# Documentation and Difficulties for Autonomous Warehouse Robots

Yasser Al Zahed

August 2021

## 1 Introduction

ML-Agents currently in beta, is an open-source Unity plugin that enables games and simulations to serve as environments for training intelligent agents. Agents can be trained using reinforcement learning, imitation learning, neuroevolution, or other machine learning methods through a Python API

There are two main aspects in creating this demo the first being the robot and the second being the environment.

## 2 The Learning Agent

The requirements of the robot include the ability for it to move cohesively when doing translational movement about its environment, it requires that parts of the robot remain in relative orientation to one another as well as scale, and that joints have the appropriate degrees of freedom. Looking at these requirements I originally went with a children hierarchy for the robot, it allows for children to be connected to their parents and allows for gross translational movement of the robot, but problems I face was due to scaling, children are affected by the scale of their parents meaning that sizes did not remain constant when rotations were applied on the joints. Instead transform constraints were used inorder to have independent movement without warping due to differences in scaling between children and parent objects. From there constraining the movement within the range that I wanted was done by bounding the transformations on the objects.

Next the Sensors were added on the ml agent, Due to the limitations of the hardware I have Ray perception sensors were the ideal choice due to their speed and low use of memory, this allowed me to train agents using multiple simultaneous instances,

## 3 Deep Learning Model

The agent was trained with Soft Actor Critic an Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor The Main problem with

learning is sample Inefficiency or the lack of sample complexity in each environment the agent must encounter the reward a certain number of times inorder for the algorithm to begin to determine the optimal behaviour, SAC contains an experience replay buffer where previous episodes are used to train the required behaviour, this makes it very efficient in environments where there are relatively many steps without reward. Therefore, for more complicated tasks SAC is the idea choice for robotic and autonomous applications. Even with an experience buffer the training was not progressing as fast as I would have liked. So, I used a method called curriculum learning, it is not unique to ML agents toolkit, but the toolkit makes it easier to implement, the idea is to progressively train the agents on environments that get harder as it starts to understand the task more. In this particular case I started off by teaching the agent to go to the target box and randomizing the location of the box and the agent so that it can learn a generalized approach, then I added obstacles in the way that were randomized in location so it can start to learn object avoidance, after it had good results, I added the arms and increased the reward for grabbing the target container with the arm, after that I created a drop off location and rewarded the agent for going to the target, attaching to the container and dropping it off at the drop off location without hitting any obstacles in the way, this was a much more involved approach but desired behaviours were seem much earlier than the brute force method without the curriculum learning. The training was run using 8 simultaneous instances after building the environment in a headless server environment. The training was done over 80 cumulative hours and reached over 10 million steps during that time. Due to lack of hardware space the experience replay was not stored between consecutive runs, this means every time I ran the environment it needed to generate a new experience buffer to train from, this results in slower learning where it would often regress back before learning the new behaviour, for future iterations storing the replay buffer may result in faster optimization of the behaviour.

# 4 Learning Environment

Building the environment was not much more different than building a regular unity environment. Extra previsions must be taken into considerations to remove and glitches or bugs in the environment that could cause the agent to develop unwanted behaviours, this includes falling through the map, jumping over walls, and flitching through walls. Due to the randomness of the agents' behaviours in the beginning of learning these behaviours may cause the agent to learn unwanted behaviours or completely avoid the desired behaviours.