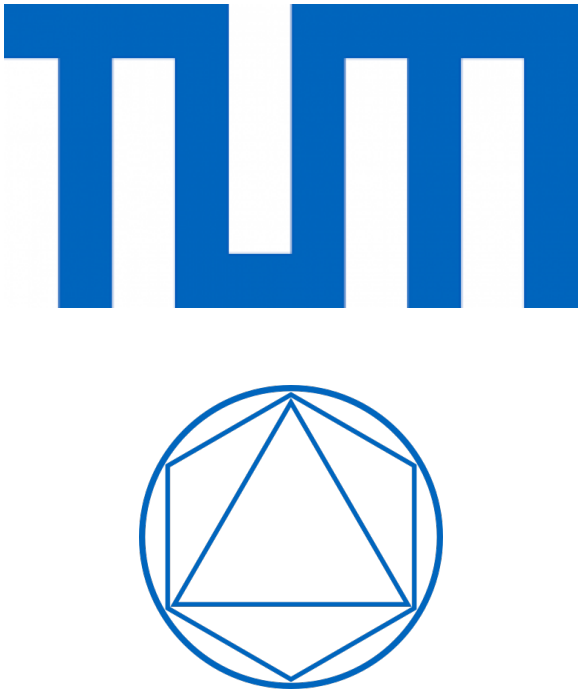


Case Studies

Motion Interpolation and Sensor Fusion

Documentation

Lennart Bastian
Antoine Keller
Sofia Morales Santiago

Table of contents

1 Introduction

Fields such as computer vision and robotics are interested in the applications of tracking 3D objects and in the understanding of position and orientation (6d pose) of an object in space relative to a stationary or moving object.

There are a variety of different systems for measuring the 6d pose of an object, such as cameras (optical tracking systems : OTS), inertial measurement units (IMU) or tactile systems. Nevertheless the possible loss of information about the 6 degrees of freedom of an object due to delayed or interrupted data is a significant limitation in real life applications.

This project aims to calculate rotation and translation given input streams of IMU and vision which work at very different frequencies and accuracy levels and ultimately find a smooth and accurate pose trajectory that describes the underlying motion through nonlinear optimization techniques.

Interpolations are performed on the set of unit quaternions, $\mathbb{H}_1$, which represent smooth interpolations between rotations. The problem is not trivial, in particular because the set $\mathbb{H}_1$ constitute a non-Euclidean space, which requires extending typical interpolation techniques such as splines.

The methods developed in this project are compared to some of the more typical ways of pose interpolation such as Linear Quaternion Interpolation (LERP), Spherical Linear Quaternion Interpolation (SLERP) and Spherical Spline Quaternion Interpolation (SQUAD). SLERP interpolates along the shortest great arc on the quaternion unit sphere, while SQUAD extends spherical interpolation for more than two quaternions.

2 Rotations in $\mathbb{R}^3$

2.1 Euler Angles and Rotation Matrices

We provide a brief introduction of concepts involved in 3D interpolation.

Interpolation between two 6d poses can be modeled by treating the rotational and translational components separately. As translations occur in the euclidean space $\mathbb{R}^3$ linear or spline interpolation can be performed without worry. Translations are non-ambiguous : there exists only one translation vector from a point P to P'. Rotations can be defined in several different ways, the most intuitive of which can yield undesirable effects when used to interpolate.

A rotation with Euler angles is written as a series of rotations about three mutually orthogonal axes in space often called *x-roll*, *y-roll* and *z-roll*. Euler angles have become the most widely used way of parameterizing rotations. Each type of *roll* has a corresponding rotation matrix in the special orthogonal group $\mathbb{SO}_3$. Performing a rotation on an object involves multiplying the three rotation matrices corresponding to each Euler angles. The resulting matrix in $\mathbb{SO}_3$ embodies a general rotation and can be applied to the points that are to be rotated. However this rotation is not unique, and can be obtained from a different set of Euler Angles. Furthermore, it is dependent upon the order of which the multiplications are performed.

Conventions have been adopted to simplify some of these issues. However, when performing interpolation several other issues also arise. Rotation matrices can lose a degree of freedom, known as gimbal lock, where the rotation about one axes has the same effect as the rotation around another. Furthermore, when interpolating between rotation matrices the orthonormal characteristic can degenerate, resulting in an undesired scaling of the points this transformation is applied to. Quaternions simplify the parameterization of rotations as they have an intuitive geometrical interpretation, and interpolation can be controlled to avoid scaling or gimbal lock.

2.2 Quaternions

A quaternion is an element of algebra $\mathbb{H}$ of the form

$$q_1 1 + q_2 i + q_3 j + q_4 k = (q_1, q_2, q_3, q_4)^T$$

with $(q_1, q_2, q_3, q_4)^T \in \mathbb{R}^4$ and $i^2 = j^2 = k^2 = ijk = -1$. A quaternion is usually written as $[s, \mathbf{v}]$. $s \in \mathbb{R}$ is denoted as the scalar part and $\mathbf{v} = (x, y, z) \in \mathbb{R}^3$ the vector part.

As all quaternions along a line through the origin perform the same rotation, it is sensible to operate on a subset of the quaternion group, namely the set of unit

quaternions $\mathbb{H}_1$. As a quaternion q and $-q$ still perform the same rotation the space of Unit Quaternions consists of a double-covering of the space $\mathbb{SO}_3$. It is important to note that quaternion multiplication is non-commutative, which coincides with the non-commutative characteristic of rotation matrices.

Quaternions can be constructed by a rotation axis v, and an angle of rotation around this axis. let $v = (v_1, v_2, v_3)^T$ by a rotation axis, with angle $\theta \in]\pi, \pi]$, then the quaternion $q = [\cos(\theta/2), \sin(\theta/2)v]$ performs a rotation around axis v, with angle 2θ.

Rotation of a point p$\in \mathbb{R}^3$ by a quaternion is performed through the "sandwich" multiplication which it's conjugate $\overline{q}$, namely $p_{new} = q * p * \overline{q}$. The conjugate of a quaternion being defined as $\overline{q} = q_1 1 - q_2 i - q_3 j - q_4$.

The composition of a rotation is achieved by multiplying the corresponding quaternions : let $q_1, q_2 \in \mathbb{H}_1$, then rotation by q_1 followed by rotation by q_2 is equivalent to rotation by $q_2 q_1$.

The use of quaternions will allow simple formulations for the interpolations discussed later, as well as avoiding the undesirable effects discussed above.

3 Interpolation in $\mathbb{SO}_3$

3.1 Linear Quaternion Interpolation : LERP

LERP is a simple linear interpolation of the end quaternions. Let $p, q \in \mathbb{H}_1$, $h \in \mathbb{R}$, then LERP is defined as

$$LERP(p, qh) \coloneqq p(1-h) + qh$$

A major drawback of LERP is that the quaternions leave the space $\mathbb{H}_1$ and have to be projected back into it through normalization. This results in undesirable changes in angular velocity, as seen in figure 1b. Aside from this difference, normalized LERP produces the same interpolation curve as SLERP.

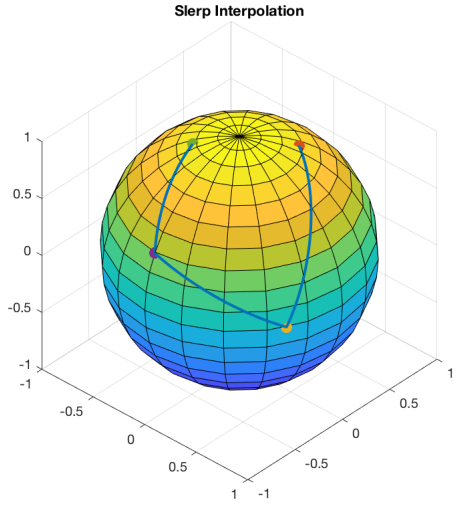
3.2 Spherical Linear Quaternion Interpolation : SLERP

SLERP interpolation performs a direct interpolation between two quaternions, along the shortest great arc of the quaternion unit sphere. It has the nice characteristic of also maintaining a constant angular velocity throughout the interpolation.

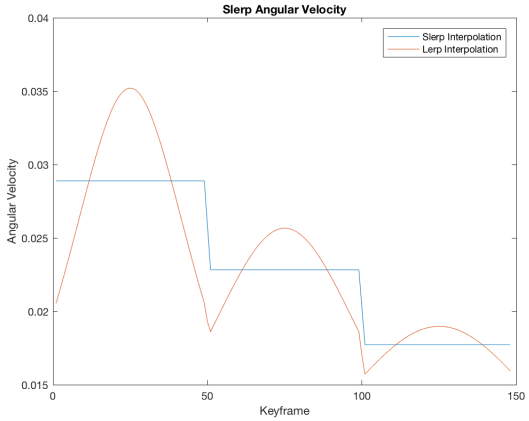
Before introducing SLERP, we will need definitions of the quaternion logarithm and exponential.

Let $q \in \mathbb{H}_1$ where $q = [\cos(\theta), \sin(\theta)\mathrm{v}]$. The logarithm is defined as

$$\log(q) \coloneqq [0, \theta v]$$



(a) Lerp/Slerp Interpolation on Unit Sphere



(b) Slerp and Lerp Angular Velocities

FIGURE 1 – Comparing Lerp and Slerp Interpolation

.

For a quaternion of the form $q = [0, \theta v]$, $\theta \in \mathbb{R}$, $v \in \mathbb{R}^3$, $|v| = 1$ we define the exponential function as

$$\exp(q) \coloneqq [\cos(\theta), \sin(\theta)v]$$

We can now define exponentiation as :

$$q^t \coloneqq \exp(t \log(q))$$

SLERP is defined with $p, q \in \mathbb{H}_1$ and $h \in [0, 1]$ where

$$SLERP(p, q, h) = p(\overline{p}q)^h$$

3.3 Bézier Quaternion curve

We can represent an n-th order Bézier curve with Bernstein basis

$$\beta_{i,n}(t) = \binom{n}{i}(1-t)^{n-i}t^i$$

in a cumulative form, where p_i are the Bézier control points, and $\beta_{i,n}$ are the Berstein basis functions :

$$p(t) = \sum_{i=0}^{n} p_i \beta_{i,n}(t)$$

For the Bézier curve given in a basis form, we can apply our quaternion curve construction method. We first reformulate the last equation :

$$p(t) = p_0 \tilde{\beta}_{0,n}(t) + \sum_{i=1}^{n} \Delta p_i \tilde{\beta}_{i,n}(t)$$

where the cumulative basis functions are given by :

$$\tilde{\beta}_{i,n}(t) = \sum_{j=i}^{n} \beta_{j,n}(t)$$

Then, by converting addition to multiplication, we can obtain the n-th order Bézier quaternion curve with control points $\{q_i\}$ as follows :

$$q(t) = \prod_{i=1}^{n} exp(\omega_i \tilde{\beta}_{i,n}(t))$$

where $\omega_i = log(q_{i-1}^{-1} q_i)$

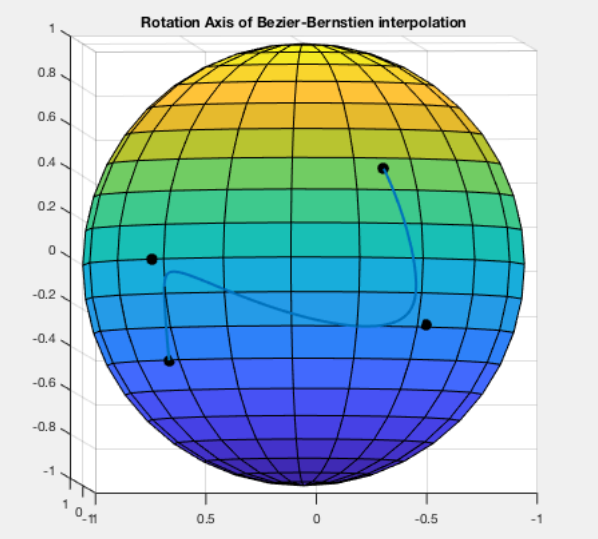


FIGURE 2 – An initial Bézier-Bernstein interpolation

A major drawback of Bézier curves is that the degree of the curve grows linearly with the number of control points. Therefore when interpolating between many points, they suffer from oscillation problems inherent to higher order polynomials. Furthermore, they have global control so updates to an individual interpolation point affects the whole curve.

3.4 B-spline Quaternion Curve

B-spline curves are popular in computer graphics due to their smoothness and local controllability. They exhibit local control, as they are a (k-1)-th order piecewise polynomials. Unlike Bézier curves, B-spline curve order does not grow with increasing number of control points, making them well suitable for interpolating and approximating many poses.

The b-spline basis functions $B_{i,k}$ are defined with the recursive de Boor algorithm.

$$B_{i,1}(t) = \begin{cases} 1 \text{ if } t_i < t < t_{i+1} \\ 0 \text{ otherwise} \end{cases}$$

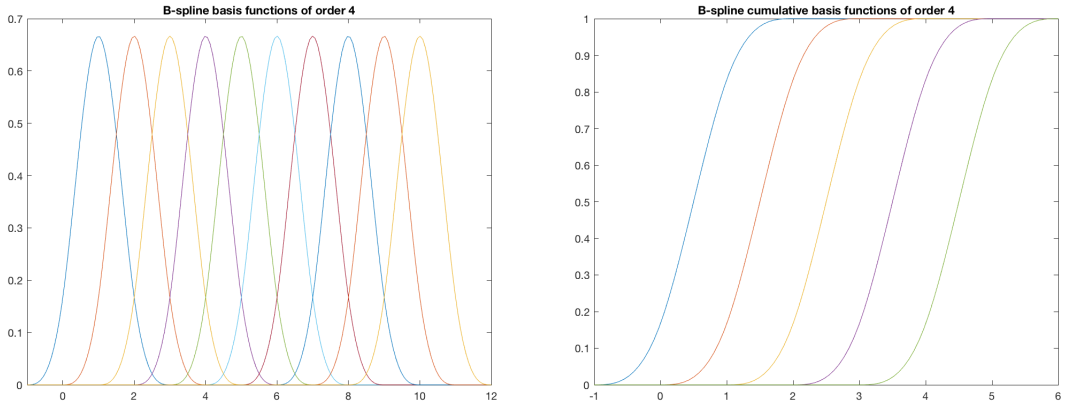
$$B_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} B_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} B_{i+1,k-1}(t)$$

They can be formulated in the following cumulative form.

$$p(t) = p_0 \tilde{B}_{0,k}(t) + \Sigma_{i=1}^{n} \Delta p_i \tilde{B}_{i,k}(t)$$

where

$$\tilde{B}_{i,k}(t) = \Sigma_{j=i}^{n} B_{j,k}(t)$$



(a) Basis functions (b) Cumulative basis functions

FIGURE 3 – B-spline Basis and Cumulative Basis Functions

Similar to the Bézier-Bernstien interpolation, B-splines are defined in the quaternion form, but with the cumulative b-spline basis functions.

$$q(t) = q_0^{\tilde{B}_{0,k}(t)} \prod_{i=2}^{n} exp(\omega_i \tilde{B}_{i,n}(t))$$

where $\omega_i = log(q_{i-1}^{-1} q_i)$.

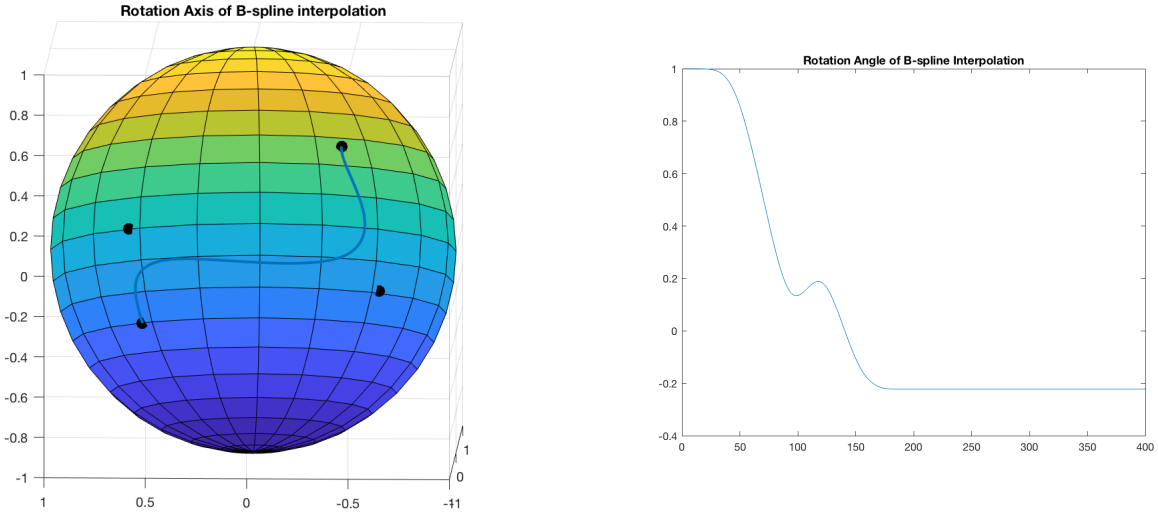


FIGURE 4 – B-spline Interpolation Rotation Axis and Angle

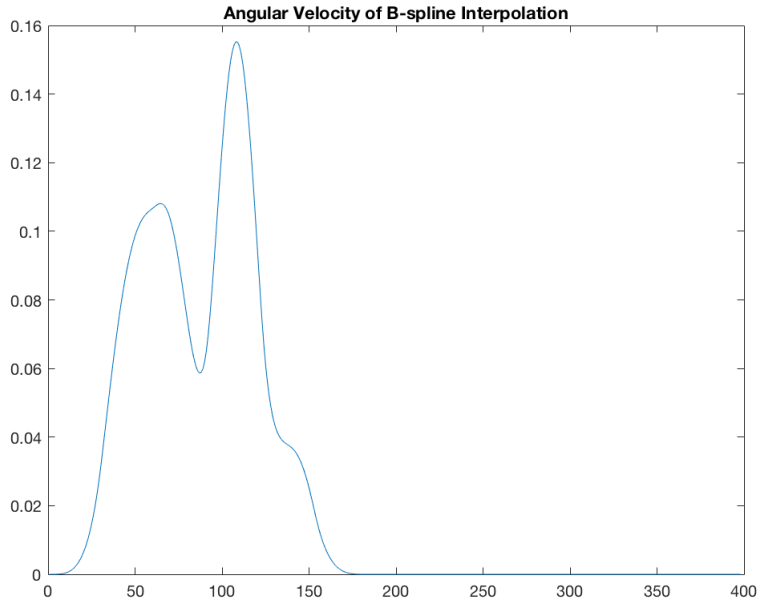


FIGURE 5 – B-spline Interpolation Angular Velocity

3.5 Optimization of Polynomial Interpolation Curves

Our main motivation in this project has been to calculate a trajectory that is both smooth and accurate, namely that total changes in angular velocity are small, and that each control point is interpolated as closely as possible. This can be formulated in the following optimization problem :

$$\min_q \sum ||q(t_i) - q_i|| + \lambda \int ||\ddot{q}(t)||^2 dt$$

where $\lambda \in \mathbb{R}_+$ a weight influencing the smoothness of the interpolation curve. Here $q(t)$ is a polynomial interpolation curve defined by control points p_i, which are initialized to be the values of the quaternion points we are interpolating over. The optimizer tunes the points p_i so that the curve minimizes the functional above.

This unconstrained optimization problem was solved in Matlab by using *fminunc* with the numerical gradient.

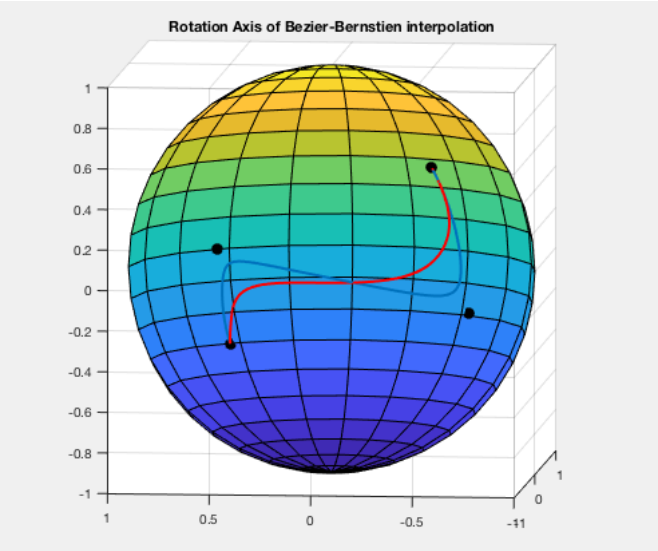


FIGURE 6 – Bézier Optimization with $\lambda = 2000$

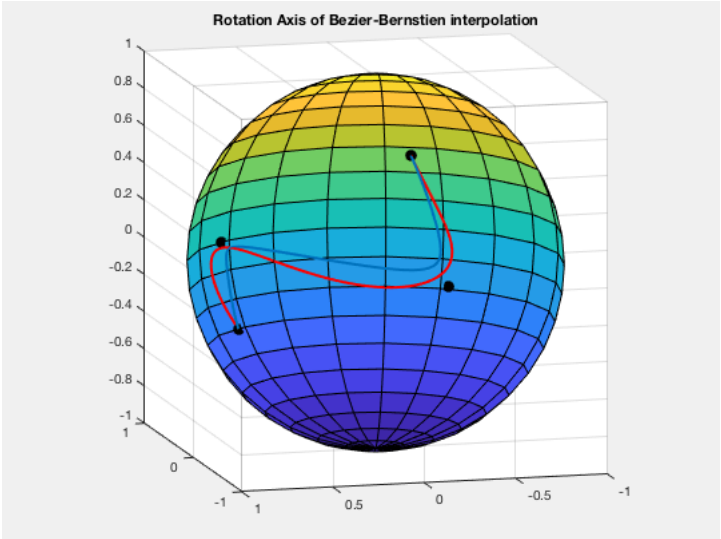


FIGURE 7 – Bézier Optimization with $\lambda = 200$

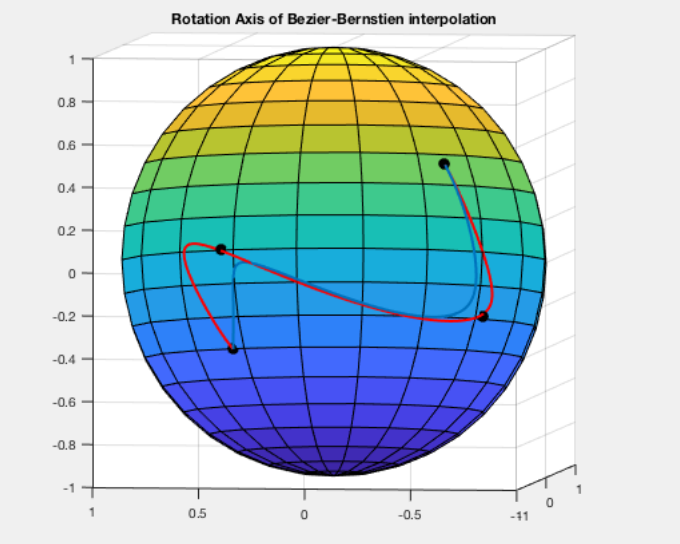


FIGURE 8 – Bézier Optimization with $\lambda = 20$

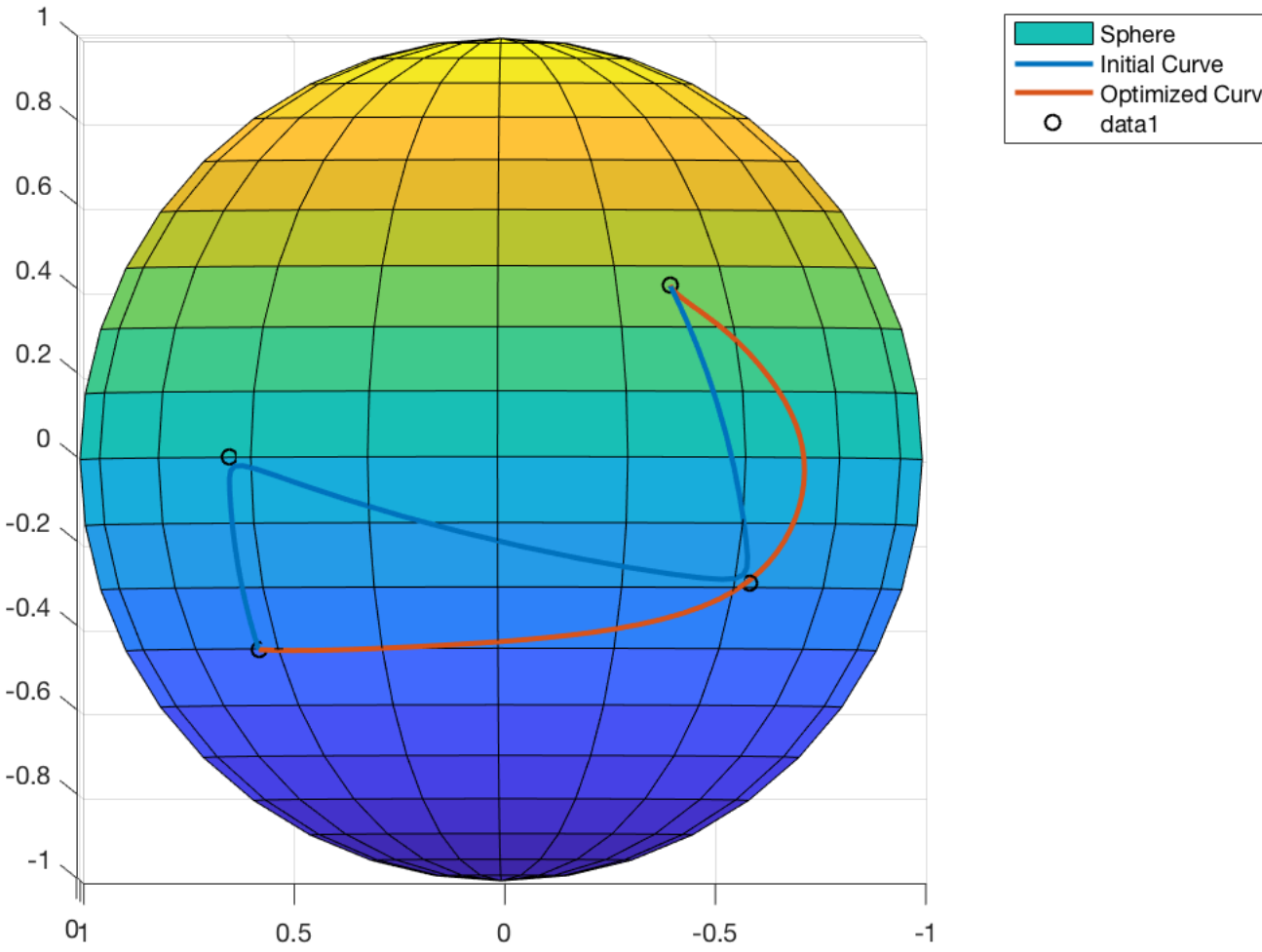


FIGURE 9 – B-spline Optimization with $\lambda = 20$

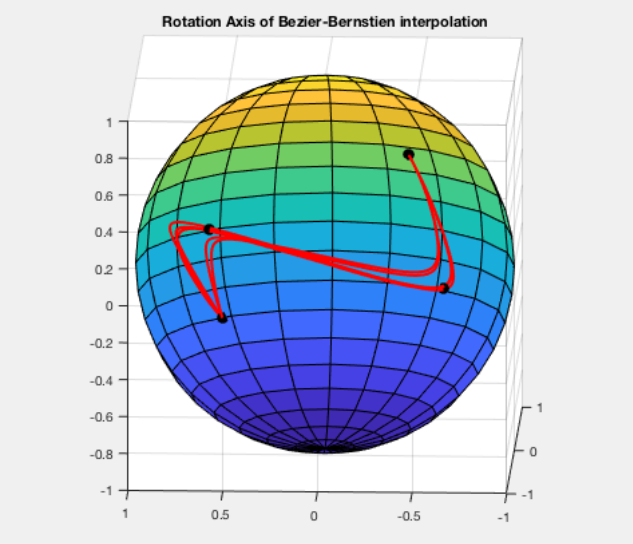


FIGURE 10 – Optimization step by step with $\lambda = 20$

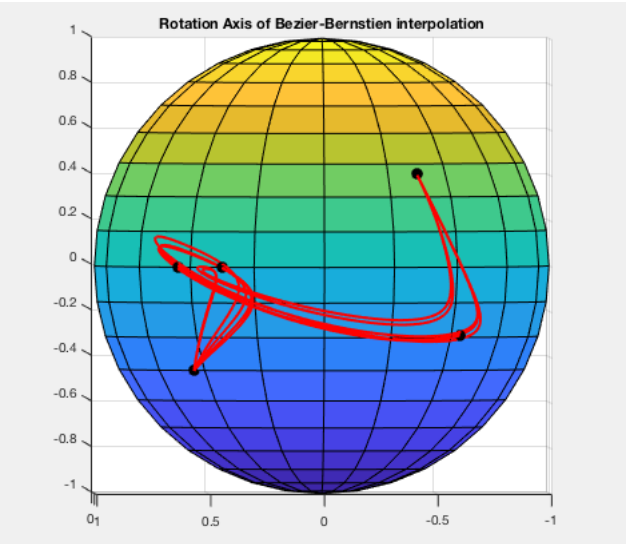


FIGURE 11 – Optimization step by step with $\lambda = 20$ and 5 quaternions

```
>> opt_bezier_bernstein

                                                          First-order
 Iteration  Func-count       f(x)         Step-size       optimality
     0          37         0.147583                        0.0224
     1         111         0.122438             10         0.0185
     2         148        0.0223001              1        0.00688
     3         185        0.0143763              1        0.00306
     4         222        0.0132457              1        0.00215
     5         259        0.0130173              1        0.00139
     6         296        0.0127847              1       0.000702

Solver stopped prematurely.

fminunc stopped because it exceeded the iteration limit,
options.MaxIterations = 5 (the selected value).
```

FIGURE 12 – Iterations of the Matlab solver *fminunc* with $\lambda = 20$ and 5 quaternions.*fminunc* stands for function minimization unconstrained. *func-count* reports the total number of objective function evaluations.

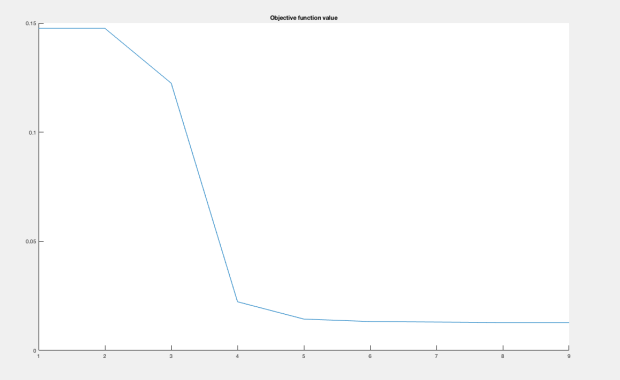


FIGURE 13 – Objective value with $\lambda = 20$, 5 quaternions, 9 iterations

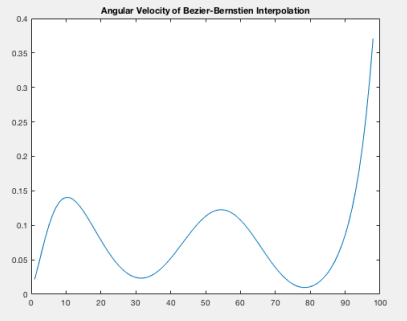


FIGURE 14 – Angle velocity for a Bézier-Bernstein curve

3.6 Distance

There are several ways we can imagine to construct a distance between two quaternions. They are especially important in the otpimization model.

Euclidean distance : $||q_{orig,i} - q_{calc,i}||_2^2$

Angular distance : $cos^{-1}(2 < q_{orig,i}, q_{calc,i} > -1)$

Even if the angular distance seems to make more sense, we used in our optimization solver the euclidean norm, for the good reason that was solved faster.

3.7 Running time

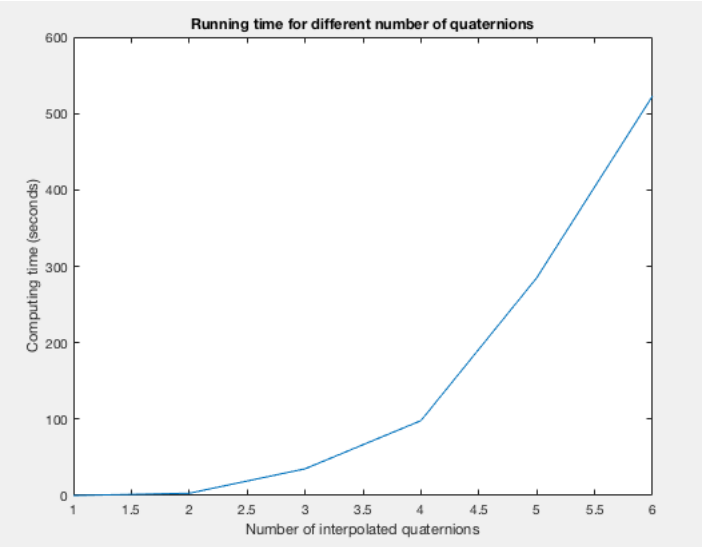


FIGURE 15 – Running time for different number of quaternions

So, you can notice that the algorithm takes almost 10 minutes for only 6 quaternions, this time explodes when you use even more quaternions.

Conclusion : the estimated gradient method is definitely not working for real-time data.

3.8 Interpolation Bases

While we initially believed that B-spline curves would provide more control and allow for better optimization, they performed inconsistently under optimization. We hypothesized this was due to the amount of degrees of freedom provided by b-splines and their local control. From this point on we only consider bézier curves in our optimization models.

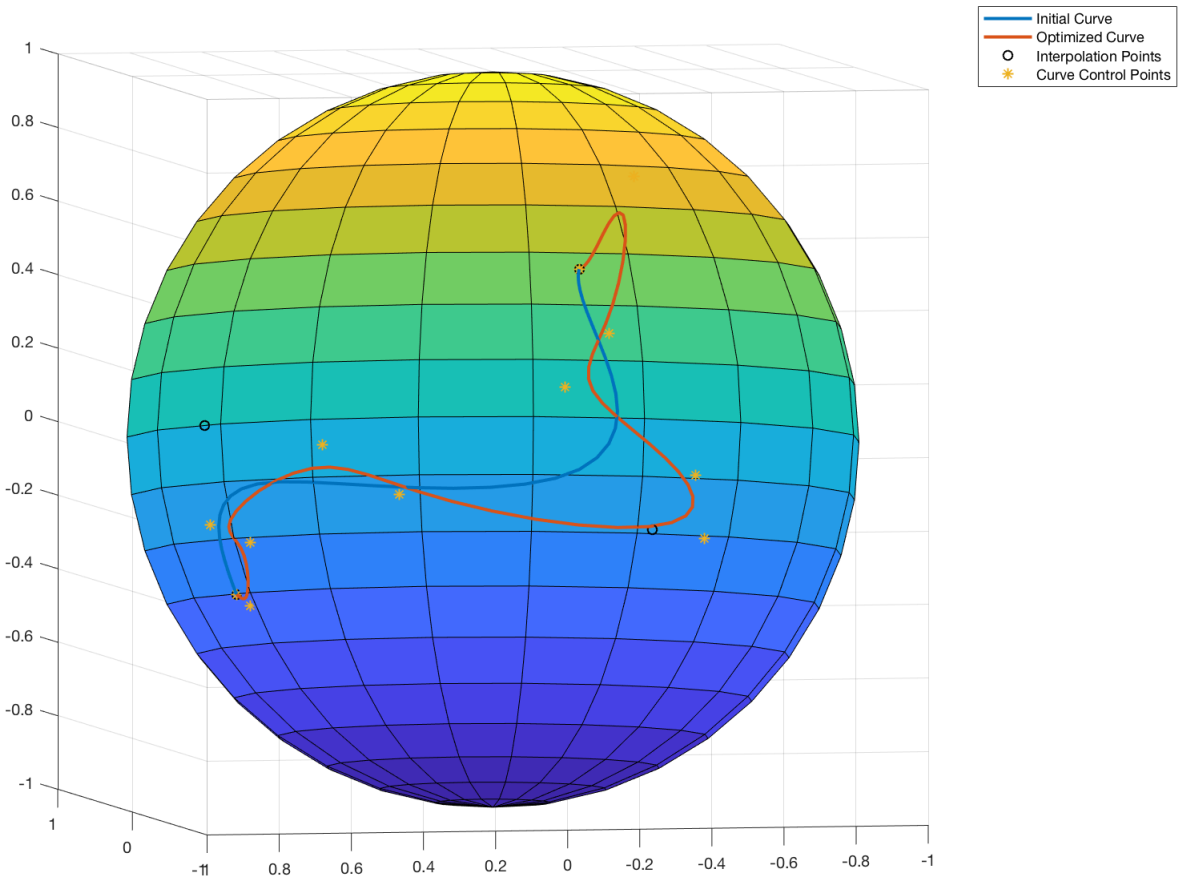


FIGURE 16 – Strange minimum found using B-spline basis curves

4 Deriving the Gradient

Let's rewrite the objective function in a more mathematical way. If we consider the B-spline interpolation, we have to optimize our objective function **regarding the control points** q_i. Those control points are of curse quaternions as well. To give us some idea, let's fix some numerical values.

— N = 120 : number of keyframes along the all trajectory.

— M = 4 : number of original quaternions $q_1^{(O)}, q_2^{(O)}, q_3^{(O)}, q_4^{(O)}$ we want to interpolate

— m = 3 : number of control points pro original quaternion

— n = m * M = 12 : total number of control points, ie $q_1, ..., q_{12}$.

— k = 4 : order of our B-spline basis functions

If we put our all $q_1, ..., q_{12}$ into a matrix V, like : $V = \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_{12} \end{pmatrix}$

Recall the expression of the B-spline curve :

$$q(t, V) = q_1^{\tilde{B}_{1,k}(t)} \prod_{i=2}^{n} exp(\omega_i \tilde{B}_{i,k}(t))$$

where $\omega_i = log(q_{i-1}^{-1} q_i)$, for t=1..N.
Here is an explicit formulation of our objective function :

$$\min_V ||q(1,V) - q_1^{(O)}||^2 + ||q(40,V) - q_3^{(O)}||^2 + ||q(80,V) - q_3^{(O)}||^2 + ||q(120,V) - q_4^{(O)}||^2 + \lambda \int ||\ddot{q}(t,V)||^2 dt$$

for some λ controlling the smoothness weight.

If our problem is convex, it could be very good to execute the following classical gradient descent :
$V^{k+1} = V^k - \tau \nabla F(V^k)$ where F is our objective function.

Let's try to compute this gradient for the very first term of F, that is to say $||q(1,V) - q_1^{(O)}||^2$

$\nabla = 2\nabla_V q(1,V)(q(1,V) - q_1^{(O)})$

Here is a first problem, because it's unclear regarding the dimension of those objects. V is a vector of quaternions, ie a matrix of size (12, 4).

As for the explicit form of the derivatives that we had to calculate, most of the literature that we found worked with minimization problems with respect to t, so if that would have been our case, for B-splines it was a matter of using the simple and stated derivation formula for the Basis : $\frac{d}{dt}\Sigma\alpha_i B_{i,k}(t)$.

The literature related to the derivatives of the power function like $f(q) = q^n$ also gave us the hint that it was not valid to use the traditional product rules that we know from calculus.

But the biggest issue we came across is that we don't have any closed-form for the derivative of the *quaternion logarithm*. And this is exactly what's used in ω_i formula.

And even if we would have successfully calculated a closed-form for the gradient, this would quite tricky to write in Matlab, because we need first to transform our V matrices into vectors, to avoid dimension mismatch. And we are not talking about potential normalization up to this point.

Conclusion : In the whole project, we only used numerical estimated gradient that Matlab can compute itself. This was not problematic, provided that we were not having to many quaternions to interpolate.

5 Sensor Fusion

5.1 Background

Part of the goal of this project was to combine OTS (camera poses) and IMU (accelerometer poses) for a more accurate trajectory. Framos software enabled us to capture both streams of data using the following ArUco marker, and IMU. Data was also calibrated via Framos software.

FIGURE 17 – ArUco Marker and IMU

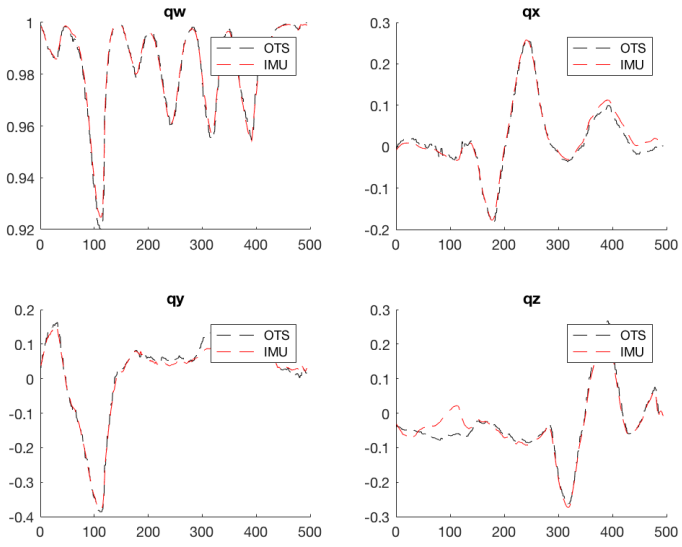


FIGURE 18 – OTS and IMU poses do not always align, some regions contain high degrees of noise.

5.2 Undesirable Minima

Due to the angular norm penalizing anti-podal points equally, the unconstrained optimization model above found minima by stretching the curve across the quaternion hypersphere. This resulted in undesirable interpolation curves, as can be seen in the image below.

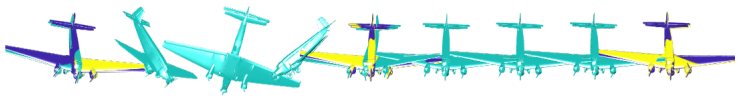


FIGURE 19 – Undesirable minimum found on IMU and OTS data

As can be seen, the interpolation curve performs an unnecessary complete 360° rotation. This can also be seen in the following angular velocity curve, where $cos(\theta)$ should remain at 1 throughout the curve, but wraps completely around to -1.

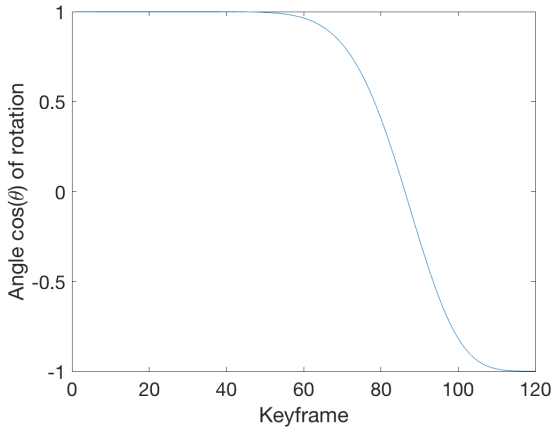


FIGURE 20 – Rotation Angle of undesirable minimum at anti-podal point

5.3 Constrained Optimization

Constraining the quaternions to the northern-hemisphere of the quaternion hypersphere resulted our final optimization mode.

— J The set of captured OTS poses

— I The set of captured IMU poses

— Quaternions are constrained to Northern Hemisphere

$$\min_{p} \quad \lambda_1 \sum_{i \in I} ||q(p,t_i) - q_i|| + \lambda_2 \sum_{j \in J} ||q(p,t_j) - q_j|| + \int ||\ddot{q}(p,t)||^2 dt$$

s.t. $q_0 \geq 0$

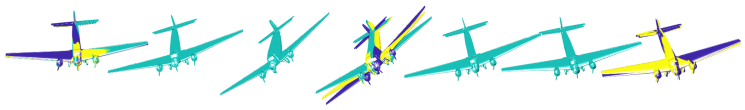


FIGURE 21 – Sensor Fusion Constrained Optimization. Purple is IMU pose, yellow is OTS pose.

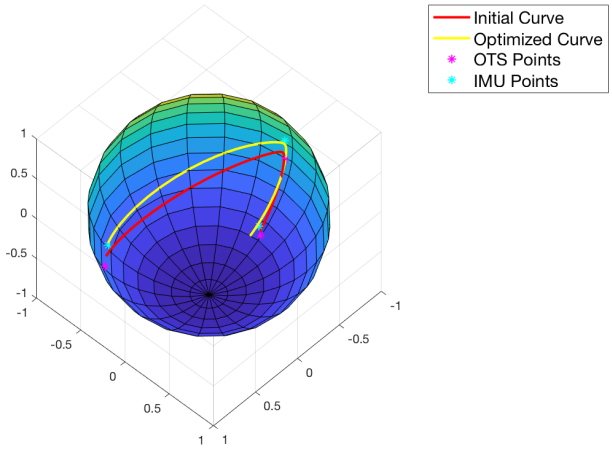


FIGURE 22 – Sensor Fusion Interpolation Rotation Axis

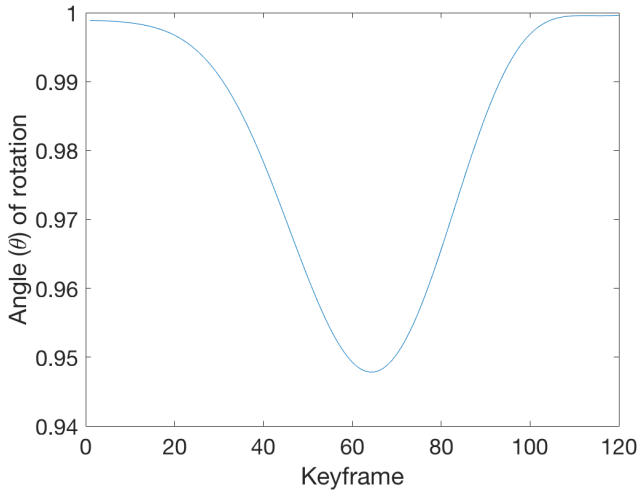


FIGURE 23 – Sensor Fusion Interpolation Rotation Axis

The resulting optimization model was very sensitive to perturbations in the starting point. The most sensible starting point seemed to be using the Markey Quaternion Average [7]. This yielded very nice convergence. However, slight perturbations to this starting point yielded very poor results, as exhibited in the figures below.

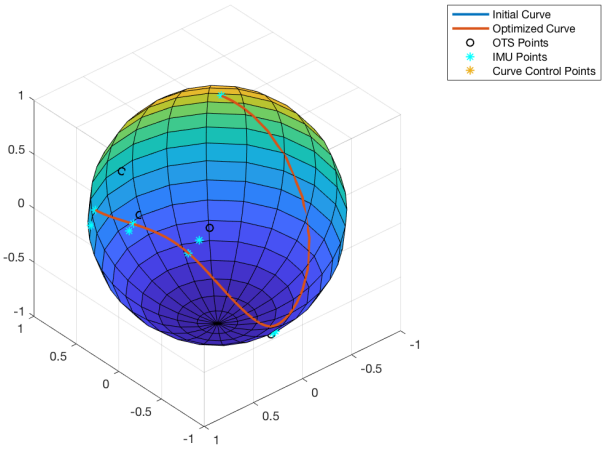


FIGURE 24 – Rotation axis with averaged starting points

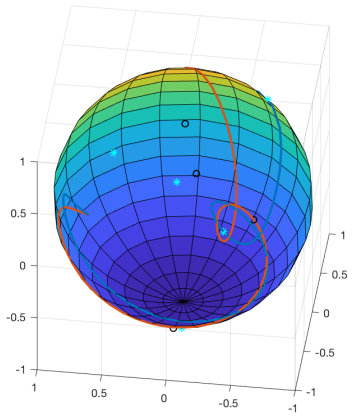


FIGURE 25 – Rotation axis with averaged starting points and slight perturbations $e_i \sim N(0, 0.1)$

The final optimization model yielded repeatable and stable results. The final implementation can be seen in the LRZ gitlab, under the branch "fusion". They are run using fmincon using the *interior-point* method. Using the markey average as a starting point usually resulted in convergence of 1-2 iterations. However, as seen in the figure above slight perturbations severely affected both the convergence and the run time.

6 Comments about the code in Matlab

— **polynomial-interpolation** : in this folder, by running *run_optimization.m*, you will be able to run the optimization algorithm for a couple of quaternions and to visualize the results on the unit quaternion sphere. You can use

either Bézier-Bernstein or B-spline interpolation methods. Notice the gitlab branch "fusion" which contains integration of OTS and IMU data, as well as constrained optimization mentioned above.

— **ExampleSlerpMatlab** : Contains several basic interpolation methods (slerp, lerp).

— **Live Visualization** : Contains code used to make the live visualization for the poster presentation, and well as some code to make plots of 3D objects.

— **DualSlerp** : Contains implementation of Dual Quaternion Sclerp (analogous to slerp) using numerically stable dual quaternion exponential and logarithm provided by framos.

— **cpp_animation** : Contains a point cloud slerp animation written in C++ used to generate our poster graphic.

— **Anim_Rot.m** : Animation of live slerp interpolation used during presentation.

7 Bibliography

1. Benjamin Busam, Tolga Birdal, and Nassir Navab. Camera pose filtering with local regression geodesics on the riemannian manifold of dual quaternions. arXiv preprint arXiv :1704.07072, 2017.

2. Benjamin Busam, Marco Esposito, Benjamin Frisch, and Nassir Navab. Quaternionic upsampling : Hyperspherical techniques for 6 dof pose tracking. In 3D Vision (3DV), 2016 Fourth International Conference on, pages 629–638. IEEE, 2016.

3. Erik B Dam, Martin Koch, and Martin Lillholm. Quaternions, interpolation and animation, volume 2. Datalogisk Institut, Københavns Universitet Copenhagen, 1998.

4.V. Lepetit and P. Fua. Monocular model-based 3d tracking of rigid objects : A survey. Foundations and Trends in Computer Graphics and Vision, 1(1) :1–89, 2005.

5. Myoung-Jun Kim, Myung-Soo Kim, and Sung Yong Shin2. A General Construction Scheme for Unit Quaternion Curves with Simple High Order Derivatives.

6. Ben Kenwright. A Beginners Guide to Dual-Quaternions : What They Are, How They Work, and How to Use Them for 3D Character Hierarchies. School of Computing Science, Newcastle University

7. Richard Hartley, Jochen Trumpf, Yuchao Dai, and Hongdong Li. Rotation averaging. Interna- tional journal of computer vision, 103(3) :267–305, 2013.

8. On the Differentiability of Quaternion Functions. A. Razmadze Mathematical Institute of I. Javakhishvili Tbilisi State University, 2, University St., Tbilisi 0186, Georgia

9. Dongpo Xu, Cyrus Jahanchachi, Clive C. Took and Danilo P. Mandic. Quaternion Derivatives : The GHR Calculus. 2004.