

Bridging Reality Gap Between Virtual and Physical Robot through Domain Randomization and Induced Noise

Maresh Ranaweera^{†,*}, Qusay H. Mahmoud[†]

[†] Department of Electrical, Computer and Software Engineering
Ontario Tech University, Oshawa, ON, Canada

Abstract

This paper investigates techniques that can be utilized to bridge the reality gap between virtual and physical robots, by implementing a virtual environment and a physical robotic platform to evaluate the robustness of transfer learning from virtual to real-world robots. The proposed approach utilizes two reinforcement (RL) learning methods: deep Q-learning and Actor-Critic methodology to create a model that can learn from a virtual environment and performs in a physical environment. Techniques such as domain randomization and induced noise during training were utilized to bring variability and ultimately improve the learning policies. The experimental results demonstrate the effectiveness of the Actor-Critic reinforcement learning technique to bridge the reality gap.

Keywords: Deep Q-learning, actor-critic, reinforcement learning, transfer-learning, virtual-to-real, robotics

1. Introduction

Developing advanced robotic systems that can mimic, have dexterity, and adapt to real-world environments that are dynamic in nature is challenging. Moreover, it is a daunting task for pre-programming robots to traverse, visualize, and have dexterity in an unfamiliar environment. The uncertainty of adapting to different environments and making decisions based on unseen data can be time consuming and expensive. Therefore, most modern robotic systems and autonomous mobile robots are domain specific. Due to advancement in machine learning techniques, more and more robotic platforms are developed to perform advanced tasks. However, some of the major disadvantages of machine learning is the requirement of a large amount of quality training data. Acquiring such quality data is expensive, time consuming and often needs to be re-trained when the platform is deployed in an unseen environment. One of the emerging areas of machine learning is reinforcement learning (RL), in which the agent can interact with the environment to learn policies through imitation learning [1]. For generating vast amounts of training data, a simulated environment can be used. However, a simulation is unable to capture the environment, dynamics and the robot [2]. The inconsistency between physical parameters and dynamics such as gravity, friction, collisions, density and simulating soft surfaces are complex to model in a virtual environment. Also, noise introduced by sensors, gear backlash, latency and environment factors such as texture quality, reflection and refraction could adversely affect performance of the trained model [3]. This phenomenon is known as the Reality Gap.

The goal is to investigate, prototype and evaluate transfer learning from a virtual environment to a physical robot. A three degrees of freedom (DoF) Stewart platform was built to evaluate the accuracy of transfer by balancing a marble. A virtual environment was also developed to train the reinforcement learning algorithm. Additionally, this research contributes to closing the existing reality gap through domain randomization and inducing variability to enhance the learning on simulated environments. Deep Q-learning and Actor-Critic RL learning techniques were utilized in this research. To this end, the rest of

* mahesh.ranaweerakaluarachchige@ontariotechu.net

the paper is organized as follows. Section 2 summarizes the related work. The proposed methods are presented in Section 3, and the experimental setup in Section 4. Evaluation results are presented and discussed in Section 5. Finally, concluding remarks and ideas for future work are summarized in Section 6.

2. Related Work

There have been multiple research projects conducted on using virtual data or virtual environment to train a deep learning model. Sim2Real has become a research concept that describes the transferring of knowledge gained from a virtual simulation to a physical robotic system. One of the major problems in robot training is the difficulty of transferring the trained model to the real-world robot. This is often due to lack of quality training data, sample inefficiency and the cost of acquiring real-world data [4]. The main motivation of Sim2Real learning is to solve the domain gap due to insufficient training data by utilizing a virtual environment that allows to obtain necessary training data. Even if a model trained on a virtual environment was transferred to a physical system there is no guarantee that the model is able to adapt to physical dynamics. According to Lipton et al. this issue is due to Quasi static kinematics [5]. There are few methods proposed by researchers to improve the quality of transfer to close the reality gap. Some of the proposed methods are photo-realism rendering (realistic virtual environment), domain randomization and domain adaptation [6].

Domain randomization is a method that creates variety in the virtual environment by changing or randomizing environment or properties. This method that has been used by multiple researchers [7], [8] shows that it enforces the model to learn from a rich distribution of training data and variations. One of the major research projects conducted by OpenAI was teaching a multi dexterous robot arm to solve a Rubik’s cube [2]. They proposed a novel improvement to the domain randomization method called automated domain randomization (ADR). With ADR they were able to generate randomized environments with increasing difficulty. Their research suggests that this method vastly improves the control policies and vision estimates of a model. Results affirm that models trained with ADR show signs of meta-learning when compared with virtual and physical performance.

Domain adaptation is a subcategory of transfer learning used to improve the performance of a target domain. Wang et al. describe two methods of domain adaptation. They are one-step and multi-step domain adaptation [9]. The two methods can be used to update the data distribution in the simulation to match the real-world physical environment. Domain adaptation is currently being used in [10–12] for improving the quality of transfer in robot grasping. As described by Tobin et al. [7] domain adaptation methods include re-training the model on the target domain, performing intermediate learning steps to learn invariant features between the domains, learning mapping from target domain to source domain and learning invariant feature space to transfer skills from source agent to target agent [13].

Another method to bridge the reality gap is to closely match the virtual environment to the physical environment. This method involves creating a virtual environment that is closely related to the physical environment through high-fidelity rendering and simulations. Currently game engines and physics engines are utilized for performing simulations and learning. GPU based simulators and game engines such as NVIDIA Flex [14], Alphabet Soup, MuJoCo [15], RAWSim-O [16], Unreal Engine [17] and Unity are widely used. However, this method requires high computation power for rendering and physics calculations.

By accounting current methods and limitations, this research proposes an approach to augment the learning by utilizing a fidelity environment, domain randomization and inducing random noise to the simulated environment to increase the fidelity of learning. Antonova et al. [18] on their research on pivoting a tool held by a robot gripper shows that inducing randomized noise to physics, friction and delaying actions on the simulator has improved

the physical robot. It also shows that trained policies are robust and prone to mechanical errors and imperfections in the real-world robot gripper.

Since this area is still under research, some limitations and shortcomings were identified in the reviewed research. The reviewed research proposed a domain-randomization method to generalize the model. However, domain-randomization methods cannot randomize effects that are not modelled in the virtual environment and cannot further generalize the model. In order to obtain observations, OpenAI utilizes multiple sets of cameras, an advanced Giiiker cube with built in sensors and a cage to house the physical environment with ideal lighting conditions. Their implementation suffers from curse-of-dimensionality due to the sheer amount of sensor data; and therefore, requires a large volume of data to maintain the quality of learning. Their platform requires precise calibration of camera, field-of-view, sensors, lighting conditions and virtual environment. The present research, in contrast, randomizes the camera location, field of view, motion and lighting to induce variability and generalizes the reinforcement learning rather than relying on accurate camera position or calibrations. Additionally, the current research evaluates performance of learning through raw image data using Q-learning methodology and using Actor-Critic for making predictions based on marble position, velocity vector and the location relative to the center.

3. Methods

To bridge the gap between virtual and reality, two environments have been created: a virtual environment for simulation and training and a physical environment for evaluating the quality of transfer. A simple goal for agents was assigned to balance a marble on a platform. The goal was to find learning techniques, procedures and restrictions that arise when performing virtual to real-world transfer learning. Figure 1 shows the overview of the proposed transfer learning process. The simulated environment provides learning for the reinforcement learning models while simulating the physical system. Simulated environment contains the learning agent, dynamic simulation module, an environment model, and rendering engine. The learning agent is the deep learning algorithm. During the initial learning period, the agent observes the environment by performing random controlled actions. After the set learning period, agents perform replay actions based on the learning. The process continues until the agent is able to achieve a persistent score/reward or if the agent reaches a predefined close condition. For this research, two reinforcement algorithms were implemented. They are Actor-critic and Deep Q-learning. These algorithms will be further discussed in section 3.2.2. In each new iteration the virtual environments parameters are randomized to improve the overall learning. The dynamic simulation module performs the control commands received from the agents. This module actuates the virtual servos by applying corresponding forces and angular motion in pre-define vectors. The environment model handles the environment conditions such as randomizing lighting conditions, changing background textures and adding random noise. This also includes the virtual environment, a 3DoF Stewart platform. The rendering modules handle the rendering of the virtual environment. On the other hand, in the physical environment there are the same counterparts implemented for performing real-world actions.

Once the agent is trained in the simulation it is then transferred over to the physical environment. The agent can now make predictions about the observations received through the sensors. It returns a probability value for each action trained on the virtual environment. Probability is translated to real-world actuation through the Actuation module. The environment is a 3DoF Stewart platform. Camera module was used to track a target marble positioned on the Stewart platform. Observed data is then fed into the agent for making continuous predictions to keep the marble in place.

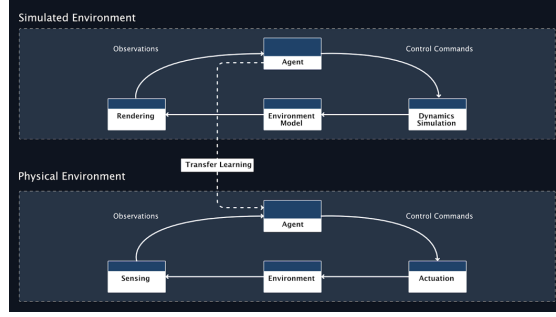


Figure 1. Overview of adapted transfer learning architecture

Both the virtual and physical environments were configured to the same physical attributes; such as number of degrees-of-freedom, angular limitations, joint limits, action delays, and surface hardness. Domain randomization techniques were employed in the virtual environment to randomize the scene to improve the learning. Domain randomization allows to increase the variability of learning [7].

3.1. Environment Setup

3.1.1. Virtual Environment

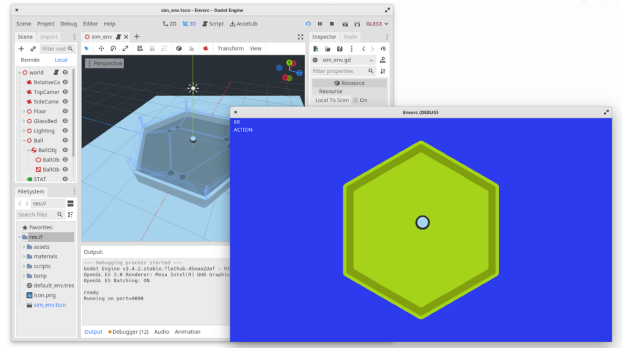


Figure 2. Virtual environment setup on Godot game engine

A virtual environment was created for simulating the real-world for training the model. Godot, an open-source game engine, was utilized for creating the virtual environment. The Virtual Stewart platform was designed using Autodesk Inventor to keep both physical and virtual environments uniform. Figure 2 shows the Godot game engine with the virtual environment setup for training. Motion of the platform, degrees-of-freedom are restricted based on the real-life physical environment. A WebSocket server was configured to the virtual environment to relay the actions during training, receive environment data, define environment variables, perform calibration and to reset the scene. Additional scripts were set up to receive command line inputs to initiate the simulation, setup WebSocket ports, and physics engine. Initiating simulation involves setting parameters to randomize the environment variables such as lighting condition, textures, adding camera noise, frame-rate, setting marble position, marble size, marble weight and surface friction. Additionally, instead of using Godot's built-in physics engine, it was configured to use Bullet, which is an open-source physics engine that is used for simulating advanced robotic simulations

[19]. Using this physics engine allowed us to configure GPU based optimization during the learning process and avoid CPU and memory bottleneck when multiple instances were invoked. The Godot scene which comprises the virtual environment was compiled to a standalone runnable executable.

3.1.2. Physical Environment

Physical environment was custom 3D printed using the Autodesk Inventor design. This allowed us to keep both virtual and physical environments uniform. As shown in figure 3 the Stewart platform was housed in a robot cage for stability and for housing attachments for monitoring the system. Single RGB camera module was placed in the center for capturing observations in the real-world. A Jetson nano compute module was used as the main computer for interfacing with inputs, making observations, performing predictions and handling servo controlling.

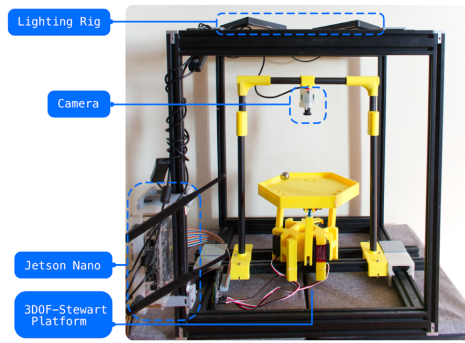


Figure 3. Physical Stewart platform housed in the robot cage

Each axis is controlled by a DS3218 PRO full metal gear digital servo. Four ball joints are used to create the linkage between the base and the axis arm. This allows the platform to gimbal based on each axis movement. The motion arc of the servo is restricted to 90 degrees from the idle position. On start-up the position of the axis is set to 45 degrees keeping the base leveled.

A generic 1080p camera was used to capture the frames which were then downscaled to 80x80 pixels based on the requirements. Camera was mounted on an adjustable jig to manually change the frame of view if necessary. The OpenCV library was used to detect the marble, track its position relative to the base, determine velocity vectors and pre-process frames for making predictions. A checkerboard calibration process was conducted on the physical camera to correct any imperfections or distortions. Field of view of the camera and the height from the camera to the base was kept at a constant. A RGB lighting rig was added to light the camera view for easy tracking of the marble. Varying environment light causes OpenCV object tracking to fail and often miscalculates the center of the marble.

3.1.3. Data Collection

During the learning process, the algorithm was set to perform following tasks to initiate the virtual environment; set environment parameters, create a socket connection for sending actions and receiving information from the virtual instance. Since reinforcement learning algorithms are used for learning, RL assumes that an agent is available in the environment [20]. The data collection module is set up to perform an action against the virtual environment in each step and receive observation through the WebSocket connection. The observation

contains information on roll, pitch, marble position, marble velocity, marbles distance to the center of the base plate and raw virtual camera image. The captured RGB image is limited to 640 by 360 pixels width and height respectively. Each new virtual instance will be assigned with a unique WebSocket port number which will be used to communicate and receive observations. Figure 4 shows the adapted flow diagram of the learning environment adapted from Open-AI gym.

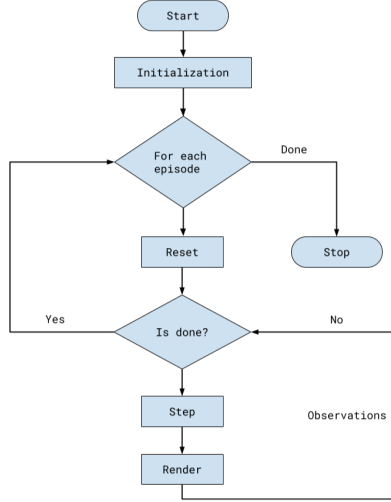


Figure 4. Flow diagram of the overall reinforcement learning algorithm

Based on the trained model, information from the observation is used to calculate the reward, update the learning weight, perform resetting and to terminate the learning. As shown in figure 5, three regions are marked to identify the location of the marble. If the marble is within the red zone, then a reward of one point will be offered. If the marble is within the red and blue zone the reward will be 0.5. If the marble reaches outside the blue zone, then the environment will be reset for next episode. In this way the learning algorithm will be able to reward if the marble is being navigated to the center of the base. Two Reinforcement learning algorithms were implemented: Actor Critic and Deep Q-learning. For the Actor critic algorithm, the position, velocity of the marble and the zone in which the marble is located was used as the observations. On the other hand, the Q-learning algorithm uses the image frame data. The captured frames are cropped and downscaled to 80x80 pixels. The information is then input into the convolutional layers.

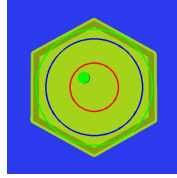


Figure 5. Virtual regions marked to calculate the reward function

3.2. Model Architecture

In this research two RL algorithms were implemented to evaluate progress of learning when Domain randomization and inducing random noise. Both algorithms were implemented with Python, utilizing Keras and TensorFlow libraries. Keras and TensorFlow are

open- source libraries that provide APIs to create artificial neural networks. Keras was configured to utilize multi-GPU for faster learning and handling larger batch sizes during the training. Since sparse data and adaptive learning rate is used, Adam optimizer was utilized for the model. Huber Loss function was used as the loss function for the models.

3.2.1. Deep Q-Learning

Deep Q-learning uses an off-policy method to separate learning and acting policies. In this research, Q-learning was implemented with Convolution Neural Network (CNN). Q-learning employs the CNN to approximate the value function for state. This implementation of Q-learning was first developed by Google's Deep Mind for playing the 2600 Atari games. There are two steps in Q-learning where it approximates the value using the Q technique and then uses the experience replay method to stabilize the neural network. This is because the neural network is highly unstable when approximating the values [21]. Figure 6 shows the architectural diagram of the implemented Q-learning model.

Q-Value Function:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[R + \gamma \max_{a'} Q(s', a') - Q(s, a)] \quad (3.1)$$

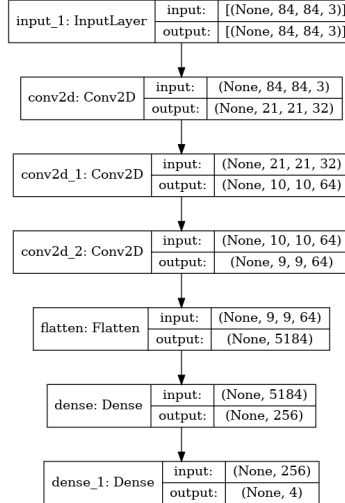


Figure 6. The model architecture of Q-learning methodology

For each step the learning algorithm takes, it receives observations from the virtual environment. These images are preprocessed, and random camera noise is added to increase the entropy. Images are stacked and downscaled to 80x80 pixels before input into the model. The network is able to learn an approximation of the Q-table which contains mapping of actions and corresponding states. In this case the network performs two actions: pitch and roll. When pitching the platform, it angled about the x-axis, when rolling, the platform angled in y-axis. Action pitch_left (-1), pitch_right (+1), roll_backward (-1) and roll_forward (+1) were executed against the virtual environment to make observations and replays. For each episode reward was calculated based on the location of the marble relative to the center of the platform. The highest reward of +1 will be offered if the marble is in the center region (refer Figure 5) and +0.5 will be offered if within the blue region else the environment will be reset for the next episode. Initially, the algorithm is set to perform random actions against the virtual environment and observe the states. Thereafter the agent

selected the best action based on the known Q-value. The algorithm terminates when it reaches the optimum exit condition.

In the physical system algorithm utilizes the camera module and OpenCV library to capture and pre-process the images required to make a prediction. Based on the action proposed by algorithm, the physical system performs a pitch or roll on the platform. These roll and pitch commands are translated into servo motion through inverse kinematics. Three servos are driven based on the proposed pitch and roll values.

3.2.2. Actor-Critic Architecture

Actor critic method uses two separate models namely actor and critic. The actor (policy) proposes a set of actions for a given state, and the critic determines and evaluates the actions proposed by the actor [22]. Actors and critics learn to perform until it maximizes the reward for the virtual environment.

The implementation of Actor-critic in this research follows a similar structure (Figure 4) as the Q-learning. The algorithm continued until the reward reaches a specific milestone or until the algorithm reaches 5000 episodes. For each step the algorithm receives the current position of the marble, the velocity vector and the location of the marble relative to the center. Similar to the Q-learning algorithm, it performs two actions, pitch and roll. Action pitch_left(-1), pitch_right(1), roll_backward(-1) and roll_forward(1) are performed in the virtual environment. Velocity vector, marble position and information on which zone it is located are passed as input to the actor-critic model. Velocity vector is obtained through the game engine, and it allows to determine the direction and velocity at which the marble travels. Based on the location of the marble, the actor-critic learns to predict the future motion required to center the marble. Figure 7 shows the model architecture of Actor-Critic where the left branch shows the actor while right branch shows the critic.

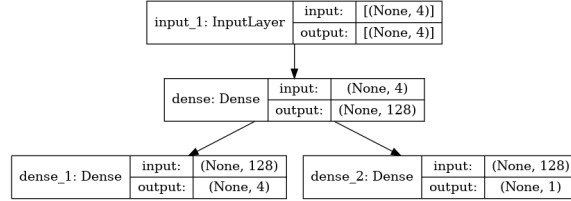


Figure 7. The model architecture of Actor-Critic methodology. Left branch denotes the Actor while right branch denotes the Critic

Similar to Q-learning, on the physical system, algorithm utilities the camera to obtain the marble position relative to the center of platform, current marble location and velocity vector approximation. Pitch and roll actions proposed by actor-critic are translated to individual servo axis movement through inverse kinematic calculations.

3.3. Domain Randomization and Inducing Noise

Domain randomization and noise was added to the virtual environment during the training process. The main hypothesis behind adding domain randomization and noise during the training is to maximize the diversity of the virtual environment. Researchers have suggested that the domain randomization method drastically improves the quality of emergent meta learning. Figure 8 shows the domain randomization process applied across the virtual scene to change the textures of the scene. In each new episode, the environment is randomized. The method directly affects Q-learning as it utilizes the raw pixel data for training.

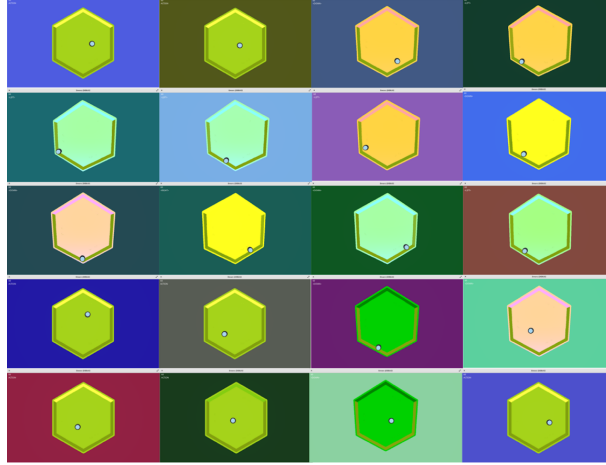


Figure 8. Domain randomization to randomize the environment textures and lighting

Additionally, randomized noise is introduced into the environment to induce variability in the physics simulation. Mass, friction, and simulation framerate were modified to a certain extent to introduce the variability. Camera field-of-view and position of the camera relative to the platform was also changed during the training process. The number of lights in the scene, orientation, focus, specular characteristics and brightness were randomized. In addition, the position of the marble relative to the platform, its radius and weight were also randomized. This small randomization allows the model to be generalized when transferred into the physical environment and has the ability to adapt to small environment, physical or mechanical changes.

4. Experimental Setup

Both Actor-Critic and Q-learning algorithms were trained on the same virtual environment with exact same parameters and environment variables. For Q-learning the learning rate was set to 0.00025 and Actor-Critic was set to 0.01 respectively. Both algorithms were set to be trained for 600 episodes. In addition, both algorithms were set to terminate when they reached a reward level of 10,000. Each model was set to save if the running reward was higher than previously recorded reward. Tensorboard was utilized to visualize the training phase, observation phase and replaying phase. The reinforcement learning process was conducted on a local system that has a Nvidia RTX 2080 GPU, RAM of 64 GB and a AMD Ryzen 5800X CPU with 3.8 MHz.

Once the initial training was conducted the trained models were transferred over to the physical robot. The models were hosted on the Jetson Nano, which interfaced with the 3-Axis Steward Platform. The camera feedback was utilized as the only source of input for the algorithm. Additional pre-processing was conducted to downscale the images to 80x80 pixels, compute the velocity vector, track the position of the marble relative to the platform. Based on input image, a roll or pitch value will be output from the model. Using Inverse kinematic each servo location for each axis will be obtained. The goal of the physical environment was to evaluate the performance of the RL model on the real-world. The monocular camera position was kept static while setting the field of view match virtual environment (similar to view on Figure 5). The lighting was kept at best at a constant. Initial position of the platform was set to 45 degrees keeping the platform leveled. A delay of 30s was added until the marble was placed on the platform to begin the prediction.

5. Results and Discussion

Multiple training sessions were conducted while optimizing the hyper-parameters of each Actor-Critic and Q-learning methodologies. On Q-learning the number frames to perform random actions against the virtual environment before observing the output was increased from 5,000 to 100,000 to optimize the learning policies. Max-memory length was decreased from 300,000 frames to 100,000 to reduce the max-memory length during the training. Different batch sizes were tested (from 32, 64, 128 and 256). The optimum batch size for the resources available was 64. Increased batch size caused memory overflow and over utilization of GPU causing the virtual environments frame-rate to drop. This directly affected the performance of the model.

Figure 9 shows the performance of Actor-Critic and Q-learning algorithm for 600 episodes. Actor-Critic algorithm took 32 hours to reach 600 episodes and by far yield the best performance. It was able to obtain a maximum reward of 9665 at 589th episode. Q-learning on the other hand took 29 hours to complete 600 episodes. This algorithm was only able to reach a max reward of 583 and overall maintained an average of 466.

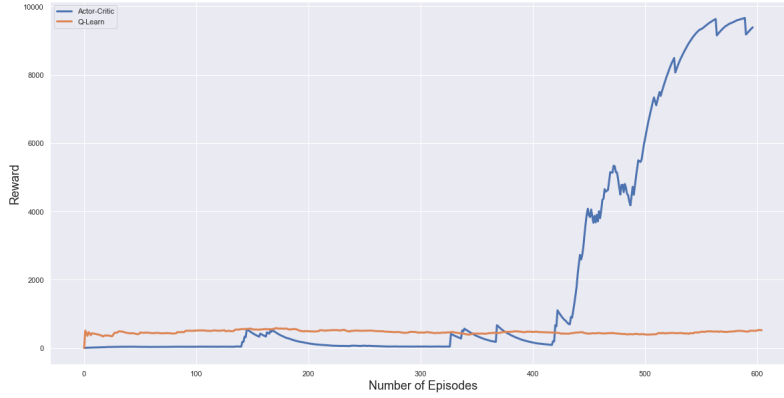


Figure 9. Accumulated reward per episode on virtual environment (Blue graph: Actor-Critic, Orange graph: Q-learning)

Figure 10 shows the number frames the model was able to keep the marble centered in the platform. Q-learning was consistent in keeping with the marble towards the blue zone from the very beginning. However, the model was not able to keep the marble centered to receive the highest reward. Actor-Critic was not able to perform well until the episode 450 at which it was able to accurately keep the marble centered.

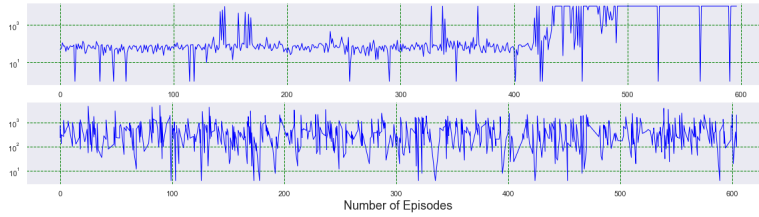


Figure 10. Episode length of time (number of frames the marble was balanced in the center) Top graph: Actor-Critic; Bottom graph: Q-Learning

To evaluate the performance of the physical environment and evaluate each algorithm's performance, each model was tested for 20 trials. Each trial was considered as one episode

and was timed to observe how much time the marble was centered on the platform. On an average, Actor-Critic algorithm was able to center the marble for 74s over a 2-minute episode. On the other-hand Q-learning model on average was able to center marble for 34s. Since Q-learning depended on raw scene as an input to the model, the inconsistency of real-world lighting, dynamics and inadequate initial reinforcement learning caused the model to perform poorly.

6. Conclusion and Future Work

This research implemented two reinforcement learning methodologies to investigate and to evaluate factors that affect the quality of virtual to real-world transfer in order to find workflows to reduce the reality-gap. In this implementation, Q-learning utilizes raw image data for making a prediction while actor-critic based its prediction on pre-processed numerical information. Since actor-critic relied on the position of the marble and velocity vector, it was able to create policies that can predict the next movement of the marble and make a prediction. On the other-hand, Q-learning struggled with environment changes such as inconsistent real-world lighting, system dynamics and the closeness of virtual environment to real-world.

Future work includes improving the accuracy of the Q-learning methodology by improving the virtual environment, evaluating the parameters that optimize the learning, and factors that affect the negative learning. Additionally, to improve the quality of learning the researchers can employ multi-agent learning environments to increase the variability and speedup the learning. This allows to tune hyper-parameters, environment variables and to observe learning accuracy. Researcher will employ cloud solutions to overcome the memory overflow and GPU utilization problems for future learning. In addition, perform experiments with additional textures, lighting, rendering techniques and utilization of domain adaptation technique will be used to improve the reinforcement learning policies.

References

- [1] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. “Deep Reinforcement Learning that Matters”. In: *arXiv:1709.06560 [cs, stat]* (Jan. 2019). arXiv: 1709.06560. URL: <http://arxiv.org/abs/1709.06560>.
- [2] OpenAI et al. “Solving Rubik’s Cube with a Robot Hand”. In: *arXiv:1910.07113 [cs, stat]* (Oct. 2019). arXiv: 1910.07113. URL: <http://arxiv.org/abs/1910.07113>.
- [3] J. Collins, R. Brown, J. Leitner, and D. Howard. “Traversing the Reality Gap via Simulator Tuning”. In: *arXiv:2003.01369 [cs]* (Mar. 2020). arXiv: 2003.01369. URL: <http://arxiv.org/abs/2003.01369>.
- [4] J. Tebbe, L. Krauch, Y. Gao, and A. Zell. “Sample-efficient Reinforcement Learning in Robotic Table Tennis”. In: *arXiv:2011.03275 [cs]* (Mar. 2021). arXiv: 2011.03275. URL: <http://arxiv.org/abs/2011.03275>.
- [5] H. Lipson, J. Bongard, V. Zykov, and E. Malone. “Evolutionary Robotics for Legged Machines: From Simulation to Physical Reality.” In: Jan. 2006, pp. 11–18.
- [6] N. Hafez, V. Dietrich, and S. Roehrl. “Analysis of Different Methods to Close the Reality Gap for Instance Segmentation in a Flexible Assembly Cell”. In: June 2020, pp. 505–515. ISBN: 978-3-030-48988-5. DOI: [10.1007/978-3-030-48989-2_54](https://doi.org/10.1007/978-3-030-48989-2_54).
- [7] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. “Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World”. In: *arXiv:1703.06907 [cs]* (Mar. 2017). arXiv: 1703.06907. URL: <http://arxiv.org/abs/1703.06907>.
- [8] S. Park, J. Kim, and H. J. Kim. “Zero-Shot Transfer Learning of a Throwing Task via Domain Randomization”. In: *2020 20th International Conference on Control, Automation and Systems (ICCAS)*. ISSN: 2642-3901. Oct. 2020, pp. 1026–1030. DOI: [10.23919/ICCAS50221.2020.9268312](https://doi.org/10.23919/ICCAS50221.2020.9268312).

- [9] M. Wang and W. Deng. “Deep Visual Domain Adaptation: A Survey”. In: *Neurocomputing* 312 (Oct. 2018). arXiv: 1802.03601, pp. 135–153. ISSN: 09252312. DOI: [10.1016/j.neucom.2018.05.083](https://doi.org/10.1016/j.neucom.2018.05.083). URL: <http://arxiv.org/abs/1802.03601>.
- [10] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, and V. Vanhoucke. “Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping”. In: *arXiv:1709.07857 [cs]* (Sept. 2017). arXiv: 1709.07857. URL: <http://arxiv.org/abs/1709.07857>.
- [11] M. Yan, I. Frosio, S. Tyree, and J. Kautz. “Sim-to-Real Transfer of Accurate Grasping with Eye-In-Hand Observations and Continuous Control”. In: *arXiv:1712.03303 [cs]* (Dec. 2017). arXiv: 1712.03303. URL: <http://arxiv.org/abs/1712.03303>.
- [12] A. Hundt, B. Killeen, N. Greene, H. Wu, H. Kwon, C. Paxton, and G. D. Hager. ““Good Robot!”: Efficient Reinforcement Learning for Multi-Step Visual Tasks with Sim to Real Transfer”. In: *IEEE Robotics and Automation Letters* 5.4 (Oct. 2020). arXiv: 1909.11730, pp. 6724–6731. ISSN: 2377-3766, 2377-3774. DOI: [10.1109/LRA.2020.3015448](https://doi.org/10.1109/LRA.2020.3015448). URL: <http://arxiv.org/abs/1909.11730>.
- [13] A. Gupta, C. Devin, Y. Liu, P. Abbeel, and S. Levine. “Learning Invariant Feature Spaces to Transfer Skills with Reinforcement Learning”. In: *arXiv:1703.02949 [cs]* (Mar. 2017). arXiv: 1703.02949. URL: <http://arxiv.org/abs/1703.02949>.
- [14] A. Kar, A. Prakash, M.-Y. Liu, E. Cameracci, J. Yuan, M. Rusiniak, D. Acuna, A. Torralba, and S. Fidler. “Meta-Sim: Learning to Generate Synthetic Datasets”. In: *arXiv:1904.11621 [cs]* (Apr. 2019). arXiv: 1904.11621. URL: <http://arxiv.org/abs/1904.11621>.
- [15] S. Gu, E. Holly, T. Lillicrap, and S. Levine. “Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates”. In: May 2017, pp. 3389–3396. DOI: [10.1109/ICRA.2017.7989385](https://doi.org/10.1109/ICRA.2017.7989385).
- [16] L. Xie, H. Li, and N. Thieme. *From Simulation to Real-World Robotic Mobile Fulfillment Systems*. Oct. 2018.
- [17] M. Bakhmudov and M. Fridheim. “Combining Reinforcement Learning and Unreal Engine’s AI-tools to Create Intelligent Bots”. en. In: (2020). Accepted: 2020-08-16T16:01:55Z Publisher: NTNU. URL: <https://ntnuopen.ntnu.no/ntnu-xmlui/handle/11250/2672159> (visited on 03/29/2022).
- [18] R. Antonova, S. Cruciani, C. Smith, and D. Kragic. “Reinforcement Learning for Pivoting Task”. In: *arXiv:1703.00472 [cs]* (Mar. 2017). arXiv: 1703.00472. URL: <http://arxiv.org/abs/1703.00472>.
- [19] T. Erez, Y. Tassa, and E. Todorov. “Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX”. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. ISSN: 1050-4729. May 2015, pp. 4397–4404. DOI: [10.1109/ICRA.2015.7139807](https://doi.org/10.1109/ICRA.2015.7139807).
- [20] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. “OpenAI Gym”. In: *arXiv:1606.01540 [cs]* (June 2016). arXiv: 1606.01540. URL: <http://arxiv.org/abs/1606.01540>.
- [21] B. Jang, M. Kim, G. Harerimana, and J. W. Kim. “Q-Learning Algorithms: A Comprehensive Classification and Applications”. In: *IEEE Access* 7 (2019). Conference Name: IEEE Access, pp. 133653–133667. ISSN: 2169-3536. DOI: [10.1109/ACCESS.2019.2941229](https://doi.org/10.1109/ACCESS.2019.2941229).
- [22] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska. “A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (Nov. 2012). Conference Name: IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), pp. 1291–1307. ISSN: 1558-2442. DOI: [10.1109/TSMCC.2012.2218595](https://doi.org/10.1109/TSMCC.2012.2218595).