# Extending Piecewise Bézier Fitting Methods to $SO(3)$ and $SE(3)$ for Robot Paths


1st Ignacio Montesino
*RoboticsLab UC3M*
Madrid, Spain
imontesi@ing.uc3m.es

2nd Juan G. Victores
*RoboticsLab UC3M*
Madrid, Spain
jcgvicto@ing.uc3m.es

3rd Carlos Balaguer
*RoboticsLab UC3M*
Madrid, Spain
balaguer@ing.uc3m.es

4th Alberto Jardon
*RoboticsLab UC3M*
Madrid, Spain
ajardon@ing.uc3m.es



***Abstract*—This paper presents a new extension of a Bézier fitting method that can now be applied in non-Euclidean spaces. While the original algorithm was restricted to the 2D plane due to the dependence on the properties of linear spaces, this new method overcomes this limitation by leveraging the power of Lie groups. By incorporating tangent vectors and gradient descent algorithms, this innovative approach produces accurate results, even in complex geometric spaces. To test its precision and performance, the algorithm was applied to digitize the recorded trajectory of a KUKA LBR 14 IIWA cobot. The results of this study reveal new possibilities for integration with VR / AR systems and enabling more complex geometric control schemes.**



***Index Terms*—Bézier curves, Lie Groups, Cobots, VR, AR, Neuromotor Rehabilitation**


## I. Introduction

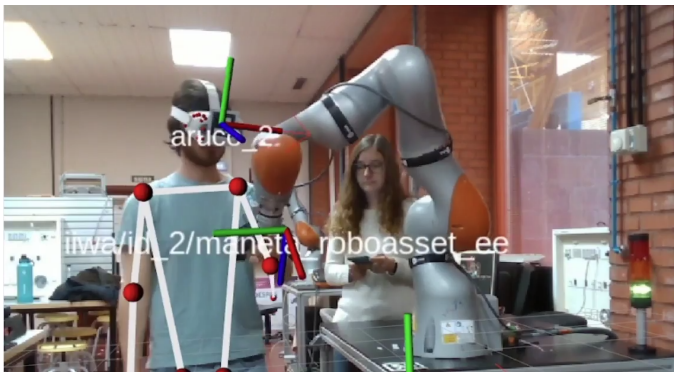
The use of Bézier curves in representing geometrical data in computer graphics has been long established. These curves are particularly advantageous because of the intuitive meaning of their parameters, which are defined by four points. The first and last points represent the endpoints of the curve, whereas the second and third points, or "handles," determine the direction of the tangent and its magnitude at those endpoints.

Currently, Bézier curves are commonly used in path planning for manufacturing robots because they are already a standard feature in CAD software [1]. They are also utilized in robot navigation as a means of smoothing out trajectories to ensure feasibility and smoothness [2].

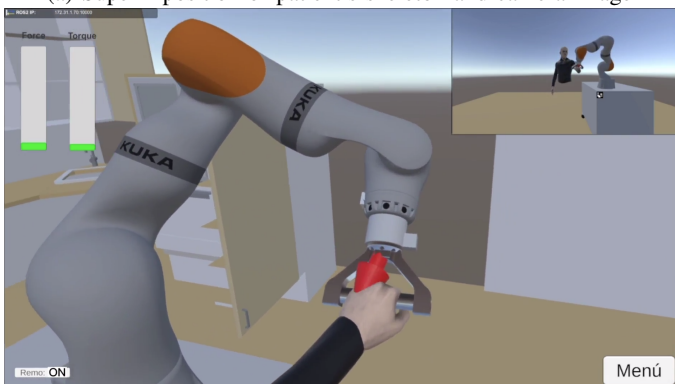
Moreover, the adoption of virtual reality (VR) technology in human-robot interaction (HRI) has significantly improved the interaction capabilities of operators of collaborative robots (cobots) by providing geometrical information displayed in their natural environment through VR and augmented reality (AR) systems [3].

In parallel, the growing applications of robots in neurorehabilitation have shown promising results in upper limb rehabilitation, where robots can perform repetitive force-controlled movements with more precision than their human counterparts [4]. The effectiveness of VR-aided treatments in motor rehabilitation outcomes further highlights the potential for improvement in VR-aided robotic rehabilitation [5].


This research has received funding from ROBOASSET, "Sistemas robóticos inteligentes de diagnóstico y rehabilitación de terapias de miembro superior", PID2020-113508RB-I00 financed by AGENCIA ESTATAL DE INVESTIGACION (AEI).

(a) Superimposition of patient's skeleton and camera image




(b) Patient's view inside the VR environment

Fig. 1: Roboasset VR-aided robotic rehabilitation system

To address this gap, this paper presents a novel method for fitting piecewise Bézier curves to data in $SO(3)$ and $SE(3)$ spaces. This approach enables a compact and flexible representation of end-effector trajectories captured in the context of motor rehabilitation.

In the Roboasset project, we aim to give physiotherapists a fully integrated robotic motor rehabilitation system gamified through the use of VR. The system is shown in Figure 1. It is in this system that the need of a geometrical description of recorded trajectories became apparent. Both to measure the deviations from the trajectory in a geometrically meaningful way and as a first step to implement control schemes along the recorded trajectories.

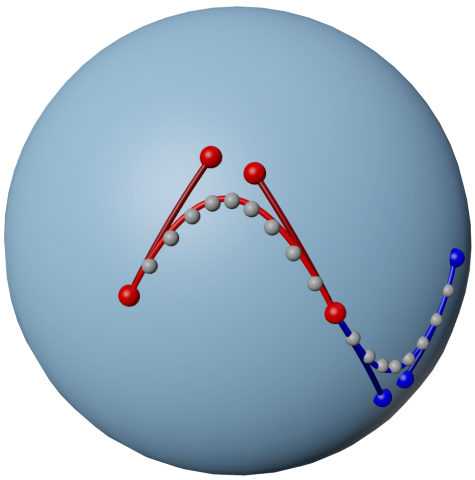


Fig. 2: A representation of a fitted Bézier curve on the sphere.

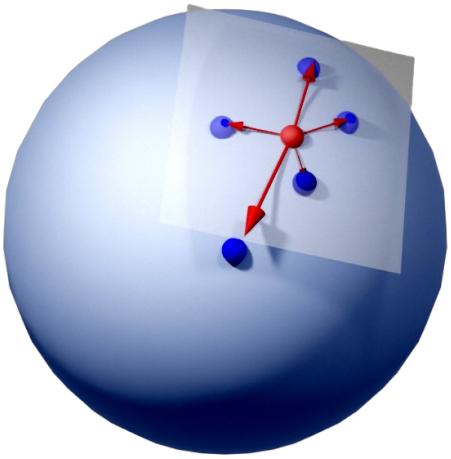


Fig. 3: Finding the tangent vector equivalents of a neighborhood around an element of $G$

## II. Lie Groups and Bézier Curves

A Bézier curve is a smooth parametric curve defined by a set of control points. When no degree is specified, the curve is assumed to be a cubic Bézier curve defined by four control points. The curve can be obtained by recursively performing linear interpolation between the points; this is known as DeCasteljau's algorithm.

$$B_i^0(t) = p_i \tag{1}$$
$$B_n^i(t) = (1-t)B_n^i(t) + tB_{n-1}^{i+1}(t). \tag{2}$$

In the case or $\mathbb{R}^n$, the formulation can be reduced to an expression using Bernstein polynomials, but not in the case of $SO(3)$ and $SE(3)$.

The expression of DeCasteljau's algorithm for Riemmanian manifolds and Lie groups is given by [6], which uses the geodesic interpolation between points

$$P(t) = X \cdot \exp(\log(X^{-1}Y) \cdot t). \tag{3}$$

In equation 3, $X$ and $Y$ are elements of the Lie Group. The element $X^{-1}Y$ can be interpreted as the element $Y$ as seen from the frame of reference of $X$. By performing the $\log$ map, we obtain the vector of the Lie algebra that represents the difference between the two elements. The exponential map is then applied to obtain the element that represents the interpolation between the two elements at a moment $t$ between 0 and 1. To obtain the Bézier curve at a time $t$ we apply the same procedure recursively using the interpolation in eq. 3 instead of eq. 2. A more thorough explanation of Lie groups and Lie algebra for robotics can be found in [7].

## III. Algorithm

The algorithm is an extension of the one presented in [8] where the objective was to digitize hand-drawn sets of $\mathbb{R}^2$ points into sets of Bézier curves.

For this the algorithm consisted of the following steps, which will need to be extended to work with our Lie groups:

1) Define the first and last point of the curve as the first and last points of the set of points.
2) Find the tangent at the first and last point by subtracting the neighboring points.
3) Obtain the other two control points along the geodesics defined at the endpoints by the aforementioned tangents.
4) Find a parametrization $u_i$ for each point $p_i$ in the set of points, that generate the curve point $q_i$ closest to it.
5) Iterate over the last steps until the error is below a certain threshold or the max number of iterations is reached.
6) If the error is still above the threshold, split the set of points at the point of maximum error and repeat the process. The tangent at the split point is computed, and then, given as the tangent of the endpoint and the beginning point of the new set. So as to ensure $G1$ continuity.

### A. Finding the tangent vectors

To find the tangent vector of the trajectory at a point, $p_0$ we first define a neighborhood of $N$ points around it. Now each of those points is expressed as a vector in the tangent space of $p_0$ by performing the modified "minus" operation defined by [7]

$$X \ominus Y = \log(X^{-1}Y) \in T_X G \tag{4}$$

This is just a shorthand notation for the operation $\log(X^{-1}Y)$, which, as described before, obtains the element $Y$ as seen from the frame of reference of $X$, and lifts it to the tangent space of $X$.

If the neighborhood of points were all along the same geodesic starting from $p_0$, their tangent vectors would be collinear. We can find a decent approximation of the tangent vector by finding the best-fit line of the points in the tangent space.

An efficient way to find said best fit line is by using the singular value decomposition (SVD) of the matrix of points. The SVD of a matrix $A$ is defined as

$$A = U\Sigma V - T. \tag{5}$$

Here $U$ and $V$ are orthogonal matrices and $\Sigma$ is a diagonal matrix. If we construct $A$ in the following way

$$A = \begin{bmatrix} x_1 - x_m & y_1 - y_m & z_1 - z_m \\ x_2 - x_m & y_2 - y_m & z_2 - z_m \\ \vdots & \vdots & \vdots \\ x_N - x_m & y_N - y_m & z_N - z_m \end{bmatrix} \tag{6}$$

where $x_i$, $y_i$ and $z_i$ are the coordinates of the $i$-th point in the neighborhood and $x_m$, $y_m$ and $z_m$ are the coordinates of the mean of all neighboring points.

The columns of $V$ will be the principal directions of the points in the tangent space. The first one will be the direction of the best-fit line.

### B. Finding the control points

Once the tangent vectors at the endpoints of the curve are found, the other two control points must take the form

$$p_1 = p_0 \cdot \exp(\lambda_1 \cdot \hat{t}_0) \tag{7}$$

$$p_2 = p_3 \cdot \exp(\lambda_2 \cdot \hat{t}_3) \tag{8}$$

Where $p_0$ and $p_3$ are the end points of the curve, $t_0 \in T_{p_0}G$ and $t_3 \in T_{p_3}G$ are the tangent vectors at those points, ang $\lambda_1$ and $\lambda_2$ are two real positive numbers.

In the original paper [8] a matrix form of the regression problem is presented by operating on the Bernstein polynomial expression of the curve. This is not possible in our case, since the equivalence $exp(v + u) = exp(v) \cdot exp(u)$ does not hold for noncommutative Lie groups. This prevents the curve from being defined as a Bernstein polynomial in the Lie algebra.

Similarly to 4 we can define a modified "plus" operation as

$$X \bigoplus v = X \cdot \exp(v) \in G \tag{9}$$

where $X \in G$ and $v \in T_X G$. This shorthand notation allows us to formulate the optimization problem is defined as

$$\begin{aligned} &\min_{\lambda_1,\lambda_2} \sum_{i=1}^{N} \| B(u_i) \bigominus p_i \|^2 \\ &\text{with } B := \{B_0, B_0 \bigoplus \lambda_0 v_0, B_3 \bigoplus \lambda_3 v_3, B_3\} \end{aligned} \tag{10}$$

where $B(u)$ is the point given by computing the DeCasteljau at time $u_i$ with the control points $B$.

The optimization problem stated in equation 10 poses certain complexities. The computation of the Jacobian of the objective function is a computationally intensive task, as it requires recursive differentiation of the exponential map. Additionally, the problem is constrained by the $\log$ function, which limits the range of the parameters $\lambda_1$ and $\lambda_2$. For example, in the case of $SO(3)$, the value of $\|\lambda_i \hat{t}_i\|$ must fall within the range of $(0, \pi)$ [9].

Therefore, the chosen optimization method will be L-BFGS-B [10]. This variant of the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm accommodates bound constraints. The algorithm leverages an approximation of the Hessian matrix of the objective function to determine the minimum.

For the first iteration, the parametrization $u_i$ for each point $p_i$ is sampled uniformly by the formula

$$u_i = \frac{i}{N-1} \tag{11}$$

### C. Recomputing the parametrization

Having found the optimal control points with the first naive parametrization, we can now recompute the parametrization of the points by solving the following optimization problem

$$\begin{aligned} &\min_{u_i} \sum_{i=1}^{N} \| B(u_i) \bigominus p_i \|^2 \\ &\text{with } B := \{B_0, B_0 \bigoplus \lambda_0 v_0, B_3 \bigoplus \lambda_3 v_3, B_3\} \end{aligned} \tag{12}$$

Since this is a single variable minimization, the problem consists of finding the roots of the derivative. Since the curve is defined for any real value of $u$, we can take a very small step size to compute an accurate numerical approximation of the first and second derivatives of the objective function.

Using the Newton-Raphson method, we can find the roots of the derivative of the objective function. The update rule for the parameter $u_i$ is given by

$$u_i^{(k+1)} = u_i^{(k)} - \frac{f'(u_i^{(k)})}{f''(u_i^{(k)})} \tag{13}$$

### D. Review of the algorithm

We now have a complete algorithm to fit a set of Bézier curves controlled by a few hyperparameters:

- $\epsilon_{\max}$: The maximum distance between the curve and a point.
- $N_s$: The maximum number of times the curve can be split. (This can alternatively be defined as the minimum number of points needed to generate a sub curve).
- $N_{\text{it}}$: The maximum number of iterations for the optimization problem.

## IV. Experimental Results

In order to verify the accuracy and effectiveness of the algorithm, we tested the fitting process on a set of trajectories captured on a real collaborative robot. First, we recorded an arbitrary trajectory of the end effector of a KUKA IIWA robot, which was captured at a rate of 100Hz. During this process, we recorded both the position and orientation of the end effector. In total, we captured 1466 points for this trajectory, which we subsequently used as input for our algorithm.

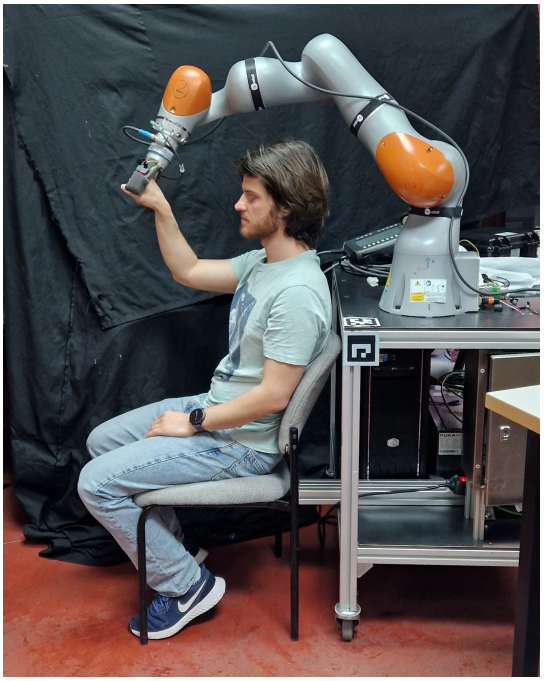


Fig. 4: Upper limb movement capture platform using a KUKA LBR 14 IIWA robot

To implement the algorithm, we utilized C++ and the manif [8] library for computations involving Lie groups. Additionally, we established certain parameters to guide the optimization process. Since $SE(3)$ is the Cartesian product of $SO(3)$ and $\mathbb{R}^3$, it is more intuitive to define the interpolation as two curves, one in $SO(3)$ and one in $\mathbb{R}^3$, each with its own set of parameters. For example, for $\mathbb{R}^3$, we set the maximum number of fitting iterations before splitting to 100. After conducting extensive testing, we found no benefit in further increasing this value. For $SO(3)$, however, we set the maximum number of fitting iterations before splitting to 2, since L-BFGS-B already iterates to find the optimal solution. We also set the target $\epsilon_{\max}$ to $0.005$ m for $\mathbb{R}^3$ and 0.05Rad for $SO(3)$.

The captured trajectory consisted of 1466 points and orientations. The algorithm took $1.58$ seconds to fit the curves using 9 $\mathbb{R}^3$ Bézier curves and 6 $SO(3)$ Bézier curves. We then used these curves to generate a fitted spline for the input trajectory. Figure 5 illustrates the results of our testing process, displaying the original $R^3$ path in green and the interpolated path in red. The accuracy of the $SO(3)$ is shown as sets of axes at each point along the path. As can be seen, the robot's end effector coincides at each point along the path. This allowed us to visually verify the effectiveness of our algorithm in producing a fitted spline that closely approximates the original trajectory.

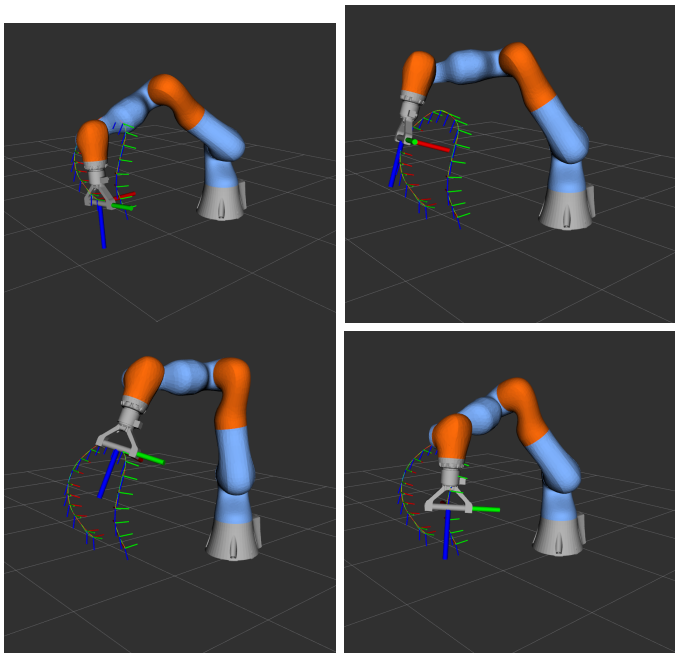


Fig. 5: Accuracy of the Captured SE3 Trajectory

## V. Conclusion and Future Work

In this paper, we have presented an extension of a well known algorithm for fitting Bézier curves to data points in Euclidean space to the case of Lie groups. The result is an efficient algorithm, fast enough to be used online with very little delay to provide visual feedback of trajectories, as well as a building block for more complex human robot interaction applications.

Currently, the algorithm has been tested on the Lie groups $R^N$, $SO(3)$ and $SE(3)$. Other Lie groups could have been explored in the futures, with special interest in one which preserves information about the null space of redundant robots, such as $SE(3) \times R^N$. Another option is the group of dual quaternions, to combine $SO(3)$ and $\mathcal{R}^3$ and using Screw Linear Interpolation, as shown in [11], to generate the Bézier curves.

Another possible extension is to parametrize the curve according to the arc-length of the curve instead of the parameter, $u$ which does not produce a uniform speed along the curve. This would allow the design of control strategies using the geometric properties of the curve.

## References


[1] C. Zhou, B. Huang, and P. Fränti, "A review of motion planning algorithms for intelligent robots," *Journal of Intelligent Manufacturing*, vol. 33, pp. 387–424, Feb. 2022.

[2] A. Ravankar, A. A. Ravankar, Y. Kobayashi, Y. Hoshino, and C.-C. Peng, "Path Smoothing Techniques in Robot Navigation: State-of-the-Art, Current and Future Challenges," *Sensors (Basel, Switzerland)*, vol. 18, p. 3170, Sept. 2018.

[3] M. Dianatfar, J. Latokartano, and M. Lanz, "Review on existing VR/AR solutions in human–robot collaboration," *Procedia CIRP*, vol. 97, pp. 407–411, Jan. 2021.

[4] E. D. Oña, R. Cano-de la Cuerda, P. Sánchez-Herrera, C. Balaguer, and A. Jardón, "A Review of Robotics in Neurorehabilitation: Towards an Automated Process for Upper Limb," *Journal of Healthcare Engineering*, vol. 2018, p. e9758939, Apr. 2018.

[5] J. A. Feitosa, C. A. Fernandes, R. F. Casseb, and G. Castellano, "Effects of virtual reality-based motor rehabilitation: A systematic review of fMRI studies," *Journal of Neural Engineering*, vol. 19, Jan. 2022.

[6] F. C. Park and B. Ravani, "Be´zier Curves on Riemannian Manifolds and Lie Groups with Kinematics Applications," *Journal of Mechanical Design*, vol. 117, pp. 36–40, Mar. 1995.

[7] J. Solà, J. Deray, and D. Atchuthan, "A micro Lie theory for state estimation in robotics," Dec. 2021.

[8] P. J. Schneider, "An algorithm for automatically fitting digitized curves," in *Graphics Gems*, pp. 612–626, USA: Academic Press Professional, Inc., Aug. 1990.

[9] J. L. Blanco-Claraco, "A tutorial on SE(3) transformation parameterizations and on-manifold optimization," Apr. 2022.

[10] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal, "Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization," *ACM Transactions on Mathematical Software*, vol. 23, pp. 550–560, Dec. 1997.

[11] A. Sarker, A. Sinha, and N. Chakraborty, "On screw linear interpolation for point-to-point path planning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9480–9487. ISSN: 2153-0866.