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:

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

Quaternion multiplication

Verifying quaternion multiplication $\mathbf{q}_1 \oplus \mathbf{q}_2$ using Rotation.jl:

```
"""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 transformation of a vector/point $\mathbf{q}_1 \cdot \mathbf{p}_A$ with quaternion

```
# 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 $v1$.

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:

$$\mathbf{x} = \begin{bmatrix} \mathbf{r} \\ \mathbf{q} \\ \mathbf{v} \\ \boldsymbol{\omega} \end{bmatrix} \in \mathbb{R}^{13} \tag{3}$$

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. $\mathbf{v}$ and $\boldsymbol{\omega}$ are the linear velocity and angular velocity in body frame $\mathcal{L}_0$ respectively.
On the input side, we start by assuming full control over force ${}^{\mathcal{L}_0}\mathbf{F} \in \mathbb{R}^3$ and torque ${}^{\mathcal{L}_0}\boldsymbol{\tau} \in \mathbb{R}^3$:

$$\mathbf{u} = \begin{bmatrix} \mathbf{F} \\ \boldsymbol{\tau} \end{bmatrix} \in \mathbb{R}^6 \tag{4}$$

And then we adjust actuation by adjusting the $\mathbf{B} \in \mathbb{R}^{6\times 6}$ matrix.

Dynamic modeling

Linear velocity in world frame can be calculated by rotating body velocity vector $\mathbf{v}$:

$$\dot{\mathbf{r}} = \mathbf{q} \cdot \mathbf{v} \tag{5}$$

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

$$\dot{\mathbf{q}} = \frac{1}{2} G(\mathbf{q})\boldsymbol{\omega} \tag{6}$$

Linear acceleration in body frame:

$$\dot{\mathbf{v}} = \frac{1}{m} \begin{bmatrix} \mathbf{I}_3 & 0 \end{bmatrix} \mathbf{Bu} \tag{7}$$

Angular acceleration in body frame:

$$\dot{\boldsymbol{\omega}} = \mathbf{J}^{-1} \begin{bmatrix} 0 & \mathbf{I}_3 \end{bmatrix} \mathbf{Bu} \tag{8}$$

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.
11. $\dot{r}_i$: linear velocity of ith link.

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.