About

This is a text version of the [NPFG Diagram](), that analyzes the PX4's NPFG library implementation, as a pre-step to completing the diagram visually.

PX4 NPFG Library: https://github.com/PX4/PX4-Autopilot/tree/main/src/lib/npfg

Assumptions

Constants

- **MIN_RADIUS** = 0.5 m: Minimum radius limit for 'navigateLoiter' logic
- **Vg_cutoff** = 1.0 m/s: Track error boundary cutoff ground velocity, hard-coded in `trackErrorBound`.
-

Variables

Function Breakdowns

navigateHeading

Input: Heading reference, Vehicle Ground Velocity, Wind Velocity (2D)
Output: Calls 'guideToPath' and 'updateRollSetpoint'

Tries to maintain the heading to match the heading reference by 'tricking' the NPFG to thinking that it's flying in a wind-less environment, with an air-mass relative speed in global frame, which would then try to match vehicle's air-mass-relative velocity to the direction on heading reference.

Note: This function isn't used anywhere in PX4.

navigatePathTangent

Input: Vehicle position (2D), Position Setpoint (2D), Tangent vector (2D), Ground velocity, Wind velocity (2D), Curvature of the path
Output: Calls 'guideToPath' and 'updateRollSetpoint'

Calculates the signed track error base on the position setpoint, and passes it on to ‘guideToPath’ like a Proxy.

navigateLoiter

Input: Loiter center (2D), Vehicle Position (2D), Radius, Loiter direction, Ground velocity, Wind velocity (2D)
Output: Calls ‘guideToPath’ and ‘updateRollSetpoint’

Note: Loiter center & Vehicle position unit definitions need to be set as ‘local NED coordinate, in meters’. Also, dist_to_center < 0.1 m probably is too conservative? GPS accuracy alone would probably interfere with this comparison.

navigateWaypoints

Input: Waypoint A (2D), Waypoint B (2D), Vehicle position (2D), Ground velocity, Wind velocity (2D)
Output: Calls either ‘guidetoPoint’, ‘guidetoPath’, and calls ‘updateRollSetpoint’

- If waypoint A & B are on top of each other, fly directly to it
- If we haven’t reached the waypoint A (judged based on bearing formed by A-B vector), guide to the extension line of A-B ‘before’ the waypoint A (‘guidetoPath’)
    - If the bearing setpoint set in the ‘guidetoPath’ is less diverging from the path (A-B vector), directly navigate to Waypoint A instead. Code Here.
- If we are between A & B, track the A-B path (‘guidetoPath’)

guidetoPoint

🔥 guidetoPath

Input: Ground velocity, Wind velocity (2D), Unit path tangent (2D), Signed track error, Path curvature (>0)
Output: Lateral acceleration setpoint, incorporating feed-forward (curvature compensation) term

This is the main NPFG logic, where a given path and vehicle states, which lateral acceleration setpoint is needed to guide it to that path.

🦝 adaptPeriod

Note: Adapts controller period

periodUpperBound

periodLowerBound

Input: Required Turn Rate on Path (with zero wind), Wind factor, Bearing feasibility on track (towards unit-tangent-vector)
Output: Theoretical lower bound for the controller's period

- If curvature is 0 (turn rate = 0) or damping < 0.5, return (**PI * RollTimeConst / damping**)
- Else, ramp-in the (**4 * PI * RollTimeConst * damping**), proportional to bearing feasibility on track, starting from (**PI * RollTimeConst / damping**).

trackErrorBound

Same as Eq.12 in the paper.

windFactor

Input: Vehicle Air Speed & Wind Speed
Output: Calculated wind factor

Grows from 0 to 2.0 (when Wind speed equals Air speed), then stays constant at 2.0

bearingFeasibilitiy

Input: Wind vector (2D), Bearing setpoint unit vector (2D), Airspeed
Output: Bearing feasibility

Implemented in CoLab Script: HERE.
Note: Actual implementation's input is cross & dot product of bearing and wind velocity

updateRollSetpoint

Input: Lateral acceleration setpoint, Roll limit, Roll slew rate limit
Output: Saturated roll setpoint with slew rate limit applied