Frame position and orientation error Jacobians

Nick Bianco

1 Frame-based inverse kinematics

Given an objective function $f(\mathbf{x})$, we would like to compute the partial derivatives with respect to the optimization design variables, $\mathbf{x} \in \mathbb{R}^n$,

$$\frac{\partial f}{\partial \mathbf{x}} = \left[\frac{\partial f}{\partial x_1} \cdots \frac{\partial f}{\partial x_n} \right]$$

This will enable efficient computation of derivatives in gradient-based optimization methods, e.g., solving a non-linear program (NLP) with IPOPT. The goal for this document is to define functions, and their derivatives, that allow us to efficiently perform inverse kinematics given data representing the position and orientation of frames in a model.

2 Cost function

We will define two functions that represent the position and orientation error between a model's frame and the reference data frame. The position error is simply the squared norm of the difference between the model's frame position, $\vec{\mathbf{p}}(\mathbf{q})$, and the position of the reference data, $\hat{\vec{\mathbf{p}}}$,

$$f^p(\mathbf{q}) = ||\vec{\mathbf{p}}(\mathbf{q}) - \hat{\vec{\mathbf{p}}}||_2^2$$

where $\mathbf{q}$ represents the model's generalized coordinate values (e.g., joint angles) and the "hat" symbol denotes a quantity from the reference data (i.e., not a function of $\mathbf{q}$).

There are several different ways we could represent the orientation error. Here, we will use a representation based on the distance between two quaternions (see [2] and [1]),

$$f^\epsilon(\mathbf{q}) = 1 - (\vec{\epsilon}(\mathbf{q}) \cdot \hat{\vec{\epsilon}})^2$$

where $\vec{\epsilon}(\mathbf{q})$ is a quaternion representing the current orientation of a model frame and $\hat{\vec{\epsilon}}$ is the frame orientation from the reference data. This representation has several advantages. First, this function is bounded between $[0, 1]$: it gives 0 whenever the quaternions represent the same orientation, and it gives 1 whenever the two orientations are 180 degrees apart. Second, it is free from expensive trigonometric operations (although in Simbody we will eventually need

to convert from a rotation matrix to a quaternion, which has a fixed cost of 40 flops). Lastly, as we will see below, it will be relatively straightforward to find the derivative of this function with respect to $\mathbf{q}$, which will be crucial for finding the Jacobian of this cost function term.

The total cost function is a weighted sum of the frame position and orientation errors,

$$f(\mathbf{q}) = w^p f^p(\mathbf{q}) + w^R f^\epsilon(\mathbf{q})$$

3 Frame Jacobians

In order to compute the derivative of the total cost function, $f(\mathbf{q})$, we will first need a way to compute the Jacobians of individual frames. Luckily, Simbody provides methods for computing frame Jacobians that we can leverage directly.

3.1 Basic frame Jacobian

The "basic" frame Jacobian (see [3]) relates generalized speeds (for now, assuming $\mathbf{u} = \dot{\mathbf{q}}$) to spatial velocities,

$${}^G\vec{\mathbf{V}}^F = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$$

where the spatial velocity is a 6×1 vector containing the angular and linear velocity of the frame F with respect to ground G,

$${}^G\vec{\mathbf{V}}^F = \begin{bmatrix} {}^G\vec{\omega}^F \\ {}^G\vec{\mathbf{v}}^F \end{bmatrix} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ v_x \\ v_y \\ v_y \end{bmatrix}$$

For a single frame, $\mathbf{J}(\mathbf{q})$ is a $6 \times n_{\mathbf{u}}$ matrix,

$$\mathbf{J}(\mathbf{q}) = \begin{bmatrix} \mathbf{z}_0 & \mathbf{z}_1 & \cdots & \mathbf{z}_n \\ \frac{\partial \vec{\mathbf{p}}}{\partial q_0} & \frac{\partial \vec{\mathbf{p}}}{\partial q_1} & \cdots & \frac{\partial \vec{\mathbf{p}}}{\partial q_n} \end{bmatrix}$$

where $n_{\mathbf{u}}$ is the number of generalized speeds and $\mathbf{z_i}$ is the axis of rotation about which $\dot{q}_i$ contributes to the total angular velocity vector, $\vec{\omega} = \sum_i^n \mathbf{z}_i \dot{q}_i$ [3]. We can define the component Jacobians $\mathbf{J}^v$ and $\mathbf{J}^\omega$ such that,

$$\mathbf{J}(\mathbf{q}) = \begin{bmatrix} \mathbf{J}^\omega \\ \mathbf{J}^v \end{bmatrix}$$

$$\mathbf{J}^\omega = \begin{bmatrix} \mathbf{z}_0 & \mathbf{z}_1 & \cdots & \mathbf{z}_n \end{bmatrix}$$

$$\mathbf{J}^v = \begin{bmatrix} \frac{\partial \vec{\mathbf{p}}}{\partial q_0} & \frac{\partial \vec{\mathbf{p}}}{\partial q_1} & \cdots & \frac{\partial \vec{\mathbf{p}}}{\partial q_n} \end{bmatrix}$$

The Simbody method `SimbodyMatterSubsystem::calcFrameJacobian()` provides a direct way for calculating $\mathbf{J}(\mathbf{q})$. However, when performing matrix-vector products with $\mathbf{J}(\mathbf{q})$, it is more efficient to use the methods `SimbodyMatterSubsystem::multiplyByFrameJacobian()` (i.e., $\mathbf{J}(\mathbf{q})\mathbf{u}_{like}$) and `SimbodyMatterSubsystem::multiplyByFrameJacobianTranspose()` (i.e., $\mathbf{J}^\top(\mathbf{q})\mathbf{F}_{like}$), where $\mathbf{u}_{like}$ is a vector of generalized speeds (or similar) and $\mathbf{F}_{like}$ is the vector of spatial forces applied to frame F (or similar).

3.2 Frame representation

To connect the basic frame Jacobian $\mathbf{J}(\mathbf{q})$ to our cost function Jacobians, we first need to be explicit about the frame representation as implied our choice of cost function. $f^p(\mathbf{q})$ requires using a Cartesian representation for positions,

$${}^G\vec{\mathbf{p}}^F = \begin{bmatrix} p_x & p_y & p_z \end{bmatrix}$$

and $f^\epsilon(\mathbf{q})$ requires a quaternion representation for rotations:

$${}^G R^F = \vec{\epsilon} = \begin{bmatrix} \epsilon_0 & \epsilon_1 & \epsilon_2 & \epsilon_3 \end{bmatrix}$$

Simbody provides the method `Rotation::convertRotationToQuaternion()` to convert a rotation matrix ${}^G R^F$, to an equivalent unit (length 1) quaternion, $\vec{\epsilon}$, which costs 40 flops.

Given our frame representation, we need to extend the relationships given by the basic Jacobian,

$$\begin{bmatrix} {}^G\vec{\omega}^F \\ {}^G\vec{\mathbf{v}}^F \end{bmatrix} = \begin{bmatrix} \mathbf{J}^\omega \\ \mathbf{J}^v \end{bmatrix} \dot{\mathbf{q}} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}}$$

to find relationships between $\dot{\mathbf{q}}$ and the derivatives of our task representations, $\dot{\vec{\epsilon}}$ and ${}^G\dot{\vec{\mathbf{p}}}^F$. This amounts to finding the representation-specific Jacobians, $\mathbf{J}^\epsilon$ and $\mathbf{J}^p$,

$$\begin{bmatrix} \dot{\vec{\epsilon}} \\ {}^G\dot{\vec{\mathbf{p}}}^F \end{bmatrix} = \begin{bmatrix} \mathbf{J}^\epsilon \\ \mathbf{J}^p \end{bmatrix} \begin{bmatrix} \mathbf{J}^\omega \\ \mathbf{J}^v \end{bmatrix} \dot{\mathbf{q}}$$

3.3 Frame position Jacobian

Since ${}^G\dot{\vec{\mathbf{p}}}^F = {}^G\vec{\mathbf{v}}^F$, the Jacobian for frame positions is simply $\mathbf{J}^p = \mathbf{J}^v$.

3.4 Frame orientation Jacobian

For the frame orientation Jacobian, we use a "task-specific" Jacobian that relates angular velocities to the time derivatives of quaternions[1],

$$\mathbf{J}^{\epsilon} = \frac{1}{2}\begin{bmatrix} -\epsilon_1 & -\epsilon_2 & -\epsilon_3 \\ \epsilon_0 & \epsilon_3 & -\epsilon_2 \\ -\epsilon_3 & \epsilon_0 & \epsilon_1 \\ \epsilon_2 & -\epsilon_1 & \epsilon_0 \end{bmatrix}$$

where

$$\dot{\vec{\epsilon}} = \mathbf{J}^{\epsilon}\vec{\omega}$$

We can then relate generalized speeds to quaternion time derivatives using the composition $\mathbf{J}^{\epsilon}$ and $\mathbf{J}^{\omega}$,

$$\dot{\vec{\epsilon}} = \mathbf{J}^{\epsilon}\mathbf{J}^{\omega}\dot{\mathbf{q}}$$

4 Frame position error

Now we have all the elements needs to compute frame errors between a model and reference data. Again, the frame position error function,

$$f^{p}(\mathbf{q}) = ||\vec{\mathbf{p}}(\mathbf{q}) - \hat{\vec{\mathbf{p}}}||_2^2$$

Then, taking the derivative of $f^{p}(\mathbf{q})$ with respect to $\mathbf{q}$,

$$\frac{\partial f^{p}}{\partial \mathbf{q}} = 2\frac{\partial \vec{\mathbf{p}}}{\partial \mathbf{q}} \cdot (\vec{\mathbf{p}} - \hat{\vec{\mathbf{p}}}) = 2\mathbf{J}^{v\top}(\vec{\mathbf{p}} - \hat{\vec{\mathbf{p}}})$$

5 Frame orientation error

Similarly, the frame orientation error function,

$$f^{\epsilon}(\mathbf{q}) = 1 - (\vec{\epsilon}(\mathbf{q}) \cdot \hat{\vec{\epsilon}})^2$$

Then, taking the derivative of $f^{\epsilon}(\mathbf{q})$ with respect to $\mathbf{q}$,

$$\frac{\partial f^{\epsilon}}{\partial \mathbf{q}} = -2(\vec{\epsilon}(\mathbf{q}) \cdot \hat{\vec{\epsilon}})\frac{\partial \vec{\epsilon}(\mathbf{q}) \cdot \hat{\vec{\epsilon}}}{\partial \mathbf{q}} = -2(\vec{\epsilon}(\mathbf{q}) \cdot \hat{\vec{\epsilon}})\mathbf{J}^{\omega\top}\mathbf{J}^{\epsilon\top}\hat{\vec{\epsilon}}$$

[1] The definition of $\mathbf{J}^{\epsilon}$ in the 2015 edition of Paul Mitiguy's *Advanced Dynamics & Motion Simulation* book [4] differs from the Simbody implementation, and it is unclear to me as to why. The $\mathbf{J}^{\epsilon}$ defined here is based on the Simbody implementation.

6 Cost function Jacobian

So far, we have defined $f^p(\mathbf{q})$ and $f^\epsilon(\mathbf{q})$ for an individual frames, but we will need to compute errors across multiple frames simultaneously. Therefore, we will use $f_j^p(\mathbf{q})$ and $f_j^\epsilon(\mathbf{q})$ to denote the frame position and orientation error, respectively, for the j-th frame in our inverse kinematics problem. Given the total number of frames, n_F, we can find the total frame position and orientation errors by summing across all the individual frame errors,

$$f^p(\mathbf{q}) = \sum_j^{n_F} f_j^p(\mathbf{q})$$

$$f^\epsilon(\mathbf{q}) = \sum_j^{n_F} f_j^\epsilon(\mathbf{q})$$

We can similarly find the total frame position and orientation error Jacobians by summing all of the individual frame Jacobians,

$$\frac{\partial f^p}{\partial \mathbf{q}} = \sum_j^{n_F} [2\mathbf{J}^{v\top}(\vec{\mathbf{p}} - \hat{\vec{\mathbf{p}}})]_j$$

$$\frac{\partial f^\epsilon}{\partial \mathbf{q}} = \sum_j^{n_F} [-2(\vec{\epsilon}(\mathbf{q}) \cdot \hat{\vec{\epsilon}})\mathbf{J}^{\omega\top}\mathbf{J}^{\epsilon\top}\hat{\vec{\epsilon}}]_j$$

Given the total cost function,

$$f(\mathbf{q}) = w^p f^p(\mathbf{q}) + w^R f^\epsilon(\mathbf{q})$$

we can now write the Jacobian of the total cost function,

$$\frac{\partial f}{\partial \mathbf{q}} = w^p \frac{\partial f^p}{\partial \mathbf{q}} + w^R \frac{\partial f^\epsilon}{\partial \mathbf{q}} = \sum_j^{n_F} [2w^p \mathbf{J}^{v\top}(\vec{\mathbf{p}} - \hat{\vec{\mathbf{p}}}) - 2w^R(\vec{\epsilon}(\mathbf{q}) \cdot \hat{\vec{\epsilon}})\mathbf{J}^{\omega\top}\mathbf{J}^{\epsilon\top}\hat{\vec{\epsilon}}]_j$$

7 Handling $\dot{\mathbf{q}} \neq \mathbf{u}$

We have assumed so far that $\dot{\mathbf{q}} = \mathbf{u}$, but this is not true for every system. For example, models with ball joints may represent rotations with quaternions, which permit orientations free of singularities. As we have seen above in the definition of $\mathbf{J}^\epsilon$, in general, $\dot{\mathbf{q}} \neq \mathbf{u}$. For example, the general relationship between generalized speeds and quaternion time derivatives is actually the following,

$$\dot{\vec{\epsilon}} = \mathbf{J}^\epsilon \mathbf{J}^\omega \mathbf{u}$$

In a multibody system, the relationship between $\dot{\mathbf{q}}$ and $\mathbf{u}$ is represented by the kinematical differential equations,

$$\dot{\mathbf{q}} = \mathbf{N}(\mathbf{q})\mathbf{u}$$

We can use these equations to find relationships in terms of $\dot{\mathbf{q}}$, which we need for our Jacobian functions. For example, the general relationship between $\dot{\mathbf{q}}$ and $\dot{\vec{\epsilon}}$,

$$\dot{\vec{\epsilon}} = \mathbf{J}^{\epsilon}\mathbf{J}^{\omega}\mathbf{N}^{-1}\dot{\mathbf{q}}$$

We an similarly substitute the kinematical differential equations $\dot{\mathbf{q}} = \mathbf{N}(\mathbf{q})\mathbf{u}$ elsewhere to develop Jacobian functions in terms of $\dot{\mathbf{q}}$, which is needed when providing Jacobians to optimization solvers when our design variables are exactly $\mathbf{q}$.

References

[1] J. Belk. "Mathematics Stack Exchange: Quaternion Distance". URL: https://math.stackexchange.com/questions/90081/quaternion-distance.

[2] D.Q. Huynh. "Metrics for 3D Rotations: Comparison and Analysis". In: *Journal of Mathematical Imaging and Vision* 35 (2 2009). DOI: https://doi.org/10.1007/s10851-009-0161-2.

[3] O. Khatib. *Introduction to Robotics*. Stanford University, 1994. URL: https://see.stanford.edu/materials/aiircs223a/handout4_Jacobian.pdf.

[4] P. Mitiguy. *Advanced Dynamics & Motion Simulation*. 2015.