pyHopper Dynamics

Description: *2D Thrust-Vectored Rocket Hopper, with the engine thrust offset from vehicle CG. This allows for 3 DOF in positional controllability with only 2 DOF in control (thrust and angle).*

The equations of motion for the system are simply derived from Newton's laws:

$$\begin{aligned} f_4(z,u) &= \ddot{x} = -\frac{F_t \sin(\theta+\phi)}{m} \\ f_5(z,u) &= \ddot{y} = \frac{F_t \cos(\theta+\phi) - mg}{m} \\ f_6(z,u) &= \ddot{\theta} = -\frac{F_t \, d \sin(\phi)}{J}, \end{aligned}$$

where

m = mass of hopper [kg]
J = rotational inertia of hopper [kg-m^2]
F_t = engine thrust [N]
θ = hopper heading angle, off vertical [rad]
ϕ = thrust vector angle, off hopper longitudinal centerline [rad]
g = gravitational acceleration [m/s^2]
d = distance from thrust vector to hopper center of gravity (i.e. thrust moment arm) [m].

A state-space formulation is more amenable to control design; but, strong non-linearities in the equations of motion don't allow for a simple determination of the A/B/C/D matrices.

However, because the hopper is expected to remain upright during initial trajectory tracking (e.g. point-to-point movement), linearizing about an upright, zero-velocity "hovering" configuration is reasonable and gives us a usable state space formulation.

Viewing the control efforts as deltas off equilibrium (trim), the following subset of the state-space defines equilibrium configurations where linearization is acceptable:

$$\begin{aligned} x_{eq} &\subset \mathbb{R} \\ y_{eq} &\subset \mathbb{R}^+ \\ \theta_{eq} &= 0 \\ \dot{x}_{eq} &= 0 \\ \dot{y}_{eq} &= 0 \\ \dot{\theta}_{eq} &= 0 \\ F_{t,eq} &= mg \\ \phi_{eq} &= 0 \end{aligned}$$

Then, the gradients of the Newtonian dynamics are calculated in this equilibrium subspace to give a linearized state-space representation:

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ \frac{\partial f_4}{\partial x} & \frac{\partial f_4}{\partial y} & \frac{\partial f_4}{\partial \theta} & \frac{\partial f_4}{\partial \dot{x}} & \frac{\partial f_4}{\partial \dot{y}} & \frac{\partial f_4}{\partial \dot{\theta}} \\ \frac{\partial f_5}{\partial x} & \frac{\partial f_5}{\partial y} & \frac{\partial f_5}{\partial \theta} & \frac{\partial f_5}{\partial \dot{x}} & \frac{\partial f_5}{\partial \dot{y}} & \frac{\partial f_5}{\partial \dot{\theta}} \\ \frac{\partial f_6}{\partial x} & \frac{\partial f_6}{\partial y} & \frac{\partial f_6}{\partial \theta} & \frac{\partial f_6}{\partial \dot{x}} & \frac{\partial f_6}{\partial \dot{y}} & \frac{\partial f_6}{\partial \dot{\theta}} \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{\partial f_4}{\partial F_t} & \frac{\partial f_4}{\partial \phi} \\ \frac{\partial f_5}{\partial F_t} & \frac{\partial f_5}{\partial \phi} \\ \frac{\partial f_6}{\partial F_t} & \frac{\partial f_6}{\partial \phi} \end{pmatrix},$$

where

$$z = \begin{pmatrix} x \\ y \\ \theta \\ \dot{x} \\ \dot{y} \\ \dot{\theta} \end{pmatrix}$$

$$u = \begin{pmatrix} \delta F_t \\ \delta \phi \end{pmatrix}$$

and

$$\dot{z} = Az + Bu.$$

Evaluating the gradients gives the A and B matrices explicitly:

$$A = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & -g & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$B = \begin{pmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & -g \\ 1 & 0 \\ 0 & -\frac{mgd}{J} \end{pmatrix}.$$

This formulation is used in the LQR and other optimal control design. Simulation is written in Python with help from the Control Systems Library (https://pypi.org/project/control/).