CRANFIELD UNIVERSITY

Daniel Peter Woodhall

Imitation learning for performing a task.

School of Aerospace, Transport and Manufacturing
Robotics

MSc
Academic Year: 2020 - 2021

Supervisor: Dr. Seemal Asif
08/2021

CRANFIELD UNIVERSITY

School of Aerospace, Transport and Manufacturing
Robotics

MSc

Academic Year 2020 - 2021

Daniel Peter Woodhall

Imitation learning for performing a task.

Supervisor: Dr. Seemal Asif
08/2021

This thesis is submitted in partial fulfilment of the requirements for the degree of Robotics MSc
(NB. This section can be removed if the award of the degree is based solely on examination of the thesis)

ABSTRACT

The work undertaken in this project outlines an imitation learning method that can be used to perform simple manipulation tasks from a variety of contexts using sparse state-only observations. Firstly, a dataset was constructed that was used to train a context-adaptation model, with the dataset samples containing raw video feeds of a pushing task taken from a diverse range of viewing angles. The context-adaptation model that was constructed can produce a translated video within an agents own context from a singular expert demonstration after training. The translated video can in turn be used to train an agent within its own environment using reinforcement learning methods. Deep-deterministic policy gradients were used as a reinforcement learning method to assess the validity of this method on the “Pusher-3DoF” environment within the RLlab simulator, comparing the results to current state-of-the-art methods in contextualisation for imitation learning. The proposed method produced a 6.67% performance increase in simulation when compared to the next-best performing contextualisation method. A major current limitation of the proposed method is the inability to contextualise to “extreme” demonstrations, such as the presence of distractor objects, without explicit prior knowledge about the context. Future developments to the work could be done to incorporate a task-embedding architecture to allow the model to generalise to varying tasks as well as contexts. Most importantly, the performance of the model should be verified in a real-world use-case.

Keywords:

Observation, context-adaption, contextualisation, domain-adaptation, CNN.

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Seemal Asif, for her continued and invaluable support and guidance throughout the project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

RL	Reinforcement learning
TCP	Tool-centre point
MDP	Markov decision process
VB	Value-based method
DQL	Deep-Q learning
PG	Policy-gradient method
VPG	Vanilla policy-gradient
TRPO	Trust-region policy optimization
AC	Actor-critic method
DDPG	Deep-deterministic policy-gradient
SAC	Soft actor-critic method
IL	Imitation learning
BC	Behavioural cloning
DPL	Direct policy learning
IRL	Inverse reinforcement learning
ILHD	Imitation learning from human demonstration
ITO	Imitation through observation
GAIL	Generative adversarial imitation learning

1 Introduction

Robotic manipulation tasks often require extensive and time-consuming manual waypoint programming for effective performance. One alternate approach to this is teleoperation, where the robot is controlled remotely by a human operator (Xu B, 2020). However, due to the need of a human operator, teleoperation is often undesirable in real-world scenarios due to the lack of ability to automate processes, thus increasing time for execution and deployment. Machine-learning and deep-learning methods can be used to facilitate the use of autonomous robotic manipulation; however, the current approaches have several major issues. There are two main methods that can be applied for the autonomous manipulation of a robotic manipulator, reinforcement learning (RL) and imitation learning (IL) (Kober J, 2010). RL involves the random exploration of an agent within its environment to construct a control policy, where the policy is optimised based upon a reward function that needs to be manually defined. RL has the distinct advantage of being able to iteratively solve complex problems without any prior knowledge of the task being completed; however, the reward functions constructed for RL methods can often be extremely time-consuming to construct. Conversely, IL provides a model with expert demonstrations of a task being completed, usually within the agents own domain, to allow the agent to infer a control policy from observing the states and actions taken in the expert demonstrations. From these demonstrations, a function that correlates current observations to optimal actions is produced. Both IL and RL have been used in conjunction with deep-learning models to facilitate complex skill learning with varying degrees of success.

Most current IL methods utilise extremely data-intensive observations consisting of high-dimensional states and actions. This high-dimensionality results in IL methods being extremely computationally expensive. Additionally, dataset generation for such a problem is usually excessively time-consuming, requiring either teleoperation, kinaesthetic learning, or skeleton-tracking of a human demonstrator making it largely unapplicable for most use-cases (Hussein A, 2017). Moreover, IL methods currently show a limited ability for adaptation to

context and domain changes between the demonstrator and the agent, with some methods having no ability to generalise at all.

The approach in this work constructs a context-adaptation model which is used to allow for demonstrations from the expert's context to be translated into the agent's context. This conversion allows a feature distribution from the expert demonstrations to be calculated and translated into the context of the agent. RL can then be performed, using the translated demonstrations to train the agent to perform a task within its own context.

1.1 Problem Definition

A major issue in IL currently is the inability of models to contextualise to a diverse range of contexts and domains, resulting in unrobust control policies that can only perform on demonstrations from within the agents own context and domain. Methods to combat this have shown limited success and have also not been tested on a diverse range of contexts; usually being limited to just two contexts, 1st and 3rd person viewpoints. The ability of an IL method to contextualise to a more diverse range of contexts would facilitate the deployment of IL in real-world scenarios with unstructured and variable environments. In addition, IL methods mostly use high-dimensional computationally expensive datasets which require excessive amounts of time to train and deploy. The ability of an IL method to contextualise would allow for control policies to be constructed from observing demonstrations in differing contexts and domains, such as using a human demonstrator.

Thus, research was undertaken into current IL methods and their limitations, proposing a solution to the issue of contextualisation in robotic manipulation tasks. The method aims to perform context-adaptation in IL using sparse state-only observations to combat the issues of computationally expensive data and contextualisation simultaneously, gathering a dataset from outside the agent's context using raw video footage. The removal of the actions from the observations increases the complexity of implementation, although the produced model can be trained significantly faster. This method will be contrasted to current state-of-the-art IL methods to assess its comparative validity and accuracy.

1.2 Research Aims & Objectives

The overarching aim for this research is to produce a proof-of-concept IL model that can accurately emulate the demonstrations of an expert from differing contexts using a model-free approach with sparse state-only observations to perform a basic robotic manipulation task. Multiple approaches will be researched and investigated prior to implementation to assess which is optimal for the use-case of robotic manipulation. The objectives for this research are:

- Research IL methods to determine which is optimal for the use-case of robotic manipulation in varying contexts.
- Produce a dataset of expert demonstrations that can be used for training a contextualisation architecture and IL model.
- Produce a contextualisation architecture that can adjust for different contexts and viewing angles of a given task.
- Produce a model that can perform IL using a model-free approach from sparse state-only observations in differing contexts.

2 Literature Review

2.1 Reinforcement Learning

RL is a process where an agent explores an environment, taking actions within the environment whilst observing the resulting states and rewards/losses of the actions taken to construct an optimal policy, (Sutton RS, 2018). Through an iterative process of trial and error, an optimal, or near optimal, policy can be constructed. Once this policy has been constructed, the task can be repeated using the constructed policy, π. There are many variations of RL techniques, each with distinct advantages dependent on the given task or state-space; usually being split based on either discrete or continuous state/action spaces.

A major issue with RL currently is the generalisation of tasks. Policies constructed from RL, generally, only perform well on the specific task that they were trained on. This means that a new policy needs to be constructed for tasks that have even small deviations from the original task, (Cobbe K, 2019). Although most RL policies can only be used for one task, the algorithms that are used to construct

the policies can usually be used ubiquitously, provided that the algorithm is designed to function in the given state/action space.

In the case of this project, the agent is a robotic manipulator, with the actions being represented by the tool-centre point (TCP) of the manipulator, and the state being defined by a video-feed of the task being completed.

2.1.1 Markov Decision Process

A Markov Decision Process (MDP) is a discrete-time stochastic control process that is often used to define RL problems with a semi-randomised nature; (MM., 2018). A standard MDP consists of a tuple as follows:

- S: the observable states
- A: the actions taken
- P_{as}: the probability of transition when taking an action, a, in a state s
- R/L (S, A) → $\mathbb{R}$: a map from the set of states, S, and the set of actions, A, to a reward or loss.

P_{as} attempts to replicate the dynamics of the environment in which the MDP is acting, thus it is required only for model-based reinforcement learning methods. MDPs can be considered a sequential process by observing the state, action, and calculated reward for a given episode; where an episode refers to a finite set of actions required to complete a given task. Defining each episode as containing a single state, action, and reward/loss, the sequential process can be defined as the following set of tuples: $\{(s_0, a_0, r_0), \dots, (s_t, a_t, r_t)\}$, (Wei Z, 2017). Reinforcement learning attempts to maximise the overall reward (or minimise the overall loss) of the process across all episodes, summing the reward (or loss) across each episode to find a cumulative total; shown in equation (1).

$$R_{total} = R_0 + \cdots + R_{t-1} \quad \textbf{1}$$

Equation (1) encompasses the simplest solution to calculating the cumulative reward/loss of a sequential process defined by a MDP. If a sequential process contains many steps, or is continuous, it can be undesirable to assign the same weights to the later episodes in the process; due to the earlier episodes usually baring the most weight on the effectiveness of the RL policy. As such, a discount

or multiplicative factor can be introduced into the equation, giving diminishing rewards later into the process, or greater rewards earlier into the process, (Littman ML, 2013). The choice of whether to use a discount or multiplicative factor is heavily affected by the task being performed, with shorter tasks favouring a multiplicative factor. An example of a discounted cumulative reward is shown in equation (2), where γ represents the discount rate.

$$R_{total} = R_0 + \sum_{t=1}^{T} \gamma^t R_t \quad \textbf{2}$$

The looping nature of this equation allows for a value of the constructed policy, π, to be calculated, giving a relative 'value' to the agent for each action in each state at a given time-step. This allows for the policy to be evaluated numerically based upon the current state of the agent in the environment, as shown in equation (3). The overall policy, incorporating the decided action at a given time-step can then be defined by equation (4).

$$V_\pi(s_t) = R_t + \gamma V_\pi(s_{t+1}) \quad \textbf{3}$$

$$Q_\pi(s_t, a_t) = R_t + \gamma Q_\pi(s_{t+1}, a_{t+1}) \quad \textbf{4}$$

The values in equations (3) & (4) are generally approximated based on previous experience of the agent, through monitoring the resulting reward/loss relative to the state that has been reached, (Sidford A, 2018).

2.1.2 Regularization

A significant issue with current RL techniques utilizing MDPs is the instability they are subject to due to high variance, which is exacerbated in high-dimensional state/action spaces. Regularization is a technique used in machine-learning methods that aims to reduce the variance in iterative procedures, at the expense of some bias being incorporated into the solution. Spatial regularization is a form of regularization that can be introduced into a model to reduce the variance as

well as reduce overfitting during training and is commonly used in image-manipulation models, (Zhu, 2017). Temporal regularization is like spatial regularization in that it reduces variance whilst also incorporating a factor based on the similarity in value predictions across episodes during RL, calculated based on the interaction of the Bellman equation (Geist M, 2019).

2.2 Reinforcement Learning Methods

2.2.1 Value-Based Methods

Value-based methods (VB) are methods used to solve a MDP that predict the value of an action, as opposed to the action itself; as in equation (3). Most VB methods are only able to perform well in discrete action and state-spaces, due to the need to predict the value of all actions in a state to determine the optimal action to take in each state, (Arulkumaran K, 2017). Standard value-based methods calculate a temporal-difference error based on the cumulative reward/loss, along with the previous and next-step estimates of the system; equation (5) demonstrates the temporal-difference loss for the Q-learning algorithm, where δ represents the temporal-difference error. This error can then be incorporated into equation (4) to give a consistent update at the end of each episode or time-step during training.

$$\delta = R_t + \gamma \max_{a_{t+1}} Q_\pi(s_{t+1}, a_{t+1}) - Q_\pi(s_t, a_t) \quad \textbf{5}$$

Deep Q-learning (DQL) is a VB method that can perform in large action/state spaces by attempting to minimise the error instead of updating an estimate based on a calculated error. DQL, as the name suggests, incorporates a neural-network to perform the function approximation instead of a standard tabular method, allowing for the value of the action to be predicted in non-discrete state-spaces, (Hester T, 2018). Through this, the problem is transformed such that the aim of the network is to minimise the temporal-difference error; shown in equation (6), where θ represents the parameters of the neural network, (Gu S, 2016).

$$\min_{\theta}\left(R_t + \gamma \max_{a_{t+1}} Q_\pi(s_{t+1}, a_{t+1}) - Q_\pi(s_t, a_t)\right) \quad \textbf{6}$$

By using this approximation, a combination of both discrete and continuous inputs can be given to the network, allowing for more complex problems to be modelled, such as in robotic manipulation. Using a single estimator for DQL often leads to overestimation of the value of an action, (H., 2010). Thus, a Double Q-learning method was proposed where two Q functions are stored, equation (4), with each Q function being updated using the value from the opposing Q function to calculate the next state or value, (Van Hasselt H, 2016). Some methods also incorporate a target network in place of a second DQL, where the target network calculates a value based on the action selected by the DQL to assess the validity of the chosen action in the current state (Yu C, 2020).

2.2.2 Policy-Gradient Methods

Policy-gradient methods (PG) calculate a distribution of probabilities, encompassing each action to select an optimal action in a state, instead of estimating the value for a given action. This results in a stochastic policy being constructed based on the probability distribution, with the highest probability action being chosen to be performed in each state. Due to the stochastic nature of PG methods, they usually give much cleaner updates to the policy, as opposed to VB methods where there can be sudden changes in a policy which can lead to jerk during robotic manipulation tasks, (Agarwal A, 2020). As the name suggests, PG methods utilise a form of gradient descent to converge towards an optimal policy over time, using the long-term cumulative reward as a guideline for the effectiveness of the overall policy. PG methods often show improved performance in continuous state and action-spaces when compared to discrete state and action-spaces, (Peters J, 2006). In addition, PG algorithms can be either model-based or model-free, with no significant drop in performance when exchanging between the two.

One major issue of PG methods is their on-policy nature, meaning that they are required to forget previous data to avoid the introduction of bias during training of

the function estimator. Whilst VB methods for tabular data representations are guaranteed to reach the global maximum, PG methods, as is the same in gradient descent regression problems, can reach a local maximum meaning that they may never produce a policy that reaches the global maximum cumulative reward (Agarwal A, 2020).

PG methods are commonly used in the field of robotics, such as for the learning of bipedal walking and robotic control. The extensive use of PG methods in robotics, as opposed to VB methods, is largely due to their ability to perform well in continuous state and action-spaces. However, as mentioned, they can produce suboptimal policies if they converge towards a local maximum instead of the global maximum, (Silver D, 2014).

PG methods aim to construct a policy that maximises the estimated cumulative reward over the entire state-action procedure. The vanilla policy gradient (VPG) method is an on-policy PG method shown in the following equations, where equation (7) demonstrates the general policy optimization function and equation (8) shows the gradient update that is used to update the policy parameters over time, where a_t denotes the learning rate and $\nabla_\theta J|_{\theta=\theta_t}|$ denotes the policy gradient estimator. Equation (9) demonstrates the calculation to find the gradient of $J|_{\theta=\theta_t}|$, where π_θ represents a policy with parameters θ, A^{π_θ} represents the advantage function of the policy, and T represents the trajectory; trajectory is interchangeable with episode.

$$J|_{\theta=\theta_t}| = E\left\{\sum_{t=0}^{T} \gamma R_t\right\} \quad \textbf{7}$$

$$\theta_{t+1} = \theta_t + a_t \nabla_\theta J|_{\theta=\theta_t}| \quad \textbf{8}$$

$$\nabla_\theta J|_{\theta=\theta_t}| = E\left\{\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)\right\} \quad \textbf{9}$$

Trust-region policy optimization (TRPO), first proposed in, (Schulman J, 2015), is a PG method that aims to further prevent sudden parameter updates that give drastic policy changes using a divergence constraint, KL, which restricts the change in policy update over each iteration. TRPO has shown to have significantly better performance than VPG in robotic manipulation tasks such as pick and place or pushing tasks. TRPO alters the objective function in equation (9) into a loss function as shown in equation (10). However, using equation (10) to train the policy in TRPO can lead to destructive policy updates, thus the KL divergence constraint is introduced to prevent the deconstruction of the policy through the limitation of the update size, equation (11), where δ represents the maximum policy size change and $\pi_{\theta old}$ represents the previous policy. This results in policy changes only occurring in trusted regions where the KL requirement is satisfied.

$$L_{TRPO} = E\{\log \pi_\theta(s_t, a_t) A^{\pi_\theta}(s_t, a_t)\} \quad \textbf{10}$$

$$\max_\theta (E\{D_{KL}[\pi_{\theta old}(s_t, a_t) || \pi_\theta(s_t, a_t)]\}) \leq \delta \quad \textbf{11}$$

Although TRPO is an improvement on policy construction when compared to VPG, the introduction of the KL constraint introduces a large amount of computational complexity, thus making it more viable for simulation based RL where computation time isn't as important.

2.2.3 Actor-Critic Methods

Actor-critic methods (AC) operate using two connected networks, using an actor which is trained to estimate and learn the policy, and a critic which is trained to assess the validity of the actions chosen by the actor network. As such, an AC method can be seen as a combination of both a VB and PG method, where the critic utilises a VB method calculating the value of an action chosen by the actor, and the actor uses a PG method to estimate a policy using a temporal-difference error (Parisi S, 2019). AC methods show significant improvements over the use

of solely VB or PG methods as they allow for action in continuous state-space whilst not being subject to the policy instability present in the simpler policy generation methods. A general overview of the actor-critic method architecture is shown in Figure 1.

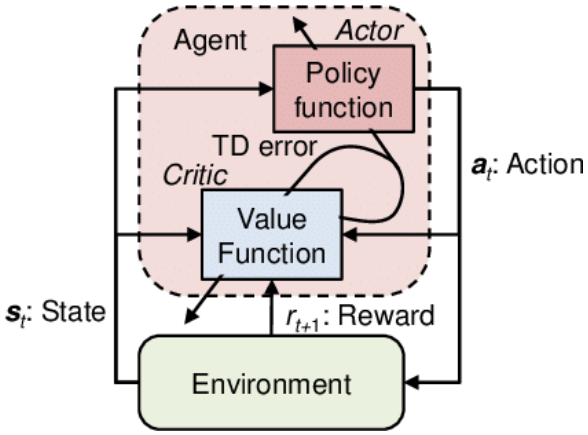

Figure 1. Actor-Critic Method Architecture (Fuji T, 2018)

One such AC method which has been used recently for robotic control with great success is Deep-Deterministic Policy Gradients (DDPG). DDPG utilises a PG method combined with a DQN VB method, where the PG is used as the actor to generate a policy in a continuous action space and the DQN acts as the critic, with the input to the DQN being the policy parameters generated by the PG network (Kumar A, 2018). Due to the DQN being a function of the PG policy, the gradient of the overall network can be calculated by using the chain-rule on the policy parameters generated from the PG over a small batch of states (N); equation (12).

$$\nabla J \approx \frac{1}{N}\sum_{t=1}^{N} \nabla Q(s_t, \mu s_t|\theta^Q)\nabla\mu(s_t|\theta^\mu) \qquad \textbf{12}$$

Figure 2 demonstrates the basic architecture of the DDPG method, where the critic takes the actions from the actor as inputs, along with the state-observations to construct a Q-value for the given set of actions, with the Q-value in turn being used to optimise the actions outputted by the actor.

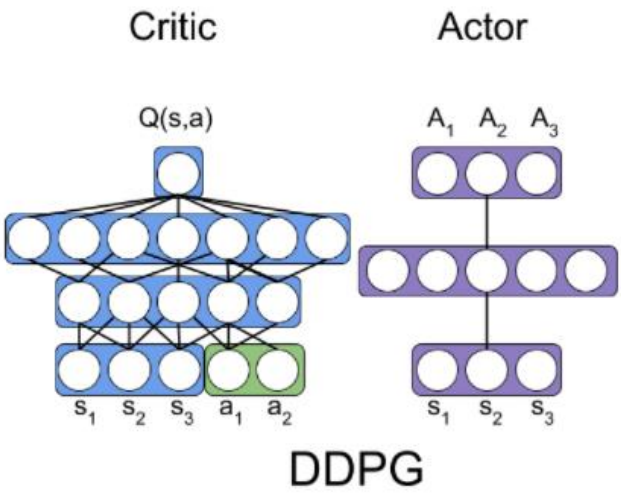

Figure 2. Deep deterministic policy gradient (DDPG) (Karunakaran, 2020)

Another AC method which has risen in popularity in robotic applications in recent years is the Soft Actor-Critic method (SAC). As well as aiming to maximise the cumulative reward, such as in DDPG, SAC also aims to increase and maximise the entropy during policy generation (Haarnoja T, 2018). By maximising the entropy, SAC reduces the chance of the actor converging towards a local maximum, encouraging the exploration of new, previously unused actions to generate a more optimal policy. Equation (13) demonstrates the objective function used, incorporating the entropy term which is denoted as ε.

$$J(\theta) = \sum_{t=0}^{T} E\{R_t + \gamma\varepsilon(\theta(\cdot\,|s_t))\} \qquad \textbf{13}$$

For the use-case of imitation learning, the increased entropy during training for SAC is likely not desirable. This is due to it encouraging deviation from the experts' demonstrations during training, which could result in a policy that gives a higher reward result but is not very similar to the given expert demonstration. This is entirely dependent on whether the goal is to accurately emulate the expert demonstrations, or to use the expert demonstrations to construct an overall more optimal policy for the task.

2.2.4 Overview

VB methods such as DQL, are commonly used to evaluate the result or predicted value of an action. However, VB methods require the number of times a given state has been reached to be counted, meaning they only work in small, discrete state-spaces and thus are largely unapplicable to the use-case of robotic manipulation (J., 2013). In non-discrete or extremely large state/action spaces, it is necessary to calculate a continuous approximation of the value function which is updated at each time-step. This can be achieved in VB methods by generalising the states, such that previously visited states can be used to inform decisions on current states which are similar, although this usually leads to suboptimal policy generation.

PG methods can perform in continuous state and action-spaces with no significant downside, unlike VB methods. However, PG methods are susceptible to producing suboptimal policies due to the gradient descent method used, meaning they can reach a local maximum instead of the global maxima for the cumulative policy reward. There are methods to combat this, such as by incorporating an entropy coefficient into the equation that aims to encourage exploration of new actions in the environment; although without a validation method the incorporation of entropy leads to significantly longer training times whilst the model explores many different options (Cen S, 2020).

AC methods incorporate both a VB and PG method that allows for the disadvantages of each to be combatted by each other's actions. The PG method performs as the actor in the network, producing the policy parameters that act directly in the environment, whilst the VB method acts as a critic evaluating the value of the generated actions from the PG's policy. As such, AC methods can perform in continuous state and action-spaces whilst reducing the risk of producing a suboptimal policy through the incorporation of a critic.

For this use-case, and robotic applications in general, VB methods would not be usable without some significant alterations, due to the continuous state and action-states inherent in robotic control. PG methods would be applicable; however, they give a greater risk of reaching a local maximum without the use of

a critic in the network. AC methods would likely give the most desirable results, with the incorporation of the critic, as well as the entropy term allowing for finer control whilst reducing the risk of generating a local maxima policy. Of the AC methods investigated DDPG appears to be the most applicable, since the goal is to accurately emulate the expert demonstrations, as opposed to producing an overall optimal policy (Cen S, 2020).

2.3 Imitation Learning

Imitation learning (IL) can be considered a subset of reinforcement learning where a policy is optimised by attempting to emulate an expert demonstration, rather than through randomised exploration of the environment by an agent. IL is usually used instead of RL when the teaching process is especially challenging, such as in environments with sparse rewards, or when it is easier to demonstrate the desired behaviour than to construct a reward function. IL techniques have been shown to vastly improve the policy construction time of an agent and can lead to more optimal policies being constructed; although they can also lead to suboptimal policies if the demonstrations given are not optimal. However, due to the nature of IL, a large dataset of expert demonstrations is usually required to construct an optimal policy (Hussein A, 2017).

There are often differences in the way that the learner and teacher in IL actuate state changes, thus leading to further generalization issues. In IL, data can be split into demonstrations and imitations. Demonstrations are collected using solely the odometry available to the agent, meaning data is not recorded directly from the expert, such as in teleoperation or kinaesthetic learning of a manipulator. Conversely, imitations are collected using the reference frame of the teacher, meaning some context-translation or mapping from the learner to the teacher is necessary for the learning process (Wu YH, 2019).

2.3.1 Behavioural Cloning

The most basic form of IL is behavioural cloning (BC) which aims to replicate the expert policy through supervised learning. BC works by dividing expert demonstrations into state-action pairs, then applying supervised learning to

determine a mapping between the states and the actions, however this requires the assumption that all variables are independent and identically distributed (i.i.d) (Torabi F, 2018). Due to the i.i.d assumption, BC methods encounter significant problems; in a MDP, an action performed in a state induces the next state, thus breaking the i.i.d assumption. This assumption also leads to the errors made in each state being cumulative, which can result in an extremely large error when integrated over time even if the error at each time-step is relatively small. This makes BC methods largely inapplicable for robotic control.

2.3.2 Direct Policy Learning

Direct policy learning (DPL) incorporates an interactive demonstrator, allowing for the expert to be queried during operation to give extra data if the model reaches a previously unseen state. Through this, the model can iteratively improve itself using the expert queries until it converges. Notwithstanding, this can be extremely time-consuming depending on the method used for the initial data collection (Chemali J, 2015).

DPL first constructs an initial policy from the expert demonstrations. The policy is then run iteratively, monitoring the trajectories created from the initial policy to estimate the state. For each state visited the expert can be queried to give additional information on an optimal action to perform in the given state, allowing for the policy to be updated (He H, 2012). This process is repeated until the constructed policy converges, and an optimal policy is constructed. Through this method, the agent can remember not only optimal actions in each state, but also suboptimal actions based upon its previous mistakes, ensuring that it does not repeat suboptimal actions again. Through a method known as Policy Aggregation the constructed policies from each iteration can be combined, looking at the optimal actions from each policy to construct the global maximum policy through a process known as geometric blending.

DPL has significant advantages over BC methods as it is not subject to the i.i.d assumption. However, it does require an interactive expert to be present during training of the agent which can drastically increase the time for training.

2.3.3 Inverse Reinforcement Learning

Instead of aiming to find the optimal policy, inverse reinforcement learning (IRL) aims to construct a reward function for the agent's environment from the expert dataset. Once a reward function has been constructed, the optimal policy can be found by choosing actions in each state that maximise the given reward, hence the name inverse reinforcement learning. The policy, once constructed, can then be compared to the expert demonstrations to assess its effectiveness in-situ (Hadfield-Menell D, 2016).

IRL can utilise either a model-based or model-free method, differing based on the reward function and the way in which the reward function is generated. For model-based IRL, the dynamics for state transition in the agent's environment are required to be known; and usually the state and action-spaces must be relatively small. This approach requires a full episode of reinforcement learning to be run to update its policy. In addition, model-based methods construct a linear reward function that is disseminable allowing for it to be more easily understood when compared to model-free approaches (Das N, 2020).

Model-free IRL does not require a dynamic model of the environment and constructs a reward function using deep neural networks. The action and state-spaces for model-free IRL are also able to be larger or continuous which gives a significant advantage over model-based methods for robotic manipulation (E., 2018). As model-free IRL works in continuous or large state-spaces the policy is updated after each time-step of RL, although it is required to be run in either a simulation replicating the agent's environment or the environment itself.

For both model-based and model-free methods, the reward function may not always be optimal or may lead to more computation than is necessary to generate an optimal policy. This is due to there usually being multiple reward functions analogous to the optimal policy. As previously described, entropy can be incorporated into the model to encourage additional exploration, selecting the trajectory with the highest entropy value as the optimal policy, however this is not ideal for emulation of demonstrations (Ziebart BD, 2008).

2.3.4 In-Domain Imitation

In-domain imitation refers to IL where the expert demonstrations are collected in the domain and context of the agent itself. In-domain imitation is considered one of the easiest methods of IL to implement. In the case of robotic manipulation, this involves gathering a dataset of the agent performing the task through either kinaesthetic learning or teleoperation, monitoring the joint angles or TCP throughout. Gathering the dataset in such a way removes the need for contextualisation or domain-adaptation techniques to be performed and gives good performance on simple tasks in a structured environment with no variability (Englert P, 2013).

A major issue with in-domain imitation is the inability to generalise to even small changes in the domain or context. The agent must remain the same the entire way through training and implementation, meaning a model trained through in-domain imitation is not applicable to any other agent. Additionally, it requires extensive camera calibration prior to each training or testing session. For this use-case, in-domain imitation would not be applicable as the imitation learning model is required to be able to generalise to differing contexts.

2.3.5 Imitation Learning from Human Demonstration

Imitation learning from human demonstration (ILHD) is an emerging field, especially in robotics, allowing for agent's to intuitively determine a policy from a set of demonstrations performed outside of their own context. Context translation and domain-adaptation in ILHD are two of the major problems currently being worked on, as human demonstrations are often captured from a different viewing angle to the agent (e.g. 1st person viewpoint versus 3rd person viewpoint). Additionally, the introduction of a human demonstrator adds further complication due to the adaptation of the domain since a human arm performing a task will not have the same representation as a robotic manipulator (Ravichandar H, 2020). The following section will refer specifically to ILHD in the setting of robotic manipulation.

2.3.5.1 Direct Kinematic Mapping

One method to perform ILHD is to construct a kinematic mapping between the human arm and the manipulator being used. Through this mapping, the movements of the human arm can be tracked in real-time using pose-estimation software and an RGB-D stereo camera (such as the Microsoft Kinect or Intel Realsense), or a motion-capture system. A significant issue with performing kinematic mapping is the translation between 7-DoF (human arm) and 6-DoF (robotic manipulator). Most methods that incorporate kinematic mapping omit all but the first 3-DoF of the human arm, to create a more direct mapping to the manipulator. However, the omission of the final 4-DoF in the human arm leads to an inability to effectively manoeuvre the wrist joint of the manipulator (Zhao L, 2016).

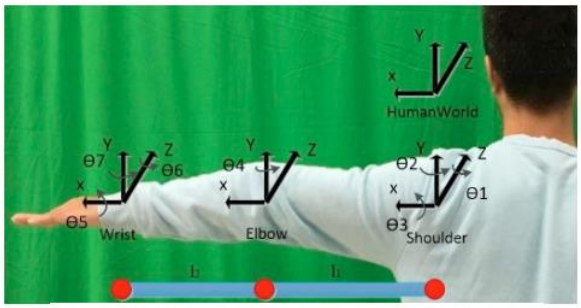

Figure 3. 3-DoF mapping of human arm (Wang Z, 2020)

To control the wrist joint of the manipulator through mapping, the kinematic mapping method often requires additional sensors placed on the hand and fingers. One method, (Liarokapis MV, 2013), utilised 3 flex sensors attached onto the wrist to actuate the manipulators wrist through the flex sensor outputs. However, due to the error inherent in the flex sensors, this method led to severe jerk in the operation of the wrist joints.

Other methods, (Wang Z, 2020) (Chen Z, 2020), used a ‘perceptive neuron’ motion capture system to capture the 7-DoF of the human arm more accurately during operation. Through this method, actuation of the wrist joint of a UR5 manipulator was achieved; although, again, still gave issues with fine control. Through this, the UR5 was able to perform basic ‘flexion’ and ‘extension’ wrist

motions, mimicking the motion of a human operator. However, it was not able to perform fine control of the wrist joint in real-time. Additionally, the above three methods, (Liarokapis MV, 2013) (Wang Z, 2020) (Chen Z, 2020), cannot be considered true ILHD methods, as they work through the construction of a direct mapping without the application of any learning element into the system.

More developed methods, (Seleem IA, 2020), incorporate a kinematic mapping, along with a learning element incorporated through a VR motion-capture system to track a task being performed by the human operator. Through this method, simple 'pushing' manipulation tasks were able to be performed, using the kinematic mapping for the general motion of the manipulator whilst using the motion-capture system to incorporate some semantic understanding of the task being performed within the environment. This method was relatively successful for simple tasks, although it was unable to perform more complex multi-stage tasks due to the domain and context-adaptation between the human-arm and the manipulator. Although the context-adaption issue is somewhat alleviated through kinematic mapping, the proposed system struggled to infer an effective policy during training due to the differences in the environment in which the task was being performed.

2.3.5.2 Meta-Learning

There have been several methods developed to attempt to combat the issue of domain and context-adaptation. One such method is meta-learning, where the expert demonstrations that are collected from the human demonstrator also have a corresponding demonstration in the agent's own domain and context. From this, meta-learning can be used to allow for a model to learn the translation between the human and agent's domain and context (Finn C, 2017) (Li D, 2018). This has shown good success in robotic manipulation tasks, although still has some significant disadvantages. Since each human demonstration requires a corresponding agent demonstration, the dataset collection time can be significantly longer, requiring teleoperation or kinaesthetic operation of the agent. To speed up the data-generation process, simulations have been used with varying effectiveness. The use of a simulator allows for the datasets to be

generated and the policy to be trained relatively quickly, however the learnt policy does not always perform well when transferred to the real-world due to further domain and context-adaptations in sim-to-real transfer (Yu T, 2018).

One method, (Bonardi A, 2020), trained a robotic manipulator to perform basic pushing tasks in MuJoCo-sim based off human demonstrations, where the human demonstrations were also collected in simulation using a 24-DoF representation of a human arm. By utilising domain-adaptive techniques where the background textures, as well as the textures of the agent/human arm, were varied during the meta-learning process, the policy generated was able to perform relatively well when transferred to the real-world; even when varying the environment in which the task was being performed. However, this method required significant development to define the tasks within the simulation, meaning the time for data-generation is only fractionally shorter when compared to real-world data-generation. This, along with the other issues of sim-to-real transfer, make this method undesirable for generalised or multi-stage complex tasks in robotic manipulation.

2.3.5.3 Imitation through Observation

Imitation through Observation (ItO) refers to systems that learn a policy through state-only demonstrations, without any information regarding the actions of the demonstrator (such as kinematic mappings) in the environment (Torabi F, 2018). Several methods for ILHD focus almost entirely on the issue of context and domain-adaptability. Time-contrastive networks, (Sermanet P L. C., 2017), allow for context and domain-translation to be performed through an additional encoding architecture; with the task being performed in this work being a pouring task. During dataset generation, the human demonstrations were captured from multiple viewpoints, 1st person and 3rd person, allowing for the encoding architecture to be trained to contextualise between the two viewpoints of the task. Through this, the encoding was able to generalise between the two viewpoints, determining what in each video-feed was similar (e.g. the task being performed) and what was differing (e.g. the non-important background features). By training the encoding architecture first and incorporating this into the ILHD problem, the

agent was able to effectively mimic the motions of a human-arm, even though the context between the two systems was entirely different. However, due to the way the dataset was generated, the system could only generalise between two contexts, 1st person and 3rd person viewpoints (Sermanet P L. C., 2018).

Task-embedding control networks (TecNets) are a method for ILHD proposed in the work of (James S, 2018) that incorporate a task-embedding architecture that enables a robotic agent to generalise between multiple similar tasks. In this method, expert demonstrations were gathered through teleoperation of a Kinova Mico 7-DoF manipulator. The TecNet architecture takes the expert demonstrations as an input, constructing a plane that contains all expert demonstrations, with demonstrations that are similar being closer together on the plane. Each task is then assigned a “sentence” which refers to a unique code for each sub-set of tasks. This allows for the agent to query the TecNet model during operation, based on the initial frame of the video feed, to assess which task it is attempting to perform. From there, the policy from the most-similar task in the plane is used as a basis to perform the task, performing several iterations until the task is successfully completed. This method showed a great ability to generalise between variations of similar pick and place tasks, being able to correctly identify which object was required to be manipulated with an accuracy of 82%.

Other systems, (Liu Y, 2018) (Torabi F, 2018), proposed an encoding and decoding architecture which was trained on a dataset containing raw video-feeds of tasks being performed in varying contexts and domains. The dataset includes both simulated video-feeds and real-world videos to construct an encoding architecture that can contextualise between different viewpoints, domains, and even basic tasks. Once the encoding architecture is fully trained, using varying viewpoints and contexts, the architecture can be given a static context image of the same task in a different context or domain. Through this, the encoding can construct a translated video of the task being completed, which can in turn be used to train an agent to perform the task within its own context using RL methods, with TRPO and VPG being used in the above works.

The context translation model in (Liu Y, 2018) gathers a feature representation that can track demonstrated behaviour from expert demonstrations into the context of the agent using a convolutional neural network (CNN). Two RL methods were then used to create an optimal policy and trajectory of the agent within its own context, mimicking the actions of the expert demonstrations. The RL method chosen was dependent on whether the training was performed in simulation or in situ; TRPO for simulation, VPG for in situ. The encoding architecture utilises a CNN with two encoders working in tandem, one for the demonstration video and one for the static context image. The decoder used takes the outputs of both encoders, performing a translation on the context image to construct a translated video in the desired domain. When training in simulation, 4500 expert demonstrations were given for the encoding performing on a pushing task. Whilst in the real-world, 135 expert demonstrations were used for, again, a pushing task. The large discrepancy in sample-size of the expert demonstrations between simulation and real-world is partially due to the ease of data-generation in simulation, as the entire process can be automated. However, the real-world encoding was shown to have no significant performance increase when going over 150 expert demonstrations. This work showed significant improvements over the work in (Sermanet P L. C., 2018) due to its ability to generalise to a greater number of viewing angles and contexts. However, the tests undertaken in the work of (Liu Y, 2018) did not demonstrate contextualisation or generalisation between more extreme viewing-angles, as well as not considering the possibility of distractor objects or extreme variations in background within the agents' environment.

2.3.6 Overview

BC methods are largely inapplicable for this use-case due to their inability to contextualise or adapt to varying domains. Similarly, DPL methods are inapplicable as they do not meet the objective relating to sparse state-only observations, requiring additional demonstrations during training in-situ. IRL methods for robotic manipulation show relatively good success, however they can produce a suboptimal reward function due to there being many analogous

reward functions to any set of state and actions. Additionally, IRL has significantly reduced performance when interacting with sparse state-only observations and is not able to contextualise or generalise to differing environments. In-domain imitation is also not applicable to this use-case as it is entirely unable to generalise, even from minor changes in the environment.

ILHD methods appear to show the greatest performance on robotic manipulation tasks with sparse state-only observations, especially when there are significant context changes in an unstructured, variable environment. Direct kinematic mapping shows promising results for motion-mimicking; however, it can only be considered an IL method if the mapping is produced using deep-learning or machine-learning techniques. Additionally, direct kinematic mapping cannot perform on sparse, state-only observations as it requires skeleton tracking for translation between the human and robot domain.

The work of (James S, 2018) was able to effectively generalise between variations of a basic pick and place task, however it required a corresponding demonstration at usage time, meaning it requires a semi-interactive demonstrator for effective use. Contrasting this, the work in (Liu Y, 2018) was able to contextualise between several viewing angles and contexts of a basic task.

The introduction of a dynamic model into the imitation learning method results in only one agent being able to be used for each model (Chemali J, 2015). Thus, a model-free approach is preferable to model-based for this application, as it allows for one model to generalise between multiple agents. Since using sparse observations is a requirement of the project, the model should be able to perform from state-only observations, which also significantly reduces the complexity of both dataset generation and training; albeit, increasing the complexity of model construction.

Overall, the approach of (Liu Y, 2018) appears to be the most applicable to this use-case due to its ability to successfully generalise between several viewing angles using state-only observations. However, its performance was not assessed on extreme viewing angles, environments with noisy backgrounds, or environments with distractor objects. As such, further experimentation into this

method should be undertaken to assess its viability across a wider range of contexts.

2.4 Novelty

Previous works into combatting the contextualisation issue inherent in imitation learning have shown good results across a small number of non-diverse contexts. However, the ability of the models to generalise and perform in cluttered environments has not been assessed, along with the ability of the models to generalise to a larger number of more diverse contexts. The work undertaken in this research will use the context-translation architecture proposed in (Liu Y, 2018) as a basis, adapting the methodology and data-generation methods to be able to generalise to a more diverse range of contexts, along with the assessment of alternative RL methods for the actuation of the agent.

3 Methodology

The method undertaken in this work was based on the work of (Liu Y, 2018), where the imitation learning approach consists of an agent (robotic manipulator) that watches expert demonstrations of a task in a diverse range of contexts before emulating the demonstrated behaviour in its own context. In this example, context relates to the attributes of the environment in which the task is being performed such as viewing angle, object positions and colours, and the background environment. Each demonstration consists of a set of 80 observations such that $\{D_1, D_2, \dots, D_T\} = \{[o_0^1, o_1^1, \dots, o_{79}^1], [o_0^2, o_1^2, \dots, o_{79}^2], \dots [o_0^T, o_1^T, \dots, o_{79}^T]\}$, where o_t is an observation comprised of a semi-observed MDP that contains the dynamics (P_{as}) modelled by $P_{as}(s_{t+1}|s_t, a_t, \omega)$, the distribution of observations at step t modelled by $p(o_t|\omega, s_t)$, and the policy $p(a_t|\omega, s_t)$; where ω is the context of the observation, s_t is the MDP state, a_t is the unobserved action. The actions being unobserved is important due to the requirement of the system to perform using state-only observations, with the observations in this instance referring to raw video footage demonstrations.

The IL method proposed must be able to accurately establish what information from the expert demonstrations correlates to information within its own context.

This is necessary to ensure that the model can effectively translate demonstrations between varying contexts. To solve the issue of context-adaptation, a model can be trained which is able to translate demonstrations across contexts. To train the context-adaptation model, roughly time-synced demonstrations from varying contexts must be collected and used as training data.

The IL method must also be able to accurately provide actions to the agent that allow for semantically similar emulation of the expert demonstrations. RL methods would facilitate this through iterative training, although the issue of determining a reward function then arises. A distance measurement can be taken between the expert demonstration and the observation in the agents' state at each timestep, however determining which distance measurement to use can be challenging. The Euclidean distance between the two observations could be used as a reward function; however, this often results in closely matching pixel magnitudes between the two observations without the agent performing the task in a semantically similar way to the expert (Chemali J, 2015). To combat this issue the context-adaptation model can be incorporated into the RL method, constructing a reward function based on the squared Euclidean distance from the translated features of the expert demonstration and the observations in the agent's context within the context-adaptation model. Since the context-adaptation model can determine the variations in the observations between the two contexts, it can produce a trajectory that is more semantically like the demonstration; as opposed to using solely the Euclidean distance from the decoder outputs of the context-adaptation network (Torabi F W. G., 2019).

To ensure that the aims and objectives were met, a stepwise approach was taken for the implementation of the imitation learning architecture. Before any model construction began, a dataset was required to be generated, ensuring that the dataset contains a disparate range of contexts to validate the model's ability to generalise. Once the dataset has been generated, a context-adaptation architecture was produced. This architecture was selected due to its ability to generalise to different contexts using state-only observations as well as using a

model-free approach for the agent's actuation which allows for multiple agents to be used for the same model. Experimentation was undertaken on the context-adaptation architecture to assess the effect on the accuracy, namely the use of differing optimizers (ADAM and AdaDelta) and the introduction or removal of specific loss functions from the architecture during training. After the context-adaptation model has been constructed, trained, and validated, RL can be performed with the agent in its own context and environment using the translated video as the expert demonstration.

Due to time-constraints a physical demonstration of the system using a real-world robotic manipulator was unable to be gathered. As such, an additional dataset was generated using the "Pusher-3DoF" environment in RLlab, based off MuJoCo simulator, to assess the performance and feasibility of the RL element of the IL solution using the context-adaptation architecture with DDPG as the RL algorithm.

3.1 Dataset Generation

Three datasets were generated during the research, consisting purely of raw video footage of various pushing tasks being completed. All videos were stored as .avi files using FFmpeg as an encoder. The two real-world datasets were gathered at 10Hz over an 8 second period, for a total of 80 frames, with a resolution of 48x48 on Logitech C270 webcams. A tool was used for object manipulation to remove the domain-adaptation between the human arm and the robotic manipulator. No initial camera calibration or strict camera placement was necessary for the dataset generation due to the use of the context-adaptation architecture. The general dataset generation setup is shown in Figure 4, demonstrating the two webcams setup to record the expert demonstrations from varying viewing angles. A desk-mounted camera stand was used to mount one of the webcams to allow for demonstrations with elevated viewing angles to be captured. The second webcam was placed in varying positions at a slightly lower elevation than the first webcam to ensure diversity in the captured footage. Additionally, the second webcam was used for all extreme viewing angle examples, such as from directly in-front of the task being performed. The object

to be manipulated was placed in approximately the same position each time, with exact placement not being necessary due to the ability of the context-adaption architecture to generalise.

Figure 4. Dataset generation setup, with webcam placement highlighted in red

OpenCV was used for image manipulation as well as for the writing and storing of the video files during data collection. The Logitech C270 webcams output at 1280x720, which was scaled down to 48x48 as it was inputted to the video writing function. As the function writes the frames to the selected file location, it monitors the number of frames being stored, leaving the function, and saving the video file after 80 frames to ensure all videos are a consistent length. The code for the data collection in the real-world can be found in Appendix A.3.

The first dataset contains two differing viewpoints, 1st person and 3rd person, with a demonstration for each being gathered simultaneously to ensure that corresponding videos are time-synced. The pushing task consisted of several coloured objects placed on a table with a black target square, with the goal of the task being to push a specified object into the target square. However, as the research progressed, this dataset was scrapped in favour of an alternative method to ensure that the model could contextualise between more than two contexts. 132 samples were collected for this dataset before it was deemed to be suboptimal for the use-case. Two examples from this dataset are shown below in Figure 5, demonstrating one example in a 1st person viewing angle and the

corresponding 3rd person sample. This dataset took approximately 5 hours to generate.

Figure 5. Dataset generation for 1st dataset, showing 1st person viewing angle (left) and 3rd person viewing angle (right)

The second dataset, which was used for the training of the context-adaptation architecture for real-world use-cases, consisted of a pushing task being performed using a single blue object and a red target square. The tool used for object manipulation was changed from the tool used in the first dataset, as it was found that the initial tool was difficult to effectively use. As such, a custom object manipulation tool was designed with a “cupped” end to ensure that the objects could not slip during manipulation. The viewing angles for each sample were varied randomly, adjusting the desk-mounted camera stand along with the free-mounted webcam to ensure a diverse range of viewing angles were captured. In addition to randomising the viewing angle, distractor objects were used to assess the viability of the context-adaptation model in semi-structured environments. 143 samples were generated for this dataset, with 10 samples being taken using “extreme” cases of distractor objects. Figure 6 shows three samples from the dataset, demonstrating a standard viewing angle example, a distractor object example, and an “extreme” viewing angle example. Several of the “extreme” viewing angle samples were taken such that the target square was partially obscured, again to increase the diversity of the dataset. Figure 7 shows an example of one of the “extreme” distractor object samples, which were used only during validation of the model. The “extreme” distractor object examples contain a variety of additional distractor objects, along with a variation on the colour of

the target square to evaluate if the model can contextualise to previously unseen contexts of the same task. The second dataset took approximately 6 hours to gather due to the need to change the viewing angle of both cameras after each sample is gathered.

Figure 6. Dataset samples showing a standard viewing angle (left), a distractor object example (centre), and an "extreme" viewing angle example (right)

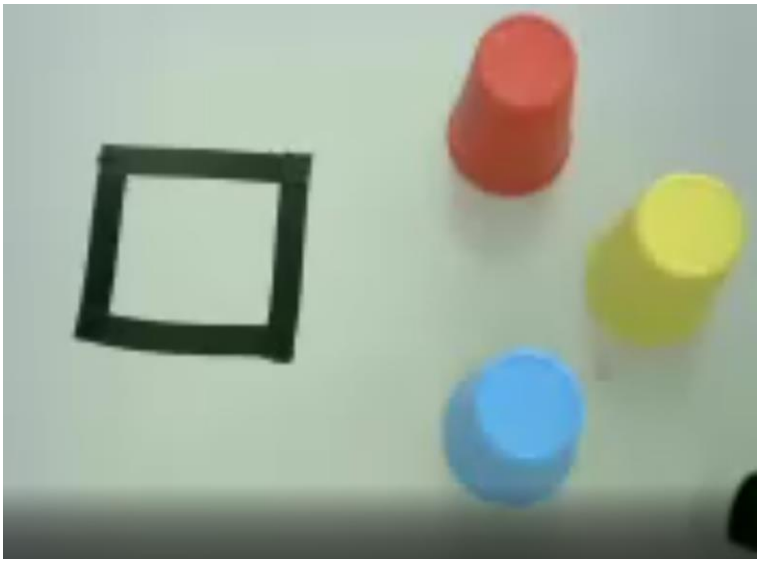

Figure 7. Dataset sample showing an "extreme" distractor object example

For the simulation-based demonstration, a dataset was generated autonomously using the *gen_videos.py* file shown in Appendix A.1, which was adapted from the work in (Liu Y, 2018). 4,000 samples were gathered for the simulation dataset, from varying viewing angles, starting positions and object distributions. The greater number of samples in the simulated dataset was due to the speed of data generation in simulation. The simulation examples were taken at 10Hz over a period of 5 seconds, for a total of 50 frames, with a slightly increased resolution of 64x64 compared to the real-world datasets. The simulated dataset took

approximately 2 minutes to collect. Due to the size of the dataset and the ease and speed of dataset generation, the video examples were not saved to a folder and the *gen_videos.py* code was run for each simulation example. A serialised pickle file was also created that can be used to autonomously generate the videos from within a jupyter-notebook. The random seed within RLlab was set to a constant value of 42 to ensure reproducibility in the dataset. The simulation-based demonstrations consist of a pushing task that is semantically like the examples in the real-world dataset. A 3-DoF manipulator was used in simulation to push a target object to a target position, with the target position displayed as a red circle. Figure 8 displays two dataset examples from the simulated dataset with differing object distributions and placement, starting positions of the manipulator, and viewing angles.

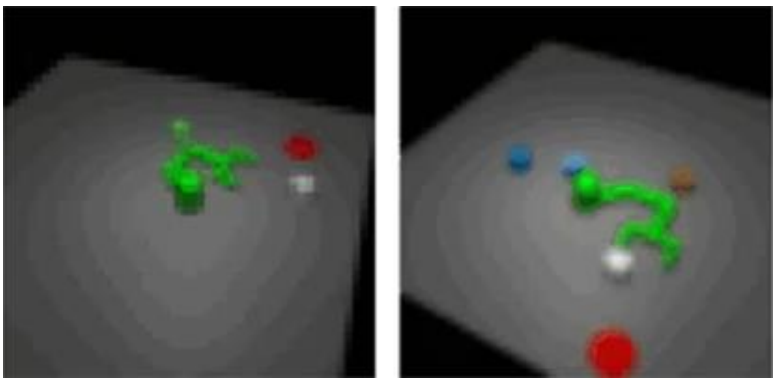

Figure 8. Dataset samples from the simulated dataset, showing two examples with differing object distributions, starting positions, and viewing angles

3.2 Context-Adaptation

The context-adaptation model was trained on a set of demonstration trajectories, containing 80 observations per trajectory. At each training iteration the model takes two demonstrations as inputs, D_i and D_j, with each demonstration containing a differing context, ω_i and ω_j, of the same task being completed. D_i refers to the expert demonstration, with D_j referring to the target context. As each demonstrations context is randomised, the agent is unable to effectively correlate the demonstration to its own context without some translation between the contexts. This is mitigated through the context-adaptation model which can

contextualise between the differing contexts without any prior knowledge about the contextual environments. The inputs to the context-adaptation model were normalized prior to being inputted to prevent anomalous pixel magnitudes, such as from highly reflective materials, affecting the output of the model (Koo KM, 2017). The context-adaptation model was constructed and trained from within a jupyter-notebook using TensorFlow.

The first frame of the target context, ω_j, can be used exclusively to translate between the expert demonstrations without any additional knowledge about the task within context ω_j. Thus, the context-adaptation model must learn to contextualise between D_i and D_j using a single observation from D_j along with the entire 80 step trajectory from D_i to predict the additional 79 trajectory steps in the context ω_j. After training, the model is then able to perform a translation between two differing contexts, ω_i and ω_j, to construct a full-length 80 step trajectory within the agents own context. D_i and D_j are assumed to be time-synced, with the method of dataset generation being selected to attempt to minimise discrepancies in the demonstrations over time (Sermanet P L. C., 2018). The overall objective of the context-adaptation model is to construct an overarching translation function, shown in equation (14), where $G\left(o_t^i, o_0^j\right)$ is the overarching translation function, $o_t^{i||j}$ represents the observation at timestep t for observation i or j, and $(\hat{o}_t^j)_{tr}$ is the translated video that aims to emulate o_t^j for all step values of t and all demonstration pairs.

$$G\left(o_t^i, o_0^j\right) = (\hat{o}_t^j)_{tr} \quad \textbf{14}$$

The context-adaptation architecture is comprised of a multi-stage CNN that consists of two encoders, $Enc_1(o_t^i)$ and $Enc_2(o_t^j)$, with Enc_1 taking the full-length demonstrations as the input and Enc_2 taking the static target context image as the input. The encoders take these inputs, generating an encoded feature representation in the expert and target contexts, z_1 and z_2. These encoded feature representations are concatenated together and passed into a translation

module that attempts to translate the encoded features in z_1 into the context of z_2, equation (15).

$$T(z_1, z_2) = z_3 \quad \textbf{15}$$

A decoder is the used, $Dec_1(z_3)$, taking z_3 as the input to decode the translated features into a static image in the target context, $\hat{o}_t^j$. This entire process is repeated iteratively for each of the 80 observations, with the output of the decoder being appended to form a full-length demonstration in the target context.

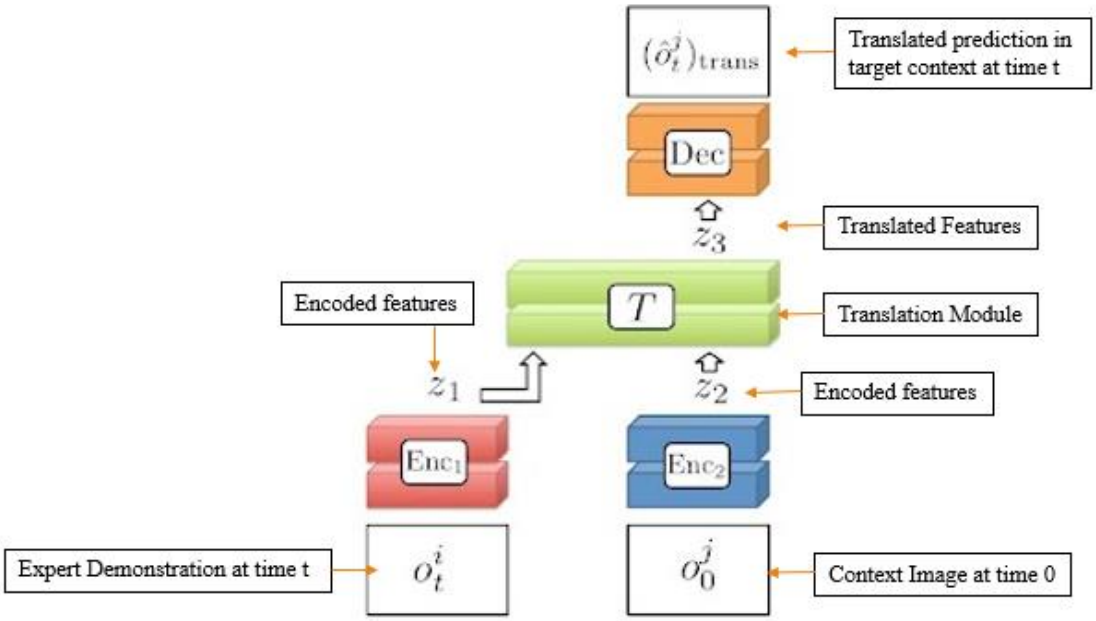

Figure 9. Context-adaptation model (Liu Y, 2018)

Throughout the training process, the model is iteratively improved using an overall loss function as the training metric, with the objective of the training being to minimise the result from the loss function. During training only, a copy of Enc_1 and Dec_1, referred to as Enc_{1B} and Dec_{1B}, were used to encode and decode the true demonstration in the target context, omitting the translation module. The output from this can then be used to construct a loss function, comparing the outputs between the true decoded target context, $(\hat{o}_t^j)_{truth}$, and the translated decoded target context, $(\hat{o}_t^j)_{tr}$. The overall loss function was comprised of three

separate losses: a loss of translation, a loss of reconstruction, and a loss of alignment. The loss of translation was calculated using the squared error between $(\hat{o}_t^j)_{tr}$ and o_t^j, comparing the translated output to the unencoded demonstration in the target context at each step t; equation (16). This loss ensures that the final translated output of the model is similar in pixel magnitude and distribution to the true demonstration in the target context.

$$L_{tr} = \left|\left(\hat{o}_t^j\right)_{tr} - o_t^j\right|_2^2 \quad \textbf{16}$$

The loss of reconstruction calculates the consistency in feature representations between Enc_{1B} and Dec_{1B} and the true unencoded output at time step t; equation (17), where $\left(\hat{o}_t^j\right)_{rec}$ refers to the output of Enc_{1B} and Dec_{1B}. As such, the loss of reconstruction ensures that the translated feature representations, z_3, are semantically similar to the encoded feature representations in z_1 through evaluation of the performance of Enc_1 and Dec_1. Enc_1 and Dec_1 are trained in tandem, essentially acting as an autoencoder (Chen M, 2017).

$$L_{rec} = \left|\left(\hat{o}_t^j\right)_{rec} - o_t^j\right|_2^2 \quad \textbf{17}$$

The final loss, the loss of alignment, determines the similarity between the translated feature representation, z_3, and the output of Enc_{1B}; equation (18). This ensures that the translation module is accurately translating between the contexts in z_1 and z_2, encouraging a static feature representation in which the encoded features of o_t^j are as close as possible to z_3.

$$L_{align} = \left| z_3 - Enc_{1B}(o_t^j)\right|_2^2 \quad \textbf{18}$$

Thus, the goal of the model during training is to minimise the overall loss, shown in equation (19) where γ_1 and γ_2 are weightings for the loss of reconstruction and the loss of alignment that are tuned during training.

$$L = L_{tr} + \gamma_1 L_{rec} + \gamma_2 L_{align} \quad \textbf{19}$$

The model was trained for a total of 11,700 iterations, with testing performed on the validation data after every 40 iterations. The “extreme” distractor object examples were omitted from the dataset during training and used for additional validation at the end of the process to assess the context-adaptation models performance on previously unseen examples, with no contextually similar examples within the training dataset.

3.2.1 System Architecture

For the context-adaptation model trained using real-world data samples, the architecture from (Liu Y, 2018) was adapted. All layers use LeakyReLu activations with a leak value of 0.2 to help prevent overfitting during the training process. The weightings of Enc_1 and Enc_2 are synced, with each encoder containing four 5x5 convolutions with filter sizes of 32, 16, 16, and 8 at each layer and alternating strides of 1, 2, 1, 2 shown in Figure 10; where strides relate to the number of pixels shifted in the input matrix. The encoders end with two connected hidden layers, each with 100 nodes. Enc_2 additionally contains skip-connections to Dec_1 by concatenating through the filters.

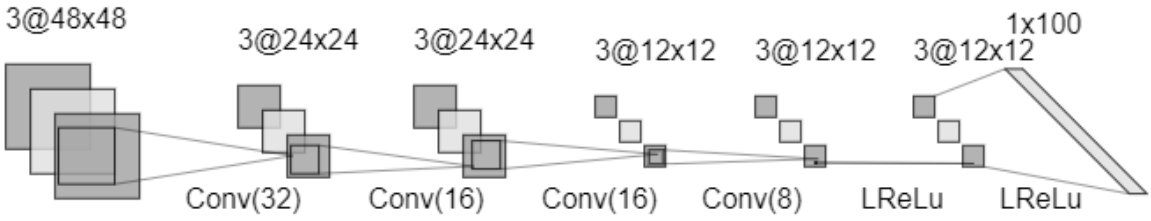

Figure 10. Encoder architecture, with numbers in brackets representing the convolutional filter size and 3@48x48 representing a 3-channel image with resolution 48x48

Following the encoder, the translation module contains one hidden layer with 100 nodes, taking z_1 and z_2 as inputs, concatenating the two beforehand.

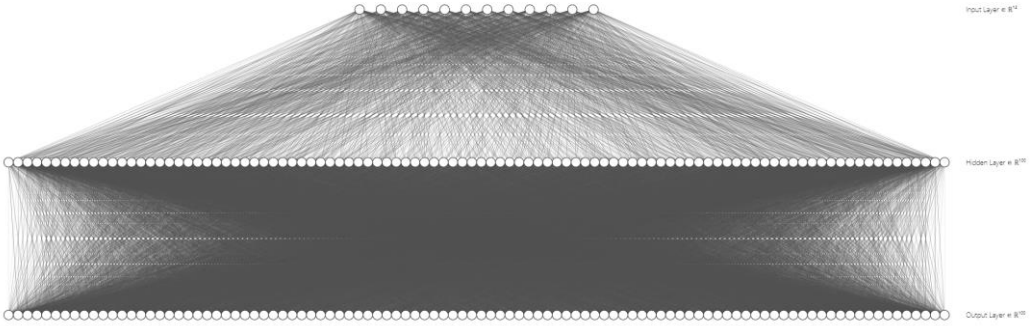

Figure 11. Translation module overview, containing a hidden layer with 100 nodes feeding into the initial hidden layer of the decoder with 100 nodes

Dec_1 contains fractional convolutions with strides of ½, 1, ½, 1 respectively. The filter sizes for Dec_1 are 16, 16, 32, 3 respectively, resulting in an output of the same dimensions as the input; shown in Figure 12. All connected layers contain a dropout function with a keep probability of 0.5 to reduce overfitting the model to the training dataset.

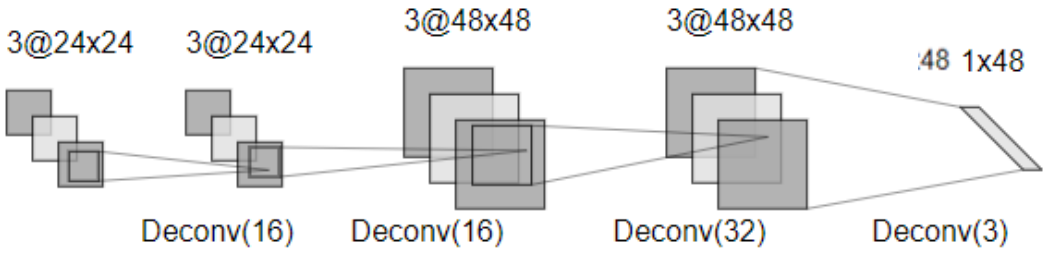

Figure 12. Decoder architecture, with numbers in brackets representing the convolutional filter size and 3@48x48 representing a 3-channel image with resolution 48x48

For the context-adaptation model trained using the simulated dataset, the weightings for Enc_1 and Enc_2 were no longer synced and all connected layers use LeakyReLu activation, again with a leak value of 0.2. Each encoder consists of four 5x5 convolutions with the filter size being equivalent to the height of the input image for the first convolution, doubling the filter size at each convolution; resulting in filter sizes 64, 128, 256, 512 respectively. The encoders then contain two connected layers with 1024 hidden nodes, with the number of hidden nodes being increased substantially from the real-world model to the simulation model

due to the additional diversity present in the simulated dataset. The translation module used in simulation contains one hidden layer with 1024 nodes. Dec_1 for the simulation model contains four 5x5 fractional convolutions with a consistent stride value of ½ and filter sizes 256, 128, 64, 3 respectively.

Both the real-world model and the simulated model were trained multiple times, varying the optimizer between ADAM and AdaDelta with learning rates for both of 1×10^{-3} to assess the performance and convergence time for each. Additionally, the loss functions described in section 3.2 were alternately omitted to assess the importance of each loss on the model's accuracy. The batch-size during training was set to 100, with a train/test split of 122 and 21 respectively.

3.3 Reinforcement Learning

Once the context-adaptation model has been trained the RL process can begin, aiming to learn a policy that outputs actions that accurately emulate the observations in the target contexts translated video, $(\hat{o}_t^j)_{tr}$, outputted from the context-adaptation model. However, before the RL process can begin a reward/loss function needs to be constructed to ensure the performance of the RL algorithm can be assessed and iteratively improved upon. For this use-case, the loss function should be used to assess the discrepancies between the translated target context video outputted by the context-adaptation model and the video-feed from the agent within its environment in real-time. As discussed in section 3, the squared Euclidean distance can be used, comparing the current observations encoding in the agent's context to the translated features from the expert demonstration; equation (20), where o_t^a is the observation of the agent at timestep t, T is the translation module, and o_t^i is the expert demonstration at timestep t.

$$L_f(o_t^a) = -|Enc_1(o_t^a) - \frac{1}{m}\sum_{i}^{m} T(Enc_1(o_t^i), Enc_1(o_0^a)\,|_2^2 \qquad \textbf{20}$$

It was shown in the work of (Sermanet P L. C., 2018) and (Liu Y, 2018) that a loss function such as this is not sufficient on its own to accurately emulate the

behaviour demonstrated in the expert demonstration. This is due to the observations from Enc_1 sometimes not closely matching the dissemination of the expert demonstrations observed in the training process. As such, when Enc_1 attempts to encode demonstrations in contexts that are outside of the training examples, the performance of Enc_1 reduces. Thus, an additional loss function was constructed that gives negative feedback to the RL model when it produces observations that directly contradict the expert demonstrations. This additional loss function was given a weighting to reduce its importance in the overall loss function, which can be tuned during the training process. Equation (21) demonstrates this loss function, with G representing the overarching translation from beginning to end. Equation (22) displays the overall loss function for the reinforcement learning training process, where φ represents the weighting of the loss function.

$$L_i(o_t^a) = -|o_t^a - \frac{1}{m}\sum_{i}^{m} G(o_t^i, o_0^a)|_2^2 \quad \textbf{21}$$

$$L_{total}(o_t^a) = L_f(o_t^a) + \varphi L_i(o_t^a) \quad \textbf{22}$$

Multiple RL algorithms are applicable to this use-case, with DDPG being selected for the simulation due to its high performance in continuous state/action spaces, along with the introduction of the DQL critic to the PG method which reduces the likelihood of the policy reaching a local maximum (Gu S, 2016). The policy was chosen as a deterministic MLP policy, producing a singular action in each state as opposed to a probability distribution of actions; this was chosen as it is less computationally expensive to produce a single action than a probability distribution encompassing all possible actions. The exploration strategy chosen was the Ornstein-Uhlenbeck process, which adds time-related noise to actions chosen in a deterministic policy to encourage a small amount of randomised exploration in the environment. This exploration strategy was chosen, again, to ensure that the policy did not reach a local maximum while encouraging emulation of the expert demonstrations.

For the simulation, the RL process was run in RLlab using the MuJoCo physics simulator, run on the “Pusher-3DoF” environment which models a basic 3-DoF manipulator. This environment was chosen as it is the most semantically like a real-world manipulator whilst being significantly quicker and easier to implement and validate the results. The process was run for 250 iterations, with a qf learning rate of 1×10^{-3} and a policy learning rate of 1×10^{-4}.

As previously mentioned, a real-world demonstration of the RL implementation was unable to be produced due to time-constraints. However, the process for implementing the RL model would be similar to the implementation within simulation, increasing the DoF for actuation relative to the DoF of the real-world robotic manipulator. Taking a UR5 6-DoF manipulator as a model agent for the real-world demonstration, the RL algorithm could be run by connecting to the UR5 through an ethernet cable using a TCP/IP connection. The official UR5 driver (Messmer, 2019) could be used to actuate the manipulator in real-time, with the output of the RL algorithm being used as the input to the UR5 driver. The iterative RL process in the real-world would be significantly more time-consuming as it requires actual actuation within its environment to converge to an optimal policy. Additionally, from the point that the target context image is taken the camera must remain static to ensure the context does not change between the target context image and the agent’s environment.

3.4 Experimentation

Experimentation was undertaken to assess the viability of the context-adaptation model in a diverse range of contexts, such as the 10 extreme distractor object examples. These “extreme” examples were omitted from the real-world training dataset to allow the assessment of the models’ performance in a previously unseen context that is not like any demonstrations from the training dataset; these were evaluated qualitatively by comparing the true output to the translated output, as well as quantitatively by assessing the loss function result during translation. In addition, further experimentation was performed to determine the performance during training using two differing optimizers, ADAM and AdaDelta. The time for training, the loss across iterations, and the convergence period were monitored

to assess the training time for each as the computational efficiency of the model during training is an important factor in the usability of the model for real-world use-cases (Liu T, 2015). The feature importance during training was also monitored, allowing for the importance of each frame in the translation to the target context to be verified. All three of the above tests were performed on the real-world dataset to ensure the context-adaptation model can perform accurately in a real-world application.

The loss functions were sequentially removed from the context-adaptation model to assess the effect of each loss function on the overall performance of the model within the RLlab simulation. The removal of the translation loss allows for the importance of the correlation between the output, $\left(\hat{o}_t^j\right)_{rec}$, and the true demonstration in the target context, o_t^j, to be determined. The removal of the reconstruction loss allows for the importance of the translated features', z_3, alignment to the encoded input, z_1, to be established. Finally, the removal of the alignment loss allows for the determination of the importance of alignment between z_3 and the encoded output of Enc_{1B}, $Enc_{1B}(o_t^j)$. Through the assessment of all these losses, a less computationally expensive loss function could be produced that would decrease the training time, although this would be undesirable if it also significantly reduced the performance of the model. The test for each loss removal consisted of 15 runs within RLlab, randomising the context between each run, with each run consisting of 250 iterations. Furthermore, the effect of the removal of the loss functions, $L_f(o_t^a)$ and $L_i(o_t^a)$, on the RL policies performance was evaluated, again assessing the performance over 15 runs with 250 iterations per run.

For the RL implementation, experimentation was undertaken to appraise the models performance compared to current state-of-the-art methods, namely the works of (Liu Y, 2018), (Stadie BC, 2017), and (Ho J, 2016) along with RLlabs inbuilt Oracle, which allows for the upper limits of the method to be determined when using DDPG with the true non-translated demonstration. The work in (Stadie BC, 2017) encompasses the third person imitation learning method, which is a form of IRL that uses an adversarial loss, comparing the target

demonstration to the current state to produce a policy. The work of (Ho J, 2016) encompasses generative adversarial imitation learning (GAIL), which is essentially an AC method that uses a CNN for image processing. Each approach was assessed over 15 runs on the “Pusher-3DoF” environment within RLlab, with 250 iterations per run. A “success” in this environment was defined as pushing the centre of the target object within 2cm of the centre of the goal destination. When comparing the work to (Liu Y, 2018), the evaluation in the performance was to assess the effect of adding a more diverse range of contexts to the dataset during training, along with the comparison in performance between the TRPO and DDPG RL algorithms within the RLlab simulation.

4 Results

4.1 Context-Adaptation

The results of the context-adaptation model when training on the real-world dataset using the ADAM optimizer, with a learning rate of 1×10^{-3}, showed a quick convergence from an initial total loss value of 47,564 to a value of 9,862 over 500 iterations. After 500 iterations the convergence plateaus, with the total loss being reduced by a smaller amount for each iteration. The minimum total loss was found to be a value of 3,214 at iteration 10,567. The validation loss showed a similar trend, converging to a validation loss of 9,624 after 500 iterations. As with the total loss, the convergence of the validation loss plateaus, with the minimum validation loss being a value of 2,615 at iteration 11,242. Figure 13 displays both the total loss and validation loss during training over all iterations. The total training time for 11,750 iterations was 8 hours using an Nvidia GTX 970 with CUDA, cuDNN and tensorflow-gpu.

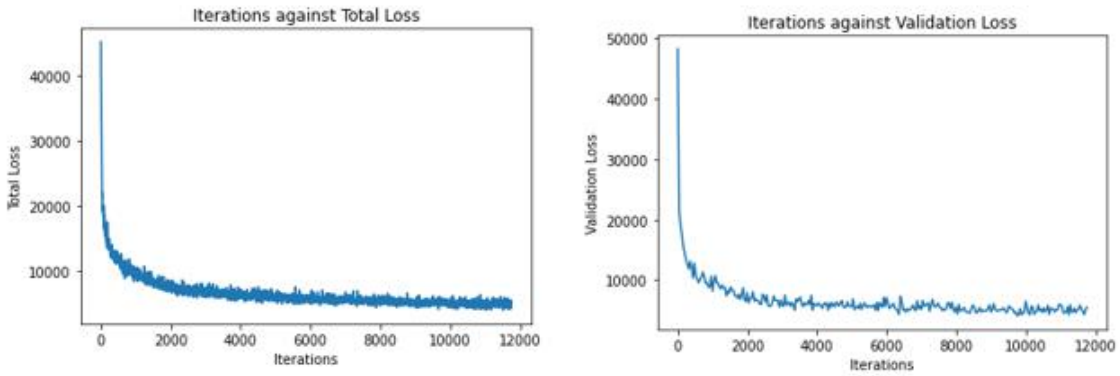

Figure 13. Context-adaptation training using ADAM optimizer, showing the total loss during training (left) and the validation loss during training (right)

When training using the AdaDelta optimizer with a learning rate of 1×10^{-3}, the initial total loss value at the first iteration was 41,323. This value converged to a total loss of 26,469 at iteration 500, showing a significantly slower convergence rate than the ADAM optimizer. After 1,500 iterations the convergence plateaus, with a minimum total loss of 13,621 at iteration 6,102. The validation loss when using the AdaDelta optimizer showed a similar trend, with an initial validation loss of 38,212, converging to a validation loss of 27,441 after 500 iterations. After 1,500 iterations the convergence begins to plateau, with a minimum validation loss of 12,232 at iteration 3,903. Figure 14 shows the total loss and validation loss over all iterations. The total training time for 6,320 iterations was 6 hours using an Nvidia GTX 970 with CUDA, cuDNN and tensorflow-gpu.

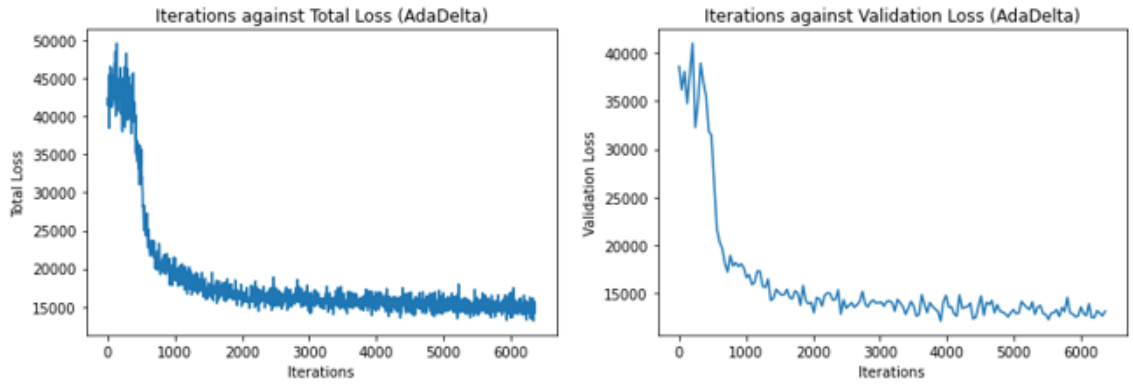

Figure 14. Context-adaptation training using AdaDelta optimizer, showing the total loss during training (left) and the validation loss during training (right)

The feature weightings were monitored during the validation of the real-world dataset, shown in Figure 15. The results from the feature weighting showed an initially high average feature weighting value of 0.016 from frames 0 to 4 before dropping to a value of 0.010 at frame 5. A spike in feature weighting can be observed between frames 30 and 40 increasing to an average value of 0.025, before dropping to a value of almost 0 between frames 40 and 48. After frame 49, the feature weighting remains relatively static, apart from a spike between frames 60 and 62 with a value of 0.008.

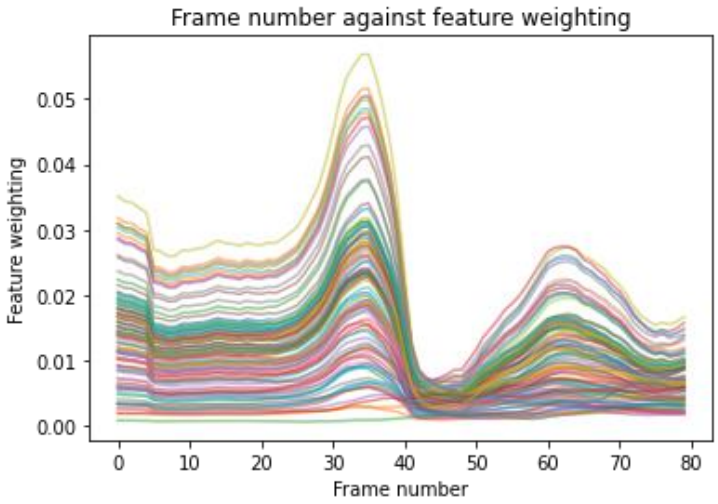

Figure 15. Feature-weighting against frame number, demonstrating the importance of each frame in the determination of the final translated video

Once the model was fully trained, the performance of the model was evaluated qualitatively on the validation dataset to assess its similarity. This was done by looking at the true target context demonstration along with the translated video frame-by-frame, evaluating the similarity every 7 frames. As can be seen in Figure 16, the true target context demonstration and the translated video appear to be well time-synced as well as displaying extremely similar pixel magnitudes and distributions. The final frames from the two videos both show the target object being placed within the target square successfully. This example gave a loss value of 3,415 for the translated video, with an average loss of 4,232 across all 21 validation examples.

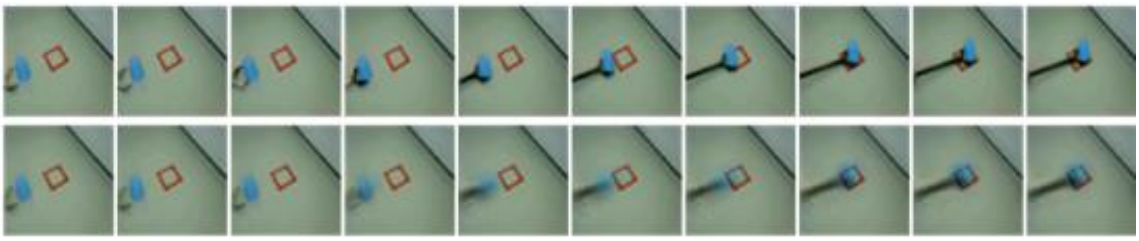

Figure 16. True target context demonstration (top) and translated video (bottom), showing a static image from every 7 frames

Testing the fully trained model on the previously unseen “extreme” distractor examples demonstrated a lack of ability to generalise to extreme contexts that have not been included in the dataset. Figure 17 shows the final frame from the translated video and the true target context demonstration, with the translated video failing to accurately emulate the target demonstration. The translated video does demonstrate a shift in pixel values to display a faint blue object within the target-square; however, this would not be usable for the RL implementation as it does not accurately capture the semantic meaning of the task. The average loss across the 10 extreme distractor examples was 11,562.

Figure 17. “Extreme” distractor example translated video (left) and true target context demonstration (right), showing the final frame from both demonstrations

The model was also trained using examples gathered from the RLlab MuJoCo simulator, described in section 3.1, to assess its performance qualitatively. Further experimentation was done into the overall system in simulation, thus

extensive testing of the context-adaptation model in simulation was not completed. Figure 18 shows the final frame from both the translated video and the true target context demonstration. This clearly demonstrates a semantically similar final frame between the two demonstrations. An average loss of 2,302 was calculated for the validation of the simulation-based dataset.

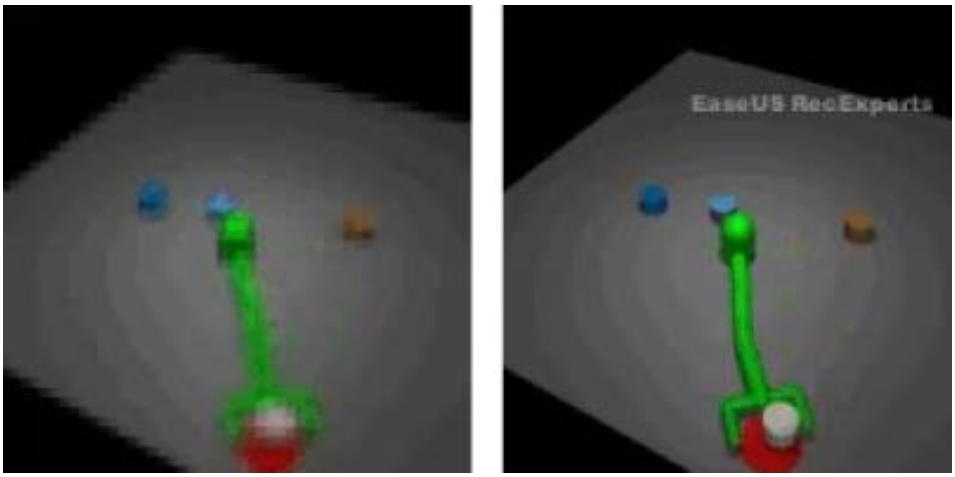

Figure 18. Simulation-based translated video (left) and true target context demonstration (right), showing the final frame from both demonstrations

4.2 Reinforcement Learning

Figure 19 displays the comparison between the approach produced in this work, along with the approaches of (Liu Y, 2018), (Stadie BC, 2017), (Ho J, 2016), and RLlab's built in Oracle. The Oracle displays the best possible result for using DDPG, using the true target context demonstrations instead of the translated demonstrations for training. Both the works of (Stadie BC, 2017) and (Ho J, 2016) were unable to perform the task successfully, with a success rate of 0%. The approach produced in this work, using DDPG, resulted in a success rate of 80% when performing the pushing task in simulation, compared to a success rate of 73.33% using the TRPO approach from (Liu Y, 2018). The Oracle demonstrated an 86.67% success rate, which is the upper limit for the overall success rate.

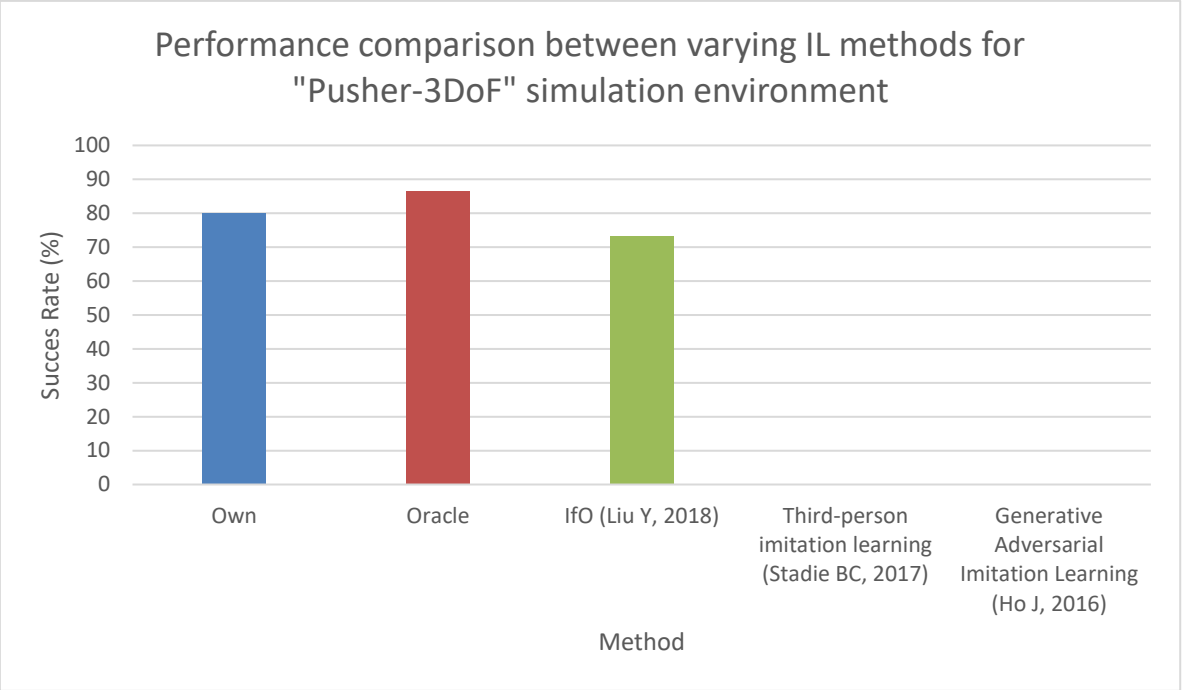

Figure 19. Performance comparison between varying IL methods for the "Pusher-3DoF" simulation environment, with third-person imitation learning and generative adversarial imitation learning both having a success rate of 0%

Figure 20 presents the comparison of results for removing the various loss functions, defined in sections 3.2 and 3.3, on the performance of the model in the Pusher-3DoF RLlab simulation environment. With all losses present, the model was able to achieve a success rate of 80%. The removal of L_{tr} from the context-adaptation model resulted in a greatly reduced success rate of 46.67%, with the removal of L_{rec} displaying a further decrease in success rate to 40%. Removing L_{align} from the model produced a lower reduction in success rate when compared to L_{tr} and L_{rec}, with a success rate of 73.33%. The removal of both L_{rec} and L_{align} gave an overall success rate of 40%, the same result as removing purely the L_{rec} loss. When removing the L_f term from the RL loss function, the success rate reduced to 46.67%, whilst removing the L_i term gave a success rate of 60%.

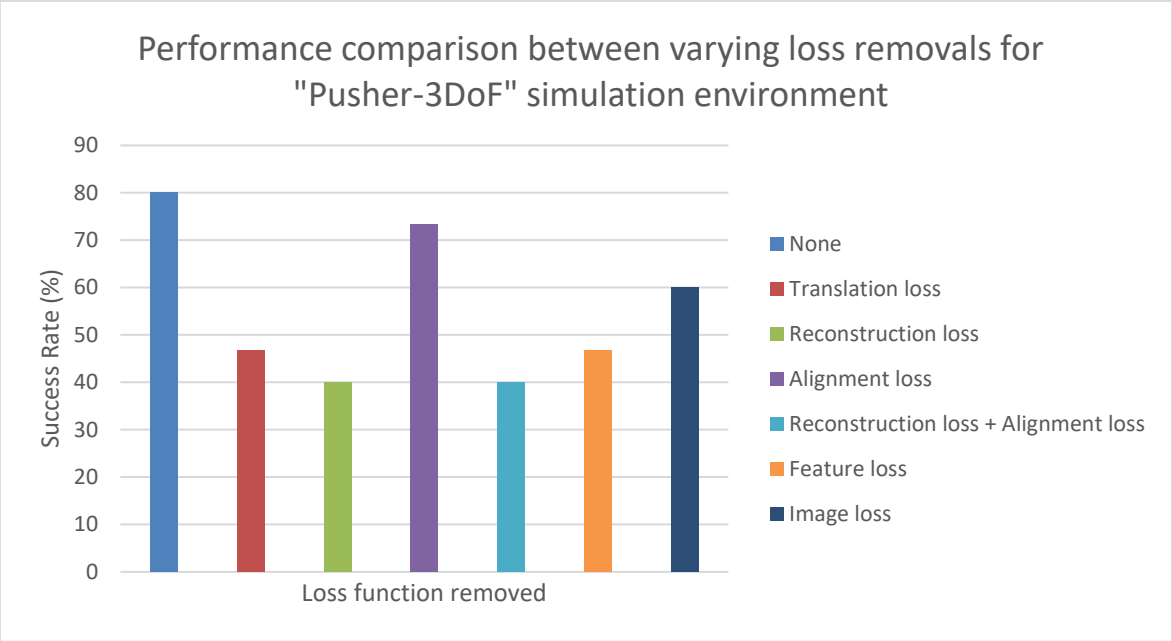

Figure 20. Performance comparison between varying loss removals for the "Pusher-3DoF" simulation environment, where the legend relates to the losses that have been removed from the model

5 Analysis & Discussion

5.1 Context-Adaptation

As can be seen from Figure 13 and Figure 14, the ADAM optimizer shows a significantly faster convergence rate compared to AdaDelta, as well as a reduced total loss and validation loss during training. The minimum validation loss when training using ADAM was 2,615, a decrease of 9,617 when compared to the minimum validation loss of 12,232 from AdaDelta; however, AdaDelta was run for 5,430 iterations fewer than ADAM so it cannot be confirmed that AdaDelta would not converge to a lower minimum validation loss if run for the same number of iterations. AdaDelta was only run for 6,320 iterations before it was decided that ADAM would be a more applicable optimizer for this use-case due to the faster convergence time. The discrepancy in results between the two optimizers is likely due to ADAM being, generally, a more robust optimizer when using step-size calculations. Theoretically, ADAM is also more structured in its optimization technique, as AdaDelta does not contain a convergence or regret guarantee (MD., 2012). ADAM also includes bias correction by incorporating a momentum

factor into the optimization process, which usually results in faster convergence due to larger gradients being generated at the beginning of training, resulting in the model reaching a local or global maximum faster (Z., 2018). Although ADAM showed faster convergence and lower loss values, it is possible that the model produced using ADAM was overfit to the training dataset, resulting in a loss that is not representative of the performance of the model in a real-world scenario. However, it was shown that the model had similar loss values on the validation dataset, suggesting that the model is not overfit to the training dataset. To substantiate this claim more validation examples would need to be used.

The feature weightings, shown in Figure 15, display the importance of each frame on the overall translation of the demonstration; with higher feature weightings demonstrating a higher importance. The first 4 frames were shown to have a relatively high average feature weighting of 0.016. This is assumably due to the model needing to initially contextualise itself based on the first frames of the expert demonstration, to ascertain when the task is initiated as well as the initial trajectory of the object in the demonstration. The feature weighting between frames 5 and 29 is relatively static, with an average weighting of 0.010, which relates to the manipulation stage of the object in the demonstrations. This reduced feature weighting between frames 5 and 29 could be due to the trajectory of the object remaining linear, with the model already determining the trajectory of the object from the first 4 frames it can extrapolate this information without extensive additional information; this was also shown to be the case in the work of (Sermanet P L. C., 2017) when contextualising between a 1st and 3rd person viewing angle.

There was a large spike in feature weightings between frames 30 and 40, with an average value of 0.025, which relates to the point in the video where the object crosses the threshold of the target square. This increased feature weighting is possibly caused by the partial obstruction of the target square by the object, resulting in the model having to calculate additional pixel transitions as the threshold is crossed. After frame 40 the feature weighting greatly decreased due to the object already being within the target square, meaning no further translation

is required. There was a small increase between frames 60 and 62, with a value of 0.008, which may be due to instability in the tool when being held along with the possibility of some demonstrations not being properly time-synced due to the manual data generation technique used. Overall, the feature weightings give a good indication of which frames are of most importance to the model for the contextualisation between demonstrations, with frames 30 to 40 being of the most importance.

The qualitative evaluation of the model proved that the model can successfully contextualise to a diverse range of contexts on previously unseen data, provided that similar contexts are provided during training; this includes examples taken from “extreme” viewing angles along with the distractor object examples shown in Figure 6. Figure 16 demonstrates the models’ ability to contextualise, along with displaying the capability of the model to produce translated videos that are time-synced with the true target context demonstrations. Comparing the translated videos to the true target context demonstrations, the translated videos competently emulate the task shown in the demonstrations. The average loss for the validation was 4,232, which is a 21% decrease when compared to the validation results in (Liu Y, 2018). This is most likely due to the increased diversity in the dataset, resulting in a model that is more capable of contextualising to a diversified set of contexts.

When testing the model on previously unseen “extreme” distractor object examples, the performance was shown to drastically decrease. Figure 17 demonstrates this, with the final frame from the translated video and true target context demonstration being significantly different, resulting in an average loss of 11,562 across all “extreme” distractor examples. This strongly suggests that the model is unable to contextualise to “extreme” context examples that are previously unseen. Thus, for the accurate contextualisation to the “extreme” demonstrations, several examples would need to be included within the training dataset. This is confirmed further as the model was successfully able to contextualise to the “extreme” viewing angle examples that were contained within the training dataset.

The qualitative performance of the model when trained using the simulation-based architecture showed excellent results. Both the translated and true target context videos were shown to be extremely close in terms of pixel magnitudes and distributions, as well as being closely time-synced throughout the task being performed. An average loss of 2,302 was calculated for the validation of the simulation-based dataset.

Overall, the context-adaptation model shows that it is capable of contextualising to a diverse range of contexts, provided that similar context examples are given during training. The ADAM optimizer was deemed to be optimal for the training of the model due to the vastly increased convergence time, without overfitting to the training dataset. The feature weightings suggest that the final 30 frames of the demonstrations are of negligible importance to the translation, thus suggesting that these frames could be omitted entirely from the demonstrations to produce shorter trajectories. However, the model produced has some clear limitations, namely the inability to contextualise to previously unseen contexts that are extremely dissimilar to contexts from within the training dataset.

5.2 Reinforcement Learning

As shown in Figure 19, the works of (Stadie BC, 2017) and (Ho J, 2016) showed a success rate of 0%, being entirely unable to generalise to differing contexts. Comparatively, (Liu Y, 2018) produced a relatively good success rate of 73.33%. The method proposed in this work was able to surpass the performance of (Liu Y, 2018) by 6.67%, only being beaten by the RLlab Oracle; which is to be expected as the Oracle gives the upper limit of RL performance for the task. The increase in performance in this work is most likely due to the increased diversity in the dataset provided during training. In turn, this produces a more robust model that can produce higher accuracy translated videos in a more diversified set of contexts; with this claim being corroborated through the comparison of the validation loss between the two models as in section 5.1. Alternatively, the performance discrepancy may be due to the RL algorithm used, with the work of (Liu Y, 2018) using TRPO and the work proposed here using DDPG. DDPG would likely produce a better overall policy than TRPO as it reduces the likelihood

of reaching a local maximum policy (Cen S, 2020) (Karunakaran, 2020). To evaluate which of these factors is the cause of the performance increase, further testing would need to be done using the TRPO RL algorithm with the model produced in this work; although it is probable that both the RL algorithm and the more robust model have a compounding effect on the overall performance in-situ.

The ablative study into loss removal, Figure 20, provided insights into which of the loss functions was most important for the application of the model using RL. With no losses removed the model produced a success rate of 80% which is the highest value across all ablative tests; this result was used as a benchmark for comparing the removal of the various losses. The removal of L_{tr} from the context-adaptation model resulted in a significant reduction of success rate to 46.67% when performing RL in simulation, a decrease of 33.33% compared to the benchmark. This shows that the translation loss is vital for the effective performance of the context-adaptation model, with the correlation between $\left(\hat{o}_t^j\right)_{rec}$ and o_t^j being an important metric for training the model. Removing L_{rec} gave a greater reduction in performance compared to the removal of L_{tr}, with a reduction of 40% compared to the benchmark. L_{align} was removed to assess the performance change when the similarity between the encoded features z_3 and $Enc_{1B}(o_t^j)$ was not considered. This resulted in a small reduction of success rate, 6.67%, implying that the similarity between encoded features is not of great importance to the model. These results suggest that the alignment between z_3 and z_1 should be held as the most important metric during training, adjusting the weighting values to put more emphasis on the reconstruction loss; however, this would need to be verified through additional testing. Furthermore, the small reduction in loss when removing L_{align} suggests that this loss could be removed from the model entirely if computational speed is of the utmost importance; however, the decrease in performance is still substantial and likely not worth the minimal reduction in training time. These results would also need further legitimization through testing in a real-world use-case to ensure that the performance changes are not solely applicable to the simulated environment

Ablations performed on the RL loss function were done to ensure that the function was not overly complex for the use-case. The L_f loss was first removed, resulting in a 33.33% reduction of success rate. This severe reduction in performance corroborates the idea that the squared Euclidean distance between the encoded features is a good metric for determining RL performance and will aid in converging towards an optimal RL policy. Removal of the L_i term gave a 20% reduction in success rate, legitimizing the claim that the L_i loss is of lesser importance to L_f for the RL implementation. Although these results suggest that L_i is of lesser importance, this may be due to the introduction of the weighting, φ, which would give a lower weighting to the L_i term. Further testing would need to be undertaken through the removal of the weighting to validate this claim. Finally, the L_i loss removal resulting in a 20% reduction in success rate confirms that the L_f loss alone is not sufficient to accurately emulate the expert demonstrations; with this claim being corroborated by the work undertaken in (Sermanet P L. C., 2018).

Overall, the model showed better performance in the RLlab Pusher-3DoF environment than current state-of-the-art methods. The success rate from the work of (Liu Y, 2018) was shown to be equitable to the success rate of the model in this work with the removal of the L_{align} term.

5.3 Novelty

The novelty in this work lies mainly in the more diverse set of contexts within the dataset, along with the alternative use of the DDPG RL algorithm for iterative training. In addition, the model was tested for robustness by determining its performance on “extreme” demonstrations that are previously unseen, which has not been assessed previously. The dataset generation method also differed from previous approaches, with a wider array of contexts being captured for training and validation. As a result, the model in this work was able to produce a 21% decrease in validation loss during training of the context-adaptation model when compared to (Liu Y, 2018). Additionally, the work was able to increase the success rate by 6.67% in the RLlab Pusher-3DoF environment.

5.4 Future Work

There are several limitations to the current approach which would need to be addressed before it could be deployed to a real-world use-case. Firstly, additionally dataset samples should be generated using a human-arm for object manipulation as opposed to the custom manipulation tool. This would assess the ability of the model to perform domain-adaptation as well as context-adaptation, with no current approaches being able to perform both simultaneously. Secondly, a frame-skip function should be introduced, where the expert demonstrations used as inputs to the model are incomplete. Through this process, the capability of the model when given incomplete demonstrations could be assessed; as in the work of (Wu YH, 2019). Furthermore, the “extreme” expert demonstrations should be incorporated into the training dataset to determine if this improves the model’s ability to contextualise to more disparate examples. The resolution of expert demonstrations should also be investigated, with it being assumable that the performance of the model would increase minutely as the resolution increases; however, this would also increase the computational time due to the additional pixel values that are required to be translated. Provided that the translated video is correctly emulating the task shown in the expert demonstration, the performance of increasing the resolution could potentially be negligible.

Through the incorporation of a task-embedding architecture, such as in the work of (James S, 2018), the model would potentially be able to generalise between both differing contexts and tasks, allowing for an acutely robust model to be produced that has high performance over a wide range of contexts and basic single-stage tasks. This should be investigated and tested, although the implementation of this is likely to be excessively complex and would result in an immensely computationally expensive model; (Torabi F W. G., 2019).

The RL algorithm implementation should also be tested further in differing simulation environments to solidify its performance across a range of environments. The most important development of the work would be the implementation of the model onto a real-world system as described in section 3.3. This is vital to prove the validity of the constructed model in a real-world

environment as the performance between simulation and real-world can be vastly different for IL methods, (Bonardi A, 2020). Along with this, more complex task demonstrations should be constructed and trained to conclude whether the model can perform multi-stage complex manipulation tasks using the proposed method.

6 Conclusion

In conclusion, over the course of this work a dataset containing expert demonstrations from a variety of contexts was constructed to validate the performance of an IL model using a context-adaptation architecture. The IL model constructed in this work was able to successfully contextualise between differing contexts, namely varying viewing angles, and perform RL to produce a policy that closely replicated the actions of the expert demonstrator within the agents own context. The IL method used takes a model-free approach using sparse state-only observations, creating a translated video in the agent's context based off the expert demonstration. The translated video is then used to train a control policy through RL implementation, with the RL algorithm in this use-case being DDPG. The results from this method showed an increase in success rate of 6.67% when compared to the work of (Liu Y, 2018), along with a reduction in validation loss for the context-adaptation model of 21%. These results, along with the increased performance compared to the works of (Stadie BC, 2017) and (Ho J, 2016), suggest that this model is more capable of contextualising to a wider array of contexts than current state-of-the-art methods. The assessment of the various constructed loss functions determined that all loss functions, with the exception of L_{align}, are necessary to produce an accurate and robust IL model. Only the removal of the L_{align} term would be feasible; however, it was determined that the reduced computational complexity was not worth the trade off in performance of 6.67% when removing L_{align}.

Further work should be undertaken to assess the validity of the model in a real-world environment. In addition, the feasibility of including a task-embedding architecture, such as in (James S, 2018), should be validated to determine if the model would be able to generalise to both differing contexts and tasks. The

performance of the model should also be appraised on multi-stage complex tasks to gauge its viability for complex real-world manipulation tasks.

REFERENCES

Agarwal A, K. S. (2020). Optimality and approximation with policy gradient methods in markov decision processes. *Conference on Learning Theory.*

Arulkumaran K, D. M. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing, 34*(6), 26 - 38.

Bonardi A, J. S. (2020). Learning one-shot imitation from humans without humans. *IEEE Robotics and Automation Letters., 5*(2), 3533 - 3539.

Cen S, C. C. (2020). Fast global convergence of natural policy gradient methods with entropy regularization. *arXiv preprint*.

Chemali J, L. A. (2015). Direct policy iteration with demonstrations. *Twenty-Fourth International Joint Conference on Artificial Intelligence.*

Chen M, S. X. (2017). Deep features learning for medical image analysis with convolutional autoencoder neural network. *IEEE Transactions on Big Data*.

Chen Z, W. Z. (2020). Virtual-joint based motion similarity criteria for human–robot kinematics mapping. *Robotics and Autonomous Systems., 125*, 10412.

Cobbe K, K. O. (2019). Quantifying generalization in reinforcement learning. *International Conference on Machine Learning.*

Das N, B. S. (2020). Model-Based Inverse Reinforcement Learning from Visual Demonstrations. *arXiv preprint*.

E., U. (2018). Model-free deep inverse reinforcement learning by logistic regression. *Neural Processing Letters.*, 891 - 905.

Englert P, P. A. (2013). Probabilistic model-based imitation learning. *Adaptive Behaviour*, 388 - 403.

Finn C, A. P. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. *International Conference on Machine Learning.*

Fuji T, I. K. (2018). Deep Multi-Agent Reinforcement Learning using DNN-Weight Evolution to Optimize Supply Chain Performance. *Proceedings of the 51st Hawaii International Conference on System Sciences.*

Geist M, S. B. (2019). A theory of regularized markov decision processes. *International Conference on Machine Learning.*

Gu S, L. T. (2016). Continous Deep Q-learning with model-based acceleration. *International conference on machine learning.*

H., H. (2010). Double Q-learning. *Advances in neural information processing systems., 23*, 2613 - 2621.

Haarnoja T, Z. A. (2018). Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *International conference on machine learning.*

Hadfield-Menell D, R. S. (2016). Cooperative inverse reinforcement learning. *Advances in neural information processing systems., 29*, 3909 - 3917.

He H, E. J. (2012). Imitation learning by coaching. *Advances in Neural Information Processing Systems.*, 3149 - 4157.

Hester T, V. M.-A. (2018). Deep q-learning from demonstrations. *Thirty-second AAAI conference on artificial intelligence.*

Ho J, E. S. (2016). Generative adversarial imitation learning. *Advances in neural information processing systems.*, 4565 - 4573.

Hussein A, G. M. (2017). Imitation learning: A survey of learning methods. *ACM Computing Surveys, 50*(2), 1 - 35.

J., H. (2013). *Monte Carlo methods.* Springer Science & Business.

James S, B. M. (2018). Task-embedded control networks for few-shot imitation learning. *Conference on Robot Learning* (pp. 783 - 795). PMLR.

Karunakaran, D. (2020, November 23). *Deep Deterministic Policy Gradient(DDPG) — an off-policy Reinforcement Learning algorithm.*

Retrieved from Medium: https://medium.com/intro-to-artificial-intelligence/deep-deterministic-policy-gradient-ddpg-an-off-policy-reinforcement-learning-algorithm-38ca8698131b

Kober J, P. J. (2010). Imitation and reinforcement learning. *IEEE Robotics & Automation Magazine*, 55 - 62.

Koo KM, C. E. (2017). Image recognition performance enhancements using image normalization. *Human-centric Computing and Information Sciences.*, 1.

Kumar A, P. N. (2018). Bipedal walking robot using deep deterministic policy gradient. *IEEE Symposium Series on Computational Intelligence.*

Li D, Y. Y. (2018). Learning to generalize: Meta-learning for domain generalization. *Thirty-Second AAAI Conference on Artificial Intelligence.*

Liarokapis MV, A. P. (2013). Mapping human to robot motion with functional anthropomorphism for teleoperation and telemanipulation with robot arm hand systems. *IEEE/RSJ International Conference on Intelligent Robots and Systems.*

Littman ML, D. T. (2013). On the complexity of solving Markov decision problems. *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence.*

Liu T, F. S. (2015). Implementation of training convolutional neural networks. *arXiv preprint*.

Liu Y, G. A. (2018). Imitation from observation: Learning to imitate behaviors from raw video via context translation. *IEEE International Conference on Robotics and Automation.*

MD., Z. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint:1212.5701.*

Messmer, F. (2019). *ros-industrial universal_robot*. Retrieved from Github: https://github.com/ros-industrial/universal_robot

MM., L. (2018). Markov decision process measurement model. *Psychometrika, 83*(1), 67 - 88.

Parisi S, T. V. (2019). TD-regularized actor-critic methods. *Machine Learning, 108*(8), 1467- 1501.

Peters J, S. S. (2006). Policy gradient methods for robotics. *IEEE/RSJ International Conference on Intelligent Robots and Systems.*

Ravichandar H, P. A. (2020). Recent advances in robot learning from demonstration. *Annual Review of Control, Robotics, and Autonomous Systems., 3*, 297 - 330.

Schulman J, L. S. (2015). Trust region policy optimization. *International conference on machine learning.*

Seleem IA, E.-H. H. (2020). Development and stability analysis of an imitation learning-based pose planning approach for multi-section continuum robot. *IEEE Access, 8*, 99366 -99379.

Sermanet P, L. C. (2017). Time-contrastive networks: Self-supervised learning from multi-view observation. *IEEE Conference on Computer Vision and Pattern Recognition Workshops.*

Sermanet P, L. C. (2018). Time-contrastive networks: Self-supervised learning from video. *IEEE international conference on robotics and automation (ICRA).*

Sidford A, W. M. (2018). Near-optimal time and sample complexities for solving discounted Markov decision process with a generative model. *Proceedings of the 32nd International Conference on Neural Information Processing Systems.*

Silver D, L. G. (2014). Deterministic policy gradient algorithms. *International conference on machine learning.*

Stadie BC, A. P. (2017). Third-person imitation learning. *arXiv preprint*.

Sutton RS, B. A. (2018). *Reinforcement learning: An introduction.* MIT Press.

Torabi F, W. G. (2018). Behavioral cloning from observation. *arXiv preprint*.

Torabi F, W. G. (2018). Generative adversarial imitation from observation. *arXiv preprint: 1807.06158*.

Torabi F, W. G. (2019). Recent advances in imitation learning from observation. *arXiv preprint*.

Van Hasselt H, G. A. (2016). Deep reinforcement learning with double q-learning. *Proceedings of the AAAI conference on artificial intelligence.*

Wang Z, L. R. (2020). Fast and Intuitive Kinematics Mapping for Human-Robot Motion Imitating: A Virtual-Joint-Based Approach. *IFAC-PapersOnLine, 53*(2), 10011 - 10018.

Wei Z, X. J. (2017). Reinforcement learning to rank with Markov decision process. *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval.*

Wu YH, C. N. (2019). Imitation learning from imperfect demonstration. *International Conference on Machine Learning* (pp. 6818 - 6827). PMLR.

Xu B, L. W. (2020). Motor imagery based continuous teleoperation robot control with tactile feedback. *Electronics*, 174.

Yu C, D. Y. (2020). Distributed multi-agent deep reinforcement learning for cooperative multi-robot pursuit. *The Journal of Engineering.*, 499 - 504.

Yu T, F. C. (2018). One-shot imitation from observing humans via domain-adaptive meta-learning. *arXiv preprint*.

Z., Z. (2018). Improved adam optimizer for deep neural networks. *IEEE/ACM 26th International Symposium on Quality of Service* (pp. 1 - 2). IEEE.

Zhao L, L. Y. (2016). An intuitive human robot interface for tele-operation. *IEEE International Conference on Real-time Computing and Robotics (RCAR).*

Zhu, F. L. (2017). Learning spatial regularization with image-level supervisions for multi-label image classification. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 5513 - 5522). IEEE.

Ziebart BD, M. A. (2008). Maximum entropy inverse reinforcement learning. *Aaai, 8*, 1433 - 1438.

Appendix A – Code

A.1 Gen_videos.py

This appendix outlines the code file for the dataset generation within RLlab simulator, adapted from the work of (Liu Y, 2018) and RLlab official documentation: https://rllab.readthedocs.io/en/latest/

```
# #/

# The MIT License (MIT)


# Copyright (c) 2016 rllab contributors


# rllab uses a shared copyright model: each contributor holds copyright over

# their contributions to rllab. The project versioning records all such

# contribution and copyright details.

# By contributing to the rllab repository through pull-request, comment,

# or otherwise, the contributor releases their content to the license and

# copyright terms herein.


# Permission is hereby granted, free of charge, to any person obtaining a copy

# of this software and associated documentation files (the "Software"), to deal

# in the Software without restriction, including without limitation the rights

# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell

# copies of the Software, and to permit persons to whom the Software is

# furnished to do so, subject to the following conditions:
```

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.

Imports

from rllab.sampler.utils import rollout

from rllab.envs.gym_env import GymEnv

from rllab.envs.normalized_env import normalize

import argparse

import os

```
import numpy as np
import joblib
import uuid
import pickle

filename = str(uuid.uuid4())



def all_videos(episode_number):
    return True

if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument('file', type=str,
                        help='path to the snapshot file')
    parser.add_argument('env', type=str,
                        help='environment')
    parser.add_argument('logdir', type=str,
                        help='logdir')
    parser.add_argument('--max_length', type=int, default=80,
                        help='Max length of rollout')
    parser.add_argument('--speedup', type=int, default=1,
                        help='Speedup')
```

```
    parser.add_argument('--loop', type=int, default=1,
                        help='# of loops')
    args = parser.parse_args()



    with open(args.file, 'rb') as pfile:
        policy = pickle.load(pfile)


    while True:


        env = normalize(GymEnv(args.env))
        env._wrapped_env.env.monitor.start(args.logdir, all_videos, force=False, resume=True)
        env._wrapped_env.env.monitor.configure(video_callable=all_videos)


        path = rollout(env, policy, max_path_length=args.max_length, animated=False, speedup=args.speedup)


        # Recording video for simulation
        vidpath = env._wrapped_env.env.monitor.video_recorder.path
        env._wrapped_env.env.monitor.close()


        if path is not None:
```

```
        true_rewards = np.sum(path['env_infos']['reward_true'])
        print(true_rewards)
```

A.2 – run_ddpg_push.py

This Appendix outlines the code file for the reinforcement learning component using DDPG in RLlab simulator, adapted from the work of (Liu Y, 2018) and RLlab official documentation: https://rllab.readthedocs.io/en/latest/.

```
# The MIT License (MIT)

# Copyright (c) 2016 rllab contributors

# rllab uses a shared copyright model: each contributor holds copyright over
# their contributions to rllab. The project versioning records all such
# contribution and copyright details.
# By contributing to the rllab repository through pull-request, comment,
# or otherwise, the contributor releases their content to the license and
# copyright terms herein.

# Permission is hereby granted, free of charge, to any person obtaining a copy
# of this software and associated documentation files (the "Software"), to deal
# in the Software without restriction, including without limitation the rights
# to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
# copies of the Software, and to permit persons to whom the Software is
```

furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all

copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR

IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE

AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER

LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE

SOFTWARE.

Imports

import os

RLLab

from rllab.baselines.linear_feature_baseline import LinearFeatureBaseline

```
os.environ["THEANO_FLAGS"] = "device=cpu"
from rllab.envs.gym_env import GymEnv
from rllab.envs.normalized_env import normalize

# RLLab policy import
from rllab.policies.gaussian_mlp_policy import GaussianMLPPolicy
from rllab.envs.normalized_env import NormalizedEnv

# RLLab DDPG import
from rllab.algos.ddpg import DDPG
from rllab.misc.instrument import stub, run_experiment_lite
import itertools
from rllab import config

stub(globals())

from distutils.dir_util import copy_tree
import numpy as np
import os, shutil
# Assigning model directory
srcmodeldirs = ['../reinforcement/modelskip/']
modeldir = 'reinforcement/'
if os.path.exists(modeldir):
```

```
    shutil.rmtree(modeldir)


for srcdir in srcmodeldirs:
    copy_tree(srcdir, modeldir)



# Determining randomised colour of objects
def getcolor():


    color = np.random.uniform(low=0, high=1, size=3)
    while np.linalg.norm(color - np.array([1.,0.,0.])) < 0.5:
        color = np.random.uniform(low=0, high=1, size=3)
    return color



# Determining which object to push
def rand_push():
    # Looping
    while True:
        # Randomising object positions
        object_ = [np.random.uniform(low=-1.0, high=-0.4),
                   np.random.uniform(low=0.3, high=1.2)]
        goal = [np.random.uniform(low=-1.2, high=-0.8),
```

```
                np.random.uniform(low=0.8, high=1.2)]
        if np.linalg.norm(np.array(object_)-np.array(goal)) > 0.45: break
    geoms = []
    for i in range(5):
        # Randomising start position
        pos_x = np.random.uniform(low=-0.9, high=0.9)
        pos_y = np.random.uniform(low=0, high=1.0)
        rgba = getcolor().tolist()
        isinv = 1 if np.random.random() > 0.5 else 0
        geoms.append([rgba+[isinv], pos_x, pos_y])
    vp = np.random.uniform(low=0, high=360)

    return dict(nvp=1, vp=vp, object=object_, goal=goal, imsize=(64, 64), geoms=geoms,
        name="push", modelname='Models/ctxskiprealtransform127_11751',
        meanfile=None, modeldata='Models/greenctxstartgoalvpdistractvalid.npy')



# Setting up 3DoF pusher environment
push_params = {
    "env" : "Pusher3DOF-v1",
    "rand" : rand_push,
}
```

```
# Defining modes
oracle_mode = dict(mode='oracle', mode2='oracle')
ours_mode = dict(mode='ours', mode2='ours', scale=1.0)
ours_recon = dict(mode='ours', mode2='oursrecon', scale=1.0, ablation_type='recon')
tpil_mode = dict(mode='tpil', mode2='tpil')
gail_mode = dict(mode='tpil', mode2='gail')
nofeat = dict(mode='ours', mode2='nofeat', scale=1.0, ablation_type='nofeat')
noimage = dict(mode='ours', mode2='noimage', scale=1.0, ablation_type='noimage')

# Setting seed for repeatability
seeds = [42]

sanity = None

for params in [push_params]:

    for nvar in range(10):
        randparams = params['rand']()
```

```
for modeparams in [ours_mode]:


    for scale in [0.1, 1.0, 10.0]:
        # Updating parameters
        copyparams = randparams.copy()
        copyparams.update(modeparams)
        copyparams['scale'] = scale
        # Defining MDP
        mdp = normalize(GymEnv(params['env'], **copyparams))


        if copyparams['mode'] == 'tpil':
            if sanity == 'change1':
                copyparams = params['rand']()
                copyparams.update(modeparams)
                mdp2 = normalize(GymEnv(params['env'], **copyparams))
            elif sanity == 'same':
                mdp2 = mdp
            elif sanity == 'changing':
                mdp2 = normalize(GymEnv(params['env'], mode='tpil'))


        if 'imsize' in copyparams:
            imsize = copyparams['imsize']
```

```
for seed in seeds:
    # Gaussian MLP Policy
    policy = GaussianMLPPolicy(
        env_spec=mdp.spec,
        hidden_sizes=(32, 32),
        init_std=10
    )

    baseline = LinearFeatureBaseline(
        mdp.spec,
    )

    batch_size = 50*250

    # DDPG Algorithm
    algo = DDPG(
        env=mdp,
        policy=policy,
        baseline=baseline,
        batch_size=batch_size,
        whole_paths=True,
        max_path_length=50,
        n_itr=150,
```

```
        step_size=0.01,
        subsample_factor=1.0,
        discount=0.99,
        **copyparams
    )

# Running experiment
run_experiment_lite(
    algo.train(),
    exp_prefix="sim_push_DDPG",
    n_parallel=4,
    snapshot_mode="all",
    seed=seed,
    mode="ec2_mujoco",
)
```

A.3 Data_collect.py

This appendix outlines the code used to generate the real-world dataset examples.

```
import numpy as np

import os

import time

import cv2


# Initializing Camera_1

video_capture_0 = cv2.VideoCapture(2)

# Setting read resolution

video_capture_0.set(3, 48)

video_capture_0.set(4, 48)


# Setting capture resolution

cap_size0 = (int(video_capture_0.get(3)), int(video_capture_0.get(4)))


# Initializing Camera_2

video_capture_1 = cv2.VideoCapture(0)

# Setting read resolution

video_capture_1.set(3, 48)

video_capture_1.set(4, 48)
```

```
# Setting capture resolution
cap_size1 = (int(video_capture_1.get(3)), int(video_capture_1.get(4)))

# Defining video encoding (XVID)
fourcc = cv2.VideoWriter_fourcc('X', 'V', 'I', 'D')

i = 142

print('Recording starting.')
# Initializing the VideoWriter class - Try 15 fps
out_0 = cv2.VideoWriter(f'Dataset2/train/{i}.avi', fourcc, 10.0, cap_size0)
out_1 = cv2.VideoWriter(f'Dataset2/train/{i+1}.avi', fourcc, 10.0, cap_size1)

frameCount = 0

while True:
    # Reading frame-by-frame
    ret0, frame0 = video_capture_0.read()
    ret1, frame1 = video_capture_1.read()

    if (ret0 and ret1):
        # Displaying image
        cv2.imshow('Cam 0', frame0)
```

```
        cv2.imshow('Cam 1', frame1)
        # Writing image to file
        out_0.write(frame0)
        out_1.write(frame1)

        frameCount += 1
        print(frameCount)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        out_0.release()
        out_1.release()
        break

    if frameCount >= 80:
        break


# Release the capture & videowriter
video_capture_0.release()
video_capture_1.release()
cv2.destroyAllWindows()
```