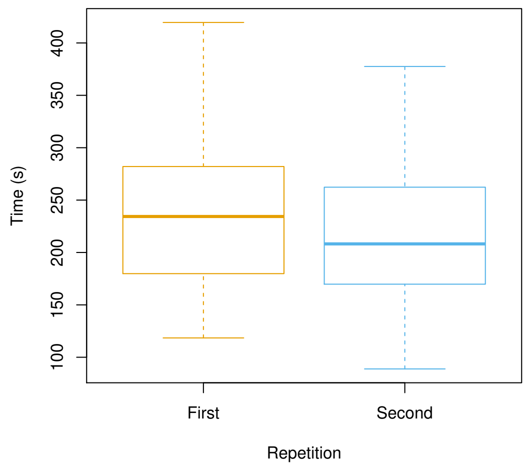
(a) Box plot (b) Group means with confidence interval of 95%

Fig. 11: Effect of the *Programming Mode* on the programming time

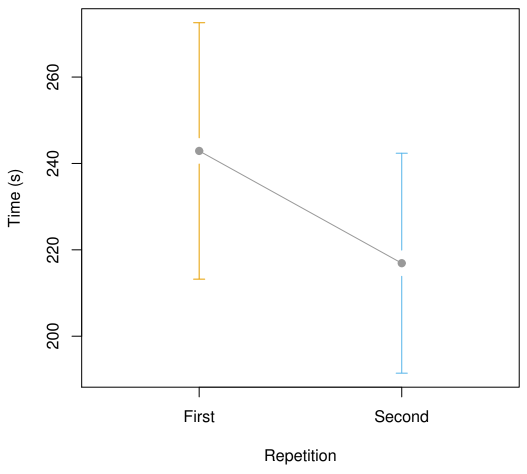
We also found a strongly significant main effect of the *Repetitions* on the programming time of the participants (p-value $< .01$). We can see in Table II and Figure 12, participants were slower during the first repetition. This result confirms there is a training effect through *Repetitions*.

Table II : Effect of the *Repetitions* on the programming time.

Repetitions	$mean_{Time}$ (s)	σ_{Time} (s)
1^{st}	242.89	85.07
2^{nd}	216.90	72.97



(a) Box plot



(b) Group means with confidence interval of 95%

Fig. 12: Effect of *Repetitions* on the programming time

There is no significant effect of interaction between *Repetitions* and the *Programming Mode* (p-value $= 0.1204$). However, we can say from Table III that participants were faster during both *Repetitions* for the *Iterative Programming Mode*.

Table III : Interaction effect between *Repetitions* and *Programming Modes* on the programming time

Mode	**Repetition**	$mean_{Time}$ (s)	σ_{Time} (s)
One shot	1^{st}	267.26	101.03
Iterative	1^{st}	218.54	58.81
One shot	2^{nd}	230.37	84.65
Iterative	2^{nd}	203.44	58.59

In conclusion, the *Iterative Programming Mode* allowed the participants to execute the task faster during both *Repetitions*, which shows that the general training effect between *Repetitions* did not influence the effect of the *Programming Mode* on the time.

8.4.2. Accuracy results. To analyze the accuracy, we performed the following two quantitative measures:

- the *RMSE* of the angle (δ_{RMSE}),
- the *RMSE* of the distance (d_{RMSE}).

There is no significant main effect of *Programming Modes* or *Repetitions* on the *RMSE* of the angle. However, we can see in Table IV that the error was smaller when participants used the *Iterative Programming Mode*.

Table IV : Effect of the *Programming Modes* on the *RMSE* of the angle

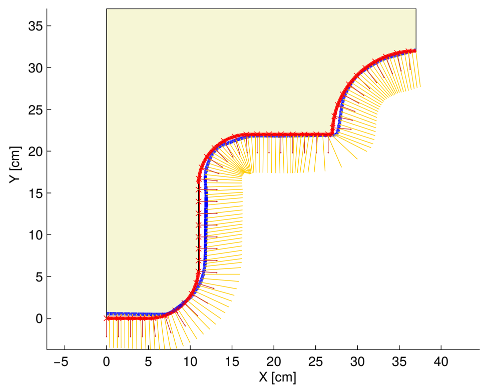
Programming Mode	$mean_{\delta_{RMSE}}$ (°)	$\sigma_{\delta_{RMSE}}$ (°)
One Shot	2.93	1.83
Iterative	2.82	2.11

There is no significant main effect of *Programming Modes* or *Repetitions* on the *RMSE* of the distance. However, Table V show that the *RMSE* of the distance is smaller when the *Iterative Programming Mode* is used.

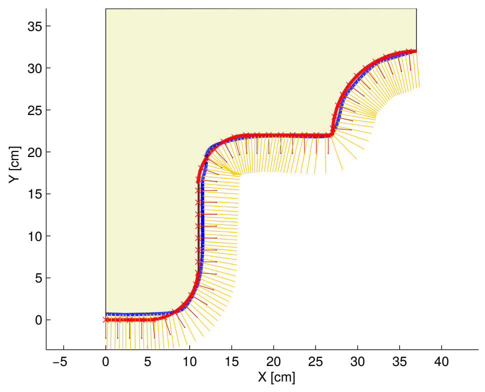
Table V : Effect of the *Programming Modes* on the *RMSE* of the distance

Programming Mode	$mean_{d_{RMSE}}$ (cm)	$\sigma_{d_{RMSE}}$ (cm)
One Shot	0.70	0.29
Iterative	0.64	0.22

In conclusion, there is no significant difference between the *Iterative* and the *One Shot Programming Modes* on the accuracy of the task, so we cannot validate H2 – The *Iterative Programming Mode* improves the accuracy of the results. However, quantitative results go in the direction of H2 and show that participants had better results when they used the *Iterative Programming Mode*. Some of the best participants' qualitative results using both *Programming Modes* are shown in Figure 13.



(a) One Shot



(b) Iterative

Fig. 13: Best qualitative results from two different participants. Top view (x-y axis) of the representation of the followed trajectory by participants. The reference path is plotted in red and participants' paths are plotted in blue. The yellow arrows represent the tool orientation vectors and the red arrows represent the normal vectors to the tangent of the path at each interpolation point.

.

8.4.3. Survey results. For clarity, the answers to the 9 questions of our user study survey for each *Programming Mode* are summarized in Table VI.

With this survey, we observed a significant effect of the *Programming Mode* on several of the studied questions. The results of the survey are reported in the following summary.

Table VI : Results of the user study survey for two *Programming Modes*

	One Shot		Iterative		
Question	**Mean**	σ	**Mean**	σ	**p-value**
Q1: Do you think that you performed the task well?	4.00	1.17	4.58	1	0.065
Q2: Do you think the task was easy to perform?	4.47	1.73	4.94	1.39	0.054
Q3: Do you think the robot was helpful during the task execution?	2.41	1.06	4.35	1.45	$\leq$0.0001
Q4: Did you feel comfortable with the robot while performing the task?	2.94	0.83	4.71	1.31	$\leq$0.0001
Q5: Do you think the programming interface was intuitive?	4.59	1.37	5.23	0.83	0.052
Q6: Do you think the robot was easy to manipulate?	2.65	1.17	4.12	1.41	$\leq$0.0001
Q7: Did you feel you had to exert a lot of physical effort to perform the task?	4.35	1.58	3.24	1.03	$\leq$0.01
Q8: Did you feel your level of concentration to perform the task was high?	5.88	1.26	3.88	0.93	$\leq$0.0001
Q9: Did you feel stressed using the robot while performing the task?	2.53	1.55	1.76	0.90	0.032

8.5. *Summary of the experiment*

Using the previous results, we can make the following conclusions about the initial hypotheses of the experiment:

H1: The main effect of the *Programming Mode* on the time execution of the task validates the hypothesis H1 – *Iterative Programming Mode* reduces the execution time of the task.

H2: Several results supported the hypothesis H2 – *Iterative Programming Mode* improves the accuracy of the results. When participants used the *Iterative Programming Mode*,

- the *RMSE* of the angle was smaller,
- the *RMSE* of the distance was smaller, and
- participants found that they performed the task better.

However, the ANOVA tests did not show a significant difference between both *Programming Modes* for both accuracy variables or for Question Q1 (Do you think you performed the task well?). Thus, we cannot validate H2. This lack of significance can be explained by the number of participants in this experiment. We believe that our results show a trend and that a user study with more participants could lead to the validation of this hypothesis.

H3: Several results supported the hypothesis H3 – The *Iterative Programming Mode* is intuitive and comfortable to use – when the participants used the *Iterative Programming Mode*,

- it was easier to program the robot (Q2),
- it was more comfortable to program the robot (Q4), and
- it was more intuitive to program the robot (Q5).

H4: The positive results of the survey Q3 and Q6 (Do you think the robot was helpful during the task execution?, Do you think the robot was easy to manipulate?) validates the hypothesis H4 – The *Iterative Programming Mode* is perceived as helpful by the users.

H5: The results of the survey questions Q6, Q7 and Q8, validate the hypothesis H5 – The *Iterative Programming Mode* reduces the user's physical effort and cognitive overload when programming the task (see table VI):

- Q6: Do you think the robot was easy to manipulate?
- Q7: Did you feel you had to exert a lot of physical effort to perform the task?
- Q8: Did you feel your level of concentration to perform the task was high?

H6: For both *Programming Modes* the mean score on the survey question Q9 (Did you feel stressed using the robot while performing the task?) was low. This result validates the hypothesis H6 – The comanipulation task with the cobot is not perceived as stressful.

With this experiment we confirmed the advantages of using 6D virtual guides for comanipulation tasks. We also showed the positive impact of our *Iterative Programming* approach – divide the programming process in two phases: first positions and then orientations. Participants found our method easier to use, more comfortable and more intuitive than the *One Shot* method (that uses gravity compensation). Participants also found that our approach reduces physical effort and cognitive overload. Finally, the comanipulation task with the cobot did not induce any stress on participants.

9. Conclusion

In this work we presented a kinesthetic teaching framework that uses virtual guiding assistance to program virtual guides in an intuitive and flexible way. We proposed a novel implementation of virtual guides using virtual mechanisms and *XSplines* to create 6D virtual guide constraints. Our approach enables non-robotics experts users to create virtual guides by demonstration. Users may also iteratively reprogram the constraints by modifying a portion of the guide through physical interaction with the cobot. We suggested using a scaled force control to escape the active guide in order to modify it. The experimental evaluation of the system with several users showed an application of our approach with an assistance cobot where the task is defined as a combination of position and orientation constraints, simulating the complexity of some manufacturing tasks such as: glue depositing and joint installation (automobile industry), grinding, and polishing. Furthermore, the user study demonstrates that our iterative programming mode improves the programming time and is perceived as intuitive, comfortable and helpful by the users. A first use case study of the use of the Isybot cobot for the manufacturing industry using virtual guides has been presented in.[52] In future work, we plan to present a study of the impact of our 6D virtual guides programming framework in a real-case scenario of assisted manufacturing tasks such as polishing, drilling and deburring, and in the aeronautics industry for assembly of heavy parts.

References

1. H. C. Lin, K. Mills, P. Kazanzides, G. D. Hager, P. Marayong, A. M. Okamura, and R. Karam, "Portability and applicability of virtual fixtures across medical and manufacturing tasks," in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pp. 225–230, May 2006.
2. A. Bettini, P. Marayong, S. Member, S. Lang, A. M. Okamura, and G. D. Hager, "Vision assisted control for manipulation using virtual fixtures," in *International Conference on Intelligent Robots and Systems (IROS)*, pp. 1171–1176, 2004.
3. G. Raiola, S. S. Restrepo, P. Chevalier, P. Rodriguez-Ayerbe, X. Lamy, S. Tliba, and F. Stulp, "Co-manipulation with a library of virtual guiding fixtures," *Autonomous Robots*, vol. 42, pp. 1037–1051, Jun 2018.
4. L. Rosenberg, "Virtual fixtures: Perceptual tools for telerobotic manipulation," in *IEEE Virtual Reality Annual International Symposium*, 1993.
5. B. Davies, M. Jakopec, S. J. Harris, F. R. Y. Baena, A. Barrett, A. Evangelidis, P. Gomes, J. Henckel, and J. Cobb, "Active-constraint robotics for surgery," *Proceedings of the IEEE*, vol. 94, pp. 1696–1704, Sept 2006.
6. J. Colgate, M. Peshkin, and S. Klostermeyer, "Intelligent assist devices in industrial applications: a review," in *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings*, Oct. 2003.
7. S. A. Bowyer, B. L. Davies, and F. R. y Baena, "Active constraints/virtual fixtures: A survey," *IEEE Transactions on Robotics*, vol. 30, pp. 138–157, Feb 2014.
8. O. David, F.-X. Russotto, M. Da Silva Simoes, and Y. Measson, "Collision avoidance, virtual guides and advanced supervisory control teleoperation techniques for high-tech construction: framework design," *Automation in Construction*, 2014.
9. T. Xia, S. Léonard, I. Kandaswamy, A. Blank, L. L. Whitcomb, and P. Kazanzides, "Model-based telerobotic control with virtual fixtures for satellite servicing tasks," in *2013 IEEE International Conference on Robotics and Automation*, pp. 1479–1484, May 2013.
10. J. Abbott, P. Marayong, and A. Okamura, "Haptic virtual fixtures for robot-assisted manipulation," *Springer Tracts in Advanced Robotics*, vol. 28, 2007.
11. P. Marayong, M. Li, A. M. Okamura, and G. D. Hager, "Spatial motion constraints: theory and demonstrations for robot guidance using virtual fixtures.," in *ICRA*, pp. 1954–1959, IEEE, 2003.
12. J. J. Abbott and A. M. Okamura, "Virtual fixture architectures for telemanipulation," 2003.
13. L. Joly and C. Andriot, "Imposing motion constraints to a force reflecting telerobot through real-time simulation of a virtual mechanism," in *, 1995 IEEE International Conference on Robotics and Automation, 1995. Proceedings*, May 1995.
14. Z. Pezzementi, G. D. Hager, and A. M. Okamura, "Dynamic guidance with pseudoadmittance virtual fixtures," in *IEEE International Conference on Robotics and Automation*, pp. 1761–1767, 2007.

15. D. Aarno, S. Ekvall, and D. Kragic, "Adaptive Virtual Fixtures for Machine-Assisted Teleoperation Tasks," in *Proceedings of the 2005 IEEE International Conference on Robotics and Automation, 2005. ICRA 2005*, Apr. 2005.
16. A. Kuang, S. Payandeh, B. Zheng, F. Henigman, and C. MacKenzie, "Assembling virtual fixtures for guidance in training environments," in *Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2004. HAPTICS '04. Proceedings. 12th International Symposium on*, pp. 367–374, March 2004.
17. V. Pruks, I. Farkhatdinov, and J.-H. Ryu, "Preliminary study on real-time interactive virtual fixture generation method for shared teleoperation in unstructured environments," in *International Conference on Human Haptic Sensing and Touch Enabled Computer Applications*, pp. 648–659, Springer, 2018.
18. S. Calinon, F. D'halluin, E. Sauser, D. Caldwell, and A. Billard, "Learning and Reproduction of Gestures by Imitation," *IEEE Robotics Automation Magazine*, 2010.
19. D. Lee and C. Ott, "Incremental kinesthetic teaching of motion primitives using the motion refinement tube," *Autonomous Robots*, vol. 31, no. 2-3, pp. 115–131, 2011.
20. M. J. Zeestraten, I. Havoutis, J. Silvério, S. Calinon, and D. G. Caldwell, "An approach for imitation learning on riemannian manifolds," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1240–1247, 2017.
21. K. Kronander and A. Billard, "Learning compliant manipulation through kinesthetic and tactile human-robot interaction," *IEEE transactions on haptics*, vol. 7, no. 3, pp. 367–380, 2014.
22. A. Bettini, S. Lang, A. Okamura, and G. Hager, "Vision assisted control for manipulation using virtual fixtures," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 2, pp. 1171–1176, IEEE, 2001.
23. J. T. Nolin, P. M. Stemniski, and A. M. Okamura, "Activation cues and force scaling methods for virtual fixtures," in *in Proc. 11th Int. Symp. Haptic Interfaces for Virtual Environment and Teleoperator Systems*, pp. 404–409, 2003.
24. R. Prada and S. Payandeh, "A study on design and analysis of virtual fixtures for cutting in training environments," in *Eurohaptics Conference, 2005 and Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, 2005. World Haptics 2005. First Joint*, pp. 375–380, IEEE, 2005.
25. M. Li, M. Ishii, and R. H. Taylor, "Spatial motion constraints using virtual fixtures generated by anatomy," *IEEE Transactions on Robotics*, vol. 23, no. 1, pp. 4–19, 2007.
26. S. A. Bowyer and F. R. y Baena, "Dynamic frictional constraints for robot assisted surgery," in *World Haptics Conference (WHC), 2013*, pp. 319–324, 2013.
27. R. A. Castillo-Cruces and J. Wahrburg, "Virtual fixtures with autonomous error compensation for human–robot cooperative tasks," *Robotica*, vol. 28, no. 2, pp. 267–277, 2010.
28. S. A. Bowyer and F. R. y Baena, "Dynamic frictional constraints in translation and rotation," in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pp. 2685–2692, IEEE, 2014.
29. S. A. Bowyer and F. R. y Baena, "Dissipative control for physical human–robot interaction," *IEEE Transactions on Robotics*, vol. 31, no. 6, pp. 1281–1293, 2015.
30. D. Zhang, L. Wang, J. Gu, Z. Li, and K. Chen, "Realization of spatial compliant virtual fixture using eigenscrews," in *Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the IEEE*, pp. 1506–1509, IEEE, 2012.
31. D. Zhang, Q. Zhu, J. Xiong, and L. Wang, "Dynamic virtual fixture on the euclidean group for admittance-type manipulator in deforming environments," *Biomedical engineering online*, vol. 13, no. 1, p. 51, 2014.
32. L. Rozo, S. Calinon, and D. Caldwell, "Learning force and position constraints in human-robot cooperative transportation," in *2014 RO-MAN: The 23rd IEEE International Symposium on Robot and Human Interactive Communication*, Aug. 2014.
33. E. S. Boy, E. Burdet, C. L. Teo, and J. Colgate, "Investigation of Motion Guidance With Scooter Cobot and Collaborative Learning," *IEEE Transactions on Robotics*, 2007.
34. Y. Mollard, T. Munzer, A. Baisero, M. Toussaint, and M. Lopes, "Robot programming from demonstration, feedback and transfer," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept. 2015.
35. T. Kastritsi, F. Dimeas, and Z. Doulgeri, "Progressive automation with dmp synchronization and variable stiffness control," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3789–3796, 2018.
36. L. Peternel, T. Petrič, and J. Babič, "Human-in-the-loop approach for teaching robot assembly tasks using impedance control interface," in *2015 IEEE international conference on robotics and automation (ICRA)*, pp. 1497–1502, IEEE, 2015.
37. M. Selvaggio, G. A. Fontanelli, F. Ficuciello, L. Villani, and B. Siciliano, "Passive virtual fixtures adaptation in minimally invasive robotic surgery," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3129–3136, 2018.
38. A. M. Martin Tykal and V. Kyrki, "Incrementally assisted kinesthetic teaching for programming by demonstration," *2016 11th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*, 2016.

39. K. Kosuge, T. Itoh, T. Fukuda, and M. Otsuka, “Tele-manipulation system based on task-oriented virtual tool,” in *, 1995 IEEE International Conference on Robotics and Automation, 1995. Proceedings*, May 1995.
40. S. Sanchez Restrepo, G. Raiola, P. Chevalier, X. Lamy, and D. Sidobre, “Iterative virtual guides programming for human-robot comanipulation,” in *IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, 2017.
41. E. B. Dam, M. Koch, and M. Lillholm, *Quaternions, interpolation and animation*, vol. 2. Datalogisk Institut, Københavns Universitet, 1998.
42. E. D. Wisama Khalil, *Modélisation, identification et commande des robots.* Paris: Hermès science, janvier 1999.
43. N. Hogan, “On the stability of manipulators performing contact tasks,” *IEEE Journal on Robotics and Automation*, 1988.
44. H. Akima, “A New Method of Interpolation and Smooth Curve Fitting Based on Local Procedures,” *J. ACM*, 1970.
45. L. Joly, *Commande hybride position/force pour la teleoperation: une approche basée sur des analogies mécaniques.* PhD thesis, Paris 6, 1997.
46. K. Shoemake, “Animating rotation with quaternion curves,” in *ACM SIGGRAPH computer graphics*, vol. 19-3, pp. 245–254, ACM, 1985.
47. D. Eberly, “Quaternion algebra and calculus,” *Magic Software Inc*, 2002.
48. G. Farin, *Curves and surfaces for computer-aided geometric design: a practical guide.* Elsevier, 2014.
49. F. N. Fritsch and R. E. Carlson, “Monotone piecewise cubic interpolation,” *SIAM Journal on Numerical Analysis*, vol. 17, no. 2, pp. 238–246, 1980.
50. A. J. Hanson, “Visualizing quaternions,” in *ACM SIGGRAPH 2005 Courses*, p. 1, ACM, 2005.
51. Y. Benjamini and Y. Hochberg, “Controlling the false discovery rate: A practical and powerful approach to multiple testing,” *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 57, no. 1, pp. 289–300, 1995.
52. R. Farel, S. Kchir, X. Lamy, and M. Grossard, “Challenges in sustainable manufacturing with industrial and collaborative robots: A case study,” in *ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pp. V004T05A040–V004T05A040, American Society of Mechanical Engineers, 2018.