Final presentation
Learning safe value functions

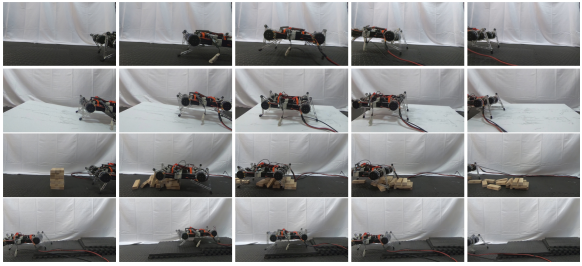
Project thesis

Tsung Yuan Tseng

Supervisor: Alexander von Rohr, M. Sc.
Institute for Data Science in Mechanical Engineering, RWTH Aachen University

July 26, 2022

Context and Problem



1

- Learning directly on hardware usually encounters lots of failures.
- Failures typically mean painful reset time and physical wear and tear in these scenarios.
- How can we encode safety and enhance sampling efficiency during learning while also preserving optimality in the end?

[1] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, et al. (2018). "Soft actor-critic algorithms and applications". In: *arXiv preprint arXiv:1812.05905*

Safety Definitions

Definition (Failure Set)

Failure Set S_F is a set containing the states considered failures.

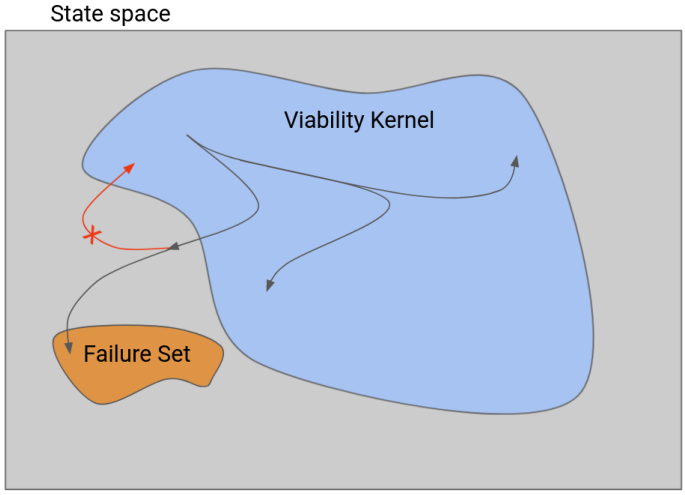
Definition (Viability Kernel [2])

Viability kernel is a maximal set of all states $s \in \mathcal{S}_V$ in which there exists at least one action $a \in \mathcal{A}$ such that the next state $s' = T(s, a)$ is still inside $\mathcal{S}_V$.

[2] Jean-Pierre Aubin et al. (2011). *Viability theory: new directions*. Springer Science & Business Media

Safety Definitions

Cont.



[Steve Heim]

Safety Definitions

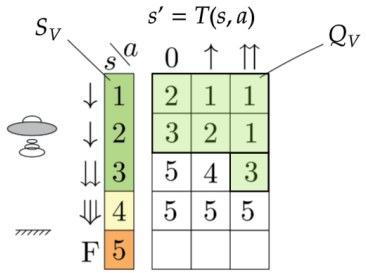
Definition (Viable Set [3])

The viable set $\mathcal{Q}_V := \mathcal{S} \times \mathcal{A}$ is a maximal set of all state-action pair $q \in \mathcal{Q}_V$ such that the next state $s' = T(q)$ is still inside $\mathcal{S}_V$.

Definition (Safety)

Safety means an agent's trajectory always stays in the viable set $\mathcal{Q}_V$.

Example:



4

[4] Steve Heim and Alexander Spröwitz (2019). "Beyond basins of attraction: Quantifying robustness of natural dynamics". In: *IEEE Transactions on Robotics* 35.4, pp. 939–952; Steve Heim, Alexander Rohr, et al. (2020). "A learnable safety measure". In: *Conference on Robot Learning*, pp. 627–639

Problem Formulation

- Consider the objective in entropy-regularized RL (SAC) [5]:

$$G(s,\pi) = \sum_{t=0}^{T-1} \gamma^t \mathbb{E}_{(s_t,a_t,s_{t+1})\sim\rho_\pi}[r(s_t,a_t,s_{t+1}) + \alpha\mathcal{H}(\pi(\cdot|s_t)) \mid s_0 = s]. \quad \text{(U)}$$

- We specify the constraint function:

$$c(s,\pi) = \sum_{t=0}^{T-1} \gamma^t \mathbb{E}_{(s_t,s_{t+1})\sim\rho_\pi}[\delta_{\mathcal{S}_F}(s_{t+1}) \mid s_0 = s]. \quad (1)$$

- Our problem becomes:

$$\max_{\pi} G(s,\pi) \; s.t. \; c(s,\pi) = 0. \quad \text{(C)}$$

[5] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, et al. (2018). "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*, pp. 1861–1870

- Penalized Problem:

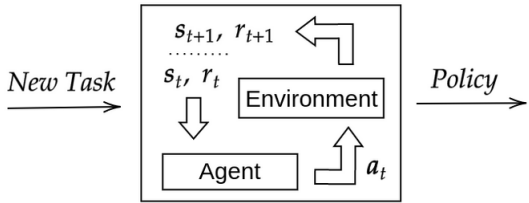
$$\max_{\pi} \; G(s,\pi) - p \cdot c(s,\pi). \tag{P}$$

- Massiani et al. (2021) prove that, under some mild assumptions, there exists p^* such that for all $p > p^*$, all maximizers of the penalized problem (P) are optimal for the constrained problem (C)[6].
- However, in our case, it is hard to tune the penalty because of the existence of the entropy term.

[6] Pierre-François Massiani et al. (2021). "Safe value functions". In: *arXiv preprint arXiv:2105.12204*

Methods

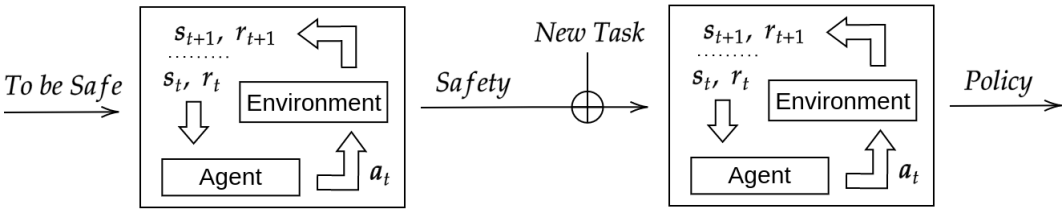
- A typical procedure for solving RL problems:



- But! Too many failures! The need for exploring really hampers safety in RL [7].

[7] Lukas Brunke et al. (2022). "Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning". In: *Annual Review of Control, Robotics, and Autonomous Systems* 5, pp. 411–444

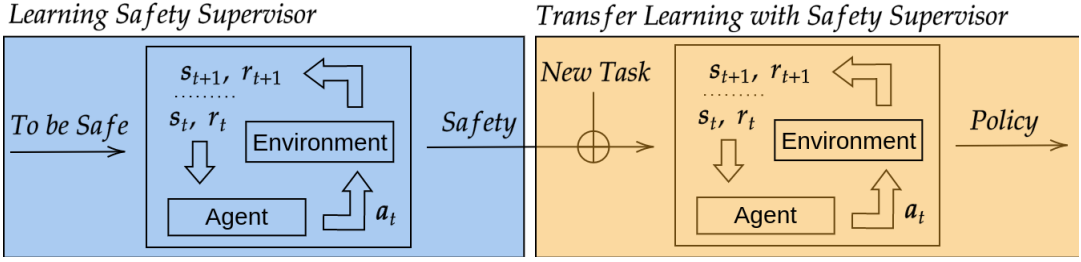
- How about learning to be safe and using this knowledge to learn whatever newly assigned tasks?



Methods

Cont.

- Our framework contains two learning stages.
  1. Learning safety supervisor (LSS) for only avoiding failures (r=0)
     - $\hat{Q}_\phi, \hat{\pi}_\theta$, and $\hat{Q}_V = \{q, \hat{Q}_\phi > 0\}$
  2. Transfer Learning with Safety Supervisor (TL-SS) (r is designed for the new task)
     - Use $\hat{Q}_\phi$ and $\hat{\pi}_\theta$ to initialize networks.
     - Use $\hat{Q}_V$ to guide exploration.



- How can we determine the penalty?

$$\max_{\pi} G(s,\pi) - p \cdot c(s,\pi). \tag{P}$$

- The Lagrangian dual problem:
  - $\mathcal{L}(\pi,\lambda) = G(s,\pi) - \lambda \cdot c(s,\pi)$, where $\lambda = p$
  - Dual function $g(\lambda) = \sup_{\pi} \mathcal{L}(\pi,\lambda) \geq \mathcal{L}(\tilde{\pi},\lambda) = G(s,\tilde{\pi})$
  - Find the best upper bound: $\min_{\lambda} g(\lambda)$
  - Iteratively solves dual sub-problem and dual problem using gradient descent.

- We propose an approach to autotune the penalty:

$$\min_{p} \max_{\pi} G(s,\pi) - p \cdot c(s,\pi). \tag{2}$$

which can be re-written:

$$\min_{p} \max_{\pi} \sum_{t=0}^{T-1} \gamma^t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[r(s_t,a_t,s_{t+1}) - p \cdot \delta_{\mathcal{S}_F}(s_{t+1}) + \alpha\mathcal{H}(\pi(\cdot|s_t)]. \tag{3}$$

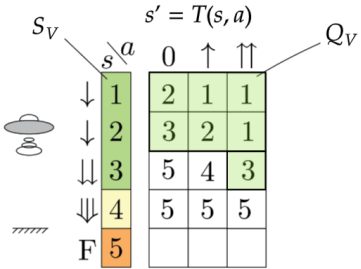
- As solving the maximization problem, by replacing reward with $r(s,a,s') - p \cdot \delta_{\mathcal{S}_F}(s')$, we can apply SAC algorithm.
- As solving the minimization problem, we update p by reducing the following loss function:

$$L(p) = -p \cdot \frac{1}{|\triangle|} \sum_{(s,a,r,s')\in\triangle} \delta_{\mathcal{S}_F}(s'). \tag{4}$$

Methods: Learning Safety Supervisor (LSS)

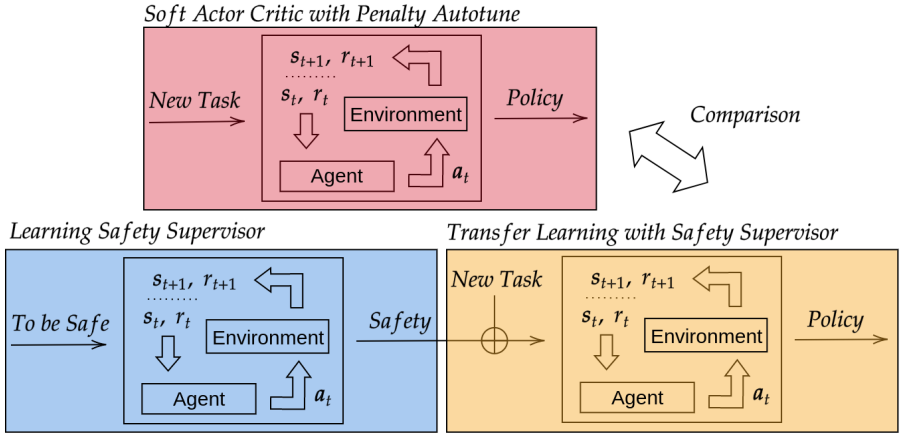
- Why can we recover viable set by $\hat{Q}_V = \{q, \hat{Q}_\phi > 0\}$?

$$\min_{p} \max_{\pi} \sum_{t=0}^{T-1} \gamma^t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi}[-p \cdot \delta_{\mathcal{S}_F}(s_{t+1}) + \alpha\mathcal{H}(\pi(\cdot|s_t)]. \tag{5}$$



- Q is exactly the above objective without counting the entropy bonus of the first step.
- Optimal policy will try to stay Q_V as long as possible.
- Yet, if the agent ever leaves Q_V, it fails ultimately.
- Thus, if the penalty is larger than the entropy bonus gathered along the way to failure, we could threshold the learned Q function by 0.

Experiment

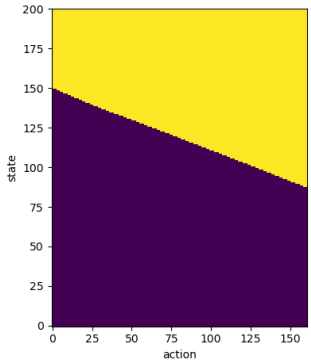


- SAC-PA is used for comparison as we want to answer if safety supervisor can guide the learning safely and sampling efficiently while preserving optimality.

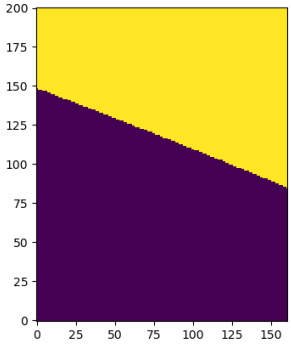
Experiment: Hovership

- We test our algorithm on hovership but in the version of continuous state-action space for 50 runs.

- $\hat{Q}_V$



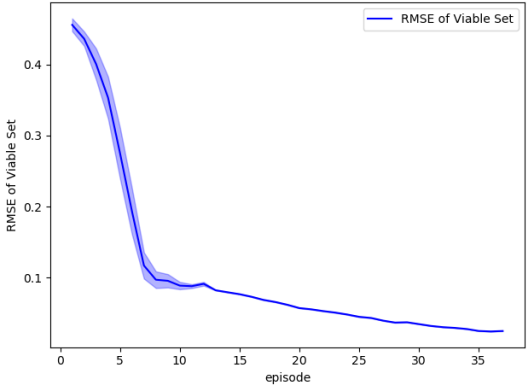
- Ground-truth Q_V by brute-force



Experiment: Hovership

- We monitor the accuracy of $\hat{Q}_V$ via root mean square error:

$$RMSE = \frac{1}{N} \sum_{(s,a) \in Grid} \sqrt{(Q_V(s,a) - \hat{Q}_V(s,a))^2}. \tag{6}$$

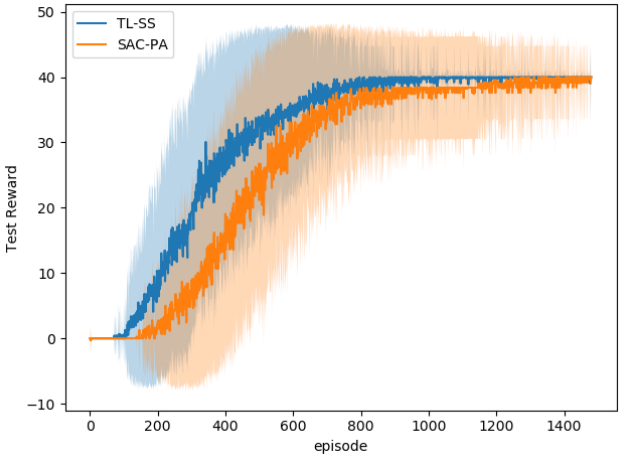


Experiment: Hovership

- The assigned task is to learn the optimal policy such that the hovership moves from the initial state $h = 1.8$ to the goal state $h = 1.3$ as fast as possible. Therefore, we define $\gamma = 0.8$ and the following reward function:

$$r = \begin{cases} 50, \ if \ h = 1.3 \pm 0.01 \\ 0, \ otherwise \end{cases} . \tag{7}$$

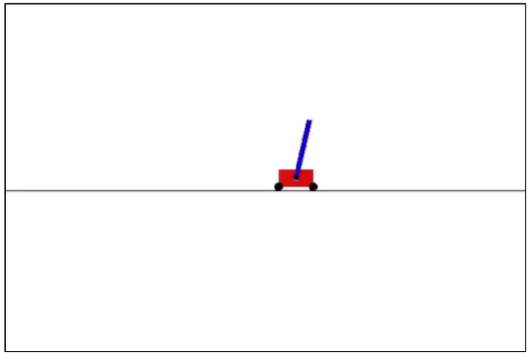
Experiment: Hovership



Algorithm	**Failure Rate**
TL-SS	0.0000 ± 0.0000
SAC-PA	0.0008 ± 0.0001

Experiment: Inverted Pendulum

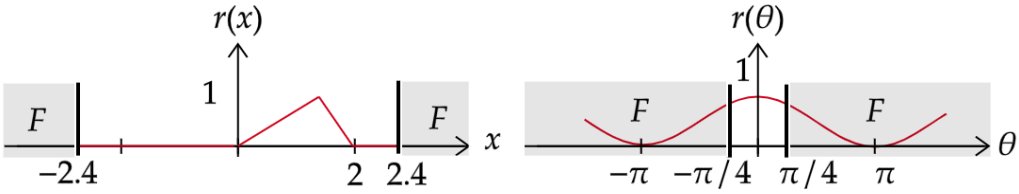
- We test our algorithm on the inverted pendulum in the upward position for 10 runs.
- The failure states are defined as the cart position x reaches the boundaries ± 2.4 or the pole angle θ exceeds the limits $\pm 45°$; $max\ ep\ steps$ and γ are set to 1000 and 1 respectively.



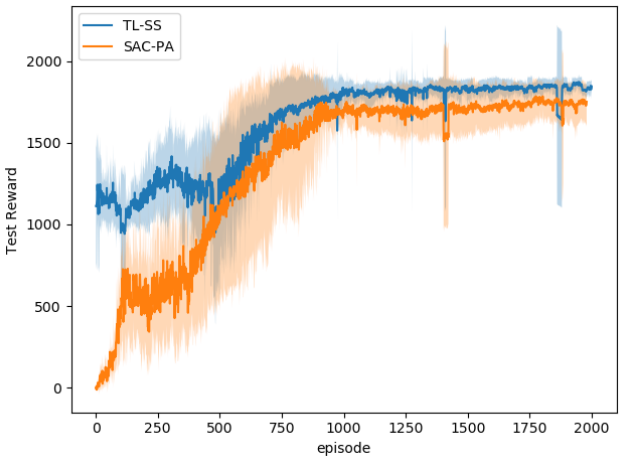
Experiment: Inverted Pendulum

- The assigned task: train the agent moving from the initial state $x = 0, \dot{x} = 0, \theta = 0, and\ \dot{\theta} = 0$ to the goal state $x = 1.5$ while stabilizing the pole.
- Reward function:

$$r = \begin{cases} \frac{1}{2}(1 + cos\theta) + \frac{2}{3}x, \ if\ 0 \leq x \leq 1.5 \\ \frac{1}{2}(1 + cos\theta) - 2x + 4, \ if\ 1.5 < x \leq 2 \\ \frac{1}{2}(1 + cos\theta), \ otherwise \end{cases} . \tag{8}$$

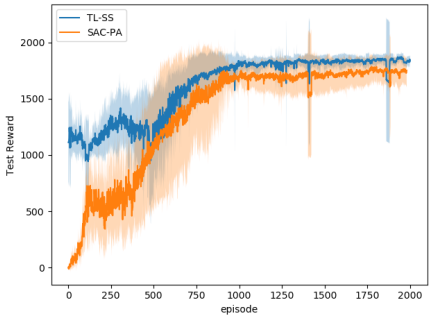


Experiment: Inverted Pendulum



Algorithm	Failure Rate
TL-SS	0.0312 ± 0.0554
SAC-PA	0.6996 ± 0.0554

Experiment: Discussion

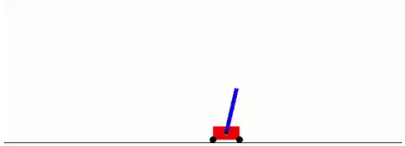


- Our algorithm starts to fail when the agent tries to accelerate to the right and samples the state-action pairs that are out-of-distribution for $\hat{Q}_V$ (Safety supervisor seldom visits before).

- Accuracy of $\hat{Q}_V$ is the essence of success for avoiding failures.
- Soft actor critic algorithm is not sufficient to explore the whole state-action space, and consequently, learn $\hat{Q}_V$.
- Some regions in the state-action space are just hard to visit (probability is rare).
- Active learning on $\hat{Q}_V$ may be one of the future works.

Contributions

1. We propose a practical algorithm, in which we first learn the safety supervisor, and then transfer the safety knowledge to learn a new given task.
2. On hovership example as a proof of concept, we show that once we learn accurate enough safety supervisor, in the transfer learning stage, the learning is safe and sampling efficiently compared to learning from scratch.
3. We evaluate our framework on the high-dimensional task, i.e., inverted pendulum, to shed light on future works.



Bibliography I

- Aubin, Jean-Pierre, Alexandre M Bayen, and Patrick Saint-Pierre (2011). *Viability theory: new directions*. Springer Science & Business Media.
- Brunke, Lukas et al. (2022). "Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning". In: *Annual Review of Control, Robotics, and Autonomous Systems* 5, pp. 411–444.
- Haarnoja, Tuomas, Aurick Zhou, Pieter Abbeel, et al. (2018). "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*, pp. 1861–1870.
- Haarnoja, Tuomas, Aurick Zhou, Kristian Hartikainen, et al. (2018). "Soft actor-critic algorithms and applications". In: *arXiv preprint arXiv:1812.05905*.
- Heim, Steve, Alexander Rohr, et al. (2020). "A learnable safety measure". In: *Conference on Robot Learning*, pp. 627–639.

Bibliography II

- Heim, Steve and Alexander Spröwitz (2019). "Beyond basins of attraction: Quantifying robustness of natural dynamics". In: *IEEE Transactions on Robotics* 35.4, pp. 939–952.
- Massiani, Pierre-François et al. (2021). "Safe value functions". In: *arXiv preprint arXiv:2105.12204*.

Appendix: Learning Safety Supervisor (LSS)

LSS Algorithm

Initialize networks $\hat{Q}_{\phi_1}, \hat{Q}_{\phi_2}, \hat{\pi}_\theta$
for each interaction steps **do**
  $a_t \sim \hat{\pi}_\theta(a_t|s_t)$
  $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
  $\triangle \leftarrow \triangle \cup (s_t, a_t, r_t, s_{t+1}, \delta_{\mathcal{S}_F}(s_{t+1}))$
  for each gradient steps **do**
    Apply SAC update for $\hat{Q}_{\phi_1}, \hat{Q}_{\phi_2}, \hat{\pi}_\theta$
    $p \leftarrow p - \lambda \hat{\nabla} L(p)$
  end for
end for
$\hat{Q}_V = \{q, min(\hat{Q}_{\phi_1}, \hat{Q}_{\phi_2}) > 0\}$
return $\hat{Q}_V, \hat{Q}_{\phi_1}, \hat{Q}_{\phi_2}, \hat{\pi}_\theta$

Appendix: Learning Safety Supervisor (LSS)

LSS Algorithm

Initialize networks $\hat{Q}_{\phi_1}, \hat{Q}_{\phi_2}, \hat{\pi}_\theta$
for each interaction steps **do**
  $a_t \sim \hat{\pi}_\theta(a_t|s_t)$
  $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
  $\triangle \leftarrow \triangle \cup (s_t, a_t, r_t, s_{t+1}, \delta_{\mathcal{S}_F}(s_{t+1}))$
  for each gradient steps **do**
    Apply SAC update for $\hat{Q}_{\phi_1}, \hat{Q}_{\phi_2}, \hat{\pi}_\theta$ **Maximization via SAC**
    $p \leftarrow p - \lambda \hat{\nabla} L(p)$ **Minimization via Penalty Autotune**
  end for
end for
$\hat{Q}_V = \{q, min(\hat{Q}_{\phi_1}, \hat{Q}_{\phi_2}) > 0\}$
return $\hat{Q}_V, \hat{Q}_{\phi_1}, \hat{Q}_{\phi_2}, \hat{\pi}_\theta$

Appendix: Transfer Learning with Safety Supervisor (TL-SS)

TL-SS Algorithm

$\hat{Q}_V, Q_{\phi_1}, Q_{\phi_2}, \pi_\theta \leftarrow$ Learning Safety Supervisor
for each interaction steps **do**
  $a_t \sim \pi_\theta(a_t|s_t) \forall a \ s.t. \ (s_t, a) \in \hat{Q}_V$
  if $\nexists a \ s.t. \ (s_t, a) \in \hat{Q}_V$ **then**
    $a_t \sim \pi_\theta(a_t|s_t)$
    $s_{t+1} \leftarrow \forall s_{closest} \in \mathcal{S}_\mathcal{F}$
  else
    $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
  end if
  $\triangle \leftarrow \triangle \cup (s_t, a_t, r_t, s_{t+1}, \delta_{\mathcal{S}_F}(s_{t+1}))$
  for each gradient steps **do**
    Apply SAC update for $Q_{\phi_1}, Q_{\phi_2}, \pi_\theta$
    $p \leftarrow p - \lambda \hat{\nabla} L(p)$
  end for
end for

Appendix: SAC Penalty Autotune (SAC-PA)

SAC-PA Algorithm

Initialize networks $Q_{\phi_1}, Q_{\phi_2}, \pi_\theta$
for each interaction steps **do**
  $a_t \sim \pi_\theta(a_t|s_t)$
  $s_{t+1} \sim p(s_{t+1}|s_t, a_t)$
  $\triangle \leftarrow \triangle \cup (s_t, a_t, r_t, s_{t+1}, \delta_{\mathcal{S}_F}(s_{t+1}))$
  for each gradient steps **do**
    Apply SAC update for $Q_{\phi_1}, Q_{\phi_2}, \pi_\theta$
    $p \leftarrow p - \lambda \hat{\nabla} L(p)$
  end for
end for