Behavior Cloning with DJI Drone

Alexander Seto, Noah Shamus, Jun Sung Tak

1 Introduction

Autonomous unmanned aerial vehicles (UAVs) can provide value often in times where the task is difficult for humans, ranging from emergency situations where only a drone can get to those in need to overhead surveillance, reaching high viewpoints that are not practical without an aerial vehicle, optimizing their flight/actions with statistical methods. Often, hard coded implementations for UAVs require a relatively high computational power as a series of complex calculations must be computed at rapid speeds. Thus, it is practical to skip the calculations and implement a reinforcement learning framework to lower the computational restraint. Currently, there are multitude of techniques in implementing reinforcement learning frameworks that allows the drone to track an object. The goal of this paper is to explore such frameworks and to identify the one able to deal with small amounts of data yet complex tasks: thus we turn to imitation learning. The general idea of imitation learning will be to have an agent learn an expert policy of operating given a certain state in its environment. Generally, imitation learning techniques aim to mimic human behavior in a given task. This is of great interest because often learning a complex task can be accelerated by making use of insight about the said task. Imitation learning instead observes another agent performing task itself and the agent is programmed to imitate an expert performing the task. As a result, if a complex task can be done by a human, it can also be imitated by an agent, even without extensive insight into the complex details of the task. In this paper, a technique called behavior cloning will be employed in an attempt to allow a drone to imitate an expert policy which tracks a human face.

2 Background Related Work

UAVs have a plethora of practical use cases, from delivery to remote places [3] to inspection of hard to reach locations in power plants [1]. The preference of UAVs to vehicles carrying people is straightforward: cost, safety, and size can all be reasons to choose a drone.

The autonomous aspect has become increasingly relevant as growth in artificial intelligence's scalability has allowed deployment of frameworks which possess the ability to both remove human error and reduce cost from having someone control the vehicle remotely. Reinforcement learning in particular has

Figure 1: Quad-rotor in AirSim environment

become a relevant solution as the environment setup is straightforward and the amount of uncertainty and skill required to make decisions when controlling a drone can be addressed through function approximation based upon sensors or cameras on the vehicle.

Outside of UAVs, the field of reinforcement learning has made strides in recent years with respect to human aided learning. In [2], the authors present a method called Advise, which integrates human feedback into the reward, attempting to maximize the effect of feedback, regardless of how scarce or inconsistent it is. In this paper, they shape the policy of an agent playing Pacman and Frogger using advice from a human at each potential position in the game. [6] also implemented a method where humans assist reinforcement learning frameworks where instead of directly altering the algorithm/reward, they engineer multiple tasks, specifically ones where learning the earlier tasks helps the convergence rate/general performance of learning the later tasks. They show that learning processes can be improved through specific, well thought out engineering of tasks where the environment is simple in some dimension, showing the RoboSoccer and Pacman learning agents can be assisted which such curriculum learning. [4] presents the TAMER framework, a general learning framework which integrates human trainers, who can guide a learning agent through observational feedback using a sort of clicker style reward, using statistical methods to propagate reward from the human to the proper time steps of experience. A particularly compelling framework is called Apprenticeship Learning, where a human controls the agent for a given amount of time, and using this experience given by the human, effectively limiting the search space for approximating the function, the agent can learn the proper reward function or policy.

Finally, DJI Tello is a light-weight UAV produced by DJI. It is controllable via a remote controller, the Tello app, and a python script utilizing Tello SDK. Furthermore it is equipped with a HD camera that allows the drone to transmit digital images to a source. In [7] the author utilizes a single shot multi-box detector in combination of an OCR algorithm to achieve tracking with the drone's on board camera. Coupled with this, we will be implementing apprentice learning to our agent to introduce an aspect of human feedback within the learning process. The learning will consist of our agent learning how to best track a given object by adjusting its velocity and position utilizing Tello SDK.

Behavioral cloning as one of the most basic methods of imitation learning, where an agent essential tries to clone the policy of an expert. The earliest instance of behavior cloning in the context of autonomous vehicles can be seen from work done by [8]. In this paper, a equipped with various sensors learned to map the sensor inputs to different actions, such as steering angles and acceleration, using behavioral cloning. This was one of the first instances of autonomous vehicles. The inputs were made into a feature vector. The generated feature vector was then fed into a neural network to make action decisions.Behavioral cloning is a simple yet very powerful tool with respect to trying to get an agent to learn a complex task.

To our knowledge, there are no existing implementations of a light weight behavioral cloning framework with object tracking drones. Although no formal evaluation metric is shown in this report, the qualitative results provide evidence of the robustness of the framework in UAVs.

3 Methodology

In behavioral cloning frameworks, there exists some optimal policy, π^*, which we would like the agent to imitate. For such optimal policy π^* we also have a set of trajectories, D, defined as a sequence of state-action pairs representing the states the expert saw and the actions they took. This can be seen in Figure 2 where τ represents a single trajectory. Thus, the goal for an agent in behavioral cloning is to find a parameterized policy, π_Θ given parameters Θ which matches the policy of the expert to the best of its ability. In order to do this, we treat each state action pair in each τ in our data set D as independently, identically distributed observations in a supervised learning problem to optimize a model such that the state is the input for the policy and an action is output. [5]

This can be seen in Figure 3, where it shows how we find π_Θ, optimizing Θ with L as the loss function between the action chosen in the optimal policy and the output of the parameterized policy of our imitation agent. The loss for optimizing Θ is one which attempts to match output of the model for a given state as input with the output of the expert policy for the given state. In a discrete action setting, this could be a classification problem, entailing a cross entropy loss but in the continuous action setting, like the one is this project, a mean squared error loss can be used. Once the policy is optimized on the entire data set, we deploy our agent by having it run the parameterized policy model's

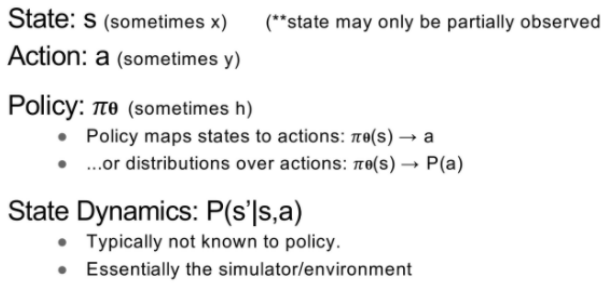

Figure 2: Definition of terms used throughout project to describe the behavioral cloning algorithm. [5]

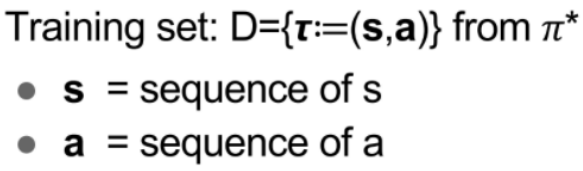

Figure 3: Definition of terms used to describe the dataset in the behavioral cloning algorithm. [5]

inference in a real world setting [5]. The full algorithm for Behavioral Cloning can be seen in Figure 4.

Thus, given the environmental constraint of using a DJI Tello drone, we set out to find an implementation of a hard coded task that we would attempt to imitate as an expert. There were many existing tasks implemented for the drone with the DJI Tello Python sdk and OpenCV pre-trained detection models. We decided to choose a face tracking drone implementation, specifically the implementation from youngsoul's tello-sandbox repo on Github [10]. His implementation essentially worked like this: the python sdk would initialize drone takeoff, allowing the drone to hover still for a few seconds. Then, the feed from the camera on the DJI tello drone was fed into numpy arrays where one of OpenCV's face detection models extracts the bounding box coordinates for a face. Based on math including the coordinates of the detected face and the time passed between face detection, the program would direct the drone in the corresponding direction such that the face was brought closer to the center of the frame. Thus, the drone would follow a face respectably, despite the s processed per second being a measly 5. The math the drone uses specifically is two proportional-integral-derivative controllers (PID controllers) which essentially are controllers where they continuously track a process, receive feedback from the environment and attempt to minimize the difference between an op-

$$\text{argmin}_\theta \, E_{(s,a^*)\sim P^*} L(a^*, \pi_\theta(s))$$

Figure 4: Definition of goal of the loss in the behavioral cloning algorithm. [5]

1. Collect demonstrations (τ^* trajectories) from expert
2. Treat the demonstrations as i.i.d. state-action pairs: $(s_0^*, a_0^*), (s_1^*, a_1^*), \ldots$
3. Learn π_θ policy using supervised learning by minimizing the loss function $L(a^*, \pi_\theta(s))$

Figure 5: Definition of the behavioral cloning algorithm, using terms defined in writing and previous figures. [5]

timal point and a process variable using this error feedback. Essentially, the controller uses a weighted sum of the error value, the integral of the error across time and the derivative of the error with respect to time in order to produce an updated variable for the monitored process. The logic behind this is that each term represents the present, past and future of the error value, respectively. A diagram of a PID controller can be seen in Figure 6 [9].

In our (the expert's) case, the process variables for the two controllers are the x and y coordinate of the face and the set point is the x and y coordinate of the center of the frame, respectively. The drone hence uses a weighted sum of the integral, derivative and value of the error between the face coordinates and center of frame coordinates to produce a single update value for both the left and right and up and down directions.

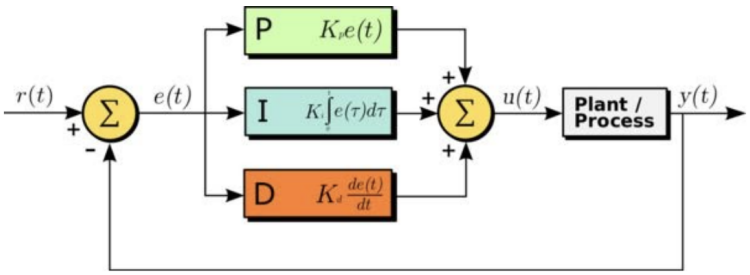

Figure 6: Diagram explaining the process of PID controllers, note how the weighted sum from the integral, derivative and error itself is used to update the process variable., [9]

This task was pretty straight forward to map as a behavioral cloning problem. Each time a PID controller received feedback it sent a drone control to change the drone velocity with respect to the left/right movements and the up/down movements. Thus it is straightforward to encode the action for a

given time step as a tuple of float values, representing each velocity change in the command sent by the PID controller. The feedback it received from the environment each decision step was an integer x coordinate for the face, an integer y coordinate for the face, and a float value representing the distance of the face from the last time it saw a face (labelled as d). The value d was only used by the program to decide whether or not to make an action, so it could be encoded as a boolean (when it was negative 1 or over 25 it didn't run an action, to prevent jitter by the drone). Thus, the state for this problem could be represented as an array with the feedback the controller received except d as a boolean value.

Thus, we went on to collect data, done very easily by inserting code right before an action was sent in the script that ran the implementation such that we recorded the state action pairs to form a behavioral cloning data set. Surprisingly, a very small amount of data was needed but details about size of data set will be reserved for the next section.

As for choosing the parameterized model to use as the policy for our drone, we decided to select linear regression models for each of the left/right and up/down commands because we felt the relationship between the input and the actions was simple enough to be modelled by a simple linear model. The variable used in each respective model was the corresponding coordinate times the boolean d representation.

After fitting the models, since the drone received the face coordinates in integer form, we were able to save our results to a python dictionary querying with the discrete states. Thus, at run time, our script would be nearly identical to the current implementation except instead of running a PID controller to get an action, each time it queried our action dictionary with the current state array serialized into a string. It should also be noted that any action with magnitude greater than 40 was clipped such that 40 was the max possible action value in either negative or positive directions.

Thus, our overall procedure was as such: (1) run the tracking script to collect trajectory csvs, each entry holding the information about the state and action of the given time step; (2) collect the entries into a single data frame and fit linear regression models for each of the left/right and up/down actions using states as input; (3) place the predictions for each possible state (finite number since all coordinate pairs with in the frame) in an action dictionary; (4) at drone run time, when face tracking begins, query the dictionary instead of running the PID controller. As for evaluation, we were not able to formulate an unbiased, valid quantitative metric to measure the drone's performance with each policy because it would be nearly impossible to collect consistent, constant trials (consistent lighting, face movement, etc.) with the resources we have. Thus, we resort to qualitative evaluation but this lack of ability to evaluate can be solved with inverse reinforcement learning or a simulator, to be discussed in the proceeding section.

4 Experimental Results/Technical Demonstration

Upon the beginning of experimentation, there were initial concerns about how much data would actually need to be collected to produce a valid policy using linear models. However, these concerns were quick to dissolve as after each time we recorded a trajectory, we attempted to fit models and iteratively checked how much data was necessary. In the end, only two trajectories were needed before the imitation learner could successfully complete the task. Through two trajectories, both under a minute, we collected 4850 state-action pairs. An example of a piece of a trajectory can be seen in Figure 7. objX and objY represent the coordinates of the face while lr_command and ud_command represent the action directions sent by the controller. d represents the distance between the last face coordinates to the current ones. When no face could be found, d was -1 but note that the last face's coordinates remain the same as they are saved to calculate the next potential value for d.

	objX	objY	lr_command	ud_command	d
70	195	124	0.000000	0.000000	-1.000000
71	195	124	0.000000	0.000000	-1.000000
72	195	124	0.000000	0.000000	-1.000000
73	195	124	0.000000	0.000000	-1.000000
74	190	138	-7.755610	9.306729	14.866069
75	190	138	-7.001346	8.401615	0.000000
76	192	136	-4.453138	10.950209	2.828427

Figure 7: Example of a portion of time steps for a given trajectory.

Of these 4850 time steps, only 2562 represented time steps the PID controller sent a velocity change to the drone. This is because of shakiness and/or not being able to locate a face. This also caused some concern as the data could potentially be skewed to certain areas of the frame based on how good the face detection model is. Nevertheless, the actions were enough to fit the models well enough to run the task, and a summary table of the time steps which were exclusively actions is shown below in Figure 8. As can be seen, the state space is limited to integers as well as having finite limits, allowing for discretization at inference. One can also see the extreme action values having magnitude 40 as this is the maximum speed.

It should be noted that before the coordinate data was used in regression, the data was centered and normalized to values 0 and 1 for (1) general performance boost but primarily (2) to be able to actually get negative outputs from

	objX	objY	lr_command	ud_command	d
count	2562.000000	2562.000000	2562.000000	2562.000000	2562.000000
mean	197.663934	150.171351	1.672228	-2.779283	0.417777
std	58.281854	34.245706	22.623763	14.933408	19.378645
min	42.000000	28.000000	-40.000000	-40.000000	-40.000000
25%	168.000000	129.000000	-15.312676	-7.924566	0.000000
50%	197.000000	152.000000	0.777762	1.000000	1.414214
75%	219.000000	172.000000	17.018564	2.236068	4.975221
max	372.000000	242.000000	40.000000	40.000000	40.000000

Figure 8: Summary table for all time steps where an action is sent by the PID controller.

actions (because if coordinate is always positive, the linear model will only be able to capture an action of a single sign as we use a single coefficient and no intercept). The linear regression models fit within this paper were implemented with sklearn's API, specifically the sklearn.linear_model.LinearRegression class. No intercept was fit for either model as conceptually any time a centered coordinate was equal to 0, the center, the drone should not do anything. The models were fit by minimizing the residual sum of squares and evaluated using the mean squared error between their outputs and the true values using the training set. No validation set was used since we understood overfitting as important to imitating the expert. Thus, the results can be seen in Figure 9 where a baseline MSE is calculated by taking the MSE between the mean of the dependent variable's value for the data set and the actual values. It can be seen each model significantly outperforms the baseline so we initially could say that they definitely capture some sort of relevant relationship for capturing the policy.

	Baseline MSE	Model MSE	Model Formula
x model	0.896006	0.072861	L/R Action = 5.877 * X_coord
y model	2.085661	0.390506	U/D Action = -3.75 * Y_coord

Figure 9: Table describing results from the linear regression models for behavioral cloning.

The model formulas are shown as well in Figure 9, where it should be noted that left and up directions were represented by positive valued actions and the right and down directions were represented by negative values. As discussed in

the previous section, these models were then run for each possible combination of face coordinates and saved in a python dictionary for inference. At inference, there was no good way to come up with a metric to evaluate the difference between the two policies for two reasons: (1) the first being that evaluating the task was dependent on having consistent trials, meaning each policy is put through the same exact conditions, nearly impossible without a simulator given our resources; (2) metrics which measured time or data consumption were heavily biased by the connectivity of the drone, which in turn was affected by its connection strength, which is hard to keep consistent across trials as well. Thus, other than the clear boost in time gained from running a python dictionary policy instead of a PID controller implemented in python, we resort to qualitative observation between the two policies in practice, described in the next section. Videos showing the test runs, and thus the data which qualitative evaluation is made is contained in the videos directory with the submitted code.

Videos of different trajectories are available,

1. Expert trajectory 1: https://youtu.be/GiiVuxKIUsg
2. Expert trajectory 2: https://youtu.be/hz_2azphcK0
3. Agent trajectory 1: https://youtu.be/HF6dQd93MKc

5 Discussion

As discussed above, the discussion section is reserved for the understanding of qualitative results as well as commentary of potential limitations of our methodology.

The two most notable results from experimentation other than straight up flight performance were inference time and the amount of data it took to train our learner. Usually, a reinforcement learning agent would need thousands of episodes in order to understand a task while our agent does it in only two! Furthermore, upon discretizing the states and querying actions from a dictionary, the policy inference time (essentially the reaction time) was much smaller (observationally) than in the hard coded expert policy.

It is highly suggested to view the videos linked above before reading this section. The way to differentiate these videos is best seen in the amount of time needed to calculate where the drone needs to move. In the expert trajectories, the drone constantly needs to recalculate its position relative to the face and center of screen. We see a dramatic improvement with the agent that implements an imitation learning method in time needed to readjust its position relative to the face. This is due largely to using a lookup table with a learned policy. All of this can be evidenced by the relative smoothness of the flight of the drone compared to the expert trajectories.

To further elaborate, when the expert drone detects that the face is not in its frame it evidently pauses for a split moment, calculate in which direction and magnitude it should move and then it makes that move. The resulting trajectory

seems rather clunky and slow. This effect is especially evident when the face being tracked changes position in a drastically matter. The phenomenon is circumvented with the IL agent trajectory. The flight seems to be smoother and make faster decisions; however, another problem is introduced. In the expert trajectory the drone will stop moving when the object is in frame. Thus, if the object is stationary the drone will hover in place. In the agent's trajectory, it tends to overshoot its movements. As an example, when the drone moves to the right to place the face in frame it moves too much and slowly continue moving even when the face is in frame. This causes the object to now be on the left side of the frame and the drone adjusts as such. Even when adjusting, the drone will overshoot and the same behavior is evident. This causes the drone to oscillate in a small margin between the frame.

Despite our successful implementation, there are various limitations that must be mentioned. First, due to limitations of the equipment, various crucial flight data was inaccessible. As a result determining metrics to evaluate with was rather tricky. Furthermore, having access to velocity data would have been beneficial because it can serve as additional state information, adding more dimensionality and better representation o our state space. As the PID controller in the drone utilizes difference of positions over difference in time, with potential velocity information an imitation learner can utilize functions that can account for the drone's overshooting of movement.

Another limitation is the size of our data set. Only 2 trials worth of data was used to train the agent in this setting. While the performance of the policy is evident through both video and statistics, increasing the number of trials will definitely lead to more stable and consistent results. Continuing, the simplicity of the current model drastically limits certain precise maneuvers, for example there is a horizontal oscillation when the object is in the frame.

Continuing, a general limitation of the behavior cloning tends to be related to the MDP assumption. As the previous action in a state induces the next state, it collides with behavior cloning's independently identically distributed (often referred to as the IID) assumption. The effect of this is that error produced in each state accumulates and could cause the supposed optimal policy to be ineffective. Furthermore, a learned policy cannot be improved upon due to the nature of the algorithm.

6 Conclusion and Future work

In conclusion, our implementation exhibited overall success. It was able to learn the optimal policy from the expert and then further showed qualitative improvement in performance during flight. The drone appeared to travel in a significantly smoother trajectory as well. Obviously, using models more complex than linear models with more information about the state of the drone could have helped the policy to learn more accurately, but nevertheless, results from our experiments were definitely respectable.

Further work in this subject also seems incredibly promising. With direct

policy learning there could potentially be improvements in various aspects. As it operates in a loop of expert feedback/demonstrations, agent rolling policy out in environment, there is a larger quantity of training data. As the agent iteratively approaches to an optimal policy with each loop, it is able to remember non-optimal actions and avoid them.

Lastly, another promising direction is inverse reinforcement learning. The main idea of this approach is to estimate a reward function based on the demonstrations of the expert. It initially assumes that the first trial of demonstrations exhibits optimal policy. It then rolls out that policy and then is compared to the expert's policy. Finally it updates the reward function based on the comparison of the two policies. This process is repeated until the found policy meets a certain threshold. This method is the most comprehensive while also requiring the largest computational and data resources. One drawback of this method is that it requires precise data regarding the environment the agent is working in. While this is could be done with relative ease in a grid world environment it is drastically more difficult in material settings.

References

[1] Jayme Garcia Arnal Barbedo. A review on the use of unmanned aerial vehicles and imaging sensors for monitoring and assessing plant stresses. *Drones*, 3(2), 2019.

[2] Shane Griffith, Kaushik Subramanian, Jonathan Scholz, Charles L Isbell, and Andrea L Thomaz. Policy shaping: Integrating human feedback with reinforcement learning. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 26. Curran Associates, Inc., 2013.

[3] Ghozali Hadi, Rivaldy Varianto, Bambang Trilaksono, and Agus Budiyono. Autonomous uav system development for payload dropping mission. volume 1, pages 72–77, 01 2014.

[4] W. Bradley Knox and Peter Stone. Interactively shaping agents via human reinforcement: The tamer framework. In *The Fifth International Conference on Knowledge Capture*, September 2009.

[5] Hoang Le, Nan Jiang, Alekh Agarwal, Miroslav Dudik, Yisong Yue, and Hal Daumé, III. Imitation learning tutorial. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 2917–2926. PMLR, 10–15 Jul 2018.

[6] Sanmit Narvekar, Jivko Sinapov, Matteo Leonetti, and Peter Stone. Source task creation for curriculum learning. In *Proceedings of the 15th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2016)*, Singapore, May 2016.

[7] Luiz Gustavo Miranda Pinto, Félix Mora-Camino, Pedro Lucas de Brito, Alexandre C. Brandão Ramos, and Hildebrando F. Castro Filho. A ssd – ocr approach for real-time active car tracking on quadrotors. In Shahram Latifi, editor, *16th International Conference on Information Technology-New Generations (ITNG 2019)*, pages 471–476, Cham, 2019. Springer International Publishing.

[8] Dean A Pomerleau. Alvinn: An autonomous land vehicle in a neural network. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ARTIFICIAL INTELLIGENCE AND PSYCHOLOGY . . . , 1989.

[9] Adrian Rosebrock. Pan/tilt face tracking with a raspberry pi and opencv, Apr 2021.

[10] Patrick Ryan. Youngsoul/tello-sandbox: Sandbox of scripts and tests programming the tello drone in python.