Evaluation of a meta-reinforcement learning controller for Fixed Wing UAVs

Final report

1st Patrick Stecher
dept. of Informatics
Technical University Munich
Munich, Germany
patrick.stecher@tum.de

2nd Moritz Schüler
dept. of Informatics
Technical University Munich
Munich, Germany
moritz.schueler@tum.de

***Abstract*—Using an off-policy meta reinforcement learning algorithm named PEARL, we aim to train an agent within the pyfly-simulator in an own gym environment to control a fixed-wing UAV to follow a pre-defined trajectory, hoping to achieve high robustness and better sample efficiency than standard SAC.**

***Index Terms*—meta RL, reinforcement learning, fixed wing UAV**

I. Introduction

As of 2020, fixed wing UAVs already have a broad spectrum of applications: forest monitoring, building inspection, archaeology or blood transfusion delivery. In most applications the high flying speed as well as high energy efficiency, thus long flight duration is valued. For rapidly changing demands or when reacting to unforeseen disturbances (weather, wind, thermals) current solutions still require either a trained human pilot, lots of training data or can only utilize a reduced flight envelope. We therefore try to achieve a solution, trained in simulation, that shows a good sample efficiency and generalizes better to unseen disturbances than current control approaches.

II. Approach

As outlined in the Milestone report, the PEARL algorithm, explained in detail in section VI, was implemented based on the stable baseline 3 Soft-Actor Critic [2]. A gym environment was contrived, specifically for the task of trajectory following with the pyfly simulator [4]. The trajectories are pre-generated with a simple trigonometric method, described in section IV. For training of the agents a curriculum was implemented, starting with simple straight flight and no disturbances. Over the course of the curriculum, wind in increasing amplitude is added to the environment to act as disturbance on the UAVs control.

III. Paper reproduction

A. Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables by Rakelly et al.

We were able to reproduce the results of [1]to a reasonably close degree not only using our own implementation but also using a different physics simulator, namely pybullet instead of mujoco. Thereby we were able to show that the PEARL algorithm works as well with Q bootstrapping.

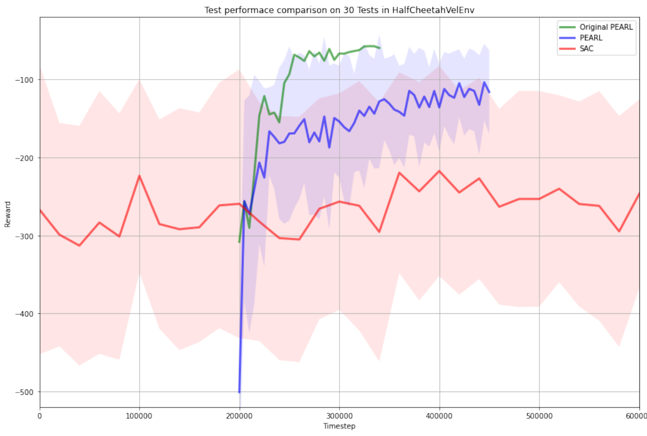

Figure 1: Both PEARL implementations and SAC performance comparison on 30 CheetahVel test tasks and corresponding standard deviation

IV. Trajectory generation

While trajectory generation was needed, it was not the main objective of our work. Therefor we set some design considerations beforehand taking this into account:

- easy and fast to implement
- easy and fast to sample trajectories
- easy to adjust difficulty of sampled trajectory
- be deterministic

With this in mind an easy trigonometric approach without a physical model was set up. The drawback of this solution might be to end up with unflyable trajectories, but this is solved by manual parameter tweaking. The basic principle of the generator is to use the current position of the aircraft and sample the next point within a ball of a specified radius R. To guarantee that it is a reachable and feasible point, the sampling is constraint by an angle α and a minimum distance *dist*. With these two parameters, the ball radius is determined by $R = \frac{dist}{\cos(\alpha)}$ as well as a cutting circle to the ball can be constructed, as can be seen in figure 2 on the left. Now by using the Rodrigues' formula (figure 2 on the right) and scaling a random point on the green area can be sampled.

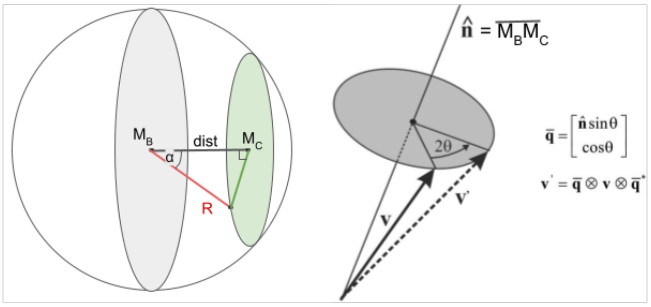
M_B
α
dist
M_C
R
$\hat{\mathbf{n}} = \overline{M_B M_C}$
2θ
$\mathbf{v}$
$\mathbf{v}'$
$\bar{\mathbf{q}} = \begin{bmatrix} \hat{\mathbf{n}}\sin\theta \\ \cos\theta \end{bmatrix}$
$\mathbf{v}' = \bar{\mathbf{q}} \otimes \mathbf{v} \otimes \bar{\mathbf{q}}^*$

Figure 2: left) illustration of trajectory sampling approach, right) Rodrigues' formula

Through tweaking of α and *dist* one can ensure the next point to be feasible as well as adjust the difficulty, i.e. $\alpha = 0$ results in a straight line or reducing the minimum distance can lead to higher curvature in turning maneuvers. Additionally to the coordinates of each point the trajectory is specified by the velocity (direction of the *dist* vector, scaled s.t. the norm lies between $5\frac{m}{s}$ and $20\frac{m}{s}$), the pitch, roll and yaw angle (as well calculated by using the current and previous point) and a randomly sampled wind vector. To guarantee deterministic generation a seed was used as well as the trajectories were only generated once and saved for later use.

V. Reward shaping

The obtained reward for the trajectory following is calculated using the Manhattan distance relative to the next waypoint, excluding pitch, roll and yaw. To ensure an even distribution of all states towards the reward the loss is normalized by the maximal operation range of each state, which is, given by the design of the trajectory, the ball radius and the maximal physical velocity of the airplane. If the principal axes would be included these could be normalized by 2π. To transform the loss into a reasonable reward which in the best case is large, we used the inverse exponential function. Therefor the reward is calculated by following formula:

$$R(t) = \frac{1}{e^{L(t)}} \; with \; L(t) = \sum^{i} \frac{|a_i - b_i|}{o_i}$$
$$a_i := goal\ state$$
$$b_i := UAV\ state$$
$$o_i := operation\ range$$

VI. PEARL Algorithm

We consider the full, detailed documentation of the PEARL algorithm a major part of this work.

The PEARL Algorithm consists of three parts; policy Network, Q Networks and a context encoder Network. The latter is a variational autoencoder to encode latent information from state-action-reward tuples into mean(s) and variance(s), which in turn parameterize normal distribution(s). From these normal distributions a vector is sampled and used by the policy network as information about the task at hand.

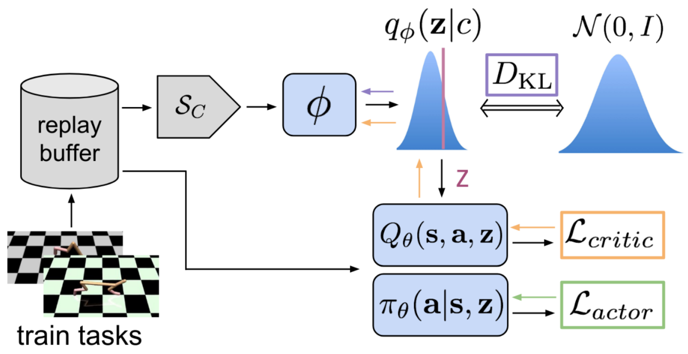

Figure 3: PEARL Algorithm illustration

A. Training

All networks are initialized with "fan in", where the weights are uniformly sampled from

$$Uniform(-bound, bound)$$

and

$$bound = \frac{1}{\sqrt{\text{width prev. layer}}}$$

. The biases are set to 0.1, except for the last layer, where weights and biases are uniformly drawn in the same manner as above, but with a bound of 3e-3. We found this initialization to be important, to avoid vanishing gradients early in training.

First in operation, the algorithm samples task experience with the untrained policy. Note that at this point all tasks are deterministic and indexed by a number. Contrary to some other RL environments a task of a specific number will always have the same initial conditions (as well as same goal(s), if applicable), and can be reset to these initial conditions.

The amount of individual replay buffers needed is twice the amount of train tasks. As the context encoder is untrained at this point, the latent vector is drawn from a zero mean normal distribution: $\mathbf{z} \sim \mathcal{N}(0, I)$, where 0 is a vector of zeros of size of the latent dimension and I is the identity matrix of same size ("prior sampling"). The state-action-reward samples gathered here are stored to both, the RL replay buffer and the encoder replay buffer.

Once finished, a small number of tasks, to explore with $\mathbf{z}$ sampled from the posterior, is drawn. The encoder replay buffers of these tasks are cleared, and then some rollouts are sampled with z drawn from the prior and saved to both buffers. Afterwards state-action-reward tupels are sampled from the encoder buffer and encoded using the policy network ϕ, and $\mathbf{z}$ is drawn. Based on this $\mathbf{z}$ additional "posterior sampling" experience is collected, but only stored into the RL replay buffer.

For the actual training loop, an amount of gradient steps to be taken is set. Each step an amount of task indices is sampled:

”meta-batch”. For the same task indices context of length L is drawn in an unordered (de-correlated) way from the encoder replay buffer. State transitions of length P are drawn from the RL replay buffer. For each step, note that therefore the context c used to infer $\phi(z|c)$ is distinct from the data used to construct the critic and actor loss.

For a detailed algorithmic description and loss formulation see Appendix. Noteworthy, the gradient for the encoder network ϕ is only back-propagated through the critic, not the actor, also see Fig 2., where the coloured arrows indicate gradient flow. Additionally the KL divergence between the predicted mean and variances is introduced as weighted loss function to act as an information bottleneck.

B. Inference

PEARL utilizes a replay buffer at inference time. First, a task will be sampled on policy, but with a latent encoding drawn from $\mathcal{N}(0, I)$, thereby not encoding any information about the task at hand. The state, action, reward pairs are stored in a unique replay buffer for each task (i.e. specific knowledge about which task to be run is required). Once a task is concluded, it shall be reset to the exact same initial conditions as before (i.e. be deterministic). Context is sampled from the previous run in an de-correlated manner. This context is encoded, the encoding in turn is used to sample a latent task embedding and used by the policy network. Online posterior inference was not tested, how well this would translate to real world applications, where running the same task under the same environmental conditions is somewhat impossible, is therefore unclear.

Algorithm 1: PEARL Meta testing

```
Require: Test task(s) $\mathcal{T}$ from $p(\mathcal{T})$, Evaluation Buffer $\mathcal{BE}^t$
for each $\mathcal{T}$ do
    $\mathbf{z} \sim \mathcal{N}(0, I)$
    while $\mathcal{T}_t$ do
        Gather data from $\pi_\theta(\mathbf{a} \mid \mathbf{s}, \mathbf{z})$ and add to $\mathcal{BE}^t$
    end
    $\mathbf{c}_j = \phi(\mathcal{BE}^t)$
    $\mathbf{z} \sim \mathcal{N}(\mathbf{c}_t)$
    reset $\mathcal{T}_t$
    while $\mathcal{T}_t$ do
        Gather data from $\pi_\theta(\mathbf{a} \mid \mathbf{s}, \mathbf{z})$ and add to $\mathcal{BE}^t$
    end
    reward = avg. reward of both runs
end
```

VII. Results

A. Performance comparison

When compared to the regular Soft-Actor-Critic, it is notable that PEARL performs better on the tested tasks and was not only much more robust in between training epochs but also showed less performance decrease when new disturbances were added in the curriculum. In Fig. 4 one can see training of PEARL (blue) and SAC (red).

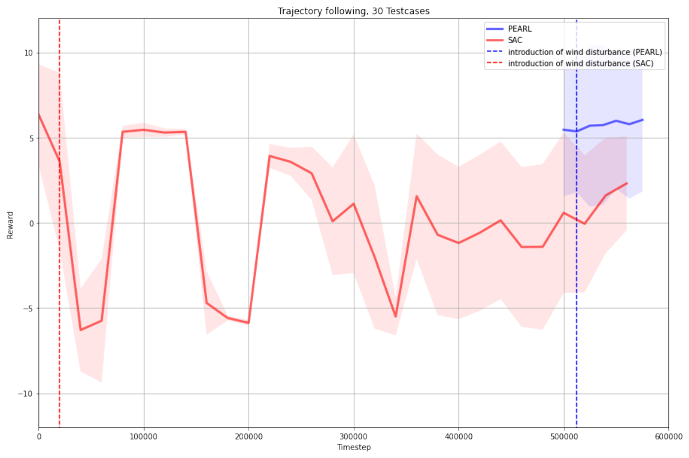

Figure 4: PEARL and SAC performance and standard deviation on 30 Trajectory following tasks over simulation steps

Both share the same x-axis with regard to total simulation steps in the environment, while each point in each series corresponds to a trained epoch. The PEARL (blue) curve only starts at around 500.000 steps, in which task trajectories are gathered. Notably, both curves start at around the same mean reward, but the PEARL curve is not only much more consistent but also has a more consistent performance standard deviation as well, hinting at higher robustness.

VIII. Discussion

The results are promising and further investigation into the solution might be beneficial. Future work should test hyper-parameter optimization, architecture search and other reward schemes or curricula. Generalization to other failures, like sensor malfunctioning, are also promising topics to conduct further experiments. However our evaluation also showed that training was quite unstable which might be the reason for the poor performance compared to [1] results. Lastly, it should be mentioned that despite promising results for the ”CheetahVel” and the trajectory tracking problem the sim2real performance remains questionable because neither the original authors of [1] nor we tested online posterior inference.
Overall the combination of meta Learning and RL is very interesting and still in its infancy, meaning that with more research highly relevant and applicable solutions for various problems could be found.

References

[1] K. Rakelly, A. Zhou, D. Quillen, C. Finn and S. Levine, “Efficient Off-Policy Meta-Reinforcement Learning via Probabilistic Context Variables,” https://arxiv.org/abs/1903.08254, 2019.
[2] A. Hill, A. Raffin, M. Ernestus et al., “Stable Baselines 3,” https://github.com/DLR-RM/stable-baselines3, 2021.
[3] E. Bøhn, “Fixed-wing aircraft gym environment,” https://github.com/eivindeb/fixed-wing-gym, 2019.
[4] E. Bøhn, “Pyfly,” https://github.com/eivindeb/pyfly, 2019.
[5] E. Bøhn, E. M. Coates, S. Moe, T. A. Johansen, “Deep Reinforcement Learning Attitude Control of Fixed-Wing UAVs Using Proximal Policy Optimization”, https://arxiv.org/abs/1911.05478

IX. Appendix

```
Algorithm 2: PEARL Meta training
1  Require:Batch of training tasks $\{\mathcal{T}_i\}_{i=1...T}$ from $p(\mathcal{T})$
     learning rates $\alpha_1, \alpha_2, \alpha_3$
2  Initialize replay buffers $\mathcal{BR}^i$ for each training task
     Initialize encoder replay buffers $\mathcal{BC}^i$ for each task
     Initialize encoder-, Q- and policy-network(s) with
     'fan-in initialization' and small biases
3  for each $\mathcal{T}_i$ do
4      for k in K do
5          $\mathbf{z} \sim \mathcal{N}(0, I)$
6          while Task do
7              Gather data from $\pi_\theta(\mathbf{a} \mid \mathbf{s}, \mathbf{z})$ and add to $\mathcal{BR}^i$
                 and $\mathcal{BC}^i$
8          end
9      end
10 end
11 while not done do
12     gather on policy experience;
13     sample J Tasks from $\mathcal{T}_i$
14     for each $\mathcal{T}_J$ do
15         clear $\mathcal{BC}^j$
16         collect experience with prior:
17         $\mathbf{z} \sim \mathcal{N}(0, I)$
18         while $\mathcal{T}_j$ do
19             Gather data from $\pi_\theta(\mathbf{a} \mid \mathbf{s}, \mathbf{z})$ and add to $\mathcal{BR}^i$
                 and $\mathcal{BC}^i$
20         end
21         infer posterior
22         $\mathbf{c}_j = \phi(\mathcal{BC}^j)$
23         $\mathbf{z} \sim \mathcal{N}(\mathbf{c}_j)$
24         collect experience with posterior:
25         while $\mathcal{T}_j$ do
26             Gather data from $\pi_\theta(\mathbf{a} \mid \mathbf{s}, \mathbf{z})$ and add to $\mathcal{BR}^i$
27         end
28     end
29     for each train step do
30         sample M tasks of $\mathcal{T}_i$
31         for each $\mathcal{T}_m$ do
32             sample RL batch $b^i \sim \mathcal{BR}^m$ of length P
                 sample context $\mathbf{c}_m \sim \mathcal{BC}^m$ of length L
33             $\mathbf{z} \sim \mathcal{N}(\mathbf{c}_m)$
34             $\mathcal{L}^m_{\text{actor}} = \mathcal{L}_{\text{actor}}(b^m, \mathbf{z})$
35             $\mathcal{L}^m_{\text{critic}} = \mathcal{L}_{\text{critic}}(b^m, \mathbf{z})$
36             $\mathcal{L}^m_{KL} = \beta D_{\text{KL}}(q(\mathbf{z} \mid \mathbf{c}^m) \| r(\mathbf{z}))$
37         end
38         $\phi \leftarrow \phi - \alpha_1 \nabla_\phi \sum_m (\mathcal{L}^m_{critic} + \mathcal{L}^m_{KL})$
39         $\theta_\pi \leftarrow \theta_\pi - \alpha_2 \nabla_\theta \sum_m \mathcal{L}^m_{actor}$
40         $\theta_Q \leftarrow \theta_Q - \alpha_3 \nabla_\theta \sum_m \mathcal{L}^m_{critic}$
41     end
42 end
```

Note that the algorithm's description shown here is purposefully more detailed than in [1].

TABLE I
Algorithm parameters

Parameter	Symbol	typical values
KL weight	β	0.1
Meta batchsize	M	16
number of train steps per epoch		500 .. 2000
number of tasks	T	50 .. 100
number of repeats per task while collecting initial samples	K	10
number of task to be sampled with posterior	J	5
number of reinforcement learning spamles per task	P	256
number of context samples	L	100
learning rates	α_i	0.0003

$$\mathcal{L}_{critic}(\phi_i, \mathcal{B}) = \underset{(s,a,r,s',d)\sim\mathcal{B}}{\mathbb{E}} \left[(Q_{\phi_i}(s,a) - y(r, s', d, \mathbf{z}))^2 \right]$$

with

$$y(r, s', d, \mathbf{z}) = r + \gamma(1-d) ... \left(\min_{j=1,2} Q_{\phi_{\text{targ}}, j}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}' \mid s', \mathbf{z}) \right)$$

and

$$\tilde{a}' \sim \pi_\theta(\cdot \mid s', \mathbf{z})$$

$$\mathcal{L}_{actor}(\theta, \phi_j, \mathcal{B}) = \max_\theta \underset{\substack{s\sim\mathcal{B} \\ \xi\sim\mathcal{N}}}{\mathbb{E}} \left[\min_{j=1,2} Q_{\phi_j}(s, \tilde{a}_\theta(s, \xi)) ... -\alpha \log \pi_\theta(\tilde{a}_\theta(s,\xi) \mid s, \mathbf{z}) \right]$$