Escaping Failure States without Experts in FPS Behavioral Cloning

***Abstract*—Behavioral cloning (BC) for first-person shooter (FPS) games has shown promising results in contexts of limited information. However, BC models are no exception to issues with compounding errors, stochasticity, and temporal dependencies commonly observed in other applications like autonomous driving. In this paper, we build upon foundational previous work to address failure states in fine-tuning. Through preliminary analysis of existing behavioral cloning models in Counter-Strike: Global Offensive (CS:GO), we find four general categories of failure states. Then, we propose a data aggregation method for adjusting the agent from a failure state, collecting the recovery action data, and fine-tuning the behavioral cloning model.**

***Index Terms*—behavioral cloning, first-person shooter, game AI, data analysis**

I. Introduction

Recent work in behavioral cloning (BC) for first-person shooter (FPS) games have shown promising results in games without convenient APIs or large-scale simulation, such as Counter-Strike: Global Offensive [1]. Behavioral cloning is a technique where an agent learns an action policy from expert demonstration in a supervised fashion, whereas an agent learns in deep reinforcement learning (DRL) through free exploration of a well-defined environment with a well-defined reward function. DRL is also more commonly used in artificial intelligence (AI) for games that are easily simulated, examples including DeepBlue Chess [2] and DOTA [3]. However, the application of vision-based behavioral cloning from first-person perspectives faces several challenges compared to their easily simulated counterparts and is much less studied in the context of limited information.

Specifically, behavior cloning suffers from three aspects: 1) compounding errors, where errors accumulate over a trajectory and lead the agent into a state unrepresented in the training data, 2) stochastic expert actions, where multiple "correct" expert actions exist and the averaged expert action is not a valid action, and 3) Non-Markovian observations, where observations depend on multiple previous states (e.g. a trajectory) rather than only the current state [4], [5]. While these limitations are well-explored in applications such as autonomous driving, it is a relatively new technique for tactical multi-agent game AI, where agents operate under a competitive zero-sum objective.

In this paper, we explore the data limitations of behavioral cloning methods on FPS games in contrast to other applications such as autonomous driving, and propose a simple method for failure state recovery based on mini-map and player localization cues. A visual summary of our method can be found in Figure 2.

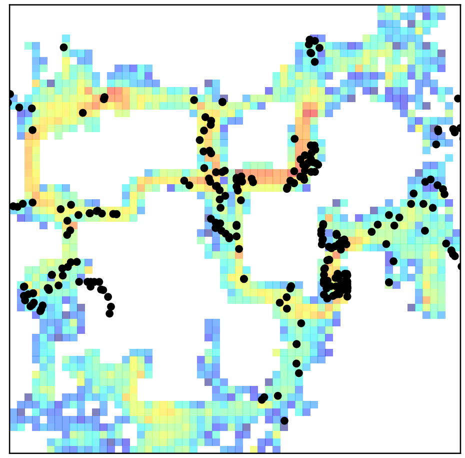


Fig. 1: Visualization of encountered failure states over a 10 minute time frame. Here, we plot the positions of failure states (black) in relation to the density of map coverage from the fine-tuned DM agent of [1]. Failure states often occurred in sparse areas adjacent to popular areas.

II. Background

A. Behavioral Cloning for FPS Games

Behavioral cloning (BC) is a common and well-explored method for replicating human behavior in applications other than video games, such as autonomous driving and robotics [6]–[8]. This method extends naturally to video games since video games can be considered simulation-only and noiseless robotics problems [9], [10]. Recently, Pearce and Zhu provide foundational work in behavioral cloning [1] for first-person shooter games, where large-scale simulation is difficult for successful deep reinforcement learning, which is what most state-of-the-art game artificial intelligence (AI) methods employ [11]–[13]. This foundational work is the first to employ behavioral cloning in first-person shooter settings, where action and observation spaces are high-dimensional.

Behavioral cloning is a form of "imitation learning" where an agent learns to mimic the action of a demonstrator, typically a human expert. This technique can be applied to both FPS games and autonomous robotics, albeit with a few differences. Video games have a much lower risk factor than autonomous driving, so incomplete datasets are acceptable and environment exploration is cost-efficient. FPS game agents can also incorporate tuning into systems since they operate in a more restricted environment than autonomous vehicles do.

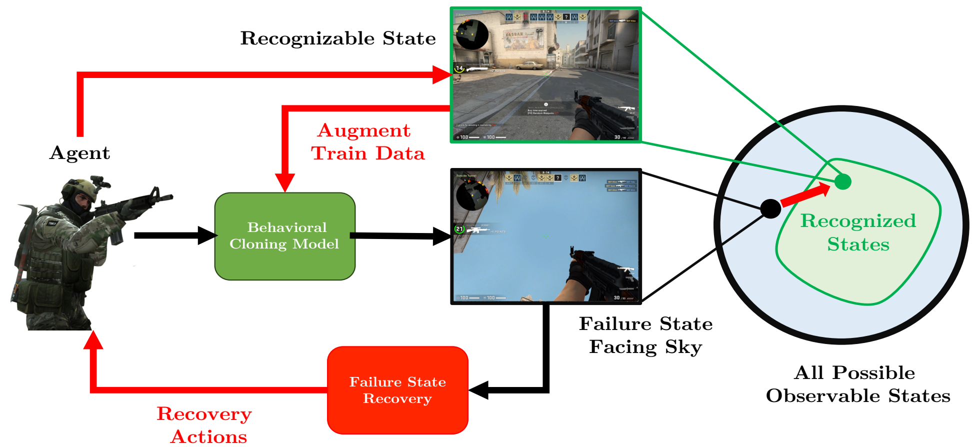


Fig. 2: Overview of our method. Our method, denoted by red arrows, depicts the failure state recovery routine for a baseline behavioral cloning model in green. The result of our routine is thus nudging the agent's observed state into a recognizable distribution, then recording the recovery actions for future fine-tuning.

For convenience, we reiterate the objective of behavioral cloning from [1]. Given offline observation data pairs $\mathcal{D} = \{\{a_1, o_1\}, ...\{a_N, o_N\}\}$ collected from an expert control policy $\pi^*(a|o)$ representing a probability distribution of actions $a \in \mathbb{R}^M$ for M degrees of freedom, the objective of behavioral cloning is to learn a policy $\pi_\theta(a|o)$, parameterized by θ, which optimizes the following objective:

$$\theta = \underset{\theta}{\text{argmin}} \sum_{i}^{N} L(a_i, \pi_\theta(\hat{a}|o))$$

where $L : A \times A \to \mathbb{R}$ represents a loss function.

III. Methodology

A. Failure State Recovery

We define a failure state as a situation where the player policy is unable to move in any degree of freedom for more than five seconds. We set this threshold of five seconds in the context of a competitive tactical shooter format, since opponents will be other players, and movement in the game, in general, should be constant.

In general, trained player policies will fall into a failure state due to compounding errors commonly present in imitation learning problems for robotics. A common behavior bias found in datasets collected from human users is that the user almost never faces the wall while also being close to the wall; in other words, the expert player which the imitation learning model learns from is almost always facing away from the walls of map corridors.

A popular technique addressing compounding errors is Data Aggregation (DAgger) [14], [15]. Intuitively, DAgger address out-of-distribution observation states by incorporating new offline data in each iteration from the expert policy π^*. As the student policy π experiences new states, the expert generates the corresponding "correct" action and adds it to the data pool for policy π to learn from.

Implementing this for FPS games, however, is difficult due to the lack of an existing expert policy. While there are large amounts of human-generated data possible, coupling new observations with expert actions requires a human to be present during DAgger iterations, making the naive application infeasible. Alternatively, we propose a modified technique for handling failure states without the need for an expert policy. This method is further outlined in Algorithm 1.

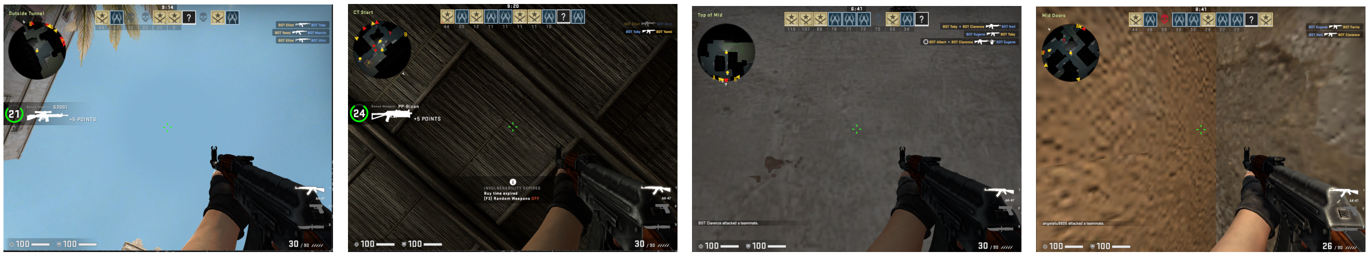
Algorithm 1: Modified DAgger for Behavioral Cloning

Result: Trained policy $\hat{\pi}^*$
Dataset $\mathcal{D} \leftarrow \varnothing$;
$\hat{\pi}_0 \leftarrow$ Random Initialization;
$o \leftarrow$ Initial observation;
for $n \leftarrow 1...N$ **do**
    $\pi_i \leftarrow Recovery(\pi_i)$; `// Modify candidate`
    $D_i = \{(s, \pi_i(s)\}$; `// Sample trajectories`
    $D \leftarrow D \cup D_i$; `// Combine datasets`
    Train $\hat{\pi}_{i+1}$;
end

We found in existing behavioral cloning models that failure states fell into three categories: overshot up, overshot down, and facing walls. Occurrences occurred the most in areas of the game map closest to edges. Examples of failure states are pictured in Figure 3.

To address these challenges, we use RGB pixel values on specific parts of the game window and used this information to make slight adjustments. This is feasible since video games are noise-less simulation environments. For example, we can expect that map textures and HUD locations and colors will stay constant over multiple iterations of a game.

We conduct checks for vertical adjustment scenarios by focusing on the values closest to the center of the screen, where the crosshair is. If this value happens to be close to the RGB value for blue, we iteratively nudge the mouse down until the color changes shade. We make the assumption that sky colors are the closest to the RGB value $(0, 0, 1)$ than any



(a) Facing sky (b) Facing ceiling (c) Facing wall (d) Facing corner

Fig. 3: Four examples of failure states.

Algorithm 2: Recovery Policy

Data: Candidate policy $\hat{\pi}$, Horizontal turning interval θ_Z, Vertical turning interval θ_Y, Max turning angle β

Result: Modified Policy $\hat{\pi}_R$ and chosen action a

$a \leftarrow \hat{\pi}(o)$;

$a \leftarrow clip(a, -\beta, \beta)$;

$s \leftarrow Environment(a)$;

if *Player facing wall in* s_t **then**

  $a \leftarrow \theta_Z + a_{RotX}$;

end

if *Player facing sky* **then**

  $a \leftarrow -\theta_Y + a_{RotY}$

end

$s \leftarrow Environment(a)$

texture on the map. Note that this does not handle areas with ceilings; we leave this to future work, as it is nontrivial to implement without a public API.

We use a similar method for horizontal adjustments, except now focusing on the minimap on the top left of the screen. The minimap is a representation of the player's local 2-dimensional position, with the player always at the origin and facing forward. The minimap rotates along with the player, allowing us to check for objects or walls in front of the player without having to perceive a complex 3D scene. Flat, black values in front of the player on the minimap are indicative of the player facing a wall. The agent's rotation is nudged again in small intervals until the pixel value on the minimap changes.

IV. Results

We show results comparing the properties of the behavioral cloning model with the vanilla routine versus our modified recovery routine. In terms of hardware, all results were produced on Intel Xeon Gold 5218 CPUs (2 cores), 64 GB of memory, and an NVIDIA RTX 2080 Ti graphics card. Game settings are replicated from [1].

Overall, we find that our routine diversifies the horizontal and vertical mouse movement labels compared to the baseline dataset. A visualization comparing the two distributions can be found in Figures 4 and 5.

To evaluate how our recovery policy may influence the performance of the baseline agent directly, we also quantify the difference in deathmatch metrics. Deathmatch is a free-for-all game mode in CS:GO where every other player is an enemy of the agent. The goal is to get as many eliminations as possible. The metrics used are Kills Per Minute (KPM) and Kill-Death Ratio (KD). Detailed comparisons for this can be found in Table I. While kill per minute and kill death ratio are worse than the original stats for each category, our routine performs better consistently for both values compared to an untouched routine that did not implement any tuning. We also quantify the occurrence of different failure state scenarios in Table II for each training model. Over a five-minute session, we find that horizontal turning happens less frequently with the best model, ak47_sub_55k_drop_d4_dmexpert_28, and that all models do not engage in vertical turning. We also find that most horizontal turning involves scenarios facing narrow corridors and sharp turns. As we see in Figure 1, most of the failure states are concentrated in sparsely explored areas directly adjacent to areas with high map coverage.

TABLE I: Evaluation comparison for Kills-Per-Minute (KPM) and Kill-Death Ratio (K/D) between the baseline behavioral cloning model and our method. Without any additional training, our method enhances the performance of the baseline model by up to 10 times.

	—— Easy ——		—— Medium ——	
Model	KPM↑	K/D↑	KPM↑	K/D↑
Baseline [1]	0.29	0.40	0.03	0.05
Ours + Baseline [1]	**0.33**	**0.44**	**0.44**	**0.32**
Expert Baselines				
Built-in Bot (easy)	2.11	1.00	–	–
Built-in Bot (medium)	–	–	2.41	1.00
Human (Non-gamer)	4.25	1.80	2.38	0.90
Human (Casual gamer)	4.20	4.20	3.51	2.48
Human (Strong CSGO player)	14.00	11.67	7.80	4.33

TABLE II: Average rate of failure states per minute for baseline fine-tuned behavioral cloning model gameplay.

Model	Wall	Sky
ak47_sub_55k_drop_d4	30	0
ak47_sub_55k_drop_d4_dmexpert_28	32	0
ak47_sub_55k_drop_d4_aimexpertv2_60	41.6	0
July_remoterun7_g9_4k_n32_recipe_ton96__e14	39	0

V. Conclusion

In this paper, we present a simple method for failure state recovery in behavioral cloning for CS:GO without the need

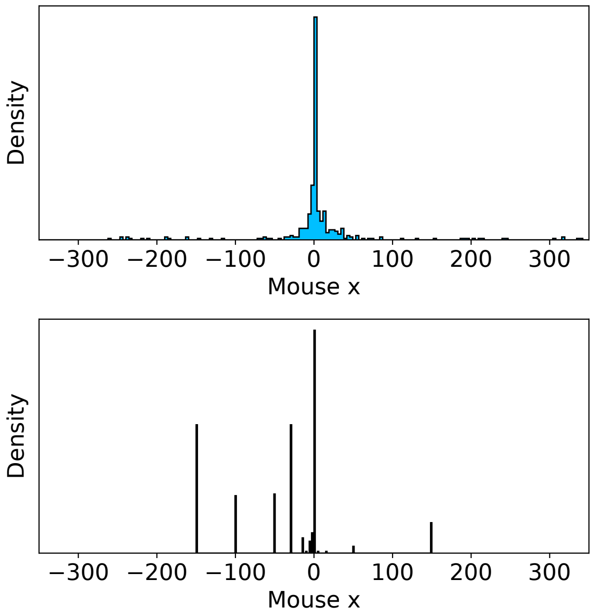


Fig. 4: Horizontal movement distribution comparison between baseline data (top) and 400 minutes of our augmented failure state recovery data (bottom).

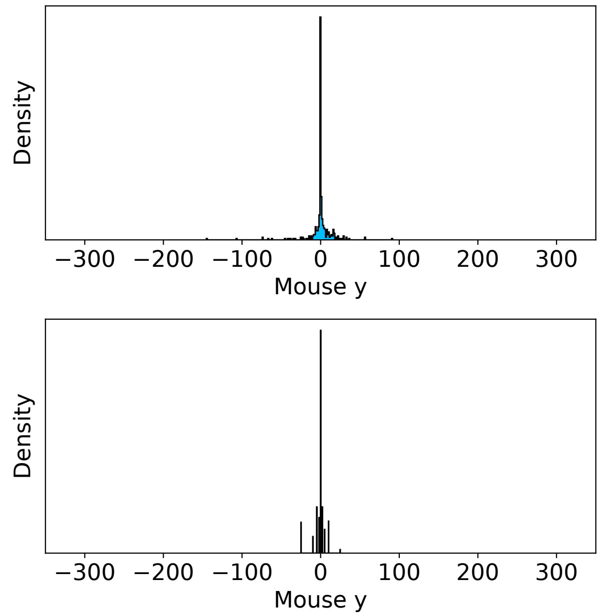


Fig. 5: Vertical movement distribution comparison between baseline (top) and 400 minutes of our augmented failure state recovery data (bottom).

for an expert policy. We find through our analysis of baselines that failure states can be easily inferred through pixel checks on the noiseless environment, and implement recovery actions accordingly to augment the training dataset for fine-tuning.

Our work has several limitations. Firstly, our method is not reflective of human-like recovery and only addresses entirely expert-free scenarios. Future work can take advantage of both limited expert recovery data and our expert-free recovery actions to produce realistic behavior. Additionally, our results are limited in terms of fine-tuning results. In future work, we seek to evaluate our augmented recovery dataset on a fine-tuned FPS behavioral cloning model and provide additional insight into its value to high-dimensional deep learning policies. Additionally, we recognize that hardware differences between our setup and baseline setups can significantly influence baseline model performance; results for baselines are re-evaluated on our setup for this reason.

References

[1] T. Pearce and J. Zhu, "Counter-strike deathmatch with large-scale behavioural cloning," in *2022 IEEE Conference on Games (CoG)*, pp. 104–111, 2022.

[2] M. Campbell, A. J. Hoane Jr, and F.-h. Hsu, "Deep blue," *Artificial intelligence*, vol. 134, no. 1-2, pp. 57–83, 2002.

[3] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

[4] K. Fragkiadaki, "Deep reinforcement learning and control: Imitation learning." https://www.andrew.cmu.edu/course/10-403/slides/S19$_{l}ecture2_{b}ehaviorcloning.pdf$, 2019. $Accessed : 2023–05 - 17$.

[5] J. A. Bagnell, "An invitation to imitation," tech. rep., Carnegie-Mellon Univ Pittsburgh Pa Robotics Inst, 2015.

[6] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[7] A. Tampuu, T. Matiisen, M. Semikin, D. Fishman, and N. Muhammad, "A survey of end-to-end driving: Architectures and training methods," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 4, pp. 1364–1384, 2020.

[8] J. Xu, H. Guo, and S. Wu, "Indoor multi-sensory self-supervised autonomous mobile robotic navigation," in *2018 IEEE international conference on industrial internet (ICII)*, pp. 119–128, IEEE, 2018.

[9] Y. Ying, "Mastering first-person shooter game with imitation learning," in *2022 2nd International Conference on Networking, Communications and Information Technology (NetCIT)*, pp. 587–591, 2022.

[10] C. Thurau, C. Bauckhage, and G. Sagerer, "Imitation learning at all levels of game-ai," in *Proceedings of the international conference on computer games, artificial intelligence, design and education*, vol. 5, 2004.

[11] OpenAI, :, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," 2019.

[12] Z. Lin, J. Li, J. Shi, D. Ye, Q. Fu, and W. Yang, "Juewu-mc: Playing minecraft with sample-efficient hierarchical reinforcement learning," in *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22* (L. D. Raedt, ed.), pp. 3257–3263, International Joint Conferences on Artificial Intelligence Organization, 7 2022. Main Track.

[13] Y. Gao, B. Shi, X. Du, L. Wang, G. Chen, Z. Lian, F. Qiu, G. Han, W. Wang, D. Ye, *et al.*, "Learning diverse policies in moba games via macro-goals," *Advances in Neural Information Processing Systems*, vol. 34, pp. 16171–16182, 2021.

[14] W. Davies and P. Edwards, "Dagger: A new approach to combining multiple models learned from disjoint subsets," *machine Learning*, vol. 2000, pp. 1–16, 2000.

[15] M. Kelly, C. Sidrane, K. Driggs-Campbell, and M. J. Kochenderfer, "Hg-dagger: Interactive imitation learning with human experts," 2019.