Computers & Graphics (2021)

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

# A robot arm digital twin utilising reinforcement learning

Marius Matulis[a,*], Carlo Harvey[a]

[a]DMTLab, Birmingham City University, Curzon Street, Birmingham, B4 7XG, United Kingdom

ARTICLE INFO

*Article history*:
Received January 27, 2021

*Keywords:* robot arm, reinforcement learning, artificial intelligence, digital twin

ABSTRACT

For many industry contexts, the implementation of Artificial Intelligence (AI) has contributed to what has become known as the fourth industrial revolution or "Industry 4.0" and creates an opportunity to deliver significant benefit to both businesses and their stakeholders. Robot arms are one of the most common devices utilised in manufacturing and industrial processes, used for a wide variety of automation tasks on, for example, a factory floor but the effective use of these devices requires AI to be appropriately trained. One approach to support AI training of these devices is the use of a "Digital Twin". There are, however, a number of challenges that exist within this domain, in particular, success depends upon the ability to collect data of what are considered as observations within the environment and the application of one or many trained AI policies to the task that is to be completed. This project presents a case-study of creating and training a Robot Arm Digital Twin as an approach for AI training in a virtual space and applying this simulation learning within physical space. A virtual space, created using Unity (a contemporary Game Engine), incorporating a virtual robot arm was linked to a physical space, being a 3D printed replica of the virtual space and robot arm. These linked environments were applied to solve a task and provide training for an AI model. The contribution of this work is to provide guidance on training protocols for a digital twin together with details of the necessary architecture to support effective simulation in a virtual space through the use of Tensorflow and hyperparameter tuning. It provides an approach to addressing the mapping of learning in the virtual domain to the physical robot twin.



## 1. Introduction

Automation technologies are ever more prevalent as they benefit industry via facets such as reduced labour costs, operating costs and increased worker safety. In the context of Industry 4.0 and of factories of the future where digital twins and simulation technology play key roles, we need to consider more efficient methods for designing, simulating and visualising these production systems [1]. Interaction with these mechanisms is important to this agenda, both physically and virtually. Virtual Environments (VEs) need to be dynamic, immersive and suited to the needs of the industry [2].

Research is currently ongoing and focused on manufacturing processes related to "digital twins", one of the most prevalent technologies in Industry 4.0 [3]. Uhlemann *et. al.* indicate that the digital twin can share data, in real-time or offline, with the physical production system. Digital twins are used to run simulations and can act as process-monitoring systems that enable users or detection subsystems to identify candidate incidents, calculate performance metrics, provide fault tolerance and re-

*Corresponding author: Tel.: +0-000-000-0000; fax: +0-000-000-0000;
*e-mail:* marius.matulis@mail.bcu.ac.uk (Corresponding Author), carlo.harvey@bcu.ac.uk (Co-author)

dundancy, and ultimately to optimise the physical system [4].

Currently, companies such as car manufacturers are planning, installing and programming robots to complete tasks on the factory floor [5]. This is costly and it is not always certain that an optimal solution is deployed [6]. Furthermore, if any or multiple changes are actioned on the production line, the size of this impact on production efficiency is often not factored into new layouts. It is possible that more efficient new layouts exist. Robot arms, in the context of manufacturing, must not only find the most efficient solution to a given task, but also be able to adapt in real-time to changing situations.

Robot arms are typically programmed to complete a given task deterministically. For a fully autonomous factory, devices such as robot arms, Automated Guided Vehicles (AGV) or even delivery ordering systems offer benefits from being integrated into one interconnected system and from being controlled by AI. These benefits range from increased productivity and output to improved return on investment for hardware. Companies such as Microsoft, Amazon and Google are providing enabling platforms, frameworks and tools for engineers to train AI. Game engines such as Unity and Unreal Engine 4 not only help engineers better understand and evaluate trained data but can also simulate a VE in the training phase, providing greater resilience to change in manufacturing processes. This allows for training multiple AIs simultaneously and thus increasing the stability of training as well as reducing training time. It is possible to simulate states that a standard deterministic process would not know how to overcome. Through exploring and observing many state-action interaction pairs, it is possible via Reinforcement Learning to have resilience to situations that would typically fail.

Training a neural network can be achieved via different techniques and these techniques have parameters that can be tuned to better succeed in the applied domain. In this project, as proof of concept, a physical robot arm has been trained in a VE whilst simulating real world observation data. Creating a virtual training environment with the game engine Unity allowed for simultaneous training of multiple robot arm instances sharing their experience as one brain. This is a step towards fully autonomous industries; robots can learn and find the most efficient and fault tolerant way to accomplish the given task.

This paper presents a simulation and communication architecture between virtual agents and physical representations of these virtual agents. We train a robot arm virtually to succeed in completing a task in interacting with an environment which is continually changing. No data stack or labelled input related to the industrial robotics research field is gathered. Furthermore, this project is seeking to prove that a trained robot can complete a given task, even if it is currently in a state which it hasn't been in before. Reinforcement learning rewards determine the best action for a highest future reward given a specific state. The trained policy experience is shared between agents. Furthermore, environment observations can be numerical (force, velocity, location, distance, etc.) and/or visual observations by the robot arm. With correct observation inputs, an AI governed by a well-generalised training phase, can accomplish the set task from any state.

The contributions of this work are as follows:

- A framework for training a digital twin in a contemporary game engine using the reinforcement learning paradigm;
- Presentation of the relationships that exist between virtual and physical portions of a digital twin and how these were approached via our case study;
- Suggested heuristics for mapping of virtual learning onto physical hardware to complete the digital twin loop.

## 2. Related Work

Considerable research has been conducted into generating digital twins for a variety of uses [7, 8, 9, 3]. There exists a common interest in the community in using digital twins to manage, mediate and learn from plausible and semantic simulations in VEs. Comparative to the work that has been conducted in manufacturing processes in physical space, less research exists in the nascent field of virtual simulation learning and the mapping of this knowledge to the real world. A variety of techniques exist that have been applied into these areas from the field of AI. This Section aims to summarise the key related work to the project. Typically, machine learning is divided into three major categories: supervised learning, unsupervised learning and reinforcement learning.

Supervised machine learning depends on data pairs of input vector and output value, also known as the supervisory signal. Moreover, the higher in quality and quantity data pairs that are fed into a training module, the better results machine learning can achieve. The main difference between supervised and unsupervised machine learning is that data for training is not labelled; the output could be unknown before training and any pattern may not be apparent to a human observer.

Reinforcement machine learning does not require any data stack of input-output pairs. Instead, AI receives a reward for an action taken in an environmental state. This training process intensely relies on a reward system design and input data, or in other words - observation of the environment. Via the reward system, it is possible to punish an agent too much for an action which leads to a decision to stop exploring the environment, even though new states can yield reward. Equally, not having enough sensory information via observations will inhibit training and too much observation greatly increases training time.

Mnih *et. al.* demonstrated the use of a variant of Q-learning, a model-free algorithm, to learn control policies from raw pixel input via convolutional neural networks (CNNs), to output a value learning. This was used to learn to play seven Atari 2600 games, demonstrating better performance than the state-of-the-art techniques in six games and also outperforming a human expert in three of the games [10]. Many algorithms have been demonstrated to show better performance for policy determination in particular scenarios [11, 12, 13]. More lately Cao *et. al.* demonstrated a technique called Hierarchical Critics that outperformed Proximal Policy Optimisation in a proof-of-concept [14]. This gives credence to consider multiple critics at a variety of levels for reinforcement learning.

A lack of digital connectivity exists between machines in the domain of manufacturing. A low-cost system of instrumentation, sensors, and cloud technologies were proposed by Barbosa *et. al.* in order to monitor manufacturing processes of a robot arm [15]. This study looked at digitising a conventional robot arm and was showcased in a drilling process for a selection of aircraft materials to demonstrate this interface. Flexible and re-configurable assembly lines are considered by Kousi *et. al.*, using virtual worlds in combination with sensor data to dynamically update the digital twin based on real-time data coming from a physical equivalent [16]. This demonstrated the possibility of assembly adaptation based on a digital twin. Havard *et. al.* demonstrate a functional architecture meshing the synergy of digital twins with virtual reality approaches for work space assessments to be carried out [1].

The premise of digital twins and using reinforcement learning offers many benefits: such as, reduced time and money spent training in physical space, or risk of damage to an expensive physical test-bed. This area is quite nascent and stems from the field of Industry 4.0. Verner *et. al.* develop a system of a reinforcement learning scenario where a humanoid robot learns the protocols necessary to lift a weight of unknown mass via exploring state space. To expedite the physical training process, the trials are conducted in virtual space by simulations in a digital twin. Parameters found by simulation learning, are mapped to the physical robot [7]. Hassel *et. al.* demonstrate a line-following robot being trained in virtual space under the paradigm of a digital twin [9]. Liu *et. al.* also show an approach using deep reinforcement learning in the context of a digital twin to control human-like arms on a robot, using human collected joint data to mitigate spare reward problems with deep reinforcement learning [8].

## 3. Methodology

This Section introduces the overarching methodology and workflow of the presented framework, this includes the training procedures, experimental setup and evaluation methods. The methodology of the digital twin robot arm is functionally overviewed in Figure 1.

We utilise the game engine Unity, which allows using a machine learning (ML) toolkit to train agent(s) to interact with the virtual environment called ml-agents [17]. Multiple agents can be trained simultaneously sharing the same trained data at high speeds, cutting down training time. This empowers the ML community towards simulation learning. It is possible to simulate physical observations in virtual space, abstract away from real-time learning, and develop reward systems using the game engine in a virtual domain. The provided Academy tool in Unity ml-agents does not compute or execute neural network training, it is only tracking and observing the environment, this then sends all data to TensorFlow via an external communication system. Unity ml-agents contains three main components: Agent(s), Brain, and Academy. The Agent component is used to define the agent in the scene, in our case the robot arm. Overriding the Agent allows for collecting observations from the environment and subsequently updating the agent based actions sent from the attached Brain. The Brain element contains the definition of the observations and action spaces. The Brain element makes the decisions for all associated Agents. We use the external Python API (External Brain) to control action decisions for the attached Agents.

The combination of using the Unity game engine and Google's machine learning tool, TensorFlow, allows implementing imitation learning, behavioural cloning (BC), generative adversarial imitation learning (GAIL) and/or curriculum learning. All these tools and options enable broader use cases for industries to adjust learning policies for different tasks. Unity ml-agents also supports many simultaneous agents for simulation.

### 3.1. The Physical Robot Arm Twin

Arm components were 3D printed with polylactic acid (PLA) filament with a fill-rate of 20%. The robot arm makes use of NEMA23, NEMA17 and NEMA14 stepper motors. Each motor's power is increased to 24v, and appropriate amps for arm sections from 1.5amp to 3.5amp are set, enabling control of the robot without losing power. Increased power requires continual temperature checks. Overheating can lead to skipping steps, which means the robot can go out of synchronisation with the AI. The stepper motors are controlled via Arduino boards with a custom external micro-step driver. We an RGB-D camera to provide observations of the environment state. The Kinect v2 camera has an API for Unity, unfortunately, this camera weighs over 1.5 kilogram, and it has a minimum distance of around 28 centimetres from the camera to start detecting the depth of pixels. This makes the camera not suitable to affix onto the robot's end affector. Stationary camera placement is not suitable either, as robot movements could potentially cover the view creating blind spots in the operating domain. The camera was fixed above the principal robot section with a slight angle as per Figure 3. The robot arm main controller runs the Marlin open source software on an Arduino board with a Ramps1.4 shield. This hardware and software were originally developed to control a 3D printer and required modifying code to adapt the movements for the robot arm.

### 3.2. Reinforcement Learning

In this RL framework, the agent (in this case the robot arm) interacts with the external VE over time steps, $t$. The set of all states, $s$ and the set of all actions, $a$, govern the interaction loop in the VE. At every time step $t$ the agent has a current state, $s_t$ and the agent observes information from the environment, this observation at the time step is denoted $O_t$. The set of Transitions, $T \subseteq s \times a \times s$, is the result of taking an action in a state and an agent resulting in a new state. An action is taken by the agent, $a_t$ and feedback is provided to the agent from the reward function, $R_t : S \times A \rightarrow [-1, 1]$. Subsequently, the agent moves to a new state, $s_{t+1}$ and the reward ($R_{t+1}$) for this given state transition ($s_t, a_t, s_{t+1}$), is evaluated. This results in a Markov reward process tuple $< S, A, P, R, \gamma >$ where $S$ is a finite set of states, $A$ is a finite set of actions and $\gamma \in [0, 1]$ is a discount factor to prioritise short-term rewards.

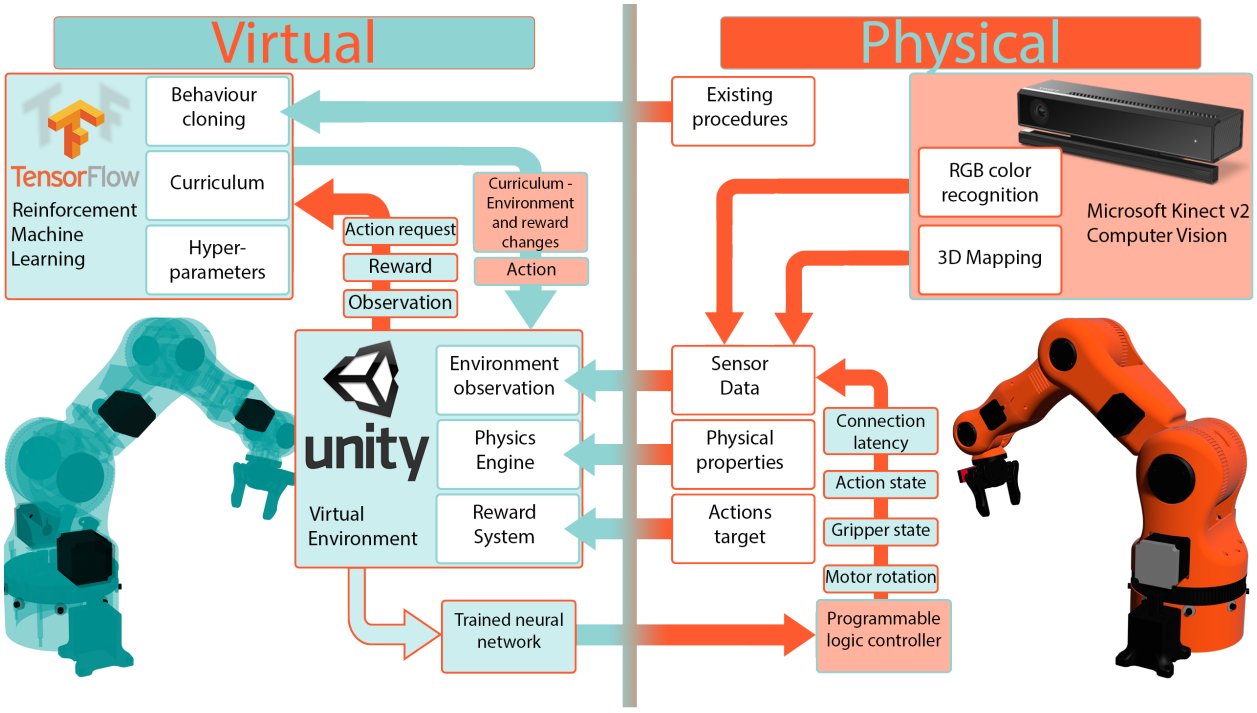


**Fig. 1. Methodology Overview: Demonstrating the framework for interaction layers between physical and virtual equivalencies in this application.**

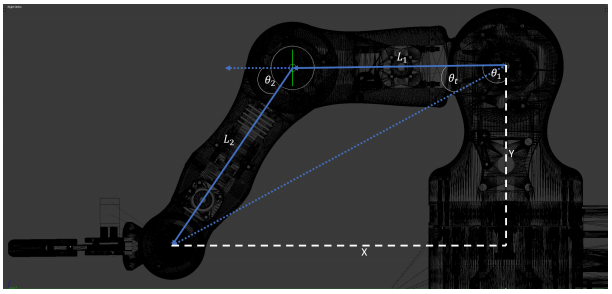

**Fig. 2. CAD representation of virtual robot arm overlaid with angles to solve for each arm component for a target position (x,y).**

Formally, $P$ is the state transition probability matrix, this provides an interface to represent the probabilities of transition for an agent $a$, from all states, $s$ to future states $s'$:

$$P^{a}_{ss'} = \mathbb{P}[S_{t+1} = s' | S_t = s, A_t = a] \quad (1)$$

$r$ is the reward function that encodes the expected reward after a transition from $P$:

$$r^{a}_{s} = \mathbb{E}[r_{t+1} | S_t = s, A_t = a] \quad (2)$$

The evaluated reward for a given time step is specified as the sum of all future discounted rewards:

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \quad (3)$$

The state value function of the reward is the expected reward evaluation for a given state, $s$ and exploration policy, $\pi(a|s) = \mathbb{P}[A_t = a | S_t = s]$:

$$V_{\pi}(S) = \mathbb{E}[R_t | S_t = s, \pi] \quad (4)$$

For a state $S$ and time step $t$, this value function $V_{\pi}$ tells the agent the expected sum of future rewards for a given policy $\pi$. This allows the agent to choose the right action that maximises the sum of rewards. The agent has autonomy of choice in determining the action from the last state visited. The ultimate goal in this process is for the agent to collect maximum reward possible in the minimum number of time steps. In general we need to specify an *exploration policy* for action selection in simulation space. We need a *reward function* to determine the feedback an agent will receive for each action. A *learning rule* allows adaptation of the policy for action selection to adapt based upon the feedback and reward received. Finally, a *discount factor* allows value of short term reward over long-term reward, in adherence with accomplishing maximal reward in minimal time steps.

### 3.3. Sensors and Experiment Setup

Unity ml-agents toolkit provides an option to detect objects in the training phase using ray casts. For our experimental setup, around the robot arm, we setup six platforms which are spread equally within reaching distance of the arm grabber and end affector. White coloured cubes are used for placing other cubes upon. The red and green cubes are the main target for a robot to find, grab, move and release onto another unoccupied post. This experimental setup is shown in Figure 4. The task the simulation will learn to complete is to be able to pick up a coloured (non-white) cube and place it onto another unoccupied pillar.

Each joint of the physical robot arm has limits of what angle it can turn; either it is a design limitation, maximum belt length or cable reaching distance limit. These limits must be captured and implemented to the virtual environment with appropriate treatment of the reward function if the virtual robot arm tries to

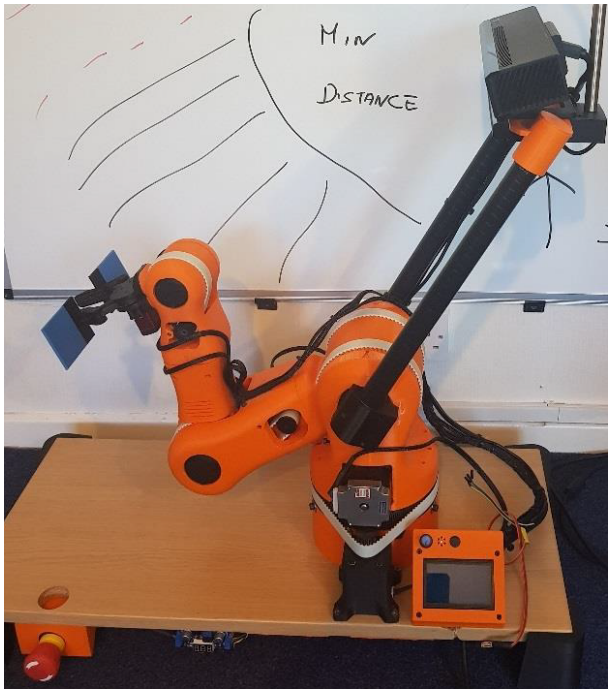


**Fig. 3. Showing the 3D printed physical 6DOF robot arm, with stepper motors controlling the joint movement, a gripper to pick objects up and a Kinect for a sensory system, to be able to perceive the environment.**

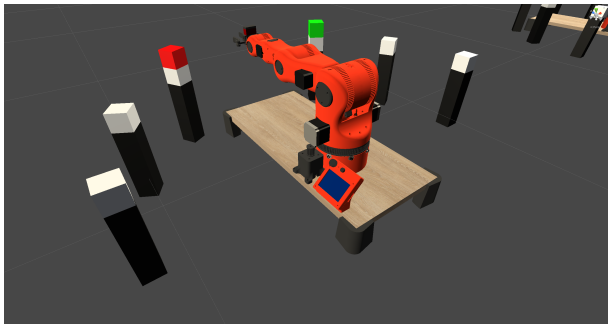

**Fig. 4. Overview of the experimental setup process showing the task at hand to solve for the robot arm.**

take an action resulting in rotating a joint outside its functional limits. We do not use inverse kinematics but instead allow machine learning to learn control of the axial joint rotations of each sub-arm of the robot. For the physical robot the law of cosine can be used to compute how many degrees each joint needs to rotate for a given target position (X,Y). For the layout shown in Figure 2, this derivation is provided:

$$\theta_1 = \texttt{acos}\frac{L_1^2 + X^2 + Y^2 - L_2^2}{2L_1\sqrt{X^2+Y^2}} + \theta_t \tag{5}$$

$$\theta_2 = \texttt{acos}\frac{X^2 + Y^2 - L1^2 - L2^2}{2L_1L_2} \tag{6}$$

The physical robot arm has a RGB-D camera attached, but this image and pixel distance data is not be fed into the neural training network. Many inputs such as these can have a negative effect on training time, though it may be possible to still train successfully. For agent sensing, we use the ml-agents 3D ray perception sensor. The ray perception setup for the virtual half our this robot arm digital twin can be observed in Figure 5. These ray bundles are used to sense the VE in which the AI is being trained. This technique could be classified as an encoder, where data is simplified, and noise is removed before feeding into the neural network. This is analogous to the approach car companies such as Tesla are using for self-driving car AI.

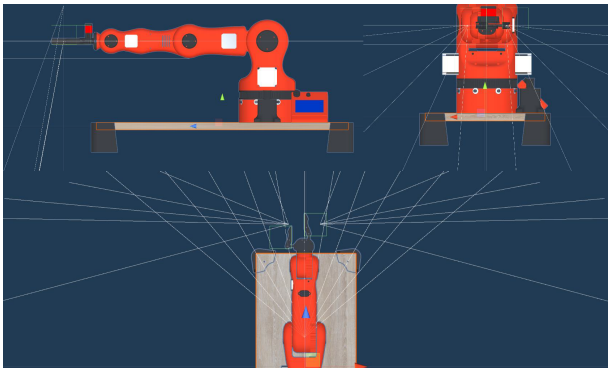

**Fig. 5. The ml-agents 3D ray perception sensors attached to the virtual digital twin robot arm. Side, front and top view.**

This sensor tool has an option to set the number of rays and angle for each direction to detect any object matching a list. These ray perception sensors are used as the vector of virtual observations that feed our neural network as inputs. The neural network architecture that governs the vector of observations and ultimately dictates the action to be selected by the agent is shown in Figure 6.

We use the Proximal Policy Optimisation strategy to train our policy [13]. A batch size of 128 is chosen with a learning rate of 1.0e-4, linear learning rate schedule and `maxSteps` is set to 1.0e8. Hyperparameters for training this network are shown in Figure 7.

### *3.3.1. Curriculum Learning*

The training plan is an important step for reinforcement learning in order to train efficiently in complex environments. The AI can use a curriculum plan to reduce the difficulty of initial training in an environment and improve convergence in reinforcement learning problems [18]. In this project new curriculum levels fall into two categories; introducing a new task, action and/or new reward; or carrying on with current experiences but requires more accuracy or speed to receive the same reward. In this project, ten levels are developed to increase both success rate and speed in training the AI, shown in Table 1.

### *3.3.2. Reward Function*

For more stable training, balancing reward between negative and positive ones is good practice because if an agent receives multiple rewards, the sum of all rewards are applied for that step. If the task requires to ignore other rewards and add only one value, then a command to set a single reward rather than a sum can be used. For this project the robot arm agent receives multiple positive and negative rewards which are enabled in different levels of the curriculum agent training as per Table 2.

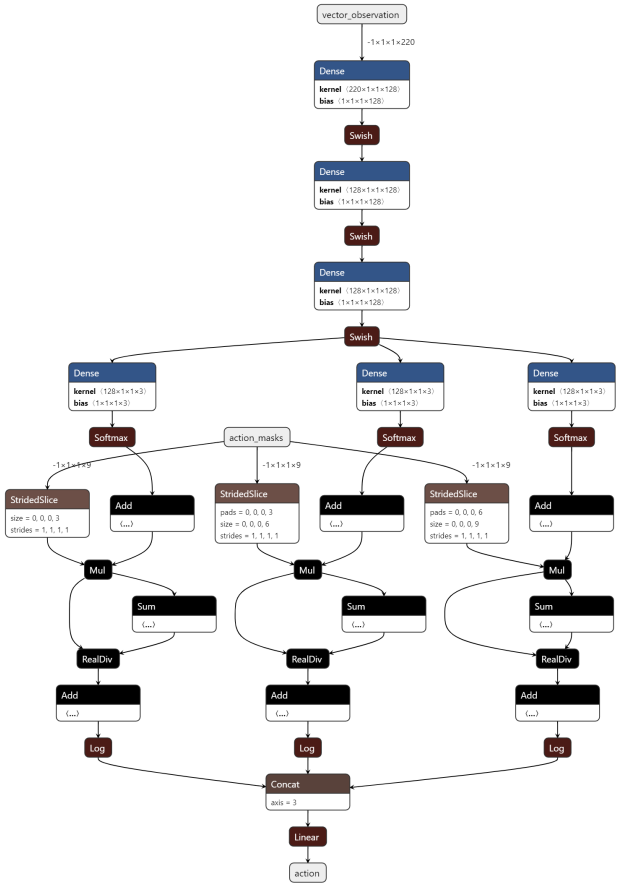


**Fig. 6. Neural Network architecture for training. Taking in a vector of observations through 3 dense layers each with swish activation functions $f(x) = x \cdot \texttt{sigmoid}(x)$. This then separates into 3 dense layers with softmax activation and a set of action masks with strided slices (3,6,9) which consider the discrete scales of expected value for a given action dictated by the policy. These condense down to an action to take, given the input vector of observations.**

```
robotArm:
    trainer: ppo
    batch_size: 128
    beta: 1.0e-3
    buffer_size: 10240
    epsilon: 0.1
    hidden_units: 128
    lambd: 0.95
    learning_rate: 1.0e-4
    learning_rate_schedule: linear
    max_steps: 1.0e8
    memory_size: 64
    normalize: false
    num_epoch: 3
    num_layers: 2
    sequence_length: 32
    summary_freq: 50000
    reward_signals:
        extrinsic:
            strength: 1.0
            gamma: 0.99
        curiosity:
            strength: 0.01
            gamma: 0.99
            encoding_size: 256
```

**Fig. 7. Hyper-parameters for the Robot Arm training process**

## 4. Results

In order to best relay experiences during the training phase of this robot arm, as this is a case study, it is important that every step that was tried is extolled in order to better explain the intuition in this process. Before converging on the approach ultimately used, and presented more completely in this manuscript, several other techniques and methods for training were used. These are presented in a stage-based approach here and are also visualised in a description of the training process online [19].

### *4.1. Initial Training*

Before reaching our final method, to start with training of the robot arm, inverse kinematics was applied to the agent. Output controls were associated with the end-effector of the robot arm. This allowed translation of the end-effector in the Cartesian axes. However, the robot was required to take in a movement from the candidate network output, then determine validity on the joint rotations to establish whether this is a viable movement. If the movement is viable then it is applied - constantly performing this check was causing training speed to be slowed significantly. A large number of sensors were attached to the robot arm end-effector with the intuition that this would best allow it to receive information on the environment states for its observations. This method failed to train in a reasonable time, largely due to the large state space to explore and increased number of observations required. In a second iteration, the number of sensors attached to the end-effector was minimised so as to only detect obstacles in the path and with this information generate a Quaternion that observed the rotation from the robot base to the target location when it was sensed. It was determined at this stage that solving for inverse kinematic movement validity, coupled with the end-effector being moved in Cartesian space resulted in unrealistic, jerky, robot arm movement and motion.

Following this we abstracted away from inverse kinematics and gave joint rotation control directly to the agent to learn. This was done in conjunction with another policy type of Soft-Actor Critic (SAC), which has maximum entropy, and it can learn from any previous experience. Benefiting from many desirable properties of reinforcement learning algorithms such as Sample Efficiency, Non-sensitive hyper-parameters and Off-Policy Learning, SAC was a natural choice as it is possible to benefit from already collected real-world training examples to shape the reward function when prototyping in a new task. The intention here was to use Behavioural Cloning to try to expedite training via feeding examples to the simulation. The curriculum levels were not present for this as an attempt was made to teach the arm by implicit example of task completion. However,

**Table 1. The ten curriculum levels, used to structure the training process to guide interim strategies determined to be converging to the ideal trained agent. Tolerance indicated is the threshold at which a reward will be given if the task is performed within a certain euclidean distance of being ultimately correct (in m). Within the Level column, a red colour introduces a new task, action and/or new reward; and a blue colour carries on with current experiences but requires more accuracy or speed to receive the same reward.**

| Level | Curriculum Adaptation |
|---|---|
| 1 | collision detection enabled - tolerance: 0.065 |
| 2 | tolerance: 0.04 |
| 3 | tolerance: 0.02 |
| 4 | tolerance: 0 |
| 5 | close gripper required for reward |
| 6 | carry cube to another free post for reward - tolerance: 0.4 |
| 7 | tolerance: 0.2 |
| 8 | tolerance: 0.1 |
| 9 | tolerance: 0 |
| 10 | open gripper to receive reward |

training failed to complete via this approach, our understanding and intuition of this is that still more examples were required for the agent to learn to mimic observed behaviour. We proceeded without this ablation and considered the PPO policy presented in Section 3 alone.

### *4.2. Training Results*

Unity allows for the time scale of the simulation to be accelerated, which can cause some issues from tunneling effects. As a result, hyper-parameters were customised for a slow-pace, stable training as illustrated in Figure 7. This took 30 hours and over 40 million steps to complete all curriculum levels. A dissection of the curricula levels and cumulative reward is shown in Figure 8. This graph illustrates the necessity of the curricula in expediting the training process for this robot arm use case. Without it, significantly more time would have elapsed before the agent had accrued some reward in the training process and started to adapt to attempt to maximise that future reward. The graph illustrates, for example, when the curriculum increments for the first time, cumulative reward deteriorates from around 55 per step to less than 10. Another significant curriculum step, can be seen in Level 5 where the gripper is required to be closed for a reward to be given and the agent, for a long period, accrues negative reward. Level 6 sees the agent training for that largest period of time in steps as once it correctly grips an object from Level 5, it now needs to understand to move the picked up object and place it onto another post.

The PPO policy achieved good results in the task described in Section 3 and succeeded in training, the trained agent can be seen here [19]. The trained neural network can be evaluated for a specific state in the virtual part of the digital twin and output codes for the physical part of the twin to correctly perform actions that solves the task. This demonstrates our contribution, showing an applied machine learning implementation within the field of robotics, using virtual environments to simulate the training phase of a robot's task set.

**Table 2. Rewards and descriptions per curriculum levels**

| Reward: Curriculum | Description |
|---|---|
| -1.f/`maxStep`: all | Reward added to incentivise agent exploration. |
| -0.05f: all | Added to disincentivise the robot to exceed any joint limits |
| -0.05f: all | Disincentivise actions to open gripper when it is open and vice versa |
| -1.0f: $\geq 1$ | Ends an episode if the arm collides with an object |
| +1.0f: $< 5$ | For the gripper to reach the object |
| +1.0f: $\geq 5$ | For grabbing the cube within tolerance threshold |
| -0.1f: $\geq 5$ | Attempt to grab cube, but end affector too far |
| +1.0f: $\geq 6$ & $\leq 9$ | Successfully moved a cube to another free post, within tolerances |
| -1.0f: $\geq 8$ | Ends an episode if a cube collides with a post to encourage lifting the end affector |
| +1.0f: 10 | Cube grabbed, moved to another location and released with no mistakes |
| -1.0f: 10 | Ends an episode if the cube is released at an incorrect location |

### *4.3. Mapping Network Outputs to the Physical Twin*

The physical robot arm joints have markers to determine their starting position before power up. This position in the robots' controller is registered as the home location for each joint. From this home position the robot arm must move to the ready position which matches virtual robot arm position. If the physical robot reaches this position, the procedure can be stopped and additional training performed before continuing with the twinned task completion. Following this setup stage, the trained task is executed and commands are sent to the robots' controller. These commands are constructed in a string of g-code format which is built dependant on the robots' new position. The command send interval to the Arduino depends on the virtual and physical robot speeds. We used 0.5s intervals with a maximum rotation speed of $25°/s$. An example of a g-code: `G1 X 10.2 Y 120.54 Z 30 E1 74.51 E2 12.03 E3 0.5`, where each element controls an axial rotation of the robot, except for `E3` which controls the gripper on a continuum [0-1]. The training of the virtual agent and also the trained agent can be seen being mapped onto the physical portion of the digital twin via YouTube [19].

## 5. Discussion

The robot arm took over 30 hours to train to complete this task. When considering the possible number of robot arms inside a factory and the complexity of the task they execute, this can be an inhibiting factor to deployment into real-time scenarios. However, net trade off in efficiency over time could offset


**Fig. 8. Review of curriculum vs. reward and episode length in training.**

this initial time investment. Naturally, more compute power would significantly alleviate this concern. This robot arm was trained with only one PC, which is not designed for machine learning purposes. Furthermore, the Unity communicator is using a network port to receive observations, rewards and return actions, which leads for an option to use multiple simulation deployments to train in the virtual environment and parallelise the operations required. This also allows for improved stability through fault tolerance.

### *5.1. Industry*

The digital twin is a much wider concept than has been explored here. This project is a small part, but a step forward towards fully autonomous industries. Implementing machine learning and accounting for ingestion of new experiences year after year will improve production in both time and quality measurements. The biggest savings are likely to be realised when production lines must be setup for the manufacture of new products. Furthermore, it may be possible to learn from observing existing production lines and using this data and the behavioural cloning approach – learn, explore, test and prove new variations of production lines to achieve more efficient results. The robot arm could be trained to detect maintenance issues, or recalibrate in real-time after tolerance differences are detected within the process. Finally, AI should be implemented into a factory if AGV based production lines are in place as this can significantly benefit from an Internet of Things approach.

### *5.2. Issues*

The most challenging part in tailoring these virtual environments and training processes is that the agent tries to find a shortcut or break the virtual environment logic to receive more reward. However, virtual training allows for a user to observe progress and if need be, to make changes to eliminate any logic flaws. Intelligent design for a good reward system is vital when considering the ultimate goals of the task. Punishing reward accumulation too much will stop the agent from taking any risk and exploring new state space, creating artificial plateaus in training convergence. The contrary is also true – giving too much reward for some facet to the agent will focus agent attention in and around the reward mechanism. To succeed with training, it is important to adjust hyperparameters for the project needs and give a reasonable amount of time for training.

### *5.3. Improvements*

In this project, to simplify the machine learning process, and also because of the delay that existed between the Arduino physical twin controller and Unity communication, the action-space type is set to discrete. This action-space type states, for example, to go left or not. It doesn't state how much to move left by increasing velocity considering current velocity. This leads to small movement jerking, which is smoothing out as the robot's training continues. For more complex tasks or if the project is requiring more accurate tracking, an infrared tracking system would be an improvement on the current implementation. A VIVE active tracker attached to the objects with a minimum of two lighthouses within the environment could return not only the precise location of the object but its rotation as well, allowing for more information to be fed into the system, improving performance and potentially reducing the time taken to convergence.

## 6. Conclusion and Future Work

This manuscript outlines the creation of a digital twin, with physical and virtual equivalencies. We develop an open-source 3D printed robot arm, analogous to contemporary robot arms used in manufacturing. We use Marlin software to control the physical portion of this digital twin, this has enough functionality to build a controlling unit and communicate with the virtual training environment solution. We utilised Unity ml-agents to both build the virtual representation of the robot arm digital twin but also communicate with TensorFlow for training in a simulation learning paradigm. This combination of hardware and software was used to virtually train a robot arm using reinforcement learning to complete a given task. It can also complete this task from previously unobserved states in testing. The main contribution of this presented case study lies in the presented connectivity between the physical and virtual components of a digital twin. The framework for training and heuristics to tackle virtual-physical connections can generalise quite well to map to other applications of digital twins. This is of use to practitioners wanting to create test scenarios such as this one involving hardware sensors that guide physical actions of automatons, informed by training in virtual space. Additionally, this is coupled with guidance on training protocols in a stage-based approach to model training, where the model is then used to evaluate and guide a physical twin where the method of this communication, virtual to physical, is presented.

We will look to implement a higher accuracy tracking system for further development, this should allow for more complex tasks to be developed for training. Furthermore, more in-depth hyper-parameter adjustments and a rigorous ablation study, including concluding the initial insight into SAC, for different use cases will be started. Lastly, the next step in this research field should be collaborative communication, two robots working together as complementary but independent agents to complete one task.

## 7. Acknowledgments

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp used for this research.

## References

[1] Havard, V, Jeanne, B, Lacomblez, M, Baudry, D. Digital twin and virtual reality: a co-simulation environment for design and assessment of industrial workstations. Production & Manufacturing Research 2019;7(1):472–489. doi:10.1080/21693277.2019.1660283.

[2] Chryssolouris, G, Mavrikios, D, Papakostas, N, Mourtzis, D, Michalos, G, Georgoulias, K. Digital manufacturing: History, perspectives, and outlook. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture 2009;223(5):451–462. URL: https://doi.org/10.1243/09544054JEM1241. doi:10.1243/09544054JEM1241. arXiv:https://doi.org/10.1243/09544054JEM1241.

[3] Uhlemann, THJ, Lehmann, C, Steinhilper, R. The digital twin: Realizing the cyber-physical production system for industry 4.0. Procedia CIRP 2017;61:335 – 340. URL: http://www.sciencedirect.com/science/article/pii/S2212827116313129. doi:https://doi.org/10.1016/j.procir.2016.11.152; the 24th CIRP Conference on Life Cycle Engineering.

[4] Ojsterek, R, Buchmeister, B. Use of simulation software environments for the purpose of production optimization. In: Katalinic, B, editor. Proceedings of the 28th DAAAM International Symposium. DAAAM International. ISBN 978-3-902734-11-2; 2017, p. 0750–0758. doi:10.2507/28th.daaam.proceedings.106.

[5] Michalos, G, Makris, S, Papakostas, N, Mourtzis, D, Chryssolouris, G. Automotive assembly technologies review: challenges and outlook for a flexible and adaptive approach. CIRP Journal of Manufacturing Science and Technology 2010;2(2):81 – 91. URL: http://www.sciencedirect.com/science/article/pii/S1755581709000467. doi:https://doi.org/10.1016/j.cirpj.2009.12.001.

[6] Krüger, J, Wang, L, Verl, A, Bauernhansl, T, Carpanzano, E, Makris, S, et al. Innovative control of assembly systems and lines. CIRP Annals 2017;66(2):707 – 730. URL: http://www.sciencedirect.com/science/article/pii/S000785061730149X. doi:https://doi.org/10.1016/j.cirp.2017.05.010.

[7] Verner, I, Cuperman, D, Fang, A, Reitman, M, Romm, T, Balikin, G. Robot online learning through digital twin experiments: A weightlifting project. In: Auer, ME, Zutin, DG, editors. Online Engineering & Internet of Things. Cham: Springer International Publishing. ISBN 978-3-319-64352-6; 2018, p. 307–314.

[8] Liu, C, Gao, J, Bi, Y, Shi, X, Tian, D. A multitasking-oriented robot arm motion planning scheme based on deep reinforcement learning and twin synchro-control. Sensors (Basel, Switzerland) 2020;20(12):3515. URL: https://pubmed.ncbi.nlm.nih.gov/32575907. doi:10.3390/s20123515.

[9] Hassel, T, Hofmann, O. Reinforcement learning of robot behavior based on a digital twin. In: Marsico, MD, di Baja, GS, Fred, ALN, editors. Proceedings of the 9th International Conference on Pattern Recognition Applications and Methods, ICPRAM 2020. SCITEPRESS. ISBN 978-989-758-397-1; 2020,.

[10] Mnih, V, Kavukcuoglu, K, Silver, D, Graves, A, Antonoglou, I, Wierstra, D, et al. Playing atari with deep reinforcement learning. 2013. arXiv:1312.5602.

[11] Sutton, RS, McAllester, D, Singh, S, Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In: Proceedings of the 12th International Conference on Neural Information Processing Systems. NIPS'99; Cambridge, MA, USA: MIT Press; 1999, p. 1057–1063.

[12] Lillicrap, TP, Hunt, JJ, Pritzel, A, Heess, N, Erez, T, Tassa, Y, et al. Continuous control with deep reinforcement learning. 2015. arXiv:1509.02971.

[13] Schulman, J, Wolski, F, Dhariwal, P, Radford, A, Klimov, O. Proximal policy optimization algorithms. 2017. arXiv:1707.06347.

[14] Cao, Z, Lin, CT. Reinforcement learning from hierarchical critics. 2019. arXiv:1902.03079.

[15] Barbosa, GF, Shiki, SB, Savazzi, JO. Digitalization of a standard robot arm toward 4th industrial revolution. The International Journal of Advanced Manufacturing Technology 2019;105:2707–2720. doi:https://doi.org/10.1007/s00170-019-04523-2.

[16] Kousi, N, Gkournelos, C, Aivaliotis, S, Giannoulis, C, Michalos, G, Makris, S. Digital twin for adaptation of robots' behavior in flexible robotic assembly lines. Procedia Manufacturing 2019;28:121 – 126. URL: http://www.sciencedirect.com/science/article/pii/S2351978918313623. doi:https://doi.org/10.1016/j.promfg.2018.12.020; 7th International conference on Changeable, Agile, Reconfigurable and Virtual Production (CARV2018).

[17] Juliani, A, Berges, VP, Teng, E, Cohen, A, Harper, J, Elion, C, et al. Unity: A general platform for intelligent agents. 2018. arXiv:1809.02627.

[18] Bengio, Y, Louradour, J, Collobert, R, Weston, J. Curriculum learning. In: Proceedings of the 26th annual international conference on machine learning. 2009, p. 41–48.

[19] Matulis, M, Harvey, C. 3d printed robot arm, unity, tensorflow and kinect v2 - reinforcement machine learning. ???? URL: https://www.youtube.com/watch?v=Vkj-trZ2NRw.