This pdf contains handwritten notes, with an effort to organize/make sense of the codeflow for better "understandability" of the PX4 Attitude-Controller.

File discussed:

[mc_att_control/AttitudeControl/AttitudeControl.cpp]

Apologies for the untidy handwriting

src/modules/mc_att_control/AttitudeControl

AttitudeControl.cpp → 1) Line 60 → `e_z = q.dcm();`

Find out 3rd column (z-column) of Rotation matrix representing current attitude of drone from quaternion.

→ 2) Line 61 → `e_z_d = qd.dcm_z();`

↳ attitude setpoint!

Find out 3rd column of Rotation matrix that represents desired/setpoint attitude of the drone from the setpoint-quaternion.

→ 3) Line 62 → `Quatf qd_red(e_z, e_z_d)`

↳★ Refer to src/lib/matrix/matrix/Quaternion.hpp.

Find out the quaternion that has the information of rotation from "e_z" vector to "e_z_d" vector. This represents rotation from current attitude (for e.g. @ time step 'k') to desired attitude (@ time-step 'k+n')

→ 4) Line 72 → $\underbrace{qd\_red}_{\text{FINAL (desired) attitude quaternion with respect to world frame.}} = \underbrace{(qd\_red)}_{\text{change in attitude (or rotation) from current to desired attitude}} ⊛ \underbrace{(q)}_{\text{current attitude quaternion with respect to world frame.}};$

NOTE: → 1), 2), 3), 4) ignore yaw & prioritize Roll+Pitch!

→5) Line 76 → $\underbrace{\text{q\_mix}}$ = $\underbrace{\text{qd\_red.inversed()}}$ ⊛ $\underbrace{\text{qd}}$;

This will probably have errors(?) ⇐ Then apply reverse rotation using qd-red that we found out in line 72. [Ideally, should probably give us the exact world frame orientation] ⇐ First go to the desired/set-point thrust-vector (or attitude representation)

→6) Line 79 & 80 → specifically eliminating numerical anomalies in yaw? ⇒ For e.g. any rotation about yaw axis

$$\begin{bmatrix} \cos\left(\frac{\psi}{2}\right) \\ \hline 0 \\ 0 \\ \bar{\omega}_z \sin\left(\frac{\psi}{2}\right) \end{bmatrix}$$

→7) Line 81 → $\underbrace{\text{qd}}$ = $\underbrace{\text{qd\_red}}$ ⊛ $\underbrace{\text{Quatf(\ldots, 0, 0, \ldots)}}$;

Final desired attitude ⇐ Then, apply rotation that takes you to the current 'q' and then to desired attitude. ⇐ First apply some rotation in yaw. (Probably error compensation???)

NOTE: → In 5) & 6) & 7) we almost did the same thing except we considered yaw somehow...

8) Line 84 → $qe = q.\text{inversed}() \circledast qd;$

* error-quaternion *

This inverse rotation

First rotate from world frame to the desired attitude (or thrust vector) qd. (NOTE → This qd was calculated from line 81)

They seem to have applied qe first, then q (?)

$$q \circledast \boxed{qe} = qd$$

$$\Rightarrow q^{-1} \circledast q \circledast qe = q^{-1} \circledast qd$$

$$\Rightarrow \boxed{qe = q^{-1} \circledast qd}$$

9) Line 88 → eq = (.....)

Extract vector (imaginary) part of the error quaternion

↳ which represents the axis-angle representation of the change in orientation.

↳ This axis-angle is used as "error" term to be multiplied by Proportional ~~Feron~~ Gain!

10) <u>Line 93-101</u> : "_yawspeed_setpoint" comes from the commander and this is represented in the world frame (i.e. about world-z)

The topic "_rates_sp" publishes rates expressed in Body frame.

(Read lines 93-99)

$$\rightarrow \text{rate\_setpoint} = \underbrace{[K_{p1}\ K_{p2}\ K_{p3}]\begin{bmatrix}\phi_1\\ \phi_2\\ \phi_3\end{bmatrix}}_{\text{Proportional Controller applied to error-term (axis angle extracted from error-quaternion)}} + \{\underbrace{q.\text{inversed}().\text{dcm\_z}()}_{\text{extract z-column from (Rotation Matrix)}^T\text{ i.e. extract the world-z axis expressed as seen from the Drone-Body Frame.}}\} * \{\text{\_yawspeed\_setpoint}\}$$