KINOVA ROBOTIC ARM MANIPULATION WITH PYTHON PROGRAMMING

by

Cameron Veit

A Thesis Submitted to the Faculty of the

College of Engineering and Computer Science

In Partial Fulfillment of the Requirements for the Degree of

Master of Science

Florida Atlantic University,

Boca Raton, Florida

August 2022

KINOVA ROBOTIC ARM MANIPULATION WITH PYTHON PROGRAMMING

by

Cameron Veit

This thesis was prepared under the direction of the candidate's thesis advisor, Dr. Xiangnan Zhong, Department of Electrical Engineering and Computer Science, and has been approved by all members of the supervisory committee. It was submitted to the faculty of the College of Engineering and Computer Science and was accepted in partial fulfillment of the requirements for the degree of Master of Science.

DEFENSE COMMITTEE

Xiangnan Zhong
Xiangnan Zhong (Jul 28, 2022 14:44 EDT)

Xiangnan Zhong, Ph.D.
Thesis Advisor

Zhen Ni
Zhen Ni (Jul 28, 2022 15:06 EDT)

Zhen Ni, Ph.D.

Erik Engeberg, Ph.D.

Hanqi Zhuang, Ph.D.
Chair, Department of Electrical
Engineering and Computer Science

Stella Batalama, Ph.D.
Dean, College of Engineering
and Computer Science

Robert W. Stackman Jr., Ph.D.
Dean, Graduate College

August 1, 2022
Date

ACKNOWLEDGEMENTS

I wish to thank my thesis advisors for all of their guidance, expertise, and support throughout the journey of creating this thesis. Additionally, I give thanks to my committee members for their support. Finally, I wish to thank my fellow graduate students and family, I could not have completed this thesis without you.

Throughout the process of completing this thesis, the support of those around me alongside the various methods used to complete the thesis has expanded my knowledge. I have learned more about the use of Python in connecting a computer to external devices as well as the use of Python in connecting to and control a robotic arm. Additionally, the research done in this thesis has introduced me to the world of computer simulations through the use of the Unity compiler in creating simulations of a robotic arm.

ABSTRACT

Author: Cameron Veit

Title: KINOVA ROBOTIC ARM MANIPULATION WITH PYTHON PROGRAMMING

Institution: Florida Atlantic University

Thesis Advisors: Dr. Xiangnan Zhong

Degree: Master of Science

Year: 2022

As artificial intelligence (AI), such as reinforcement learning (RL), has continued to grow, the introduction of AI for use in robotic arms in order to have them autonomously complete tasks has become an increasingly popular topic. Robotic arms have recently had a drastic spike in innovation, with new robotic arms being developed for a variety of tasks both menial and complicated. One robotic arm recently developed for everyday use in close proximity to the user is the Kinova Gen 3 Lite, but limited formal research has been conducted about controlling this robotic arm both with an AI and in general. Therefore, this thesis covers the implementation of Python programs in controlling the robotic arm physically as well as the use of a simulation to train an RL based AI compatible with the Kinova Gen 3 Lite. Additionally, the purpose of this research is to identify and solve the difficulties in the physical instance and the simulation

as well as the impact of the learning parameters on the robotic arm AI. Similarly, the issues in connecting two Kinova Gen 3 Lites to one computer at once are also examined.

This thesis goes into detail about the goal of the Python programs created to move the physical robotic arm as well as the overall setup and goal of the robotic arm simulation for the RL method. In particular, the Python programs for the physical robotic arm pick up the object and place it at a different location, identifying a method to prevent the gripper from crushing an object without a tactile sensor in the process. The thesis also covers the effect of various learning parameters on the accuracy and steps to goal curves of an RL method designed to make a Kinova Gen 3 Lite grab an object in a simulation. In particular, a neural network implementation of RL method with one of the learning parameters changed in comparison to the optimal learning parameters. The neural network is trained using Python Anaconda to control a Kinova Gen 3 Lite robotic arm model for a simulation made in the Unity compiler.

KINOVA ROBOTIC ARM MANIPULATION WITH PYTHON PROGRAMMING:

LIST OF FIGURES

I: INTRODUCTION

With the explosive growth of the field of artificial intelligence (AI) in the past few years, an increasing amount of research has been conducted for the use of AI in robotic arm manipulation. Be it a simulation or real-word application, the use of neural networks and AI in general with robotic arms has been studied, with a multitude of new algorithms being developed specifically for this purpose. These new AI controlled robotic arms can be used for a variety of assignments, such as industrial or everyday tasks. One particular field of interest in the research of AI controlled robotic arms is the use of deep reinforcement learning (RL) in training robotic arm agents. These agents learn policies based on the environmental feedback from their chosen actions. As such, the agent learns the optimal series decisions it needs to take in order to complete its task in a minimal number of movements through a trial and error process (Liu 2020) (Osiński 2018). Due to the agent learning from the environment rather than a particular dataset, an AI trained with reinforcement learning (RL) has a much larger pool of data to use to identify behavior patterns that help it reach the goal in a minimal number of steps (Sutton 2018).

The use of deep reinforcement learning has been recorded for many robotic arms. Deep reinforcement learning is a popular learning method for intelligent control of robotic arms. The reason for this is that robotic arms trained with RL learn through trial and error. This allows a reinforcement learning-based robotic arm to encounter situations that cannot be found using other learning methods such as supervised or unsupervised

learning, teaching the robotic arm agents more complex behaviors, and making the RL based AI more adaptable than intelligent robotic arms trained with other learning methods, such as Supervised or Unsupervised learning (Amarjyoti 2017). On top of the higher adaptability and range of behaviors that an RL based agent can learn, AIs trained with reinforcement learning do not need a dataset in order to be properly trained (Liu 2021). As such, agents trained with RL are much more compatible with simulated environments, which can iterate a multitude of times in order for the AI to learn the proper behavior required to complete its designated task. On top of basic robotic arm manipulation using an AI, precise robotic arm manipulation has also been attempted in recent studies, reducing the flexibility of the agent in exchange for better performance on a specific task (Popov 2017). Finally, studies have created methods to train a RL based AI in the real world in a reasonable amount of time (Gu 2017).

Despite recent developments in AI based robotic arms, the use of AI has not been explored for every single robotic arm. In particular, the robotic arm covered in this thesis, the Kinova Gen 3 Lite, has had no studies detailing the use of artificial intelligence in controlling it. The Kinova Gen 3 Lite is a 6 DoS robotic arm developed with the purpose of being used in close proximity to the user, with a focus on safety. Furthermore, an easy to use guide as well as Python manipulation demos has been developed for the Kinova Gen 3 Lite. While detailed guides about the Kinova Gen 3 Lite and its manipulation do exist, there have been a few research studies about the control of the Kinova Gen 3 Lite overall. As such, this research endeavors to address the control of the Kinova Gen 3 Lite using Python; in particular the use of a Python based deep reinforcement learning (RL) for controlling the robotic arm.

In this thesis, the feasibility of using Python to control a Kinova Gen 3 Lite is explored using multiple methods. First, after identifying existing research on dual robotic arm AIs for inspiration, the Kinova Gen 3 Lite is connected to the computer and manipulated in the real world. By controlling the robotic arm in the real world, this paper identifies that Python can be used to control the Kinova Gen 3 Lite in general, and that Python can be used to conceivably control the robotic arm with an AI. Thus, a simulation of the Kinova Gen 3 Lite is created to grab an object intelligently based on reinforcement learning algorithms. In particular, due to a lack of prior artificial intelligence studies for the Kinova Gen 3 Lite, the impact of changing different learning parameters on the accuracy and step to go curve of the agent are evaluated. Thereby, the results of the various agents trained using the simulation clarify the importance of the various learning parameters in training an AI algorithm for the Kinova Gen 3 Lite.

Through the identification of the various methods by which Python can be used to control the Kinova Gen 3 Lite, the overall automatic manipulation of the aforementioned robotic arm can be better understood and implemented. This is especially important due to the Kinova Gen 3 Lite's existence as a robotic arm built for cooperation with humans, meaning that the automation of the robotic arm can have a large impact on the range of possible users. In particular, automatic control methods for the Kinova Gen 3 Lite would have great use in the daily lives of those who are unable to control the robotic arm directly. Additionally, while the creation of an AI for the Kinova Gen 3 Lite will not greatly advance the field of autonomous systems and AI as a whole, such a development can possibly be used to advance the study of human-interaction based autonomous

systems and AI, be it in conjunction with other robots that interact with humans or in a standalone fashion.

In the remainder of the thesis, previous work related to dual robotic arm based AI is reviewed first. Next, in Section III, the methods of the successful experiments, the physical manipulation of the Kinova Gen 3 Lite and Unity simulation of the Kinova Gen 3 Lite, are explored in detail. The thesis continues with the results of the successful experiments in Section IV, starting with a brief discussion of physical manipulation of the robotic arm and continuing with the exploration of the impact of different learning parameters on the AI for the simulation. Then, in Section V, there is a discussion about the extra experiments that act as an intro to physical manipulation of the Kinova Gen 3 Lite using Python, an outlier learning parameter for the simulation, the learning rate parameter, and the difficulties in controlling two robotic arms at once both physically with Python and within a Unity simulation. Lastly, Section VI concludes the thesis with the main ideas discovered during this research, with a larger focus on the impact of changing different learning parameters for the simulation AI.

II: LITERATURE REVIEW

The Kinova Gen-3 Lite robotic arm is designed to provide robotic assistance to increase accessibility for individuals with limited mobility. In order to identify the optimal method to control the Kinova robotic arm, articles covering the existing applications of the Kinova robotic arm and the use of AI to control the device were reviewed.

Foundational Knowledge of the Kinova Gen3 Lite

The Kinova robotic arm has two main methods of control, one which uses a high level API using either angular or cartesian control methods from the Kinova controller library. The other, the low level API allows direct manipulation of the robotic arm by accessing individual actuators, permitting the robotic arm more flexibility and faster feedback (Campeau-Lecours, 2019). Campeau-Lecours et all outlined various methods to control the robotic arm that do not use python, these include the web app, accessed on the web browser via a link located in the user guide, the joystick, and ROS libraries. However, for the purposes of demonstrations and methods tested, controlling the robotic arm with the high level API is used due to its compatibility with Python, unlike the other methods of control. Additionally, while the article focuses on JACO and MICO, the Kinova Gen3 Lite also uses the above control methods.

Applications of the Kinova Robotic Arm

Research for the Kinova Gen3 Lite is limited. Zohour et all discusses the inverse kinematics of the Kinova Gen3 Lite (Zohour 2021). While the research is helpful for identifying which positions correspond to which joint angles, the Kinova Gen3 Lite can be manipulated with less effort by using the high level API. Similarly, Tagliavini et all describes platform access for optimal method of movement. Tagliavini et all does not address the actual movement of the Kinova Gen3 Lite itself, rather, the degree by which the base of the Kinova Gen3 Lite is changed in response to different tasks covered (Tagliavini 2021). While the Kinova Gen3 Lite does not yet have many applications, Campeau-Lecours et all reviews various existing commercial products that integrate JACO and MICO Kinova robotic arms alongside other robots. These commercial products include a multitude of products from Clearpath Robotics Inc., the Pioneer Manipulator created by Adept Technology, the Quanser Robotics Platform, the BioTac system made by SynTouch, the Stanley Robotics ARTI platform, and the Robotnik XL-MICO (Campeau-Lecours, 2019). However, it should be noted these products use the MICO and JACO Kinova robotic arms rather than the Kinova Gen3 Lite. Similarly, Campeau-Lecours et all also names various research studies on the JACO and MICO Kinova Robotic arms, but like the commercial products, do not use the Kinova Gen3 Lite (Campeau-Lecours, 2019).

Dual Robotic Arm Artificial Intelligence Research

Various studies outline the use of two robotic arms working in unison. Additionally, most of the articles emphasize the importance of properly denoting the environment while training a robotic arm. Arntz et all discusses training of an AI for a

dual-arm robot using Virtual Reality with the importance of placing penalties for collisions against the other arm, the human, and the environment (Arntz 2021). However, this research does not outline either the ability to train an AI for a dual-arm robot to act semi-autonomously or the translation on how effective the trained AI for the dual-arm robot functions in reality.

Aljalbout reviews a learning method for two arm robots called “dual-arm adversarial robot learning (DAARL)” with the goal of creating a training method that could replace simulations (Aljalbout 2022). DAARL was designed to train a dual-arm robot without human interaction, via having the arms swap roles of trainer and trainee in order to develop two AIs that control each robotic arm (Aljalbout 2022). Despite the article’s solid reasoning and explanations, Aljalbout neither identifies a functional example of the DAARL algorithm or the possible applications of DAARL in conjunction with simulations. Similar to the prior study, Wu et all describes the use of reinforcement learning for a 7 DOF dual-arm space robot by which the experiment and learning algorithm were designed, covering most aspects encountered by a dual-arm space robot (Wu 2020). Despite covering the majority of possible environment variables, Wu et all does not identify the 7 DOF dual-arm robot’s ability to deal with either environmental obstructions or non-target objects in the environment.

Reinforcement learning has been proven successful on dual-arm robots. Dual-Arm Deep Deterministic Policy Gradient Algorithm (DADDPG)” and “Hindsight Experience Replay” have been utilized to train a dual-arm robot to complete simple tasks (Liu 2021). Additionally, the dual-arm robot is trained for collaboration between the two robotic arms. However, the research does not address the ability of the dual-arm robot AI

to operate in a changing environment or with multiple objects. The centralized dual arm robot researched by Alles et all also uses reinforcement learning (Alles 2021). Research comparing three different types of policies to control a centralized dual-arm robot to place a peg in a hole was attempted. While the method used by Alles et all to train centralized dual arm robot is useful for training the robotic arm to complete the desired task, the arm was limited to one skill.

Deep learning has also proven effective to train a dual-arm robot AI to tie a rope (Suzuki 2021). In this application, two deep neural networks are used, "Convolutional Auto-Encoder (CAE)" with extracts significant image features and "Long short-term memory (LSTM)" which was used to train the motions of the dual-arm robot (Suzuki 2021). While Suzuki et all does address the ability of the dual-arm robot AI to correctly tie a rope when the rope was a different color and therefore less identifiable, the ability of the dual-arm robot to tie differently shaped rope or rope made with different materials was not addressed.

Meanwhile, studies using optical sensors and hidden Markov models have been utilized to train various types of robotic arms by mimicking human movement (Ge 2013). Specialized sensory equipment, such as a glove, is required in order to properly train the robotic arms, eliminating the simulation process of training an AI. As such, since the Kinova Gen3 Lite does not have an inbuilt sensor and the desire to use a simulation when training the AI, Ge's method of training an AI using sensors is not suitable for the purpose of training the Kinova Gen3 Lite.

Goal of Thesis

Overall, the main goal of this thesis is to address the manipulation of the Kinova Gen3 Lite using Python. Specifically, the ability of a reinforcement learning AI for the Kinova Gen3 Lite, the ability to control two Kinova Gen3 Lite robotic arms at once using one computer, and the ability of an AI to control two Kinova Gen3 Lite robotic arms at once if two Kinova Gen3 Lite robotic arms can be controlled by the same computer are explored.

III: METHOD

Basic Manipulation of the Kinova Gen 3 Lite in Python

The basic compatibility and control of the Kinova Gen 3 Lite with the Python programming language was tested by using the Kinovarobotics kortex API located in github. In order to use the Kinovarobotics kortex API, some prior setup was required. This setup involves the Kinova Gen 3 Lite being plugged into the computer on which the python program is run using an ethernet cord, as can be seen in the figure below.

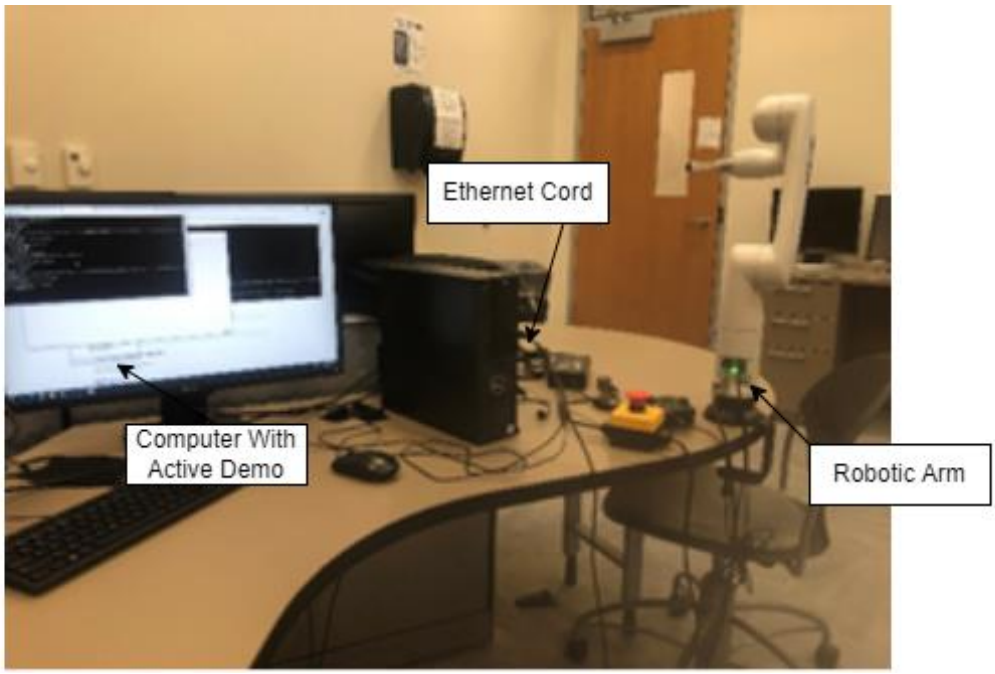

Figure 1 - Kinovarobotic Kortex demo setup, this is also the setup for the robotic arm in general.

In order to properly run the API programs, it is important to identify the properties and limits of the Kinova Gen 3 Lite. The Kinova Gen 3 Lite has six degrees of freedom and a reach of 760 millimeters, or 0.76 meters, from its base. Additionally, the

robotic arm can carry a weight of 0.5 kilograms and can move at around 25 centimeters per second. Lastly, the joint ranges of the Gen 3 Lite are also an important factor in the actual maneuverability of the robotic arm, with most of the joints having between ± 155 to 160 degrees of motion.

After properly connecting the Kinova Gen 3 Lite to the computer, the Kinovarobotics kortex API can then be tested with the demos and two programs that were created based off of the demos in the kortex python API. The created programs either move an object automatically given a set starting position, or move an object manually given the user input and a set starting position of the object. Both of the developed programs used cartesian movements instead of angular movements to control the robotic arm. Additionally, the set object position was at the edge of the Kinova Gen 3 Lite's reach, and the end position of the object varied based on whether robotic arm was controlled automatically or manually. Furthermore, it should be noted that the python programs both used a TCP connection as the method to control the robotic arm. After the programs were created, they were run with the goal of identifying if they had the capability to move the object to the end position designated by the program or the user without disrupting the shape of the targeted object. After each run of either program, the object was moved back to its original position, while the robotic arm was moved back to the home position, a neutral position that allows movement in all directions. Additionally, due to the fact that the Kinova Gen 3 Lite did not have inbuilt tactile sensors, another method of grabbing objects while not deforming them was developed. This method was developed after a multitude of iterations and uses the current in the gripper motor from the robotic arm's feedback to identify if an object has been grasped firmly without being

distorted. This position of the gripper is designated by a current in the gripper motor being above 0.75. It should be noted that the finger.value, which is the speed at which the gripper fingers move was set to 0.3, which was sufficiently fast to not be tediously long, but not too fast to crush an object before contact with the object was identified.

Kinova Gen 3 Lite Simulation in Unity

The Kinova Gen 3 Lite simulation in unity was highly inspired by the "How to train your Robot Arm" article on medium by Raju K (Raju 2020). This article served as the basis for both the neural network and the agent scripts used. Contrary to the article however, the used neural network has four hidden layers with 512 nodes each. Additionally, the network was trained in anaconda, where a ml-agents environment was created using "conda create – n ml-agents-1.0 python=3.7" and run using "conda activate ml-agents-1.0". Furthermore, the training process consisted of two and a half million iterations, each with a batch size of 512 and with the values being directly noted every 10000 iterations. Additionally, the memory size and length was set to 256 and the learning rate was changed to 3.0e-3. Once the training process was complete, it was then sent to TensorBoard by typing "tensorboard –logdir results" in the command line.

On the other hand, the scripts had quite a few differences from the scripts used to inspire them. The main script, called RobotControllerAgent used ActionBuffers to hold the change in degrees for each of the six axes. Similarly, the penalty existed between -1 and 1 depending on the distance from the object that the robotic arm is trying to grab. Furthermore, there are six collected observations from the robotic arm that act as inputs to the neural network. These observations include the current axis angles, the normalized change in position of the robotic arm, the normalized position of the item being grabbed,

the normalized position of the end effector, the normalized difference between the object being grabbed and the end effector, as well as the current step count of the robotic arm over 5000. It should be noted that some of the if functions regarding the training state were removed and some floats were changed to ActionBuffers in comparison to the scripts made by Raju K. Additionally, the step penalty was changed from -0.0001 to -0.01 and -0.05. Similarly, the collision penalty was greatly different than the one made by Raju K. Specifically, collisions with both the ground and other parts of the robotic arm had a -1 penalty. Lastly, the reward for reaching the object was set to +1.

Despite the inspiration for the scripts and neural network coming from Raju K., the Kinova Gen 3 Lite model was entirely self-generated. To create the model of the Kinova Gen 3 Lite, several steps must be followed:

Step 1: Create the ground object. The ground object is flat and has a box collider used to identify collisions with the robotic arm. If a collision occurs, the Penalty Trigger function is called, stopping the episode and applying a -1 penalty.

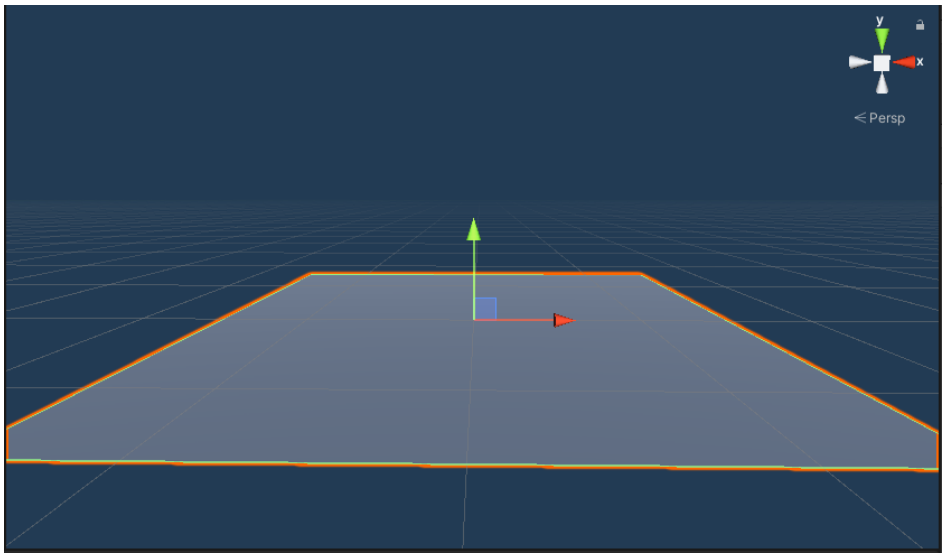

Figure 2 - Step 1 of creating the Kinova Robotic Arm model: create the ground.

Step 2: Create the component to be grabbed by the robotic arm. This component can have any shape that fits between the grippers on the robotic arm's end effector. Additionally, this component has a mesh and the "component" tag to identify if it collides with the grab area of the robotic arm. When this collision occurs, the robotic arm has reached the object and can grab it. As such, the Jackpot function is called, ending the episode and giving a +1 reward. The component also has the Penalty Trigger function to give a penalty if it collides with any part of the robotic arm other than the end effector.

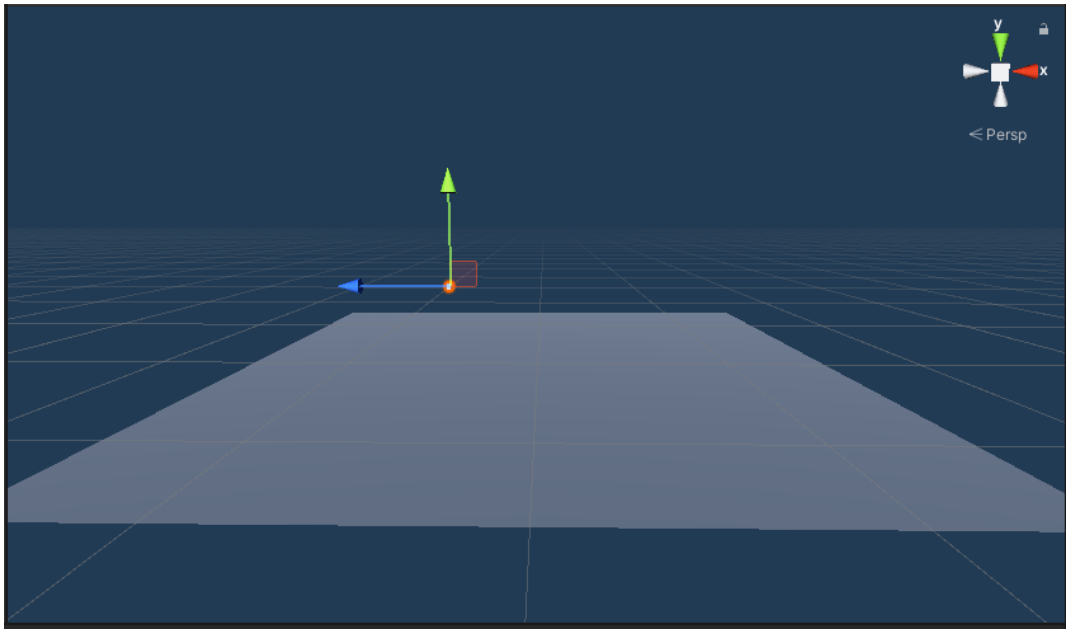

Figure 3 - Step 2 of creating the Kinova Robotic Arm model: create the component to be grabbed.

Step 3: Create the base of the robot arm. The base consists of a hemisphere and a cylinder on top of the hemisphere, both of which are colored black. The base is placed in the middle of the ground object and has no penalty collider because it is covered by the penalty collider of the first segment. The first axis is placed at the top of the cylinder and has ± 154.1 degrees of motion. Additionally, every other component except for a few on the gripper is colored white.

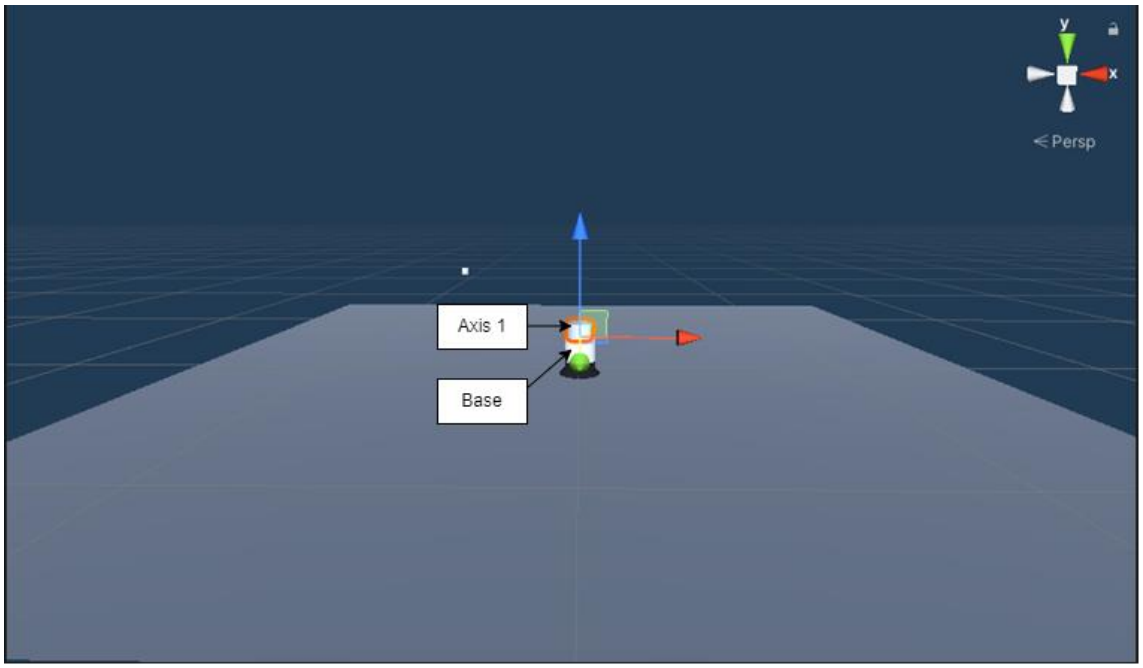

Figure 4 - Step 3 of creating the Kinova Robotic Arm model: create the base of the arm and axis 1. The base is the new white and black structure in the middle of the figure and axis 1 is the highlighted portion on top of the base.

Step 4: Create the first segment of the robotic arm. The first segment is connected to the first axis, making the segment rotate alongside the axis. The segment consists of a few differently shaped parts, starting with a cylinder and ending with another cylinder pointing to the right when aligned with the same world axes as the first axis. Connected to the end of the first segment is the second axis, which is rotated 90° along the world z-axis in relation to the first axis. The second axis has ± 150.1 degrees of motion and is placed slightly to the right in orientation to the first axis. Additionally, a capsule collider is used to cover the base and first segment of the robotic arm. The collider has the Penalty Trigger function activated in order to give a penalty of -1 if the later segments of the robotic arm collide with it and has the “Robot Internal” tag.

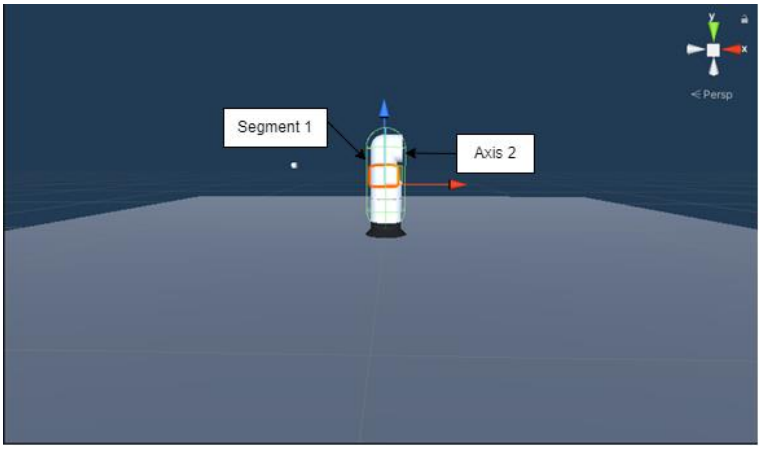

Figure 5 - Step 4 of creating the Kinova Robotic Arm model: create segment 1 of the arm and axis 2. Segment 1 is placed on top of the base, and axis 2 is facing right at the top of segment 1.

Step 5: Create the second segment of the robotic arm, the straight arm. The straight arm is a rectangle component connected to the second axis that points directly upwards and has the third axis connected at the other end. The third axis has ± 150.1 degrees of motion and the straight arm has a box collider for penalty collisions with either the rest of the robotic arm or the ground. A "Robot Internal" tag is used for penalties during ground collision, and a Penalty Trigger is used for penalties with other robotic arm segments. The third axis is aligned with the second axis, but is rotated 180 degrees along the world z-axis, making it point in the other direction.

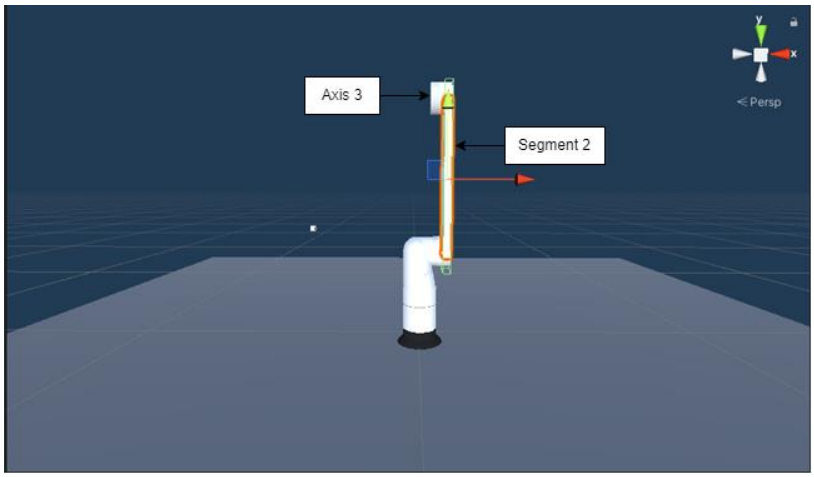

Figure 6 - Step 5 of creating the Kinova Robotic Arm model: create segment 2 of the arm and axis 3. Segment 2 is the long rectangle portion to the right of axis 2, and axis 3 is placed facing left at the top of segment 2.

Step 6: Create the third segment of the robotic arm. The third segment is shaped like the first segment except flipped, with the third axis connecting to the same point on the third segment as the second axis connects to on the first segment. At the other end of the third segment is the fourth axis that has ± 148.98 degrees of motion. Like the prior two segments, the third segment has a capsule collider for penalty collision with the ground and other parts of the robotic arm. While this section has the "Robot Internal" tag, the section does not have a Penalty Trigger, preventing the penalty from occurring twice during collisions with other parts of the robot.

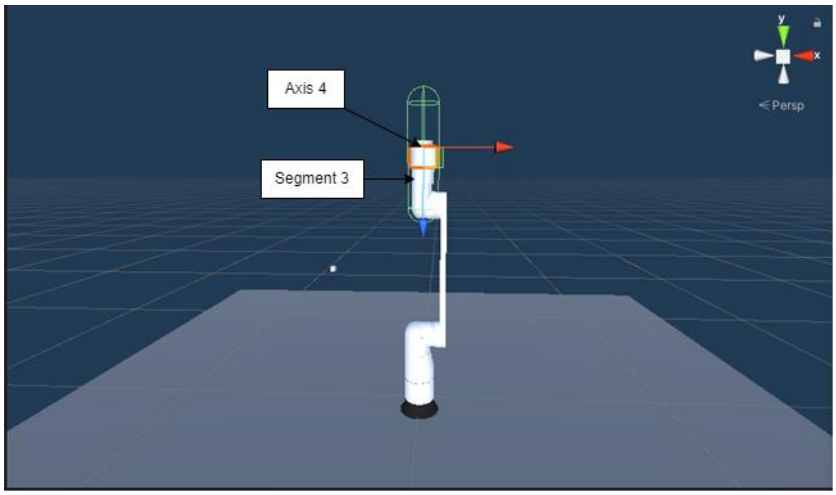

Figure 7 - Step 6 of creating the Kinova Robotic Arm model: create segment 3 of the arm and axis 4. Segment 3 is located to the left of axis 3, placing it in the middle of the figure, and axis 4 is placed at the top of segment 3.

Step 7: Create the fourth segment of the robotic arm. This segment is connected to the fourth axis and is created by copy pasting the third section and flipping it. Then, the fourth section is orientated so that the part of the arm connected to the fifth axis is facing away from the camera at the original orientation of the robotic arm. This segment does not have a collider because it is covered by the collider from the third segment of the model. At the end of this segment is the fifth axis, connected in a similar manner as the second axis on the first segment. The fifth axis has -144.97 to 145 degrees of motion.

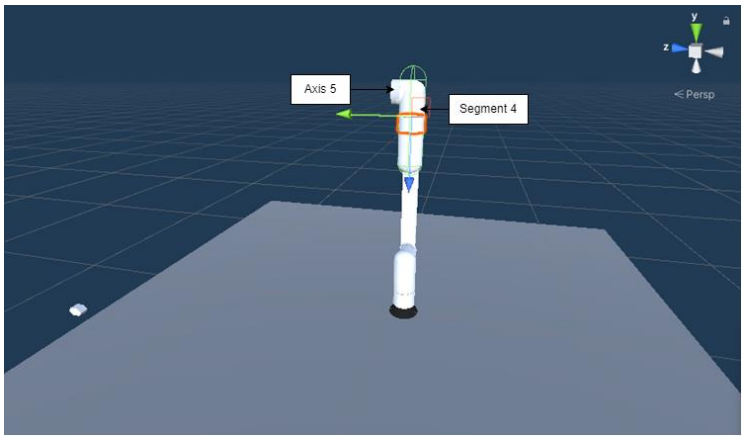

Figure 8 - Step 7 of creating the Kinova Robotic Arm model: create segment 4 of the arm and axis 5. Segment 4 is placed on top of axis 4, and axis 5 is facing to the left at the end of segment 4 in the image, away from the perspective of figure 7.

Step 8: Create the fifth segment of the robotic arm. The fifth segment is connected to the fifth axis and can be created by copy pasting the fourth segment and flipping, the part on the fourth segment connected to the fifth axis is connected to the fifth axis on the fifth segment. This segment also has a capsule collider and the "Robot Internal" tag. However, the fifth segment does not have the Penalty Trigger function enabled to prevent giving a double penalty, and because it cannot collide with the fourth segment. The sixth and final axis is connected at the end of the fifth segment, the highest part. Additionally, the sixth axis has ± 148.98 degrees of motion.

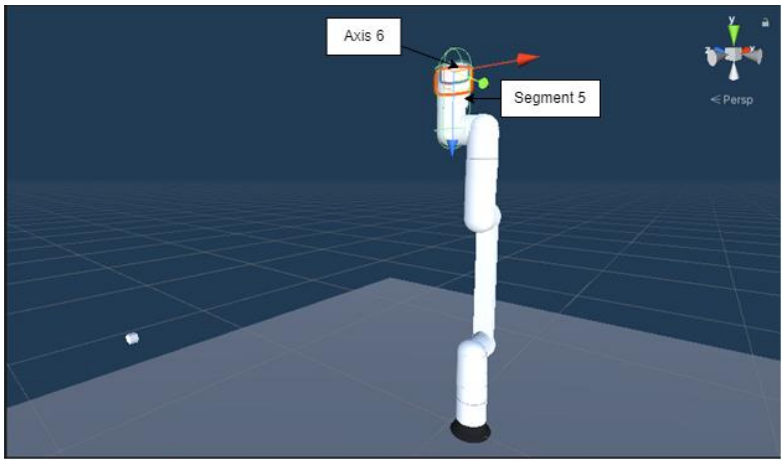

Figure 9 - Step 8 of creating the Kinova Robotic Arm model: create segment 5 of the arm and axis 6. Segment 5 is attached to axis 5, placing it at the top of the model, and axis 6 is placed on top of segment 5.

Step 9: Create the final part of the Kinova Gen3 Lite model, the end effector. The end effector is attached to axis 6 and consists of a short cylinder as well as a sphere. Connected to the sphere are two grippers consisting of two rectangle components each. Additionally, a rectangle component is placed partially within the sphere and connects the two grippers. Parts of the grippers and the middle rectangle connecting them are colored black to represent the pads on the end effector of the Kinova Gen3 Lite. The cylinder and sphere portion of the end effector is covered by the capsule collider from the fifth segment.

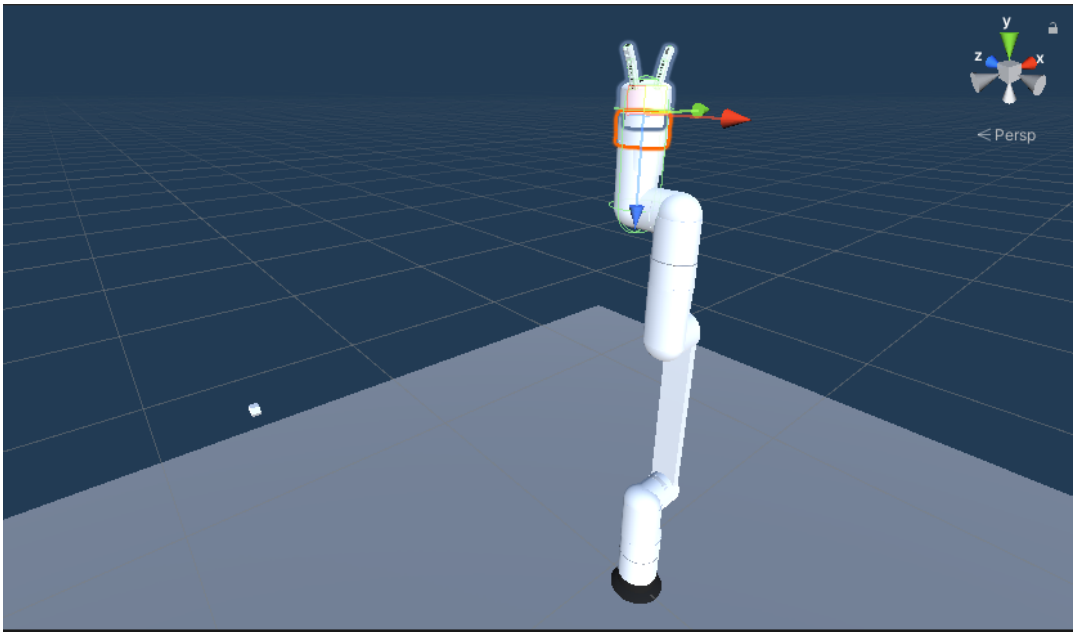

Figure 10 - Step 9 of creating the Kinova Robotic Arm model: create the End Effector.

Step 10: Create the "End Effector" Collider of the robot arm's model. The "End Effector" collider is a sphere collider that does not intersect the capsule collider from the prior segment and is placed between the two fingers on the grippers. This sphere collider has the "End Effector" tag and calls the EndEffector function, giving a +1 reward for reaching the object or a penalty for colliding with the ground.

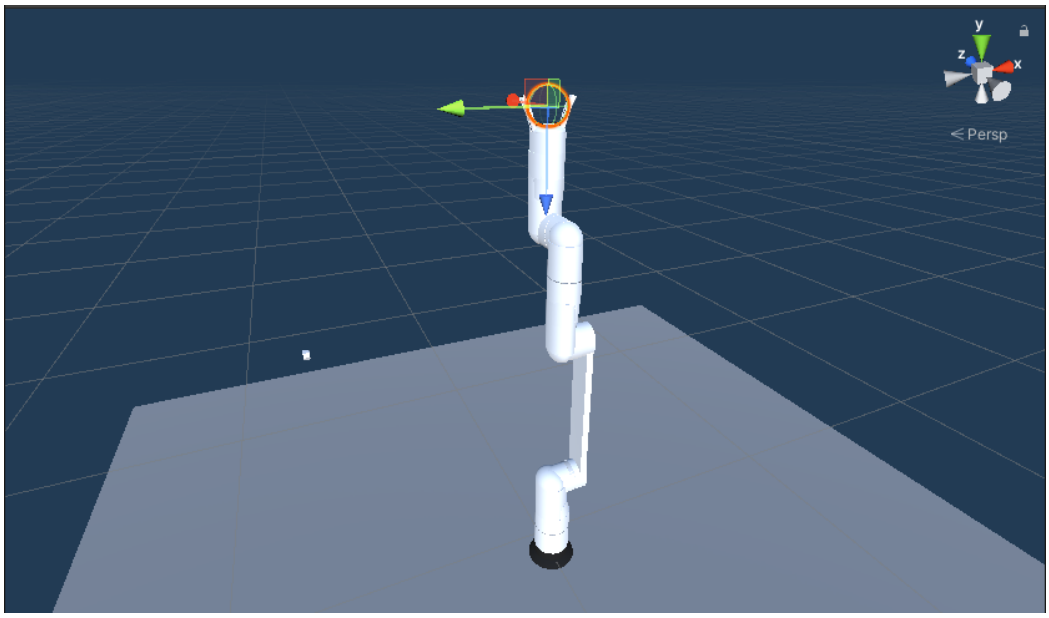

Figure 11 - Step 10 of creating the Kinova Robotic Arm model: create the "End Effector" collider.

Lastly, all of these parts are contained within a prefab called RobotWithColliders. The end result, as well as a flowchart summarizing the construction process, are visible in the following figures:

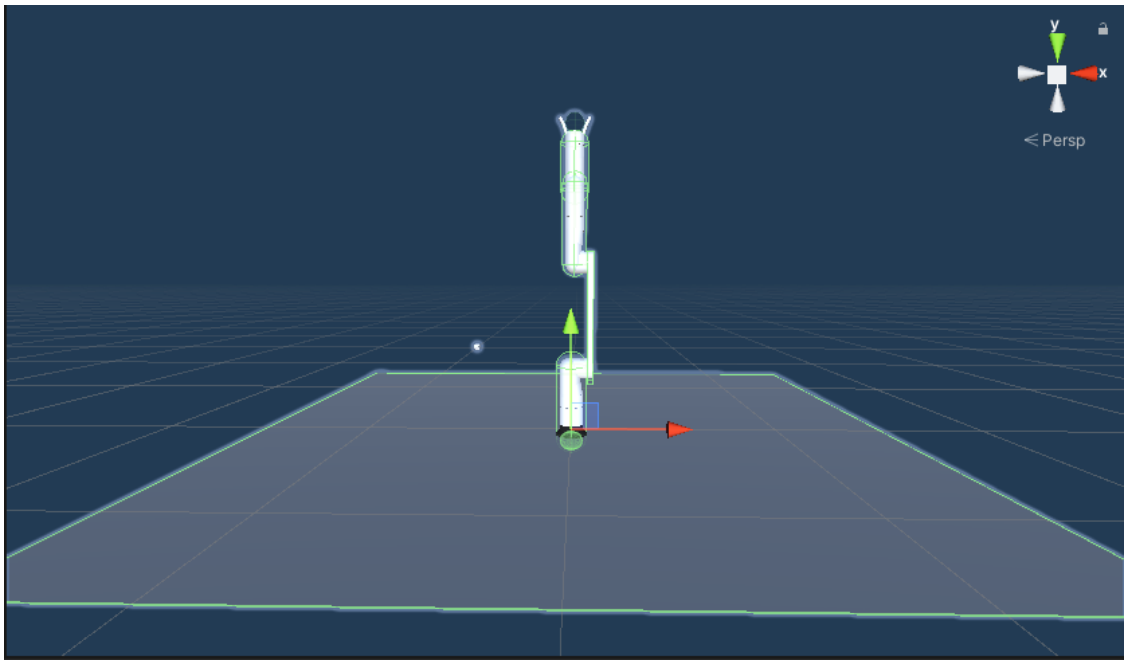

Figure 12 - The completed Kinova Robotic Arm model for the simulation.

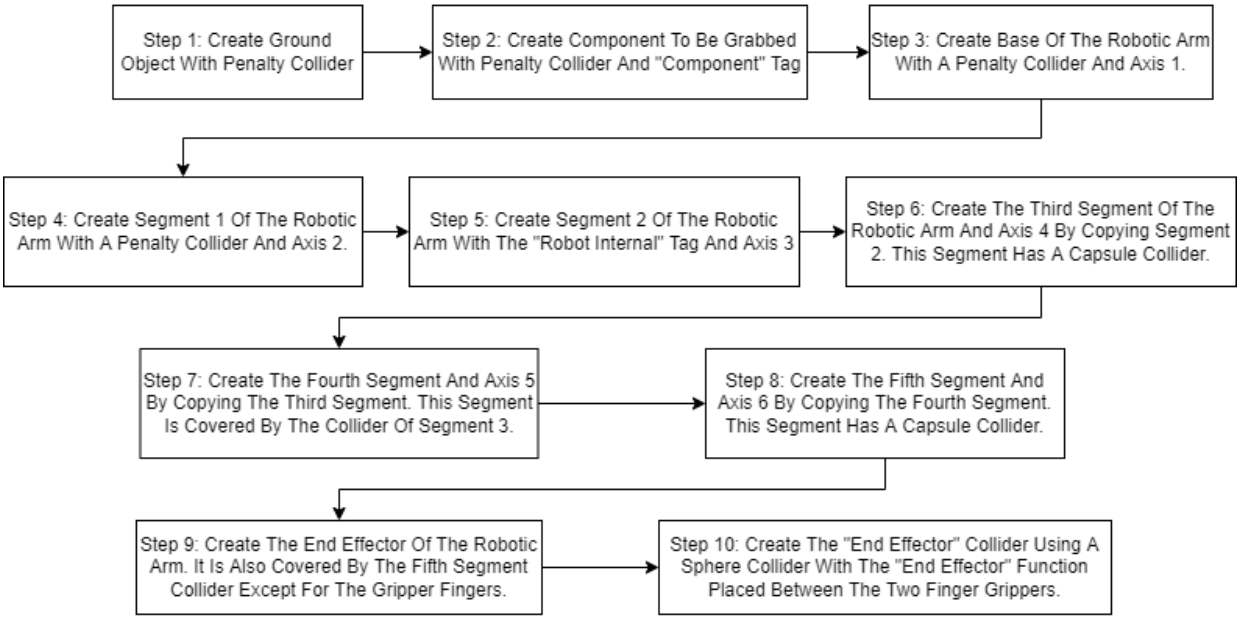

Figure 13 - A flowchart depicting the model creation process. in the flowchart, "Penalty Collider" refers to a collider with the penalty trigger function enabled, and all parts of the robotic arm have the "Robot Internal" tag.

This prefab uses three scripts, the Robot Controller Agent script, the Behavior Parameters script, and the Decision Requester script. The Robot Controller Agent script has the arm axes set to each of the axes one through five, as the goal is to have the end effector reach the object, not position it in a way that allows it to best grab the object. The End Effector setting of the Robot Controller Agent script is set to the EndEffector object mentioned earlier. Similarly, the training mode is set to true and the Nearest Component setting is set to be the component. Moreover, the behavior parameters script includes the behavior name; called Robotarm, the vector observation size, which is the input to the neural network, of 19. It is important to mention, that there are 19 inputs because some of the observations consist of arrays. The actions subsection having Continuous Actions set to five and Discrete Branches set to zero. The Model is set to "None", while training the robotic arm. The Inference type is set to GPU and the behavior type is set to Default. Furthermore, the team Id is set to zero, child sensors are used, and any observable

attribute handling is ignored. Finally, the last script is Decision Requester, which has a decision period of five and “Take Actions Between Deciding” set to true. It should be emphasized that in order to test the model after it has been trained, the Training Mode setting in the Robot Controller Agent needs to be set to false. On top of the parameters set in Unity, the training configuration file also requires specific parameters. These parameters include trainer_type set to ppo, max training steps set to 5 million, summary frequency set to 10000, and time_horizon set to 128.

IV: RESULTS

Results of Python Programs Made To Control the Kinova Gen 3 Lite

It is possible to use python to control the Kinova Gen 3 Lite through a TCP connection to a computer. This was tested by running the various demos contained in the Kinovarobotics Kortex API. While each of these demos gave a different result, they were able to successfully control the Kinova Gen 3 Lite using a python language compiler. The results of one of the more interesting demos, the sequence demo in module 3, can be seen in the following figure.

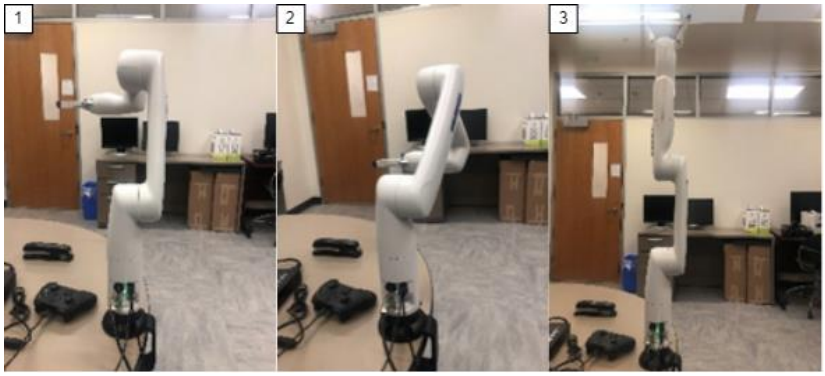

Figure 14 - The three positions the robotic arm visits in the sequence demo. The first image is the home position, the second image has the robotic arm moving slightly outwards, and the third position has the robotic arm moved straight upwards by changing all axis angles to 0 degrees.

As can be seen from figure 14 above, the demo run using python was able to successfully control the robotic arm, displaying that it is possible to use python to manipulate the Kinova Gen 3 Lite. This fact is collaborated by the successful manipulation of the robotic arm using various other demos. Some such demos include the protection zones configuration demo in module 1 as well as the gripper command demo in module 7, both of which can be seen in the figures below:

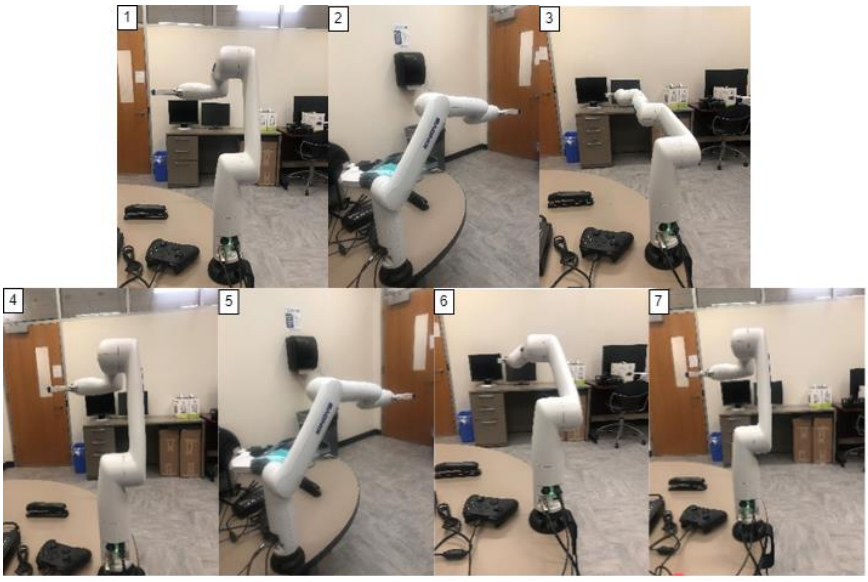

Figure 15 - The positions visited by the robotic arm in the protection zones configuration demo. The arm starts in the home position in the first image. It then extends fully outwards in the second position. Once the outwards movement is completed, the robotic arm reaches the third position, in which all axes are 0 degrees except axis 2, which is at a 90 degree angle. Then, the robotic arm returns to the home position as is seen in the fourth position. After reaching the home position, the process repeats, with the arm extending outwards in the fifth and sixth positions. Finally, the demo ends with the robotic arm returning to the home position once more, as is seen at position seven.

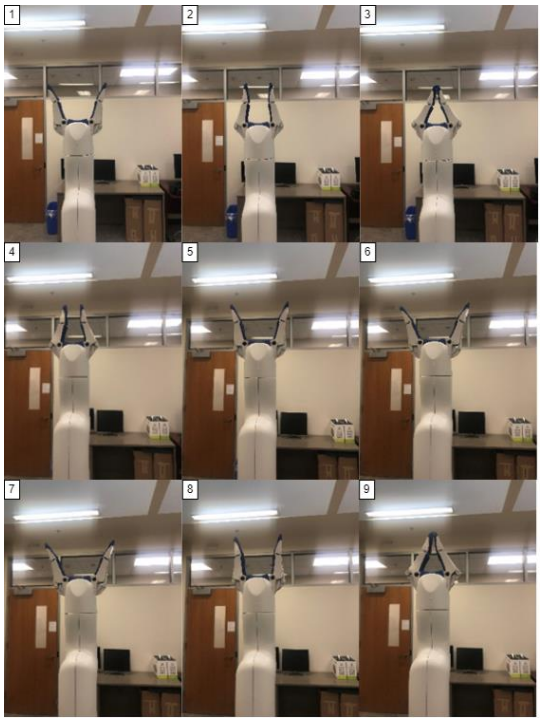

Figure 16 - The movement of the robotic arm's gripper in the gripper command demo. In the demo, the gripper starts completely unclosed in position one. Then, the gripper slowly closes in positions 2 and 3, becoming near fully closed at position 4. Then, the gripper starts opening, with position 5 indicating one of the steps along the way and position 6 indicating the fully opened gripper. Finally, the gripper closes once more, with positions 7 and 8 being intermediate states before the gripper is fully closed in position 9, where the demo ends. It should be noted that the gripper makes jerky movements the first time it closes, but smoothly opens and closes otherwise.

Meanwhile, the two created python programs identify that as long as caution is taken in order to prevent the arm from going into different caution zones, cartesian movements designated in python code can move the robotic arm to the desired position in order to pick an object up and place it down somewhere else without crushing the object. The movements of the robotic arm for the automatic control python program are visible in the figure below.

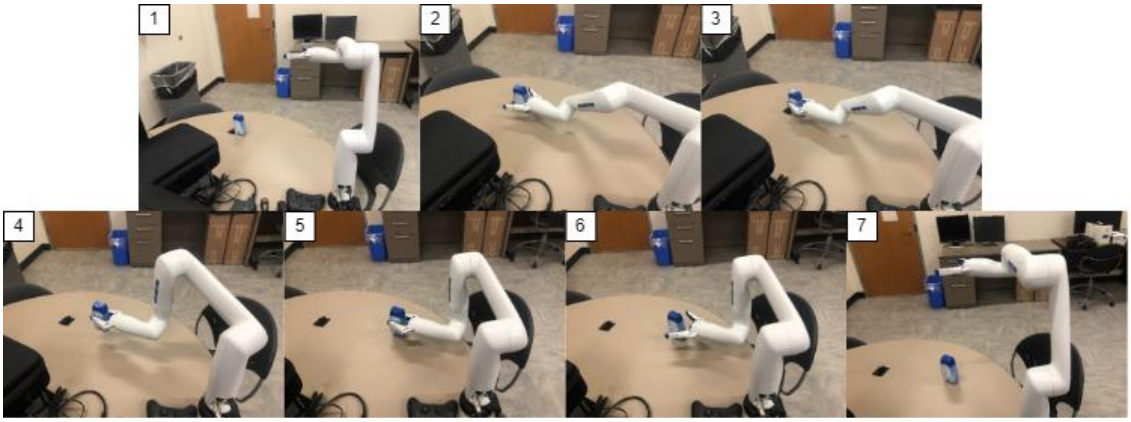

Figure 17 - Positions of the created automatic control program to pick up an object and place it somewhere else. The positions visited are the same in both the automatic and used manual demos, despite the manual demo having the capability of different movements. The robotic arm starts in the home position shown at position 1. It then moves towards the object until the object can be grabbed, like in position 2. After reaching the object, the robotic arm proceeds to grab it, as can be seen from the third position. Once the robotic arm has grabbed the object, it is picked up and placed back down, as is indicated by the fourth and fifth positions respectively. Then, the robotic arm releases the object at the desired end position as is shown in position 6. Finally, the robotic arm returns to the home position as the seventh position, completing the automatic Kinova Gen 3 Lite control program. It should be noted that the reason why the object is moved vertically upwards while being moved is to prevent the object from dragging across the table.

Figure 17 successfully is able to pick up the object and move it to another location as desired. Moreover, the manual manipulation where the user enters the movements along the X and Y axes also functions in a similar manner as long as caution is taken to not move to a range outside the robotic arm's reach. In contrast to the automatic control of the Kinova Gen 3 Lite, the manual manipulation of the arm allows additional movements in comparison to Figure 17, with the program only ending by entering "ESC" as the user's input. Lastly, by having the arm stop closing the gripper when the current

went above 0.75, the Kinova Gen 3 Lite was able to grab objects without deforming them in any permanent manner.

Simulated Gen 3 Lite Robotic Arm Results

An AI built to control a simulation of the Kinova Gen 3 Lite was tested by finding the accuracy regarding how often the simulated Kinova Gen 3 Lite model grabbed the component as well as the average number of steps per successful attempt of grabbing the object, with a failed attempt defaulting to the maximum possible number of steps, 5000. During the simulation, the position of the object to be grabbed was changed each episode, forcing the AI to adapt in order to learn how to grab the object and preventing it from becoming overfitted towards grabbing an object at a specific location. Additionally, this changed position often remained close to the location of the object before its position was changed. Despite this, it was possible for the object to be set slightly outside of the radius of the robotic arm model's reach; however, this occurrence was quite rare so such instances can be excluded from the evaluation of the AI's performance. Due to the lack of prior AIs for the Kinova Gen 3 Lite, there is no comparison between the performance of the AI created in this article and other AIs to control the Kinova Gen 3 Lite. Rather, the performance of the reinforcement learning neural network, a deep Q-network, in response to different parameters is examined.

Change in Behavior Parameter Learning Rate

First, a neural network with four 512 node hidden layers is trained on three different learning rates for the behavior parameter: 0.05, 0.03, and 0.01, when the maximum number of steps possible per episode is equal to 5000, the maximum number of permitted steps per episode. Additionally, the step penalty for this comparison is set to

-0.01. By comparing the performance of the AI when trained with different learning rates, it is possible to identify the effect of the learning rate on the deep Q-network for the Kinova Gen 3 Lite AI when all other parameters are kept the same. The graphs depicting the success rate and average steps to the goal for the each of the neural networks trained with different learning rates are visible below:

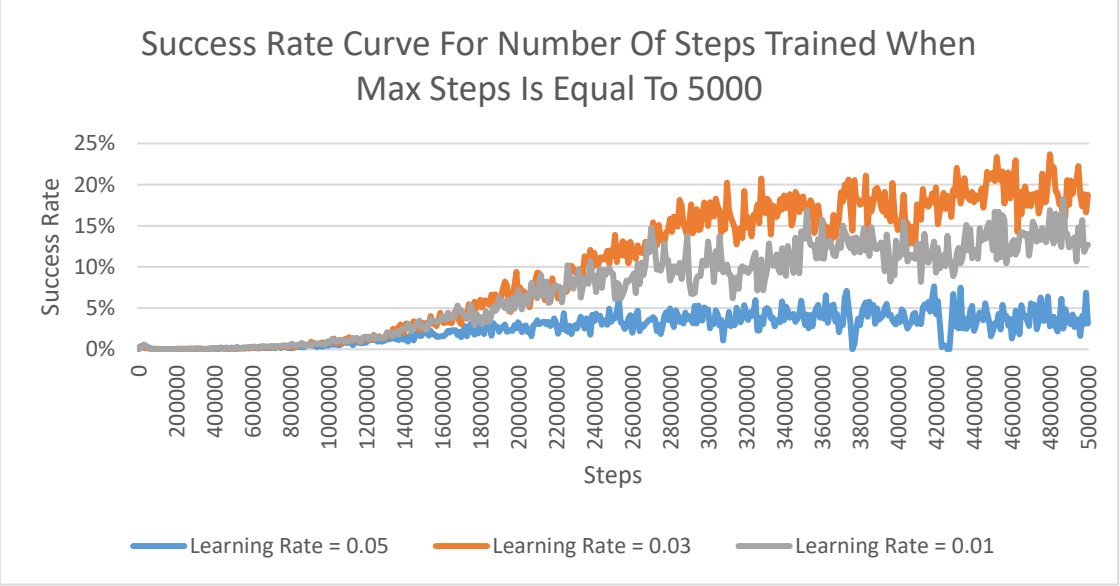

Figure 18 - Success rate curve on a four layer neural network for various learning rates when maximum steps is 5000.

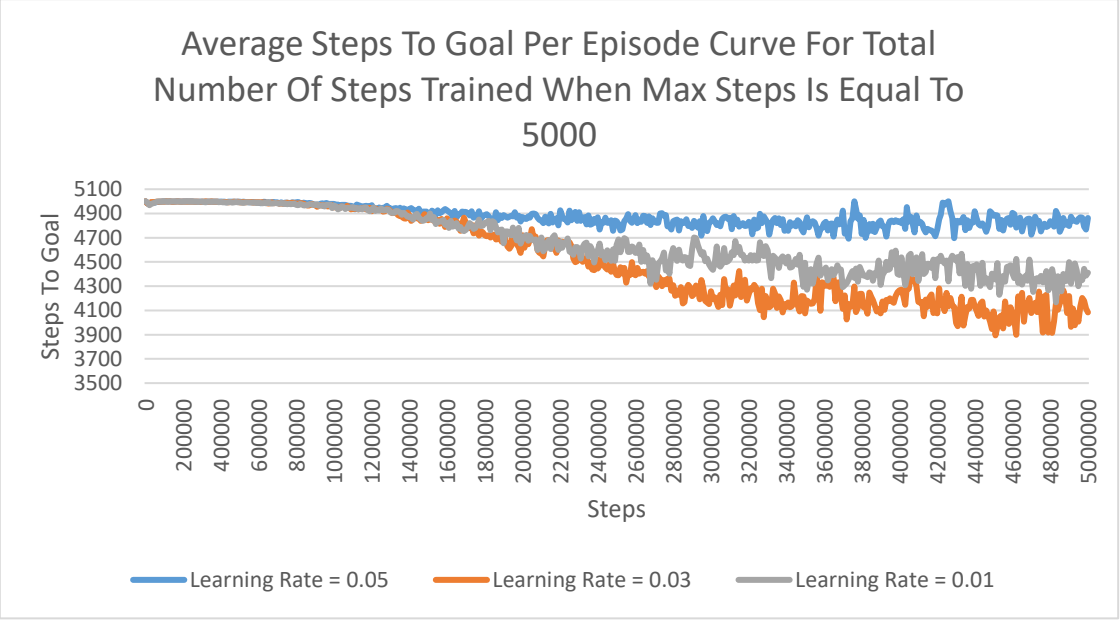

Figure 19 – Steps to goal curve on a four layer neural network for various learning rates when maximum steps is 5000.

As can be identified from the above graphs, the Kinova Gen 3 Lite AI performs the best when the learning rate is set to 3.0e-2, or 0.03. Therefore, it can be identified that without the correct learning rate, the AI can either be overfitted towards specific positions or not learn the overall pattern to reach the object fast enough. In the case of overfitting towards a specific position, the AI becomes unable to grab the object after its position is changed. Evidence of this is visible in the performance of the neural network when the learning rate is 0.05. On the other hand, the AI just does not learn fast enough when the learning rate is 0.01, often preventing the arm from properly reaching the object because the weights are not biased enough to reach towards the location of the object in response to a particular set of inputs. Specifically, the neural network becomes unable to move the object after the position is changed, while still maintaining the ability to move towards the general direction of the object.

On top of the general inference that can be taken from the above graphs, it is important to know what the parameters in each of the graphs mean, as there are multiple parameters with similar names and meanings. In particular, the steps are the total number of steps taken, which are observations collected and actions taken. Meanwhile, the step to goal is the same as the steps taken except that the steps to goal resets whenever the object is grabbed or 5000 steps have been taken. Furthermore, the agent resets when the steps to goal resets. Additionally, there is around a 28.608 second time interval on average between each different instance of data. It should also be noted that because the decision interval is set to five, the agent only takes an action every five steps, meaning that while the maximum number of steps for a single episode is 5000, the robotic arm's agent can only take 1000 steps at maximum each episode.

Example of Simulated Model for Successful and Failed Episodes

While the overall performance of the agent is the most important metric of the Kinova Gen 3 Lite AI, it is also important to identify the position of the simulated robotic arm in both a failed episode and a successful episode. Images identifying a failed episode and a successful episode respectively are visible in the two following figures:

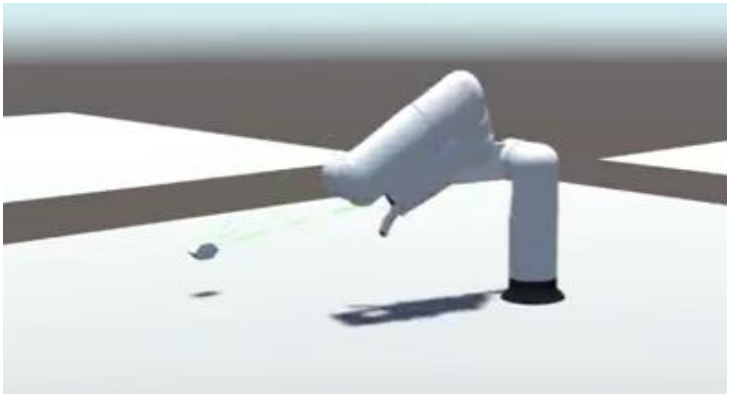

Figure 20 - Failed episode in which robotic arm hits itself.

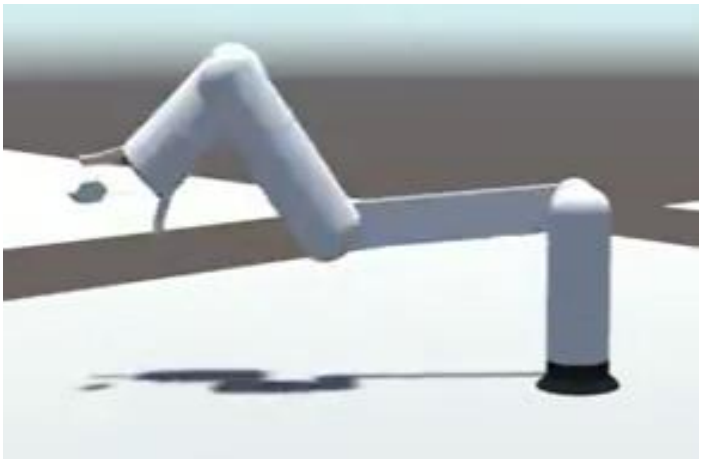

Figure 21 - Successful episode in which the object is grabbed.

As can be seen in the above two images, an episode is only counted as successful when the object is within the spherical "End Effector" collider placed between the two gripper fingers on the Kinova Gen 3 Lite model, meaning the object can be grabbed. Otherwise, the episode is counted as a failed episode if it ends due to a collision or reaching the maximum allowed steps.

Change in Maximum Steps Permitted

On top of identifying the optimal learning rate for the robotic arm's agent, the optimal max step count can be found by comparing the accuracies of the agent on the optimal learning rate, 0.003. To be specific, the accuracy of the agent with a max step of 5000 is compared to the accuracy of the agent with a max step of 500. Accuracy is the only metric used for comparison because the steps to goal curve will have different maximum values due to differences in the max step count of each neural network, making it a bad indicator of which max step count results in a better performing agent. The graph for the success rate curve of each neural network trained with a different maximum step count is visible in the following figure:

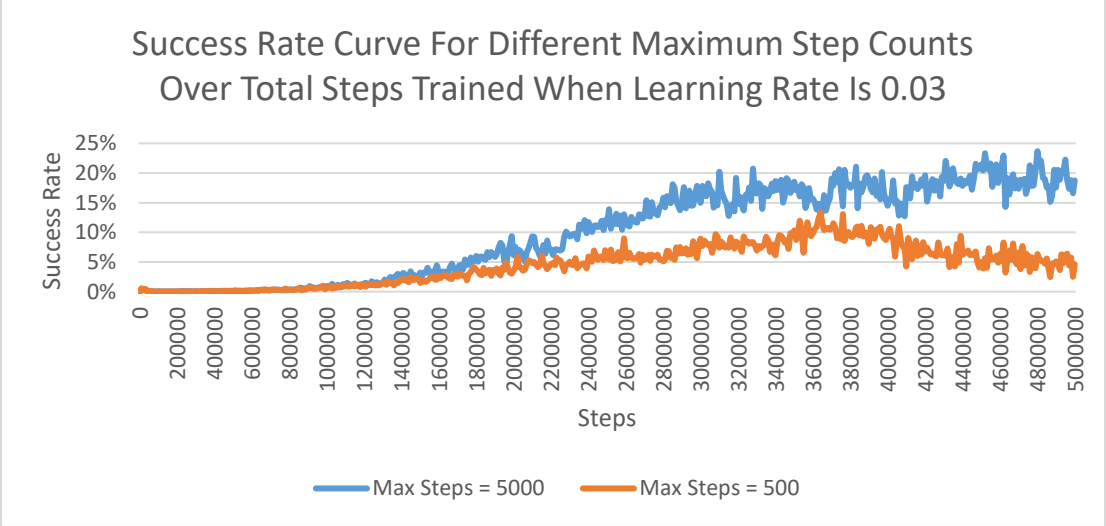

Figure 22 - Success rate curve on a four layer NN for various max step counts when the learning rate is 0.03.

As can be identified from the above figure, the neural network with a max step equal to 5000 is a better performing model. The most likely cause of this is that the neural network with 500 max steps does not have enough steps within a particular episode in order to map a sufficient route to the object. Rather, it is more likely that the robotic arm agent does not learn the proper weights in order to reach towards the goal earlier on when

the robotic arm moves completely randomly. This cascades into having a lower accuracy in reaching the goal later on because the weights did not have enough time per episode in order to update themselves to better help the robotic arm reach the object. Thus, a max step count of 5000 is the optimal max step count for the neural network.

Consistency of Results and Average Performance

On top of identifying the maximum step count, it is important to determine the consistency of the performance and the average performance of the neural network. To evaluate these metrics, one can take the average success rate and steps to go curve for multiple runs and compare it to an earlier instance of a training run with the same parameters, particularly the performance of the training run with a learning rate of 0.03 in figures 18 and 19. Thus, by repeating the process of creating the neural network with a learning rate of 0.03 and max steps of 5000, one can identify the consistency and average performance of the neural network. The graphs showing the average success rate and step to go curve over all training runs of the neural network can be found below:

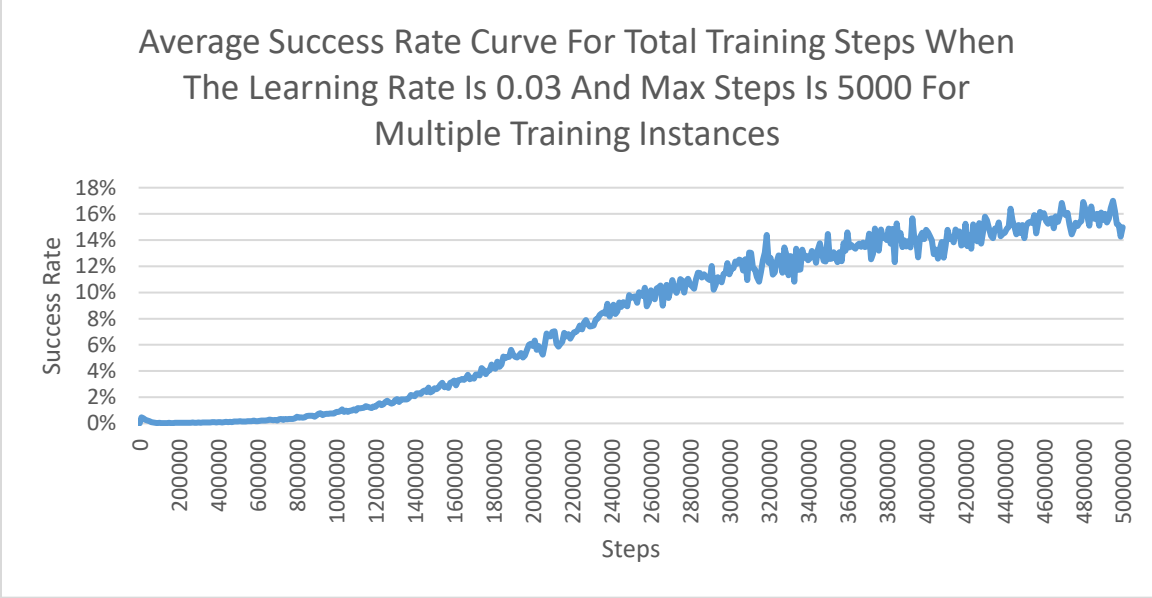

Figure 23 – Average success rate curve of a four layer NN with a learning rate of 0.03 and a max step count of 5000.

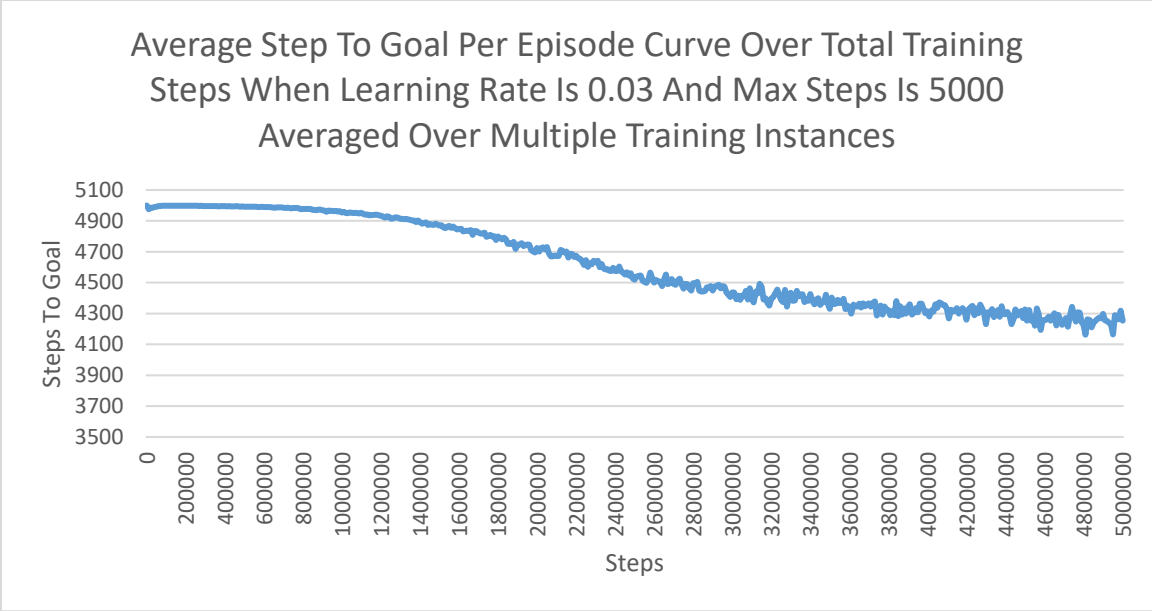

Figure 24 - Average steps to goal curve of a four layer NN with a learning rate of 0.03 and a max step count of 5000.

As can be observed from the above graphs, the agent ends at around 16% accuracy on average and takes around 4300 steps to the goal on average per episode. It should be noted that the steps to goal curve is heavily influenced by the accuracy of the agent in that the steps to the goal for a particular episode is set to 5000 if the agent fails to reach the goal. As such, the number of steps taken on a successful attempt is much lower but has no particular pattern in general. It can also be identified that the first attempt of a neural network with a learning rate of 0.03 and a max step count of 5000 was an outlier, performing quite a bit above average at around 18% accuracy at the end of training.

Training With One Instance of the Environment

There are multiple possible causes of the agent's low accuracy, be it the randomization of the objects end position placing the object out of range or the robot arm getting stuck. On top of this however, there was the potentiality that the number of instances used in training also had an effect on the end performance of the agent. In particular, three instances of the entire environment were used in unison to train the AI.

As such, it was possible that the successful attempts were only being recorded for one of them, while all three increased the total number of training steps. Therefore, by using a training with only one instance of the simulation, it was possible to verify if the number of simulated environments used for training had an effect on the end performance. The graphs for the success rate and step to goal curve for the training instance of the robotic arm in which only one environment was used are visible in the following figures:

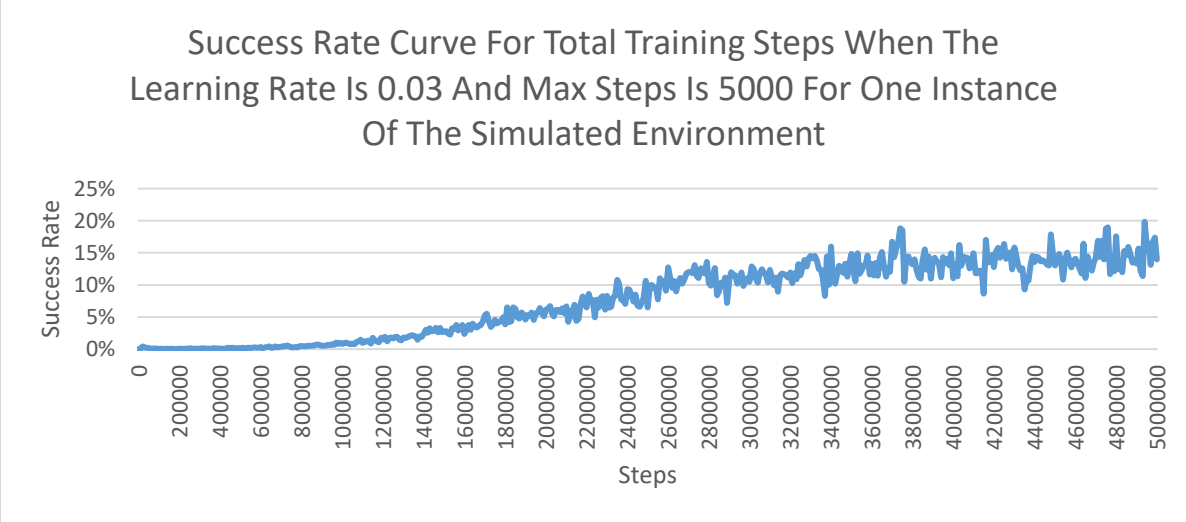

Figure 25 – Solo instance success rate curve of a four layer NN with a 0.03 learning rate and a max step count of 5000.

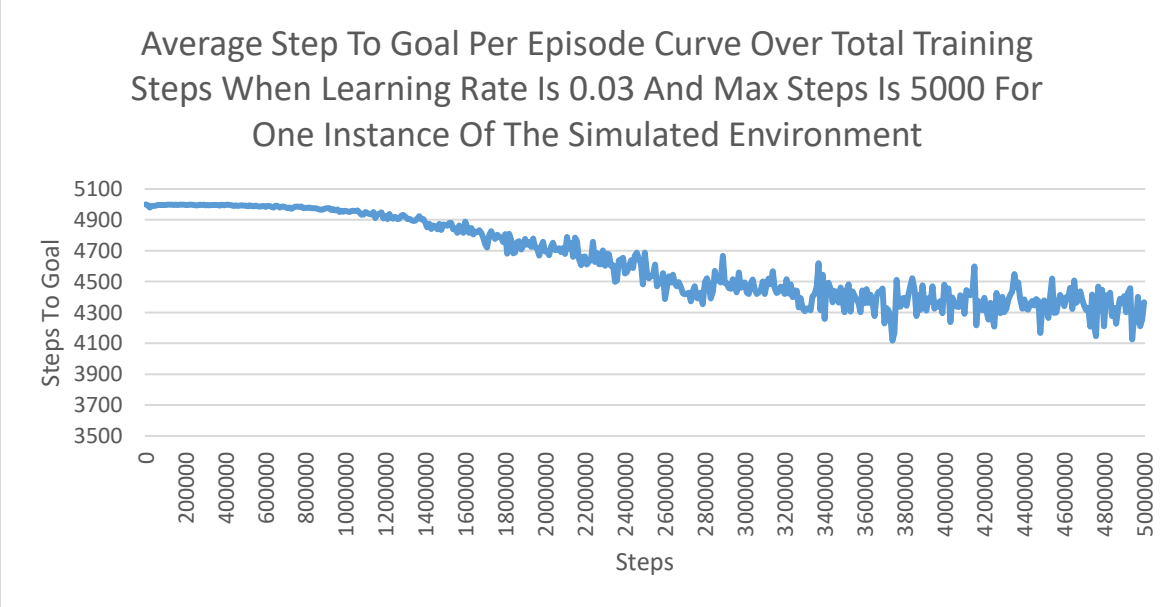

Figure 26 – Solo instance steps to goal curve of a four layer NN with a 0.03 learning rate and a max step count of 5000.

The above two figures identify that the number of environments run in parallel when training the agent does not have an effect on the end result, as is visible from the similar end accuracies of 15% for the solo environment compared to 16% for the average accuracy of three environments. Similarly, the agent trained with one environment and the agent trained with three environments both had an end average steps to goal per episode of around 4300. Thus, the conclusion was made that the number of instances of the robotic arm did not affect the end outcome of the agent. Worthy of note is that multiple instances do not refer to multiple robotic arms within the same environment, but rather parallel environments with their own robotic arms that also use the same agent.

Change in Number of Layers in Neural Network

The structure of the neural network can also play a role in the overall accuracy of the neural network. In particular, a network with more layers can have a different performance than the basic neural network. The following figures display the accuracy and steps to goal curves of a neural network with five layers each with 512 nodes and a slightly lower learning rate of 0.003.

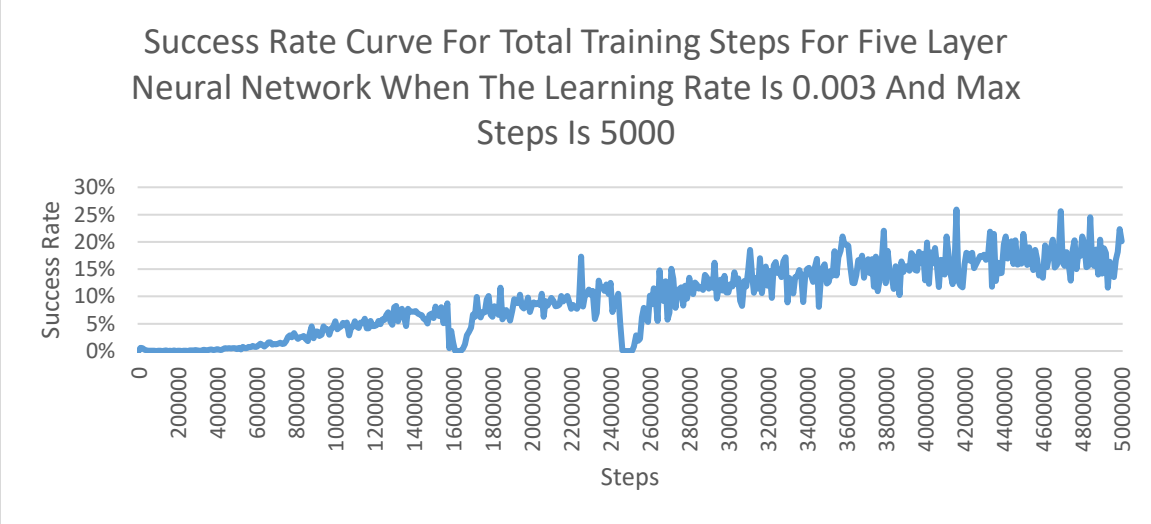

Figure 27 – Success rate curve of a five layer NN with a learning rate of 0.003 and a max step count of 5000.

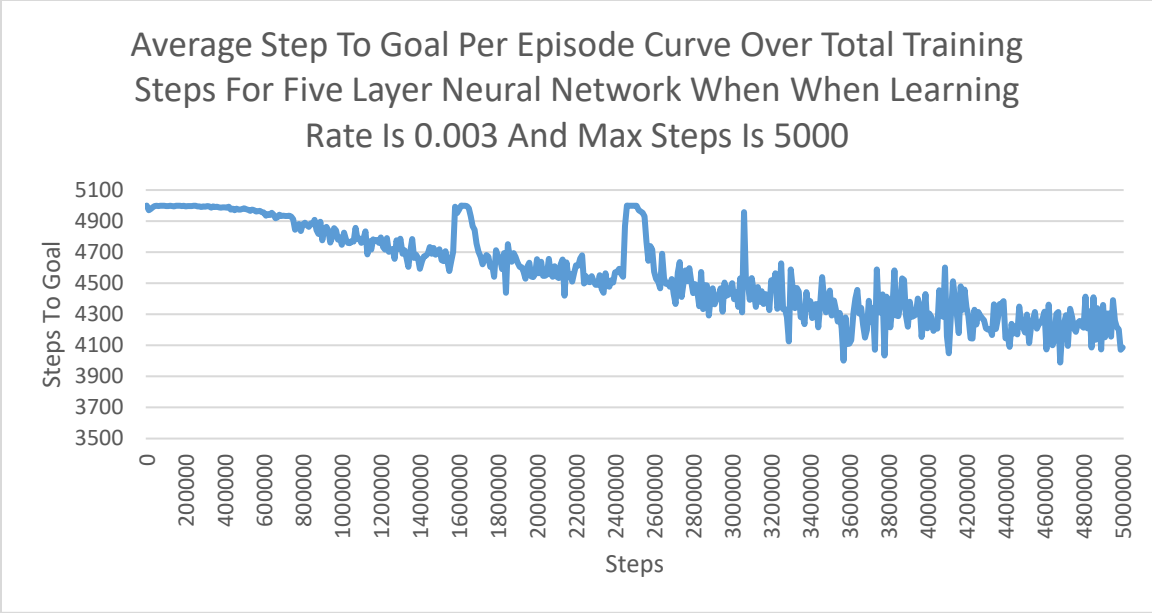

Figure 28 - Steps to goal curve of a five layer NN with a learning rate of 0.003 and a max step count of 5000.

The above figures identify that a larger neural network size slightly increases the overall performance. In particular, the five layer neural network with the same parameters has just below 20% accuracy in comparison to a four layer neural network. Similarly, the end steps to goal is around 4100 for the five layer NN in comparison to an average steps to the goal of around 4300 at the end of training for a four layer NN. Thus, it can be identified that the size of the neural network has a decent impact in the performance of a Kinova Gen 3 Lite agent. Additionally, a smaller neural network was tested with only three layers, but all attempts ended up crashing before the training session could be completed. Furthermore, these training instances of the smaller neural network performed worse on average than the four layer neural network, originally having a very good performance in comparison to the four and five layer neural networks and then becoming completely unable to reach the component at around one million training steps. Despite this, the earlier results of the smaller neural networks support the fact that the number of layers in a neural network can impact the network's performance.

Change in the Step Penalty

Finally, the last main parameter to most likely have a large impact on the neural network is the step penalty. In particular, the performance of a smaller and larger network with a step penalty of -0.05 and -0.005 in comparison to the earlier step penalty of -0.01 is explored. It should be pointed out that in order to match the step penalty of -0.05, the batch size was changed for both networks, the number of nodes per hidden layer was changed to 1024 for the five layer neural network, and the learning rate was changed to 0.0003 for the three layer neural network. Similarly, during this phase, it was identified that the learning rate changed prior was the learning rate of the behavior parameters neural network and not the neural network for the robotic arm agent. Despite this discovery, there were no successful complete training instances of a robotic arm agent with a changed learning rate, as most of them became overfitted very early, causing them to have an accuracy near zero. The changes to the batch sizes depended on the size of the neural network, with the three layer neural network having a batch size of 128, and the five layer neural network having a batch size of 256. Graphs displaying the accuracy and step to go curves of both networks tested with a step penalty of -0.05 can be found below:

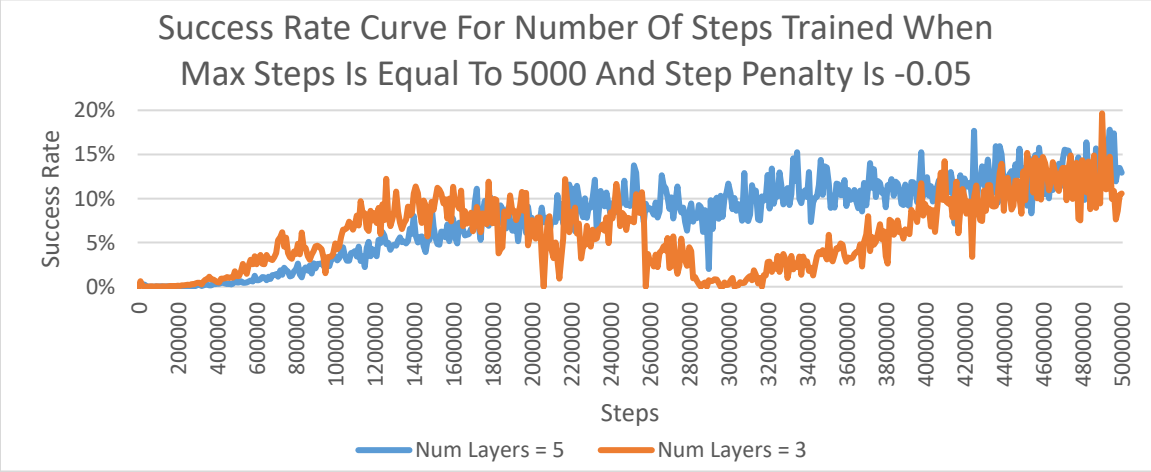

Figure 29 - Success rate curve of two NNs when step penalty is -0.05 and max step count is 5000.

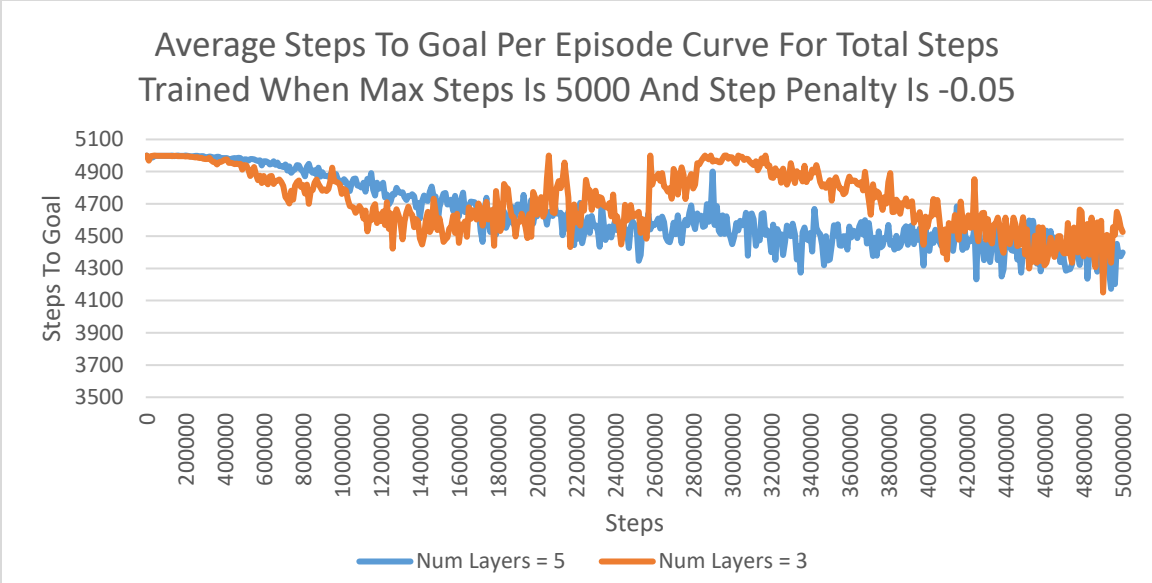

Figure 30 - Steps to goal curve of two NNs when step penalty is -0.05 and max step count is 5000.

As depicted in the above graphs, the neural network with five layers is more consistent in its accuracy and step to go curves than the three layer network. Despite this, both networks have a similar end performance. It can also be identified that the step penalty has a large impact on the result of the networks, as both networks trained with a step penalty of -0.05 perform significantly worse than the networks trained with a step penalty of -0.01. In particular, the networks trained with a step penalty of -0.05 have an end accuracy of around 13% and an end average steps to goal per episode of 4400 in comparison to the average accuracy of 16% and average steps to goal per episode of 4300 that the neural networks trained with a step penalty of -0.01 have.

While the networks trained with a step penalty of -0.05 have a worse performance than the networks trained with a step penalty of -0.01, the same cannot be said about the networks trained using a step penalty of -0.005. In order to match the step penalty of -0.005, the learning rate of both networks was set to 0.0003, the batch size of the three layer network was set to 256 and the batch size of the five layer network was set to 512.

The following graphs display the accuracy and step to go curves for both networks trained with a -0.005 step penalty:

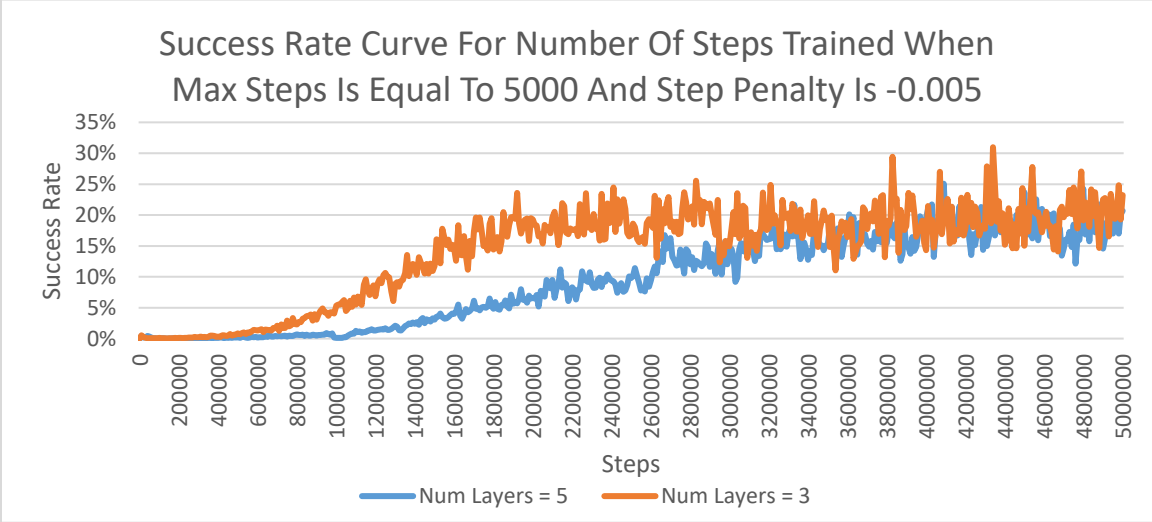

Figure 31 - Success rate curve of two NNs when step penalty is -0.005 and max step count is 5000.

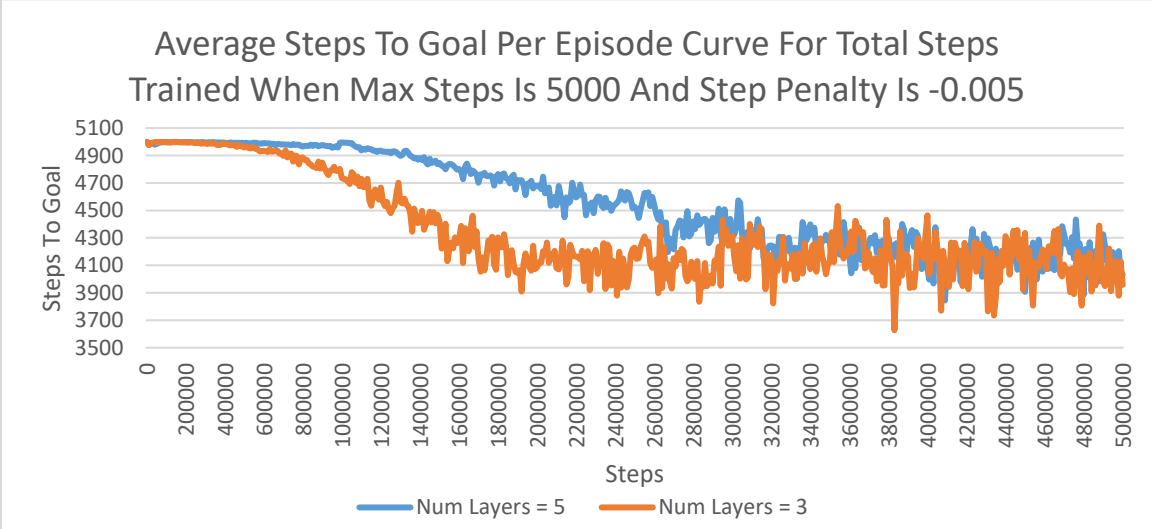

Figure 32 - Steps to goal curve of two NNs when step penalty is -0.005 and max step count is 5000.

As can be identified from the above figures, both the three layer and five layer neural networks have a significantly better performance than the neural networks trained with a step penalty of -0.01. Specifically, both networks trained with a step penalty of -0.005 have an accuracy of just above 20% and an end average steps to goal per episode of 4000, which are better in comparison to the average performance metric values for the

networks trained with a step penalty of -0.01 mentioned earlier. Thus, it can be concluded that the step penalty has a large impact on the end performance of the Kinova Gen 3 Lite AI.

V: DISCUSSION

Physical Manipulation of the Kinova Gen 3 Lite Robotic Arm

On top of using Python to control the robotic arm, the general ease of use of the other two control methods was compared. While this comparison does not have an impact on the study of Python manipulation of the robotic arm, it can be used as an introduction into controlling the Kinova Gen 3 Lite in general, providing an easy transition into the aforementioned subject. In particular, the two other methods to control the robotic arm other than programs are the keyboard and the controller. To use a controller, the controller must be plugged into the robotic arm. From there, the robotic arm can be controlled by moving the joysticks and pressing certain buttons, such as the left and right buttons to open and close the gripper or to change the mode of control. On the other hand, using the keyboard requires connecting the robotic arm to the computer and then logging into the Kinova Web API at a specific local port, 192.168.1.10, on a browser. The setup of keyboard control for the Kinova Gen 3 Lite can be seen in the below image:

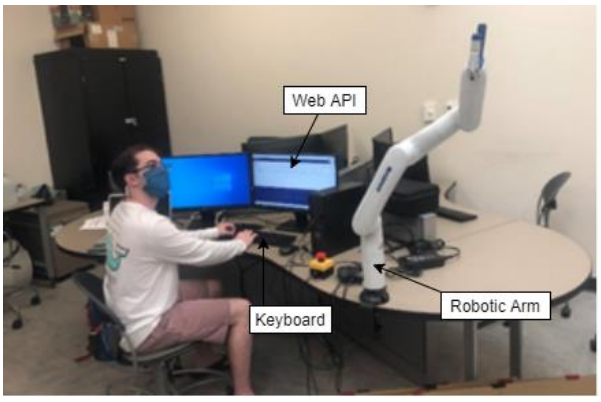

Figure 33 - Keyboard control of the Kinova Gen 3 Lite using the Web API.

Both methods of control were used for a variety of tasks, such as moving around objects, in order to evaluate their ease of use, with two different participants giving their opinion. Overall, both testers found that while the controller was more difficult to use initially, it became more convenient to use than the keyboard after a period of time was spent moving the robotic arm with the controller. Despite this outcome, the actual ease of use of the different methods of control cannot be properly evaluated due to the small test pool and lack of proper metrics to compare the two control methods. Thus, the better method of control remains undetermined. The following figure identifies one of the tests conducted to evaluate the control methods ease of use:

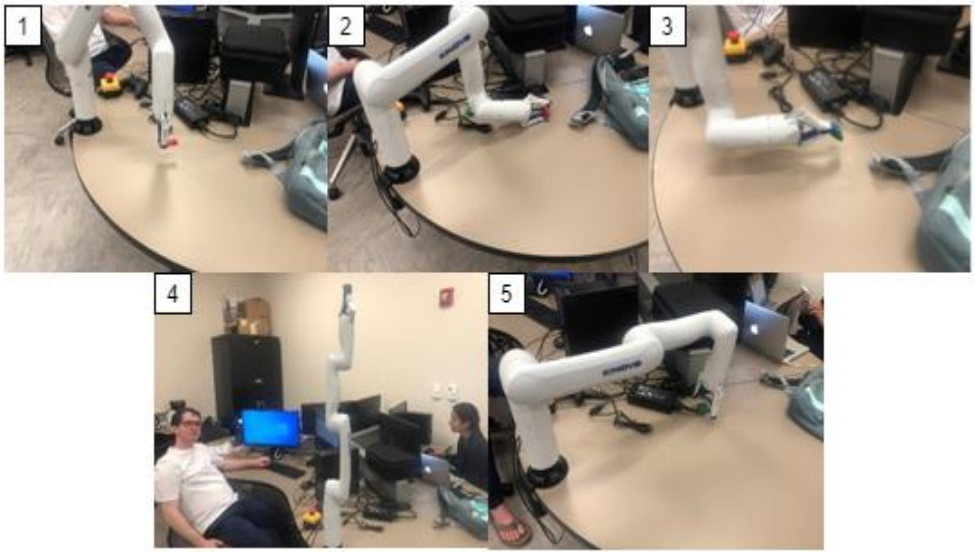

Figure 34 - Keyboard is used to control the Kinova Gen 3 Lite to interact with a red and green clip. In steps 1 and 2, a red clip is grabbed and clipped onto a cord using the robotic arm. Meanwhile, in steps 3 and 5, a green clip is grabbed and clipped onto a cord. Lastly, step 4 shows the keyboard being used to point the robotic arm straight upwards. Step 4 is done after the green clip is grabbed in step 3, but before the green clip is released in step 5.

Kinova Gen 3 Lite Robotic Arm Manipulation in a Unity Simulation Using Python

The majority of the instances with a larger learning rate for the robotic arm agent completely failed in learning the desired pattern, having accuracy near 0. The agents trained with a larger learning rate for the robotic arm agent sometimes had a better performance earlier in the training process than the overall end performance of the agents

trained with a smaller learning rate. In order to run a training instance with a larger learning rate for the robotic arm, a corresponding change to the batch size was required. The changed learning rate used was double the original learning rate of the robotic arm agent, 0.0003, making the new learning rate 0.0006. In particular, the batch sizes used when the step penalty was set to -0.05 were doubled to 256 for the small network and 512 for the large network. On top of increasing the batch size, a smaller step penalty of -0.005 was used in order to further compensate for the larger learning rate. These changes in the variables were considered when evaluating the effect of the change in the learning rate parameter of the robotic arm agent, with both of the changes lowering the chance for the agent to become overfitted towards one movement while also lowering the rate at which a specific pattern of movements was learned to reach towards the object.

In the neural networks created with the learning parameters described above, the three layer network performs significantly well before 1.6 million training steps and completely fails after 1.6 million training steps are reached. Meanwhile, the five layer neural network has accuracy near zero throughout the entire training process. Using the metric curves of the 0.0006 learning rate trained neural networks, it can be easily established that the performance of both neural networks differ significantly from the performance of all neural networks with a robotic arm agent learning rate of 0.0003. Thus, it can be inferred that the learning rate of the robotic arm has the largest effect on the performance of the neural network but requires a large amount of calibration in order to find the optimal learning rate. It should be noted that while the instance of the five layer neural network completely failed very early into the training process, a majority of the other five layer neural network's run with similar parameters had a significantly

better performance before they completely crashed early on, causing the training process to stop. As such, the performance seen in the graphs for the five layer neural network is not completely reflective of its actual performance with a larger learning rate. Rather, for the instances that crashed early on, they performed slightly worse than the agents trained on a five layer neural network with the basic robotic arm agent learning rate. Therefore, the three layer neural network is the network used for comparison with the other trained neural networks. In particular, before the network became overfitted around 1.6 million training steps, it performed significantly better than any other robotic arm agent trained during the research, reaching accuracy around 20% on average early into the training process and reaching steps to goal of around 4100 at around the same point. These performance metrics were close to that of the end performance of the agents trained with a robotic arm learning rate of 0.0003. Consequently, it was determined that the learning rate has the largest impact on the performance of the robotic arm agent but requires a significant amount of fine tuning in order to train the agent without becoming overfitted.

The various training runs used to create the Kinova Gen 3 Lite agent also identify some other aspects of the simulation alongside the impact of the neural network parameters. In particular, some limitations of the simulation due to its setup can be inferred from the agent's best performance, which are somewhat similar across all iterations of the agent that do not become overfitted. Specifically, the end or best performance of the agent is usually somewhere around a 20% accuracy and 4100 steps to the goal at best. The reason why this near shared performance across all training runs identifies a possible limitation of the simulation setup is because a similar accuracy is encountered as the best performance of the agent with a robotic arm learning rate of

0.0006, even though the agent becomes overfitted around 1.6 million training steps. Therefore, it may be the case that the robotic arm is unable to get an accuracy above 25% on average due to the overall environment of the robotic arm simulation, be it that the object is out of reach of the robotic arm, or at a position that the Kinova Gen 3 Lite cannot reach with its limited range of movement.

Another possible cause of the agent's limitations is that the end position of the object may not change enough between each episode. As such, the AI may become overfitted towards reaching to a specific direction if the position of the object does not change enough within a specific amount of training steps.

On the other hand, it cannot be discounted that the agent's low performance is a result of the neural network. Whether it is the tested parameters or the parameters not tested during the research, such as the memory length, the neural network may lack the necessary fine tuning required in order for the agent to reach an excellent performance when reaching the object.

Attempted Dual Kinova Gen 3 Lite Robotic Arm Physical Manipulation in Python

In order to control two Kinova Gen 3 Lite Robotic Arms at once, two arms were moved into the same room and had their bases placed just out of reach of the other robotic arm. The robotic arms were then connected to the same computer in a setup visible in the below figure:

Figure 35 - Dual arm physical setup.

From there, I tested a multitude of methods to connect to both robotic arms at once, moving onto the next method after encountering an error I was unable to fix. The first method tested was the usb.core python extension. This extension required the installation of libusb0 as well as MinGW, which was needed to install libusb0. Then, the autogen.sh file was run repeatedly to identify any errors and fix them; all such errors were solved by changing a pointer to look at the correct location. After fixing all these pointer errors, a final error in downloading libusb0 was encountered in the configure.ac file of MinGW. This error was caused by the AC_PREREQ function, which was unable to be edited in any way. After failing to install libusb0 with MinGW, another method of installing libusb0 was attempted. This installation was completed by downloading the libusb-32win.zip and extracting the libusb0.dll file to the DLL folder of python. From here, the final error in trying to use usb.core was identified, that the robotic arm does not

register as a usb to usb.core. The entire process of attempting to control two robotic arms at once using usb.core can be found in the following figure:

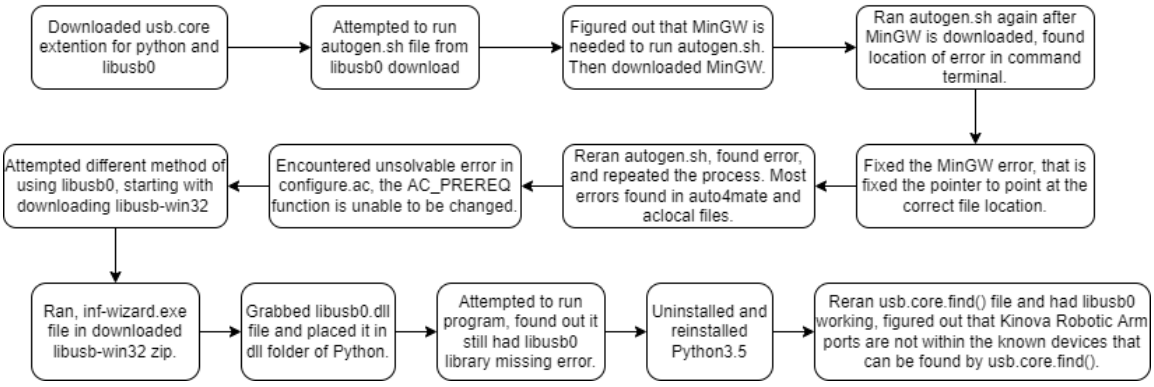

Figure 36 - Flowchart of using usb.core to try and control two robotic arms at once.

After attempting to use usb.core to control two robotic arms, the usb.busses extension was used to try and control two robotic arms at once. In the end, usb.busses had the same error as usb.core that the robotic arms did not count as busses. The process of trying to use usb.busses to control two Kinova Gen 3 Lites at once can be seen below:

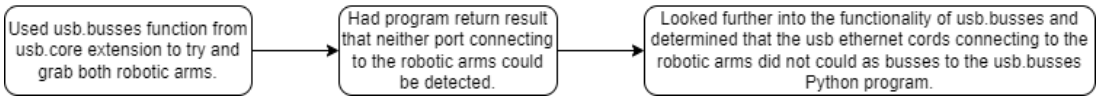

Figure 37 - Flowchart of using usb.busses to try and control two robotic arms at once.

Similarly, the pyserial python extension was attempted and had the same result that the robotic arms did not register in any ports. The reason for this is because the Kinova Gen 3 Lites were connected using an ethernet cord and as such counted as network adapters, as can be seen in figure 38 below. The overall process of trying to use pyserial to control two robotic arms at once can be found in the second following figure.

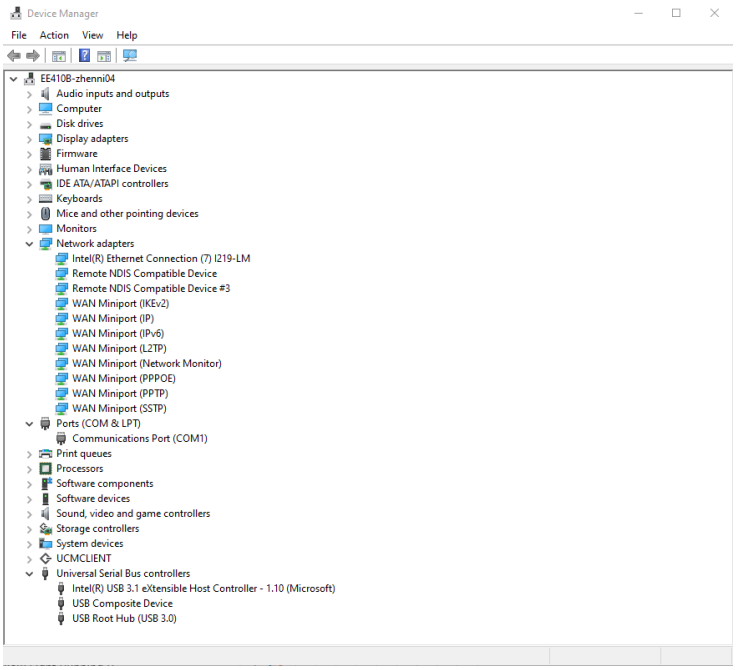

Figure 38 - Device Manager Window of computer. Shows that robotic arms count as network adapters.

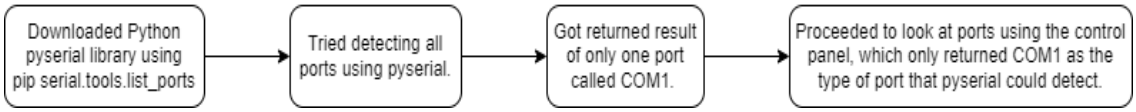

Figure 39 - Flowchart of using pyserial python extension to try and control two robotic arms at once.

Other than the above three attempted methods to control two Kinova Robotic Arms at once, there were three other methods used to try and control both robotic arms at once which also failed. One such other method was the netifaces extension, which had the issue of only displaying the local port that the robotic arms connected to and the inability to connect to either robotic arm. The output of netifaces in python can be seen in the following figure, figure 40. Additionally, the overall process of downloading and attempting to use netifaces to connect both robotic arms to the same computer can be found in figure 41.

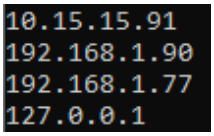

Figure 40 – The two middle addresses are the local ports of the two robotic arms connected to the computer at once.

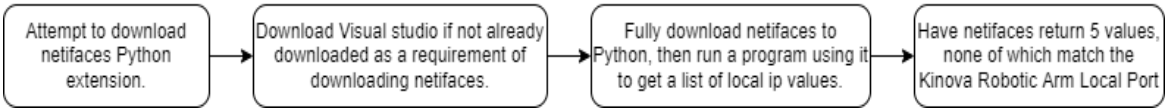

Figure 41 - Flowchart of using netifaces python extension to try and control two robotic arms at once.

Contrarily, the socket python extension had the ability to connect to one of the robotic arms and not the other, like the last attempted method to control both robotic arms at once. The steps followed to discover this outcome about socket can be seen below:

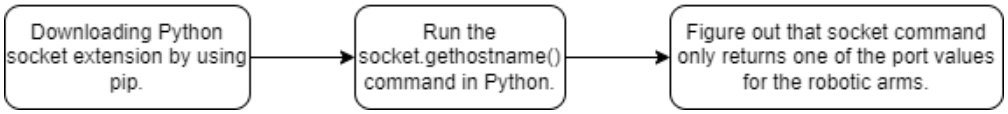

Figure 42 - Flowchart of using socket python extension to try and control two robotic arms at once.

The last method used to try and control both Kinova Gen 3 Lites involved the netmiko extension. This extension had the ability to connect to one of the robotic arms but had the issue of being unable to connect to both robotic arms at the same time because the external id had to be used instead of the local id. This led to a check of the active ports and the realization that only one of the robotic arms was active. Similarly, through trial and error, it was determined that there was no way to change the foreign, or external, id, even connecting to one of the robotic arms wirelessly did not fix this issue. The example of only one robotic arm being active at a time can be seen in the below figures, it should be noted that the local id changes if a robotic arm is turned off and then turned on again, causing the slight difference in the local id numbers.

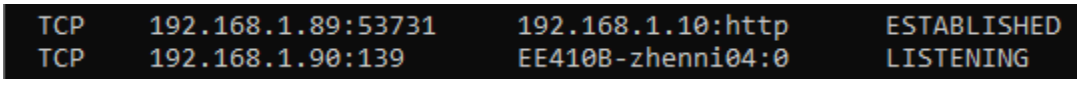

Figure 43 - TCP connection between one robotic arm and remote address 192.168.1.10. Other arm does not connect.

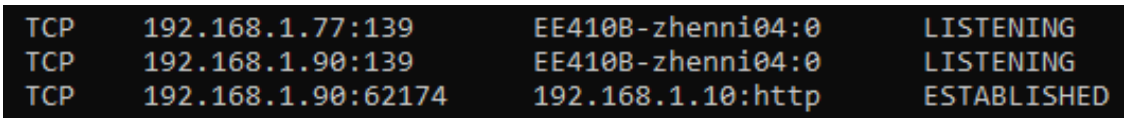

Figure 44 - TCP connection between other robotic arm and remote address 192.168.1.10. This occurs after the first arm is disconnected and reconnected to the computer. After this occurs, the original arm does not connect.

The steps taken when attempting to use netmiko as well as the steps that led to the discovery of the robotic arms having the same remote address are visible below:

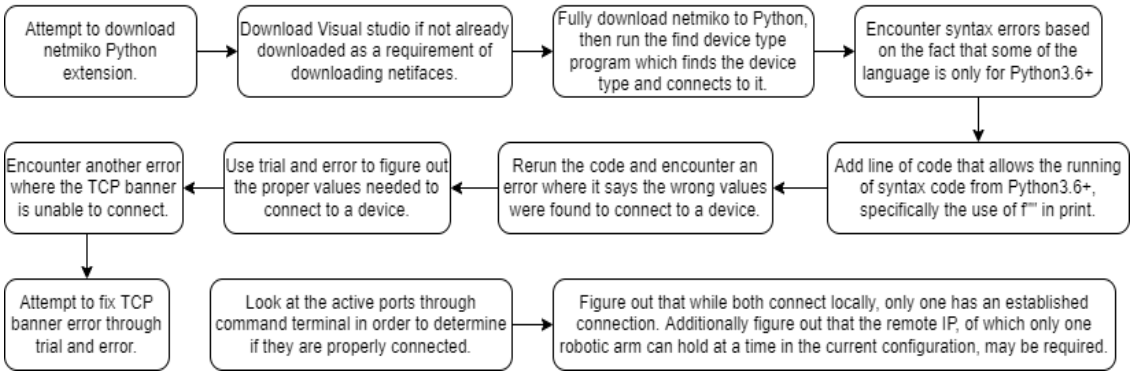

Figure 45 - Flowchart of using netmiko python extension to try and control two robotic arms at once.

Despite this, it must be noted that the attempts to control two robotic arms at once are by no means totally comprehensive. There most likely exist a multitude of other methods which could be used to control two Kinova Gen 3 Lite robotic arms at once. Not every possible control method was explored, such as connecting one of the robotic arms to a router, which may allow a change in foreign id, solving the issue.

Dual Kinova Gen 3 Lite Arm Control Attempt in a Unity Simulation Using Python

An attempt to create an AI to control two Kinova Gen 3 Lite robotic arms in unison using a simulation was also undertaken. To be specific, the unity simulation used two separate agents, one for each arm, with the goal of picking up an object, passing it to the arm that did not pick up the object, and then placing the object at an end location. A multitude of changes to the solo arm simulation model and scripts were required in order to attempt to train agents to control a dual arm simulation. For starters, the model is created by expanding the length of the ground component. After the ground has been expanded, the entire robotic arm from the solo arm simulation is copied and moved an arm's length away from the first arm. For clarification, an arm's length refers to the

maximum distance the Kinova Gen 3 Lite simulated model can reach away from its own base. The copied arm is then rotated 180 degrees so that the arms are facing one another, and the location of the penalty triggers are changed so that the parts of the first robotic arm that do and do not have penalty triggers are swapped on the second robotic arm. In other words, the penalty trigger is enabled on the third, fourth, and fifth segment on the second robotic arm, while the penalty trigger is disabled on the first and second segments of the second robotic arm. The reason for the penalty triggers being swapped is to make sure that collisions between the first and second robotic arm are penalized, as only the third, fourth, and fifth segments can collide with each other during the collaboration step of the task. Once the models of the two arms are complete, the final part of the total dual arm simulation model is introduced, the goal component. The goal component is a flat circle that is located near the second robotic arm and gives a reward when the second robotic arm collides with it while holding the object. The overall model for the dual arm simulation can be seen in the following image:

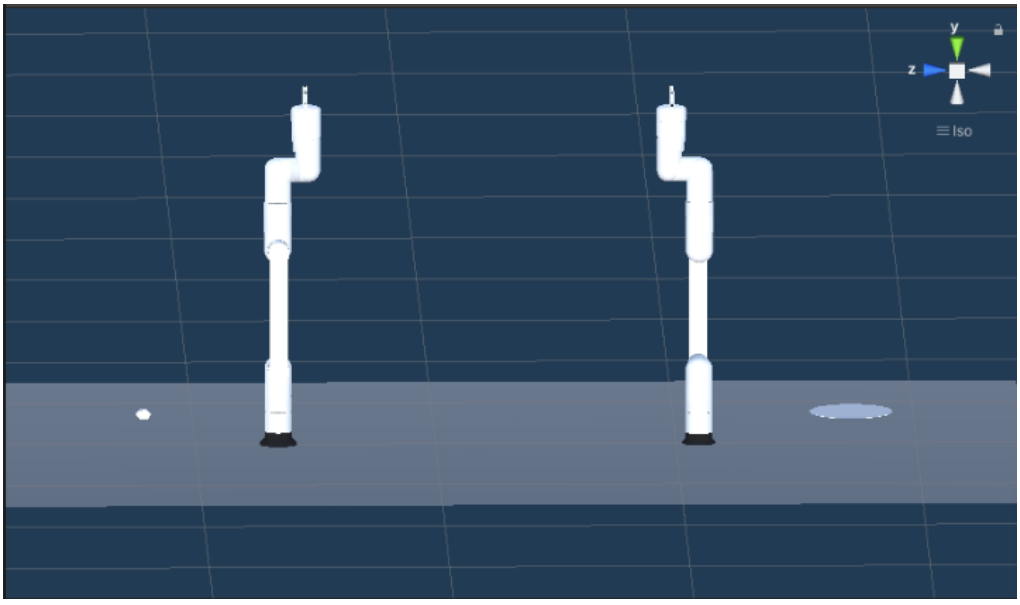

Figure 46 - The completed dual Kinova Robotic Arm model for the simulation.

On top of the change to the model, there exist multiple changes in the neural network setup as well as the scripts used by the agent during the simulation. The first of the changes in the scripts exist in the scripts for the end effector as well as the penalty collision. The end effector script is modified into a two end effector script which has four different collision conditions instead of the original two. These conditions are a collision with the object component, the other end effector, the goal component, and the ground. In the case of a ground collision, the end effector can give two different types of penalties; a normal ground hit penalty, and the other arm ground hit penalty, in which the episode is ended without a penalty if the other end effector collides with the ground. The other arm ground hit penalty function is necessary in order to keep the arm's agent updates in sync with one another, as if one arm resets and the other does not, then the arm that does not reset can become stuck and accumulate a large step penalty. The other types of collisions trigger different functions that give varying rewards. A collision with the object component calls the grabReward function and gives a reward of 0.5 to the first arm and no reward to the second arm. When the grabReward function is called, both arms swap modes into pass mode, in which both agents try to move the two end effectors towards one another, and the first arm grabs the object. In order to prevent the GrabReward function from triggering repeatedly due to the constant collision of the first end effector and the object, the grab mode is set as a requirement to trigger the function in the script for the agents. It should also be noted that both agents require two end effectors as parameters and that the same end effectors are assigned to both, with no swaps whatsoever. The lack of change in end effector parameters has the effect of syncing the mode swaps between the two agents, making sure that both of them are attempting the

necessary task when required. During the passing mode, the two end effectors can trigger the PassReward function, which gives both agents a reward of 0.5, passes the object to the second arm, and changes the mode to placement mode. Placement mode allows the triggering of the PlaceReward function when the second end effector collides with the goal component. The PlaceReward function gives a reward of 0.5 to the second agent and ends the overall episode for both agents if called. In the end, if all three possible end effector reward functions are called, then both agents will get a reward of one for a completely successful episode. The other collision script change, the penalty collision script, simply adopts the same ground penalty change that the end effector function takes, that both arms are reset by a ground collision and that only the agent causing the collision gets penalized.

A majority of the changes to the scripts controlling the agents occur in the robot controller agent function, which has multiple adjustments in order to accommodate two robotic arms calling on the same script as two separate agents. The first change to the robot controller agent script is the addition of several parameters. These parameters include a second list of arm axes for the second robotic arm, a second end effector parameter that is always assigned the end effector connected to the arm that places the object, a goal parameter that uses the goal component as the value, a Boolean value that is used to identify if the agent in question is controlling the first robotic arm, the arm that grabs the object and a change to the maximum number of possible steps to 10000. While quite a few of these parameters, the second list of axes in particular, are not necessary in order for each agent to control their robotic arm, they are required for use as inputs to the neural networks. As such, because both agents use the axes from both arms as parameter

values, the process of changing the axes values of each robotic arm is dependent on whether the agent has the first arm Boolean set to true or not, with the first arm agent causing the first list of axes to be changed and the second robotic arm causing the second list of axes to be changed at each action. The penalties and rewards received by the agents for moving closer or further away to the object of interest are also changed, with only specific arms getting penalties or rewards during specific modes. In particular, only the first agent is rewarded or penalized for movements during the grabbing phase, both agents are rewarded and penalized during the passing phase, and only the second agent is rewarded during the placement phase. Similar to the rewards and penalties given for movements towards and away from the desired object, the step penalty induced on each agent is changed with the mode: the grabbing mode causes only the first arm's agent to receive a step penalty, the passing mode causes both arm's agents to receive a step penalty, and the placement mode causes the second arm's agent to receive a step penalty. Finally, each of the three modes has different ways of measuring the distance used for rewards and penalties, with the distance utilized corresponding to the two components that collide to cause the reward function for the mode in question.

The change in the parameters used as well as the actions taken by the agent for the robotic arm are not the only changes present in the robot controller agent script. Another change is the inclusion of a new function called UpdateGoal. UpdateGoal randomizes the position of the goal component at each episode, similar to how the object component is randomized at each new episode for the solo robotic arm simulation. On top of the new UpdateGoal function, the reward functions are divided into three different reward functions. The first two divisions of the reward function, the GrabReward and

PassReward functions, no longer end the episode upon being called. Instead, the GrabReward and PassReward functions changed the mode to passing mode and placement mode respectively. Additionally, the two aforementioned reward functions added only part of the total completion to the accuracy metric, with the GrabReward adding 0.25 to accuracy, and PassReward adding 0.5 to accuracy due to the higher importance on cooperation between the two robotic arms. Similarly, the steps to goal curve only records one fourth of the steps taken in the episode for the metric if only GrabReward is called and records three fourths of the steps taken in the episode for the metric if the PassReward function is called, once again emphasizing the focus on cooperation between the two arms. In the case of the prior two scenarios, the remaining portion, three fourths for GrabReward and one fourth for PassReward, is multiplied by the maximum number of possible steps and is added to the steps taken in the episode. The final reward function, PlaceReward, remains quite similar to the reward function for the solo robotic arm simulation. PlaceReward ends the episode upon being called, resetting both arms. Furthermore, PlaceReward also changes the success rate of the individual episode to one and causes all steps taken in the episode to be counted towards the step to goal curve.

Despite the exorbitant amount of setup required for the dual arm simulation in comparison to the solo arm simulation, a fully completed training run of the agents in the dual arm simulation was never completed. Multiple factors can be attributed to the inability to complete the dual arm simulation's training attempt. These factors include the already low accuracy of the solo robotic arm simulation, which mirrors the first stage of the dual arm simulation, combined with the lower rewards for reaching the object. The

combination of the two aforementioned aspects lowers the accuracy even further to a minimum. Additionally, the dual robotic arm simulation requires a much larger training time and network in order for the robotic arm to complete the additional task of passing the object to the other arm without collision. The larger total training time and network size required are unable to be fully run on the available computer equipment without running out of memory, which immediately crashes the run.

VI: CONCLUSION

In this paper, the use of Python in manipulating the Kinova Gen 3 Lite was explored. In particular, the use of Python in direct manipulation of the robotic arm, as well as the use of Python in training an AI to control the robotic arm in a Unity simulation was covered. The direct Python manipulation was simple to create and easy to use. Both a manual and automatic version of a program were successfully generated to grab an object and move it to another location, as is seen in figure 17. While the direct manipulation of the robotic arm using Python programming and a set location of the object was successful, the AI control of the Kinova Gen 3 Lite in a Unity simulation had a multitude of difficulties that prevented proper evaluation of the agent. These difficulties ranged from a lack of prior AI for the Kinova Gen 3 Lite to the uncertain environmental setup conditions for the simulation. In particular, the randomization of the position of the object to be grabbed as well as the lack of fine tuning most likely caused issues for the agent. Therefore, due to the lack of prior AI for use in comparison, this study explores the effect of the various network parameters on the end performance of a reinforcement learning neural network based agent. Through evaluation of the agent's performance with a variety of different network parameters, it was concluded that the step penalty and robotic arm learning rate have the largest impact on the end performance of the robotic arm. Specifically, it was identified that a larger learning rate increases the robotic arm's performance and that a larger step penalty decreases the robotic arm agent's performance. Similarly, the sensitivity of the parameters was identified, demonstrating that a minor

change in parameter value can easily cause the entire agent to become overfitted, greatly reducing its accuracy to a near 0%. Finally, the issue with controlling two Kinova Gen 3 Lite arms at once using one computer was identified in the study. Particularly, that both robotic arms share the same remote address, forcing one of the robotic arms to remain unconnected while the other is connected due to Python's inability to connect with external devices using the local address; the remote address is needed for Python to connect to an object. Similar to the simulations with one robotic arm, the dual robotic arm simulation had an insignificant performance with the agents being unable to cooperate in any way due to the arm grabbing the object immediately colliding with either itself or the ground after the object is grabbed. As such, the performance of a dual arm simulation was not evaluated in detail because both robotic arms could not be connected to the computer at once and due to the poor performance of the singular robotic arm simulation, which reflects the performance of only a third of the entire dual robotic arm simulation. While this paper does not successfully generate a robotic arm AI with an adequate level of competence, it does address the various parameters and setup required to make a simulation for the Kinova Gen 3 Lite.

Future works can attempt to properly fine tune the AI and simulation environment in order to create an AI with a better performance. A second possible extension of this paper is the use of a router in controlling two robotic arms at once with one computer. In addition to controlling two robotic arms at once, another feasible continuation of this research is to compare the RL agent with other conventional control methods for robotic arms. These conventional control methods include imitation learning (Losey 2020), direct user control (Baby 2017) (Bouteraa 2017), inverse kinematics (Serrezuela 2017)

(Morishita 2017), pre-defined mappings (Quere 2020), and shared autonomy (Losey 2021) (Jeon 2020) along with other control methods. All the following conventional control methods require explicit system models with a completely known environment: inverse kinematics, pre-defined mappings, and shared autonomy. Contrary to control methods that need an explicit model, reinforcement learning allows an agent to be trained through exploration of either a known or unknown environment. The combination of RL with conventional methods of control may also be explored, such as Zhan's use of reinforcement learning and imitation learning together (Zhan 2020). Additionally, an alternative composite of control methods that could be explored is the joint use of RL and inverse kinematics. Alongside the aforementioned avenues of future research, further study could include the creation of a proper AI for a dual robotic arm simulation.

BIBLIOGRAPHY

Aljalbout, Elie. “Dual-Arm Adversarial Robot Learning.” *PMLR*, PMLR, 11 Jan. 2022, https://proceedings.mlr.press/v164/aljalbout22a.html.

Alles, Marvin, and Elie Aljalbout. “Learning to Centralize Dual-Arm Assembly.” *ArXiv.org*, 6 Dec. 2021, https://arxiv.org/abs/2110.04003.

Amarjyoti, Smruti. “Deep Reinforcement Learning for Robotic Manipulation-The state of the art.” *ArXiv.org*, 31 Jan. 2017, https://arxiv.org/abs/1701.08878.

Arntz, Alexander, et al. “Machine Learning Concepts for Dual-Arm Robots within Virtual Reality.” *2021 IEEE International Conference on Artificial Intelligence and Virtual Reality (AIVR)*, 15 Nov. 2021, pp. 168–172., https://doi.org/10.1109/aivr52153.2021.00038. Accessed 14 Mar. 2022.

Baby, Ashly, et al. “Pick and Place Robotic Arm Implementation Using Arduino.” *IOSR Journal of Electrical and Electronics Engineering*, vol. 12, no. 02, 2017, pp. 38–41., https://doi.org/10.9790/1676-1202033841. Accessed 23 July 2022.

Bouteraa, Yassine, and Ismail Ben Abdallah. “A Gesture-Based Telemanipulation Control for a Robotic Arm with Biofeedback-Based Grasp.” *Industrial Robot: An International Journal*, vol. 44, no. 5, 2017, pp. 575–587., https://doi.org/10.1108/ir-12-2016-0356. Accessed 23 July 2022.

Campeau-Lecours, Alexandre, et al. “Kinova Modular Robot Arms for Service Robotics Applications: Concepts, Methodologies, Tools, and Applications.” *ResearchGate*, ResearchGate, Jan. 2019, https://www.researchgate.net/publication/330745213_Kinova_Modular_Robot_Arms_for_Service_Robotics_Applications_Concepts_Methodologies_Tools_and_Applications.

Ge, Jing-Guo. “Programming by Demonstration by Optical Tracking System for Dual Arm Robot.” *IEEE ISR 2013*, 24 Oct. 2013, pp. 1–7., https://doi.org/10.1109/isr.2013.6695708. Accessed 14 Mar. 2022.

Gu, Shixiang, et al. “Deep Reinforcement Learning for Robotic Manipulation with Asynchronous off-Policy Updates.” *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 29 May 2017 pp. 3389–3396., https://doi.org/10.1109/icra.2017.7989385. Accessed 31 May 2022.

Jeon, Hong Jun, et al. “Shared Autonomy with Learned Latent Actions.” *ArXiv.org*, 11 May 2020, https://arxiv.org/abs/2005.03210.

K, Raju. “How to Train Your Robot Arm?” Medium. XRPractices, August 9, 2020. https://medium.com/xrpractices/how-to-train-your-robot-arm-fbf5dcd807e1.

Liu, Dong, et al. “A Reinforcement Learning-Based Framework for Robot Manipulation Skill Acquisition.” *IEEE Access*, vol. 8, 9 June 2020, pp. 108429–108437., https://doi.org/10.1109/access.2020.3001130. Accessed 31 May 2022.

Liu, Luyu, et al. "A Collaborative Control Method of Dual-Arm Robots Based on Deep Reinforcement Learning." *Applied Sciences*, vol. 11, no. 4, 2021, pp. 1816.

ProQuest, https://go.openathens.net/redirector/fau.edu?url=https://www.proquest.com/scholarly-journals/collaborative-control-method-dual-arm-robots/docview/2492459881/se-2, doi:http://dx.doi.org/10.3390/app11041816.

Liu, Rongrong, et al. "Deep Reinforcement Learning for the Control of Robotic Manipulation: A Focused Mini-Review." *MDPI*, Multidisciplinary Digital Publishing Institute, 24 Jan. 2021, https://www.mdpi.com/2218-6581/10/1/22.

Losey, Dylan P., et al. "Controlling Assistive Robots with Learned Latent Actions." *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 378–384., https://doi.org/10.1109/icra40945.2020.9197197. Accessed 23 July 2022.

Losey, Dylan P., et al. "Learning Latent Actions to Control Assistive Robots." *Autonomous Robots*, vol. 46, no. 1, 4 Aug. 2021, pp. 115–147., https://doi.org/10.1007/s10514-021-10005-w. Accessed 23 July 2022.

Morishita, Takeshi, and Osamu Tojo. "Integer Inverse Kinematics for Arm Control of a Compact Autonomous Robot." *Artificial Life and Robotics*, vol. 22, no. 4, 14 July 2017, pp. 435–442., https://doi.org/10.1007/s10015-017-0379-9. Accessed 23 July 2022.

Osiński, Błażej, and Konrad Budek. "What Is Reinforcement Learning? The Complete Guide." *Deepsense.ai*, Deepsense.ai, 5 July 2018, https://deepsense.ai/what-is-reinforcement-learning-the-complete-guide/.

Popov, Ivaylo, et al. "Data-Efficient Deep Reinforcement Learning for Dexterous Manipulation." *ArXiv.org*, 10 Apr. 2017, https://arxiv.org/abs/1704.03073.

Quere, Gabriel, et al. “Shared Control Templates for Assistive Robotics.” *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 2020, pp. 1956–1962., https://doi.org/10.1109/icra40945.2020.9197041. Accessed 23 July 2022.

Serrezuela, Ruthber Rodríguez, et al. “Kinematic Modelling of a Robotic Arm Manipulator Using MATLAB.” *ARPN Journal of Engineering and Applied Sciences*, vol. 12, no. 7, Apr. 2017, pp. 2037–2045., http://www.arpnjournals.org/jeas/research_papers/rp_2017/jeas_0417_5860.pdf. Accessed 23 July 2022.

Sutton, Richard S., et al. *Reinforcement Learning: An Introduction*. Second ed., MIT Press Ltd, 2018, *IncompleteIdeas*, http://incompleteideas.net/book/the-book-2nd.html, Accessed 2 June 2022.

Suzuki, Kanata, et al. “In-Air Knotting of Rope Using Dual-Arm Robot Based on Deep Learning.” *ArXiv.org*, 29 Aug. 2021, https://arxiv.org/abs/2103.09402.

Tagliavini, Luigi, et al. "Paquitop.Arm, a Mobile Manipulator for Assessing Emerging Challenges in the COVID-19 Pandemic Scenario." *Robotics*, vol. 10, no. 3, 2021, pp. 102. *ProQuest*, https://go.openathens.net/redirector/fau.edu?url=https://www.proquest.com/schol arly-journals/paquitop-arm-mobile-manipulator-assessing/docview/2576485010/se-2, doi:http://dx.doi.org/10.3390/robotics10030102.

Wu, Yun-Hua, et al. “Reinforcement Learning in Dual-Arm Trajectory Planning for a Free-Floating Space Robot.” *Aerospace Science and Technology*, Elsevier Masson, 3 Jan. 2020, https://www.sciencedirect.com/science/article/pii/S1270963819325660.

Zhan, Albert, et al. “A Framework for Efficient Robotic Manipulation.” *ArXiv.org*, Cornell University, 14 Dec. 2020, https://arxiv.org/abs/2012.07975.

Zohour, Hamed Montazer, et al. “Kinova Gen3-Lite Manipulator Inverse Kinematics: Optimal Polynomial Solution.” *ArXiv.org*, 1 Feb. 2021, https://arxiv.org/pdf/2102.01217.pdf.