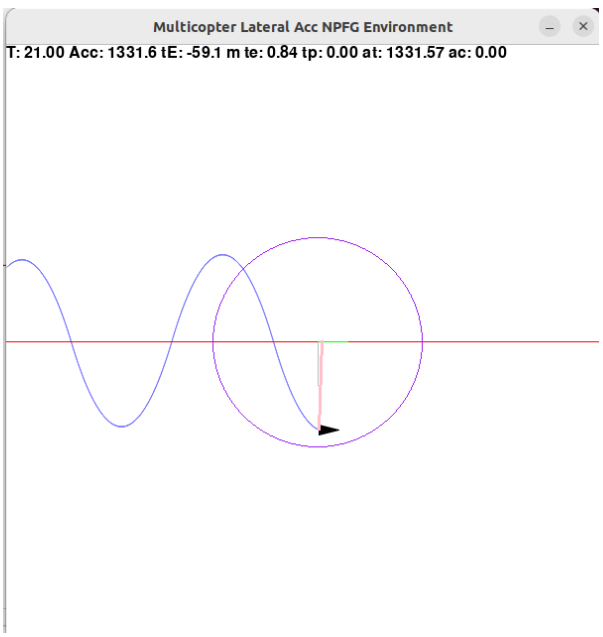
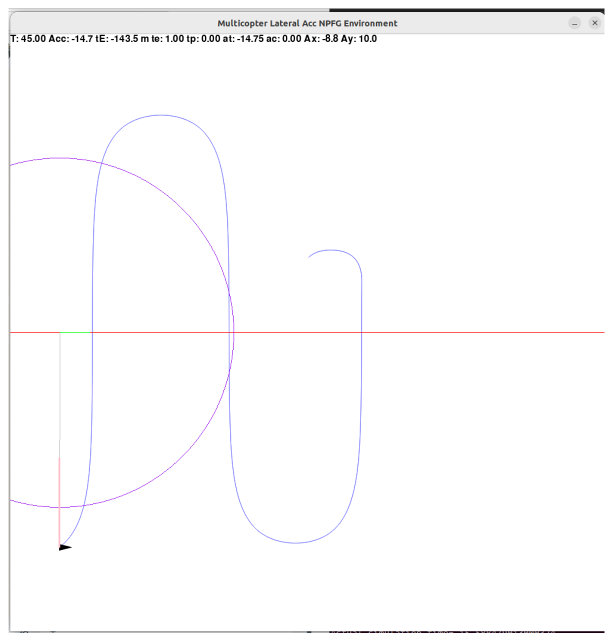
About

This document describes how the Multicopter NPFG control is implemented in the OpenAI Gym simulation environment, as requested by TJ.

- The implementation for this environment is [here](#).
- ~~Commit that tried to address the oscillation/control issue by applying acceleration command in the velocity frame is [this](#).~~

Context

Here are the simulated results from the current implementation.

Multicopter NPFG, airspeed ref = 0.1 Acceleration command applied in body frame	Multicopter NPFG, airspeed ref = 0.1 Acceleration command applied in velocity frame
 	

Multicopter NPFG Equations

$Global\ Frame\ of\ reference:\ X_g\ (East),\ Y_g\ (North),\ Z_g\ (Up)$

$Body\ Frame\ of\ reference:\ X_b\ (Front),\ Y_b\ (Left),\ Z_b\ (Up)$

NOTE: This is a different frame of reference from aerospace academia (usually Front-Right-Down & North-East-Down convention)!

$We\ define\ a\ discrete\ time\ interval\ between\ simulation\ steps:\ dt\ =\ 0.01$

$We\ define\ a\ vehicle's\ minimum\ acceleration\ limit\ to:\ a_{min} = [a_{longitudinal-min},\ a_{lateral-min}]$

$We\ define\ a\ vehicle's\ maximum\ acceleration\ limit\ to:\ a_{max} = [a_{longitudinal-max},\ a_{lateral-max}]$

Environment State

$Position\ of\ the\ Multicopter\ (global\ frame):\ p^g\ =\ [p_x,\ p_y]$

$Velocity\ of\ the\ Multicopter\ (body\ frame):\ v^b\ =\ [v_{longitudinal},\ v_{lateral}]$

$Heading\ of\ the\ Multicopter\ (global\ frame):\ \lambda^g$

$Path\ position\ setpoint\ (global\ frame):\ P^g\ =\ [P_x,\ P_y]$

$Path\ heading\ setpoint\ (global\ frame):\ \xi^g$

$Curvature\ of\ the\ path\ setpoint\ (global\ frame):\ \mu^g$

Action Space (output of NPFG)

$Acceleration\ setpoint:\ u\ =\ [u_{longitudinal},\ u_{lateral}]$

Auxiliary variable/states

$Air\ Velocity\ reference\ vector\ (indirect\ output\ of\ NPFG):\ v_{ref}^{npfg} = [v_x,\ v_y]$

$Thrust\ setpoint\ (contains\ attitude\ information\ \&\ collective\ thrust)\ t_{setpoint} = [t_x, t_y, t_z]$

Multicopter dynamics

🛸 Original (one that had sinusoidal oscillation)

$Vehicle\ velocity\ in\ global\ frame:\ v^g\ =\ H[\xi^g]\cdot v^b.\ Where\ H[\theta]\ =\ \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix},\ rotation\ matrix.$

$Note,\ longitudinal\ acc\ is\ set\ to\ 0:\ u_{npfg}\ =\ [0.0,\ NPFG.navigatePathTangent(p^g,\ P^g,\ \xi^g,\ v^g,\ \mu^g)]$

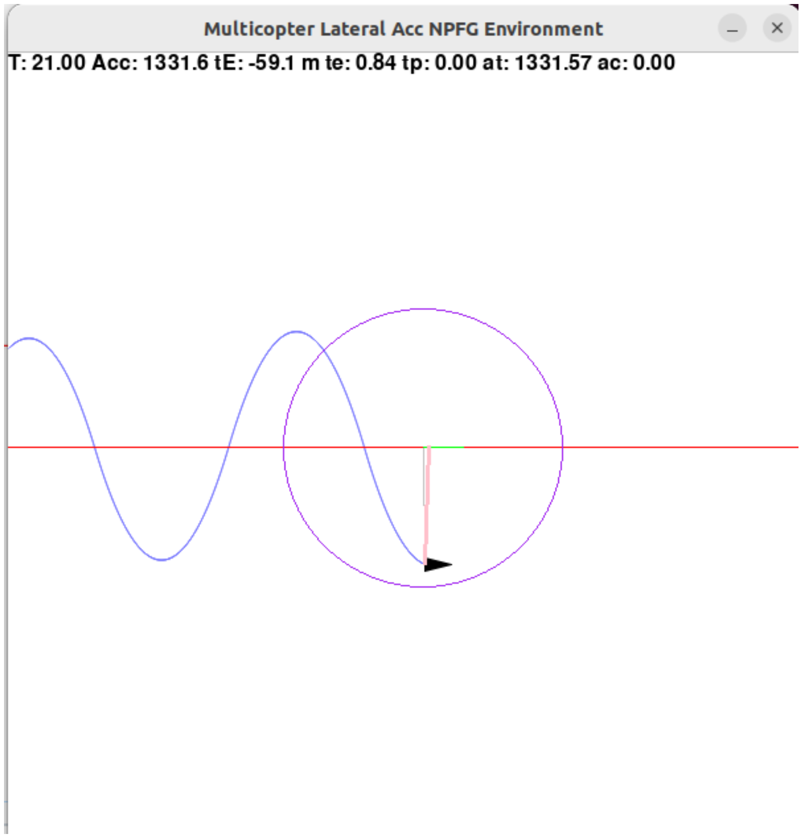
$p^g\ =\ p^g + v^g * dt$

$Constrain\ the\ control:\ u_{npfg} = constrain(u_{npfg},\ a_{min},\ a_{max})$

$v^b{}_{lateral}\ =\ v^b{}_{lateral} + u_{npfg}[1]\ *\ dt$

$v^b{}_{longitudinal}\ =\ v^b{}_{longitudinal} + u_{npfg}[0]\ *\ dt$

Simulated result for path-following (follow a path heading in X-axis direction):



I noticed the flaw in the logic because v^b and v^g has a rotation of λ^g $(vehicle\ heading)$ in between, and applying the control output from NPFG directly in the body frame (integrating v^b) would result in different vehicle movement depending on the vehicle's heading.

Modified (one that draws slower rectangular oscillation)

$Vehicle\ velocity\ in\ global\ frame$: $v^g = H[\xi^g] \cdot v^b$. $Where\ H[\theta] = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix}$, $rotation\ matrix.$

$Note,\ longitudinal\ acc\ is\ set\ to\ 0$: $u_{npfg} = [0.0,\ NPFG.navigatePathTangent(p^g,\ P^g,\ \xi^g,\ v^g,\ \mu^g)]$

$We\ deduce\ the\ ground\ velocity\ bearing$: $\chi^g = atan2(v^g[1],\ v^g[0])$

$p^g = p^g + v^g * dt$

$Interpret\ the\ longitudinal\ axis\ of\ the\ control\ in\ the\ direction\ of\ ground\ velocity$

$Acceleration\ control\ in\ global\ frame$: $u^g = H[\chi^g] \cdot u_{npfg}$

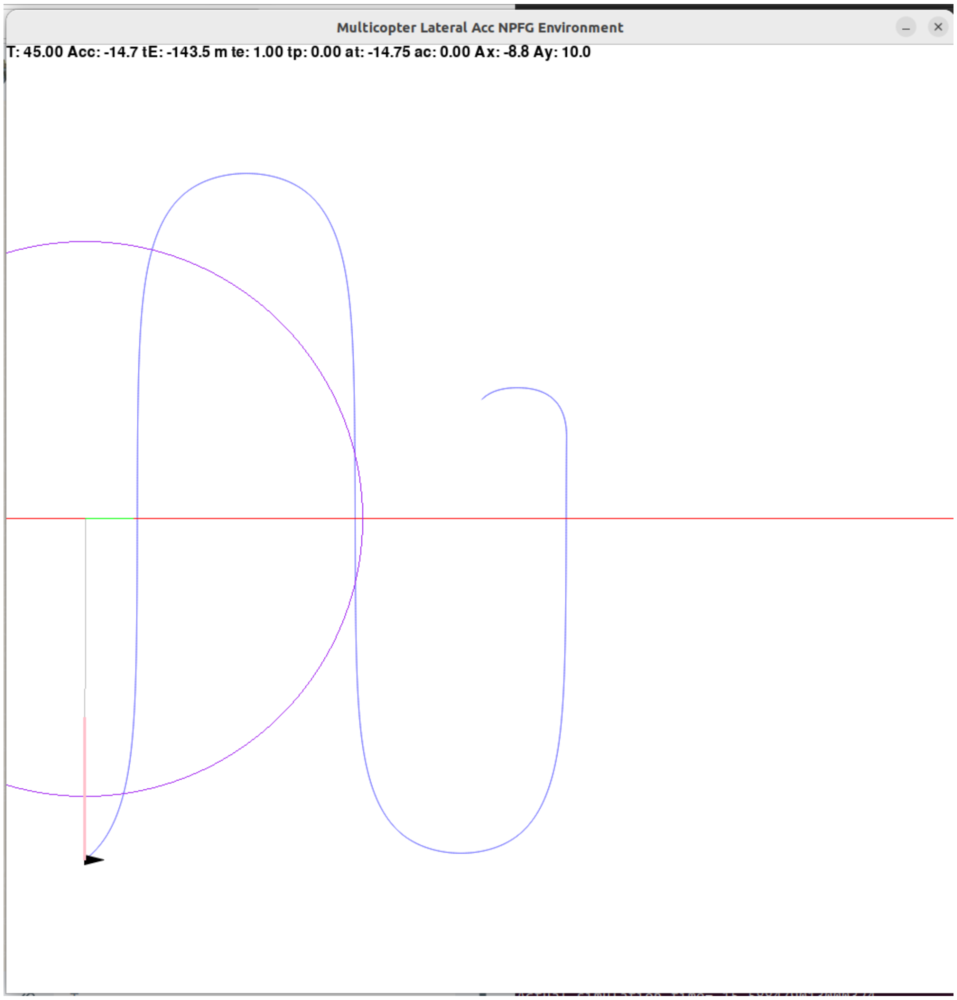
$Acceleration\ control\ in\ body\ frame$: $u^b = H[-\lambda^g] \cdot u^g$

$Constrain\ the\ control$: $u^b = constrain(u_{npfg},\ a_{min},\ a_{max})$

$v^b_{lateral} = v^b_{lateral} + u^b[1] * dt$

$v^b_{longitudinal} = v^b_{longitudinal} + u^b[0] * dt$

Simulated result for path-following (follow a path heading in X-axis direction):



🎉 TJ Recommended: Remove the Unicyclic commands

For fixed-wing, the unicyclic control (applying P-gain multiplied airspeed cross product error as lateral acceleration) makes sense, as it always flies in a circle (airmass relative), like a unicycle.

But for Multicopter, this acceleration command *can be bypassed. And the velocity reference vector can be directly fed into a speed control, as it has the control authority for it.

And we can still feed-forward the acceleration command regarding the curvature compensation.

$Vehicle\ velocity\ in\ global\ frame:\ v^{g} = H[\xi^{g}] \cdot v^{b}.\ Where\ H[\theta] = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix},\ rotation\ matrix.$

$Air\ velocity\ reference\ vector\ (global):\ v^{npfg}_{ref} = NPFG.navigatePathTangentGetRefVel(p^{g},\ P^{g},\ \xi^{g},\ v^{g},\ \mu^{g})$

Set velocity setpoint: $v_{setpoint} = v^{npfg}_{ref}$

Set acceleration feed-forward for path curvature: $a_{ff} = NPFG.getAccelFFcurvature()$

Apply velocity control & get thrust setpoint: $t_{setpoint} = VelocityControl(v^{g},\ v_{setpoint},\ a_{ff})$

Apply some vehicle dynamics? On attitude changes: **TODO**

$$v^{b}_{lateral} = v^{b}_{lateral} + t_{setpoint}[1] * dt$$

$$v^{b}_{longitudinal} = v^{b}_{longitudinal} + t_{setpoint}[0] * dt$$