Sensor-based Robot Control: using Deep Reinforcement Learning

by Abayomi Omotosho-Ikuru
Student ID: F136167

Loughborough University

21COP327: Artificial Intelligence Project

Supervisor: Dr Qinggang Meng
Submitted on 26th August 2022

Abstract

Robots are increasingly being used to solve complex tasks and Deep Reinforcement Learning (DRL) has been the driving force. This project approaches DRL from the basics, exploring the methods, processes and challenges faced designing and setting up a robot arm to perform simple movement tasks using the ubiquitous camera sensor with a depth component. Training the robot is performed entirely in Isaac Gym, a highly-performant and accurate physics simulator using learning algorithms that can learn to control the 6 degrees of joint movement on the robot and understand the given task. Experiments are carried out to objectively measure the performance of the task given to the robot and to compare the different algorithms and modifications made to the simulated environment. Some interesting challenges were also faced that proved unsolvable during the duration of this project, discussed in this paper.

Key words: Deep Reinforcement Learning, Robot Manipulator, Vision-based

Acknowledgements

I would like to thank my supervisor, Dr. Qinggang Meng for the support and guidance offered during the development of this project. I would also like to thank Dr. Mohammed Saada for the advice and assistance he offered, helping me solve some technical challenges faced. James Skevington from the IT department was also instrumental to setting up the compute workstation needed for training.

Contents

1 Introduction

1.1 Background

Robots are increasingly being used to solve complex tasks in the real world, this has largely been facilitated by Deep Reinforcement Learning (DRL). DRL enables a robot to learn how to perform complex tasks and interact with its environment through trial and error. The robot is able to receive feedback as rewards from its environment using various sensors to then perform an action based on its state. This is the basis of the agent-environment interaction in Reinforcement Learning (RL) shown in Figure 1.

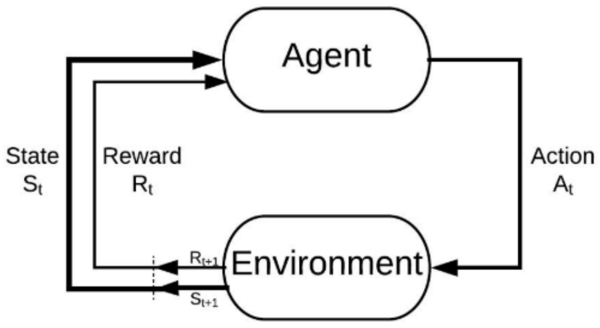

Figure 1: Reinforcement Learning agent-environment interaction. Diagram from Sutton and Barto (2018).

DRL has seen an increase in popularity as a result of its great success in multiple areas, performing as well or better than humans in certain tasks. An example is AlphaGo being able to beat a professional human player in the game of Go ("Mastering the game of Go with deep neural networks and tree search" 2016). DRL differs from RL by employing the use of neural networks to learn complex multi-dimensional tasks. This allows robots to be used in dynamic environments that can not be solved using programmed solutions.

Robot manipulation in the past involved an extensive use of sensors which required an accurate model of the robot. Algorithms such as inverse kinematics (D'Souza, Vijayakumar, and Schaal 2001) were required to position and orient a robot. The use of sensors evolved to combine sensor data using sensor-fusion (Harashima 1990; Sasiadek 2002), this provided a more consistent approximation of the actions of the robot and its environment.

Similarly, the characteristics and deficiencies of the sensors in a sensor-fusion setup were baked into the model that drives the robot which is a limiting factor for complex tasks.

DRL in conjunction with vision-based sensors has largely removed the requirement of having fixed models that pool sensor data. Robots are able to perform actions using visual-feedback that a neural network is trained on. This has greatly improved the capabilities of a robot to truly act autonomously without human intervention.

The current challenges faced using DRL for robotics are largely a result of the complexity of actions and variability of its environment much like in the real world. As of this moment, DRL models requires a lot of computational resources to train, this exponentially increases as the number of states and actions of the robot increases. This is also a result of sample inefficiency in continuous action space (model-free) RL algorithms. DRL experiments are not typically reproducible as the manner of learning is intended to be stochastic. This can often lead to scenarios where a DRL model performs completely differently on different software versions and compute machines even with the same initialisation seed. Knowledge transfer is also a challenge which when solved could improve the adaptability of a robot trained on one task to perform well on another task. Meta-learning (Hospedales et al. 2020), the ability of the DRL model to optimise its learning parameters to adapt to changes is actively being researched and developed on, with the end goal of continual life-long learning for robots (Parisi et al. 2018).

1.2 Aims and Objectives

This project aims to explore and develop a DRL model for use with a robot manipulator to perform simple tasks. The robot manipulator available during this project is the Niryo One 6-axis robot as shown in Figure 2.

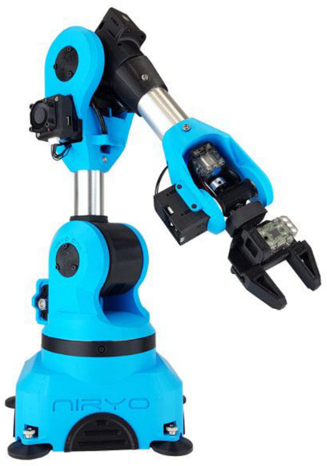

Figure 2: Niryo One 6DOF robot arm (Niryo 2019)

The initial aim during the preparation of the project was to train a DRL model on the task of picking up and placing down objects with the robot's gripper but the scope of the project has been narrowed due to challenges faced in the initial development of the DRL model. The revised aim of the project is to train a DRL model to move the end-effector of the robot to the position of an object in its environment. The project aims to use sim2real transfer, training the robot in a simulation environment before being transferred to the real robot to speed up the training process. The objectives of the project have remained largely the same and are as follows:

1. **The simulator**: selecting a suitable physics simulation environment to facilitate training the model on the task. The simulator would need to support loading the robot URDF (Unified Robotics Description Format) file to accurately model the robot's movements and collision boxes. The simulator would also require support to interface with ROS (Robot Operating System) (Quigley et al. 2009) for use with the physical robot during Sim2Real transfer.

2. **The vision system**: determining the optimal setup for the vision system to enable the robot to learn and perform the task. This would involve selecting and positioning an RGB camera or cameras to use in the environment to observe the robot and the object. The importance

of depth information in the robot's ability to perform the task would also be measured.

3. **Task performance**: selecting a DRL algorithm, collecting and measuring model training statistics such as average reward, policy-gradient loss, value loss and training time. Comparing multiple training times and observing the actions of the simulated robot and the physical robot. Tuning the hyper-parameters of the model and shaping the rewards received during training to observe how it affects the learning process.

The challenges faced with the initial aim are largely a result of the time that would be required train and test multiple iterations of the DRL model due to the increased complexity of the task. Adding the action of picking up and placing down objects was forecasted to triple training time based on approximations from DRL research examples found during the development of this project.

1.3 Outline

The remainder of this paper will be structured as follows; section 2 provides a review of the literature, detailing the algorithms, performance and techniques used for similar DRL and robot manipulation tasks. Section 3 shows the methodology and the design of the simulation environment and Sim2Real transfer methods. Section 4 shows the training iterations, training statistics and other experiments run, and evaluates the results of the experiments against the objectives set out. Forming the basis of the conclusion, the limitations and future work in section 5.

2 Literature Review

This section further introduces the main aspects of reinforcement learning and will then look into RL and DRL research in broader terms, and research more specific to operating robots in the context of this project.

2.1 Background

RL falls under the bracket of Machine Learning but is not typed as a supervised learning or an unsupervised learning algorithm (Sutton and Barto 2018). The agent in an RL task attempts to maximise the rewards received as opposed to identifying an inherent data structure or data distribution. The agent may also be in a number of different states while interacting with its environment. To model an RL task, Markov Decision Processes (MDP) is used and is represented by:

$$(S, A, P_a, R_a, \gamma) \tag{1}$$

Where S is a set of possible states called the states space. A is a set of possible actions called the action space. P_a is the probability that an action a at a specific time will lead to another state at a future time. R is the reward received after a state transition due to action a and γ is the discount factor that determines the importance of immediate to future rewards. The state and action spaces may be finite or infinite and the state transitions may also deterministic or stochastic.

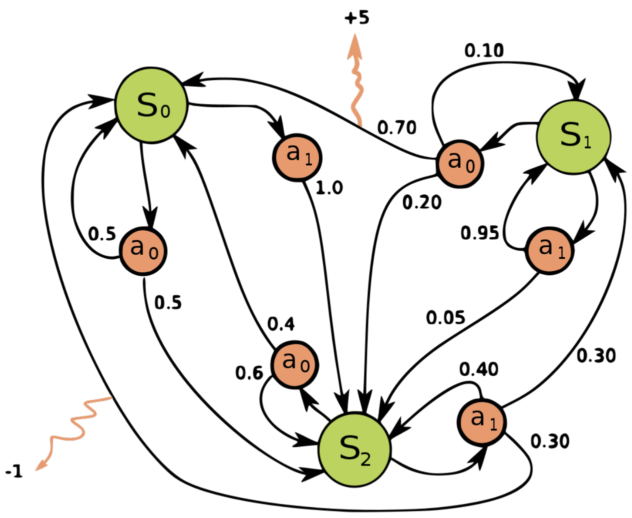

Figure 3: Markov Decision Process diagram showing states in green, actions in orange, state transitions as black arrows and rewards as orange arrows (*Markov decision process* 2022).

$$\pi(a|s) = P[A_t = a|S_t = s] \tag{2}$$

The agent is governed by a policy that determines its actions at any time. The policy function (equation 2) is a probability distribution over all possible actions (A) for all possible states (S). The policy-function may be classed as on-policy or off-policy. On-policy learning methods actively control the actions of the agent while off-policy learning methods improves the policy that does not directly influence the actions of the agent.

As mentioned previously, the agent aims to maximise the rewards received over a period of time. Two challenges associated with rewards are sparse rewards and the credit assignment problem. The rewards an agent receives is designed specifically to the action the agent is intended to perform and sometimes the agent may fail to randomly perform the action due to the complexity of the task. The rewards the agent receives are sparse and the agent has difficulties learning. Reward shaping is used to solve this by modifying the reward function such that the agent receives rewards when performing prior-actions that lead to intended action - guiding the agent. By guiding the agent, the challenge of credit assignment is reduced. Sometimes the RL algorithm is unable to determine the correct series of actions that led to maximising the reward. This may cause mostly good actions being discarded as a result of bad actions that had more severe penalties. This challenge can sometimes be overcome by more training, increasing the experience of the agent and optimising for future rewards sometimes at the expense of immediate rewards.

$$V^{\pi}(s) = E_{\pi}\{R_t|s_t = s\} = E_{\pi}\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\} \tag{3}$$

$$Q^{\pi}(s,a) = E_{\pi}\{R_t|s_t = s, a_t = a\} = E_{\pi}\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s, a_t = a\} \tag{4}$$

The state-value function estimates what state is good for the agent to be in, regarding the rewards received. It estimates the rewards of the agent based on the agent's future states. Equation 3 shows the value of a state under a policy π where E_{π} denotes the expected value given that the agent follows the policy at a particular time t. The action-value function in equation 4 similarly estimates the rewards received based on the actions that the agent can perform.

RL algorithms may be classed as model-based or model-free. The MDP shown in Figure 3 is an example of dynamic programming (Bellman 1966) where the states of the agent in the environment are defined. As a result, the state transition probability P and reward for each step can be learned. This is referred to as a model-based RL task. A model-free RL algorithm instead is more suited to tasks without a defined environment. It trains its policy by interacting with its environment by trial and error. Doing so, the RL agent learns how the environment responds to its actions. Model-free algorithms make little assumptions about their environment and rely mainly on the reward function. They are good at learning complex policies but are sample inefficient, requiring a lot of experience which is slow, and the learned experience are not transferable. Model-based algorithms instead are sample efficient and learned experiences are more easily transferable across tasks but they have poor task optimisation and need assumptions to learn complex skills.

Convolutional Neural Networks (CNN) are important to the vision system in DRL. CNNs are modelled after the biological visual system and can extract features and patterns from image frames (O'Shea and Nash 2015). It extracts spatial information using relevant filters in its structure. This is an important feature for autonomous robot applications and the CNN is also able to act as an RL policy, deciding the actions of the agent.

Vision-based systems are preferred for autonomous robot applications due to the flexibility of the robot agent to perform a task without modelling the robot itself or its environment. Without a vision-based DRL algorithm inverse kinematic algorithms would need to be developed specifically for each robot manipulator like the Niryo One being used and in a closed-loop feedback system, would require a fixed mounted vision system with a known position relative to the robot. The task would also need to be carefully designed, accounting for as many scenarios the robot will encounter. As a result, vision-based DRL solutions excluding the computer for deep learning training, tend to be of lower cost because it can utilise fewer low-cost sensors.

2.2 Research

Similar projects and research related to this paper have been carried out using various DRL methods. One of the first DRL milestones was the Deep Q-Network (DQN) developed in 2013 by DeepMind which performed with human level proficiency in Atari games using raw pixel data as an input (Mnih et al. 2013). This can be seen as the foundation of vision-based DRL.

The first DRL robot demonstration by Levine, Wagener, and Abbeel (2015) learned and optimised trajectories for robot manipulation using a non-linear neural network trained with a guided policy search algorithm. This allowed the robot manipulator to generalise how to move from one target location to another. Behaviours such as stacking Lego blocks and assembling a toy airplane were able to be learned. The algorithm used was also sample efficient but knowledge of the state of the environment was required during training.

Levine, Pastor, et al. (2016) conducted another research in 2016, learning hand-eye coordination for robotic grasping. Using an array of 6 to 14 monocular cameras and robot manipulators, a large CNN was trained to predict the probability of a successful grasp of objects in trays below the robot manipulator's gripper. The experiment collected over 800,000 successful and failed grasps in a period of two months. Variations in the array of setups was introduced by varying camera pose relative to the robot, independently calibrating the cameras, and the uneven wear from the robot and grippers. The trained network was largely guided by the CNN with the RL component acting secondarily. The CNN received two input images, one before the grasp attempt guided by the RL algorithm, and a current image. The CNN then predicts the probability of a success grasp given the current motion vector, aiming to maximise the probability. The results were an average of ¡20% grasp failure rate for novel objects on the first 30 attempts. The network was also able to identify and modify its grasping strategy for varying object sizes and material properties such as soft and hard object.

Deep Deterministic Policy Gradient (DDPG), an algorithm proposed by Lillicrap et al. (2015) solves the discrete action space limitation with DQN. DDPG is an actor-critic model-free algorithm with a policy that works with continuous action spaces, performing as well as DQN for low-dimensional observations. The actor in the actor in the actor-critic relationship is the policy neural network that estimates the action-value function, while the critic neural network estimates the state-value function. An experiment carried out to test its performance used a simulated 2D robot, learning an action policy directly from pixels. This experiment shows that DDPG is a good candidate for robot control.

Vecerik et al. (2017) proposed a solution to DRL robot control by leveraging the off-policy memory-replay in DDPG. Demonstrations of task transitions created by the researchers were injected into the replay buffer and used in conjunction with reward shaping to quickly improve the knowledge base of the robot agent for challenging tasks. Experiments carried out showed good performance on the task with as little as one human demonstration. It also allowed the robot to learn faster than the unmodified DDPG. The possible limitations of this algorithm are the reduced opportunity for the robot to explore its environment and action space, and increased complexity in shaping rewards for multi-stage tasks.

Hindsight Experience Replay (HER) (Andrychowicz et al. 2017) proposed by OpenAI builds on DDPG and is similar to the modified DDPG proposed by Vecerik et al. (2017). It eliminates the need to engineer the reward function as a result of sparse rewards. It does so by storing and replaying experiences with different potential goals. The researcher explains it using the analogy that "a sequence of actions would be successful if the net had been placed further to the right" for a game of hockey. This is intended to say that the failed action was not completely useless. As a result, HER is sample efficient by being able to learn from very sparse rewards. Experiments using DDPG+HER were carried out in a non-standard MuJoCo (Todorov, Erez, and Tassa 2012) simulation environment that had a 7DOF robotic arm with a gripper. Pushing, sliding and pick-and-place tasks were trained on the robot and shown to perform better than DDPG. The algorithm was also deployed to a physical robot without fine-tuning and showed robustness to small errors after training in the simulator. Adding noise to the visual observation further improved the success rate.

Proximal Policy Optimisation (PPO) is an on-policy algorithm that can be used with discrete and continuous action spaces, that maximises the biggest improvement to policy in one time step using available data but without moving too far in a step to the point of performance collapse. Proposed by Schulman, Wolski, et al. (2017), PPO builds on TRPO (Schulman, Levine, et al. 2015) while being simpler to implement. It removes the need to compute KL divergence by clipping the change that the policy can make to prevent large gradient steps. This also enable the PPO algorithm to undo the previous gradient step.

Soft Actor-Critic (SAC) proposed by Haarnoja, Zhou, et al. (2018) is an off-policy algorithm with a policy that is trained to maximise entropy - randomness in the policy. This results in the RL agent opting to explore more to increase learning rate and this can prevent the algorithm from getting stuck in a bad local optimum. SAC is robust and sample-efficient enough to perform as well or better in real-world experiments when comparing with TD3 (Fujimoto, Hoof, and Meger 2018) and PPO (Schulman, Wolski, et al. 2017). Haarnoja, Ha, et al. (2018) designed an experiment using a quadruped robot and using the SAC algorithm showing that real world RL learning can be done in a stable manner with minimal per-task tuning and without relying on a model or simulation. The quadruped with the proposed algorithm was able to achieve a stable gait within two hours of training, a first for a DRL trained entirely in the real world without pre-training in a simulator.

The Dreamer algorithm (Wu et al. 2022) is the most recent algorithm that inches more towards sample efficient DRL. Unlike the previous models discussed, Dreamer is a model-based algorithm that learns the world (environment) model from past experience. This has the advantage of generalising the dynamics of the environment that can be used for a wide range of tasks. Dreamer does not make use of simulators and the experiments carried out were done entirely on physical robots. Dreamer uses two neural networks, a world model learning network that uses a replay buffer and a highly parallelisable behaviour learning network using imagined rollouts that reconstructs sensory inputs. Experiments were carried out on a quadruped, two robot manipulators and a wheeled robot. In both robot manipulation tasks involving vision-based pick and place with and without depth, Dreamer performed near human level without the need to change hyper-parameters, outperforming PPO and Rainbow (Hessel et al. 2017) algorithms. The quadruped robot was able to walk within an hour with no prior experience or training, outperforming SAC on the same task. While the researchers note that learning on hardware over many hours creates wear on the robots, the speed at which the robots learn in comparison to other algorithms may be a worthwhile trade off.

2.3 Summary

The literature review shows the evolution of DRL with multiple examples for robot control using a mix of vision and ground-truth observations to learn complex tasks. Many more algorithms and modifications to the algorithms discussed exist but this covers the current most common and widely used algorithms for DRL.

3 Design

3.1 Introduction

This section details the steps and considerations taken to reach the objectives set out in Section 1. It also details the experimental methods used to evaluate the performance of the DRL agent - the robot, on the task.

3.2 Software

The project aims to simulate the Niryo One robot in a physics environment capable of accurately representing the forces generated by the stepper motors on the robot, as well as the physical dimensions of the robot to calculate collision boxes. The 3D mesh, joint-links, joint-drive forces and mass of components are present in the Unified Robotics Description Format (URDF) (Quigley et al. 2009) for the robot arm provided by Niryo in a GitHub repository (Niryo-Robotics 2018).

Nvidia Omniverse Isaac Gym (Makoviychuk et al. 2021) was selected as the GPU enabled simulator due to its ability to run the neural network policy training on the GPU with direct data passing from the physics buffer to PyTorch tensors without needed to go through the CPU. This greatly improves the training time by about 2 orders of magnitude. Isaac Gym also allows for thousands of parallel environments to be trained on a single GPU while supporting the ability to make variations in the different environments to improve its robustness - Domain Randomisation (Tobin et al. 2017). These were the main reasons to choose Isaac Gym over other popular physics engines for DRL namely; MuJoCo (Todorov, Erez, and Tassa 2012), PyBullet (Coumans and Bai 2016) and Unity (Juliani et al. 2018).

Isaac Gym also supports communications with Robot Operating System (ROS) (Quigley et al. 2009) to interface with physical robots. It is able to mirror poses of the physical robot in the simulator and it is able to send joint actions to the physical robot based of vision and sensor information received from the real world.

Isaac Gym offers two methods to setup an environment, using the GUI and using Python. The Isaac Gym simulator for this project is set up as follows. The .urdf.xacro file from the Niryo One GitHub repo is first converted to a .urdf which can be imported into Isaac Gym. The robot in the simulator is then set up to have a single RGB-D camera fixed in space relative to the base of the robot. The parameter of the camera such as focal length, aperture and aspect ratio are then modified to closely match the intended physical camera to be used which in this case is the Intel RealSense D435i (Keselman et al. 2017). The environment is then saved as a Universal Scene Description (USD) (PixarAnimationStudios 2019) file to be loaded using Python.

Using Python to setup the DRL environment allows for complex and fully custom setups that cannot be achieved with graph editor in the GUI. The Isaac Gym Python API is based on the OpenAI Gym standard API for RL (Brockman et al. 2016) with an env (environment) base class that has functions for time-stepping, resetting the environment, getting observations and computing rewards. The env class is created for the Niryo One, adding other necessary information and components such as robot world position and orientation, illumination in the form of light sources and a ground plane.

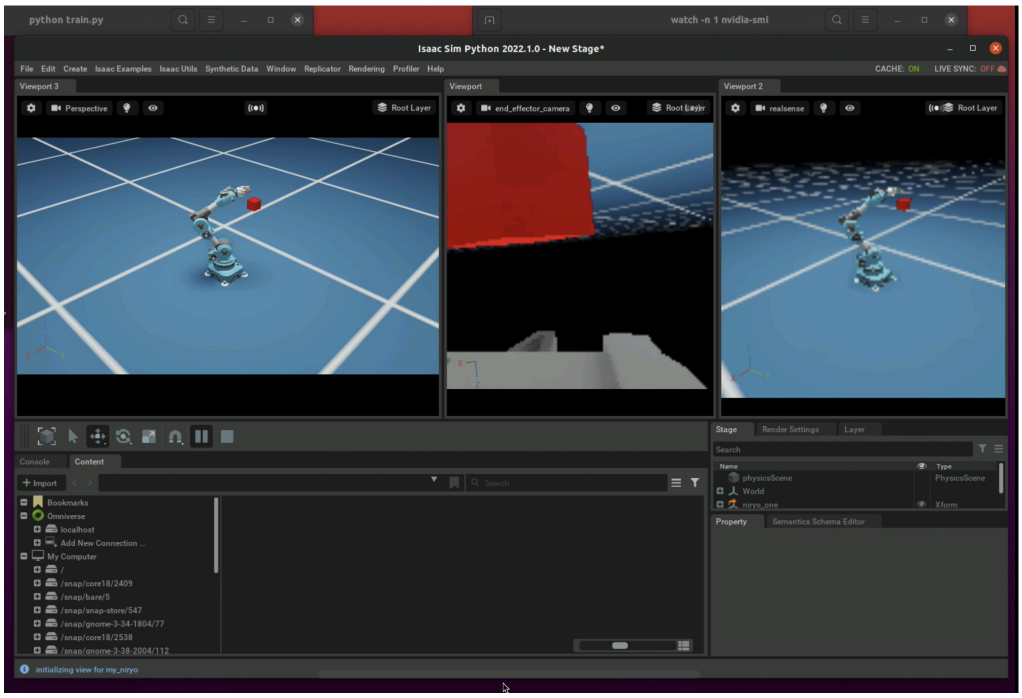

Figure 4: Omniverse Isaac Gym interface (Makoviychuk et al. 2021) showing initial training of the Niryo One robot with two rgb cameras. One camera on above the end-effector (middle) and a second camera overlooking the robot from a fixed position (right). The view on the left is the perspective view shown by the simulator to edit and configure the scene. A video of this training on YouTube can be found here: YomiTosh (2022).

3.3 Hardware

In addition to the Niryo One and the Intel RealSense camera, the computer for training makes up the hardware required for this project. The minimum requirements for training are a minimum of 32GB of general memory and a minimum of 8GB of video memory on a CUDA based graphics card. Training was performed on 2 machines during this project, an Nvidia RTX 3060Ti and an Nvidia RTX 3090. The results from both machines will be shown together without differentiation as it does not directly affect the performance of DRL training using the same parameters.

3.4 Experimental Methodology

The simulated environment is set up to have a red cube (the goal) positioned randomly in space shown in Figure 4 based on sine and cosine functions. The intended robot action is to move its end-effector to coincide with the center point of the red cube. The observation is a 256x256 image from the Intel RealSense camera pointed at the robot and the space around it to view the red cube as well. Two main reward functions were experimented with and shown below:

$$R1_t = norm(D_{t-1} - D_t) \tag{5}$$

$$R2_t = \begin{cases} R1_t + (0.01/norm(G_t - D_t)) & norm(G_t - D_t) < 0.2 \\ -norm(G_t - D_t) * 10 & norm(G_t - D_t) > 0.2 \end{cases} \tag{6}$$

Where $R1_t$ and $R2_t$ are the rewards at the current time t, $norm$ is the euclidean distance, D_t and D_{t-1} are the end-effector positions at the current time t and the previous time step $t-1$, and G_t is the position of the cube.

The action spaces are continuous, meaning that the actions given to the robot are non-discrete and thus requires a DRL algorithm that has a policy that can generate continuous values. The two best common algorithms that are suitable are PPO and SAC which are both tested. A CNN policy is used due to its well documented and state of the art performance in image feature extraction.

Two RL libraries were tested, Stable-Baselines3 (SB3) (Raffin et al. 2021) and SKRL (Serrano-Muñoz et al. 2022). Both have nearly identical feature-sets and performance but have different implementation methodologies. SB3 hides its internal structure using an easy-to-use API whereas SKRL is very modular and requires a good understanding of the components of the specific DRL algorithm to make use of its APIs.

The key metrics to be evaluated for training are average rewards over time, policy loss, value loss and a visual observation of the robot performing the task. The average rewards represents the robots ability to perform the task over time, this should increase over time showing that the robot is learning.

The policy loss indicates whether the policy is changing, a decreasing policy loss shows that the policy is giving the agent an action that earns the most rewards - understanding the task. The value loss indicates the models ability to predict the value at each state. This should increase during the early phases of training before decreasing as reward stabilises.

4 Experiments

4.1 Proximal Policy Optimisation

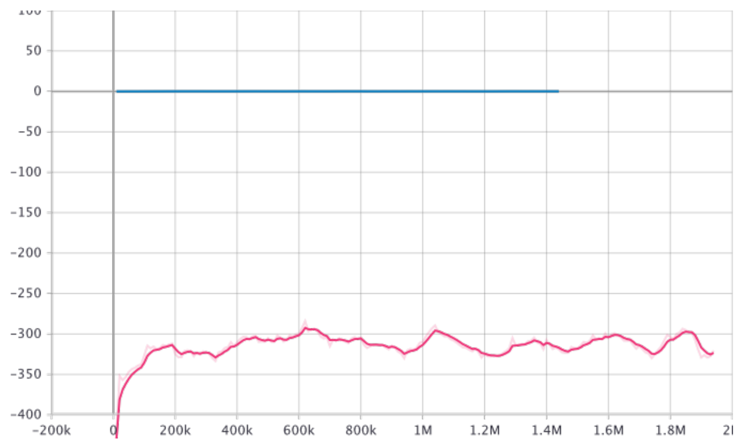

Figure 5: Average rewards for PPO-6 (blue) and PPO-8 (pink).

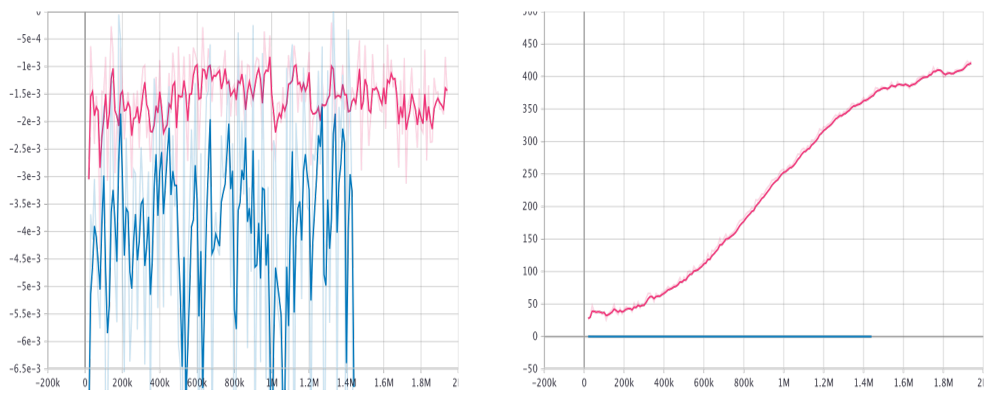

Figure 6: Policy-gradient loss (left) and value loss (right) for PPO-6 (blue) and PPO-8 (pink).

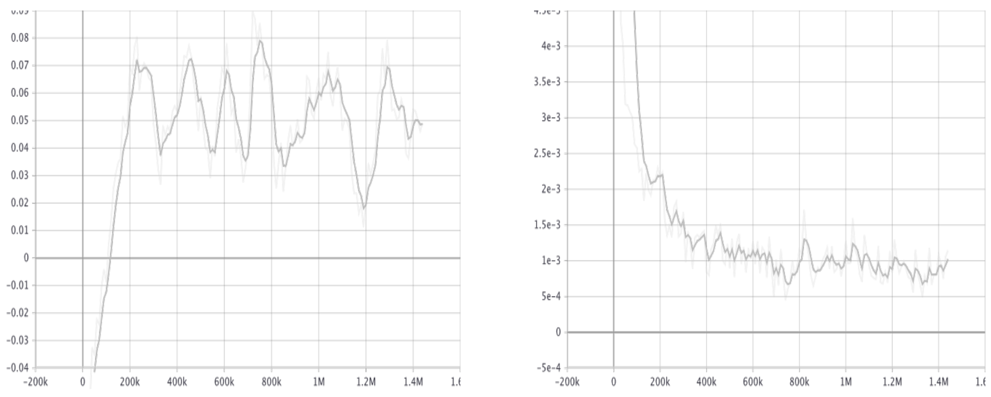
Figure 7: Scaled view of average rewards (left) and value loss (right) for PPO-6.

Figure 5, Figure 6 and Figure 7 show the results for the experiments, PPO-6 and PPO-8 over a number of timesteps. PPO-6 used the reward function from Equation 5 showing an initial growth of average rewards before reaching a point where the average rewards oscillated about 0.055. PPO-8 used the reward function from Equation 6, that introduced severe reward penalties for unintended actions. Both PPO-6 and PPO-8 used CNN policies with an RGBD observation as described in Section 3. The policy-gradient loss for both PPO-6 and PPO-8 show oscillations that trend sideways. The oscillations for PPO-6 were also more severe than PPO-8. The value loss showed different trending directions, with PPO-6 trending exponentially towards zero and PPO-8 trending almost linearly away from zero.

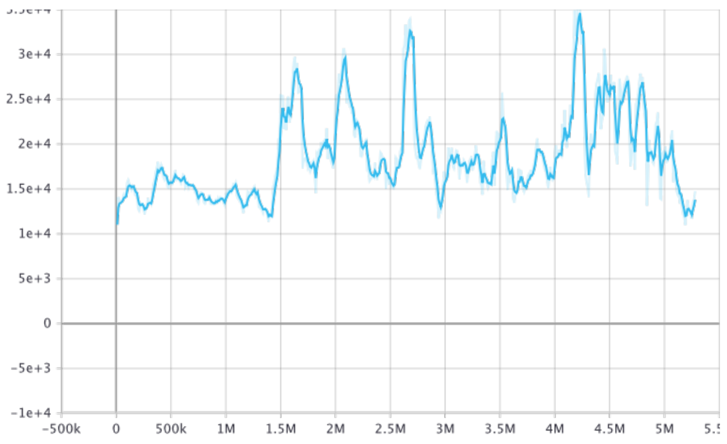
Figure 8: Average rewards for PPO-4.

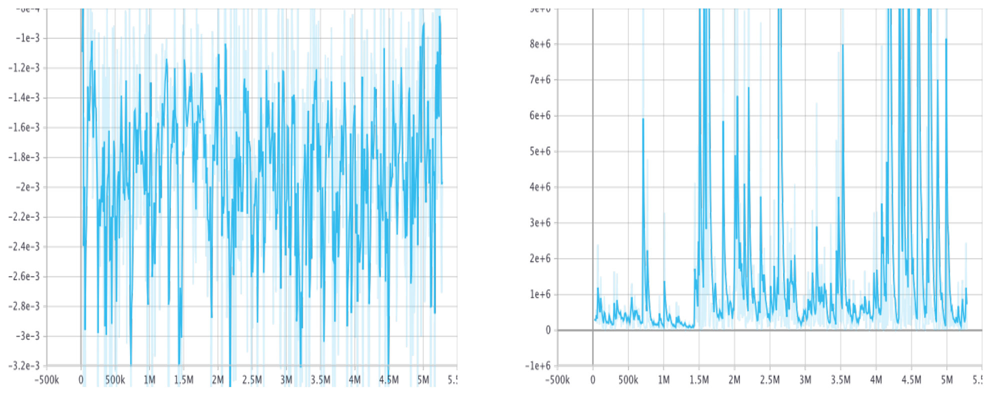
Figure 9: Policy-gradient loss (left) and value loss (right) for PPO-4.

$$R3_t = 1/norm(G_t - D_t)^2 \tag{7}$$

PPO-4 shown in Figure 8 and 9 uses a different reward function shown in Equation 7 in an attempt to exponentially increase the reward received when closer to the goal. The average rewards shows no observable trend and appears random. This is also the case for the policy-gradient loss and value loss.

4.2 Soft Actor-Critic

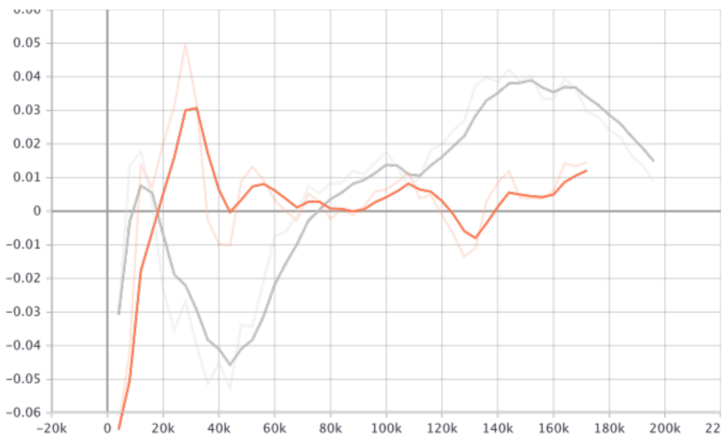
Figure 10: Average rewards for SAC-1 (grey) and SAC-2 (orange).

The experiments using the SAC algorithm is shown in Figure 10 and 11 for SAC-1 and SAC-2. SAC-1 used a replay buffer size of 75,000 and SAC-2

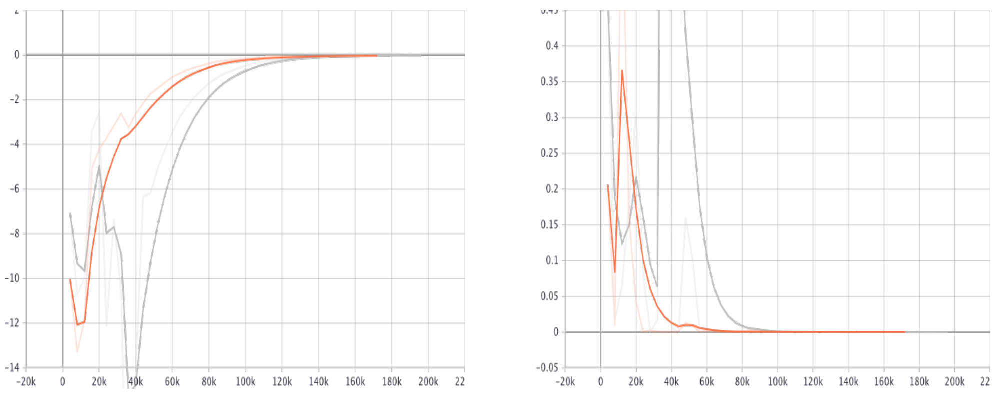

Figure 11: Actor loss (left) and Critic loss (right) for SAC-1 (grey) and SAC-2 (orange).

used a replay buffer size of 200,000. The average rewards for SAC-1 shows a general trend upward and SAC-2 shows a sideways trend. Both actor and critic losses are similar for SAC-1 and SAC-2, showing a trend towards zero in both cases.

Training times for the simulations were dependent on the framerate of the simulator, the episode length and the algorithm used. The PPO algorithm was able to maintain a stable 56 frames per-second (fps), taking about 45 minutes to complete 100,000 steps of 1000 episodes in a single robot environment. SAC was much slower at 13fps due the inverse relation with the size of its replay buffer.

4.3 Discussion

The results from the experiments have been unfavourable, many modifications to the episode length, camera setup, policy and hyper-parameters not shown were made but with little success. The model evaluation in the simulator showed the robot moving to a fixed position and remaining at the fixed position after every reset. It appeared that the agent was stuck in a local optima. This was the same for algorithms like PPO-4 with graphs that appeared random and SAC-1 that appeared to be learning by accumulating more rewards over time. The policy loss and value loss figures were evaluated and included as it could show if the agent is learning while the rewards are low. The policy loss should decrease for an agent that is learning and the value loss should initially increase but stabilise and decrease as the agent learns and the rewards stabilise. The graphs for SAC seemed to show that the agent was learning but the model evaluation showed the same fixed position behaviour.

As a result of the poor performance in the simulator, the Sim2Real objective could not be carried out as there was not a suitable model to deploy to the physical robot that would be safe for observers and the robot.

5 Conclusion and Future Work

5.1 Conclusions

This paper explored the use of Deep Reinforcement Learning (DRL) for robot manipulation tasks, primarily using vision-based control. Nvidia Isaac Gym, an accurate physics simulator was used to carry out experiments with the aim of performing a Sim2Real transfer. Two DRL algorithms, Proximal Policy Optimisation and Soft Actor-Critic were experimented with, and their performance on a simple task of moving the robot's end-effector to a target location was documented. Unfortunately, the simulated agent failed to learn the intended behaviour and as such, the DRL model could not be deployed to the real robot. The project however did not result in a total loss, as performance comparisons albeit limited could be made between the various configurations experimented in the simulated environment.

5.2 Limitations

Some limitations were encountered during the development of this project. The simulation environment, Isaac Gym does not currently have an API that supports cameras in parallel environments. This greatly reduces the learning sample size available, which is especially important for on-policy DRL algorithms like PPO. Access to a compute workstation with sufficient main memory and video memory to carry out multiple simulations in parallel was also limited.

5.3 Future Work

The intended future work would continue experimenting on reasons why the robot was unable to learn the assigned task. Performance using parallel environments would also be tested once an API is available from Nvidia. The viability and limitations of Sim2Real transfer would also need to be explored in addition to real world sample-efficient training of robot manipulators that does not require a simulator.

References

Andrychowicz, Marcin et al. (2017). "Hindsight experience replay". In: *Advances in neural information processing systems* 30.

Bellman, Richard (1966). "Dynamic programming". In: *Science* 153.3731, pp. 34–37.

Brockman, Greg et al. (2016). *OpenAI Gym.* eprint: arXiv:1606.01540.

Coumans, Erwin and Yunfei Bai (2016). *PyBullet, a Python module for physics simulation for games, robotics and machine learning.* URL: http://pybullet.org.

D'Souza, Aaron, Sethu Vijayakumar, and Stefan Schaal (2001). "Learning inverse kinematics". In: *Proceedings 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems. Expanding the Societal Role of Robotics in the the Next Millennium (Cat. No. 01CH37180).* Vol. 1. IEEE, pp. 298–303.

Fujimoto, Scott, Herke van Hoof, and David Meger (2018). *Addressing Function Approximation Error in Actor-Critic Methods.* DOI: 10.48550/ARXIV.1802.09477. URL: https://arxiv.org/abs/1802.09477.

Haarnoja, Tuomas, Sehoon Ha, et al. (2018). *Learning to Walk via Deep Reinforcement Learning.* DOI: 10.48550/ARXIV.1812.11103. URL: https://arxiv.org/abs/1812.11103.

Haarnoja, Tuomas, Aurick Zhou, et al. (2018). *Soft Actor-Critic Algorithms and Applications.* DOI: 10.48550/ARXIV.1812.05905. URL: https://arxiv.org/abs/1812.05905.

Harashima, Fumio (1990). "Sensor based robot control systems". In: *Proceedings of the 1990 IEEE Colloquium in South America, COLLOQ 1990*, pp. 203–208. DOI: 10.1109/COLLOQ.1990.152832.

Hessel, Matteo et al. (2017). *Rainbow: Combining Improvements in Deep Reinforcement Learning.* DOI: 10.48550/ARXIV.1710.02298. URL: https://arxiv.org/abs/1710.02298.

Hospedales, Timothy et al. (2020). *Meta-Learning in Neural Networks: A Survey.* DOI: 10.48550/ARXIV.2004.05439. URL: https://arxiv.org/abs/2004.05439.

Juliani, Arthur et al. (2018). *Unity: A General Platform for Intelligent Agents.* DOI: 10.48550/ARXIV.1809.02627. URL: https://arxiv.org/abs/1809.02627.

Keselman, Leonid et al. (2017). *Intel RealSense Stereoscopic Depth Cameras.* DOI: 10.48550/ARXIV.1705.05548. URL: https://arxiv.org/abs/1705.05548.

Levine, Sergey, Peter Pastor, et al. (2016). *Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection.* DOI: 10.48550/ARXIV.1603.02199. URL: https://arxiv.org/abs/1603.02199.

Levine, Sergey, Nolan Wagener, and Pieter Abbeel (2015). “Learning Contact-Rich Manipulation Skills with Guided Policy Search”. In: DOI: 10.48550/ARXIV.1501.05611. URL: https://arxiv.org/abs/1501.05611.

Lillicrap, Timothy P. et al. (2015). *Continuous control with deep reinforcement learning.* DOI: 10.48550/ARXIV.1509.02971. URL: https://arxiv.org/abs/1509.02971.

Makoviychuk, Viktor et al. (2021). *Isaac Gym: High Performance GPU-Based Physics Simulation For Robot Learning.* DOI: 10.48550/ARXIV.2108.10470. URL: https://arxiv.org/abs/2108.10470.

Markov decision process (2022). Accessed: 2022-08-25. URL: https://en.wikipedia.org/wiki/Markov_decision_process.

"Mastering the game of Go with deep neural networks and tree search" (Jan. 2016). In: *Nature 2016 529:7587* 529.7587, pp. 484–489. ISSN: 1476-4687. URL: https://www.nature.com/articles/nature16961.

Mnih, Volodymyr et al. (2013). *Playing Atari with Deep Reinforcement Learning*. DOI: 10.48550/ARXIV.1312.5602. URL: https://arxiv.org/abs/1312.5602.

Niryo (2019). *Niryo One User Manual*. URL: https://niryo.com/docs/niryo-one/user-manual/complete-user-manual/ (visited on 05/12/2022).

Niryo-Robotics (2018). *ROS Stack*. URL: https://github.com/NiryoRobotics/ned_ros.

O'Shea, Keiron and Ryan Nash (2015). *An Introduction to Convolutional Neural Networks*. DOI: 10.48550/ARXIV.1511.08458. URL: https://arxiv.org/abs/1511.08458.

Parisi, German I. et al. (2018). "Continual Lifelong Learning with Neural Networks: A Review". In: DOI: 10.48550/ARXIV.1802.07569. URL: https://arxiv.org/abs/1802.07569.

PixarAnimationStudios (2019). *Universal Scene Description*. URL: https://openusd.org.

Quigley, Morgan et al. (2009). "ROS: an open-source Robot Operating System". In: *ICRA workshop on open source software*. Vol. 3. 3.2. Kobe, Japan, p. 5.

Raffin, Antonin et al. (2021). "Stable-Baselines3: Reliable Reinforcement Learning Implementations". In: *Journal of Machine Learning Research* 22.268, pp. 1–8. URL: http://jmlr.org/papers/v22/20-1364.html.

Sasiadek, J. Z. (Jan. 2002). "Sensor fusion". In: *Annual Reviews in Control* 26.2, pp. 203–228. ISSN: 1367-5788. DOI: 10.1016/S1367-5788(02)00045-7.

Schulman, John, Sergey Levine, et al. (2015). *Trust Region Policy Optimization.* DOI: 10.48550/ARXIV.1502.05477. URL: https://arxiv.org/abs/1502.05477.

Schulman, John, Filip Wolski, et al. (2017). *Proximal Policy Optimization Algorithms.* DOI: 10.48550/ARXIV.1707.06347. URL: https://arxiv.org/abs/1707.06347.

Serrano-Muñoz, Antonio et al. (2022). *skrl: Modular and Flexible Library for Reinforcement Learning.* DOI: 10.48550/ARXIV.2202.03825. URL: https://arxiv.org/abs/2202.03825.

Sutton, Richard S and Andrew G Barto (2018). *Reinforcement learning: An introduction.* MIT press.

Tobin, Josh et al. (2017). *Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World.* DOI: 10.48550/ARXIV.1703.06907. URL: https://arxiv.org/abs/1703.06907.

Todorov, Emanuel, Tom Erez, and Yuval Tassa (2012). "Mujoco: A physics engine for model-based control". In: *2012 IEEE/RSJ international conference on intelligent robots and systems.* IEEE, pp. 5026–5033.

Vecerik, Mel et al. (2017). *Leveraging Demonstrations for Deep Reinforcement Learning on Robotics Problems with Sparse Rewards.* DOI: 10.48550/ARXIV.1707.08817. URL: https://arxiv.org/abs/1707.08817.

Wu, Philipp et al. (2022). *DayDreamer: World Models for Physical Robot Learning.* DOI: 10.48550/ARXIV.2206.14176. URL: https://arxiv.org/abs/2206.14176.

YomiTosh (2022). *Nvidia Omniverse Isaac Gym DRL Training.* URL: https://youtu.be/vA678-93FRs.