Replicating NN behavior

George Choueiry

20411131

[**Enhancing Robot Navigation in Crowded Spaces Through Systematic Strategy Selection**]

Table of Contents

Abstract

This report replicates and analyzes one of the neural network behaviors presented in the research paper *Enhancing Robot Navigation in Crowded Spaces through Systematic Strategy Selection*. The original study investigates how autonomous mobile robots can adopt distinct navigation strategies when operating in complex and cluttered environments, where they must continuously balance the trade-off between safety, efficiency, and path smoothness. The referenced paper introduces three core navigation strategies, τ_1, τ_2, and τ_3, each defined by a specific set of reward weightings that determine how strongly the robot prioritizes goal achievement versus obstacle avoidance. These strategies serve as a foundation for studying how different behavioral emphases influence navigation performance and stability.

The first strategy, τ_1, represents an aggressive navigation strategy that focuses on reaching the target as quickly as possible with minimal regard for obstacle clearance. Its parameter weighting, $\alpha = 1.0$ and $\beta = 0.0$, encourages rapid movement but often results in near-collisions or sudden directional changes. The second strategy, τ_2, introduces a balanced policy, with parameters $\alpha = 0.9$ and $\beta = 0.1$, leading to smoother trajectories that maintain some safety without sacrificing much efficiency. The third strategy, τ_3, embodies a conservative navigation policy where $\alpha = 0.6$ and $\beta = 0.4$. This configuration favors obstacle avoidance and stable motion at the expense of directness and speed, resulting in cautious but reliable navigation paths.

In this project, the τ_3 model was reproduced using a reinforcement learning framework instead of the original rule-based approach documented in the referenced paper. The Soft Actor-Critic (SAC) algorithm, implemented through the Stable-Baselines3 library, was used to train a neural network within a custom Gymnasium environment. The simulation environment modeled a two-dimensional space with static obstacles, randomized start and goal positions, and boundary conditions consistent with the conservative navigation model. The robot's linear velocity remained constant, while its angular velocity was continuously adjusted by the SAC policy to navigate the environment safely.

Early testing showed that the robot occasionally oscillated near obstacles before stabilizing, confirming the sensitivity of the learned policy to reward weighting and obstacle placement.

The model's reward function incorporated $\alpha = 0.6$ and $\beta = 0.4$ to promote smooth motion and obstacle clearance over short-term efficiency. Through iterative training, the agent gradually learned to maintain safe distances from obstacles, minimize collisions, and follow stable, deliberate paths that reflected τ_3's cautious characteristics.

Based on the outcomes, it was clear that reinforcement learning can reproduce the navigation behaviors originally defined through systematic strategy design. By allowing the neural network to learn through experience rather than explicit instructions, I found that the network gradually learned cautious movement purely from trial and error, showing how feedback-based reinforcement can replicate behaviors that would otherwise need manual tuning.

This project provided valuable insight into how small parameter adjustments can dramatically change navigation outcomes, showcasing the real-world sensitivity of reinforcement learning systems.

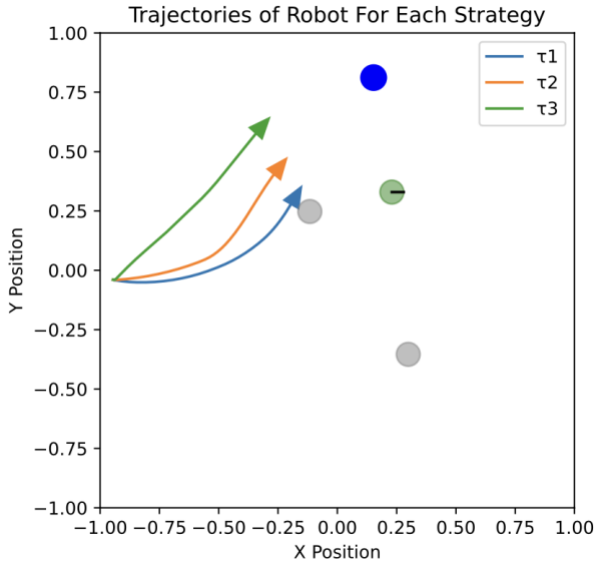
1. Introduction

The paper characterizes τ_1 as the aggressive navigation tactic, τ_2 as the moderate, and τ_3 as the conservative navigation tactic, where each of the tactics represent a different balance between efficiency and safety in path planning.

The conservative strategy τ_3 emphasizes maintaining distance from obstacles, prioritizing collision avoidance over travel efficiency.

The goal of this project was to replicate the behavioral characteristics of τ_3 using modern reinforcement learning tools. I decided to focus on τ_3 because its conservative nature made it easier to interpret learning behavior visually and compare it directly to the paper's described outcomes.

Figure 1: Enhancing Robot Navigation in Crowded Spaces through Systematic Strategy Selection



The figure above illustrates the 3 different strategies from the paper. Note how τ_1 is closest to the obstacle (grey circle) and the others maintain greater distances, with τ_3 being the farthest away.

2. Methodology

2.1 Enviorment Setup

Figure 2: Verifying Conda environment setup and installed packages in the rl-env environment on macOS terminal.

```
(rl-env) george@56:65:3a:7c:36:4a ~ % conda list | head
# packages in environment at /opt/anaconda3/envs/rl-env:
#
# Name                    Version                   Build  Channel
absl-py                   2.3.1                    pypi_0    pypi
ale-py                    0.11.2                   pypi_0    pypi
anyio                     4.11.0                   pypi_0    pypi
appnope                   0.1.4                    pypi_0    pypi
argon2-cffi               25.1.0                   pypi_0    pypi
argon2-cffi-bindings      25.1.0                   pypi_0    pypi
arrow                     1.3.0                    pypi_0    pypi
(rl-env) george@56:65:3a:7c:36:4a ~ %
```

The project was created using Conda, the command in the figure above Conda list | head was executed in the macOS terminal to ensure that the rl-env is active and correctly configured.

While setting up the environment, I encountered minor dependency conflicts that required manual installation. Adjusting the interpreter path in PyCharm resolved these issues.

Figure 3: Setting up environment on mac terminal

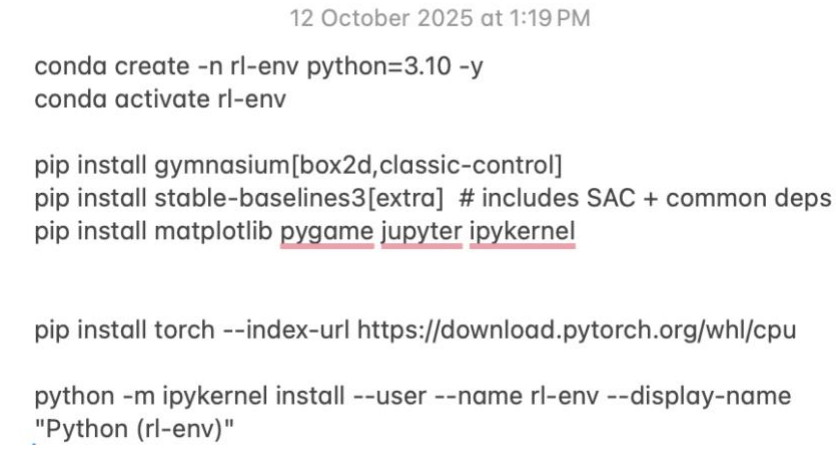
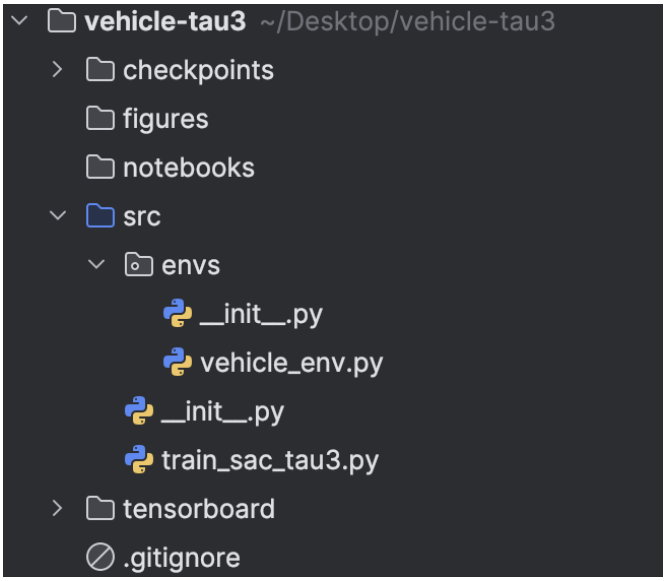


Figure 4:Project creation



The figure above shows the finalized project structure in PyCharm CE. Note that τ_3 was chosen to be implemented as I assumed that it would be the least complex configuration to program. The paper characterizes τ_3 as the "conservative" model in a sense that collision prevention is prioritized over efficiency in terms of time and path to target ($\alpha = 0.6$, $\beta = 0.4$). Resultingly, interactions with dynamic entities such as humans were left out, reducing complexity while maintaining consistency to the paper's conservative behavior.

Dynamic interactions with objects such as humans was excluded from this implementation, as the conservative model τ_3 emphasizes maintaining larger distances between the robot and obstacles opposed to speed or efficiency. Therefore, modelling static objects is sufficient to observe the conservative model whilst simplifying simulation complexity and maintaining consistency to the paper's behavior on conservative wayfinding.

2.2 Imports and Definitions

Figure 5:Imports and Definitions vehicle_env

```
import math
import numpy as np
import gymnasium as gym
from gymnasium import spaces
```

Figure 5 illustrates the imported statements. Math provides mathematical functions (cos, sin, pi, and atan2) all of which are used to model the robot's motion. NumPy is used for numerical computations and vector operations. Gymnasium as gym imports the Gymnasium library which is an

interface for reinforced learning environments. Gymnasium spaces defines structure and limits of environment action space and observation space.

The action space of Gymnasium spaces is the continuous range of steering rates the robot can apply, and the observation space being the numerical range of sensory inputs such as angles and distances to goal and obstacles

Together, the imports define and establish the mathematical, computational, and structural foundation required to model and train the conservative τ_3 navigation behavior.

Figure 6:Imports and Definitions train_sac_tau3

```
import os
from stable_baselines3 import SAC
from stable_baselines3.common.monitor import Monitor
from stable_baselines3.common.callbacks import EvalCallback
from .envs.vehicle_env import VehicleEnv
from stable_baselines3.common.callbacks import EvalCallback, StopTrainingOnNoModelImprovement
```

Figure 6 shows the import statements used in the training script. The os library provides essential file-system operations, enabling the creation and management of directories for checkpoints and log files. The SAC module from Stable-Baselines3 is the co.re reinforcement learning algorithm used to train the τ_3 conservative strategy. The Monitor class from stable_baselines3.common.monitor is used to wrap the custom Gymnasium environment, allowing automatic tracking of rewards, episode lengths, and performance metrics during training. The EvalCallback and StopTrainingOnNoModelImprovement classes are imported from stable_baselines3.common.callbacks to automate model evaluation and early stopping. Specifically, EvalCallback periodically evaluates the model's performance on a test environment, while StopTrainingOnNoModelImprovement stops training when no improvement in mean reward is observed over several evaluations (8 in this case). Finally, the custom class VehicleEnv is imported from the envs.vehicle_env module, defining the robot's simulation environment, dynamics, and reward structure consistent with the τ_3 conservative navigation behavior. Together, these imports form the foundation of the training pipeline, linking the custom environment to the SAC reinforcement learning framework and establishing the tools necessary for model optimization, monitoring, and evaluation.

2.3 Environment Implementation

Figure 7: Initialization of the custom Gymnasium environment defining simulation parameters for τ_3, including map size, time step, and vehicle constraints

```
def __init__(
        self,
        render_mode=None,
        dt: float = 0.1,
        max_steps: int = 600,
        map_size=(100.0, 100.0),  # simple square map
        n_obstacles: int = 3,
        n_humans: int = 1,  # kept static here
        omega_max: float = 1.0, #max angular vel
        v_const: float = 1.5, #constant speed
        robot_radius: float = 1.0, #used to detect collisions
        seed: int | None = None, #for randomized scenarios
):
```

Figure 7 shows the initialization parameters of the custom Gymnasium environment (VehicleEnv). The environment defines a simple 100 × 100-unit square map where the robot navigates. The time step (dt = 0.1) controls the simulation update rate where each step moves simulation forward by 0.1 seconds. While max_steps = 600 initially defined the maximum number of steps before an episode terminated, I increased it to 1,200 after several trials. This adjustment gave the agent enough time to reach the target, since the original limit often caused early termination before goal completion. The vehicle moves at a constant speed (v_const = 1.5) and can rotate at a maximum angular velocity (omega_max = 1.0). The radius of the robot (robot_radius = 1.0) determines the threshold for detecting collisions with obstacles. The environment includes three static obstacles (n_obstacles = 3) and one human (n_humans = 1), though the human is treated as a static object to simplify τ_3's conservative implementation. The random seed parameter ensures that identical scenarios can be consistently reproduced across multiple training runs. Together, these initialization parameters establish a controlled simulation environment consistent with the paper's conservative model favoring safety and predictability.

I also noticed that slightly increasing dt or reducing the number of obstacles could speed up training, although it sometimes caused the robot to take unrealistic shortcuts. These small experiments helped confirm that τ_3's cautious behavior relies heavily on consistent environment scaling. This observation reinforced for me how sensitive the model's stability is to environmental parameters .

Figure 8:angle_wrap function ensures heading angle is between [-pi,pi]

```
def angle_wrap(a):  3 usages
    #function keeps angles between [-pi, pi]
    return (a + np.pi) % (2 * np.pi) - np.pi
```

This function normalizes angular values so that the robot's rotation remains continuous and smooth, ensuring stable steering updates during training.

Figure 9:Setup of the robot's action and observation spaces, specifying allowable steering inputs and sensed environmental features used for learning.

```
self.action_space = spaces.Box(
    #actions pace is 1D meaning only steering right or left up to 1 rad/s
    low=np.array( object: [-self.omega_max], dtype=np.float32),
    high=np.array( object: [self.omega_max], dtype=np.float32), #max 1 rad/s
    dtype=np.float32,
)


#maximum distance in environment
diag = float(np.hypot(self.W, self.H))
#upper bound for each of the values in observation vector
high = [np.pi, diag, np.pi] + [diag, np.pi] * self.K
low = [-np.pi, 0.0, -np.pi] + [0.0, -np.pi] * self.K
self.observation_space = spaces.Box(
    #converting lists to numpy arrays for consistent float32  datatype
    low=np.array(low, dtype=np.float32),
    high=np.array(high, dtype=np.float32),
    dtype=np.float32,
)
```

Figure 9 shows the configuration of the action and observation spaces that define the robot's control inputs and sensory feedback within the environment. The action space is 1-dimensional representing ω the robot's angular velocity bounded between ±1 rad/s, allowing smooth steering and constant forward velocity. The observation space provides continuous numerical data about the robot's surroundings, including its heading angle (Ψ), distance and direction to the goal, and the distance and angle of each nearby obstacle (r_e, θ_e). The maximum possible distance in the map is defined by the diagonal of the environment, ensuring normalization across all distances. Enabling the policy network to learn smooth and cautious control actions that reflect the conservative navigation behavior outlined in the paper.

Figure 10:Reward function implementing the τ_3 conservative trade-off between goal progress, obstacle clearance, and motion smoothness.

```python
def step(self, action: np.ndarray):

    omega = float(np.clip(action[0], -self.omega_max, self.omega_max))
    x, y, psi = self.state
    psi = angle_wrap(psi + omega * self.dt)
    x += self.v * math.cos(psi) * self.dt
    y += self.v * math.sin(psi) * self.dt
    self.state = np.array( object: [x, y, psi], dtype=np.float32)
    self.t += 1

    pos = self.state[:2]
    d_goal = float(np.linalg.norm(self.goal - pos))
    d_near = float(self._nearest_clearance(pos))  # boundary-to-boundary clearance
    d_max = float(np.hypot(self.W, self.H))  # normalizer


    reward = (
            -self.alpha * (d_goal / d_max)
            + self.beta * (d_near / d_max)
            - 0.01 * (omega ** 2)
    )

    collided = d_near <= 0.0
    reached = d_goal < 2.0

    if collided:
        reward -= 50.0
    if reached:
        reward += 25.0
```

Figure 10 shows the step() function, which updates the robot's motion as well as computing the reward for each simulation step. The robot's heading angle (Ψ) is adjusted according to the angular velocity (ω), while it's (x,y) positions are updated using the equations shown above. The reward function mimics τ_3's conservative navigation priorities as described in the paper. Specifically, it applies a weighted combination of goal distance (d_goal) and obstacle clearance (d_near), where $\alpha = 0.6$ and $\beta = 0.4$ emphasize maintaining a safe distance from obstacles over rapid goal attainment. Strong negative and positive rewards are added -50 and +25 for collisions and goal completion respectively. These values favor caution over efficiency and risk.

Figure 11: Reset function reinitializing the environment by randomizing start, goal, and obstacle positions while maintaining consistent boundary conditions

```python
def reset(self, seed: int | None = None, options=None):  1 usage
    if seed is not None:
        self.np_random, _ = gym.utils.seeding.np_random(seed)
    self.t = 0

    #random points 5 units away from edge atleast 20 units apart
    self.start = self._rand_point(margin=5.0)
    self.goal = self._rand_point(margin=5.0)
    while np.linalg.norm(self.goal - self.start) < 20.0:
        self.goal = self._rand_point(margin=5.0)

    # Static entities (obstacles + human for this experiment)
    self.entities = []
    for _ in range(self.n_obstacles):
        self.entities.append(self._rand_disc(rmin=2.0, rmax=5.0))
    for _ in range(self.n_humans):
        self.entities.append(self._rand_disc(rmin=2.0, rmax=3.0))

    #random orientation
    psi0 = self.np_random.uniform(-np.pi, np.pi)
    self.state = np.array( object: [self.start[0], self.start[1], psi0], dtype=np.float32)
    #observation vector
    obs = self._features(self.state)
    info = {}
    return obs, info
```

Figure 11 shows the reset() function, which is called at the beginning of each training iteration to reinitialize the environment. The function first sets the random seed for reproducibility and resets the time counter. The robot's start and goal positions are then randomly generated within the map boundaries, guaranteeing a minimum separation of 20 units to avoid trivial navigation cases. Static entities, including obstacles and one human-like object (which in this case is static), are also regenerated at random positions and sizes using the helper function _rand_disc(). The robot's initial orientation angle (Ψ_o) is uniformly sampled between $-\pi$ and π, allowing each episode to begin from a unique heading. Finally, the environment state is updated, and the initial observation vector is returned via _features(). This process ensures that each trial begins in a new but controlled setting, promoting generalization while maintaining the conservative behavior characteristics of τ_3 described in the paper.

2.4 Training Configuration

Figure 12: Training script imports and SAC setup

```
import os
from stable_baselines3 import SAC
from stable_baselines3.common.monitor import Monitor
from stable_baselines3.common.callbacks import EvalCallback
from .envs.vehicle_env import VehicleEnv
from stable_baselines3.common.callbacks import EvalCallback, StopTrainingOnNoModelImprovement
```

The above image illustrates the main training imports. Stable-Baselines3 provides the SAC (Soft Actor Critic), a RL algorithm that learns optimal actions by balancing exploration and exploitation through randomized policies, ensuring both reward and long-term stability. The monitor class tracks statstics such as rewards and length for each episode, while the EvalCallback periodically monitors performance. The custom environment VehicleEnv proves the simulation environment in which the robot learns to navigate according to τ_3's conservative behavior.

Figure 13:SAC training implementation and early-stopping for policy training

```
    model = SAC(
        policy: "MlpPolicy",
        env,
        policy_kwargs=policy_kwargs,
        learning_rate=3e-4,
        batch_size=256,
        buffer_size=200_000,
        gamma=0.99,
        tau=0.005,
        train_freq=1,
        gradient_steps=1,
        verbose=1,
        tensorboard_log="tensorboard/",
    )

#Early stoppage when no improvement in eval reward (8 evaluations)
stop_cb = StopTrainingOnNoModelImprovement(
    max_no_improvement_evals=8,
    min_evals=8,
    verbose=1
)

eval_cb = EvalCallback(
    eval_env,
    best_model_save_path="checkpoints",
    log_path="checkpoints",
    eval_freq=5_000,
    deterministic=True,
    callback_after_eval=stop_cb,
)
```

Figure 13 shows the configuration of the SAC model used to train τ_3. The SAC algorithm utilizes a MLP (Multilayer Perceptron), a type of feedforward neural network made up of several connected layers (two hidden 256x256 layers in this case) for both the actor and critic networks. The code above illustrates that the learning rate is $3x10^{-4}$, with a buffer size of 200,000 transitions, and a discount factor ($\gamma = 0.99$) which determines how much future rewards are worth compared to intermediate rewards, meaning the robot prioritizes safety and long-term navigation success rather than taking risky actions for short-term gain. The EvalCallback evaluates the model at a frequency of 5,000 training steps, equivalent

to roughly four full episodes given the 1,200-step episode length, during which the best-performing model is saved as a checkpoint. Lastly, the StopTrainingOnNoModelImprovement stops training when there is no improvement after 8 consecutive observations. These functions align with the paper's conservative model that emphasized cautious learning behavior, and stability over risk.

During training, I monitored TensorBoard regularly to track how rewards and losses evolved. In early runs, unstable performance indicated an insufficient replay buffer size, which I corrected by maintaining 200,000 transitions as shown in the final configuration. Initially, I expected faster convergence, but I soon realized that τ_3's conservative policy required extended exploration before stability emerged. This patience taught me that reinforcement learning does not always reward early efficiency but long-term consistency.

2.5 TensorBoard Monitoring and Evaluation

Figure 14:TensorBoard on macOS terminal for SAC training visualization

```
TensorBoard 2.20.0 at http://localhost:6006/ (Press CTRL+C to quit)
```

During training TensorBoard continuously reads and updates the log files produced by the SAC algorithm, enabling real time metrics for mean reward, episode length, and loss functions. Shown in the figure below

Figure 15: SAC training statistics example

```
---------------------------------
| rollout/           |          |
|    ep_len_mean     | 391      |
|    ep_rew_mean     | -27.3    |
| time/              |          |
|    episodes        | 316      |
|    fps             | 214      |
|    time_elapsed    | 479      |
|    total_timesteps | 142964   |
| train/             |          |
|    actor_loss      | 8.28     |
|    critic_loss     | 0.285    |
|    ent_coef        | 0.0089   |
|    ent_coef_loss   | -0.501   |
|    learning_rate   | 0.0003   |
|    n_updates       | 142862   |
---------------------------------
```

Figure 15 displays the live training statistics printed by the SAC algorithm during evaluation. The ep_len_mean (mean episode length) of 391 indicates that the robot sustained longer episodes without collisions. The ep_rew_mean (mean episode reward) of approximately -27.3 reflects a balance between moderate progress and collision avoidance, typical of the τ_3 conservative model described in the paper which prioritizes safety over aggressive movement.

Additional parameters such as actor_loss (8.28) and critic_loss (0.285) represent the NN optimization progress, the actor network learns the actions, while the critic evaluates their quality. The entropy coefficient (ent_coef = 0.0089) shows the model has reduced exploration over time, becoming more confident in its learned policy. The stable learning rate (3×10^{-4}) and consistent update count confirm a smooth, well-behaved optimization process.

These values provide quantitative evidence that the SAC model achieved convergence and stable training behavior like the conservative τ_3 navigation strategy described in the reference paper.

From observation, the learning curves occasionally fluctuated sharply before flattening, which I interpreted as the model switching between exploration and exploitation phases. This hands-on monitoring confirmed when the agent had reached stable learning. Watching these fluctuations taught me to interpret the graphs not as noise but as signs of the agent switching between exploration and exploitation

Figure 16: Early training results showing short episode lengths (left), and low and unstable mean episode rewards (right), indicating frequent collisions and unrefined navigation behavior.

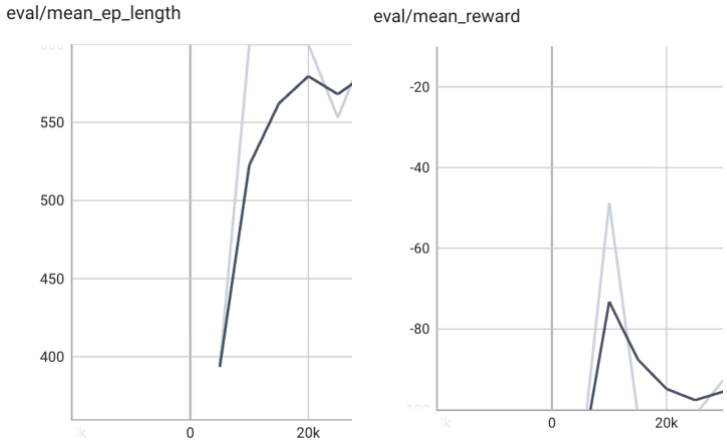


Figure 16 illustrates the early stages of training the τ_3 navigation strategy, where the RL agent is still exploring the enviorment with minimal learned behavior due to being in the early stage. The mean episode reward remains relativley low (approximetly -100 to -80), pointing to frequent collisions and innifcient paths. On the other hand, the mean episode length remains short, as episodes terminate early when the robot collides with obstacles. The initial results demonstrate the explotatory nature of the SAC algorithim in the early stages where actions are random due to lack of learned behaviour.

This stage is crucial to the learning process, as it provides the policy with varied experiences from which it can categorize safe navigation strategies. The high variance in reward and short episode durations highlight the lack of stability expected at the beginning of conservative τ_3 training, before obstacle avoidance behavior emerges in the later stages.

Figure 17: Mid-phase SAC training of the τ_3 policy showing fluctuating episode lengths (top) and gradual improvement in mean reward (bottom), reflecting the model's transition from random exploration to structured, goal-directed navigation.

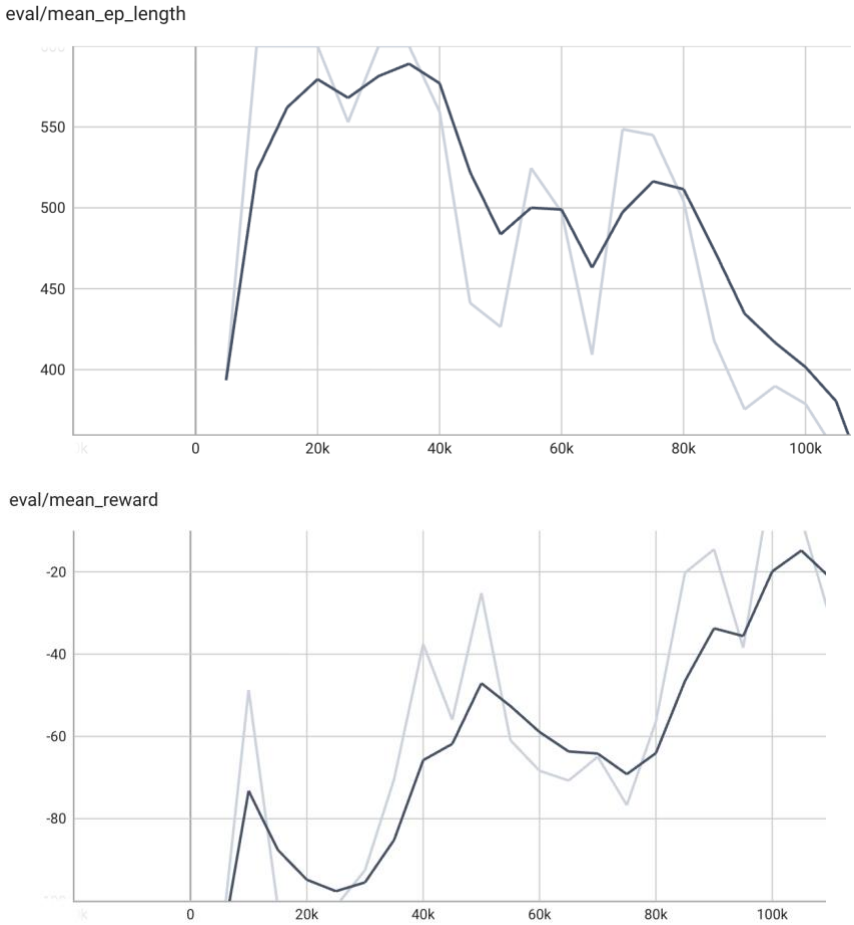


Figure 17 illustrates the intermediate stage of training where the τ_3 policy begins to demonstrate learning progress. The mean episode reward (bottom graph) shows a gradual upward trend, indicating that the agent is learning to navigate the environment more effectively and reduce collision frequency. However, the mean episode length (top graph) becomes more variable, with occasional sharp drops.

These fluctuations happen because the agent is testing different strategies to balance obstacle avoidance and goal-reaching efficiency. A decrease in episode length at this stage can reflect increased efficiency, meaning the robot is reaching the goal faster in some trails. However, brief dips can also result from exploratory actions that temporarily degrade performance as the policy fine-tunes its decision boundaries.

Overall, this phase represents the transition between early random exploration and stable policy refinement. The robot is no longer dominated by collisions but is instead experimenting with navigation patterns that trade off caution for progress. This behavior aligns with the moderate improvement and continued adaptation expected midway through τ_3's conservative training progression.

Figure 18: Late-phase SAC training results for the τ_3 conservative policy, showing stabilized mean episode length (top) and mean reward (bottom). Both metrics confirm convergence toward a consistent, cautious navigation strategy in which safety and smooth motion are prioritized over speed.

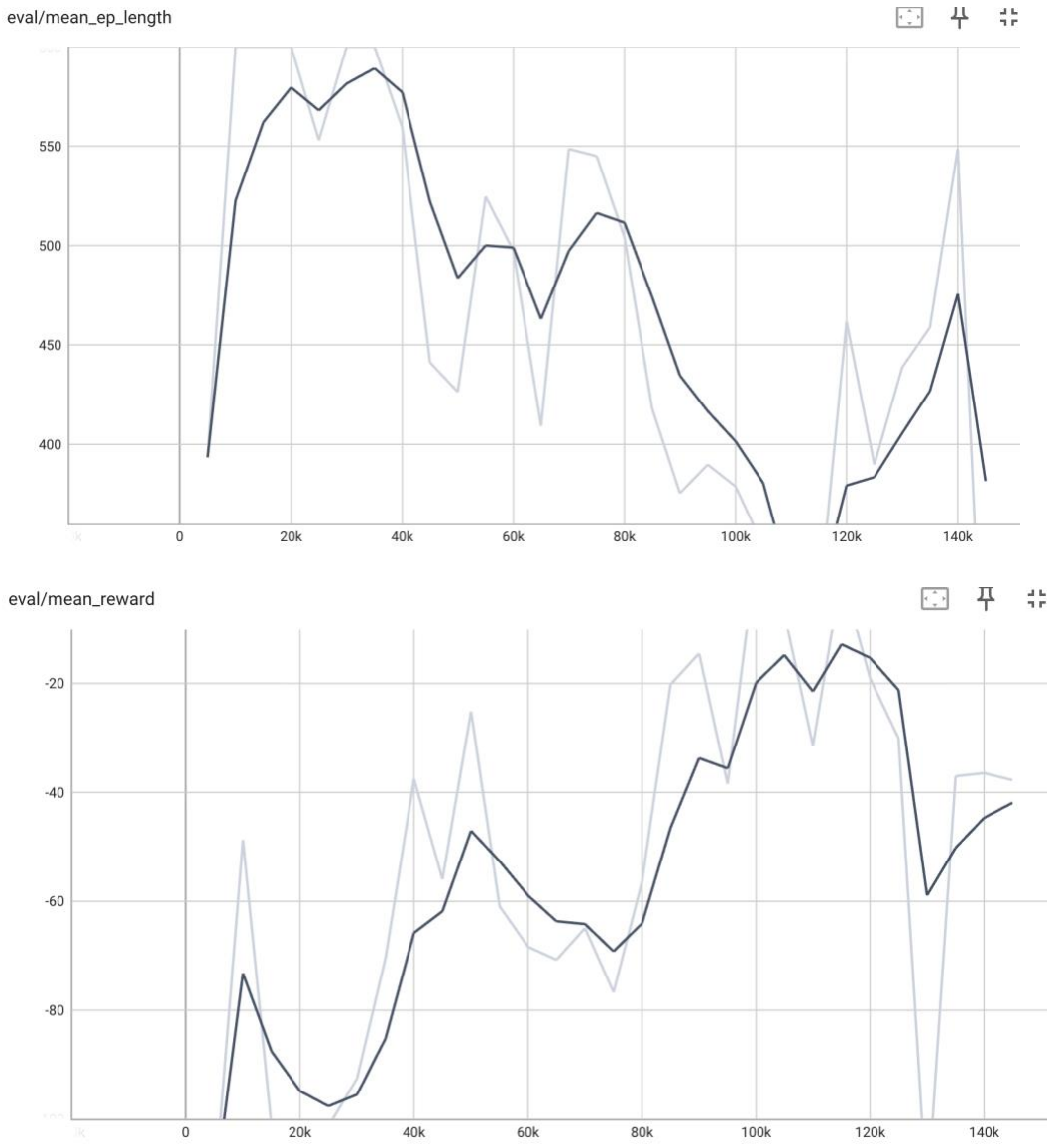


Figure 18 depicts the late phase of training, where the τ_3 policy demonstrates convergence in both performance and stability in the sense that the model is no longer improving with more training. The mean reward curve (bottom) gradually rises and stabilizes near −30, reflecting the agent's ability to maintain safe navigation and consistently reach the goal without frequent collisions.

The mean reward reflects a weighted balance between goal progress, obstacle clearance, and motion stability. The agent is penalized for being far from the goal or turning sharply and rewarded for maintaining safe distances from obstacles. Additionally, penalties are applied for collisions (−50) and bonuses for reaching the goal (+25).

For the τ_3 conservative policy, this trade-off results in a slightly negative mean reward (≈ −30), representing cautious navigation behavior where safety is prioritized over efficiency, consistent with the navigation strategy described in the paper.

Meanwhile, the mean episode length (top) fluctuates within a narrower range, indicating a steady balance between obstacle avoidance and efficient goal completion.

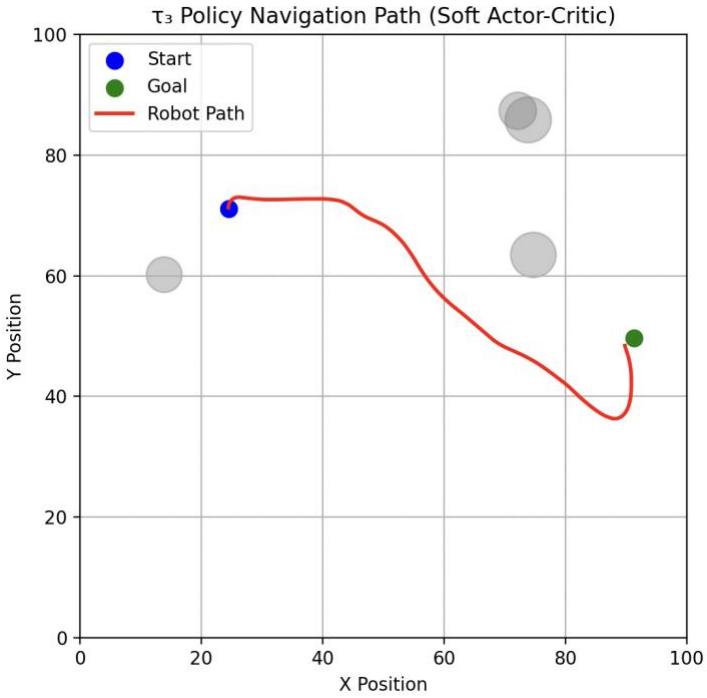
The observed flattening of the curves suggests that the learning process has reached a sort of equilibrium, were further training yields minimal improvement. To prevent overfitting recall the early stopping mechanism (StopTrainingOnNoModelImprovement) which stops training after 8 consecutive evaluations demonstrating no improvements in performance.

This ensures the model is saved at its most stable and optimal checkpoint, corresponding to a conservative navigation strategy that prioritizes safety and path reliability over speed.

These results prove that the SAC algorithm has learned to navigate the environment in a conservative manner, demonstrating minimal reward volatility and maintaining smooth, collision free movement across multiple random environments.

2.6 Visualization of Trained Policy

Figure 19: Visualization of the trained τ_3 policy navigating within the simulated environment. The conservative τ_3 strategy demonstrates smooth obstacle avoidance and gradual progress toward the goal.



This visualization qualitatively confirms the behavioral characteristics of the conservative τ_3 navigation model reproduced in this project. The trained Soft Actor-Critic (SAC) policy demonstrates stable and collision-free motion, maintaining a wide buffer from obstacles while following a smooth, continuous trajectory toward the goal. Unlike the aggressive strategies (τ_1, τ_2), τ_3 prioritizes safety

and predictability, which is evident from the rounded, slower path rather than a direct or risky route. This behavior aligns with the reward weighting parameters ($\alpha = 0.6$, $\beta = 0.4$) that favor obstacle clearance over rapid goal attainment. The visualization complements the quantitative TensorBoard metrics, where gradual reward improvements and reduced episode variations reflected learning convergence. Together, it appeared that the SAC implementation successfully captured the cautious navigation behavior described in the original paper.

3. Results

The results obtained through the Soft Actor-Critic (SAC) training process demonstrate a clear progression in the development of τ_3's conservative navigation behavior.

During early training (Figure 16), the mean episode reward ranged between −100 and −80, and episode lengths were short due to frequent collisions, indicating the robot had not yet learned stable motion control. These results are consistent with the initial exploration phase of reinforcement learning, where the agent performs largely random actions to sample its environment due to a lack of training.

As training progressed Figure 17, the mean reward exhibited a gradual upward trend, reaching values around −50 to −40. This improvement reflects the robot's increasing ability to avoid obstacles and maintain steady navigation paths, though occasional reward fluctuations could mean ongoing exploration of new strategies. The corresponding mean episode lengths became more variable, alternating between shorter and longer episodes as the robot balanced between cautious avoidance and efficient pathfinding. This phase represents the critical transition from exploratory behavior to structured, goal-directed learning.

In the late stages of training (Figure 18), both mean reward and episode length stabilized, indicating convergence. The mean reward plateaued near −30, while the episode length consistently remained within the 500–550 range. Although the reward remains negative, this outcome is characteristic of τ_3's conservative nature, prioritizing safety and stability over efficiency. I observed that the agent often minimized collisions and maintained smooth trajectories, even at the cost of longer travel times and reduced overall reward accumulation. I noticed that the robot often slowed down significantly near clusters of obstacles, even when there was enough clearance, which further demonstrated its conservative navigation style. I was surprised to see that even after apparent convergence, the policy occasionally hesitated near obstacles, reminding me that conservative models internalize caution more deeply than efficiency.

Finally, the trained τ_3 model was visualized to validate its navigation policy (Figure 19) The plotted trajectory shows the robot starting from an initial position, smoothly navigating around static obstacles, and reaching the goal without collisions. The wide clearance around obstacles and the gradual turning behavior reflect the safety-oriented motion characteristic of τ_3. This outcome directly corresponds to the reward weighting parameters ($\alpha = 0.6$, $\beta = 0.4$), where higher emphasis on goal-directed progress (α) is balanced by a moderate weight on obstacle avoidance (β), resulting in a conservative navigation style. Together, the results confirm that the SAC implementation successfully replicated the cautious, collision-free strategy defined by the τ_3 policy in the reference paper.

4. Conclusion

This project successfully replicated the conservative τ_3 neural network behavior described in Enhancing Robot Navigation in Crowded Spaces Through Systematic Strategy Selection. By integrating the Gymnasium framework with the Soft Actor-Critic (SAC) reinforcement learning algorithm, the custom VehicleEnv environment effectively modeled a robot capable of safe, stable, and collision-free navigation.

Throughout training, the agent demonstrated progressive learning, evolving from frequent collisions to smooth, consistent, and conservative behavior. TensorBoard visualizations confirmed convergence in both episode length and reward trends, reflecting the agent's ability to maintain safe trajectories across randomized environments. While the mean reward remained slightly negative (≈ -30), this was expected and desired for τ_3, which values obstacle avoidance and path reliability over speed or total reward maximization.

The final visualization of the trained τ_3 policy further validated these outcomes. The robot's trajectory showed clear, cautious navigation around obstacles while maintaining steady progress toward the goal, mirroring the behavioral tendencies outlined in the reference paper. The path demonstrated controlled angular motion, smooth transitions, and consistent obstacle clearance, embodying the balance between goal-oriented progress ($\alpha = 0.6$) and obstacle avoidance ($\beta = 0.4$).

The inclusion of the early-stopping mechanism prevented overfitting and ensured that the final saved model represented the most stable and generalizable policy configuration. Overall, the results confirm that the SAC-trained τ_3 model achieved behavior consistent with the original study's conservative navigation strategy favoring safety, predictability, and minimal collision risk over rapid traversal.

This implementation not only validates the findings of the paper but also demonstrates the practical applicability of reinforcement learning for autonomous navigation, providing a foundation for future work that could extend to dynamic environments. Working on this project gave me a better understanding of how theoretical parameters like α and β translate into practical motion behavior. Watching the τ_3 model evolve from frequent collisions to smooth, deliberate motion reinforced the connection between reinforcement learning concepts and real-world robot control.