Prediction of Human Behaviour using Imitation Learning

by

Matthaios Zidianakis

MTN2008

Submitted
in partial fulfilment of the requirements for the degree of

Master of Artificial Intelligence

at the

UNIVERSITY OF PIRAEUS

September 2022

Author 

Matthaios Zidianakis

II-MSc “Artificial Intelligence”

September 15, 2022

Certified by.....

George Vouros

Professor, University of Piraeus

Thesis Supervisor

Certified by.....

Maria Dagioglou

Researcher, NCSR Demokritos

Member of Examination Committee

Certified by.....

George Petasis

Researcher, NCSR Demokritos

Member of Examination Committee

Prediction of Human Behaviour using Imitation Learning

By

Matthaios Zidianakis

MTN2008

Submitted to the II-MSc "Artificial Intelligence" on September 15, 2022, in partial fulfilment of the
requirements for the MSc degree

Summary

This thesis explores the use of the Info-GAIL algorithm, which is based on the generative adversarial imitation learning framework to model modalities of human behaviour towards performing tasks. The goal of this thesis is to use behaviour models learnt through Info-GAIL to predict the modality of executing a specific "object grasping" task. This is done through learning sub-task policies from unsegmented demonstrations of tasks. Specifically, this thesis uses a dataset with trajectories regarding human behaviour towards grasping objects of different sizes in specific. These are pre-processed to correct imperfections and exploited to extract features of trajectory states that are used during training. Then, the implemented method is tested and evaluated utilizing the extracted features. The thesis concludes with a thorough presentation of results and proposals for further work towards using multi-modal imitation learning to predict human behaviour in executing tasks.

Αναγνώριση Ανθρώπινης Συμπεριφοράς μέσω Μιμητικής Μάθησης

Από

Ματθαίος Ζηδιανάκης

MTN2008

Υποβλήθηκε στο ΔΠΜΣ «Τεχνητή Νοημοσύνη» την 15 Σεπτεμβρίου 2022 ως υποχρέωση για την λήψη Μεταπτυχιακού Διπλώματος Σπουδών

Περίληψη

Στην παρούσα εργασία μελετάται η χρήση ενός αλγορίθμου μιμητικής μάθησης ώστε να μοντελοποιηθεί η ανθρώπινη συμπεριφορά για την επίτευξη μιας εργασίας με διαφοροποιημένους στόχους. Στόχος είναι να εξεταστεί η ικανότητα των μοντέλων που δημιουργούνται, στην αναγνώριση του στόχου μιας επιτελούμενης εργασίας, σε πραγματικές συνθήκες. Αυτό επιτυγχάνεται μέσω της μάθησης πολιτικών από μη τμηματοποιημένες επιδείξεις εκτέλεσης εργασιών. Ειδικότερα, η διπλωματική εξετάζει διάφορες τεχνικές μιμητικής μάθησης , υποστηρίζοντας τη χρήση του αλγορίθμου InfoGAIL που βασίζεται στον αλγόριθμο GAIL, και που έχει τη δυνατότητα συσχέτισης τρόπου συμπεριφοράς και επιτέλεσης εργασίας. Ο αλγόριθμος μελετάται στα πλαίσια αναγνώρισης συμπεριφοράς «αρπαγής» αντικειμένων διαφορετικών μεγεθών. Παρέχονται δεδομένα που αφορούν ανθρώπινη συμπεριφορά σε συγκεκριμένες συνθήκες για την αρπαγή αντικειμένων, από τα οποία εξάγονται χαρακτηριστικά και χρησιμοποιούνται για την εκπαίδευση και αποτίμηση της προτεινόμενης μεθόδου. Η διπλωματική παρουσιάζει τα αποτελέσματα και ευρήματα, και προτείνει μελλοντικές δράσεις για την χρήση μεθόδων μιμητικής μάθησης για την αναγνώριση συμπεριφοράς.

Acknowledgments

I would like to gratefully thank my supervising Professor George Vouros and his PhD student Christos Spatharis for their immense help, support and patience throughout the development of my MSc thesis, both in theoretical and technical matters. This was an excellent opportunity that I was given by my Professor who introduced such a cutting-edge research topic to me and shared his knowledge and guidance that helped me broaden my horizons towards an amazing field of research and practice on new and thrilling subjects.

I would also like to give my thanks and regards to the rest of the Committee; the Researchers Maria Dagioglou and George Petasis for all their comments and questions that helped me try and proceed one great step further at a time.

Additionally, I would like to thank my BSc Professor Panagiotis Stamatopoulos and the Researcher Stasinos Konstantopoulos who provided recommendation letters for my entry to the MSc programme.

Last but not least, I would like to thank my family, my friends and, especially, Olga, who were there supporting me 24/7.

Any opinions, findings, conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the view of University of Piraeus and Inst. of Informatics and Telecom. of NCSR “Demokritos”.

Table of Contents

List of Figures

List of Tables

1 Introduction

Machine Learning (ML) and Deep Learning (DL) pave the way towards building models that approximate high dimensional variable distributions that are difficult to be approximated by other means. Supervised ML methods rely on the ability of the model to map the labels of a dataset to the correct examples, while unsupervised methods are ideal for unlabelled datasets, and they aim to discover patterns inside the data. Figure 1.1 shows the broad groups of ML methods.

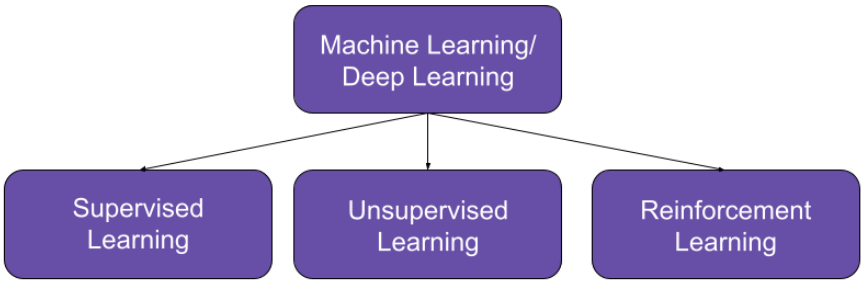

Figure 1.1: ML/DL main categories and methods

The ML methods' categories that are studied in this thesis is Reinforcement Learning and Imitation Learning, which are primarily studied in the context of human behaviour recognition and prediction.

Imitation Learning is ideal for mimicking behaviour of a subject that is considered an expert to a domain and holds valuable information on the sequence of actions that are performed towards achieving a goal / performing a task.

Human behaviour describes the human activities and motions that affect their surroundings. One of the most common activities that humans perform in their everyday life is grabbing different objects that maintain particular properties. Object size is one of the main object properties that affect the course of that activity.

The main topic for this thesis is object size prediction based on imitating human hand trajectories towards the object, expressed by noisy human body joint data,

by implementing and evaluating the InfoGAIL [1] algorithm. The problem belongs to a subgroup of human behaviour prediction problems where human hands aim to grasp an object that is placed at a specific distance from the subject (human), so the primary viewpoint for the problem is the human intention that is hidden in the trajectory of the hand. For example, the shape of the hand between reaching a small and a large item may differ. In the settings studied in this thesis, there are totally three objects of the same shape but different in size in the dataset, which defines the grasping trajectory of the hand.

The most recent approaches of this task involve prediction with classical ML supervised methods using hand kinematic features, such as Random Forest and Support Vector Machine [2], in order to recognize the human intention and to classify the object size. The InfoGAIL algorithm used in this thesis aims to also learn to mimic the hand trajectory given the object size and, at the same time, to map trajectories to the object size in order to make predictions about the human intention. The core advantage of InfoGAIL is self-learning of the model and self-recognizing of the trajectories' intrinsic properties using trial and error besides supervised labelling.

The structure of this thesis begins with Section 2 which contains several paradigms and methods that follow the ideas of Reinforcement and Imitation Learning. It provides an introduction to the topic of human behaviour and specifies the main problem addressed in this thesis and the experimental setup.

In Section 3 the InfoGAIL algorithm outline is discussed, along with its detailed properties.

Section 4 presents the implementation outline. InfoGAIL is used to autonomously model how to imitate trajectories and to recover meaningful information of the trajectories, such as the object size.

Lastly, Section 5 presents the experimental results, aiming to provide answers to the following questions:

1. Can the model identify the object size by observing the hand trajectories?
2. What is the accuracy of identifying the object size?
3. Can the model mimic hand trajectories, given the object size

These questions are examined by evaluating the model with both the training and testing examples to measure the accuracy between the actual and predicted object size.

2 Preliminaries

The following sections present the background knowledge for the methods used in this thesis, *reinforcement learning* and *imitation learning*.

2.1 Reinforcement Learning

In the context of Artificial Intelligence, Reinforcement Learning (RL) refers to the learning task of an agent that focuses on learning behaviour through trial-and-error interaction with a dynamic environment [3].

2.1.1 Process Outline

The recurring RL interaction process includes the *state* of the environment that changes every time when the agent chooses and performs a certain *action* that enables a *state transition*. A state denotes the current snapshot of the agent's environments and depending on whether the environment is *fully* observable or *partially* observable, the agent perceives the appropriate state information. The RL process also includes a *reinforcement signal (reward)* that is provided by the environment and notifies the agent for the goodness of the action it took in the specific state applied. Every time step the *reward* that the agent receives through the reinforcement signal is discounted by a constant γ factor in the range of *(0, 1)* that represents the interest of the agent to future rewards.

The actions that the agent selects to perform state transitions until it reaches a specific goal state is called *policy* and each sequence of states from start to finish is called a *trajectory*. Ultimately, the goal of the agent is to choose a policy that maximizes the expected sum of the discounted rewards, i.e., the optimal policy. The formula for the optimal policy is described as

$$\pi^* = arg \max_{\pi} \mathbb{E}[\sum_{t \geq 0} \gamma^t \, r_t | \pi],$$

where r is the reward and π is the policy.

Furthermore, the environment and policies can be either deterministic or stochastic. A deterministic environment ensures that the agent always ends up

in the state designated by the action it took, while a stochastic environment is random and may place the agent in a different state than the intended one. Respectively, a deterministic policy links a specific action to each state. A stochastic policy specifies a probability distribution on the available actions at each state. Figure 2.1 presents the RL iterative process of agent-environment interaction in order to form trajectories and solve sequential problems.

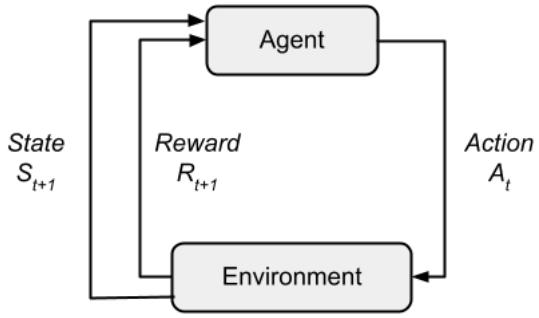

Figure 2.1: The iterative interaction between the agent and the environment to form a trajectory and solve sequential problems.

2.1.2 Markov Decision Process

The terms describing an RL task are concretely depicted by the *Markov Decision Process* (MDP) which groups all these pieces of information together.

The MDP configuration consists of the tuple *(S, A, P, r, ρ_o, γ)*: *S* is the state space in the environment, *A* is the action space for the agent, *P* is the transition probability of each state, *r* is the reward function, ρ_0 is the distribution of the initial states of the trajectories and γ is the RL discount factor. Moreover, MDP is independent of previous state information, as it does not allow landing on a transitional state by gathering information of the previous states other than the current one.

In general, the agent aims to recover the best policy towards that goal state. The total estimated γ-discounted reward from the initial state towards the goal state is the maximizing criterion of the agent that defines the optimal policy. The basic *value function* that composes the discounted reward for every state is given by the equation

$$V^{\pi}(s) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t * r_t],$$

which states that, for a specific policy π, the reward value of state *s* is the total expected discounted reward sum following the policy from that state and

afterwards. There are two major elements for calculating the optimal criterion based on V^{π}, the optimal *state-value function* and the optimal *action-value* (or *q-value*) *function,* given by their respective formulas

$$V^*(s) = \max_a Q^*(s,a),$$

$$Q^*(s,a) = \sum_{s \in \mathcal{S}} P(s,a,s') \left[r(s) + \gamma \, V^*(s') \right], \forall \, s \in \mathcal{S}.$$

The latter computes the expected reward for every state *s* of the state space *S*, given an action a for that state and then following the policy optimally for each transition probability P to state s'. The optimality is succeeded by using the value function which calculates the maximum expected reward from the state *s* and afterwards recursively. This recursive solution to the MDP problem is achieved by dynamic programming principles with the Bellman *value iteration* process [4].

The RL solutions for MDPs are based on either *model-based* or *model-free* learning [5, 6]. Model-based learning occurs when an algorithm requires the transition and/or reward functions of the environment, so the agent needs to discover the function outputs through exploration and interaction with the environment. Model-free learning does not explicitly demand a model of the environment, thus the solutions for this case require only samples of transition and reward function outputs obtained through episodes of exploration. The samples from each episode can be used for computing a running average of the value and/or the q-value function (such as the *Q-Learning* algorithm).

2.1.3 Deep Reinforcement Learning

There are cases where the state and action space of a task is very large to effectively execute the costly iterative RL process. Furthermore, the environment may not be fully observable to be able to explore all the possible paths towards a goal state. For example, in the self-driving car domain the information available to the agent about the environment comes down only to observations relative to the agent's perception proximity (hence the partially observable environment). As a result, the agent is not aware of the complete state space. In such cases, *neural networks* can be very helpful since they can efficiently approximate the policy function [7]. Neural networks can replace every part of plain RL iterative processes, such as the q-value or the state-value

function, by training multiple networks simultaneously. Figure 2.2 below illustrates an example of a policy function model that recovers a policy for the agent to follow.

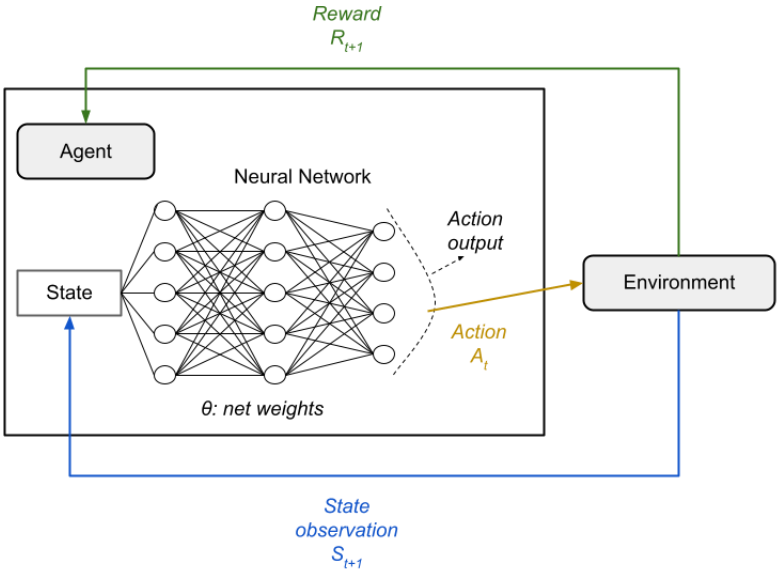

Figure 2.2: The Deep Reinforcement Learning policy model that trains a network with parameters θ in order to recover each action from state inputs

The network is trained at every iteration by updating its parameters. There can be deep network models for the *value-based* model and/or the *policy-based* model or even for the environment model.

An example of the policy model is presented in Figure 2.2. Policy-based algorithms approach the unknown policy directly by updating the parameters of the policy model with gradients $\nabla_\theta log\pi_\theta(a_t|s_t)$ by means of *policy-gradient* algorithms such as REINFORCE [8]. Value-based algorithms such as Deep Q-Networks (DQN) [9] learn the q-value function of each action using deep neural networks and then form the policy that prescribes actions with the maximum q-value per state. Lastly, the methods that leverage both policy- and value-based methods are called *actor-critic* [10]. The actor approximates the policy function and the critic network learns the value function that the actor tries to maximize. For that, the critic uses a baseline in order to evaluate the predicted value, such as the Q-value, which is parameterised by a deep neural network (*Q actor-critic*). The critic output essentially controls the policy gradients of the actor network and appraises the actions produced by the actor. Figure 2.3 shows the categorization of different methods solving MDPs.

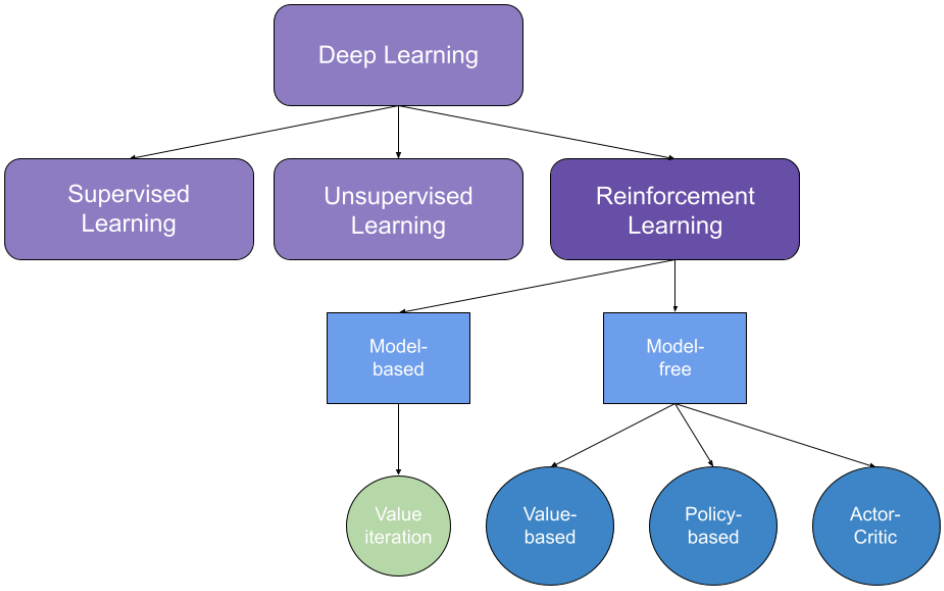

Figure 2.3: Deep Learning chart with the extended RL features

The issue that arises when policies are involved in deep learning is that decisions on actions are greatly affected by the changes in the parameter step during a network parameter update. For that matter, there are several techniques, such as Trust Region Policy Optimization (TRPO) [11] algorithm, which ensure a moderate step of the network parameter updates. Briefly, TRPO maximizes the loss function

$$L(\theta_{old}, \theta) = \mathbb{E}_{\pi_{\theta_{old}}}\left[\frac{\pi_\theta}{\pi_{\theta_{old}}} A_{\theta_{old}}(s,a)\right],$$

$$s.t.\, \overline{D}_{KL}(\theta_{old}, \theta) = \mathbb{E}_{\pi_{\theta_{old}}}\left[D_{KL}\left(\pi_{\theta_{old}}, \pi_\theta\right)\right] \leq \delta.$$

This loss function basically translates into estimating the expected ratio of the policy π_θ after the parameter update over the old policy $\pi_{\theta_{old}}$, multiplied by the *advantage function* $A_{\theta_{old}}$ of the state-action pairs sampled from the old policy. The loss function is calculated subject to $\overline{D}_{KL}$ that denotes the average KL divergence [12] of the old and new policies, which is bounded by a constant δ. It essentially means that the new policy is allowed to deviate from the old policy *at most* δ.

The advantage function defined by the formula

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) = R_{t+1} + \lambda\, \gamma\, V(s_{t+1}) - V(s_t),$$

where R is the surrogate reward at time $t+1$ of the trajectory, which denotes the reward obtained at the next state of the transition, γ is the discount factor,

$Q(s_t, a_t)$ is the q-value for the state-action pair at time t and V is the value function. λ is a regularizing hyperparameter.

2.2 Imitation Learning

Imitation Learning (IL) is the broad category of methods in which the involved agents aim at directly mimicking expert demonstrations in a task of interest using supervised learning [13, 1]. The expert demonstrations may originate from humans or even from other agents that perform actions to complete a specific task. Imitation learning is divided into two key groups, *behavioural cloning* (BC) and *inverse reinforcement learning* (IRL) [1, 14, 15, 16].

2.2.1 Behavioural Cloning

Behavioural Cloning (BC) denotes the simplest and straightforward imitation learning method, fusing supervised learning, directly through expert behaviour without any access to a reward function. The expert demonstrations mainly consist of pairs of states and actions performed in the environment by the agents. The core difference from reinforcement learning revolves around the absence of the reinforcement signal that supports the agents for the action selection and there is no interaction with any part of the environment. This offers the advantage of not having to directly compute and develop a complex reward function that yields a problem-specific reinforcement signal, which usually is computationally intensive, while at times the reward is obscure and unknown. Moreover, since IL method and specifically BC method is supervised, the agent does not learn a policy using experience earned by trial and error, it rather needs data from the expert. In other words, the more data the expert provides to the agent, the more accurately it learns the expert state-action distribution.

A significant drawback of BC is the compounding error in the approximated policy. This error occurs when small errors in the approximated policy function gradually lead the agent to unseen states and does not know how to act on them.

2.2.2 Inverse Reinforcement Learning

IRL designates the approximation of a reward function related to a desired problem using *apprenticeship learning* (AL), an alternative term for learning from demonstrations, and then the agent solves a reinforcement learning problem using the recovered reward so as to calculate the expert policy. The agent tries to infer the hidden preferences of the expert which define the reward function [17]. At each step, the parameterized reward network makes a prediction that yields a candidate reward which is used to solve the MDP pipeline in order to calculate the optimal policy given a reward approximation. The network parameters may be updated by minimizing the distance between the learned and the expert policy. The algorithm is described briefly in Figure 2.4.

Algorithm 1: Template for IRL.

Input: $\mathcal{M} \backslash_{R_E} = \langle S, A, T, \gamma \rangle$,
Set of trajectories demonstrating desired behavior:
$\mathcal{D} = \{\langle (s_0, a_0), (s_1, a_1), \ldots, (s_t, a_t) \rangle, \ldots\}$, $s_t \in S$, $a_t \in A$, $t \in \mathbb{N}$,
or expert's policy: π_E, and reward function features
Output: $\hat{R}_E$

1. Model the expert's observed behavior as the solution of an MDP whose reward function is not known;
2. Initialize the parameterized form of the reward function using any given features (linearly weighted sum of feature values, distribution over rewards, or other);
3. Solve the MDP with current reward function to generate the learned behavior or policy;
4. Update the optimization parameters to minimize the divergence between the observed behavior (or policy) and the learned behavior (policy);
5. Repeat the previous two steps till the divergence is reduced to a desired level.

Figure 2.4: Template algorithm for estimating a reward function using IRL (source: [17])

There are several IRL optimization methods for the divergence between the expert and learned policy, which are displayed in Figure 2.5.

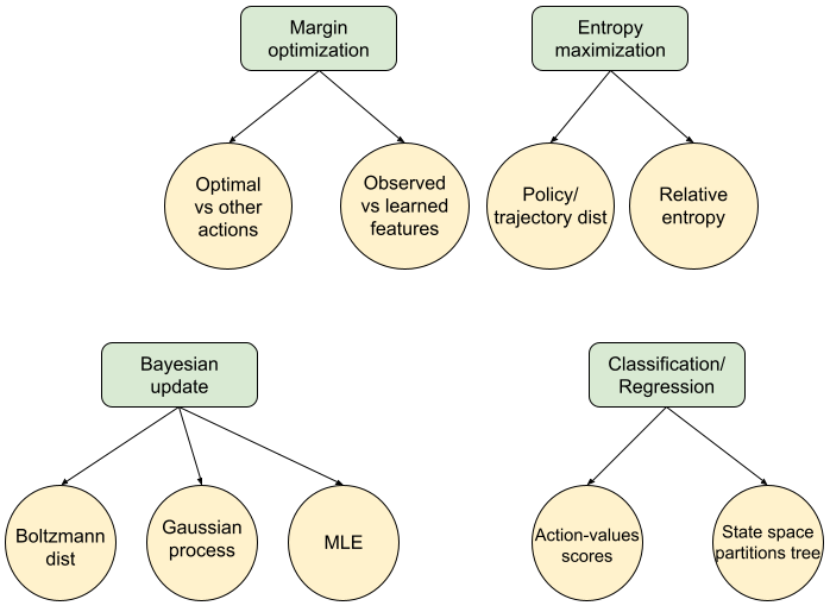

Figure 2.5: IRL objective function optimization methods

Like any imitation learning method, IRL assumes that the expert policy is optimal. The main challenge that poses an ambiguity to the solution is the fact that there can be multiple reward functions that could explain the optimality of the expert policy. The optimization step of the IRL algorithm in order to alleviate the ambiguity issue uses *entropy maximization*. This method returns the maximum entropy policy that is calculated by AL via IRL.

2.2.3 Imitation Learning: Algorithms and State of the art

Imitation Learning is a task that requires interaction samples in order to succeed a robust and interpretable result. Many real-world settings prohibit the collection of such data efficiently, especially in the field of robotics. This usually leads to choosing reasonably plain simulations of actual human behaviour problems in order to compensate for the sample complexity [18].

Recent work has shown that combining the benefits of the techniques described in the previous sections, human behaviour datasets can be integrated into both deep RL solutions and BC-AL algorithms that contribute to human movement imitation.

In [19] classic BC is used to learn state-action pairs towards performing various tasks such as reaching, grasping and pushing objects by hand, creating optimal policies using a virtual reality robot that is teleoperated to perform such

movements. The created demonstrations are images containing the viewpoint of the VR robot performing these tasks. Then, the BC model learns the policy by imitating the collected demonstrations and the evaluation of the model is performed utilizing an actual robot to perform the tasks by executing the learned policy.

Dataset Aggregation (DAgger) is an upgraded version of the standard BC algorithms that is used for expert trajectory imitation by iteratively collecting data not only from the expert policy but also from the policy network instance that the agent uses at each iteration [20]. At each step, the agent asks for the expert feedback in the form of expert policy actions, given the visited states. Contemporary experiments have been performed on an autonomous driving task, using both simulated and real-world datasets. These experiments feature the DAgger algorithm, as well as Human-Gated DAgger (HG-DAgger) which adds a risk metric that helps the agent remain in a human-defined state space to avoid deviating from the expert state space. The real and the simulated automobiles are tested in a constrained environment with other cars as obstacles [21].

Expert demonstrations are crucial when BC algorithms are used. Policy-based methods that use policy gradient to recover an optimal policy can be initialized randomly, which may slow the training process down, since the agent has no clue of the state and action space. This can be avoided when injecting expert demonstrations into the policy gradient methods. Demonstration Augmented Policy Gradient method (DAPG) [22] combines supervised techniques with self-learning, by pre-training the policy model with BC or splitting the expert demonstrations in sub-tasks, in cases where smaller sequential tasks are pipelined to form the original trajectories. The latter choice requires an augmented surrogate objective function that extends the policy gradient objective. This auxiliary function aims to grasp the inherent sequential information of the mentioned sub-tasks, which the BC method fails to capture. The experimental setup of DAPG includes a virtual environment in which a 24-DoF robotic hand is used to produce expert trajectories for object grasping and relocation, object manipulation using the fingers, usage of hammer and door opening. The rewards are manually crafted depending on the type of task and the experiments are tested on sparse task completion.

Generative Adversarial solutions

While the BC family highlights supervised procedures, there are IL techniques that directly learn the policy from expert demonstrations, using Generative Adversarial Network (GAN) aspects [23].

GAIL algorithm outline

Generative Adversarial Imitation Learning (GAIL) [15] offers a core example of reclaiming the best of both GANs and IL that constitutes a baseline for other methods of GAN learning. This algorithm excels in large environments with high-dimensional state-action space that hinder the agent from keeping a steady trajectory close to that of the expert. The previously mentioned methods mainly rely on expert demonstrations without evaluating the improvement of the new policy, compared to the previous one, throughout the course of training. This does not allow the agent to qualify the parameter update at each step. GAIL utilizes a discriminator network that evaluates the actions generated by the policy network, with respect to the corresponding reached state after performing a specific action. It also has the role of the cost function. The goal is to maximize the aggregated expected cost for the learned policy, while trying to find the policy that minimizes the loss.

The study in [15] shows tests of the algorithm on some baseline RL experiments such as cartpole, acrobot and mountain car, as well as on more complex simulative tasks such as 3D humanoid locomotion. Other experiments that specialize in human pose sequence prediction utilize Wasserstein-divergence GAIL algorithm (WGAIL-div), a variation of the standard GAIL algorithm that effectively approximates trajectories such as photo-shooting and walking [24].

GAIL features explained

GAIL is founded on the IRL principles. However, its purpose is to escape from the AL process which solves RL problems by approximating the reward function, in order to yield the policy.

The structure is inspired by the GANs framework, where the cost function is formed by the output of a discriminator network. Formally, the discriminator aims at learning to discriminate between learner-agent and expert state-action pairs:

$$\psi_{GA}^{*}(\rho_{\pi} - \rho_{\pi_E}) = \max_{D\in(0,1)} \mathbb{E}_{\pi}\big[\log\big(D(s,a)\big)\big] + \mathbb{E}_{\pi_E}\big[\log\big(1 - D(s,a)\big)\big],$$

where D is the discriminator and ρ_π, ρ_{π_E} are the *occupancy measure* values for the imitator policy π and the expert policy π_E, respectively. Occupancy measure denotes the distribution of the state-action pairs visited by the agent when exploring the environment following a policy.

Substituting the cost function in the optimization problem along with λ that controls the causal entropy H_π, the final objective function for the GAIL algorithm is formed:

$$\min_{\pi} \max_{D\in(0,1)} \left(\mathbb{E}_\pi\left[\log\left(D(s,a)\right)\right] + \mathbb{E}_{\pi_E}\left[\log\left(1-D(s,a)\right)\right]\right) - \lambda H(\pi),$$

with $\lambda \geq 0$ being the regularizing variable for the causal entropy. The RL step for getting the optimal imitator policy is replaced by a generator network that at each iteration yields actions based on the input state. The TRPO method is proposed to control the policy network parameter update steps. The GAIL algorithm is presented in Figure 2.6.

Algorithm 1 Generative adversarial imitation learning

1: **Input:** Expert trajectories $\tau_E \sim \pi_E$, initial policy and discriminator parameters θ_0, w_0
2: **for** $i = 0, 1, 2, \ldots$ **do**
3: Sample trajectories $\tau_i \sim \pi_{\theta_i}$
4: Update the discriminator parameters from w_i to w_{i+1} with the gradient

$$\hat{\mathbb{E}}_{\tau_i}[\nabla_w \log(D_w(s,a))] + \hat{\mathbb{E}}_{\tau_E}[\nabla_w \log(1-D_w(s,a))] \tag{17}$$

5: Take a policy step from θ_i to θ_{i+1}, using the TRPO rule with cost function $\log(D_{w_{i+1}}(s,a))$. Specifically, take a KL-constrained natural gradient step with

$$\begin{aligned} &\hat{\mathbb{E}}_{\tau_i}\left[\nabla_\theta \log \pi_\theta(a|s) Q(s,a)\right] - \lambda \nabla_\theta H(\pi_\theta), \\ &\text{where } Q(\bar{s},\bar{a}) = \hat{\mathbb{E}}_{\tau_i}[\log(D_{w_{i+1}}(s,a)) \mid s_0 = \bar{s}, a_0 = \bar{a}] \end{aligned} \tag{18}$$

6: **end for**

Figure 2.6: GAIL algorithm (source: [15])

The final algorithm shows that the cost function approximator is $\log D(s,a)$, which means that the discriminator guides the generator into predicting the right policy, without the means of a reinforcement signal or an iterative RL process. The generator *G* and the discriminator *D* are two distinct networks and each one has its own parameters to update. *G* yields a policy and tries to confuse *D*. This essentially means that if *D* is not able to distinguish the state-action pairs generated by G by assigning a small cost compared to that for the expert state-action pairs, it is inferred that the occupancy measures of the expert and generated policy are indeed very close, thus the data generated by *G* is close to the expert data. The discriminator is updated with the Adam optimizer [25].

3 Info GAIL

Having described the GAIL baseline above, the main algorithm derived from GAIL is now discussed, which is used for discovering the salient factors of the trajectories when following a policy.

3.1 InfoGAIL Outline

Information Maximizing Generative Adversarial Imitation Learning (InfoGAIL) is another algorithm of the GAIL family that introduces the concept of latent codes in the learning process [1]. There are problems which demand distinguishing the intrinsic labels of the expert. Latent codes denote these exact labels and InfoGAIL, apart from training a discriminator and a generator network like standard GAIL, it also takes on maximizing the mutual information between the generated trajectory and the latent variable by training a neural network that indicates the posterior probability of the latent code, given the generated trajectory. Figure 3.1 shows that InfoGAIL recognizes the generated trajectory latent factors and matches the expert.

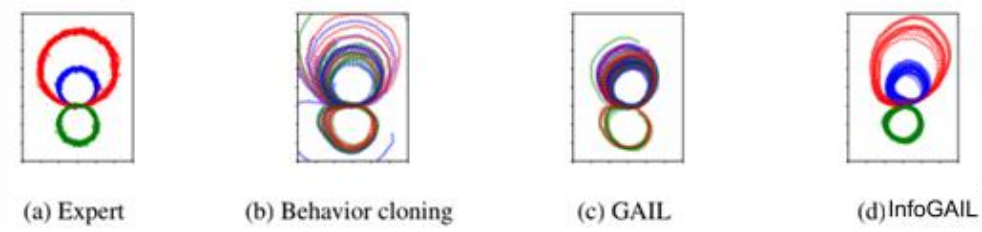

Figure 3.1: Comparison between InfoGAIL and other IL algorithms for the synthetic experiment (source: [1])

The main experiment in [1] processes visual positions of cars driving in a simulated environment in order to distinguish whether the car is driving throughout a turn in the environment or is passing another vehicle. In Figure 3.2 there are 37 epochs of turning left or right during the training process, along with the distance travelled for 60 trajectory rollouts.

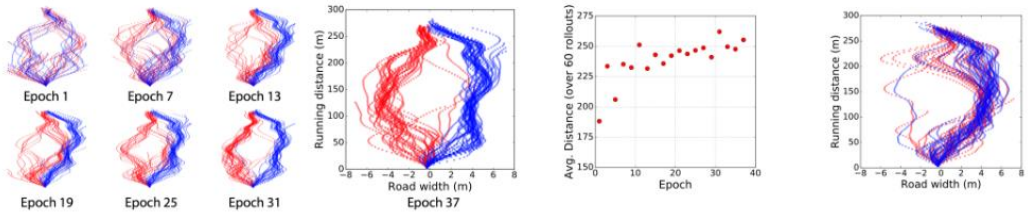

Figure 3.2: Results of InfoGAIL for certain epochs and compared to GAIL, for the visual demonstration experiments (source: [1])

The images infer the effectiveness of InfoGAIL to recognize the type of turn the virtual vehicle took and map the trajectory to the right color and type. The far-right image depicts the GAIL trajectories of passing vehicles and it is obvious that the algorithm cannot distinguish between left and right pass, compared to InfoGAIL.

3.2 Details on InfoGAIL

The standard GAIL algorithm is effectively able to predict trajectories based on expert state-action pairs in high dimensional environments where classic RL and IRL concepts fail to succeed in real time intervals. However, trajectories in general and specifically in human behaviour context, are determined by several latent factors that explain the very behaviour. Furthermore, the same demonstrations may originate from different experts whose skill in the task area may also differ, thus the imitation learning process in such cases inducts variability.

The motivation behind InfoGAIL is the interpretability of the expert policy by discovering the desired latent variables along with the state-action prediction. In order to do so, maximization of the mutual information between the expert demonstrations and the latent space that contains the latent variables can be achieved. This intuitively translates to mapping every generated state-action pair to a latent variable. In [26] the mutual information formula is displayed as

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X),$$

where H is the entropy term for the variables X and Y. This formula defines how much information can be extracted from the variable X, given the knowledge about Y. So, if the expert policy π_E consists of the expert policies that are matched with their respective known latent variable c, then each expert

trajectory τ_E is defined as the consequent state-action pairs that are generated following the policy and each action is sampled from the distribution $p(\pi|c)$, starting from an initial state s_0. Now, the imitator policy $\pi(a|s,c)$ tries to approach the occupancy measure of the expert policy π_E by generating trajectories τ which are qualified by the discriminative network D, like in the standard GAIL setting.

The mutual information formula must contain the latent variable and trajectories, so the problem is reduced to calculating $I(c;\tau)$. While this seems a straightforward solution over GAIL, the final objective function needs to contain the posterior probability $P(c|\tau)$, since the entropy term contains this quantity. It is difficult to directly compute this posterior to maximize the mutual information in its current form, thus a lower bound can be placed, which includes an approximation of the posterior and therefore does not contain the quantities $H(c|\tau)$ or $H(\tau|c)$. This lower bound is given by

$$L_I(\pi, Q) = \mathbb{E}_c[\log Q(c|\tau)] + H(c) \leq I(c;\tau).$$

Q denotes the approximation of the posterior and $H(c)$ is the entropy of the latent variable c. This bound needs to be included in the GAIL objective in order to achieve the required semantic features. As a result, the final transformed objective function that is gradable and constitutes InfoGAIL is

$$\min_{\pi,Q} \max_{D} \mathbb{E}_{\pi}[\log D(s,a)] + \mathbb{E}_{\pi_E}[\log (1 - D(s,a))] - \lambda_1 L_I(\pi, Q) - \lambda_2 H(\pi),$$

where $\lambda_1 > 0$ is the regularizing variable for the lower bound, similar to the regularizing variable for the policy entropy from GAIL. The posterior approximation Q is modelled by a neural network as well, with Adam optimizer for updating its parameters. Overall, there are three main networks for the core InfoGAIL algorithm, the generator π_θ, the discriminator D_ω and the posterior Q_ψ, with θ, ω and ψ being their respective parameters. Figure 3.3 shows the main InfoGAIL algorithm for computing the gradients and using them to update networks weights.

Algorithm 1 InfoGAIL

Input: Initial parameters of policy, discriminator and posterior approximation $\theta_0, \omega_0, \psi_0$; expert trajectories $\tau_E \sim \pi_E$ containing state-action pairs.
Output: Learned policy π_θ
for $i = 0, 1, 2, ...$ **do**
Sample a batch of latent codes: $c_i \sim p(c)$
Sample trajectories: $\tau_i \sim \pi_{\theta_i}(c_i)$, with the latent code fixed during each rollout.
Sample state-action pairs $\chi_i \sim \tau_i$ and $\chi_E \sim \tau_E$ with same batch size.
Update ω_i to ω_{i+1} by ascending with gradients

$$\Delta_{\omega_i} = \hat{\mathbb{E}}_{\chi_i}[\nabla_{\omega_i} \log D_{\omega_i}(s,a)] + \hat{\mathbb{E}}_{\chi_E}[\nabla_{\omega_i} \log(1 - D_{\omega_i}(s,a))]$$

Update ψ_i to ψ_{i+1} by descending with gradients

$$\Delta_{\psi_i} = -\lambda_1 \hat{\mathbb{E}}_{\chi_i}[\nabla_{\psi_i} \log Q_{\psi_i}(c|s,a)]$$

Take a policy step from θ_i to θ_{i+1}, using the TRPO update rule with the following objective:

$$\hat{\mathbb{E}}_{\chi_i}[\log D_{\omega_{i+1}}(s,a)] - \lambda_1 L_I(\pi_{\theta_i}, Q_{\psi_{i+1}}) - \lambda_2 H(\pi_{\theta_i})$$

end for

Figure 3.3: InfoGAIL algorithm (source: [1])

Since the expert demonstrations usually come from humans, it is natural that some expert policies are prone to human error and perform sub-optimally. Apart from the main course of the algorithm, an additional *reward augmentation* is proposed when this scenario is encountered, in order to counteract the sub-optimal approximated rewards. Essentially, reward augmentation provides an extra invented constraint $\eta(\pi_\theta)$, specifically crafted to better inform the agent about a more accurate route towards the states it should visit. This surrogate reward is inserted into the objective function as well.

Lastly, an additional optimization step is suggested that solves potential *vanishing gradient* and *mode collapse* issues when there are high dimensional expert data. In that case, the maximization part of the objective function is replaced with the Wasserstein GAN (WGAN) technique. The updated objective with both the reward augmentation and WGAN is described by the following formula:

$$\min_{\theta,\psi} \max_{\omega} \mathbb{E}_{\pi_\theta}[D_\omega(s,a)] - \mathbb{E}_{\pi_E}[D_\omega(s,a)] - \lambda_0\, \eta(\pi_\theta) - \lambda_1\, L_I(\pi_\theta, Q_\psi) - \lambda_2\, H(\pi_\theta).$$

4 Predicting Human Behaviour

This section introduces:

- the dataset used for InfoGAIL
- the formulation of the problem
- the setup and implementation of the InfoGAIL algorithm

4.1 Overview

So far various methods and setups have been presented that depend on imitation learning principles in order to model expert policies as these are revealed through task demonstrations. The agent observes the expert state-actions pairs and imitates them as closely as it gets, or it makes its own path in the environment to learn from its own mistakes, always keeping the expert policy as a guide.

As stated in the InfoGAIL section, human behaviour can be noisy in multiple examples and slight variations of the same policy could explain different semantic features. In the hand grasping environment, these features must keep up with the interpretability of the respective trajectories. Hence, the goal of this analysis is to bring the dataset to a form that best highlights the modes of behaviour and to remove any excess noise from the features.

4.2 Human Behaviour Dataset

The utilized raw dataset for the current thesis is a collection of *OpenPose* (OP) estimations of the right human hand positions in the 2D space as it reaches an object [2]. Three solid cubes are used that vary among three categories, *small* (*2.5cm*), *medium* (*5.5cm*) and *large* (*7.5cm*). A *single RGB-D sensor* was used by the authors to capture the hand joint data of 8 participants who took 30 movements towards each object. There are totally *715* grasping movements towards the cubes, after the removal of 5 movements because the recording was

defective. Every participant initiates the hand movement from the same fixed point that has a specific distance from the object. OP groups joint data into right hand data and full body data. It reads the recordings and recognizes the hand joint and full body data, as depicted in Figure 4.1.

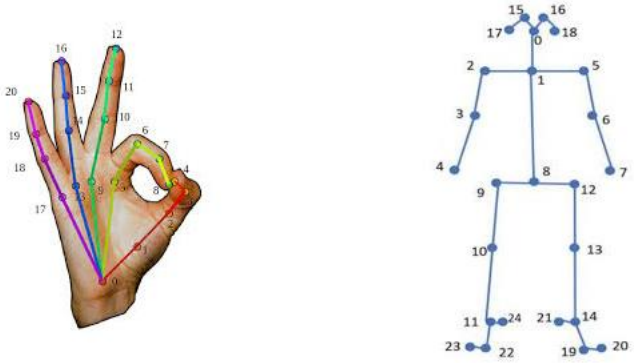

Figure 4.1: (Left) OP right hand joint data, (Right) OP full body joint data

Each joint number corresponds to a label that is used to save the data in *CSV* files. Table 4.1 matches the labels with each joint number from both right hand and full body.

Table 4.1: OP key point labels

Label	Number
RWrist	Full body - 4
RPalmBase	Right hand - 0
RThumb1CMC	Right hand - 1
RThumb2Knuckles	Right hand - 2
RThumb3IP	Right hand - 3
RThumb4FingerTip	Right hand - 4
RIndex1Knuckles	Right hand - 5
RIndex2PIP	Right hand - 6
RIndex3DIP	Right hand - 7
RIndex4FingerTip	Right hand - 8
RMiddle1Knuckles	Right hand - 9
RMiddle2PIP	Right hand - 10

RMiddle3DIP	Right hand - 11
RMiddle4FingerTip	Right hand - 12
RRing1Knuckles	Right hand - 13
RRing2PIP	Right hand - 14
RRing3DIP	Right hand - 15
RRing4FingerTip	Right hand - 16
RPinky1Knuckles	Right hand - 17
RPinky2PIP	Right hand - 18
RPinky3DIP	Right hand - 19
RPinky4FingerTip	Right hand - 20

Each movement consists of several frames with a sampling frequency of *60Hz* that feature the OP key-points, along with the OP confidence probability that denotes how correctly OP estimated the coordinates of each point, the sequence number of the frame and the key-point timestamp. The total number of frames differs in each trajectory. Furthermore, the actual movement begins from the 10^{th} frame, because the first 9 frames are used to calculate the standard deviation of the RWrist y-coordinate for manual dataset reproduction. Lastly, every hand movement ends its course just right before object grasping. Figure 4.2 shows the frame distribution among the hand movements.

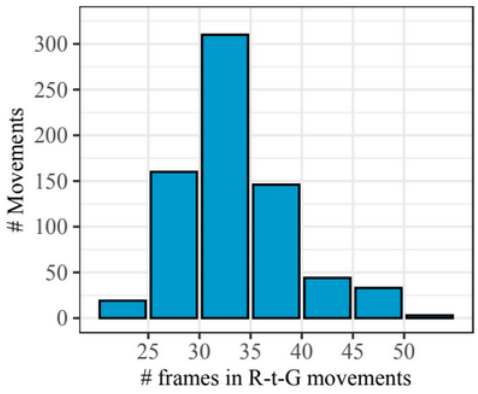

Figure 4.2: Frame distribution for each movement (source: [2])

The timestamp is in seconds and the sequence number is an integer that OP yielded, counting from the first frame until the last frame captured and ranges in [0, MAX_TRAJECTORY_FRAME]. Figure 4.3 displays the timestamp

difference between every two consecutive frames of each trajectory, collapsed into one dimension as seen in the x-axis.

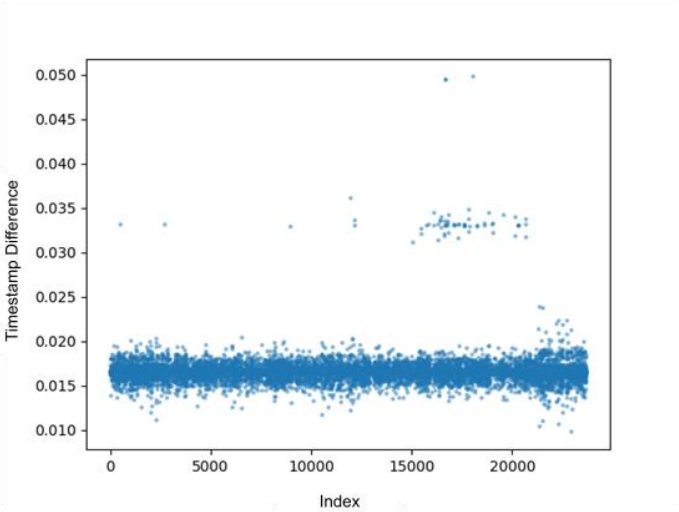

Figure 4.3: Consecutive timestamp differences (in seconds) between frames for all the trajectories

The sequence numbers of a movement in the dataset do not necessarily start from zero and they are not definitively consecutive, since some frames have been filtered out due to noisy values. Figure 4.4 shows an example frame taken from a hand movement towards a large cube. Each dot corresponds to a hand (RWrist) joint key point. The recording of that movement capturing the total 22 key points is achieved by exploiting the OpenCV framework [27] to read and extract the appropriate information from the dataset.

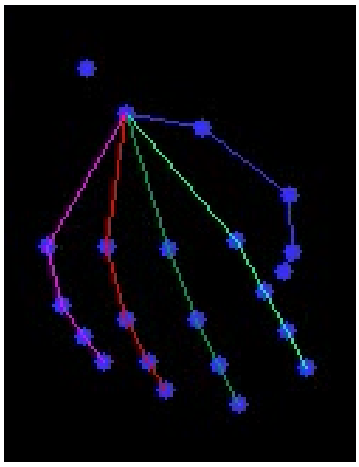

Figure 4.4: Example frame with hand points and the wrist point reaching for a large object

Each movement is contained in its own CSV file and Table 4.2 shows the format of the CSV containing the values mentioned above.

Table 4.2: CSV file layout

Sequence No.	Timestamp	Point probability	Point x-coordinate	Point y-coordinate	Rest Points
first_No.	*first_stamp*	*(0, 1)*	*float_value*	*float_value*	...
		. . .			
last_No.	*last_stamp*	*(0, 1)*	*float_value*	*float_value*	

The *Point* headers in the table annotate each label mentioned in Table 4.1. So, every Point triplet is repeated in the CSV file for all *22* key points.

In order to reduce the data dimensionality and end up with meaningful features, it is showed in [2] that three of the total features yielded the best outcome: the 2D *wrist* key-point coordinates (x and y), which matches with the RWrist label, and the *thumb-index finger aperture*. The y-RWrist coordinate is quite important because it is observed that most of the hand motion happens on the y-axis and its confidence probability is larger than *0.6*. The *thumb-index finger aperture* denotes the Euclidean distance between the 2D coordinates of the thumb and index fingertips, which correspond to *RThumb4FingerTip* and *RIndex4FingerTip* labels from Table 4.1, respectively.

Figure 4.5 displays the distributions of Dataset A endpoints for every type of objects in order to demonstrate the diversity of each object approaching situation.

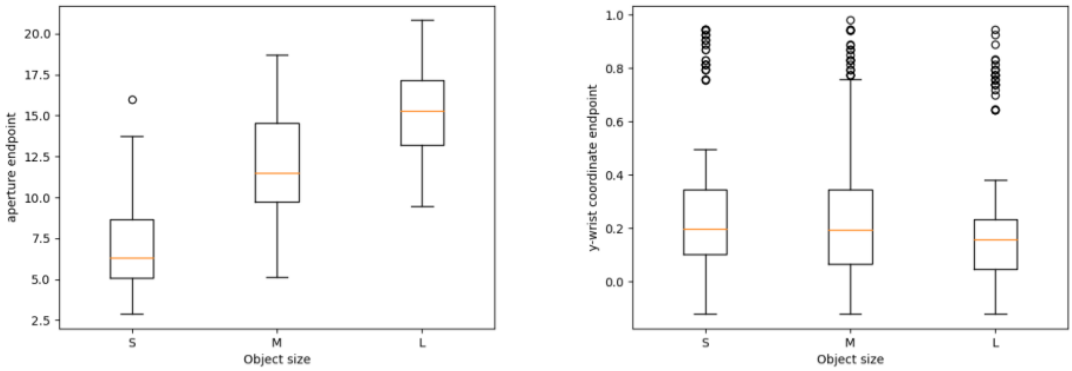

Figure 4.5: Aperture and y-wrist endpoint distribution for the trajectories of Dataset A. Each distribution (boxplot) refers to the relative object size (S=small, M=medium, L=large). The orange line represents the mean value of each distribution, and the white circles are the outliers

Even though most of the movement takes place in the y-axis, the mean values (orange lines in the boxplots) are not well-distinguished for each object size, as opposed to the aperture values during the trajectory. So, an upcoming challenge is to determine whether the combination of the aperture feature (as the most important one that defines the approached object size) and the spatial wrist x, y features that define the generic hand position in relation to both the environment and the object is actually helpful, or the aperture is adequate by itself in order to produce trajectories that exhibit the association of the aperture to the object. Thus, two distinct feature sets are explored:

- All three features *(x-wrist coordinate, y-wrist coordinate, thumb-index aperture)*
- Only the *thumb-index aperture* feature

Figure 4.6 shows the scatter plots of all the aperture values of the dataset with respect to five hand movement intervals (*20%*, *40%*, *60%*, *80%*, *100%*), which refer to the movement completion percentage from the initial frame of each movement.

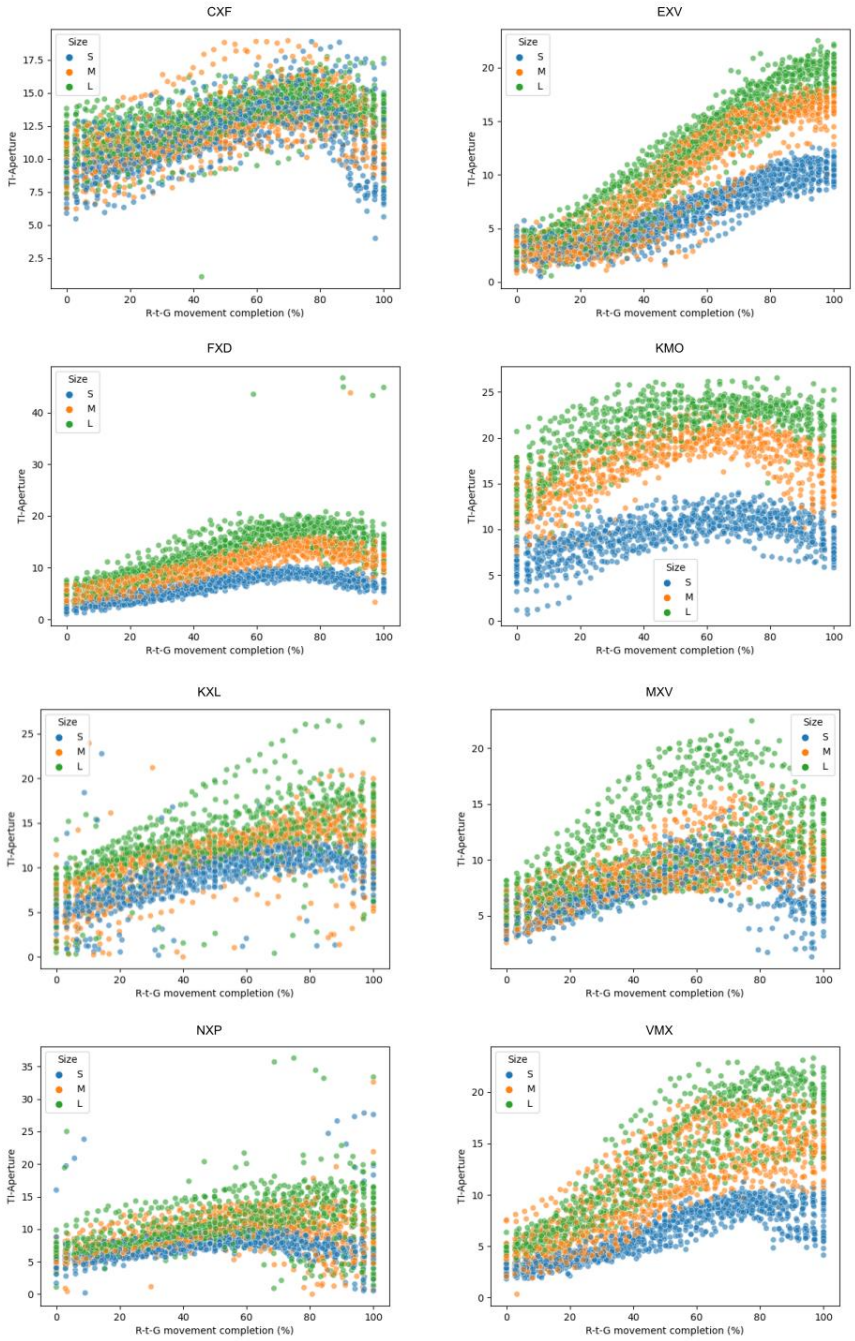

Figure 4.6: The scatter plots of all the aperture values of the dataset with respect to five hand trajectory intervals (*20%*, *40%*, *60%*, *80%*, *100%*), which refer to the trajectory completion percentage from the initial frame of each trajectory, for the 8 participants and each object size

Before incorporating the aforementioned effective features in a new dataset, extra preprocessing steps need to be performed in order to clean and normalize the dataset.

While the x and y coordinates of the wrist point variables are filtered out to accommodate a confidence probability close to *0.6* and over, some of the thumb

and index fingertip values needed to form the aperture feature exhibit confidence less than *0.6*. The boxplot in Figure 4.7 shows the confidence distribution of the wrist, thumb and index key points of every trajectory in the dataset.

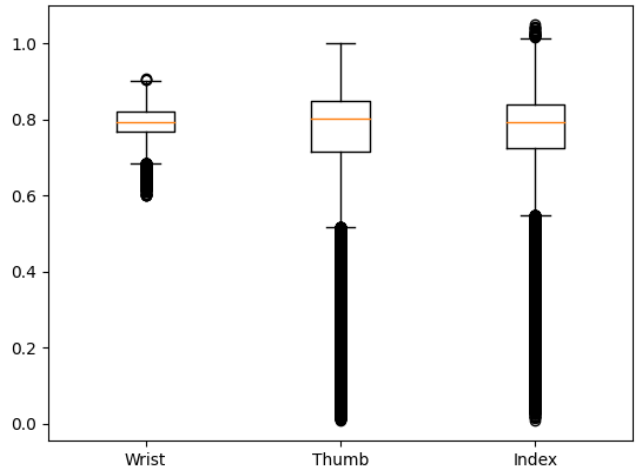

Figure 4.7: Confidence probability distribution for each key point (wrist, thumb and index)

The image shows that the mean confidence distribution lies around *0.8* for all the key points and most of the outliers are indeed in the thumb and index key points. As a result, all the thumb and index points that exhibit OP confidence less than *0.6* can be safely removed, and all the remaining points ensure a stable confidence level. Furthermore, another constraint for the thumb-index points is that the minimum distance of x and y coordinates between two thumb and index points needs to be less than or equal to 10 pixels, as proposed in [2].

Filtering out frame values while calculating the apertures does not imply removal of the whole frame, but it rather labels these specific values in the corresponding frame as unusable and blank, because there can be valid wrist points and invalid apertures in the same frame. In Table 4.3 there is an example where the thumb key point value has been removed due to low confidence level (the removal is marked with 'X'), but the frame is still in its position, along with the wrist key point.

Table 4.3: Example frame after thumb value removal

Wrist probability	**Wrist Point**	**. . .**	**Thumb probability**	**Thumb Point**
0.86	*(112.74, 101.73)*		*0.36*	X

Then, the aperture feature is formed using the available thumb and index points (i.e., the points that have real values, like the Wrist Point in Table 4.3) and every aperture value corresponding to unavailable thumb and index fingertip points (i.e., the points that do not have real values, like the removed Thumb Point in Table 4.3) become unavailable. Now the dataset contains only three features, the *apertures* (some of which are unavailable) and the *wrist point coordinates* (x and y), along with the respective *timestamp*. Furthermore, each trajectory now begins from the tenth frame of the initial raw dataset.

Figure 4.8 shows the percentage of NaN aperture values for every hand trajectory and object size. NaN stands for *not-a-number,* and it is the non-numerical blank value that some apertures have from the previous step.

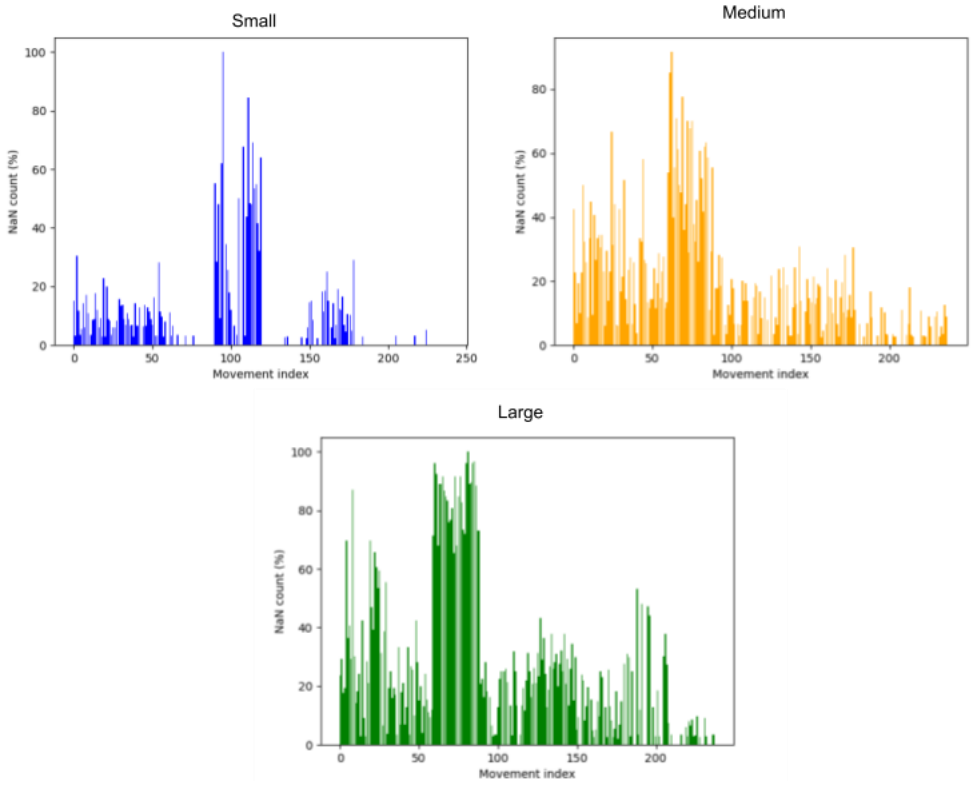

Figure 4.8: Aperture blank (NaN) percentage in each trajectory for each object size. Each bar plot corresponds to one of the three object sizes. The x-axis represents the index of the total number of trajectories for the corresponding object size.

The image implies that the non-available (NaN) aperture values are less than *50%*, with the obvious exception of trajectories that have more than *70%* NaN apertures. In order to ensure that the hand trajectories contain at least one usable frame, the trajectories with *100%* NaN aperture values are removed. By doing so, the total remaining hand trajectories are *713* (only *2* were removed).

This process may leave unusable starting and ending frames due to the NaN aperture values, which are important movement-wise since they indicate the spatial limits of a trajectory. Moreover, the quantity of the dataset examples is significant for the training efficiency. Consequently, the NaN aperture values at starting and ending frames, as well as the rest of the NaN aperture values can be filled with real values in order to maintain the current number of hand trajectories.

At first, the starting and ending aperture NaN values can be replaced with samples from the distributions of the respective remaining real aperture values, after the feature filtering step described above. The sampling depends on the participant making the hand trajectory and on the respective size of the reached object. In cases where there is no available example for the corresponding combination of participant and object size, the chosen aperture value is the mean aperture value from all the participants, for the desired object size. Figure 4.9 shows an example of the latter occasion for the FXD participant and the large cube. The image indicates that there is not any available real aperture value for the particular participant and object size combination (FXD participant and large object size), in order to complete an ending unavailable (NaN) aperture value. This occurs since the box plots of small and medium objects (S and M respectively) are displayed, while there is no box plot for the large object (L).

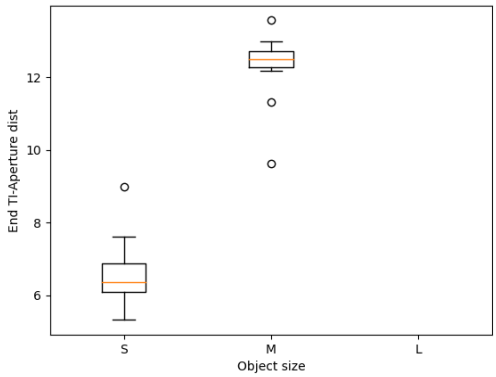

Figure 4.9: Ending aperture value distribution for the FXD participant for every object size. There is no available value for the large object.

To complete the dataset, the remaining NaN aperture values are replaced with real values using *linear interpolation*. This method replaces NaN aperture values a_x, where $x \in [2,\ n-1]$ and n is the length of a hand trajectory. The formula for linear interpolation is $a_x = a_i + (x-i)\frac{a_j - a_i}{j-i}$, where $i < x < j$ and a_i and a_j denote the closest aperture values to a_x that are not NaN, so that $a_i < a_x < a_j$. Linear interpolation is performed for all the remaining *713* hand trajectories. This method is significant because each frame (hence, each aperture value) of a hand trajectory corresponds to a timestamp. As seen on Figure 4.3, most of the timestamp difference values are around 0.016. If all the NaN values are removed, along with the entire frame, then the timestamp difference values are completely unequal. Thus, linear interpolation ensures stable frame rate. Figure 4.10 below presents the interpolated values which complement Figure 4.6.

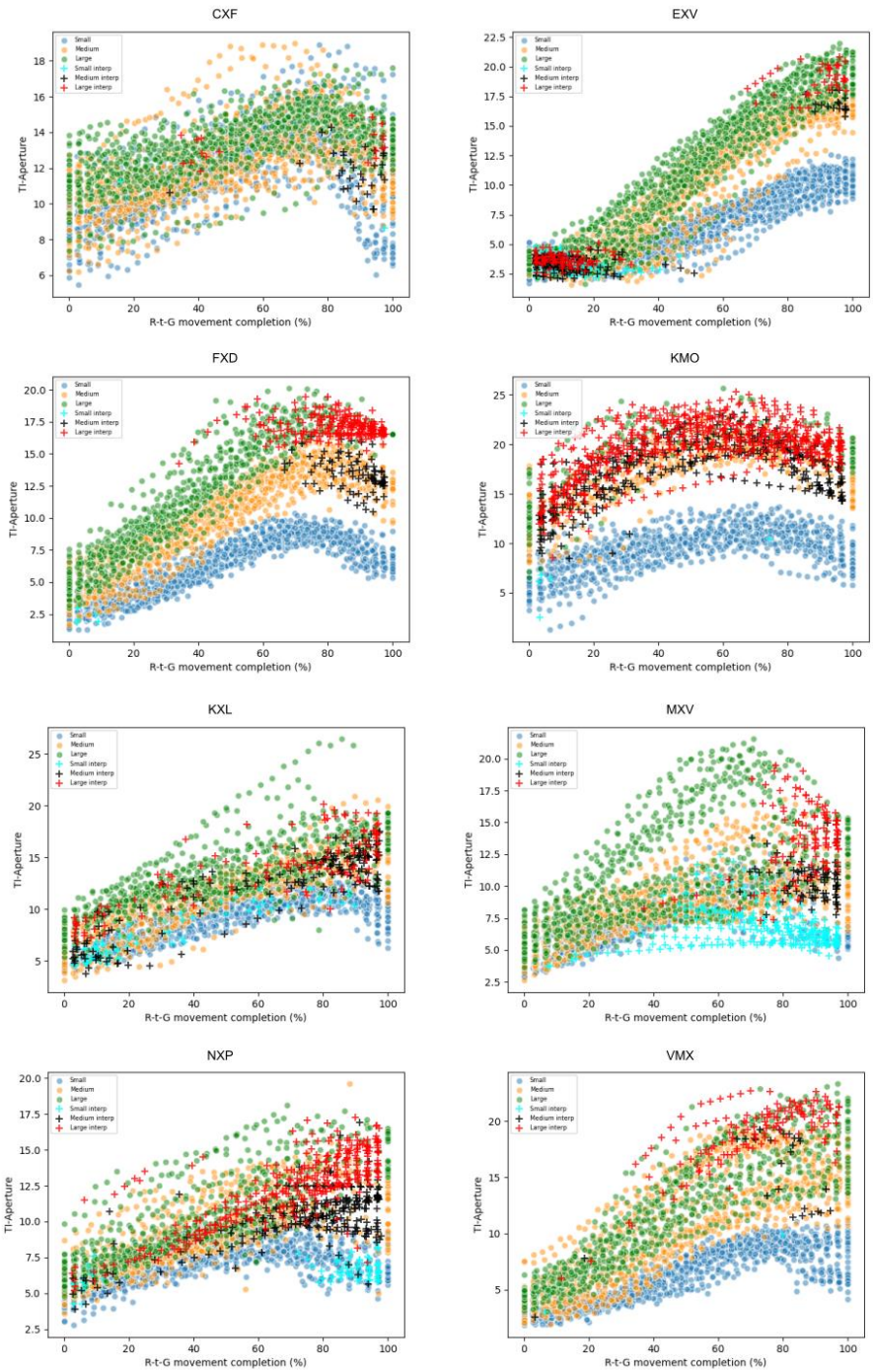

Figure 4.10: *Dataset A* - Interpolated aperture value scatterplots for the 8 participants and each object size. The '+' symbols represent the interpolated aperture values

The plots in Figure 4.10 show that the number of outliers in the dataset has been decreased due to the mentioned preprocessing steps, especially for the FXD and NXP participants.

It is noticeable that the dataset in Figure 4.6 and Figure 4.10 exhibits small overlap in the aperture values between small and large sized objects, while there

is a significant overlap between the medium sized object and the rest object sizes. In order to increase the separability of the objects, a second sub-dataset is also formed by removing the medium sized object. Figure 4.11 depicts the aperture values of the dataset for each completion interval, without the medium size objects.

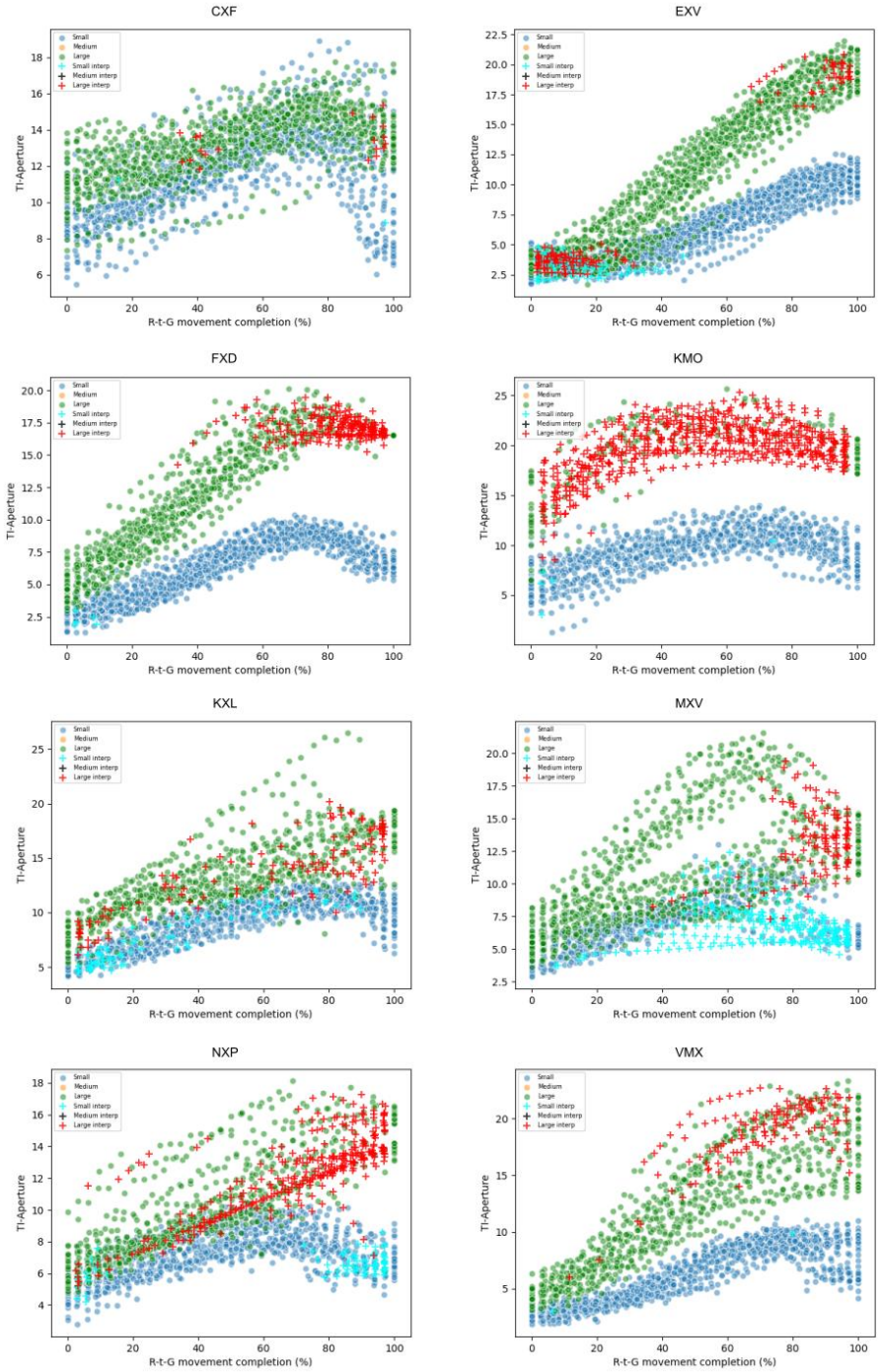

Figure 4.11: *Dataset B* - Interpolated aperture value graphs for the 8 participants, without medium size objects

Each resulting dataset is split into training and test sets, with ratios of *80%* and *20%* respectively, by randomly selecting trajectories from each type.

Both the training and the test datasets are normalized with *Min-Max scaling* in the range of *[-1, 1]* for the wrist x and y features and with subtracting the minimum aperture value from the aperture values, in order to end up with zero as the minimum aperture value. The scaling is selected to ensure that every wrist value spreads in the same range and the aperture values lie in a range of greater values' magnitude than those of the wrist, in order to provide more importance to this feature.

Totally, the information for the split datasets is the following:

- Dataset with 3 object sizes (small, medium, large): *Dataset A*
    - the training dataset contains *569* trajectories with approximately *18000* frames
    - the test dataset *144* trajectories with approximately *4600* frames.
- Dataset with 2 object sizes (small, large): *Dataset B*
    - the training dataset contains *400* trajectories with approximately *12000* frames
    - the test dataset *96* trajectories with approximately *3000* frames

4.3 Problem Formulation: MDP

The goal of exploiting the OP dataset is to construct a model that predicts grasping trajectories towards the cubes depending on their size, starting from pre-determined positions of the selected hand features. Ultimately the model must be able to recognize the size of the reached object. This configuration fits exactly to the concept and the arrangement of the InfoGAIL algorithm described in the previous section. This algorithm is explored to evaluate its efficiency in the hand trajectory and object prediction problem.

In order to obtain a concrete and working setup for InfoGAIL, the grasping trajectory prediction problem needs to be modelled into the format of MDP. The MDP follows the configuration of Section 2.1.2:

The environment is partially observable by the agent, so each state is an observation of a sliding window of *5* subsequent trajectory timesteps of the final dataset. On this account, each state includes *3* features (wrist x, wrist y and

aperture) of 5 consequent frames, so the dimensionality of the S space is 15. Intuitively, the observation at a specific timestamp t of the trajectory holds information about the mentioned features at frames with timestamps from $t-4$ to t.

A denotes the outcome of subtracting two consequent frame feature values of a trajectory (i.e., two consequent (x-wrist, y-wrist, aperture) tuples), so the action set space has dimensionality 3 (one action for the x-wrist, y-wrist and aperture, respectively). The actions are continuous.

The reward r for MDP is the sigmoid of the discriminator network output, as stated in the InfoGAIL specification. The reward $r \in (0,1)$ needs to be in this range in order to express the ability of the generator to produce policies that, in turn, produce the distribution of state-action pairs demonstrated.

ρ_0 is defined to be the set of all the initial trajectory states. It is also important to investigate the behaviour of the InfoGAIL algorithm in real-time scenarios where the object size needs to be predicted starting from intermediate states of the entire expert trajectories. In order to achieve this, ρ_0 set is enhanced with expert trajectories' intermediate states that serve as initial trajectory generation points, taken from 20%, 40%, 60% and 80% completion of each expert trajectory, in addition to the very starting state taken from 0% completion. An illustrative example is shown in Figure 4.12.

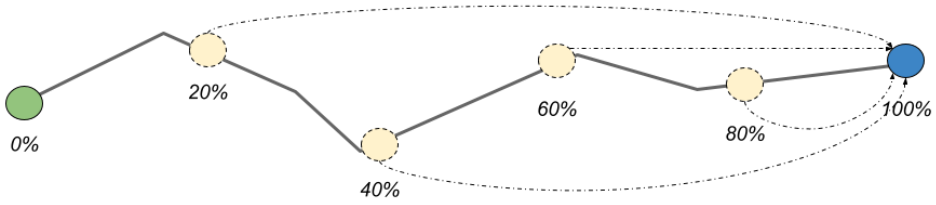

Figure 4.12: Partial sub-trajectories from expert trajectory states. 0%, 20%, 40%, 60%, 80% are the initial states of the expert trajectories

4.4 InfoGAIL Implementation

The InfoGAIL implementation for this thesis is based on the InfoGAIL version that is designed for TORCS environment [28]. The practical algorithm used in the TORCS experiment expands the details of the InfoGAIL specification in order to create a viable and optimized implementation, specifically on TRPO. The TensorFlow [29] framework is utilized for the implementation of the

generator, discriminator, posterior and value networks, as well as for the training and validation of the mentioned networks. The value network essentially approximates the value function V of each generated state-action pair, used by the advantage function of TRPO.

Figure 4.13 below depicts the implementation of the mentioned networks.

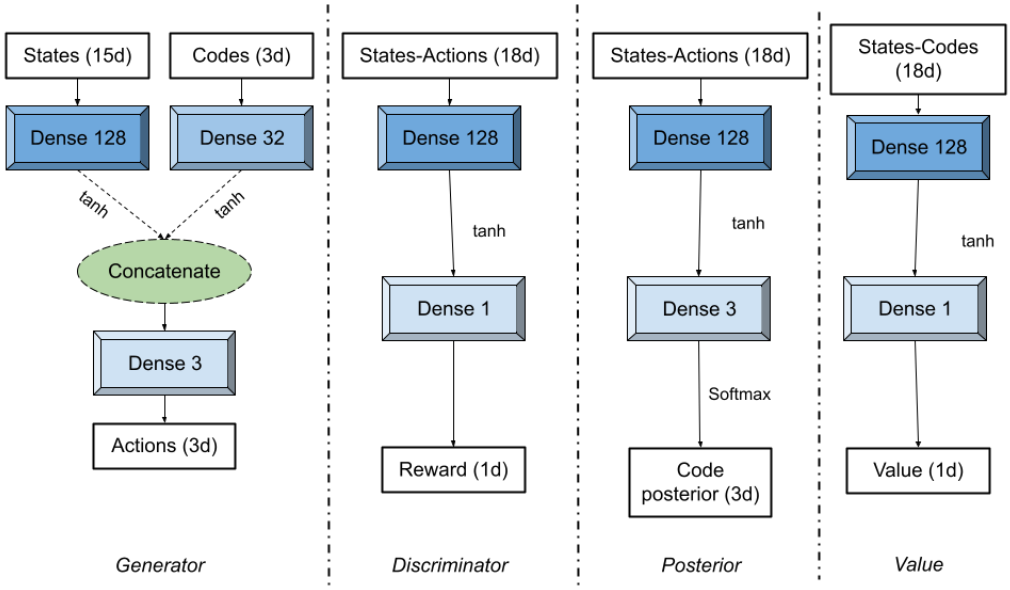

Figure 4.13: The Deep Learning networks that constitute the InfoGAIL models

The discriminator and the posterior networks consume a concatenated vector of the state-action pairs in the same dense layer. The value network accepts a concatenated vector of the states and latent codes, while the generator has two distinct hidden layers that accept states and latent codes respectively. Their outputs are then concatenated. The latent code property c is denoted by the one-hot vector of the object size, its values being matched with small, medium, large, respectively. The output of the posterior network is a vector with the softmax probabilities assigned to each object size. Every network uses the *tanh* activation function which outputs values in the range of *[-1, 1]*. Table 4.4 groups the loss functions and learning rates for the generator, discriminator, posterior and value networks.

Table 4.4: Loss functions and learning rates for the neural networks used in InfoGAIL

Neural Network	Loss Function	Learning Rate
Generator	$L(\theta_{old}, \theta)$ of TRPO	-
Discriminator	Binary cross-entropy	10^{-4}
Posterior	Categorical cross-entropy	10^{-5}
Value	Mean squared error	10^{-4}

In order to increase the efficiency of the posterior network predictions, a validation set is created by isolating some samples from the generated state-action pairs in order to be exploited by a *target posterior network*. When the main posterior network is trained, the target posterior network's weights are set based on the formula

$$targetQ_{weights} = 0.5 * mainQ_{weights} + 0.5 * targetQ_{weights},$$

where $Q_{weights}$ denotes the weights of the posterior network. The target is initialized with the same weights as the main posterior network. At each episode, the created validation set is exploited to calculate the posterior validation loss and the target network is used for predictions.

The loss function of the value network is the error between the approximated value function output and the surrogate reward of TRPO, which follows the formula

$$R(s,a) = -\log\left(sigmoid\big(D(s,a)\big)\right) + \sum_{j=1}^{m} \log Q(s,a) * c_j,$$

where $D(s,a)$ and $Q(s,a)$ are the outputs of the discriminator and the posterior networks respectively, *(s, a)* are the state-action pair inputs and c denotes the latent code one-hot vector. Thus, the one-hot vector has length $m = 3$ when using all the available object sizes.

The generator network is initially trained using the BC method. That step makes the generator network not to begin with random weights, gaining efficiency.

Table 4.5 provides information for training the BC.

Table 4.5: Training information for the BC method on the generator network

BC training and validation information	
Training episodes	100
Loss function	Mean Squared Error
Validation method	10-fold cross-validation [30]

The stochasticity of the generated policy for InfoGAIL is achieved by introducing Gaussian noise to actions when performing a state transition during trajectory generation, in the form of Gaussian standard deviation (std).

Table 4.6 groups the hyperparameters of InfoGAIL.

Table 4.6: Hyperparameters of InfoGAIL

InfoGAIL Parameters	
Discounting Factor γ	0.997
Gaussian std	0.008
TRPO advantage function λ	0.97
Maximum KL divergence δ	0.01
Latent code sample	400
Training episodes	10000
Training state-action pair sample	2000

5 Results

This section presents the training and testing results of InfoGAIL execution. Several plots are drawn along with the training process that show the losses of the generator, discriminator, posterior and value networks, which indicate their successful training, followed by plots and tables that evaluate the outcome.

5.1 Overview

In Section 4.2, two datasets are defined: Dataset A and Dataset B. Dataset A is the main dataset with all the object sizes and Dataset B is the dataset without the medium object. Furthermore, two feature sets are also defined: the set with all three features (x-wrist, y-wrist, aperture) and the set with only the aperture feature. The training and testing of the InfoGAIL algorithm is repeated four times, one for each feature set-dataset combination:

- *All features – all object sizes*
- *Only aperture – all object sizes*
- *All features – small, large objects (without medium)*
- *Only aperture – small, large objects (without medium)*

5.2 Training Results

At first, the training losses of the BC method are demonstrated. The BC training mean squared error for all object sizes is presented in Figure 5.1.

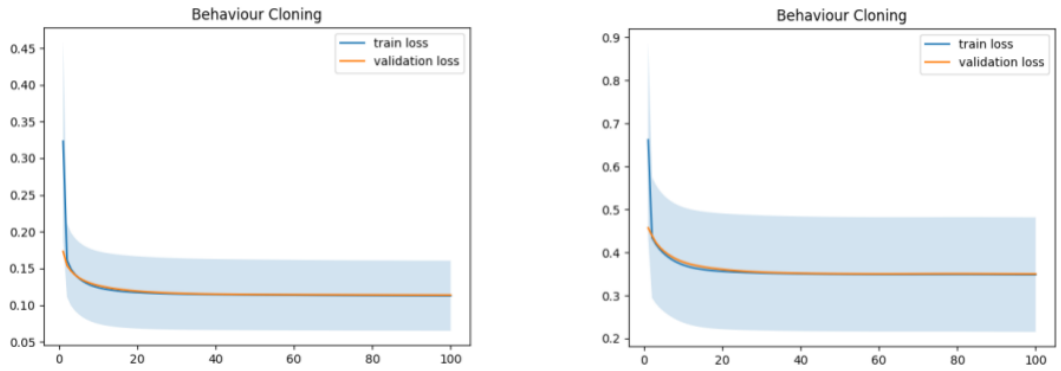

Figure 5.1: Training (blue line) and validation (orange line) BC mean squared error loss (y-axis) for 100 epochs (x axis) and for all object sizes. (Left) The loss when all features are exploited, (Right) The loss when only the aperture feature is exploited

The shaded area around the losses is the standard deviation of the training loss per epoch. The loss drops close to zero which indicates that the network learns the expert distribution. It is obvious that the standard deviation is larger on the right plot (aperture only), but the overall range of the aperture values is larger compared to the wrist coordinate values. This may result in bigger loss fluctuations. The loss output is quite expected, especially when exploiting all the features (left plot), due to the accuracy of the cross-validation that yields the best version of the network loss, which is *0.1175*.

Figure 5.2 shows the training BC losses for small and large object sizes (without the medium-sized cube).

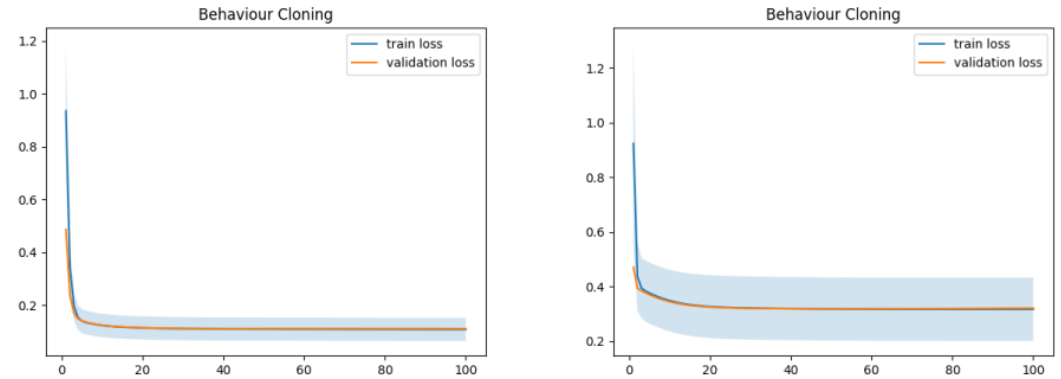

Figure 5.2: Training (blue line) and validation (orange line) BC mean squared error loss (y-axis) for 100 epochs (x axis), without the medium-sized cube. (Left) The loss when all features are exploited, (Right) The loss when only the aperture feature is exploited

These losses exhibit identical behaviour to the BC training with all three object sizes. Here, the standard deviation around the losses is smaller, meaning that the certainty for the model's prediction is increased.

Then, Figure 5.3 presents the training loss values of the discriminator, main posterior and target posterior networks for all object sizes.

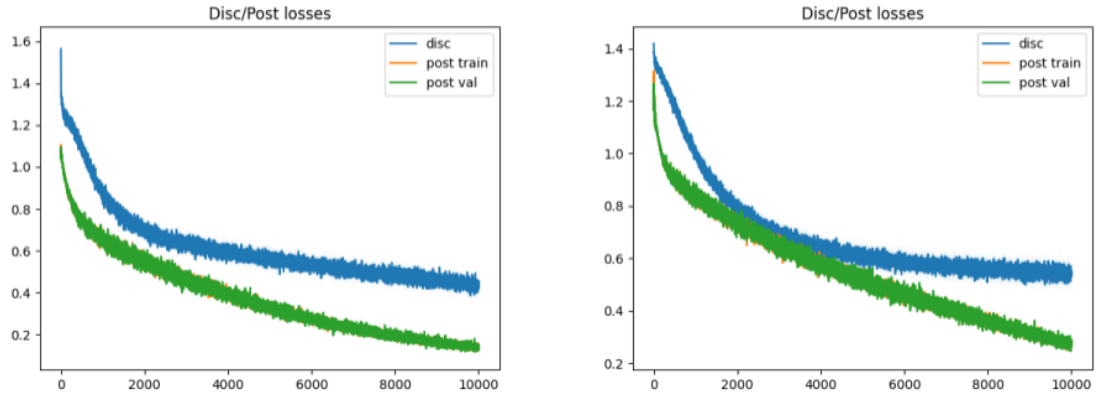

Figure 5.3: Discriminator (blue), main posterior (orange) and target posterior (green) network training losses (y-axis), for 10000 episodes (x-axis) and for all object sizes. (Left) The loss when all features are exploited, (Right) The loss when only the aperture feature is exploited

In both left and right plots, the discriminator loss starts to stabilize around *0.5* while the posterior loss continues to drop. This indicates that the costs assigned by the discriminator model on the state-action pairs generated by the generator (policy) network and on the expert state-action pairs are adequately balanced. The right plot for the aperture feature only shows better discriminator loss stabilization than the left plot for all the features. This outcome is better because the cost that is assigned by the discriminator on the generated state-action pairs approaches 1 (big cost) and the cost for the expert state-action pairs approaches 0 (small cost). Thus, the discriminator loss function yields a loss value that is close to the average value (0.5).

Furthermore, the target posterior network validation loss is almost identical to the main posterior network training loss. This means that the posterior model learns to generalize and does not overfit. Lastly, the loss values of the posterior networks (main and target) are both close to zero near the final episodes of training, meaning that the posterior model learns to distinguish between the object sizes that match the respective generated state-action pairs and the modes, which represent the object sizes predicted by the posterior model.

Next the discriminator and posterior loss values for only the small and the large object sizes, are presented in Figure 5.4.

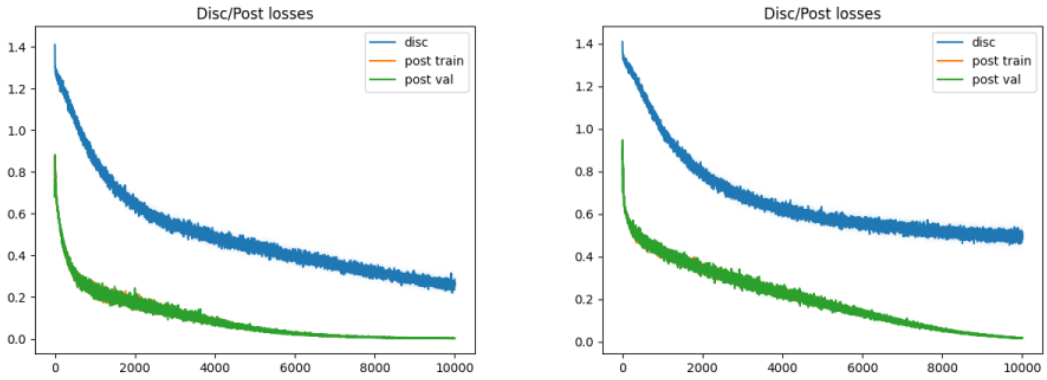

Figure 5.4: Discriminator (blue), main posterior (orange) and target posterior (green) network training losses (y-axis), for 10000 episodes (x-axis), without the medium-sized cube. (Left) The loss when all features are exploited, (Right) The loss when only the aperture feature is exploited

The explanation for these plots is similar to the respective loss plots when exploiting all the object sizes in Figure 5.3. Specifically, the loss plots of Figure 5.4 show that the posterior model learns to predict the object size faster, since the loss reaches o at the end of the training. Furthermore, observing the discriminator loss values after the 2000th episode, the loss value for only the aperture feature (right plot) seems to stabilize faster than when using all the three features.

Figure 5.5 depicts the mean aggregated surrogate reward of TRPO when exploiting all object sizes. The mean aggregated surrogate reward is calculated by summing the surrogate rewards of the state-action pairs of each generated trajectory and finding their mean value.

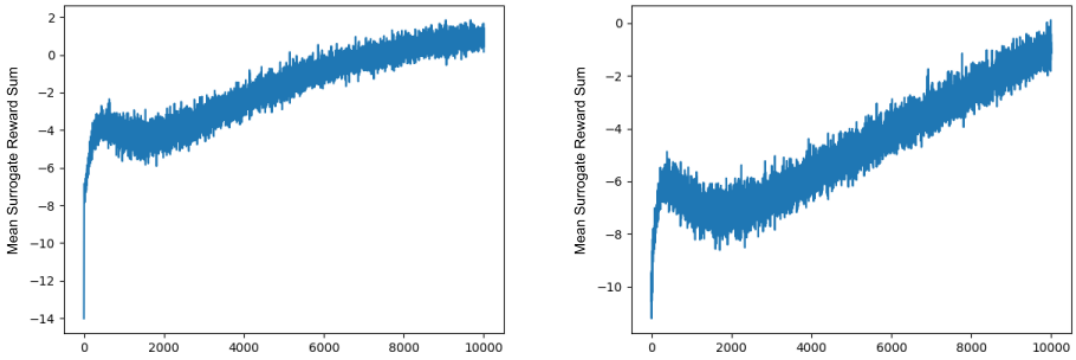

Figure 5.5: Mean aggregated TRPO surrogate reward (y-axis) for the generated trajectories of each episode (x-axis) for all object sizes, when all features are exploited (left) and when only the aperture feature is exploited (right)

Since the surrogate reward depends on the outputs of the discriminator and the posterior, it is expected that the surrogate reward value approaches zero, during the final training episodes. This value concerns the generated state-action pairs, so the trained discriminator is supposed to assign rewards close to 1 and the trained posterior assigns probabilities close to 1 for the object size that is thought to be the same as of the expert trajectory. As a result, the output of each logarithm function of the surrogate reward function approaches zero. The general case for zero surrogate reward value is when

$$\log\left(sigmoid\big(D(s,a)\big)\right) = \sum_{i=1}^{m} \log Q(s,a) * c_i,$$

which indicates that the reward of the discriminator is equal to the predicted probability for the corresponding expert object size. Lastly, negative values of the surrogate reward function imply big discriminator rewards and small posterior probabilities, while positive values imply small discriminator rewards and large posterior probabilities. The plots above show that there are potentially big rewards for small probabilities at the early stages of training and the surrogate value gradually increases. Thus, if the system learns the expert distribution, then the zero mean aggregated surrogate reward is explained by very small value differences between the posterior and discriminator outputs, which are both close to 1.

Figure 5.6 shows the surrogate reward loss plots when exploiting only the small and large object sizes.

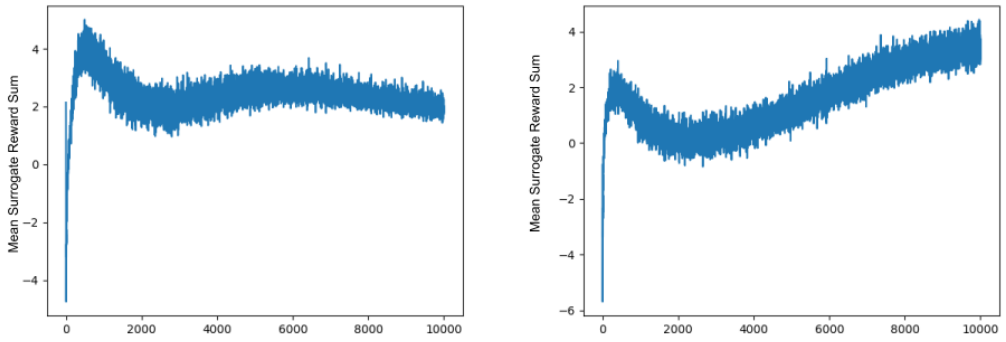

Figure 5.6: Mean aggregated TRPO surrogate reward (y-axis) for the generated trajectories of each episode (x-axis) without the medium-sized cube, when all features are exploited (left) and when only the aperture feature is exploited (right)

The surrogate reward on the right exhibits a larger value during the end of the training than the surrogate reward on the left. The plot on the right implies that the predicted probability of the posterior is larger than the discriminator reward, which explains the large mean surrogate reward for TRPO. The same explanation applies to the left mean aggregated surrogate reward, but here the difference between the output values of the two networks is smaller.

The value network loss for all object sizes is depicted in Figure 5.7. The loss function is calculated between the surrogate reward and the respective advantage values of the generated state-action pairs. The plots show stabilization around zero for both feature sets.

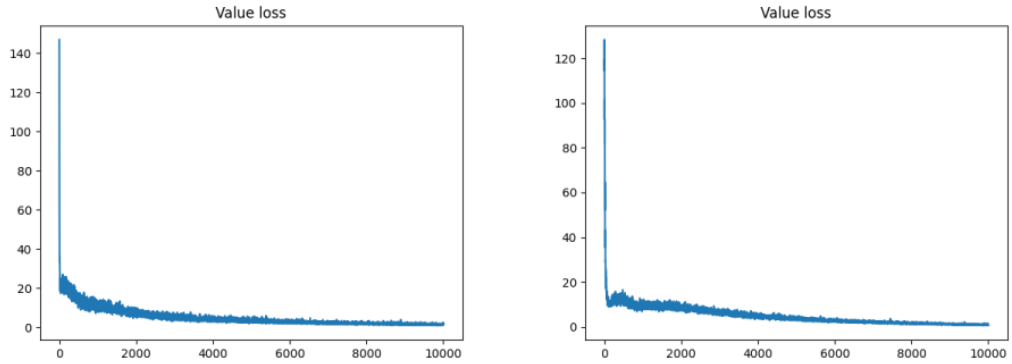

Figure 5.7: Training value network loss values (y-axis) for 10000 episodes (x-axis) and for all object sizes, when all features are exploited (left) and when only the aperture feature is exploited (right)

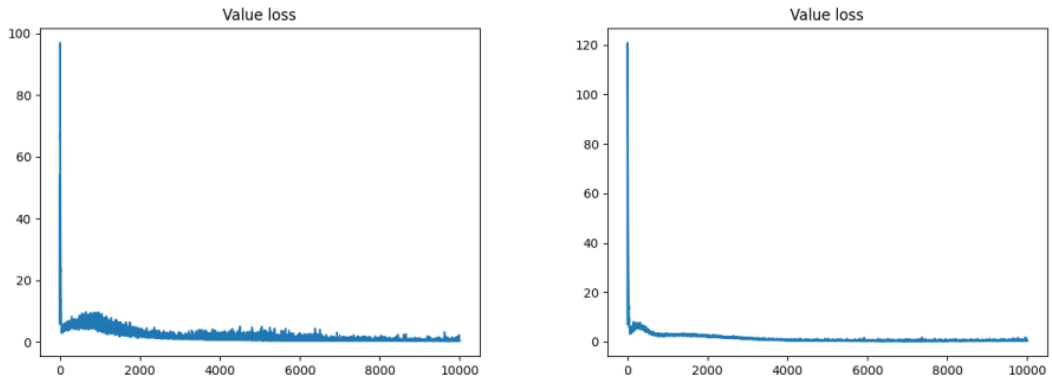

Figure 5.8: Training value network loss values (y-axis) for 10000 episodes (x-axis) and for small and large object sizes only, when all features are exploited (left) and when only the aperture feature is exploited (right)

Figure 5.8 shows the value network loss for small and large object sizes.

Here, the behaviour seems to be the same as the loss values in Figure 5.7, except that the variance of the loss values when exploiting only the aperture feature is smaller.

Finally, the surrogate TRPO loss presented in Figure 5.9 converges close to zero with almost identical variance for both plots. This fact implies that the generated policies do not change very much throughout the course of episodes.

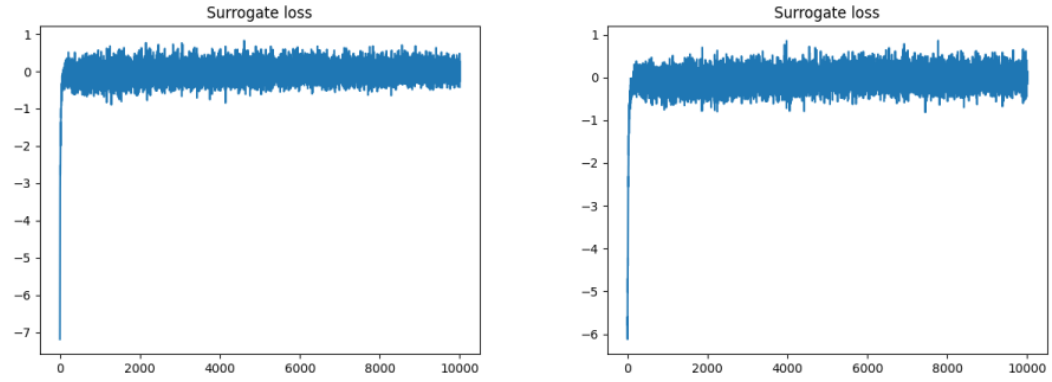

Figure 5.9: Surrogate TRPO loss (y-axis) for 10000 episodes (x-axis) and for all object sizes, when all features are exploited (left) and when only the aperture feature is exploited (right)

Figure 5.10 displays the respective surrogate loss plots, but without the medium-sized object. These plots exhibit the same behaviour as those in Figure 5.9.

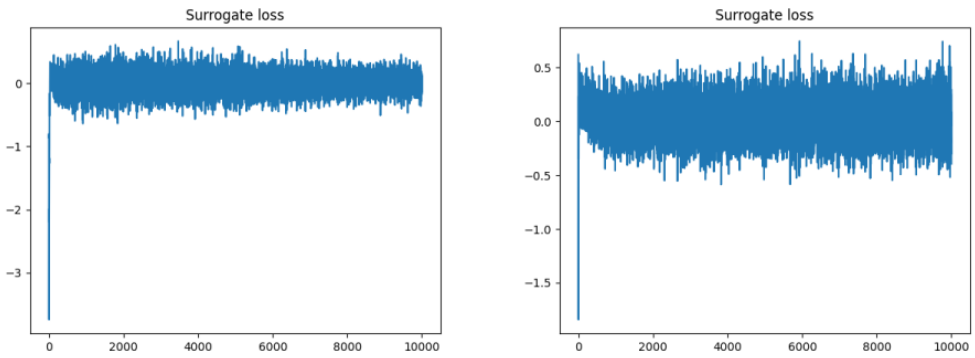

Figure 5.10: Surrogate TRPO loss (y-axis) for 10000 episodes (x-axis) without the medium-sized cube, when all features are exploited (left) and when only the aperture feature is exploited (right)

5.3 Testing Results

The InfoGAIL testing is adjusted depending on the question to be answered, defined in Section 1.

Firstly, all the expert state-action pairs are fed to the posterior model in order to determine whether it can distinguish the expert from the generated distribution. The actual object size for each expert pair is compared with the predicted size. Table 5.1 shows the results when exploiting all object sizes. In these tables, *All* refers to the InfoGail execution when exploiting all three features and *Aper* refers to the execution when exploiting only the aperture feature. *Support* denotes the total number of the state-action pairs for each object size

Table 5.1: Posterior accuracy over the training expert action pairs, for all object sizes

	Precision		**Recall**		**F1-score**		**Support**	
	All	***Aper.***	***All***	***Aper.***	***All***	***Aper.***	***All***	***Aper.***
Small	0.56	0.47	0.50	0.63	0.53	0.54	6271	6263
Medium	0.37	0.35	0.20	0.15	0.26	0.21	6136	6211
Large	0.42	0.42	0.67	0.53	0.52	0.47	5876	5812
Accuracy					0.46	0.43	18283	18286
Macro av.	0.45	0.41	0.46	0.43	0.44	0.41	18283	18286

The accuracy metrics present low scores, especially for the medium-sized object.

Table 5.2: Posterior accuracy over the training expert action pairs, without the medium-sized cube

	Precision		**Recall**		**F1-score**		**Support**	
	All	***Aper.***	***All***	***Aper.***	***All***	***Aper.***	***All***	***Aper.***
Small	0.69	0.65	0.69	0.67	0.69	0.66	6234	6227
Large	0.67	0.63	0.68	0.61	0.67	0.62	5879	5812
Accuracy					0.68	0.64	12113	12039
Macro av.	0.68	0.64	0.68	0.64	0.68	0.64	12113	12039

Table 5.2 shows the results when exploiting the small and large object sizes. This case demonstrates higher accuracy (68%) compared to the InfoGAIL execution with all the object sizes (46%).

Now, the accuracy of the posterior model for identifying the object size given an generated trajectory is examined. The approach for this answer is based on comparing the actual object sizes with the predicted object sizes. An object size that corresponds to the highest probability in the posterior model's softmax output, given a state-action pair, is considered as the predicted object size. The predicted object size $\hat{c}$ corresponding to the whole generated trajectory (and not only to distinct state-action pairs) follows the formula

$$\hat{c} = arg\max[\bar{P}(small), \bar{P}(medium), \bar{P}(large)].$$

$\bar{P}$ denotes the mean posterior probability of an object size for all the state-action pairs corresponding to a trajectory. It follows the formula

$$\bar{P}(c) = \frac{\sum_{i=1}^{n} Q(c|s_i,a_i)}{n},$$

where c is the object size (one-hot vector) and $Q(c|s,a)$ is the posterior probability for c, given a state-action pair (s, a). n denotes the number of the generated state-action pairs that constitute the trajectory.

The generated trajectories are produced by selecting initial states from the initial state set ρ_0 and, from each one, three trajectories are generated, one per object size (or two trajectories, depending on whether Dataset A or Dataset B is exploited). The mean probability $\bar{P}$ is calculated for the generated trajectory that is predicted to be closer to the respective expert trajectory (initiating from the

same starting point as the generated). The *Root Mean Squared Error* (RMSE) between the expert and the generated actions of the respective trajectories is calculated to determine the distance between the expert and generated trajectories. The generated trajectory with the smallest RMSE value is the closest to the expert.

If c denotes the actual object size of the expert trajectory, the comparison occurs between c and $\hat{c}$. The metrics for comparing these two quantities are *Precision, Recall, F1-score,* for every object size, and *Accuracy* and *Macro Average*, which express accumulated information. Precision and Recall are expressed as:

$$Precision_i = \frac{TP_i}{TP_i + FP_i},$$

$$Recall_i = \frac{TP_i}{TP_i + FN_i}.$$

TP, FP and FN stand for true positive, false positive and false negative, respectively. The true/false positive/negative values are defined by the following expressions:

- TP_i: number of occurrences where $c = \hat{c} = i$
- FP_i: number of occurrences where $c = j$ and $\hat{c} = i, \forall j \neq i$
- FN_i: number of occurrences where $c = i$ and $\hat{c} = j, \forall j \neq i,$

where i denotes the index of the object size list ($[Small, Medium, Large]$).

F1-score is calculated by

$$F1_i = \frac{2 * Precision_i * Recall_i}{Precision_i + Recall_i}.$$

Accuracy is expressed as

$$Accuracy = \frac{\sum_i TP_i}{\sum_i (TP_i + FP_i)},$$

while Macro Average is expressed as

$$Macro\ av. = \frac{\sum_{i=1}^{m} F1_i}{m},$$

where m stands for the length of the object size list.

Table 5.3 to Table 5.7 show the results for the trajectories starting from 0%, as well as for the trajectories from 20% to 80% of expert trajectories, and for all three object sizes. In these tables, *All* refers to the InfoGail execution when exploiting all three features and *Aper* refers to the execution when exploiting

only the aperture feature. Here, *Support* denotes the number of the trajectories in the testing dataset.

Table 5.3: 0% of the trajectory, for all three object sizes

	Precision		**Recall**		**F1-score**		**Support**	
	All	***Aper.***	***All***	***Aper.***	***All***	***Aper.***	***All***	***Aper.***
Small	0.86	0.59	0.40	0.81	0.54	0.68	48	48
Medium	0.33	0.41	0.46	0.35	0.39	0.38	48	48
Large	0.54	0.65	0.62	0.50	0.58	0.56	48	48
Accuracy					0.49	0.56	144	144
Macro av.	0.58	0.55	0.49	0.56	0.50	0.54	144	144

Table 5.4: 20% of the trajectory, for all three object sizes

	Precision		**Recall**		**F1-score**		**Support**	
	All	***Aper.***	***All***	***Aper.***	***All***	***Aper.***	***All***	***Aper.***
Small	0.74	0.68	0.29	0.75	0.42	0.71	48	48
Medium	0.28	0.41	0.38	0.33	0.32	0.37	48	48
Large	0.54	0.67	0.69	0.73	0.61	0.70	48	48
Accuracy					0.45	0.60	144	144
Macro av.	0.52	0.59	0.45	0.60	0.45	0.59	144	144

Table 5.5: 40% of the trajectory, for all three object sizes

	Precision		Recall		F1-score		Support	
	All	*Aper.*	*All*	*Aper.*	*All*	*Aper.*	*All*	*Aper.*
Small	0.77	0.70	0.35	0.83	0.49	0.76	48	48
Medium	0.26	0.47	0.23	0.33	0.24	0.39	48	48
Large	0.52	0.70	0.85	0.77	0.65	0.73	48	48
Accuracy					0.48	0.65	144	144
Macro av.	0.52	0.62	0.48	0.65	0.46	0.63	144	144

Table 5.6: 60% of the trajectory, for all three object sizes

	Precision		Recall		F1-score		Support	
	All	*Aper.*	*All*	*Aper.*	*All*	*Aper.*	*All*	*Aper.*
Small	0.84	0.68	0.56	0.88	0.68	0.76	48	48
Medium	0.33	0.48	0.25	0.29	0.29	0.36	48	48
Large	0.57	0.72	0.90	0.79	0.69	0.75	48	48
Accuracy					0.57	0.65	144	144
Macro av.	0.58	0.63	0.57	0.65	0.55	0.63	144	144

Table 5.7: 80% of the trajectory, for all three object sizes

	Precision		Recall		F1-score		Support	
	All	*Aper.*	*All*	*Aper.*	*All*	*Aper.*	*All*	*Aper.*
Small	0.73	0.56	0.69	0.92	0.71	0.70	48	48
Medium	0.41	0.40	0.38	0.25	0.39	0.31	48	48
Large	0.55	0.72	0.62	0.54	0.58	0.62	48	48
Accuracy					0.56	0.57	144	144
Macro av.	0.56	0.56	0.56	0.57	0.56	0.54	144	144

Observing the tables above, both columns (*All* and *Aper*) demonstrate almost equal average accuracy that reaches approximately 60%. As mentioned in Section 4.2, a significant reason for the low accuracy is due to the trajectories for medium-sized object, since it exhibits the lowest scores. Compared to the metrics for the expert state-action pairs in Table 5.1, these results demonstrate similar score values.

The tables below (Table 5.8 to Table 5.12) show the accuracy metrics when exploiting only the small and large object size.

Table 5.8: 0% of the trajectory, for small and large object sizes

	Precision		**Recall**		**F1-score**		**Support**	
	All	***Aper.***	***All***	***Aper.***	***All***	***Aper.***	***All***	***Aper.***
Small	0.79	0.77	1.00	0.96	0.88	0.85	48	48
Large	1.00	0.94	0.73	0.71	0.84	0.81	48	48
Accuracy					0.86	0.83	96	96
Macro av.	0.89	0.86	0.86	0.83	0.86	0.83	96	96

Table 5.9: 20% of the trajectory, for small and large object sizes

	Precision		**Recall**		**F1-score**		**Support**	
	All	***Aper.***	***All***	***Aper.***	***All***	***Aper.***	***All***	***Aper.***
Small	0.81	0.83	0.96	0.90	0.88	0.86	48	48
Large	0.95	0.89	0.77	0.81	0.85	0.85	48	48
Accuracy					0.86	0.85	96	96
Macro av.	0.88	0.86	0.86	0.85	0.86	0.85	96	96

Table 5.10: 40% of the trajectory, for small and large object sizes

	Precision		Recall		F1-score		Support	
	All	*Aper.*	*All*	*Aper.*	*All*	*Aper.*	*All*	*Aper.*
Small	0.90	0.88	0.92	0. 88	0.91	0. 88	48	48
Large	0.91	0.88	0.90	0. 88	0.91	0. 88	48	48
Accuracy					0.91	0. 88	96	96
Macro av.	0.91	0.88	0.91	0. 88	0.91	0. 88	96	96

Table 5.11: 60% of the trajectory, for small and large object sizes

	Precision		Recall		F1-score		Support	
	All	*Aper.*	*All*	*Aper.*	*All*	*Aper.*	*All*	*Aper.*
Small	0.86	0.83	0.90	0.94	0.88	0.88	48	48
Large	0.89	0.93	0.85	0.81	0.87	0.87	48	48
Accuracy					0.88	0.88	96	96
Macro av.	0.88	0.88	0.88	0.88	0.87	0.87	96	96

Table 5.12: 80% of the trajectory, for small and large object sizes

	Precision		Recall		F1-score		Support	
	All	*Aper.*	*All*	*Aper.*	*All*	*Aper.*	*All*	*Aper.*
Small	0.72	0.67	0.85	0.96	0.78	0.79	48	48
Large	0.82	0.93	0.67	0.52	0.74	0.67	48	48
Accuracy					0.76	0.74	96	96
Macro av.	0.77	0.80	0.76	0.74	0.76	0.73	96	96

By removing the medium sized cube and keeping the rest of the configuration as is, it can be observed that the evaluation metrics have been significantly improved. The accuracy lies between 76% and 91%, which means that the posterior network can confidently predict most of the object sizes of the

generated state-action pairs. However, the accuracy is higher compared to the case when the posterior model is fed with the expert state-action pairs (86%) and without the medium-sized object, in Table 5.2. This means that the generated state-action pairs are not quite the same as the expert ones.

6 Conclusions

In this thesis, the foundation of Reinforcement Learning and Imitation Learning has been discussed, in order to explore the InfoGAIL algorithm in a real-world human behaviour dataset that describes object size prediction from hand trajectories towards specific objects.

The implementation of the algorithm is designed to answer the major question whether the algorithm can effectively predict the right object size from various stages of the hand trajectory, while the minor question denotes whether the generated trajectories match these of the expert, i.e., how well the agent imitates the expert action in each state.

Below we attempt to provide answers to the questions poses in the introductory section of this thesis.

1. *Can the model identify the object size by observing the hand trajectories?*

The results indicated that the aperture feature has the greatest impact on the performance and there is a common outcome for both when exploiting all features (wrist-x, wrist-y and aperture) and the aperture only. However, the separability of expert trajectories is very important, since the results showed greater accuracy when the medium sized object was removed, which blurred the line between the trajectories regarding the two other object sizes (small and large cube). Generally, the outcome showed that the posterior network can adequately predict the right object mode when provided with the state-action pairs of the generated trajectories as input.

2. *What is the accuracy of identifying the object size?*

The best accuracy value reached 91% when the generated trajectory initiates from the point at 40% completion of the respective expert trajectory, and when exploiting only the small and large cubes and all features (wrist coordinates and aperture). Concerning the case when exploiting all the object sizes (small, medium and large), the best accuracy value reached 65% when the generated

trajectory starts from the point at 40% and 60% completion of the respective expert trajectory. This latter accuracy value is lower since the medium-sized object blurs the separability of the other two classes and the model did not perform adequately.

3. *Can the model mimic hand trajectories, given the object size?*

The results demonstrated that the model yielded an accuracy of approximately 70% when fed with the expert state-action pairs. Since the posterior model is trained with generated state-action pairs, the accuracy for the expert ones should be close to the accuracy mentioned in the answer to question 2. In that case, the generated trajectories would be closer to the expert.

In future work, this question can be highlighted in order to further explore and improve the action generation of the policy network so as the distribution of states reached by the agent to be closer to the expert distribution. This step may also aid to the improvement of the accuracy of the posterior network when giving the training state-action pairs as input. The potential steps that can be addressed in order to further clarify the issue are the following:

- Execute the InfoGAIL algorithm for the trajectories of each participant separately in order to observe and compare the result groups regarding the expert and generated trajectories.
- Experiment with different surrogate reward functions that favour specific participants in order to examine the relation between the nature of the generated trajectories and the crafted reward, with respect to the expert trajectories.
- Execute the InfoGAIL algorithm experimenting with different values of Gaussian standard deviation on the trajectory generation step, in order to examine and compare the expert state-action pairs with the generated state-action pairs.

7 References

[1] Y. Li, J. Song and S. Ermon, "InfoGAIL: Interpretable Imitation Learning from Visual Demonstrations," *arXiv preprint arXiv:1703.08840*, p. 14, 2017.

[2] M. Dagioglou, N. Soulounias and T. Giannakopoulos, "Object Size Prediction from Hand Movement Using a Single RGB Sensor," HCI International, 2022.

[3] L. P. Kaelbling, M. L. Littman and A. W. Moore, "Reinforcement Learning: A Survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237-285, 1996.

[4] R. Bellman, "A Markovian Decision Process," *Indiana University Mathematics Journal*, vol. 6, pp. 679-684, 1957.

[5] T. M. Moerland, J. Broekens, A. Plaat and C. M. Jonker, "Model-based Reinforcement Learning: A Survey," arXiv, 2020.

[6] J. Chen, B. Yuan and M. Tomizuka, "Model-free Deep Reinforcement Learning for Urban Autonomous Driving," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 2765-2771.

[7] K. Arulkumaran, M. P. Deisenroth, M. Brundage and A. A. Bharath, "Deep Reinforcement Learning: A Brief Survey," *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26-38, 2017.

[8] R. Sutton, D. McAllester, S. Singh and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in *Advances in Neural Information Processing Systems*, 1999, pp. 1057-1063.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," arXiv, 2013.

[10] I. Grondman, L. Busoniu, G. A. D. Lopes and R. Babuska, "A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 6, pp. 1291-1307, 2012.

[11] J. Schulman, S. Levine, P. Moritz, M. I. Jordan and P. Abbeel, "Trust Region Policy Optimization," arXiv, 2015.

[12] J. J.M., "Kullback-Leibler Divergence," in *Lovric M. (eds) International Encyclopedia of Statistical Science*, Berlin, Heidelberg, Springer, 2011.

[13] A. Hussein, M. M. Gaber, E. Elyan and C. Jayne, "Imitation Learning: A Survey of Learning Methods," *ACM Comput. Surv.*, vol. 50, no. 2, 2017.

[14] P. Abbeel and A. Y. Ng, "Apprenticeship Learning via Inverse Reinforcement Learning," in *Proceedings of the Twenty-First International Conference on Machine Learning*, New York, Association for Computing Machinery, 2004, p. 1.

[15] J. Ho and S. Ermon, "Generative Adversarial Imitation Learning," arXiv, 2016.

[16] A. Y. Ng and S. J. Russell, "Algorithms for Inverse Reinforcement Learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, San Francisco, 2000.

[17] S. Arora and P. Doshi, "A survey of inverse reinforcement learning: Challenges, methods and progress," *Artificial Intelligence*, vol. 297, no. 103500, 2021.

[18] F. Torabi, G. Warnell and P. Stone, "Recent Advances in Imitation Learning from Observation," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence (IJCAI-19)*, Texas, 2019.

[19] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg and P. Abbeel, "Deep Imitation Learning for Complex Manipulation Tasks from Virtual Reality Teleoperation," arXiv, 2017.

[20] S. Ross, G. Gordon and D. Bagnell, "A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, vol. 15, Florida, PMLR, 2011, pp. 627-635.

[21] M. Kelly, C. Sidrane, K. Driggs-Campbell and M. J. Kochenderfer, "HG-DAgger: Interactive Imitation Learning with Human Experts," in *2019 International Conference on Robotics and Automation (ICRA)*, 2019, pp. 8077-8083.

[22] A. Rajeswaran, V. Kumar, A. Gupta, G. Vezzani, J. Schulman, E. Todorov and S. Levine, "Learning Complex Dexterous Manipulation with Deep Reinforcement

Learning and Demonstrations," arXiv, 2017.

[23] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, "Generative Adversarial Networks," arXiv, 2014.

[24] B. Wang, E. Adeli, H.-k. Chiu, D.-A. Huang and J. C. Niebles, "Imitation Learning for Human Pose Prediction," arXiv, 2019.

[25] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," arXiv, 2014.

[26] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever and P. Abbeel, "InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets," arXiv, 2016.

[27] I. Culjak, D. Abram, T. Pribanic, H. Dzapo and M. Cifrek, "A brief introduction to OpenCV," in *2012 Proceedings of the 35th International Convention MIPRO*, 2012, pp. 1725-1730.

[28] [Online]. Available: https://github.com/YunzhuLi/InfoGAIL.

[29] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu and X. Zheng, "TensorFlow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265-283.

[30] P. Refaeilzadeh, L. Tang and H. Liu, "Cross-Validation," *L. Liu, M.T. Özsu (eds.), Encyclopedia of Database Systems*, 2016.

[31] S. Wang, D. Jia and X. Weng, "Deep Reinforcement Learning for Autonomous Driving," arXiv, 2018.

[32] N. Ratliff, J. Bagnell and S. Srinivasa, "Imitation learning for locomotion and manipulation," pp. 392 - 397, 2008.