Planning with Attitude

Brian Jackson[1] and Zachary Manchester[1]

I. INTRODUCTION

Many useful robotic systems—including quadrotors, airplanes, satellites, autonomous underwater vehicles, and quadrupeds—can perform arbitrarily large three-dimensional translations and rotations as part of their normal operation. While representing translations is straightforward and intuitive, effectively representing the nontrivial group structure of 3D rotations has been a topic of study for many decades. Although we can intuitively deduce that rotations are three-dimensional, a globally non-singular three-parameter representation of the space of rotations does not exist [25]. As a result, when parameterizing rotations, we must either a) pick a three-parameter representation and deal with discontinuities, or b) pick a higher-dimensional representation and deal with constraints between the parameters. While simply representing attitude is nontrivial, generating and tracking motion plans for floating-base systems is an even more challenging problem.

Early work on control problems involving the rotation group dates back to the 1970s, with extensions of linear control theory to spheres [4] and $SO(3)$ [3]. Effective attitude tracking controllers have been developed for satellites [29], quadrotors [8, 18, 17, 12, 27, 21], and a 3D inverted pendulum [5] using various methods for calculating three-parameter attitude errors.

More recently, these ideas have been extended to trajectory generation [32], sample-based motion planning [33, 15], and optimal control. Approaches to optimal control on attitude problems include analytical methods applied to satellites [24], discrete mechanics [14, 13, 16], a combination of sampling-based planning and constrained trajectory optimization for satellite formations [9, 2], projection operators [22], or more general theory for optimization on manifolds [28]. Nearly all of these methods rely heavily on principles from differential geometry and Lie group theory; however, despite these works, many recent papers in the robotics community continue to apply traditional methods for motion planning and control with no regard for the group structure of rigid body motion [1, 6, 30, 10].

In this paper, we make a departure from previous approaches to geometric planning and control that rely heavily on ideas and notation from differential geometry, and instead use only basic mathematical tools from linear algebra and calculus that should be familiar to most roboticists. Similar to [19, 31], in Sec. III we introduce a quaternion differential calculus, but take a significantly simpler and more general approach that unifies both planning and control, enabling straight-forward adaptation of existing algorithms to systems with quaternion states. To make this concrete, in Sec. IV we apply our approach to derive an extension to the linear-quadratic regulator (LQR) that we call multiplicative LQR (MLQR), which is the control dual to the multiplicative extended Kalman filter (MEKF) from the state estimation literature [20]. In Sec. **??** we provide several simulation results demonstrating the application of our method to tracking control and constrained trajectory optimization. In summary, our contributions include:

- A unified approach to quaternion differential calculus entirely based on standard vector calculus and linear algebraic operations
- Derivation of multiplicative LQR, an adaptation of LQR to the control of systems with quaternion states
- A set of benchmark motion-planning and trajectory-tracking problems in which we compare our approach to existing methods commonly used in the robotics community to account for 3D rotations

[1]Robotics Institute, Carnegie Mellon University, 5000 Forbes Ave, Pittsburgh, PA, USA

II. BACKGROUND

We begin by defining some useful conventions and notation. Attitude is defined as the rotation from the robot's body frame to a global inertial frame. We also define gradients to be row vectors, that is, for $f(x) : \mathbb{R}^n \to \mathbb{R}$, $\frac{\partial f}{\partial x} \in \mathbb{R}^{1\times n}$.

A. Unit Quaternions

We leverage the fact that quaternions are linear operators and that the space of quaternions $\mathbb{H}$ is isomorphic to $\mathbb{R}^4$ to explicitly represent—following the Hamilton convention—a quaternion $\mathbf{q} \in \mathbb{H}$ as a standard vector $q \in \mathbb{R}^4 := [q_s \;\; q_v^T]^T$ where $q_s \in \mathbb{R}$ and $q_v \in \mathbb{R}^3$ are the scalar and vector part of the quaternion, respectively.

Quaternion multiplication is defined as

$$\mathbf{q}_2 \otimes \mathbf{q}_1 = L(q_2)q_1 = R(q_1)q_2 \tag{1}$$

where $L(q)$ and $R(q)$ are orthonormal matrices defined as

$$L(q) := \begin{bmatrix} q_s & -q_v^T \\ q_v & q_s I + [q_v]^\times \end{bmatrix} \tag{2}$$

$$R(q) := \begin{bmatrix} q_s & -q_v^T \\ q_v & q_s I - [q_v]^\times \end{bmatrix}, \tag{3}$$

and $[x]^\times$ is the skew-symmetric matrix operator

$$[x]^\times := \begin{bmatrix} 0 & -x_3 & x_2 \\ x_3 & 0 & -x_1 \\ -x_2 & x_1 & 0 \end{bmatrix}. \tag{4}$$

The inverse of a unit quaternion q^{-1}, giving the opposite rotation, is equal to its conjugate q^*, which is simply the same quaternion with a negated vector part:

$$\mathbf{q}^* = Tq := \begin{bmatrix} 1 & \\ & -I_3 \end{bmatrix} q \tag{5}$$

The following identities, which are easily derived from (2)–(5), are extremely useful:

$$L(Tq) = L(q)^T = L(q)^{-1} \tag{6}$$
$$R(Tq) = R(q)^T = R(q)^{-1}. \tag{7}$$

We will sometimes find it helpful to create a quaternion with zero scalar part from a vector $r \in \mathbb{R}^3$. We denote this operation as,

$$\hat{r} = Hr \equiv \begin{bmatrix} 0 \\ I_3 \end{bmatrix} r. \tag{8}$$

Unit quaternions rotate a vector through the operation $\hat{r}' = \mathbf{q} \otimes \hat{r} \otimes \mathbf{q}^*$. This can be equivalently expressed using matrix multiplication as

$$r' = H^T L(q) R(q)^T H r = A(q) r, \tag{9}$$

where $A(q)$ is the rotation matrix in terms of the elements of the quaternion [20].

B. Rigid Body Dynamics

In the current work we will restrict our focus to rigid bodies moving freely in 3D space. That is, we consider systems with dynamics of the following form:

$$x = \begin{bmatrix} r \\ R \\ v \\ \omega \end{bmatrix}, \quad \dot{x} = \begin{bmatrix} v \\ \texttt{kinematics}(R, \omega) \\ \frac{1}{m} F_G(x, u) \\ J^{-1}(\tau_L(x, u) - \omega \times J\omega) \end{bmatrix} \tag{10}$$

where x and u are the state and control vectors, $r \in \mathbb{R}^3$ is the position, $R \in SO(3)$ is the attitude, $v \in \mathbb{R}^3$ is the linear velocity, and $\omega \in \mathbb{R}^3$ is the angular velocity. $m \in \mathbb{R}$ is the mass, $J \in \mathbb{R}^{3\times3}$ is the inertia matrix, $F_G(x, u) \in \mathbb{R}^3$ are the forces in the global frame, $\tau_L(x, u)$ are the moments in the local (body) frame, and `kinematics` are the kinematics for the chosen attitude representation. The kinematics for unit quaternions are

$$\dot{q} = \frac{1}{2} \mathbf{q} \otimes \hat{\omega} = \frac{1}{2} L(q) H \omega. \tag{11}$$

III. Quaternion Differential Calculus

We now present a simple but powerful method for taking derivatives of functions involving quaternions based on the notation and linear algebraic operations outlined in Sec. II-A.

Derivatives consider the effect an infinitesimal perturbation to the input has on an infinitesimal perturbation to the output. For vector spaces, the composition of the perturbation with the nominal value is simple addition and the infinitesimal perturbation lives in the same space as the original vector. For unit quaternions, however, neither of these are true; instead, they compose according to (1), and infinitesimal unit quaternions are (to first order) confined to a 3-dimensional plane tangent to $\mathbb{S}^3$ (see Fig. 1).

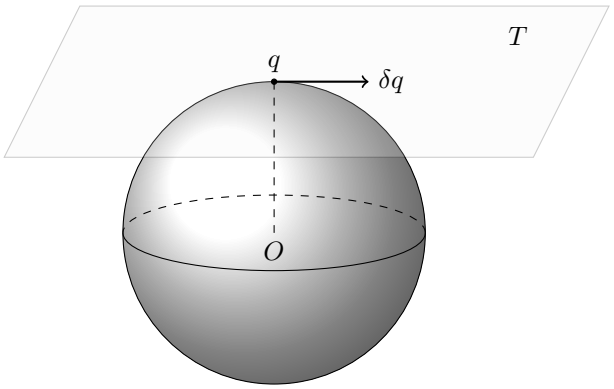

Fig. 1. When linearizing about a point q on an sphere $\mathbb{S}^{n-1}$ in n-dimensional space, the tangent space T is a plane living in $\mathbb{R}^{n-1}$, illustrated here with $n = 3$. Therefore, when linearizing about a unit quaternion $q \in \mathbb{S}^3$, the space of differential rotations lives in $\mathbb{R}^3$.

The fact that differential unit quaternions are three-dimensional should make intuitive sense: Rotations are inherently three-dimensional and differential rotations should live in the same space as angular velocity, i.e. $\mathbb{R}^3$.

skinny dog runs after skinny cat

There are many possible three-parameter representations for small rotations in the literature. Many authors use the exponential map [3, 32, 16, 22, 23, 7, 28], while others have used the Cayley map (also known as Rodrigues parameters) [14, 13], Modified Rodrigues Parameters (MRPs) [26], or the vector part of the quaternion [8]. We choose Rodrigues parameters [20] because they are computationally efficient and do not inherit the sign ambiguity associated with unit quaternions. The mapping between a vector of Rodrigues parameters $\phi \in \mathbb{R}^3$ and a unit quaternion q is known as the Cayley map:

$$q = \varphi(\phi) = \frac{1}{\sqrt{1 + \|\phi\|^2}} \begin{bmatrix} 1 \\ \phi \end{bmatrix}. \tag{12}$$

We will also make use of the inverse Cayley map:

$$\phi = \varphi^{-1}(q) = \frac{q_v}{q_s}. \tag{13}$$

A. Jacobian of Vector-Valued Functions

When taking derivatives with respect to quaternions, we must take into account both the composition rule and the nonlinear mapping between the space of unit quaternions and our chosen three-parameter error representation.

Let $\phi \in \mathbb{R}^3$ be a differential rotation applied to a function with quaternion inputs $y = h(q) : \mathbb{S}^3 \to \mathbb{R}^p$, such that

$$y + \delta y = h(L(q)\varphi(\phi)) \approx h(q) + \nabla h(q) \phi. \tag{14}$$

We can calculate the Jacobian $\nabla h(q) \in \mathbb{R}^{p\times3}$ by differentiating (14) with respect to ϕ, evaluated at $\phi = 0$:

$$\nabla h(q) = \frac{\partial h}{\partial q} L(q) H := \frac{\partial h}{\partial q} G(q) = \frac{\partial h}{\partial q} \begin{bmatrix} -q_v^T \\ sI_3 + [q_v]_\times \end{bmatrix} \tag{15}$$

where $G(q) \in \mathbb{R}^{4\times3}$ is the *attitude Jacobian*, which essentially becomes a "conversion factor" allowing us to apply

results from standard vector calculus to the space of unit quaternions. This form is particularly useful in practice since $\partial h/\partial q \in \mathbb{R}^{p\times 4}$ can be obtained using finite difference or automatic differentiation. As an aside, although we have used Rodrigues parameters, $G(q)$ is actually the same (up to a constant scaling factor) for any choice of three-parameter attitude representation.

B. Hessian of Scalar-Valued Functions

If the output of h is a scalar ($p = 1$), then we can find its Hessian by taking the Jacobian of (15) with respect to ϕ using the product rule, again evaluated at $\phi = 0$:

$$\nabla^2 h(q) = G(q)^T \frac{\partial^2 h}{\partial q^2} G(q) + I_3 \frac{\partial h}{\partial q} q, \tag{16}$$

where the second term comes from the second derivative of $\varphi(\phi)$. Similar to $G(q)$, this ends up being the same (up to a scaling factor) for any choice of three-parameter attitude representation.

C. Jacobian of Quaternion-Valued Functions

We now consider the case of a function that maps unit quaternions to unit quaternions, $q' = f(q) : \mathbb{S}^3 \to \mathbb{S}^3$. Here we must also consider the non-trivial effect of a differential value applied to the output, i.e.:

$$L(q')\varphi(\phi') = f(L(q)\varphi(\phi)). \tag{17}$$

Solving (17) for ϕ' we find,

$$\phi' = \varphi^{-1}\left(L(q')^T f(L(q)\varphi(\phi))\right) \approx \nabla f(q)\,\phi. \tag{18}$$

Finally, the desired Jacobian is obtained by taking the derivative of (18) with respect to ϕ:

$$\nabla f(q) = H^T L(q')^T \frac{\partial f}{\partial q} L(q) H = G(q')^T \frac{\partial f}{\partial q} G(q). \tag{19}$$

The leading $G(q')^T$ comes from the fact that as $\phi' \to 0$, $L(q')f(q) \to I_q$, where I_q is the quaternion identity. Differentiating through the inverse map, evaluated at the quaternion identity, we find that $\partial\varphi^{-1}/\partial q \to H^T$ for any three-parameter attitude representation.

IV. Multiplicative LQR

Leveraging the methods from the previous section, we derive multiplicative LQR (MLQR), a variant of LQR that correctly accounts for the group structure of rotations. For concreteness, we consider a system with rigid body dynamics, as presented in Sec. II-B, and design a controller to stabilize the system about a dynamically feasible discretized reference trajectory $\bar{x}_{0:N}, \bar{u}_{0:N}$ with N time steps. We begin by linearizing the dynamics about the reference trajectory using (19). Our linearized error dynamics become

$$\delta x_{k+1} = A_k \delta x_k + B_k \delta u_k \tag{20}$$

where

$$\begin{aligned} A_k &= E(\bar{x}_{k+1})^T \frac{\partial f}{\partial x}|_{\bar{x}_k, \bar{u}_k} E(\bar{x}_k), \\ B_k &= E(\bar{x}_{k+1})^T \frac{\partial f}{\partial u}|_{\bar{x}_k, \bar{u}_k}, \end{aligned} \tag{21}$$

and $\delta x_k \in \mathbb{R}^{12}$ and $E(x_k) \in \mathbb{R}^{12\times 13}$ are the state error and state error Jacobian:

$$\delta x_k = \begin{bmatrix} r_k - \bar{r}_k \\ \varphi^{-1}(\bar{\mathbf{q}}_k^{-1} \otimes \mathbf{q}_k) \\ v_k - \bar{v}_k \\ \omega_k - \bar{\omega}_k \end{bmatrix}, \; E(x) = \begin{bmatrix} I_3 & & & \\ & G(q) & & \\ & & I_3 & \\ & & & I_3 \end{bmatrix}. \tag{22}$$

With our linearized system, we can apply the traditional LQR Riccati recursion,

$$\begin{aligned} P_k &= W_k + A_k^T P_{k+1} A_k \\ &\quad - A_k^T P_{k+1}^T B_k (R_k + B_k^T P_{k-1} B_k)^{-1} B_k^T P_{k+1} A_k, \end{aligned} \tag{23}$$

which minimizes the quadratic cost function

$$\frac{1}{2}\sum_{k=0}^{N} \delta x_k^T Q_k \delta x_k + \delta u_k^T R_k \delta u_k \tag{24}$$

of the state *error*, where $Q_k \in \mathbb{R}^{12\times 12}$ is the weight matrix on the state *error*, and $R_k \in \mathbb{R}^m$ is the weight matrix on the controls. From this, we get our non-linear feedback policy

$$u = K_k \delta x_k + \bar{u}_k. \tag{25}$$

where δx_k is computed online using the non-linear error function (22) and the linear feedback gain K_k is calculated using the standard LQR formula:

$$K_k := -(R_k + B_k^T P_{k+1} B_k)^{-1} B_k^T P_{k+1} A_k. \tag{26}$$

The multiplicative Linear Quadratic Regulator is summarized as follows:

1) Pick $Q_k \in \mathbb{R}^{12\times 12}$ to weight the state *error*, and $R_k \in \mathbb{R}^{m\times m}$
2) Linearize the dynamics using the state error Jacobian (22)
3) Compute K_k using the standard LQR Riccati recursion
4) Online, compute the control using (22) and (25).

A. Constrained Iterative MLQR

As demonstrated above, it is very straight-forward to adapt LQR to use quaternions. We similarly use this technique to adapt nonlinear trajectory optimization algorithms. Here we present the modifications to the ALTRO solver [11], which uses iterative LQR (iLQR) combined with an augmented Lagrangian approach to handle constraints.

For each iteration of iLQR within ALTRO, we can calculate a quadratic approximation of the cost function using (15) and (16). Identical to (20), we use (19) to linearize the dynamics and use MLQR to solve for the feedback and feedforward gains.

The only other modification to the algorithm is during the iLQR "forward pass" that simulates the system with the closed-loop MLQR controller, where we use (22) to calculate the nonlinear feedback policy during the forward roll-outs. For robustness, we may also add an additional criterion to the line search that checks for singularities in the three-parameter error state which, for the Cayley map, occur at $180°$, at which point the step size should be reduced.

B. Quaternion Cost Functions

In addition to the straight-forward modifications to the LQR algorithm itself, we need to carefully consider the types of cost functions we minimize. We frequently minimize costs that penalize distance from a goal state, e.g. $\frac{1}{2}(x-x_g)^T Q(x-x_g)$; however, naïve substraction of unit quaternions is ill-defined. We propose two different cost functions that accomplish similar behavior. For sake of clarity and space, we only consider the costs on the quaternion variables: costs on the other states and the control variables remain unaffected.

1) Error Quadratic: Rather than simple subtraction, we can use a quadratic function on the three-parameter error state (22):

$$J_{\text{err}} = \frac{1}{2}\phi^T Q \phi = \frac{1}{2}\left(\varphi^{-1}(\delta q)\right)^T Q \left(\varphi^{-1}(\delta q)\right). \tag{27}$$

where $\delta q = L(q_g)^T q$, and $\phi = \varphi \delta q$. The gradient and Hessian of (27) are

$$\nabla J_{\text{err}} = \phi^T Q D(\delta q) G(\delta q) \tag{28}$$

$$\nabla^2 J_{\text{err}} = G(\delta q)^T \left(D(\delta q)^T Q D(\delta q) + \nabla D\right) G(\delta q) + I_3 (\phi^T Q D(\delta q)) \delta q \tag{29}$$

where, for the Cayley map,

$$D(q) = \frac{\partial \varphi^{-1}}{\partial q} = -\frac{1}{q_s^2}\begin{bmatrix} q_v & -\frac{1}{q_s} I_3 \end{bmatrix} \tag{30}$$

$$\nabla D = \frac{\partial}{\partial q}\left(D(q)^T Q \phi\right) = -\frac{1}{q_s^2}\begin{bmatrix} -2\frac{q_v}{q_s}^T Q\phi & \phi^T Q \\ Q\phi & 0 \end{bmatrix}. \tag{31}$$