Review in project development #3

19th 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()**

PPO Implementation

- Action Space:

Trying to implement binary thrust control for each motor in the initial implementation.

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

❖ f_i =1 (full power)

❖ f_i = 0 (motor off)

Therefore, Action space is [0,1]^4

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.

PPO Formulation

- Surrogate Loss:

$$L(\theta) = \mathbb{E}[r(\theta)A(s,a)], \quad \text{where } r(\theta) = \frac{\pi_{\theta_{new}}(a|s)}{\pi_{\theta_{old}}(a|s)}$$

- Clipped loss/objective function:

$$L(\theta) = \mathbb{E}[\min(r(\theta)A(s,a), \text{clip}(r(\theta), 1-\varepsilon, 1+\varepsilon)A(s,a))]$$

- Advantage: R(t) – b(s_t)
- Baseline: b(s_t) := Monte-Carlo Estimate, bootstrapping (value function estimate)

PPO Formulation

- Other hyperparameters:

❖ The clip coefficient of PPO can be annealed similar to how the learning rate is annealed. However, the clip range annealing is actually used by default.

❖ The policy gradient is calculated in parallel using multiple processes.

❖ Early stopping of optimization: it starts by tracking an approximate average KL divergence between the policy before and after one update step to its network weights. In case said KL divergence exceeds a preset threshold, the updates to the policy weights are preemptively stopped. (to tune the number of update epochs)

- Checking ratios, policy and value loss, KL-updates, seeding.
- Trying out the implementation in PyTorch + Tensorflow.
- Running gradient descent: SGD, RMS Prop, Adam Opt.

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()**