Review in project development #4

26th November 2022

Soham Kulkarni

PPO Implementation

- Observations for our agent: [cmdFullState(pos, vel, acc, yaw, omega]
- **pos** (*array-like of float[3]*) – Position. Meters.
- **vel** (*array-like of float[3]*) – Velocity. Meters / second.
- **acc** (*array-like of float[3]*) – Acceleration. Meters / second^2.
- **yaw** (*float*) – Yaw angle. Radians.
- **omega** (*array-like of float[3]*) – Angular velocity in body frame. Radians / sec.

In use: **cmdPosition(*pos*, *yaw=0*) (high level planner + onboard controller) , cmdStop()**

- **Full state vector not in usage.**

PPO Implementation

- Action Space:

As discussed last time:-

Let f_i, i ε [1,4] be the thrust of each motor.

❖ f_i =1 (full power)

❖ f_i = 0 (motor off)

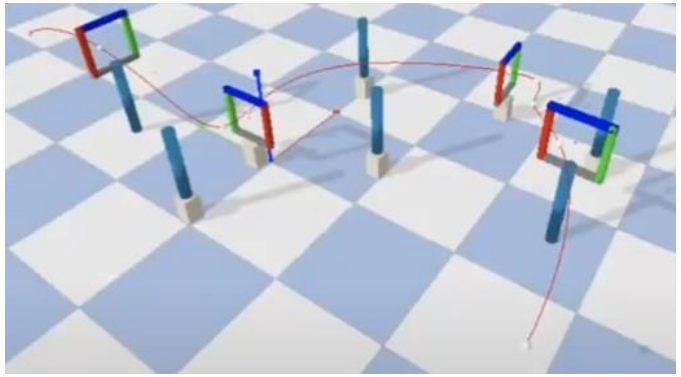
Action space is [0, 0.2, 0.4, 0,6, 0.8, 1]^4 : *Discretised*

As of now, there are *no added inertial disturbances.*

PPO Implementation

- Reward function:

$$R_t = \max\left(0, 1 - \|\boldsymbol{x} - \boldsymbol{x}_{\text{goal}}\|\right) - C_\theta \|\boldsymbol{\theta}\| - C_\omega \|\boldsymbol{\omega}\|$$



- *The first term rewards the agent when the drone is close to the target.*
- *Other terms are penalizing for spinning at turns, etc.*
- *The constants are kept very small.*
- This reward function needs a gate-pass term because it has learned to dodge the gates in the environment.
- We might need velocity control and other angular control terms if gates are introduced.

PPO Implementation

Training:

- Will plot the reward function for comparison.
- 10^6 trainings steps set: around 700k steps to dodge the lower gate railing (only thrusts controlled)
- Remaining hyperparameters are kept same: batch size, gamma, buffer size, epsilon (clipping),…

PPO Implementation

- The algorithm is trained on level 0 and works on seeding randomly.
- Yet to train with randomized inertial properties and gates.

Evaluation Scenario	Constraints	Rand. Inertial Properties	Randomized Obstacles, Gates	Rand. Between Episodes	Notes
`level0.yaml`	**Yes**	*No*	*No*	*No*	Perfect knowledge
`level1.yaml`	**Yes**	**Yes**	*No*	*No*	Adaptive
`level2.yaml`	**Yes**	**Yes**	**Yes**	*No*	Learning, re-planning
`level3.yaml`	**Yes**	**Yes**	**Yes**	**Yes**	Robustness
sim2real	**Yes**	Real-life hardware	**Yes**, injected	*No*	Sim2real transfer