Master 2 - Données, Apprentissage, Connaissances

Reinforcement Learning:
SuperTuxKart piloting project

RLD - Master DAC - Sorbonne University - S1-24

Done by:
KEBIR Ahmed Rayane
AZIZI Walid

16th February 2025

Contents

1 Introduction

In this work, we explore reinforcement learning (RL) and behavioral cloning (BHC) techniques to train autonomous agents for the `SuperTuxKart` racing environment. By leveraging a combination of pre-existing and custom-designed wrappers, we optimize agent training through structured observation preprocessing and targeted environment modifications. We examine the impact of normalization on behavioral cloning and evaluate how pretraining with expert demonstrations improves RL performance. Our experiments compare the efficiency of different RL algorithms, emphasizing the benefits of initializing policy networks with behavioral cloning weights.

2 SuperTuxKart environment

2.1 Description

`SuperTuxKart` is a free and open-source cross-platform kart racing game, originally forked from TuxKart in 2006, featuring open-source mascots and actively developed by the community. The `PySuperTuxKart2` Gymnasium wrapper is a Python package that provides RL environments for SuperTuxKart, allowing for flexible agent control, various observation and action spaces, and different difficulty levels. It supports multiple environment variations, including simplified, flattened, and multi-agent setups, with optional wrappers to modify observations (e.g., polar coordinates) and actions (e.g., discrete or continuous controls).

2.2 Wrappers

2.2.1 Pre-existing Wrappers

- **ConstantSizedObservations** ensures that the number of observed items, karts, and paths remains fixed at a predefined value (default: 5 each).
- **PolarObservations** converts all 3D Cartesian coordinates in the environment into polar coordinates (horizontal angle, vertical angle, and distance).
- **DiscreteActionsWrapper** discretizes the acceleration and steering actions into a limited number of possible values (default: 5 acceleration steps, 7 steering steps).
- **FlattenerWrapper** merges all continuous and discrete observation/action spaces into a unified format, creating a structured dictionary with ‘discrete‘ and ‘continuous‘ entries.
- **FlattenMultiDiscreteActions** converts a multi-discrete action space into a single discrete space by assigning a unique action index to every combination of choices.

2.2.2 Implemented Wrappers

We implement the following wrappers:

- **StuckStopWrapper** is a wrapper that monitors recent rewards and terminates the episode if the agent remains ”stuck” for a defined number of timesteps (n). Specifically, if all rewards over the last n steps are equal to -0.1, the environment is truncated, forcing the episode to end. The idea is that given (n) timesteps, if the agent remains stuck, we truncate the environement to speed up training by giving it another chance instead of being stuck until the default environment truncation at around 1500 steps.
- **PreprocessObservationWrapper** is a wrapper that preprocesses observations by flattening mixed observation spaces (both continuous and discrete) into a single vector. Discrete variables are encoded using one-hot encoding. It also optionally applies normalization using precomputed mean and standard deviation values from a trained agent.
- **SkipFirstNStepsWrapper** is a wrapper which skips the first n steps of the environment by performing random actions.

3 Behavioral Cloning (BHC)

We perform behavioral cloning on the `flattened multi-discrete` environment:

- This environment is obtained by applying the following wrappers: ConstantSizedObservations (we experimented with size 5 and 10), PolarObservations, DiscreteActionsWrapper, FlattenerWrapper and finally PreprocessObservationWrapper. By fixing the constant size to 5 and 10, we obtain an observation space of size 154 and 264 repsectively. We denote them num5 and num10.
- We generate and fill a replay buffer using expert trajectories. We save for each observation its corresponding action and reward. We set the difficulty as 2 to simulate trajectories against the most difficult opponents. We generate trajectories for all tracks, with varying number of karts [2-15]. We set the number of laps as 2-3, to get varying scenarios of the kart crossing the finish line area. For each trajectory, we ignore the first 20-30 steps, which we noticed impact the start of the race, where the kart would learn to stay in place instead of accelerating after the green light. This loss of information is compensated thanks to the higher number of laps, which would help generate observations around the finish line area with the kart sprinting forward.
- Given 2 million and 1.5 million pairs of (observation, actions) for the environments num5 and num10 respectively, we perform a classification for each discrete action, using cross-entropy loss. The global loss is the sum of the losses of each separate classification. The number of classifications is 7, with action 6 being steering.

- We experiment with varying network architectures. We found that bigger networks result in better generalization and stability, but with slower training. We found a good compromise to be three hidden layers of size 1024 and Tanh activation. The last layer is split according to the number of options per action, and softmax is applied to each part to generate a probability distribution for the action.
- We experiment with applying normalisation of the observations using the mean and standard deviation computed from the millions of samples in the datasets, which are split into training and test subsets with a ratio of 80%/20%.
- We employ advantage normalization which stabilizes training by ensuring that advantage values have zero mean and unit variance, preventing high-variance updates and leading to smoother, more efficient policy learning.

Figure 1 illustrate the impact of observation normalization on behavioral cloning performance, which is measured by the evolution of the accuracy of the predicted actions on the test subset.

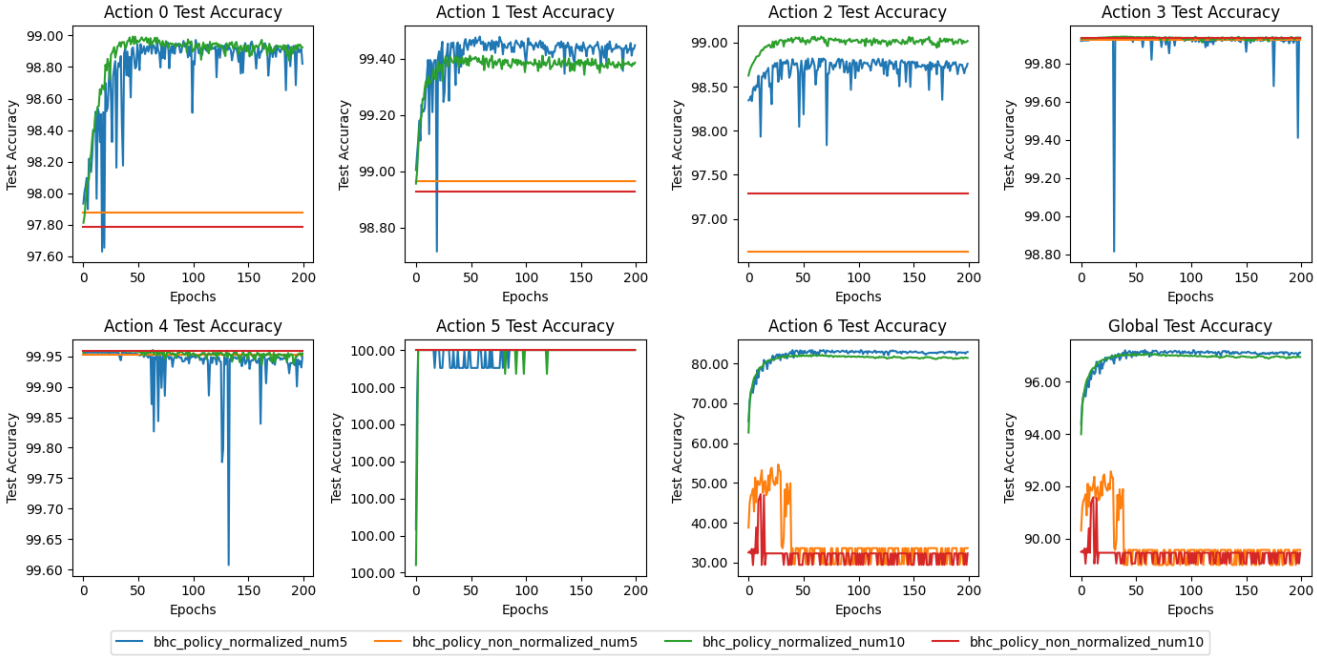

Figure 1: Training results of behavioral cloning with and without observation normalization.
The plot shows test accuracy per action category across training epochs. Normalized policies (blue and green) achieve significantly higher accuracy than non-normalized ones (red and orange), especially for complex actions like steering (action 6). This indicates that normalization improves generalization and stability during training.

The accuracy of most actions is significantly higher for the normalized policies (blue and green) compared to the non-normalized policies (red and orange). Actions 0–5 show strong performance, with normalized policies achieving test accuracies close to 99%. In contrast, non-normalized policies remain significantly lower. Action 6 (steering) exhibits a stark contrast, where normalized policies achieve a much higher accuracy compared to non-normalized ones. The non-normalized models struggle, with performance fluctuating at low levels. Global test

accuracy follows the same pattern: normalization leads to better performance, suggesting that normalized inputs help the model generalize better.

Figure 2 shows the evaluation of different BHC policies in terms of Cumulative Rewards over the number of steps. Normalized policies (blue and green) achieve significantly higher cumulative rewards than non-normalized ones (red and brown), reinforcing the advantage of preprocessing observations. The non-normalized models (red and brown) struggle to accumulate positive rewards, indicating that they fail to generalize well to real driving scenarios. The standard deviation (shaded regions) is larger for normalized models, particularly in the early steps. This suggests that while they achieve higher performance, they also exhibit more variability in their behavior.

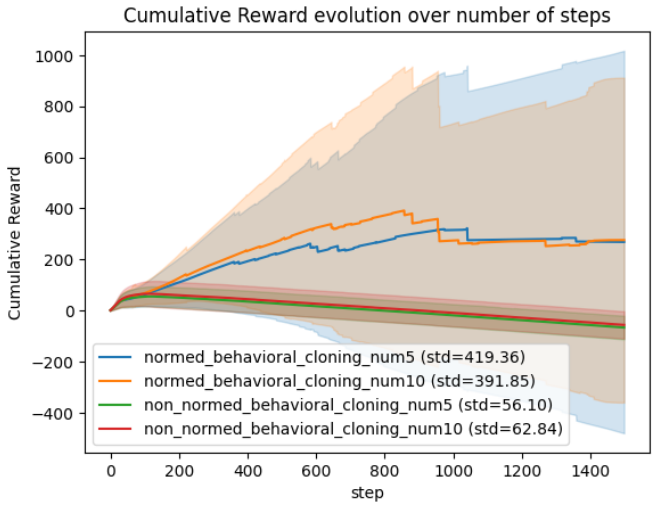

Figure 2: Evaluation results of behavioral cloning with and without observation normalization. *The plot displays cumulative reward per step, with shaded areas representing standard deviation. Normalized policies (blue and green) achieve higher cumulative rewards, demonstrating better generalization to real driving scenarios. In contrast, non-normalized policies (red and brown) struggle to accumulate rewards, reinforcing the importance of preprocessing observations for stable learning.*

Key Results. Observation normalization significantly improves both training accuracy and driving performance. The improvement is especially critical for complex actions like steering, which require fine-tuned decision-making. Larger observation sizes (num10) provide additional benefits, but the differences between num5 and num10 appear minor compared to the effect of normalization itself. The behavioral cloning model effectively learns expert behavior when trained on properly preprocessed data, reinforcing the importance of feature engineering in imitation learning tasks.

Side Note. We experimented with Imitation Learning algorithms *Generative Adversarial Imitation Learning (GAIL)* and *Adversarial Inverse Reinforcement Learning (AIRL)* but we didn't get interesting results like simple behavioral cloning which quickly learns a good policy without needing to rollout the model.

4 Reinforcement Learning

We experiment with PPO and A2C to train a smart agent for PySuperTuxKart.

- We notice that there is a need to augment the number of steps beyond the default parameters, since the environments take a long time to finish, especially if the agent performs poorly, usually reaching the default truncation after 1500 time steps.
- To speed up the training, we use the implemented **StuckStopWrapper** to truncate the environment earlier. We notice that for a smaller number of steps, i.e. from 8 to 64 and a bit above, the agent behaves weirdly by stopping in the middle of the race if an action would lead to being stuck somewhere. From 128 steps and above, this behavior is gone. Thus, we set the number of steps to 128.
- Following the observation in BHC, we employ the **SkipFirstNStepsWrapper** to skip the first 20 steps, which impact the learning because whatever action is performed, the kart is stuck in place awaiting the green light to start the race.
- We set 21 parallel environments, each one working on a unique track. This allows our agent to focus on all tracks equally, rather than relying on a random order that might be either biased, or makes the RL algorithm privilege tracks on which the agent is doing well rather than tracks on which it performs poorly. Difficuly is set to 2 for strong opponents.
- To improve training efficiency, we initialize the RL policy network with the weights of a pretrained behavioral cloning model. This provides a strong beginning, allowing the agent to learn more effectively and reach higher rewards compared to training from scratch.

Figure 3 compares the cumulative reward of different agents trained with A2C and PPO, with and without initialization by behavioral cloning.

The blue curve (normed_a2c_num5_best) and orange curve (normed_ppo_num5_best) represent an agent initialized with behavioral cloning and fine-tuned with A2C and PPO respectively, achieving the highest cumulative reward with significant improvement over time, though with some variability across runs. The green curve (normed_behavioral_cloning_num5) shows an agent trained purely with behavioral cloning, which starts off stronger than A2C without initialization but quickly plateaus, indicating its limitations. The red (normed_a2c_num5_no_init) and purple (normed_ppo_num5_no_init) curves represent an agent trained with A2C and PPO respectively from scratch, which struggle to improve and remains at a consistently lower reward level, highlighting the benefit of behavioral cloning initialization.

Key Results. Behavioral cloning provides a useful initialization that speeds up learning and results in better overall performance when fine-tuned with A2C. A2C without initialization struggles to achieve the same level of performance. Behavioral cloning alone is not sufficient to reach optimal performance but serves as a helpful pre-training phase.

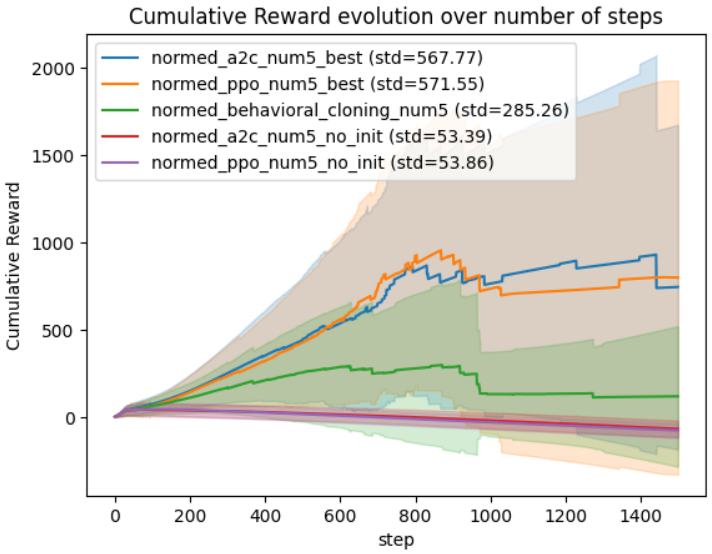

Figure 3: Comparison of reinforcement learning (RL) results with and without behavioral cloning initialization. *The plot shows cumulative reward per step with standard deviation to illustrate variability. The blue (A2C with initialization) and orange (PPO with initialization) curves achieve the highest rewards. The green curve (pure behavioral cloning) initially performs well but plateaus, while the red (A2C from scratch) and purple (PPO from scratch) curves struggle to improve, emphasizing the importance of initialization.*

5 Leaderboard

Table 1 presents the final performance of different trained agents, ranked by their average finishing position and average reward. The best-performing agent, A2C+Normalization+Size5+InitBHC, achieves an average race position of 2.09 with a standard deviation of 0.92, indicating strong and consistent performance. It also attains the highest average reward of 14.53, confirming the advantage of behavioral cloning initialization.

agent_name	avg_position	position_std	avg_reward	reward_std
AC2+Normalization+Size5+InitBHC	**2.095238**	0.920909	**14.535377**	5.521040
PPO+Normalization+Size5+InitBHC	2.190476	1.219643	14.498195	5.682283
BHC+Normalization+Size10	3.285714	1.749636	12.881247	6.777753
BHC+Normalization+Size5	3.714286	1.419016	10.420558	7.314071
AC2+Normalization+Size5+NoInit	5.904762	1.230747	0.696442	0.477738
BHC+Size5	6.285714	1.277753	0.548363	0.485289
PPO+Normalization+Size5+NoInit	6.285714	1.277753	0.546978	0.493031
BHC+Size10	6.380952	1.396465	0.519989	0.526661

Table 1: Performance comparison of the agents, ranked by their average finishing position and average reward. *The best-performing agent achieves an average position of 2.09 with a reward of 14.53, highlighting the impact of behavioral cloning initialization. Purely BHC agents plateau in performance. Agents trained without normalization or pretraining perform significantly worse, underlining the importance of these techniques for generalization and stability.*

The second-best model, PPO+Normalization+Size5+InitBHC, follows closely with an average position of 2.19 and a cumulative reward of 14.49. This suggests that both PPO and

A2C benefit significantly from behavioral cloning pretraining, though A2C appears to provide slightly better stability and reward optimization.

Agents trained purely with behavioral cloning perform relatively well but plateau in performance. The *num10* variant achieves an average position of 3.29, outperforming the **num5** variant (3.71), indicating that larger observation spaces may provide additional benefits. However, behavioral cloning alone is insufficient for optimal performance, as it lacks long-term strategy adaptation.

Interestingly, BHC agents without normalization show a significant drop in performance. The *BHC+Size10 (No Normalization)* and *BHC+Size5 (No Normalization)* variants achieve lower rewards and higher average positions, reinforcing the importance of normalization in improving generalization and stability.

In contrast, RL agents trained from scratch without behavioral cloning perform significantly worse. They finish in average positions of 5.90 and beyond, with very low rewards. This reinforces the idea that reinforcement learning alone struggles to learn an effective driving policy from the ground up, emphasizing the importance of expert demonstration initialization and proper normalization techniques.

6 Conclusion

Our experiments demonstrate that combining behavioral cloning with reinforcement learning leads to the best-performing agents. Behavioral cloning provides a strong initial policy, allowing A2C and PPO to fine-tune and achieve superior results compared to training from scratch. Additionally, observation normalization significantly enhances model performance, particularly in complex actions such as steering. The results also suggest that larger observation spaces (num10) may provide slight advantages, but initialization plays a much more crucial role. Future work could explore alternative pretraining strategies, advanced reward shaping, or multi-agent training to further improve performance.

Key Learnings

- **Normalization is critical:** Applying normalization to observations significantly improves both behavioral cloning and reinforcement learning performance, especially for complex actions like steering.
- **Behavioral Cloning as Pretraining:** Initializing reinforcement learning agents with weights from a pretrained behavioral cloning model accelerates training and results in higher cumulative rewards.
- **Environment Wrappers Improve Training Efficiency:** Custom wrappers such as `StuckStopWrapper` and `SkipFirstNStepsWrapper` help speed up training by avoiding inefficient states and truncating unproductive episodes.

- **Parallelized Training Enhances Generalization:** Training across multiple tracks in parallel prevents agents from overfitting to specific tracks and ensures robust generalization across different racing conditions.
- **Algorithm Selection Matters:** PPO and A2C exhibit different learning behaviors, with A2C performing better in our setup when combined with behavioral cloning pretraining.
- **Larger Networks Improve Stability:** Increasing the network size enhances generalization and stability, though at the cost of slower training.
- **Imitation Learning Limitations:** While GAIL and AIRL are promising approaches, simple supervised behavioral cloning proved to be more effective for this task.