Quaternion math

Define the unit quaternion as $\mathbf{q} \in \mathbb{R}^4 := [q_s \; q_v^T]^T$ where $q_s \in \mathbb{R}$ and $q_v \in \mathbb{R}^3$.
We'll be using the following packages for Julia implementation:

```
using Rotations
using LinearAlgebra
using Test
using StaticArrays
using ForwardDiff
const RS = Rotations
```

Quaternion multiplication

As unit quaternions can be viewed as a pose, quaternion multiplication can be written as a pose composition $\mathbf{q}_1 \oplus \mathbf{q}_2$, and a quaternion-vector multiplication can be written as a transformation $\mathbf{q}_1 \cdot \mathbf{p}_A$, both are NOT standard matrix/vector multiplication.

Julia implementation

```
"""Returns the cross product matrix """
function cross_mat(v)
    return [0 -v[3] v[2]; v[3] 0 -v[1]; -v[2] v[1] 0]
end

"""Given quaternion q returns left multiply quaternion matrix L(q)"""
function Lmat(quat)
    L = zeros(4,4)
    s = quat[1]
    v = quat[2:end]
    L[1,1] = s
    L[1,2:end] = -v'
    L[2:end,1] = v
    L[2:end, 2:end] = s*I + cross_mat(v)
    return L
end

"""Given quaternion q returns right multiply quaternion matrix Rmat(q)"""
function Rmat(quat)
    L = zeros(4,4)
    s = quat[1]
    v = quat[2:end]
    L[1,1] = s
    L[1,2:end] = -v'
    L[2:end,1] = v
    L[2:end, 2:end] = s*I - cross_mat(v)
    return L
end

# Define quaternions using Rotations.jl
```

```
q1 = RS.UnitQuaternion(RotY(pi/2))
q2 = RS.UnitQuaternion(RotY(pi/5))
# Get standard vectors representation
q1_vec = RS.params(q1)
q2_vec = RS.params(q2)
# test L(q) and R(q)
@test RS.rmult(q1) ≈ Rmat(q1_vec)
@test RS.lmult(q1) ≈ Lmat(q1_vec)
# test multiplication results
@test Lmat(q1_vec)*q2_vec ≈ RS.params(q2 * q1)
@test Rmat(q2_vec)*q1_vec ≈ RS.params(q2 * q1)
```

Verifying quaternion-vector multiplication:

```
# General vector/point A
PA = [0;0;2]
# Rotate π/2 along Y axis
PB = H'*Lmat(q1_vec)*Rmat(q1_vec)'*H*PA
@test PB ≈ q1*PA
```

Quaternion Differential Calculus

From section III in Planning with Attitude[1], define a function with quaternion inputs $y = h(q) : \mathbb{S}^3 \to \mathbb{R}^p$, such that:

$$y + \delta y = h(L(q)\phi(q)) \approx h(q) + \nabla h(q)\phi \tag{1}$$

where $\phi \in \mathbb{R}^3$ is defined in body frame, representing a angular velocity. We can calculate the jacobian of this function $\nabla h(q) \in \mathbb{R}^{p\times 3}$ by differentiating (1) wit respect to ϕ, evaluated at $\phi = 0$:

$$\nabla h(q) = \frac{\partial h}{\partial q} L(q) H := \frac{\partial h}{\partial q} G(q) \tag{2}$$

where $G(q) \in \mathbb{R}^{4\times 3}$ is the attitude Jacobian:

```
# a random quaternion
q = rand(UnitQuaternion)
q_vec = RS.params(q)
# G(q)
@test RS.∇differential(q) ≈ RS.lmult(q)*H
```

and $\frac{\partial h}{\partial q}$ is obtained by finite differences:

```
@test Rotations.∇rotate(q,v1) ≈ ForwardDiff.jacobian(q->UnitQuaternion(q,false
    )*v1, Rotations.params(q))
```

In the code above,function $h(q)$ is rotation of a vector $v1$.

Quaternion error state

The inverse Cayley map:

$$\phi = \varphi^{-1}(\mathbf{q}) = \frac{q_v}{q_s} : \mathbb{R}^4 \to \mathbb{R}^3 \tag{3}$$

can be used to calculate the error of two quaternions defined as $\delta\mathbf{q}$.

$$\delta\mathbf{q} = \varphi^{-1}(\mathbf{q}_2^{-1} \oplus \mathbf{q}_1) \tag{4}$$

Julia implimentation

```
@test RS.params(q2^(-1) * q1)[2:4] / (RS.params(q2^(-1) * q1)[1] ) ≈ RS.
    rotation_error(q1,q2, RS.CayleyMap())
```

Single rigid body with quaternion

Consider a cubic base, noted as link 0, floating in space, we can define its state vector in the following form:

$$\mathbf{x} = \begin{bmatrix} \mathbf{r} \\ \mathbf{q} \\ \mathbf{v} \\ \boldsymbol{\omega} \end{bmatrix} \in \mathbb{R}^{13} \tag{5}$$

Where $\mathbf{r}$ is the simplified form of ${}^{\mathcal{I}}\mathbf{r} \in \mathbb{R}^3$ representing COM position in Inertial reference frame(world frame). $\mathbf{q}$ is a unit quaternion representing rigid body's relative orientation to the world frame. $\mathbf{v}$ is the linear velocity in world frame, and $\boldsymbol{\omega}$ is angular velocity in body frame $\mathcal{L}_0$.
On the input side, we start by assuming full control over force ${}^{\mathcal{L}_0}\mathbf{F} \in \mathbb{R}^3$ and torque ${}^{\mathcal{L}_0}\boldsymbol{\tau} \in \mathbb{R}^3$:

$$\mathbf{u} = \begin{bmatrix} \mathbf{F} \\ \boldsymbol{\tau} \end{bmatrix} \in \mathbb{R}^{6} \tag{6}$$

And then we simply constraint a input term to zero if we do not have full control.

Continues time dynamic modeling

Linear velocity in world frame is straightforward:

$$\dot{\mathbf{r}} = \mathbf{v} \tag{7}$$

Quaternion rate $\dot{\mathbf{q}}$:

$$\dot{\mathbf{q}} = \frac{1}{2} G(\mathbf{q}) \boldsymbol{\omega} \tag{8}$$

Linear acceleration in world frame can be found by rotating input force in body frame:

$$\dot{\mathbf{v}} = \frac{1}{m} \mathbf{q} \cdot \begin{bmatrix} \mathbf{I}_3 & 0 \end{bmatrix} \mathbf{u} \tag{9}$$

Angular acceleration in body frame from Euler's equation:

$$\dot{\boldsymbol{\omega}} = \mathbf{J}^{-1} \left(\begin{bmatrix} 0 & \mathbf{I}_3 \end{bmatrix} \mathbf{u} - \boldsymbol{\omega} \times \mathbf{J} \boldsymbol{\omega} \right) \tag{10}$$

Discrete time dynamics and linearization

We need to analyze the problem in error state: given a reference $\bar{\mathbf{x}}_k$, $\bar{\mathbf{u}}_k$ for discrete-time system $f(\mathbf{x}_k, \mathbf{u}_k)$:

$$f(\mathbf{x}_k, \mathbf{u}_k) = \begin{bmatrix} \mathbf{r}_k \\ \mathbf{q}_k \\ \mathbf{v}_k \\ \boldsymbol{\omega}_k \end{bmatrix} + \begin{bmatrix} \mathbf{v}_k \\ \frac{1}{2} G(\mathbf{q}_k) \boldsymbol{\omega}_k \\ \frac{1}{m} \mathbf{q}_k \cdot \begin{bmatrix} \mathbf{I}_3 & 0 \end{bmatrix} \mathbf{u}_k \\ \mathbf{J}^{-1} \left(\begin{bmatrix} 0 & \mathbf{I}_3 \end{bmatrix} \mathbf{u}_k - \boldsymbol{\omega}_k \times \mathbf{J} \boldsymbol{\omega}_k \right) \end{bmatrix} dt \tag{11}$$

NOTE: this is just a standard Euler step, we can also use RK4 or Symplectic Methods.

$$\begin{aligned} \bar{\mathbf{x}}_{k+1} + \Delta \mathbf{x}_{k+1} &= f(\bar{\mathbf{x}}_k + \Delta \mathbf{x}_k, \bar{\mathbf{u}}_k + \Delta \mathbf{u}_k) \\ &\approx f(\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k) + \left. \frac{\partial f}{\partial \mathbf{x}} \right|_{\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k} \Delta \mathbf{x}_k + \left. \frac{\partial f}{\partial \mathbf{u}} \right|_{\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k} \Delta \mathbf{u}_k \end{aligned} \tag{12}$$

Here, $\left. \frac{\partial f}{\partial \mathbf{x}} \right|_{\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k} \in \mathbb{R}^{13 \times 13}$, $\left. \frac{\partial f}{\partial \mathbf{u}} \right|_{\bar{\mathbf{x}}_k, \bar{\mathbf{u}}_k} \in \mathbb{R}^{13 \times 6}$.

We define a new error state vector $\delta\mathbf{x} \in \mathbb{R}^{12}$:

$$\delta\mathbf{x}_k = \begin{bmatrix} \mathbf{r}_k - \bar{\mathbf{r}}_k \\ \varphi^{-1}(\bar{\mathbf{q}}_k^{-1} \oplus \mathbf{q}_k) \\ \mathbf{v}_k - \bar{\mathbf{v}}_k \\ \boldsymbol{\omega}_k - \bar{\boldsymbol{\omega}}_k \end{bmatrix} \tag{13}$$

Thus we can get:

$$\begin{aligned} \delta\mathbf{x}_{k+1} &= E(\bar{\mathbf{x}}_{k+1})^T \frac{\partial f}{\partial \mathbf{x}}\Big|_{\bar{\mathbf{x}}_k,\bar{\mathbf{u}}_k} E(\bar{\mathbf{x}}_k)\delta\mathbf{x}_k + E(\bar{\mathbf{x}}_{k+1})^T \frac{\partial f}{\partial \mathbf{u}}\Big|_{\bar{\mathbf{x}}_k,\bar{\mathbf{u}}_k} \delta\mathbf{u}_k \\ &= \mathbf{A}_k\delta\mathbf{x}_k + \mathbf{B}_k\delta\mathbf{u}_k \end{aligned} \tag{14}$$

Where:

$$E(\bar{\mathbf{x}}) = \begin{bmatrix} I_3 & & & \\ & G(\bar{\mathbf{q}}) & & \\ & & I_3 & \\ & & & I_3 \end{bmatrix} \in \mathbb{R}^{13\times 12} \tag{15}$$

$$\delta\mathbf{u}_k = \Delta\mathbf{u}_k \in \mathbb{R}^{6\times 1} \tag{16}$$

$$\mathbf{A}_k \in \mathbb{R}^{12\times 12} \tag{17}$$

$$\mathbf{B}_k \in \mathbb{R}^{12\times 6} \tag{18}$$

$$\tag{19}$$

Controlibility Analysis

[2] This is trickier than I though with quaternions in the state, the regular rank method doesn't seems to work.

Formulating a SQP

Appendix

Definitions and Notations

1. Operator $\cdot$: transforms the vector.
2. Operator $\oplus$: composition of relative poses.
3. $\mathcal{I}$: Inertial reference frame.
4. $\mathcal{L}_i$: Link frame.
5. $\mathcal{J}_i$: Joint frame.
6. $\hat{i}$: axis along the link.
7. $\hat{j}$: defined by right hand triad.
8. $\hat{k}$: axis along the revolute joint.
9. ${}^{\mathcal{I}}T_{\mathcal{L}_i}$: A homogeneous transformation matrix from $\mathcal{L}_i$ frame to $\mathcal{I}$ frame.
10. ω_i: angular velocity of ith link, in body frame.
11. $\dot{r}_i$: linear velocity of ith link, in world frame.

Euler's Equations

References

[1] Jackson B E, Tracy K, Manchester Z. Planning with Attitude[J]. IEEE Robotics and Automation Letters, 2021.

[2] Jiang B X, Liu Y, Kou K I, et al. Controllability and Observability of Linear Quaternion-valued Systems[J]. Acta Mathematica Sinica, English Series, 2020, 36(11): 1299-1314.