Lab Report

Ruyi TANG, Guanyu CHEN

In this report, we use the algorithm of DDPG to study the effect of partial observability of the CartPoleContinuous-v1 environment. And with different wrappers, we explore their i -nfluences on the partial observability issue.

Exercice 1: FeatureFilterWrapper

This wrapper is designed to filter out specific feature from the observation space. By removing certain features from the environment's observation, the agent must learn to operate with partial information, which can help in analyzing the importance of certain features for decision-making.

Key parameters:

- `env`: For the environment to be wrapped and from which we remove the designated features, we use:`gym.make('CartPoleContinuous-v1')` to create the environment.
- `feature_index`: The index of the feature to be removed from the observation space. And in the exercice, we need to filter the features of **velocity (index=1)** and **angular velocity (index=3)**.

Functions:

All the wrappers in this lab have three functions:`__init__()`, `reset()`, and `step()` but are designed differently for each wrapper.

- `__init__()`: In this function, we designate the index of features to be removed and adjust the observation space to delete the designated features.
- `reset()`: When the environment is reset, it returns the observation with the designated feature removed.
- `step()`: In this function we execute the action to remove the designated features and return the filtered observation.

The algorithm of feature filtering is to test whether the agent can learn and adapt without certain features in the environment.

Exercice 2: ObsTimeExtensionWrapper

This wrapper extends the observation space by including both the current observation and the previous one. This effectively provides the agent with a memory of the last state, which can be useful in partially observable environments, where the current observation alone may not provide enough information to make an optimal decision.

Key parameters:

- `env`: we continue to use the same environment to be wrapped

Functions:

- `__init__()`: In this function, we get the shape of the observation space and extend it by concatenating the current observation with the previous one, meaning double the dimensional size of the observation space and we finally initialize to save the previous observations.
- `reset()`: Upon resetting, the previous observation is initialized to one, and we extend the the observation space with current observation..
- `step()`: In this function the previous observation is updated to be the current observation, and the observation returned to the agent includes both the previous and current observations.

The algorithm of observation extension is to provide additional temporal information (current + previous state) to the agent.

Exercice 3: AcrionTimeExtensionWrapper

This wrapper extends the action space by allowing the agent to output a sequence of **M** actions at once, instead of a single action. The environment only executes the first action in the sequence during each step, but the wrapper allows for experiments where an agent can plan or provide action sequences in advance.

Key parameters:

- `env`: we continue to use the same environment to be wrapped.
- `M`: The number of actions in the extended action sequence. This parameter determines how many actions are outputted by the agent at each step, although only the first action will be executed. And in the experiment, we first give results for cases when **`M=2`**, and in the Bonus part, we will show results when **`M=3`**.

Functions:

- `__init__()`: In this function, we Store the original action space and Extend the action space size to M times the original action space.
- `reset()`: We call the reset method of the inner environment to initialize it and returns the initial observation.
- `step()`: In this function we Extract the first action from the extended action. Then we clip the action to make sure it's within the action space bounds.

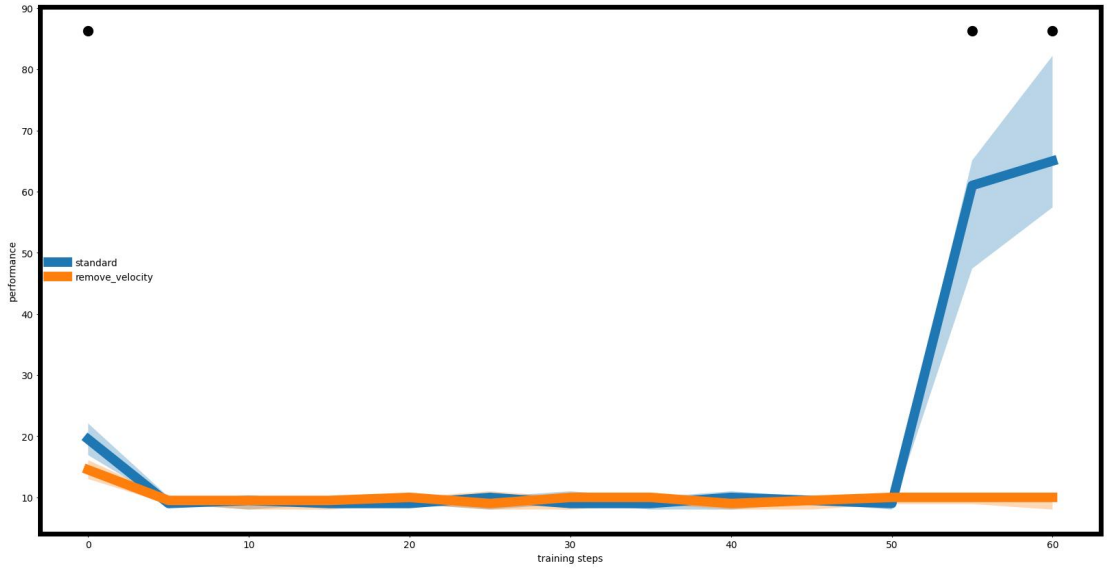
The algorithm of action extension is for the agent to plan over an extended horizon by providing action sequences, but only the first action is used, which could be useful for exploring environments where action sequences are meaningful, or for agents that need to learn to refine long-term action planning.

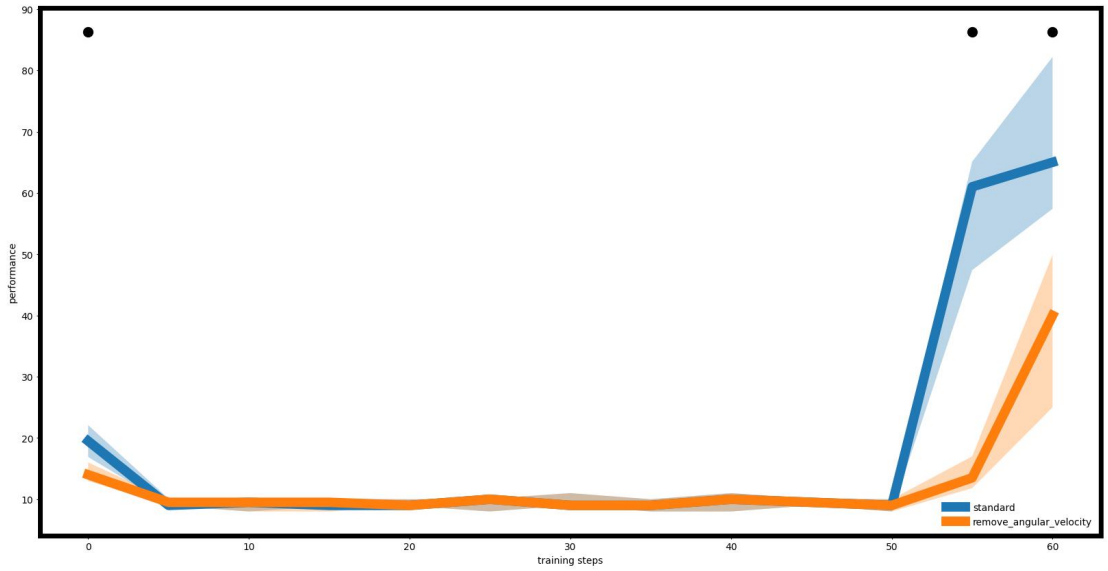
Exercice 4: Learning Curve

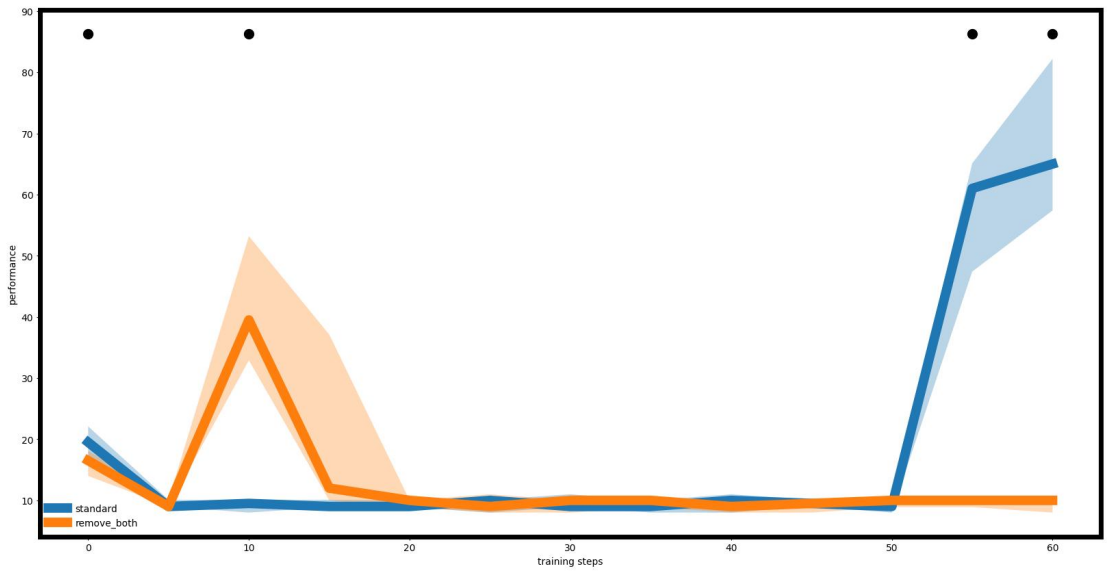
In this exercise, we plot four learning curves under the standard condition, using the ObsTimeExtensionWrapper, and using the ActionTimeExtensionWrapper. The learning curves have training steps on the x-axis and reward on the y-axis, with smoothing applied. By comparing the performance of the learning curves, we study the impact of removing features from the environment on training performance, as well as the impact of temporally extending the agent to mitigate partial observability, using both observation and action extension.

Standard Condition

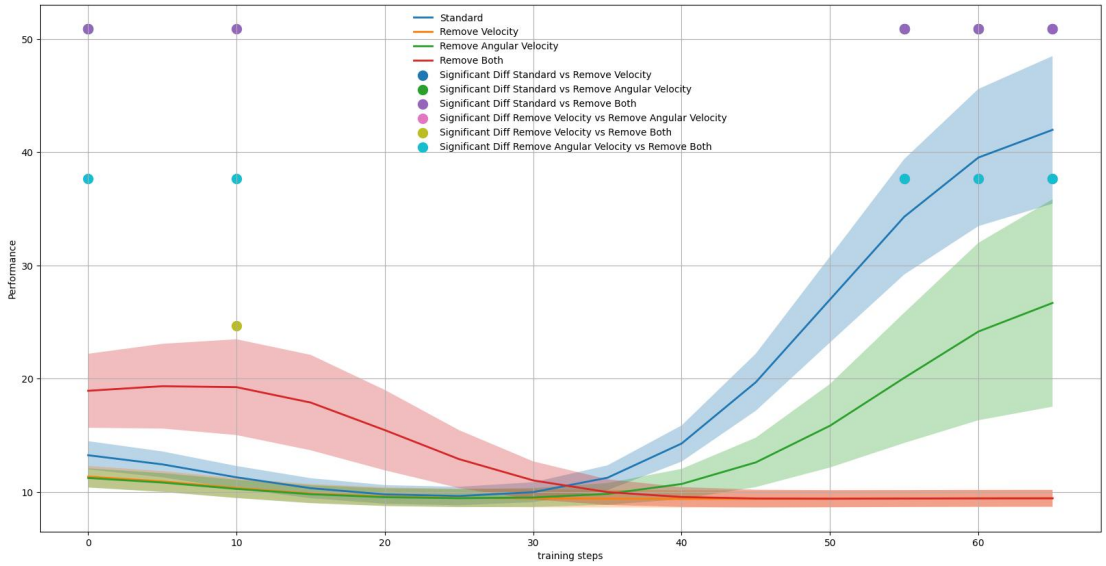
In the first experiment, we use the standard environment condition. By observing the environment without using the any extension wrapper, we study the impact of removing velocity and angular velocity on training performance.
We used the Welch T-Test to compare the performance between the standard case and three other partially observable cases.


standard
remove_velocity
performance
training steps


standard
remove_angular_velocity
performance
training steps


standard
remove_both
performance
training steps

In these three comparison plots, we can see that the algorithm in the standard case performs as expected, reaching a relatively high level after training, indicating that the DDPG algorithm is running successfully. In the case where velocity is removed, the algorithm's reward remains consistently low, suggesting that it fails to develop an effective strategy. When angular velocity is removed, the reward reaches a relatively high level after training, although it is still lower than in the standard case, indicating that removing angular velocity leads to a slightly less effective strategy compared to the standard case. In the scenario where both velocity and angular velocity are removed, the reward fluctuates significantly in the early stages and ultimately stabilizes at a low level. This suggests that, with the loss of both features, the algorithm's performance becomes unstable and it is unable to learn an effective strategy.

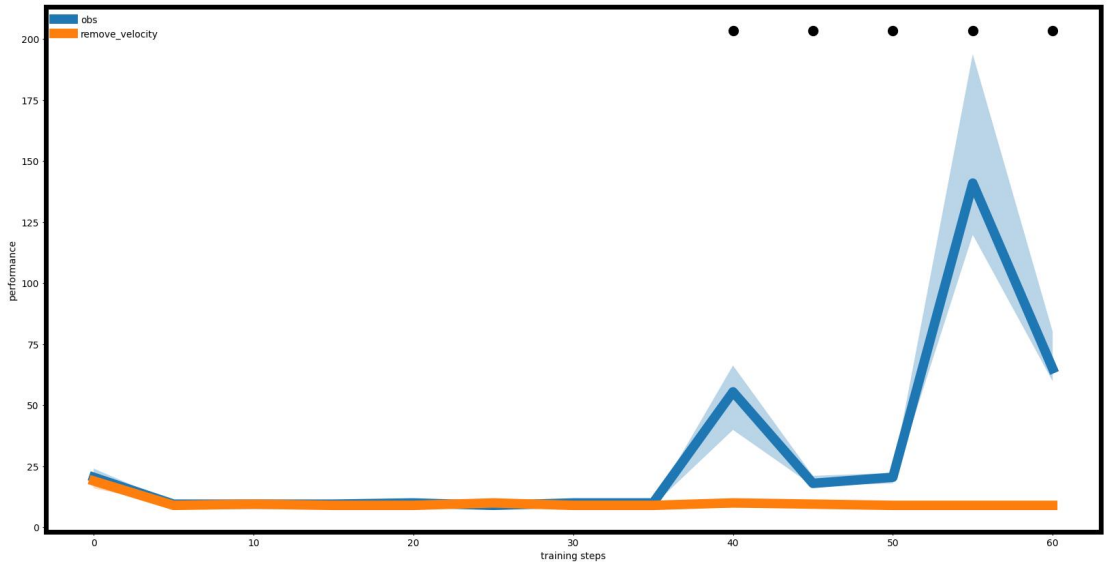


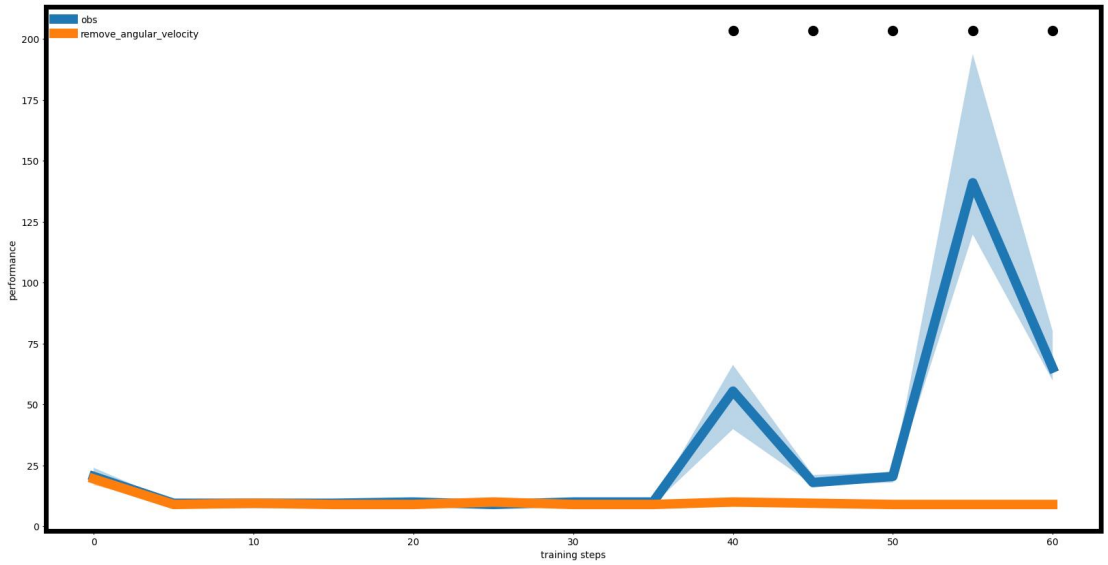
Additionally, we plotted a comparison graph of the reward curves for the four scenarios together. From this, we can see that when only angular velocity is removed, the DDPG algorithm is still able to find an effective deterministic strategy. However, when velocity is removed, it becomes difficult to achieve a strategy that yields a high reward. This suggests that for the DDPG algorithm in the CartPoleContinuous-v1 environment, velocity is a more important feature than angular velocity.

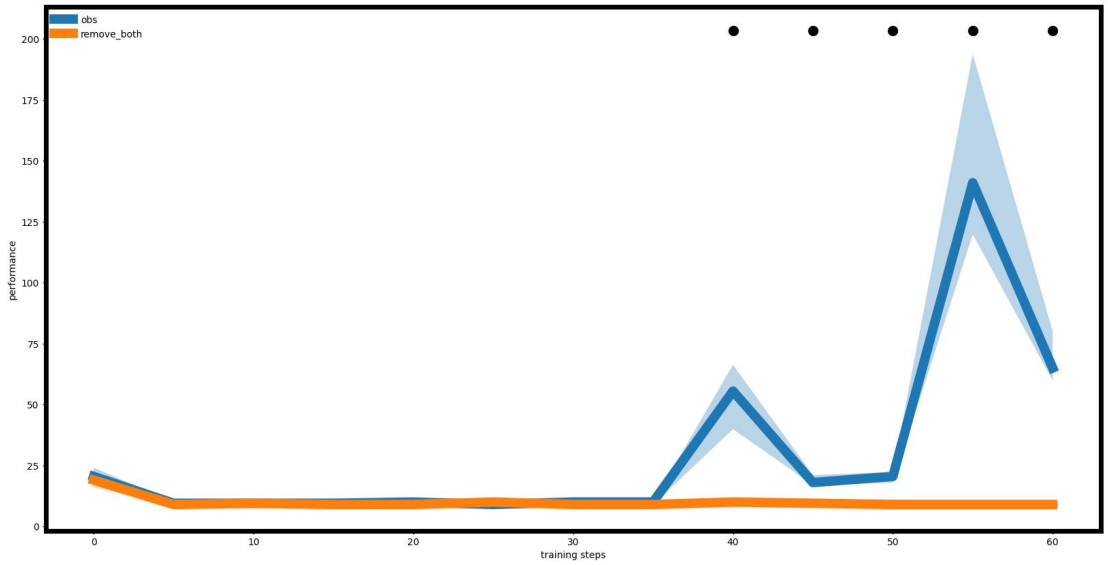
ObsTimeExtension

In the second experiment, we use the ObsTimeExtensionWrapper to equip the agent with a memory of the past. By observing the environment using the ObsTimeExtensionWrapper, we study how a memory of previous states compensate for partial observability.

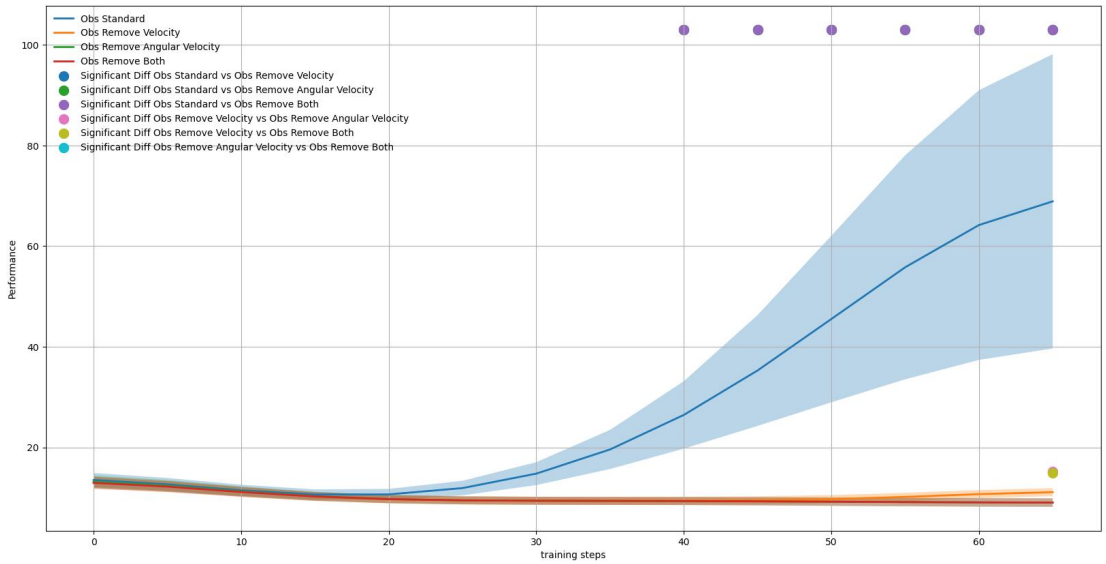
We also used the Welch T-Test to compare the performance.


obs
remove_velocity
performance
training steps
0
25
50
75
100
125
150
175
200
0
10
20
30
40
50
60


obs
remove_angular_velocity
performance
training steps
0
25
50
75
100
125
150
175
200
0
10
20
30
40
50
60


obs
remove_both
performance
training steps
0
25
50
75
100
125
150
175
200
0
10
20
30
40
50
60

In the second experiment, we equipped the agent with a list-like memory of past observations and extended the critic and policy networks to take both the current observation and previous ones as inputs. From the comparison graphs, we can see that in the standard scenario, the algorithm's performance improved significantly, achieving a deterministic policy with higher rewards. However, in partially observable environments, the results were not as expected. We initially hypothesized that the memory of past observations would compensate for the missing derivative features, thus improving performance in partially observable cases. However, in the three partially observable scenarios, the algorithm failed to train a strategy capable of achieving high rewards. We speculate that the compensation effect of the memory of past observations was not as effective, and due to the randomness in the training process, even after 1500 training episodes, the memory of past observations was insufficient for the DDPG algorithm to reliably produce an effective strategy. But from the markers of Welch's test, we can still derive that there is significant difference between the learning curves of velocity removal and of both features removal.

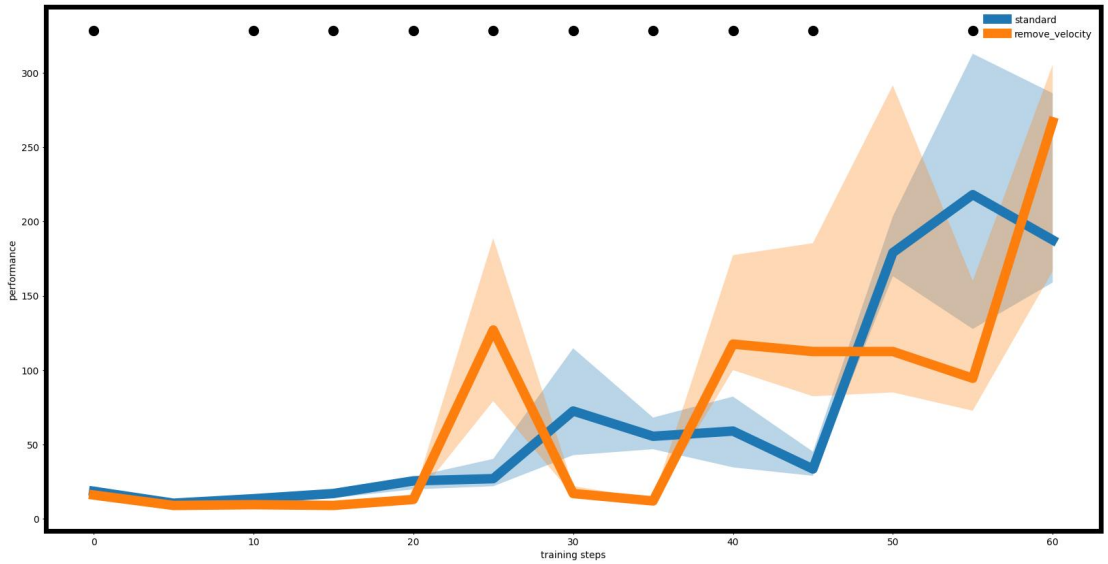


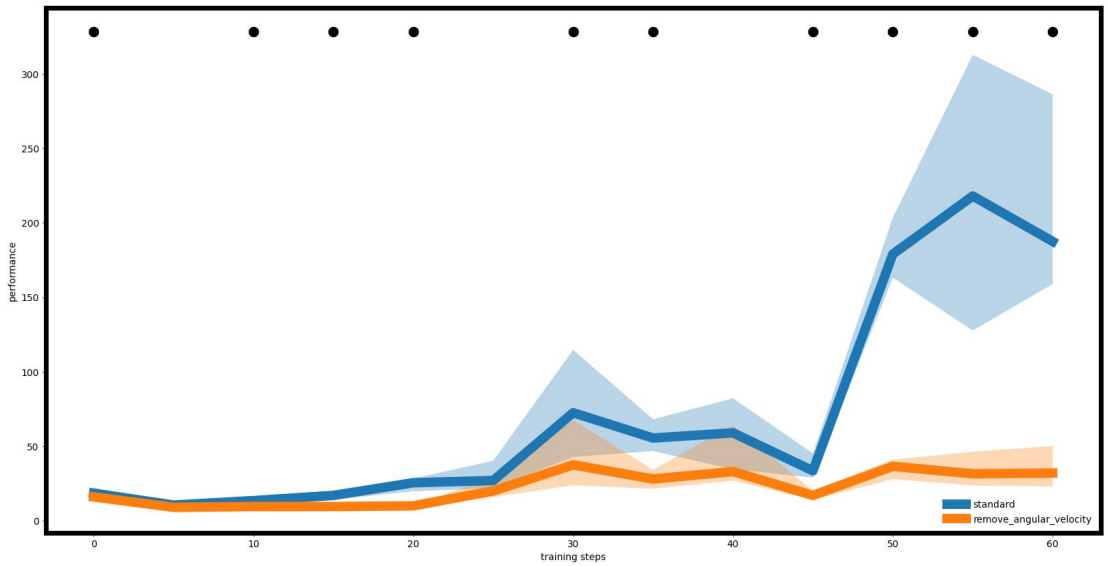
From the overall comparison graph of the four scenarios, we can also see that the DDPG algorithm failed to achieve effective strategies in the three partially observable environments. Its performance even worsened in the scenario without angular velocity. We believe this may be due to the addition of the memory of past observations, which likely slowed down the convergence speed of the DDPG algorithm, preventing it from finding a good strategy within 1500 episodes.

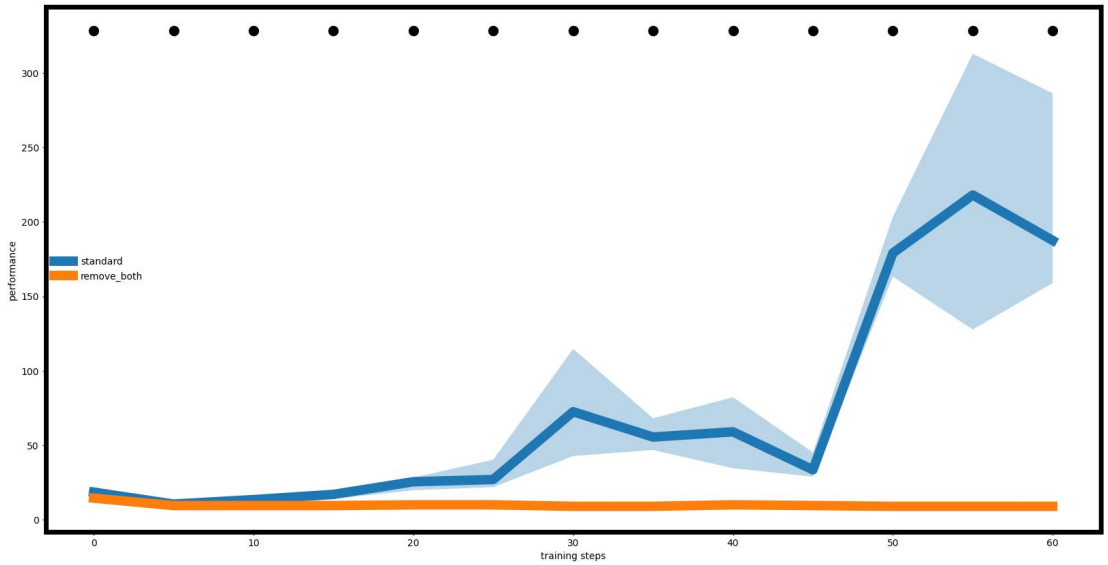
ActionTimeExtension (M = 2)

In the third experiment, we use the ActionTimeExtensionWrapper to extend the action space of the enviroment to M = 2 times. By observing the environment using the ActionTimeExtension -Wrapper, we study the impact of extending the action space to partial observability.

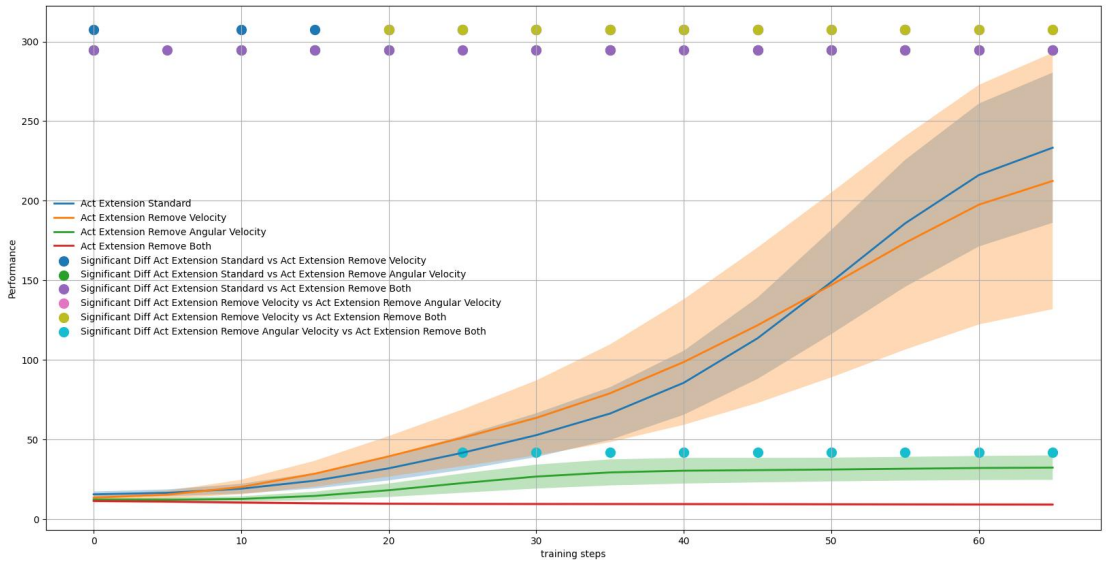
We still use the Welch T-Test to compare the performance.







In the third experiment, we expand the action space to be twice its original size (M = 2) and selected the first action to execute. Under fully observable conditions, the DDPG algorithm achieved significant improvements, and the deterministic policy obtained through action space expansion outperformed the previous ones in terms of reward. Additionally, in the scenario without angular velocity, the DDPG algorithm shows substantial improvement; although it still performs slightly below the fully observable scenario, it achieves a notable enhancement compared to before. In the scenario without velocity, the improvement is less pronounced, but the algorithm still manages to obtain a deterministic policy. However, in the scenario without both velocity and angular velocity, the DDPG algorithm shows little improvement.



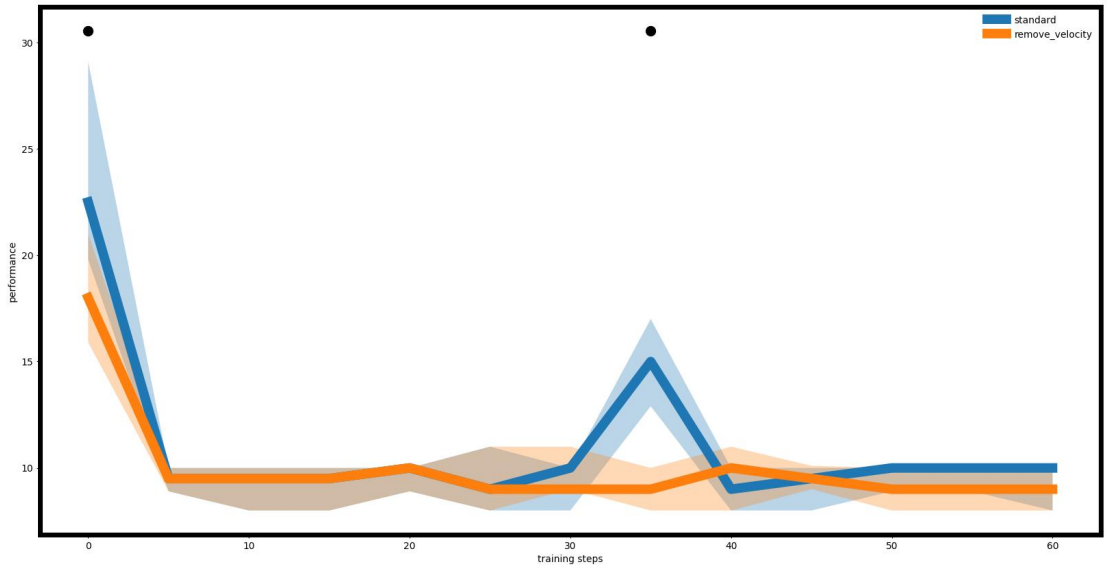
From the comparison graph of the four scenarios, we can see that the case without angular velocity experienced a greater improvement than the case without velocity. We believe that by expanding the action space to be twice its original size (M = 2) in the CartPoleContinuous-v1

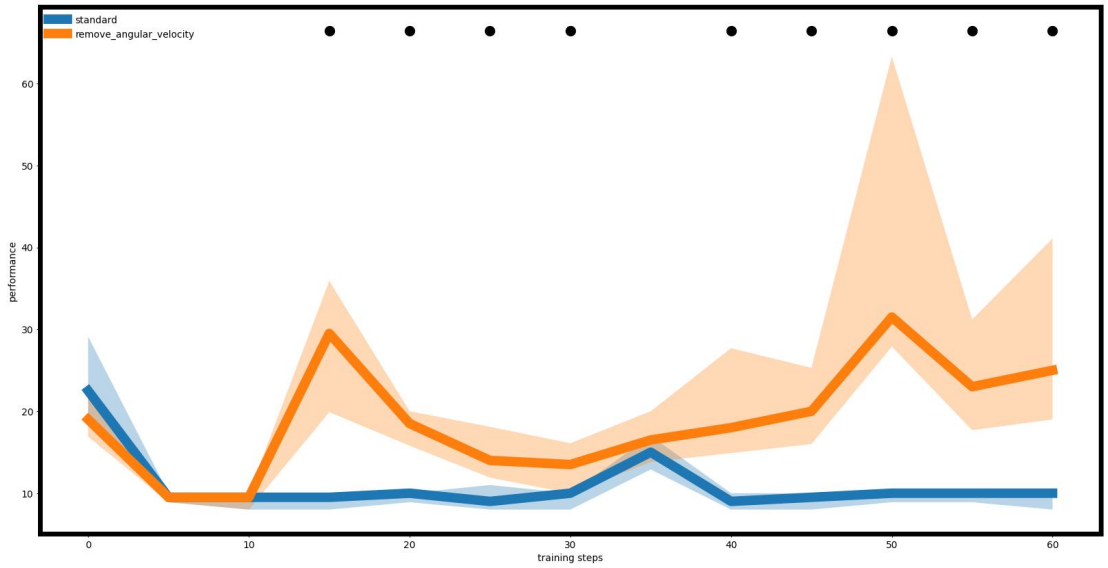
environment, the angular velocity feature provides more information than the velocity feature, and the compensatory effect is significant.

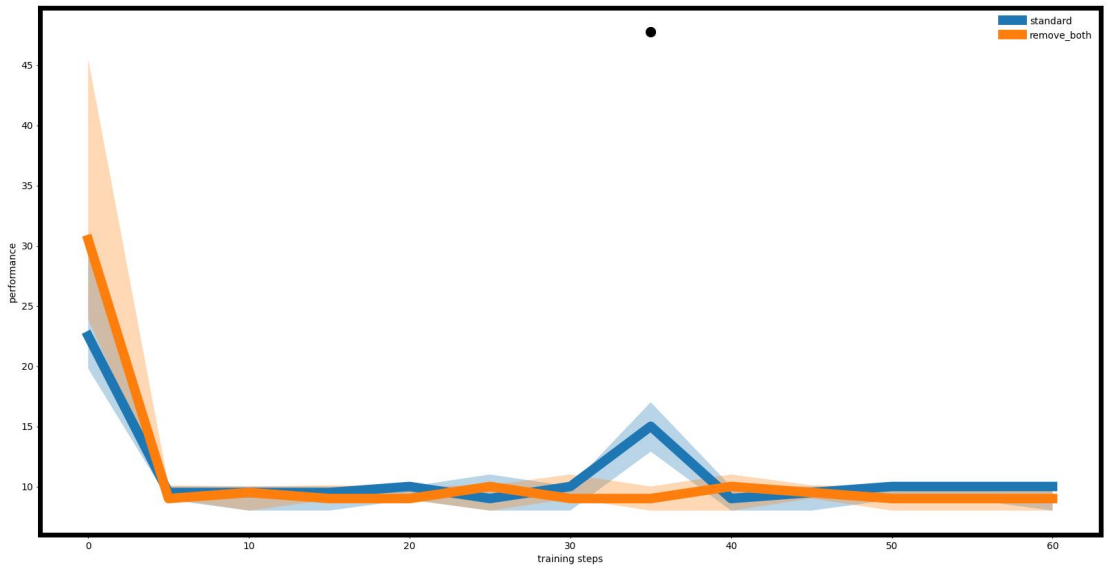
Bonus: ActionTimeExtension (M = 3)

In the fourth experiment, we use the ActionTimeExtensionWrapper to extend the action space of the enviroment to M = 3 times. By observing this enviroment, we study the differences between M = 2 and M = 3.

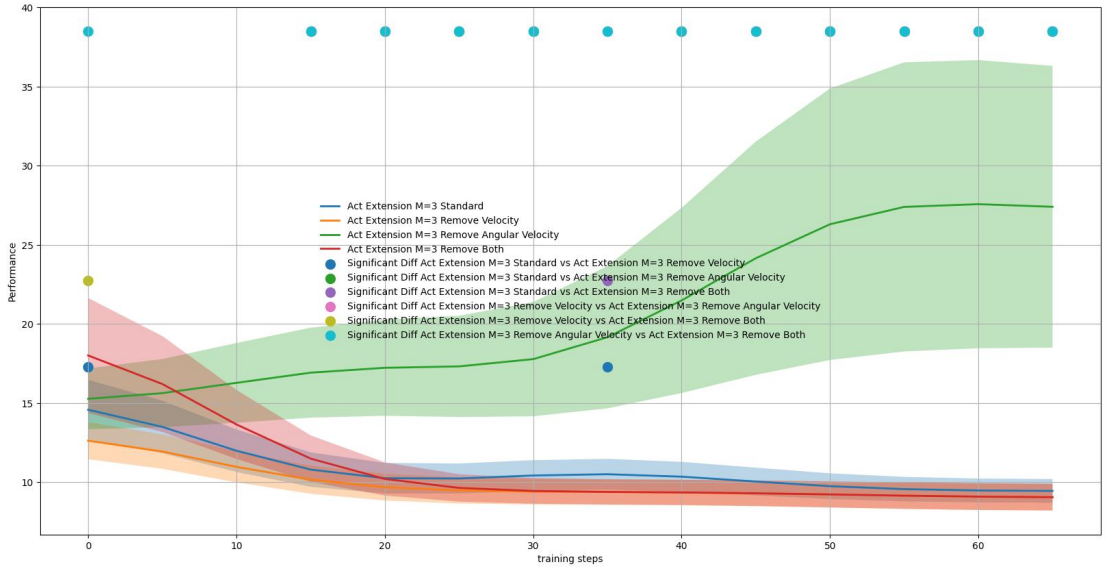
Similarly, we first perform the Welch T-Test to compare the performance.







With the action space expanded to three times its original size (M = 3), the results are less favorable. First, in the fully observable environment, the algorithm is unable to learn an effective deterministic policy, indicating that such an action space expansion actually reduces the performance of the DDPG algorithm. In the partially observable environment without velocity, a similar result was observed, as no effective deterministic policy was learned. However, in the partially observable scenario without angular velocity, the algorithm achieved a result similar to the M = 2 case, making its performance slightly better than the other three environments. In the case where both speed and angular velocity were removed, performance still did not improve.



In the comparison graph of the four curves, the performance in the environment without angular velocity is similar to that of the third experiment, while the other environments performed poorly. We speculate that expanding the action space too much (M > 2) may render the information provided by the angular velocity feature unstable, making it more challenging to obtain a good

strategy. In contrast, the velocity feature is not significantly affected, which is why the environment without angular velocity shows similar performance.

Summary

In the four experiments, we reached the following conclusions:

1. In the fully observable environment, equipping the agent with memory of past observations and expanding the action space (M = 2) both improved the performance of the DDPG algorithm in the CartPoleContinuous-v1 environment. However, expanding the action space further (M = 3) led to a decline in performance.

2. In our experiments, equipping the agent with memory of past observations did not achieve the expected improvement in partially observable environments. We believe this may be due to the compensatory effect not being as pronounced or potential issues with our ObsTimeExtension.

3. In the experiments with expanded action spaces (M = 2 and M = 3), we observed significant changes in the environment without velocity. We think that expanding the action space primarily affects the angular velocity feature, significantly increasing the useful information provided by the angular velocity feature in the M = 2 case. However, in the M = 3 case, the information from the angular velocity feature appears to reduce the algorithm's performance. Furthermore, expanding the action space has less of an impact on the velocity feature, but it does not lower performance due to increased action space.