Back to interpolation

- if we want to interpolate between $\vec{w}_t^t R_0$ and $\vec{w}_t^t R_1$
- recall the desired interpolation frames in matrix form is something like $\vec{w}_t^t (R_1 R_0^{-1})^\alpha R_0$
  - where the power operator of a rotation keeps its axis but scales its angle.
- suppose that $R_0$ and $R_1$ are modeled as $q_0$ and $q_1$.
- so we want to calculate something like $(q_1 q_0^{-1})^\alpha q_0$
- so we need a power operation on quaternions
- to do power, we should first extract the axis and angle of a quaternion.

Extract

- let's see how to factor a unit quaternion into axis angle form.

\[
\begin{bmatrix}
  w \\
  x \\
  y \\
  z
\end{bmatrix}
\]

- set $\beta$ to be the norm of $[x, y, z]^t$.
- set

\[
\hat{k} = \frac{1}{\beta} \begin{bmatrix}
  x \\
  y \\
  z
\end{bmatrix}
\]

  - must be unit!
  - this gives us a positive $\beta$ and a $\hat{k}$ so that

\[
\begin{bmatrix}
  w \\
  \beta \hat{k}
\end{bmatrix} = \begin{bmatrix}
  w \\
  x \\
  y \\
  z
\end{bmatrix}
\]

  - with $w^2 + \beta^2 = 1$ (on unit circle, $w$ is horizontal axis).
  - $\beta^2$ is the sum of squares of $[x, y, z]$.

angle

- Extract $\theta$ using the $\text{atan2}$ function in C++.
- $\text{atan}(\beta, w)$ returns the unique $\phi \in (-\pi..\pi]$ such that $\sin(\phi) = \beta$ and $\cos(\phi) = w$.
- because our $\beta \geq 0$, we will have $\phi \in [0..\pi]$
  - this also means we could have gotten away with $\text{acos}$.
- this gives us $\phi$ and $\hat{k}$ so that

\[
\begin{bmatrix}
  \cos(\phi) \\
  \sin(\phi) \hat{k}
\end{bmatrix} = \begin{bmatrix}
  w \\
  x \\
  y \\
  z
\end{bmatrix}
\]

angle II

- so we get a unique value $\theta/2 \in [0..\pi]$, and thus a unique $\theta \in [0..2\pi]$. 
• this gives us \( \theta \) and \( \hat{k} \) such that

\[
\begin{bmatrix}
\cos\left(\frac{\theta}{2}\right) \\
\sin\left(\frac{\theta}{2}\right)\hat{k}
\end{bmatrix} = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}
\]

• so we are done with extraction.

**power**

• define

\[
\begin{bmatrix}
\cos\left(\frac{\alpha \theta}{2}\right) \\
\sin\left(\frac{\alpha \theta}{2}\right)\hat{k}
\end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\alpha \theta}{2}\right) \\ \sin\left(\frac{\alpha \theta}{2}\right)\hat{k} \end{bmatrix}
\]

• where \( \theta \) and \( \hat{k} \) were extracted uniquely, as above.
  – where \( \theta \in [0..2\pi] \)
• As \( \alpha \) goes from 0 to 1, we get a series of rotations with angles going between 0 and \( \theta \).

**short quaternion**

• given a quaternion on which we want to power: \( \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ \sin\left(\frac{\theta}{2}\right)\hat{k} \end{bmatrix} \)

• suppose \( \cos\left(\frac{\theta}{2}\right) \geq 0 \)
  – and recall \( \theta \geq 0 \).
• this means \( \theta/2 \in [0..\pi/2] \)
  – and thus \( \theta \in [0..\pi] \).
• so when we interpolate, we will get a sequence that spans at most than 180. good.

**long quaternion**

• but suppose \( \cos\left(\frac{\theta}{2}\right) < 0 \),
• this means \( \theta/2 \in (\pi/2...\pi] \)
  – and thus \( \theta \in (\pi...2\pi] \).
  – so \( \alpha \theta \) would go more than 180 degrees which we are not going to want during interpolation
• in this case suppose we can simply negate the quaternion, giving us the previous case, which we argued is a short quaternion.
  – (fwiw, if we were to refactor this negated quaternion, we would extract the negated axis and the positive angle \( 2\pi - \theta \).)
• so when we interpolate, before calling the power operator, we should first check the sign of the first coordinate, and conditionally negate the quaternion.
• we call this the conditional negation operator “\( cn \)”.

**to interpolate**

• if we want to interpolate between \( \vec{w}^t R_0 \) and \( \vec{w}^t R_1 \)
• and suppose that \( R_0 \) and \( R_1 \) are modeled as \( q_0 \) and \( q_1 \).
• then we should calculate \( (cn(q_1q_0^{-1}))^\alpha q_0 \)
• this is called slerping (see book for more).
rbt Interpolation

• now lets get back to the affine setting, so we have starting and ending frames with origins.

• given two frames, \( \vec{o}_0 = \vec{w}_0 O_0 \) \( \vec{o}_1 = \vec{w}_1 O_1 \)
  
  – we will write it as matrices \( O_0 = (O_0)^T(O_0) R \) and \( O_1 = (O_1)^T(O_1) R \), but implement it using two RigTForm variables.

• here is the plan:
  
  1) linearly interpolate the two translation to get: \( T_\alpha \),
  
  2) slerp between the rotation quaternions to obtain the rotation \( R_\alpha \),
  
  3) set the interpolated RBT \( O_\alpha \) to be \( T_\alpha R_\alpha \).

  set \( \vec{o}_\alpha = \vec{w}_\alpha O_\alpha \).

behavior

• origin of the frame travels in a straight line with constant velocity,
  
  – read right to left
  
• the vector basis of the frame rotates with constant angular velocity about a fixed axis.
  
• physically natural if origin is at center of mass.

• has intrinsic description, so it is left invariant

• note: origin plays special role. if use different object frames for same geometry, we get different interpolation
  
  – this dependence is called “non right invariance” (see fig)