Research Report

Research Question

- What is the maximum policy frequency we can conceivable learn at? Though we have the compute power on Digit to run policies at multiple 100s of Hz, we have them output PD targets at 40Hz. This is because if the policy is running at a really high frequency, there is so little time between actions learning becomes really difficult (and really slow, now you need more samples to sample the same amount of simulation time). It becomes really hard to attribute reward to the correct action since the state evolves so little in between actions. So how quick can we make the policy execution frequency while still learning successfully? I've tried like 100Hz and that worked/transferred ok if I remember correctly. If you go too quickly, you get into some smoothness issues since the actions can change so quickly. How large of a difference does a quicker policy frequency make? Does it make the policy more performant or robust? Does it help sim2real to have state estimation inputs more often?

Table of Contents

Introduction

- In this study, I trained policies under varying frequencies to explore the limits of policy frequency and its impact on learning efficiency and performance. Modifying select parameters in the training script (train.py), I experimented with different policy rates, specifically adjusting arguments dependent on the policy rate, such as trajectory length and number of steps, to ensure adequate simulation time for each episode.

Training

- The training setup used default arguments except for the “policy-rate” and parameters influenced by it. For example, increasing the policy rate reduces the simulation time per episode, requiring an increase in trajectory length to capture enough simulation time within each episode. Key parameter values are as follows:
    a. Policy Rate: 50 Hz
    b. Trajectory Length: 300 steps
    c. Number of Steps: 2000
- These settings yield a trajectory length of 6 seconds in simulation time and a total of 40 seconds per episode across all policies. Consistent trajectory length and episode lengths were maintained at 6 and 40 seconds, respectively, for each policy.

Evaluation

- Policies were evaluated across a range of state noise levels, from 0 to 0.30, in 0.01 increments. State noise was uniformly applied across all state parameters, including orientation noise, angular velocity noise, motor position noise, motor velocity noise, joint position noise, and joint velocity noise. Example state noise values include:
    a. [0, 0, 0, 0, 0, 0]
    b. [0.01, 0.01, 0.01, 0.01, 0.01, 0.01]
    c. …
    d. [0.3, 0.3, 0.3, 0.3, 0.3, 0.3]
- Each policy and state noise combination was tested with a trial count of 100 to ensure robust evaluation.

Constant Arguments

a_lr.	0.0001.
action_dim.	20.
arch.	ff.
backprop_workers.	-1.
batch_size.	64.
bounded.	False.
Workers.	4.

c_lr.	0.0001.
clip.	0.2.
clock_type.	von_mises.
discount.	0.99.
dynamics_randomization.	True.
entropy_coeff.	0.0.
env_name.	LocomotionClockEnv.
epochs.	3.
eps.	1e-06.
eval_freq.	200.
full_clock.	False.
full_gait.	False.
gae_lambda.	1.0.
grad_clip.	0.05.
integral_action.	False.
kl.	0.02.
layers.	256,256.
learn_stddev.	False.
min_eplen_ratio_eval:	0.0.
mirror.	1.
nonlinearity.	relu.
num_eval_eps.	50.
obs_dim.	76.

prenorm.	False.
prenormalize_steps.	100.
previous.	
redis.	None.
reward_name.	locomotion_vonmises_clock_reward.
robot_name.	digit.
save_freq.	-1.
simulator_type.	mujoco.
state_est.	False.
state_noise.	[0, 0, 0, 0, 0, 0].
std.	0.13.
std_array.	
terrain.	
timesteps.	100000000.0.
update_norm.	False.

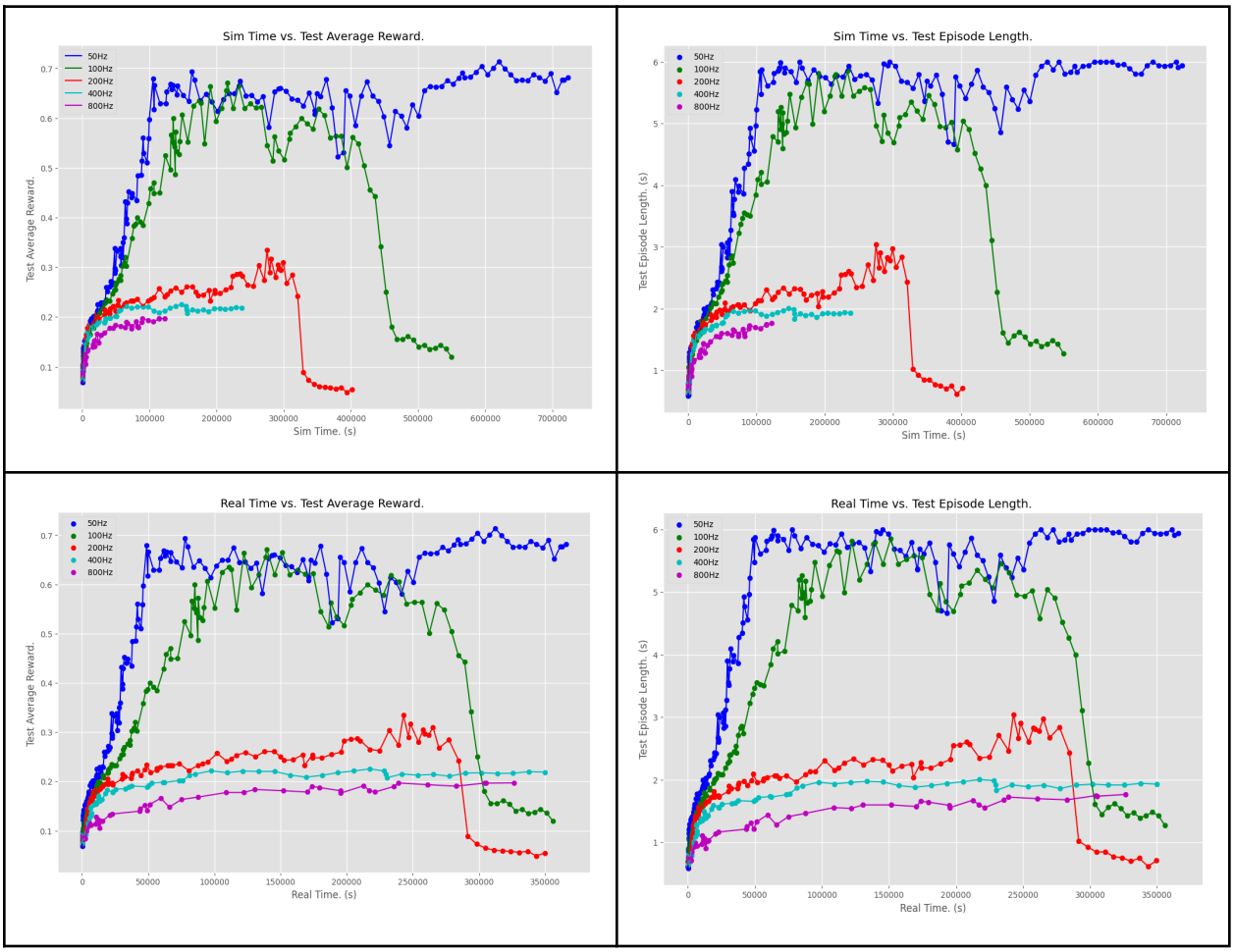
Experiment 1

- In this experiment, a broad range of frequencies was tested to assess their impact on policy performance and robustness. For each frequency, a reinforcement learning policy was trained on an exclusive node with 48 CPUs over a 96-hour period. This approach enabled a comprehensive analysis of how varying frequencies influence both the effectiveness and stability of the policy outcomes.

Policy Rate.	Trajectory Length. (steps) (--traj-len)	Trajectory Length. (converted to simulation time in seconds)	Number of time steps to sample each iteration. (--numsteps)	Number of time steps to sample each iteration. (converted to simulation time in seconds)

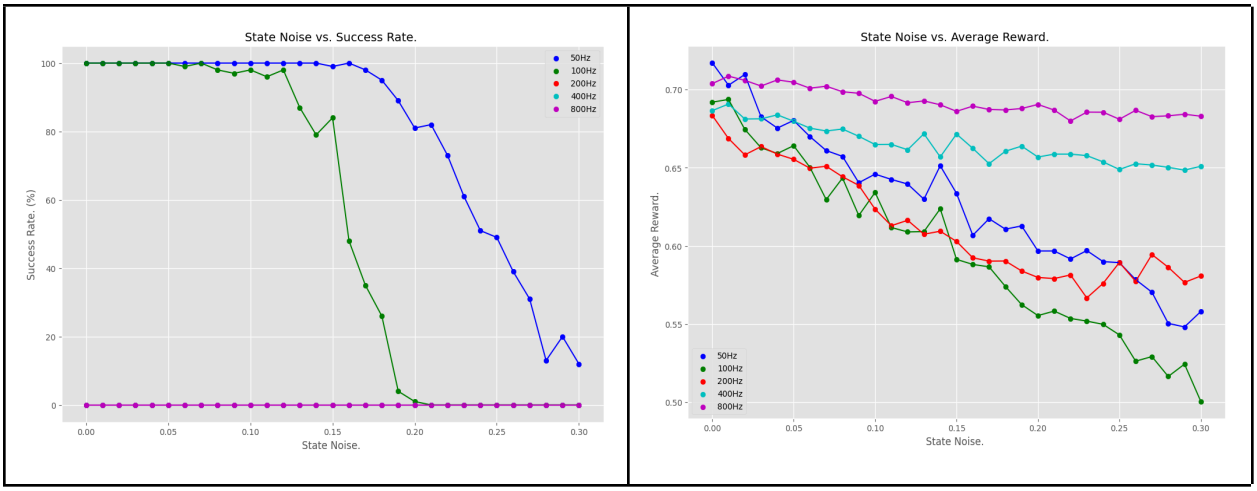
50Hz.	300.	6.	2000.	40.
100Hz.	600.	6.	4000.	40.
200Hz.	1200.	6.	8000.	40.
400Hz.	2400.	6.	16000.	40.
800Hz.	4800.	6.	32000.	40.

Training Results



- The results indicate that policies trained at 50 Hz and 100 Hz succeeded, while those at 200 Hz, 400 Hz, and 800 Hz failed, even with extended training time. The failure at higher frequencies is attributed not only to slower learning rates but also to the inherent difficulty of learning effectively at such rapid action intervals.

Evaluation Results



- The policy trained at a 50 Hz rate demonstrated significantly greater robustness compared to the 100 Hz policy. As learning becomes unfeasible at frequencies above 100 Hz, the optimal policy rate likely lies within the range of 0 to 100 Hz.

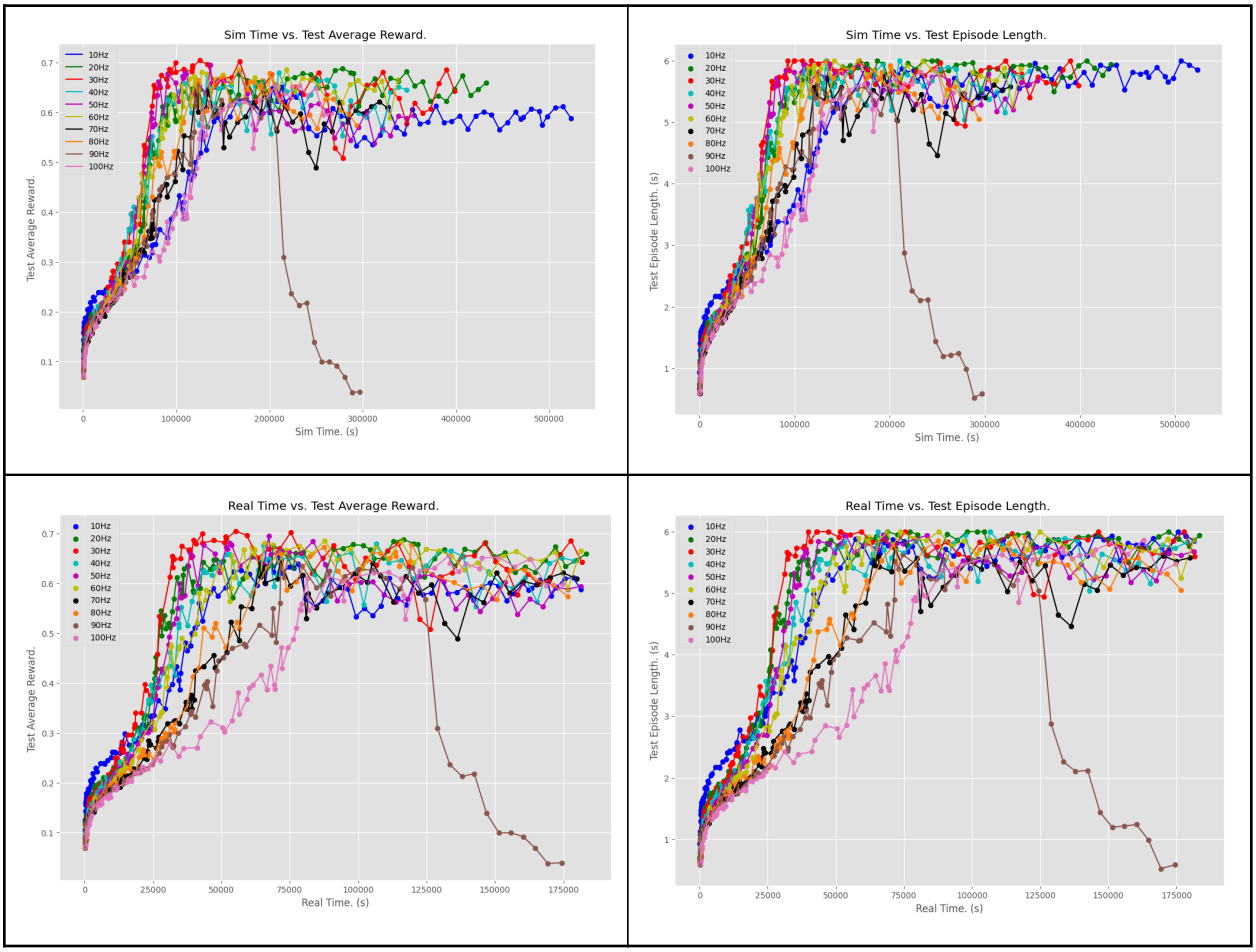
Experiment 2

- In this experiment, a narrower frequency range (10 Hz to 100 Hz) was tested to pinpoint the optimal learning frequency. Each frequency was used to train a reinforcement learning policy on an exclusive node with 48 CPUs over a 48-hour period. This targeted approach aimed to refine the frequency range further and identify the most effective rate for learning.

Policy Rate.	Trajectory Length. (steps) (--traj-len)	Trajectory Length. (converted to simulation time in seconds)	Number of time steps to sample each iteration. (--numsteps)	Number of time steps to sample each iteration. (converted to simulation time in seconds)
10Hz.	60.	6.	400.	40.
20Hz.	120.	6.	800.	40.
30Hz.	180.	6.	1200.	40.

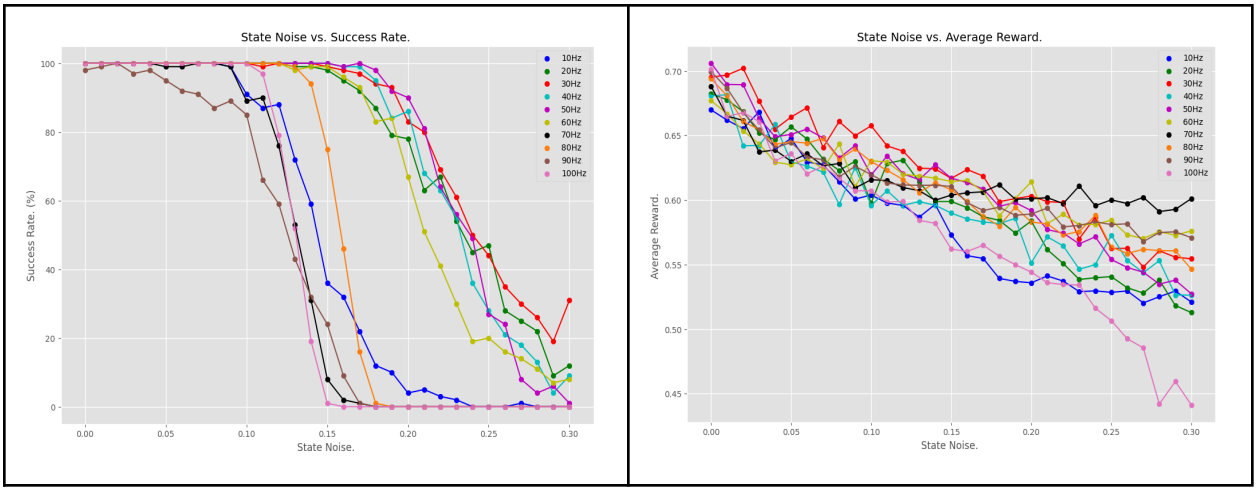
40Hz.	240.	6.	1600.	40.
50Hz.	300.	6.	2000.	40.
60Hz.	360.	6.	2400.	40.
70Hz.	420.	6.	2800.	40.
80Hz.	480.	6.	3200.	40.
90Hz.	540.	6.	3600.	40.
100Hz.	600.	6.	4000.	40.

Training Results



- All tested frequencies successfully achieved learning, with lower-frequency policies reaching milestones more quickly than higher-frequency ones.

Evaluation Results



- Policies trained at frequencies of 20 Hz, 30 Hz, 40 Hz, 50 Hz, and 60 Hz demonstrated greater robustness compared to those at other frequencies.

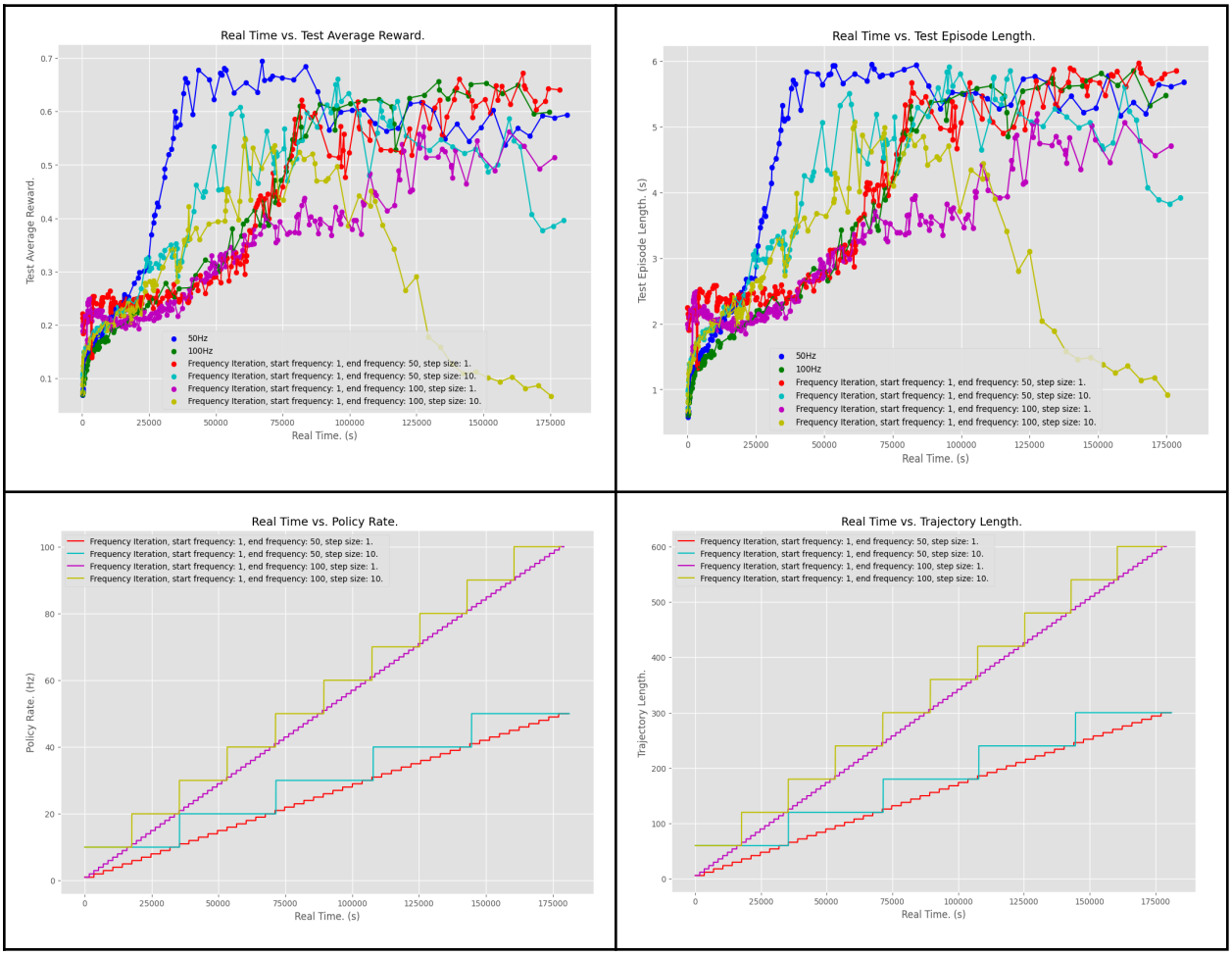
Experiment 3

- Building on previous findings, I observed that policies with lower frequencies learn faster initially, likely due to their ability to sample more simulation time within the same real-time period. However, as training progresses, policies with relatively higher frequencies demonstrate superior performance and robustness. This experiment explores whether training can start with a lower frequency for faster early learning, then progressively increase to a higher frequency, maintaining robustness and performance. This process, termed "frequency iteration," is tested here using varying hyperparameters.
- For instance, with a start frequency of 10 Hz, end frequency of 50 Hz, a run time of 50 minutes, and a step size of 10 Hz, the algorithm adjusts the policy rate as follows:
    a. 0-10 minutes: 10 Hz
    b. 10-20 minutes: 20 Hz
    c. 20-30 minutes: 30 Hz
    d. 30-40 minutes: 40 Hz
    e. 40-50 minutes: 50 Hz
- For each frequency, a reinforcement learning policy was trained on an exclusive node with 48 CPUs over a 48-hour period to assess this method's effectiveness.

Start Frequency.	End Frequency. (Hz)	Run Time. (seconds)	Step Size.

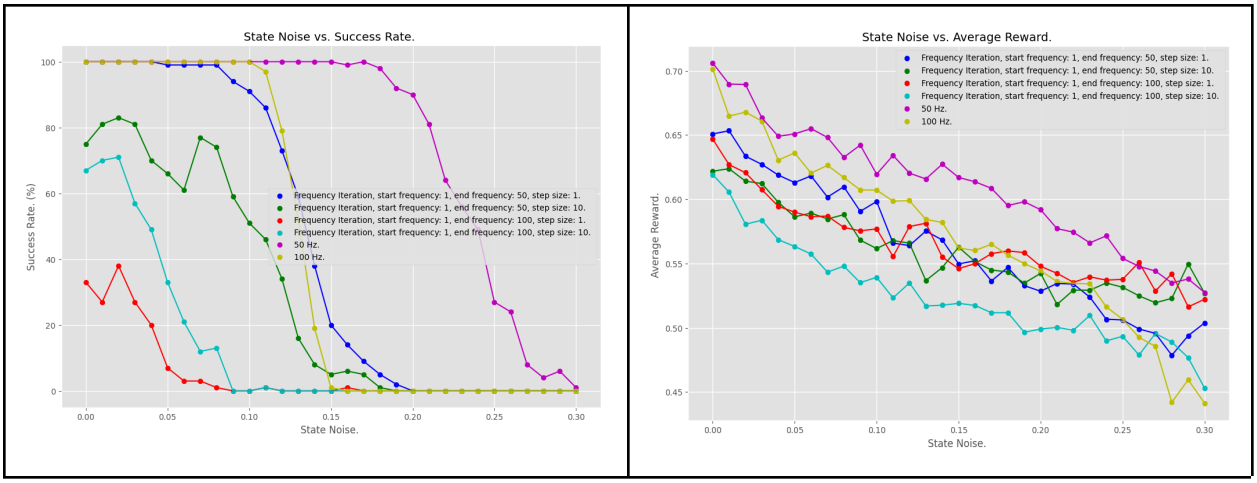
(Hz)			
1.	50.	48*60*60.	1.
10.	50.	48*60*60.	10.
1.	100.	48*60*60.	1.
10.	100.	48*60*60.	10.

Training Results



- The frequency iteration algorithm is functional with suitable hyperparameters; however, it is slower than the baseline policy and exhibits lower performance and robustness.

Evaluation Results



- Although the frequency iteration algorithm was intended to allow for more simulation time within the same real-time training duration, the `compute_policy_rate` function introduced additional processing delays. Consequently, policies using the frequency iteration algorithm sampled significantly less simulation time over 48 hours compared to the policies that don't use frequency iteration.

Conclusion

- The experiments reveal that policy frequencies up to 100 Hz can be effective, but the optimal range for training lies between 10 Hz and 100 Hz, with 50 Hz to 60 Hz offering the best balance of performance, robustness, and learning efficiency. Frequencies beyond 100 Hz were impractical due to challenges in reward attribution and state evolution, while lower frequencies (<50 Hz) enabled faster early learning but less fine-grained action control.
- The proposed frequency iteration algorithm, intended to combine benefits of both low and high frequencies, was functional but inefficient due to computational overhead, reducing its effectiveness compared to static frequency policies.