TALLINN UNIVERSITY OF TECHNOLOGY
SCHOOL OF ENGINEERING
Department's title

CONSAC: A GENERAL UNREAL ENGINE FRAMEWORK FOR REINFORCEMENT LEARNING

CONSAC: ÜLDINE STIIMULÕPPE RAAMISTIK UNREAL ENGINE MÄNGUMOOTORILE

MASTER THESIS

Student: Hakan COLAKOGLU

Student code: 194262MAHM

Supervisor: Saleh Alsaleh, Engineer

Co-Supervisor: Dr. Aleksei Tepljakov, Research Scientist

Tallinn 2021

AUTHOR'S DECLARATION

Hereby I declare, that I have written this thesis independently.
No academic degree has been applied for based on this material. All works, major viewpoints and data of the other authors used in this thesis have been referenced.

"....." ..... 20…..

Author: .....
/signature /

Thesis is in accordance with terms and requirements

"....." ..... 20….

Supervisor: ….....
/signature/

Accepted for defence

"....."..... 20… .

Chairman of theses defence commission: .....
/name and signature/

Non-exclusive licence for reproduction and publication of a graduation thesis[1]

I Hakan Colakoglu

1. grant Tallinn University of Technology free licence (non-exclusive licence) for my thesis CONSAC: A General Unreal Engine Framework For Reinforcement Learning, supervised by Saleh Ragheb Saleh Alsaleh and Dr. Aleksei Tepljakov,

1.1 to be reproduced for the purposes of preservation and electronic publication of the graduation thesis, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright;

1.2 to be published via the web of Tallinn University of Technology, incl. to be entered in the digital collection of the library of Tallinn University of Technology until expiry of the term of copyright.

2. I am aware that the author also retains the rights specified in clause 1 of the non-exclusive licence.

3. I confirm that granting the non-exclusive licence does not infringe other persons' intellectual property rights, the rights arising from the Personal Data Protection Act or rights arising from other legislation.

---

18 May 2021

Department of Electrical Power Engineering and Mechatronics

---

[1] *The non-exclusive licence is not valid during the validity of access restriction indicated in the student's application for restriction on access to the graduation thesis that has been signed by the school's dean, except in case of the university's right to reproduce the thesis for preservation purposes only. If a graduation thesis is based on the joint creative activity of two or more persons and the co-author(s) has/have not granted, by the set deadline, the student defending his/her graduation thesis consent to reproduce and publish the graduation thesis in compliance with clauses 1.1 and 1.2 of the non-exclusive licence, the non-exclusive license shall not be valid for the period.*

THESIS TASK

Student: Hakan Colakoglu, 194262MAHM

Study programme, MAHM02/18Mechatornics

main speciality: Mechatronics

Supervisor(s): Saleh Ragheb Saleh Alsaleh, Engineer, 620 3201;

Dr. Aleksei Tepljakov, Research Scientist, 620 2116

Consultants:

Thesis topic:

(in English) CONSAC: A General Unreal Engine Framework For Reinforcement Learning

(in Estonian) CONSAC: Üldine Stiimulõppe Raamistik Unreal Engine Mängumootorile

Thesis main objectives:

1. Importing CAD models into the game engine
2. Creating a generic reinforcement learning format using a suitable library
3. Connecting reinforcement learning library to the game engine

Thesis tasks and time schedule:

No	Task description	Deadline
1.	Researches on current technologies and usefull tools	05/03/2021
2.	Understaing CAD models and their compatibality for the project	25/03/2021
3.	Testing different reinforcement learning libraries	15/04/2021
4.	Creating a TCP connection between the library and game engine	01/05/2021
5.	Compiling the framework into a user-friendly format	14/05/2021

Language: English **Deadline for submission of thesis:** "18" May 2021

Student: Hakan Colakoglu ..... "18" May 2021

/signature/

Supervisor: …… …….. ".....".....20….a

/signature/

Head of study programme: …… ..... ".....".....20…..a

/signature/

CONTENTS

PREFACE

This project has been dedicated to bring a versatile and an easy to use reinforcement learning solution to a very popular game engine, Unreal Engine. I always believed that, simplifying subjects will not only make things easier for us, it will also give us extra time to contribute more and more. While I am compiling this work, I would like to thank to Professor Mart TAMRE wholeheartedly for giving me the opportunity of being a part of his team when I needed it the most, so I could come up with such a project. His support, generosity in sharing his knowledge and sincere Estonian hospitality always encouraged me during my studies.

Also, my supervisor Saleh Al-Saleh has always been a great tutor, and a mindful mentor to me. Not only he gave me the idea of this framework, he also appreciated every single effort I put in this work. Another person I should be thanking is my co-supervisor Doctor Aleksei TEPLJAKOV, who has great achievements and is a great role model for an engineer like myself.

This work and having a new life in a new country gave me hard times but TalTech has been my home and a safe place thanks to Merle KUTSAR and Svetlana GROMOVA.

And last but not the least, a wonderful study partner and a companion, dear Shuang Lee; a remarkable friend and a great support Fatma ERTOY, thank you for the journey. I made it until here and I made it with you.

1. INTRODUCTION

Robotic technology and Industry 4.0 movement have brought newer industrial robots and solutions into manufacturing industry [1]. Even though the production has increased significantly, there are handicaps of those robotic systems. One of the main challenges is robots are not as flexible as humans in adapting new or dynamic environments. It is not easy to change the working area of the robot for that robots would need to be updated, re-planed or structured again.

On top of that, we want to integrate robots into human everyday lives in a way they can work with humans and learn from them, namely collaborative robots. Most of the industrial robots are pre-programmed for their tasks and interacting with a human worker or with a complex working environment is a challenging task. These challenges are even bigger if we expect from a robot to perform numerous tasks. For some tasks robots may not have flexible movement abilities as humans do. It is an open problem to program such robots with those human-like abilities, so the robots can assist humans in various tasks such as medical surgery [2] [3] and rehabilitation [4].

Due to mentioned problems, artificial intelligence methods are being used for training collaborative robots. Trained robots can be more adaptive to their changing environments or to various tasks but training a system requires considerable amount of data and the collecting training data is another challenge on its own.

1.1 Motivation

Fully autonomous robotic technology, including natural interaction, learning from and with a human, safe and flexible multi-tasking ability for challenging tasks in unstructured and dynamic environments will remain out of reach for the near future. In the predicted future factory scenarios, home and office environments, humans and robots are expected to share the same workspace and perform different tasks in a collaborative manner. Before having robots and humans in the same environment, safety must be assured and considered in every possible state which can only be done by testing and analysing both human and robot factors. In a virtual/digital environment, where collecting training data, robot training and testing were possible and easily

accessible, creating data sets and testing robots in possible or extreme environments would be much easier, safer and faster.

1.2 Structure of the thesis

This paper, in its first chapter, gives a general overview on the current needs and development of the industry. Some problems we are having and we might be having in the future are also highlighted along with the suggested solutions.

The second chapter will look into virtual robot testing applications, the term "digital twins" and its differences than classic simulation environments to emphasize the need of better digital applications. Also, collaborative robots, robot trainings and robotic simulation softwares will be evaluated by both positive and negative sides.

The third section will be dedicated to our reinforcement learning framework, our intentions on the work and our approach to the solution.

Of course, we will face problems on the way but those problems are the learning checkpoints for us. We will discuss the problems, solutions and improve our work for the future works.

At last, in the fifth chapter, we will conclude our work to have couple of words about what we have done and why we are happy with it.

2. LITERATURE REVIEW

2.1 Preparation for the research

Digital Twins: The digital twin is a comprehensive digital model of an individual product. It consists of the same properties, condition and behaviour of the real-life object through digital models and received data. The digital twin is a set of realistic models that can simulate its actual behaviour in the actual environment. And the digital twin is developed alongside its physical model and remains its virtual look and abilities throughout the entire product lifecycle [5]. Every update and changes on the physical twin should be applied to the digital twin as well. A digital twin can be considered as a collection of all digital replicas of the actual model that is linked with all data that is generated during product use. Yet, it is argued [6] that a digital twin is a collection of connection between only the relevant data and parts of the physical models. The digital models which are used as digital twins are specifically designed for their intended purpose. There are already methods which use digital twins and digital environments in order to collect data and train neural networks for a specific purpose [7] which shows us using a digital twin in a digital environment to execute different situations is a valid method. Hence, relevant data can be collected in such digital environments.

Training data collection and data sources: Data and data collecting is vital for both technological research and development, which is why data mining and big data terminologies are being commonly used nowadays. Regardless its methods, whether empiric or theoretical, there are numerous approaches to gather relevant data, but, for the purpose of this work, empiric methods are where we put our efforts in. Training data types differ for different training purposes [8]. Some examples to commonly used data types are: angle, rotation, location, speed, image, pressure… All those data can be observed from different sources, be it objects, animals, humans, plants etc. The observation can be done by different cameras, sensors, microphones; and the quality of the data depends on the quality of the sensor which can increase the price. Apart from that, needed sensor may not be presented or affordable for the time being. For instant, human motion and movement can be measured using gloves or different types of cameras or a group of cameras, namely depth cameras [9]. Currently, motion capturing or object tracking is mostly done by tracking markers by a set of cameras or placing sensors at specific places at human body and collect relevant data [10]. A student or an organization may not afford or have a space to mount such equipment. On top of it, some scenarios may be hard or harmful to create and observe in real world

(jumping from a high wall, falling from a building, rolling down from a hill, etc.). Creating sensors and scenarios in a virtual environment could provide higher flexibility for prototyping and testing purposes.

Collaborative robot: Collaborative robots (cobot) are the robots that can share the same workspace while working together or working side by side [11][12]. Many robotics manufacturers like KUKA and ABB produce cobots. Many other cobots are also available for different applications [13]. The difference between any other robot and a cobot is that cobots are designed to work and interact with humans which of course brings extra safety measurements. To guarantee safety, there are restrictions on power, speed, working range, maximum load limit, etc. (ISO-TS 15066 [14]).

Reinforcement Learning: Reinforcement learning is a way of Machine Learning which evaluates the learning environment without needing initial datasets. It is sometimes classified as an unsupervised learning method but unlike unsupervised methods, reinforcement learning does not require a set of correct behaviors to create a pattern [15].

Reinforcement learning solves problem by focusing on "what to do", "how to address the situations for next steps" and "maximize the final reward with the best result". Reinforcement systems are closed-loop problems, which is to say, a result of an attempt is used as an input for the next attempt. In reinforcement learning, the system is not told how to execute actions but actions give positive or negative feedback to system. Depending on the final feedback, system decides how to operate next time to get a better results and it tries all the possible ways. These three characteristics; the necessity of operating in a closed-loop, working without having direct instructions and the possibility of a result of a previous action may be affecting the system in the future, are differences of reinforcement learning from other methods [16].

In a practice, reinforcement learning algorithm produces an action and introduces that action to the environment through an agent. In return, it collects an observed state and rewards it to distinguish how well the system acted.

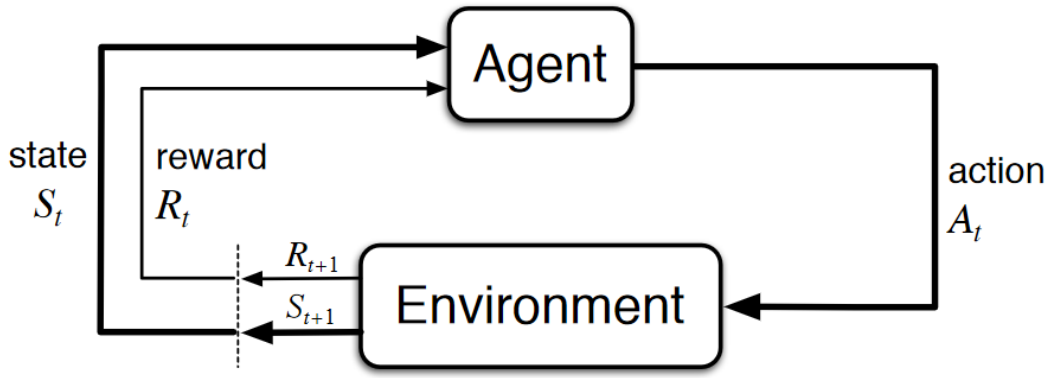

Figure 2.1 Reinforcement learning structure

Some pros of the reinforcement learning are:

1) It is not limited to a data set, therefore it can maximize the performance when positive rewarding is used (however long however short time it requires).

2) It is close to human learning so it is much easier to design a reinforcement learning model.

3) It can correct errors as the training continues.

Some cons of the reinforcement learning are:

1) Creating a logic, framework is much different than we expect. The model can find loops we cannot think of, it may not be able to distinguish actions if the reward system is not designed properly.

2) It requires a lot of data, hence long computations.

Game Engines: A game engine is a software-development tool for creating video games. Developers can create games for different platforms such as consoles, mobile devices, and computers. A game engine provides a renderer for graphics, a physics engine and collision detection along with other extra tools such as scripting, animation, artificial intelligence, threading, localization and so on.

A physics engine in a game engine is a software that provides an approximate simulation of physical laws within the software, such as rigid body dynamics, fluid dynamics… Physics engines term is used generally to describe any software system for simulating physics, such as scientific simulation.

2.2 Digital testing methods

Virtual environments, simulations, and virtual products are widely used for advertisement purposes, testing and development stages. IEEE defines "testing" as "the process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component" [17], we can see that observation and data collection is necessary in the testing stages. And that is what makes a virtual twin different than a simple digital model of a product. A virtual twin should be able to produce outputs which are close to the real life data or at least those data should be meaningful for the sake of testing or development. As producing reliable data is the expectation, it is also one of the most discussed subject about digital twins if we can rely on data produced by a digital source such as simulations [18].

Currently, there are several testing methods used for testing hardware and software. The two most known and used testing methods are white box method and black box testing method. White box testing can be used for the products which we have information about inner structure and details. This can be the source codes, libraries, hardware components, design details, etc. If we can track the output and reason it based on the internal structure, white box methods can be used. This testing is mostly valid for the product owners, producers. Besides the white box, black box testing can be used for those products we cannot interfere or inspect which generally happens when we purchase a product, and use it without having access to see what is in it and how it

works. Such products generally take an input and produce an output. We may not be able to see the logic behind it, or the evaluation of the input and the output, such as Labview blocks [19]. Therefore, we can only compare inputs and outputs which is called input-output conformance (ioco) testing [20].

2.3 Digital design methods

Apart from testing, there are different design methods that have been designed to estimate and overcome the possible defects and system errors before they occurred or the final product is produced. There are six design methods we can address for their closeness to digital twin concept; Layered, Component-Based, V-model, Model Based Development, Virtual Integration and the most similar one, Platform Based Design[21].

Those methods have two principles, horizontal and vertical processes. Layered, V Model and Model Based Development are focused on designs at different levels and processes between them. Those models show us how different stages and different components (progresses) affect one another (vertical process). In those models, output of an element can be used as an input for another element.

For Model Based design, we can give Matlab and Simulink as an example. Those softwares can be used for virtual testing or early stage prototyping. The idea here is to use pre-made models as examples and put them together to estimate a final result. Depending on that plan, a real design can be shaped. If virtual models of the products are available, then virtual integration can be used. Virtual models put together in a simulation environment before they are assembled in real life to see if any problem would occur during the installation or the connections work as intended.

Component based, on the other hand, focuses on analysis of elements at the same level (horizontal process). Executions take place at the same time and their effects on one another, or individual results are taken into account for component based design. Thus, using component based design may require numerous progress to be validated if there are many at the same processing level.

However, platform based design brings those vertical and horizontal principles together. Platform based design includes available components and customizable indicators. Available components can be put together but if the components are not available or yet to be designed, custom parts can replace those components. Those custom parts can be set to the expected behaviour (like blocks with user defined functions, inputs or outputs). If the system works properly, those custom parts can be designed later.

2.4 Requirements and benefits of digital twins

All those system design solutions and testing methods being considered, we can say that, virtual twins covers all those testing, development and tracking purposes to some extent, hence all models and methods mentioned above and many others contributes to digital twin concept directly or indirectly. Nonetheless, the word "twin" suggests that a virtual twin should be created next to it is physical pair and should be able to produce similar outputs which means a digital twin is not an early stage prototype (unlike the platform based design) but an exclusive representation of an existing model. To have a digital twin, there should be a reference product to be compared and the digital twin should be validated accordingly. Since a digital twin is a current representation of a real model, it can be used to discover new abilities and limitations of the physical models in different environments and under different conditions without having to try them in real world.

Real time/world testing is expensive due to the time needed for physical allocation and human resources. Also the subject unit may not be accessible for the time being (it can be purchased overseas and delivery may take time, system may not be running for a reason, etc.) or running tests may harm system or the testing environment. That is why digital methods reduce the cost and physical casualties. To do so, generally a new task or/and a new environment is introduced in a virtual environment. Those tasks or environments can be produced manually or automatically (mostly created randomly by a script, such as random maze or moving objects in random directions). Owing to the flexible nature of the digital twins and digital environments, both new behaviour of the twin or its interaction with a new environment can be observed. Generally, machine learning algorithms or new production lines or work spaces tried out using digital twin to see the results before taking any action in real life.

2.5 Current digital twin and cobot applications

A recent application with digital twins [22] shows how digital twins are used to see adaptation of robots' behaviour in production lines. Another study focuses on only digital twins regardless its environment [6]. More and more, from small sized production lines to bigger factories, digital transformation is taking place. As the robots replace humans or learn how to work with humans, production areas need to be re-planned and this is where digital twins come in handy. Several works show how factories, companies are transforming their facilities using cobots [22][23]. Around 30 cobot application presented in [24] shows how widely cobots taking place in our lives. Therefore, they have to be more flexible and suitable for different tasks and they should be able work with humans safely. The most common way of preparing cobots for new tasks is using teaching pendants. Cobots like ABB YuMi, can take instructions through a teach pendant. Robot programming by using the teach pendant (conventional teaching) can be tiring and time-consuming and may require significant technical experience [25]. However conventional teaching is still one of the most used method of all. Yet still, new and more practical approaches to program robots are needed. As an alternative to conventional teaching, machine learning techniques are being used. Even though artificial intelligence is a strong tool, there is not a universal solution for all robotics systems. Therefore, the most suitable solution must be applied for a specific solution.

For example, cameras, motion capturing methods and vision assistance can be used for teaching robots by demonstrating tasks and solutions [26][27]. In such scenario, robot learns from its demonstrator which is not fully autonomous. In this times, we expect robots to understand and create solutions on their own. Even better, we expect them to work as a colony on some occasions and learn from one another as they explore their environment [28]. In many works, the contribution of artificial intelligence for collision avoidance solutions [29] and workspace identification [30] can be seen. Amongst many artificial intelligence solution, reinforcement learning seems to be promising for creating adaptive robots. A study shows how to implement reinforcement learning on an ABB YuMi cobot to solve complex problems [31]. But still, there are challenges with reinforcement learning [32] such as smoothness, safety, scalability, etc.

Considering all the works mentioned above, we can point out some of the problems with cobots:

- Cobots are taught using teach pendants or by offline programming which requires time and specialists which means the user has to complete a training beforehand.

- Cobot can be taught by demonstrating which includes machine learning and machine vision. As it is robust, it teaches cobots how to operate like humans. That practice limits the potential of the cobot

- Some artificial intelligence methods are used for cobots but most of them requires data sets a clear objectives to train cobots which adds an extra "data collection" phase to the teaching progress.

2.6 Simulation environment and software

So far, we discussed the purpose of digital testing, digital design, and digital twins and tried to highlight the differences. Also, we mentioned collaborative robots, and different artificial intelligence solutions being applied on them, such as reinforcement learning. Before we conclude the literature review, it is also important to look into simulation software solutions we have nowadays.

As we mentioned above, Matlab and Simulink are used frequently for testing and simulation systems. Besides them, Gazebo, Simscape and similar applications are being used as concluded in a survey [33]. However, the price of such advance softwares and license considered, they all come at a cost [25]. Also they can be used if only the robot environment is known and modelled precisely [34]. Still, manufacturers may have their software which comes with the product, for instance, RobotStudio from ABB Robotics. However, in recent years, robot simulation techniques improved considerably, owing to advance computing and graphical animation technologies. Nowadays, game technologies started to copy industrial methods to create more realistic games and they put many effort for improving their physics engines and environmental design tools. They became way too advance, they can be used for architectural purposes to create realistic urban scenarios.

Following those game industry, virtual reality and augmented reality gained power and eventually they took their places in robotics as [35] much as they did in game industry. On top of that, it is a growing trend to use game engines and virtual reality tools for education purposes. Different than classic methods, e.g. Matlab, Solidworks, game engines are mostly free and allows users to design robots and whole environment [36] with high accuracy. Some works shows that, nowadays, people are switching to game engines for simulation purposes [37] [38] [39].

2.7 Objectives of the thesis

After considering all the studies and applications mentioned in previous sections, we can summarize digital twins and their differences as:

- Digital twins are not prototypes, but an executive digital copy of a physical model
- Digital twins cannot exist without their physical model
- Digital twins can their physical models cannot have different versions (if a model is updated, digital twins must be updated as well) or different working environments.
- Digital twins can be used for testing, and they can be modified for estimating a newer version but the modified version cannot be considered as a digital twin for having differences than its physical model.

It is also important to know the distinction of collaborative robots than any other robots:

- Collaborative robots are designed to share the same workplace with humans or work/interact with them.
- Being close to humans requires restrictions on robots' design (hardware and software). Therefore, collaborative robots should fulfil safety requirements.

On top of that, it is beneficial to highlight the advantages of game engines against classical simulation and CAD softwares:

- Game engines allow users to script without having to use additional software (such as using Matlab with Solidworks).

- Game engines allow users to design their systems/models as well as with their environments with supported physics engine features (fluids, winds, lighting, etc.).

- Game engines are easy to create virtual reality and augmented reality solutions which can be used for Human-Computer/Robot interaction applications.

- Game engines offer high resolution and rendering

All those being said, in this work;

- We will first find a way to import classic CAD files into the game engine environment. Even though most of the robot models are available in CAD format, game engines do not support CAD files directly.

- After, we will decide which reinforcement learning libraries we can use. Machine learning libraries require different dependencies and additive libraries. We will try to scale our needs down and work on minimum requirements to abstain from too many dependencies.

- Then we will start implementing decided libraries and address the challenges we might encounter.

- At the end, we will finalize this work with a generic reinforcement learning framework for Unreal Engine which then can be used for future works and researches.

During the work, 3D model of a collaborative robot ABB YuMi will be imported in Unreal Engine 4 environment to represent a digital twin. This will also show the readers how Unreal Engine handles CAD formats.

3. METHODOLOGY

3.1 Importing CAD files into Unreal Engine

The subject cobot, ABB YuMi, is selected for this work for its secure and precise technical capacities.

Figure 3.1 ABB YuMi

ABB YuMi is designed for high security to satisfy cobot requirements. A cage is not needed for ABB YuMi, which allows humans to interact and work closely. It also has a very light design and rather slow acceleration rate which significantly increases security.

YuMi also supports various communication interfaces, and vision equipment which can be used in the future for advance machine learning purposes. In this work, YuMi is selected to show how to import SolidWorks files into Unreal Engine 4. Using reinforcement learning algorithms to train such a robot is a challenging task and requires a decent understanding of machine learning concepts and methods.

Generally, game engines are compatible with certain 3D modelling softwares, however, CAD formats are not supported in many cases. Those models must be converted into

different file formats, which may cause deformations or produce errors along the way. Unreal Engine is powered by a package named “DataSmith” to convert CAD files.

Unreal Engine and similar game engines do not have any tools to align or assemble objects. It is important to import an assembled model. The ABB Yumi model we used was taken from the official website and assembled in SolidWorks. The assemble file imported into Unreal Engine by using “DataSmith” .

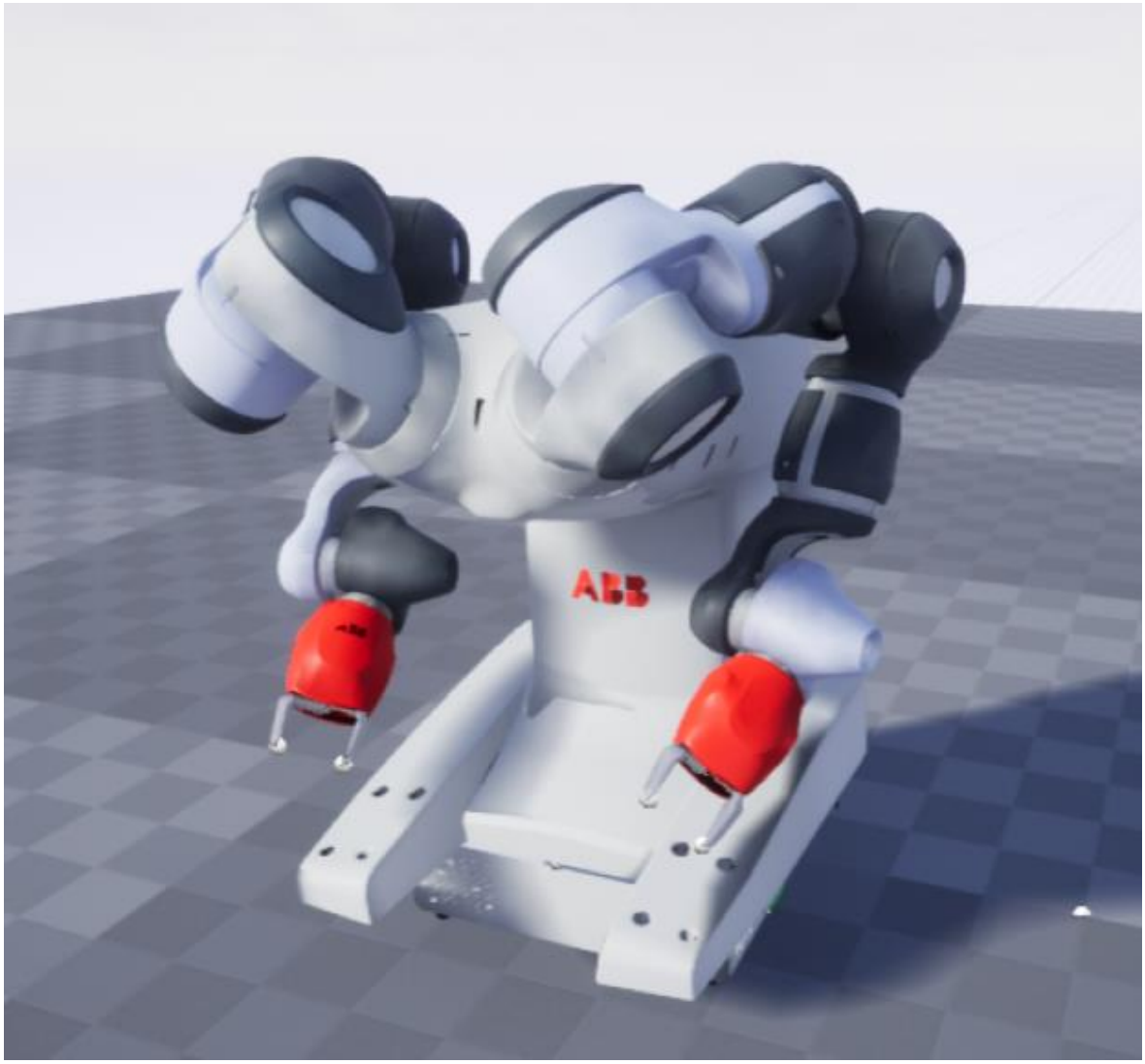

Figure 3.2 Imported YuMi model

At first glance, the model does not look too different than a classic CAD model. The conversion of the model and the coordinates of each part of it is calculated relatively to the world coordinates of the environment. The arms rotates around their joints and the axis that represents this center point of the rotation is set to X axis. In the CAD model,

the X axis is perpendicular to the joint surface but the converted model uses Unreal Engine world coordinates which does not look quite right.

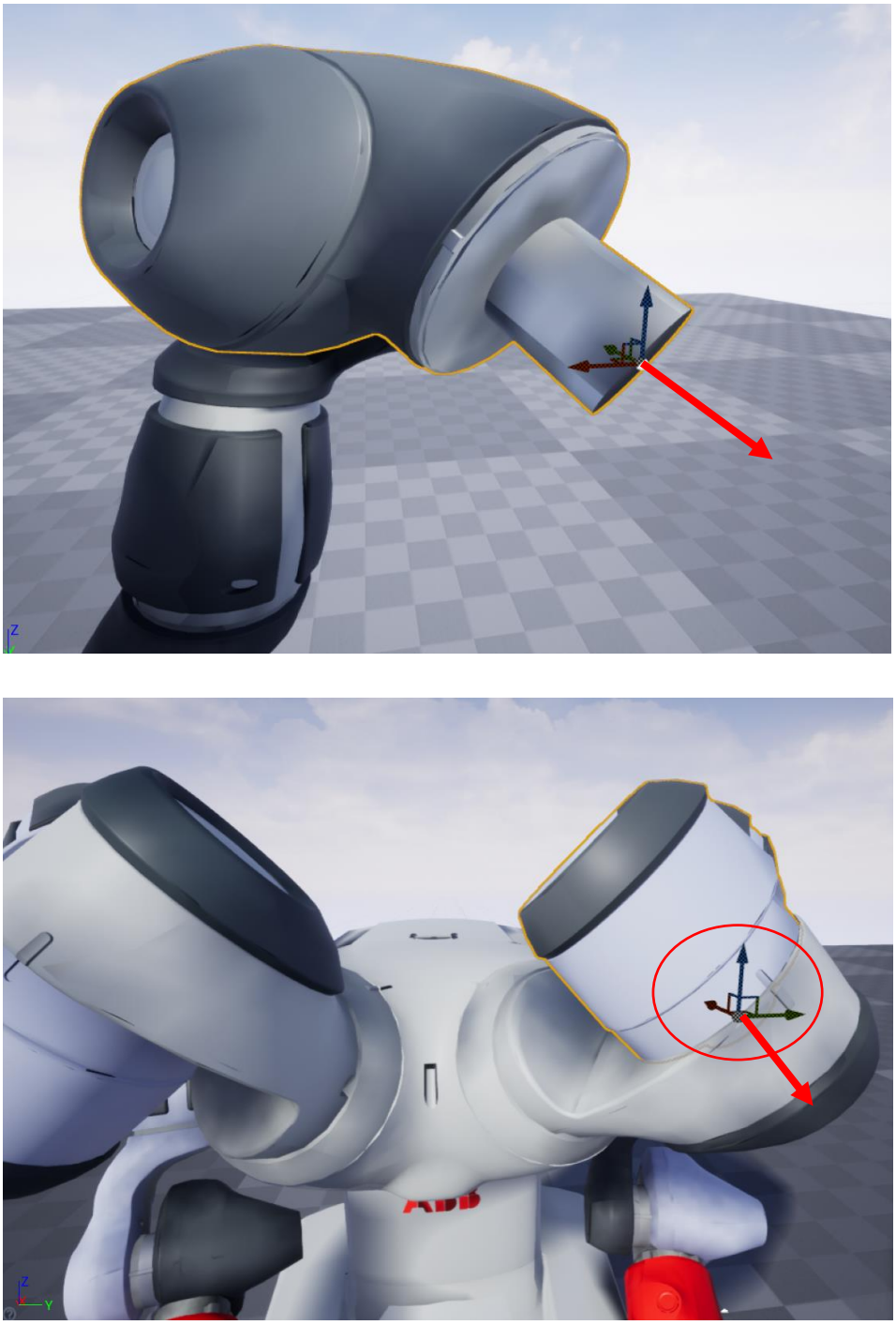

Figure 3.3 Difference between original axis (red arrow) and Unreal Engine axis

The transformation between CAD axis and Unreal Engine axis is calculated by "DataSmith" and the values can be seen in the transformation menu.

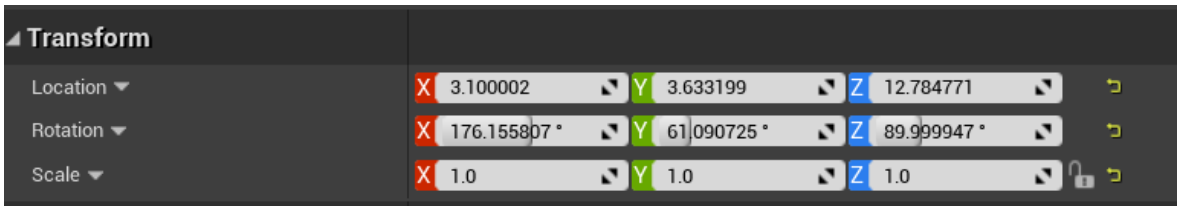

Figure 3.4 Transformation result of converted axis

Transformation is not the only thing that is effected, hierarchy between parts is also lost. Hierarchy tree must be formed from scratch.

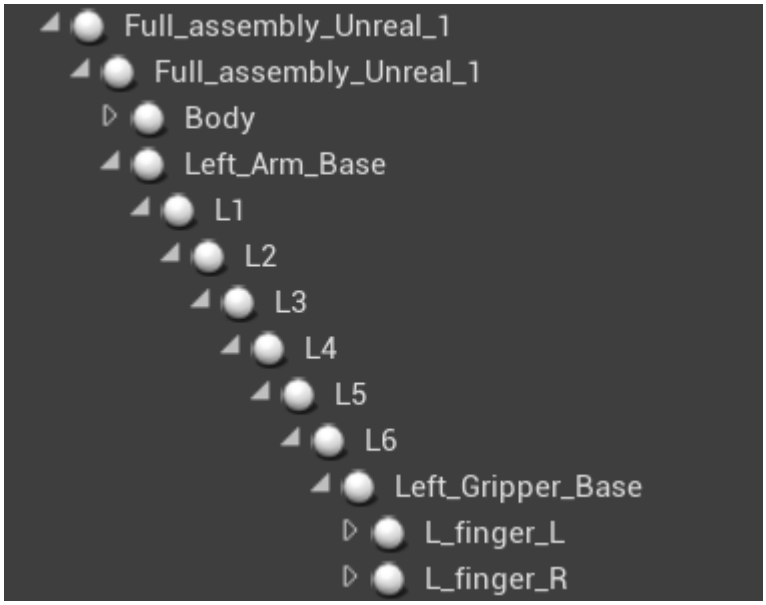

Figure 3.5 Joint hierarchy

Now it should be possible to move a joint by adjusting its rotation around X axis and all the child parts should follow the parent.

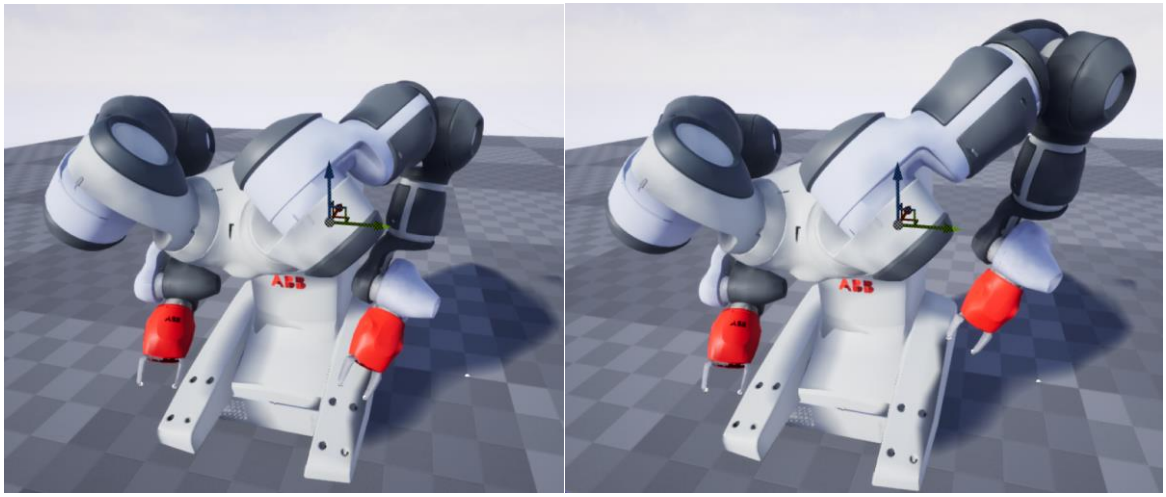

Figure 3.6 Movement of a joint and children joints

3.2 Reinforcement Learning Libraries and Frameworks

Reinforcement learning requires a well-organized action-reward system. There is not one standard way of training the system, there is however a certain format which determines how to introduce the inputs to the system.

OpenAI offers an open-source reinforcement learning framework which can be used as a base for different libraries. Simply, OpenAI Gym reinforcement learning framework consists of 3 main functions:

1) **Init:** Init function is used for initializing the environment and the spaces. This function sets the action space and observation space.

    a. **Action Space:** An action space represents the actions to be produced during the training process and, consequently, by the trained model. In another words, this is the output we receive at each training episode. Action spaces can be defined in several forms, but not every available machine learning libraries support those possible formats. The decision must be made upon a library's capacity. In this framework, only two types of action space formats will be used.

        i. *Discrete Action Space:* A discreet action is simply an integer value produced within an evenly distributed range. Based on the rules

of OpenAI Gym, the value is produced in a range starting from 0 to a given integer value (excluded).

ii. *Continuous (Box) Action Space:* A continuous action value is a float value and is produced in a range of minimum value(s) and maximum value(s) range.

b. **Observation Space:** Observation space is the definition of the range of the values to be tracked/observed. Those values are produced a user-defined custom function and received as an input to create an action based on them. Unlike action spaces, continuous observation space works for both discrete and continuous values. Minimum and maximum limits of the observation space must be set in the same way action space is set.

2) **Step:** Step function is a collection of functions which will be executed based on the action produced by the system. Simply, it takes an action and the current state, computes the logic set by the researcher, and produces a new observation state, a reward and an acknowledgement message to indicate if the training episode is ended or not. An episode may terminate on reaching a total number of episodes or on occurrence of any particular. There are several points worth mentioning:

a. Discrete functions produces actions starting from 0 to a maximum value. In that case, the action should be adjusted by the user. Say, a negative value is needed as an action, then a subtraction must be applied to the action.

b. Reward values should be kept as small as possible but a balance must be preserved between negative and positive rewards. The difference should be distinguishable.

3) **Reset:** Reset function is used for resetting the environment, the reward and the observation criteria if needed.

Using this framework, and a library that supports OpenAI Gym [40] can be used for further training. The most used libraries are TensorFlow [41], Keras [42] and Stable Baselines [43]. In this work, we will be using Stable Baselines for following reasons:

1) It is designed to work with OpenAI Gym.

2) It has limited action space formats which keeps the framework under control, too many features cause complexity.

3) It offers 12 different policies and multi agent training option. In this work, only one is shown but users can access rest of the policies in the next versions of the framework.

4) It is easy to save and reuse the trained model later on.

The versions of the libraries are extremely important. Even two different versions of the same library may differ considerably. For this work, we will be using Python 3.6.13, OpenAI Gym 0.18.0, Tensorflow 1.14.0 and Stable Baselines 2.10.2. The policy we will be using is Proximal Policy Optimization 2 [44].

3.3 Socket connection workflow

The networking blueprints should be able to send and receive data back and forth. Unreal Engine should work as the server and python script should connect to the server. If the connection is successful, the following data flow should be possible and acknowledgement messages can be sent and received to make sure the messages are being transferred correctly.

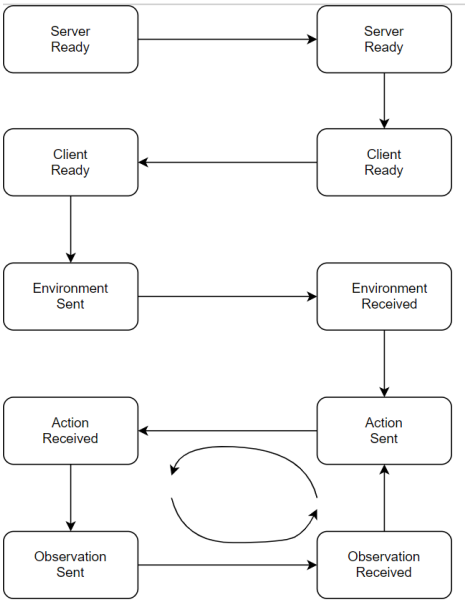

Figure 3.7 Data flow between server and the client

4. UNREAL ENGINE AND REINFORCEMENT LEARNING IMPLEMENTATION

Unreal Engine, just like other popular game engines, does not support too many artificial intelligence features. But it is not hard to find a way to work around it. Working with data processing, data transferring is an essential step for robotics and this is mostly done by socket connections. We could use this opportunity to connect new artificial intelligence libraries to our game engine. Another option would be using wrappers but in this work, Conscious Actors (CONSAC), we will be using socket connections to bring reinforcement learning feature to Unreal Engine.

Stable baselines uses python and python offers a very simple and easy to use socket connection. If a TCP connection can be set up in Unreal Engine as well, two sides of this project would work fluently.

Unreal Engine is a game engine and by the nature of it, it has multiplayer network connection but it is not designed for a third party library. Which means, a TCP connection has to be done by a user defined script. Unreal Engine uses C++, however, a language like C++ would cause performance failures if not used properly. To prevent this, Unreal Engine uses its own custom variables, structures and function (TArray, FString, FName, eg.)[47] which is almost a new language. Classic C++ networking methods would not work in this case but Unreal Engine has its own networking functions. It was intended to use Unreal Engine as the server side and the script was written based on this decision. For the ease of the usage, the C++ script was exposed to Blueprints so it can be used for graphical programming by researches.

With all those being said, it is worth mentioning why it is important to use blueprints and what challenges and what benefits it has. Blueprints, as mentioned in the previous sections, are designed to enable users to code without having to write the code on their own. Blueprints has a logic that consists of fundamentals of algorithms such as loops. Decision blocks, arithmetical and logical expressions which is not too different than that of Matlab or similar softwares. Given the fact that researches are using Matlab widely, blueprints make it easy to adopt for those users who have an algorithm background in any language. However, not every method, member or variable types that Unreal Engine has can be accessed by the blueprints directly. That is why, some functionalities should be placed in blueprints compatible functions and served as a blueprint library. In this project, it was needed to use IP and port numbers, socket classes and pointers to those classes, socket related functions but none of them could be exposed to blueprints

directly. Instead, those functions should be placed in a blueprint suitable function format, say a function which takes a string of and IP address as an input, process the string under the hood into an IP address variable and returns a bool value if the socket is created successfully.

Yet again, this brings us back to the coding which is not that easy to code in Unreal Engine and requires a solid C++ knowledge. But, what we want from this framework is to allow people to train their models only by using blueprints. Therefore, we had to create blueprint blocks on our own for establishing a socket connection which is where we started at.

4.1 TCP connection

A TCP connection consists of 2 sides, a server and a client. The blueprints we will we creating should provide the following communication flow between the server and the client.

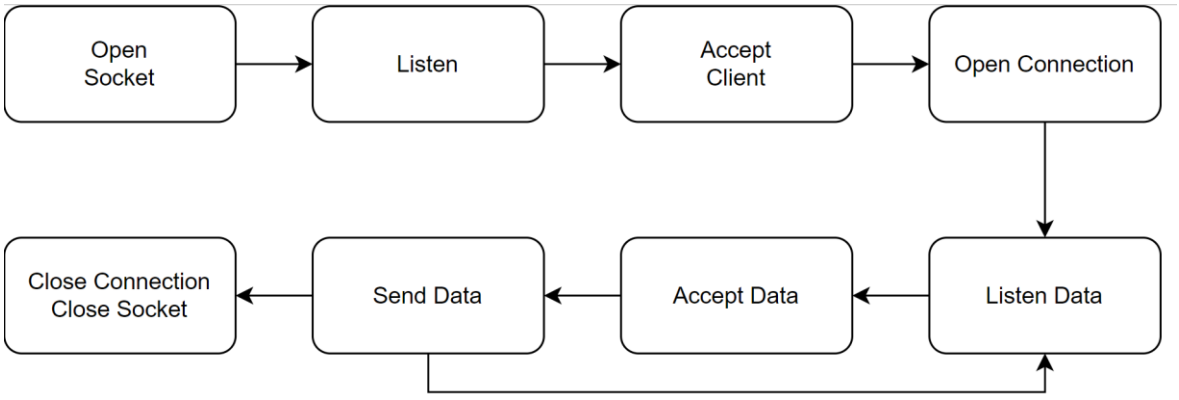

Figure 4.1 General TCP/IP connection

First blueprint node we created is the "Create Socket" node. The user has to set up buffer sizes and the end point (IP address and the port). There are no overloaded functions, therefore the values cannot be empty. This connection can be used for any other application but for this study, it is set to the following end point. The given end point will be decoded and used in an Unreal Engine function which cannot be exposed to the blueprints.

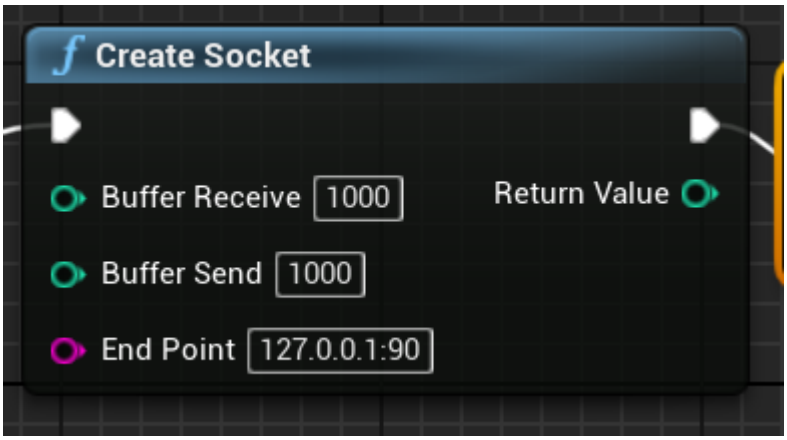

Figure 4.2 Blueprint node for creating a socket

"Create Socket" will create a socket but will not activate it right away. Following the create function, the socket connection can be started and closed by using two functions. The reason to put "Open Socket" functionality out of Create Socket is that Unreal Engine (and all the other engines) have an Awake/Start function that runs only once at the beginning of the application. Than the engine runs at a frame based update function which runs in a loop. Socket can be created at the beginning but if the user wants, it can be activated elsewhere on the runtime.

Figure 4.3 Blueprint node for opening and closing a socket

If a socket is open at the given end point, it must be closed after used so it can be used again. If not, the next time, Unreal Engine will crash on an attempt to open a socket which is already open.

Once the socket is on, the user can set it to listening mode to listen incoming clients by using check connection function and accept the client if there is any (if the function returns true, trying to accept a non-existing client will crash the engine). On accepting the client, a new socket will be assigned to the client and this socket is handled in the script and not exposed to the blueprints.

Figure 4.4 Blueprint node for listening and accepting a client

Incoming data can be checked by the "Check Pending Data" function and if there is any (if the function returns true), it can be accepted and a string will be received. Unreal Engine uses UTF-8 decoding, in case the user is using the connection blueprints for a personal use. Accepted data will be decoded and the output string doesn't need to be decoded again.

Figure 4.5 Blueprint node for listening and accepting a data

Sending data can be done by the Send Data function. A string can be sent and encoding will be handled automatically. As mentioned before, it will be encoded in UTF-8, so the receiving client should decode accordingly. Return value will show the data sent, if the user is sending a custom string, created by appending different values into a string, then it may need to be debugged.

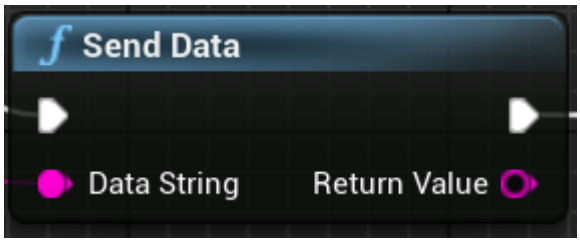

Figure 4.6 Blueprint node for sending data

If the connection with the client is to be closed, "Close Connection" function should be used. This is different than closing the socket's itself. Opening a socket will require a proper closing. If the user fails to close the socket on a certain end point, the next attempt to open a socket on the same end point will fail as it is not closed and still assigned. If it fails, it means the endpoint was assigned but now is released and it can be used again.

Figure 4.7 Blueprint node for closing a client connection

If Unreal Engine is being used in editor mode and the running simulations is stopped without closing the socket properly, the next attempt on running the simulation will fail the whole engine and Unreal Engine will shut down with an error. However, not closing the connection between server and the client is not that severe and it will not cause any problem.

4.2 Reinforcement learning environment in Unreal Engine

To set up reinforcement learning environment, an action space and an observation space must be created. This is done in 2 ways:

1) Discrete spaces: Discrete spaces are integer value spaces and exist in a range. For example, a discrete space of 3 means this space can only consists of two values, starting from 0: 0, 1 and 2. To set up a discrete space, discrete option should be checked and the length should be set. Low and high values must be set to zero as the function doesn't have and overloaded option.

2) Continuous spaces: A continuous space can be any value within a range. For instance in the range of 0 to 1, there are infinite values that can be produced. Those values however can be bypassed, clamped to avoid high computation volume. Lower value and high value can be set separately, unlike discrete

spaces. However, the size of the space must be defined, low and high values must be separated with a coma.

All the details regarding the environments can be found in the documentations of OpenAI Gym and Stable Baselines. In the scope of this work, it is not intended to explain how those frameworks function but rather create an easy working environment for researches to power their simulations with game engines and machine learning algorithms.

In this section, we will design our system in order to satisfy the structure in the following figure. But before we achieve the desired result, we need to design several core blueprints. Later on, we will combine them into nodes to have a user friendly framework.

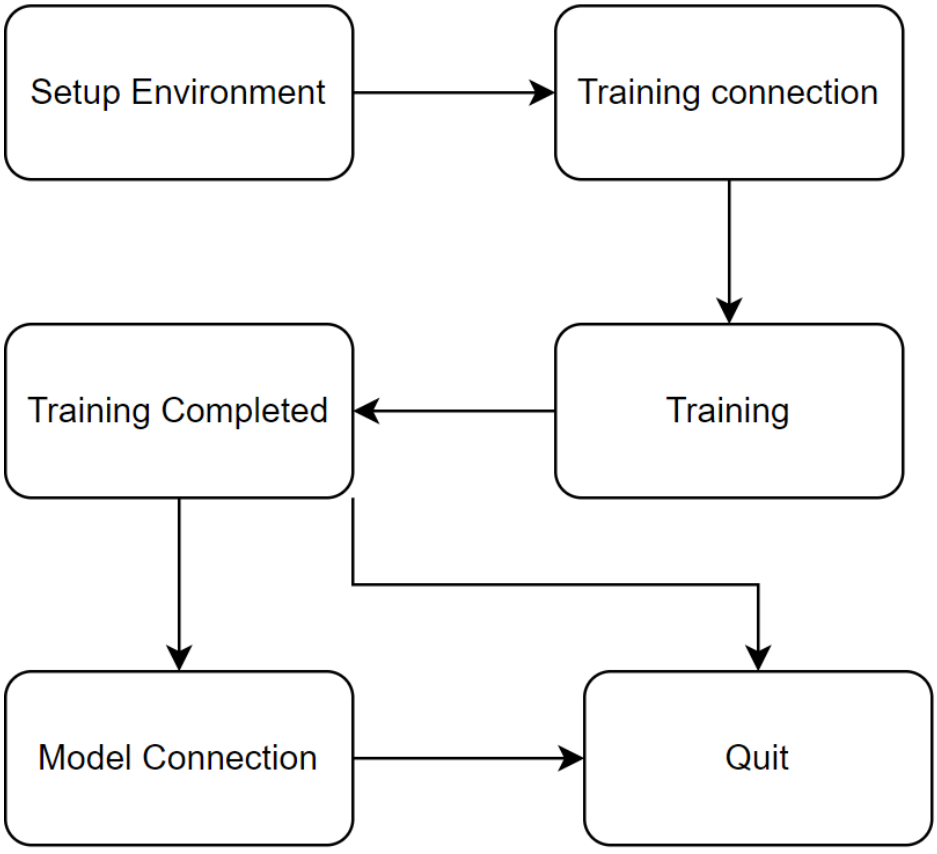

Figure 4.8 Reinforcement learning framework design

We have started with writing a C++ script and exposed it to blueprints for setting up the environment.

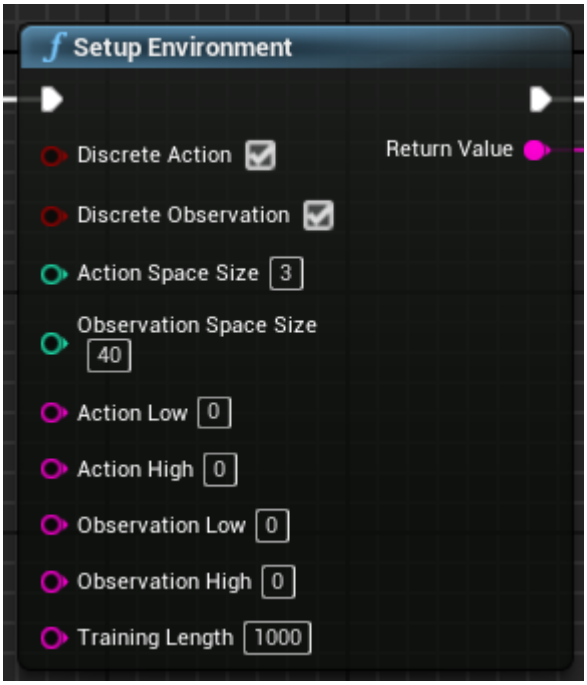

Figure 4.9 Blueprint node for compiling environment message

Training length must be set at the beginning but each episode in a training cycle will be determined by the user with custom functions.

The return value is a string that consists of all the required parameters and it can be sent to the client directly. The workflow and the format of this message will be explained in the following pages.

The environment parameters are sent only once and it cannot be changed during the training. Once it is set, the next things is to start sending observation, reward and done information and receive an action in return. This is the loop that the user needs to create. Once again, details about how the reinforcement learning works with OpenAI Gym environments and Stable baselines can be found in the documentation of OpeanAI Gym and Stable Baselines.

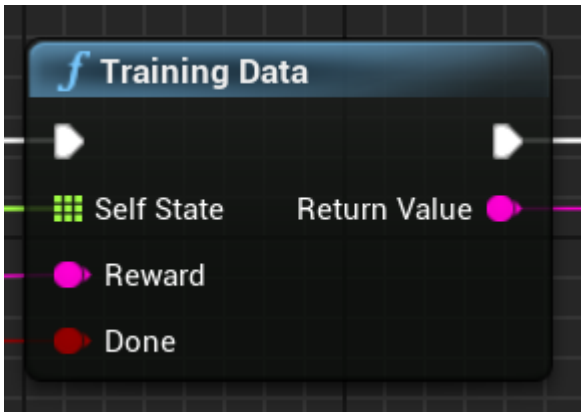

Figure 4.10 Blueprint node for compiling training data

The training data is a collection of a self-state (observation), reward and done status. The function for formatting inputs into a string is created in C++ and exposed to blueprints. The reward can be in any numerical type, Unreal Engine will handle the conversion. Return data is a formatted string, ready to be sent to the client.

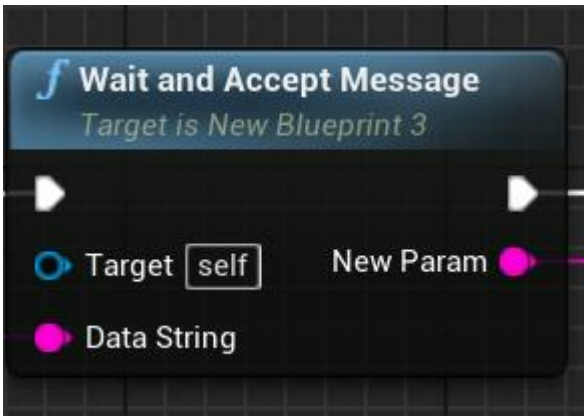

Figure 4.11 Blueprint node for accepting data from reinforcement learning algorithm

A blueprint node "Wait and Accept Message" is also available for accepting incoming action data and sending the observation data back. This is a collection of socket functions mentioned above. The user can create a custom "Check Pending Data" – "Accept Pending Data" – "Send Data" but the "Wait and Accept Message" is created to keep it easy and organized for the user. It takes the observation data as an input to send and returns the action received.

The actions can be decoded from a string form into a numerical array format to be used for the calculation. This format is an array even if there is only one action to receive. This is a generic format to support multiple actions.

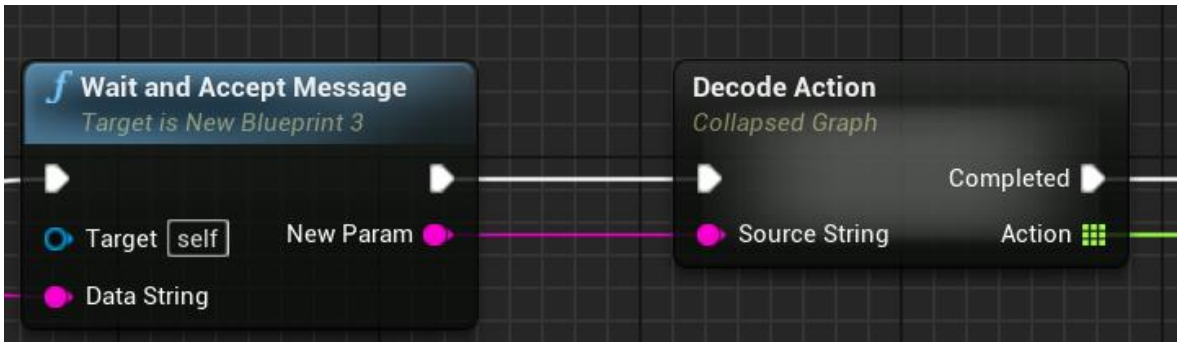

Figure 4.12 Blueprint node for decoding action string into an array

4.3 Reinforcement learning environment in Python

Python side of this project is close to the users and cannot be changed for the following reasons:

1) The flow of the data should not be disturbed, any small change in the source code could violate this fluency.

2) Stable baselines, OpenAI gym and their dependencies are version sensitive. If the script is not running in a proper python environment, it would suffer from versioning errors.

3) Simply, it is not possible to see the source code in the last product as it is delivered as a package to run without needing a python environment setup.

The general syntax for the python package is:

function init()

envData = receiveEnvironment()

[actionSpace, observationSpace] = decodeEnvironmentData(envData)

function step(self,action)

sendAction(action)

[observation, reward, done] = receiveObservation()

Return observation, reward, done

function reset()

resetValues()

receiveResetObservation()

step(action)

There are three types of messages being sent and received by the both sides:

1) **Environment setup message**

   a. ActionSpaceType # ActionSpaceSize#ActionSpaceLowAndHigh

   b. ObsSpaceType # ObsSpaceSize#ObsSpaceLowAndHigh

   c. TrainingLength

   Discrete space is represented with 'D' and Continuous space is represented with 'B' which stands for Box. All the variables are separated by a delimiter '#'.

   For instance; D#3#0#B#3#0,0,0,5,6,7#1000

2) **Observation message**

   a. Observation

   b. Reward

   c. Done

   Observation is an array of observed values, it cannot exceed the size of observation space, specified at the beginning. Reward can be an integer or a float number. Done is a bool value but it is represented with 1 and 0.

   As an example; 2.23,4.1,5#2.3#0

3) Action value

Action value is received as an array, containing action values only. It is then decoded into a float array. Actions can be accessed by their index and used as intended.

If the connection works as it is supposed to, the progress of the training can be tracked as follows. Every once in a while, stable baselines will give a feedback on the status of the training. Total time steps indicates the training cycle. When it reaches to the training maximum length the user provided, training session ends.

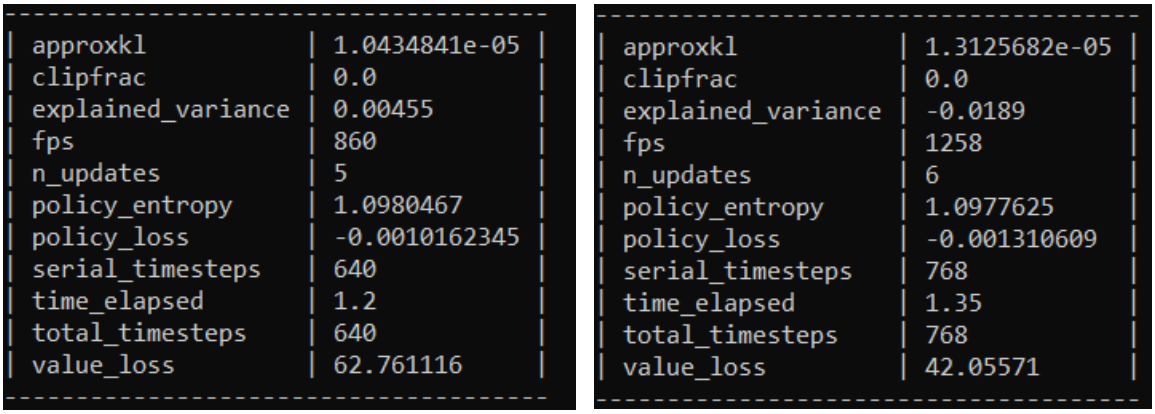

Figure 4.13 Feedback from reinforcement learning algorithm

4.4 Saving and reusing the trained models

OpenAI Gym does not support trained model saving, hence Stable Baselines handles the saving and loading models. Multiple models can be saved and loaded separately, as well as they can be overwritten.

To load and save a model, user has to supply a name for the model to be saved or to be loaded. Above, an input regarding models did not exist yet and implemented at the last step of this work. Now the user can give a name to the model at the beginning, using "Setup Environment" node. This node is also improved with a socket property under the hood.

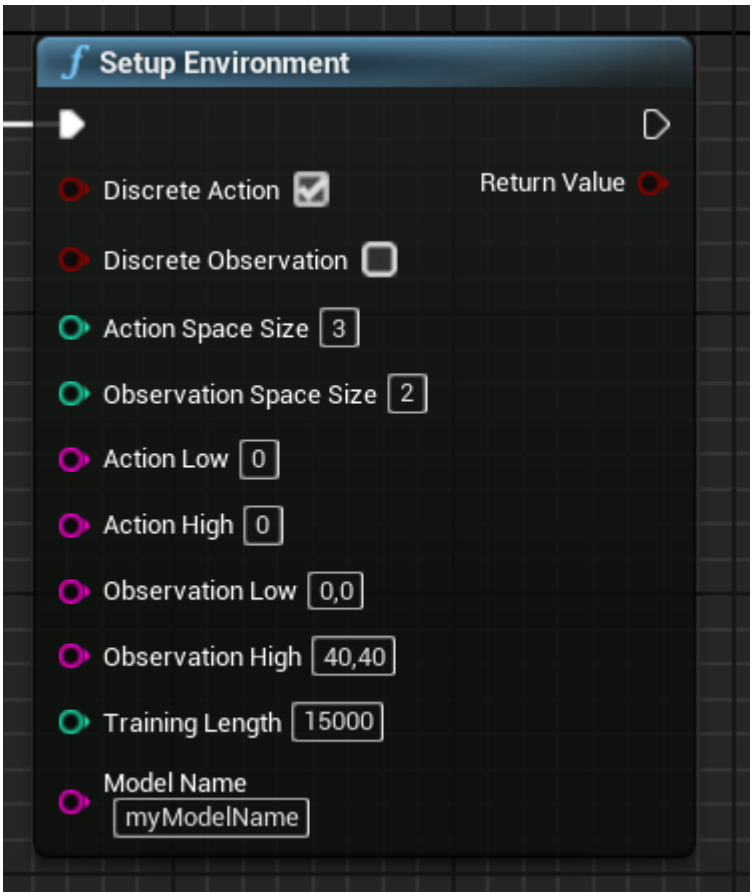

Figure 4.14 Setup environment with model feature

Accepting the client which is our python package, is handled by a new node block "Training Connection". This node will listen for an incoming connection and accept it. And if there is a client connected, it will return true. This function should be used with an if statement (a branch in blueprints). The reason will be explained later with a real example.

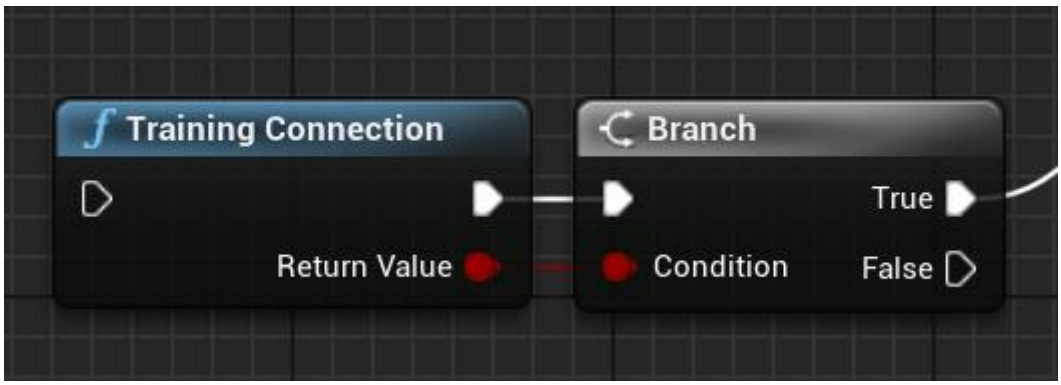

Figure 4.15 New node for handling client connection

Another new node “Training” will take the observation data, send it to the client, receive a new action in return, decode it and convert it into an array.

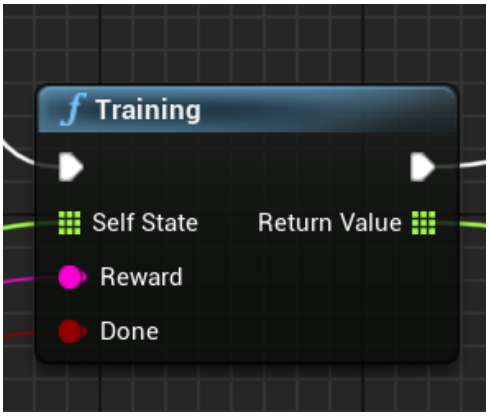

Figure 4.16 New node to manage training data flow

If the training is completed, “Training Completed” node will return true.

Figure 4.17 New node the control training status

At the end of the training, you can use the new model now.

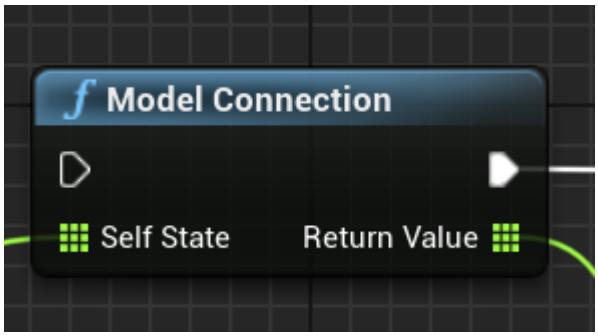

Figure 4.18 New node to load models

“Model Connection” node will take an observation value and return an action from the trained model. As you can see, observation value does not have to include a reward and done indicator anymore. To be able to save a model, “Environment Setup” node should be used at the beginning with a proper model name.

We can see the whole construction on an example. First, we setup an environment using “Setup Environment” node and open a socket under the hood. This block handles the socket opening, therefore it must be used for both training and loading a model. This has to be done once which is why we used it after an “Event Begin Play”.

Figure 4.19 Use case of 'Setup Environment'

Now it is possible to listen for a client and accept the client before starting the training session. We mentioned that the “Training Connection” node must be used with an if statement (called Branch in Unreal Engine). The “Event Tick” creates a loop that runs at every frame. There should be a mechanism which will block the process if there is no client, otherwise the functions used for sending and receiving data in the following steps will try to send a message to a non-existing client and end up with an error. Consequently, we need a decision making block to allow us to send data if only a client is ready for it.

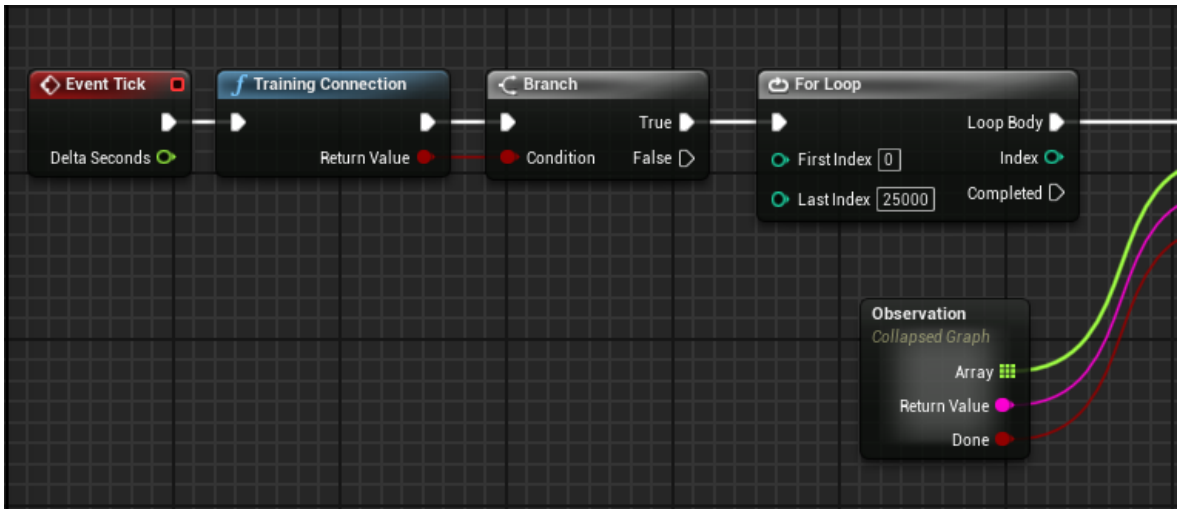

Figure 4.20 Use case of 'Training Connection'

In the example, there is a 'for loop' shown. This loop is not necessary, because the "Event tick" already creates a loop. But event tick loop runs once at each frame, whereas a "for loop" runs at much higher rates. It can be removed but the speed will decrease drastically.

If the connection is successful, training process can be started with a "Training" node. The "Observation" node in the image above is a custom node, user can produce the inputs for the "Training" node in the way they like. The "Reset" node is a custom node created by us for this specific example, for resetting the training episode, it is not included in the framework.

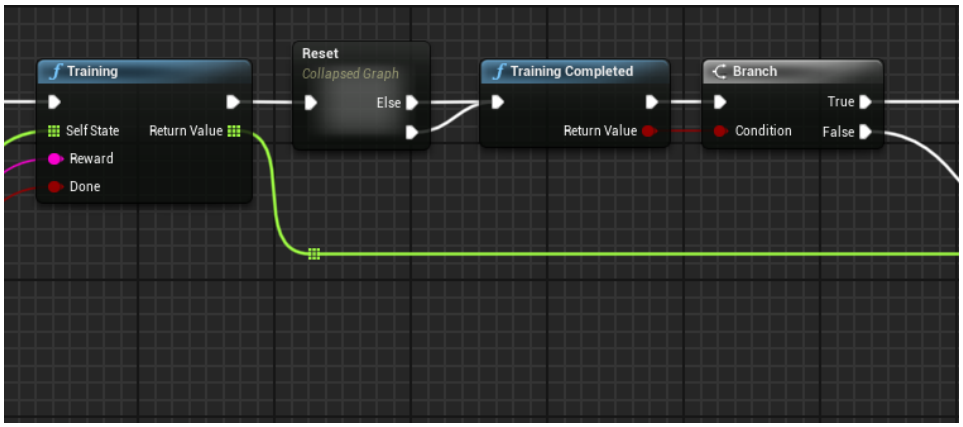

Figure 4.21 Use case of 'Training' and 'Training Completed'

In every step, user should check with “Training Completed” node if the training is completed. The “Step” node is a custom node, it takes the received action and uses it as the user defined. When the training process comes to an end, “Training Completed” node returns true. In this example, we closed all the connections and stopped editor.

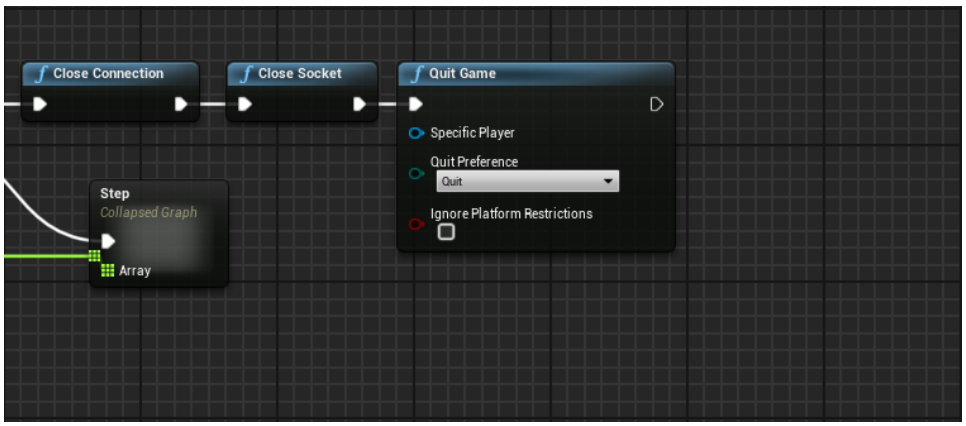

Figure 4.22 Closing connection properly

It is then possible to reuse the model with a “Model Connection” node. This still requires an “Environment Setup” block to open a socket and send the model name to be loaded.

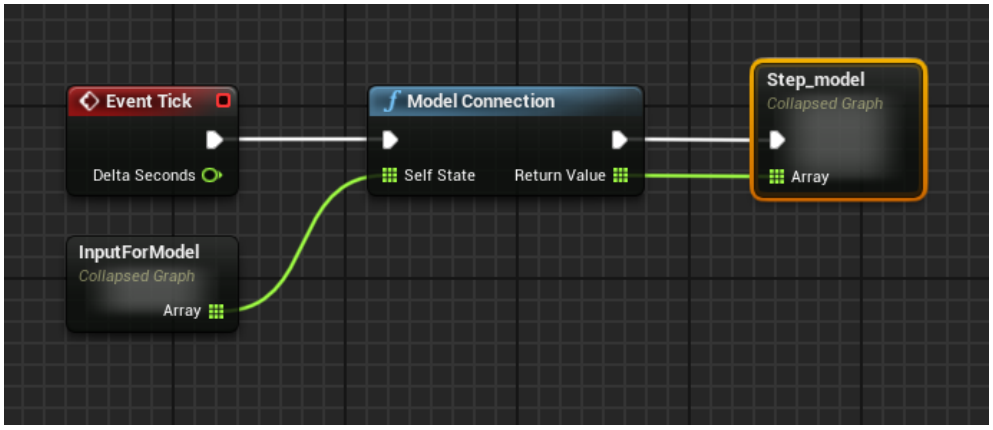

Figure 4.23 Model loading

We have shown a use case of Unreal Engine blueprints above. Now we can see the output of it on the python side.

The definition of our environment: We have a discrete road, where the start point A is 0 and the end point B is 40. Our actor Mario is placed at 10 and a golden coin is placed randomly. After every 50 episode, coin location is assigned to a new random value. We expect our actor to move to the gold coin. Our observation consists of those two values, the state of our actor Mario and the state of gold coin.

Figure 4.24 Actor and objective for the training

Rewarding system: If our actors move towards to our goal destination, it gets +1 point. If not, it gets 0 point. If the actor goes out of our 0-40 limits, the system signals "Done" and everything resets. Actor gets a random starting point, goal destination gets a random point.

We trained our model for 20000 training episodes which took around 20 seconds. This is the benefit of using a for loop as we discussed before. At a fixed frame rate, training would take 4 to 5 times longer.

In the feedback message, total timesteps shows how many training episode has passed. This message is published by Stable Baselines and it occurs only when training the model. A trained model do not give any feedback. The state received message where you can see the last observation, the first part separated with a delimiter '#' showing actor state and the destination state, before resetting the system is added by us.

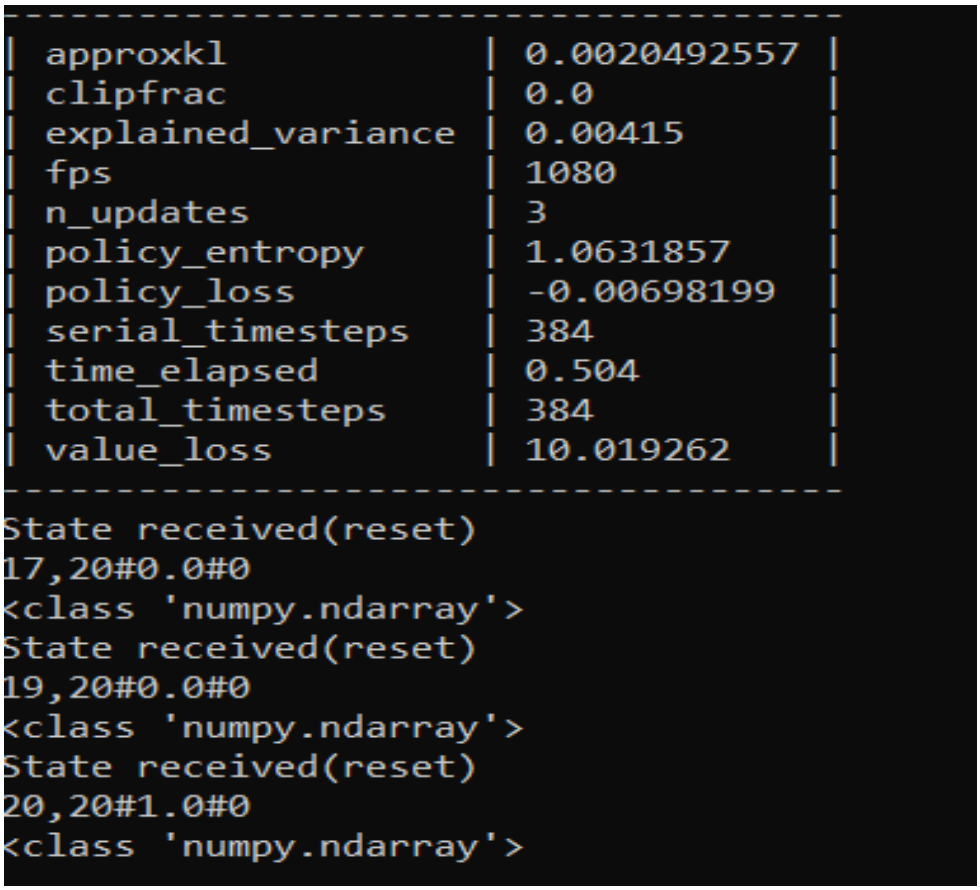

Figure 4.25 Feedback during the training

The training data is saved into a logged file which has the same name as the model. The progress can also be tracked by TensorBoard using the log file.

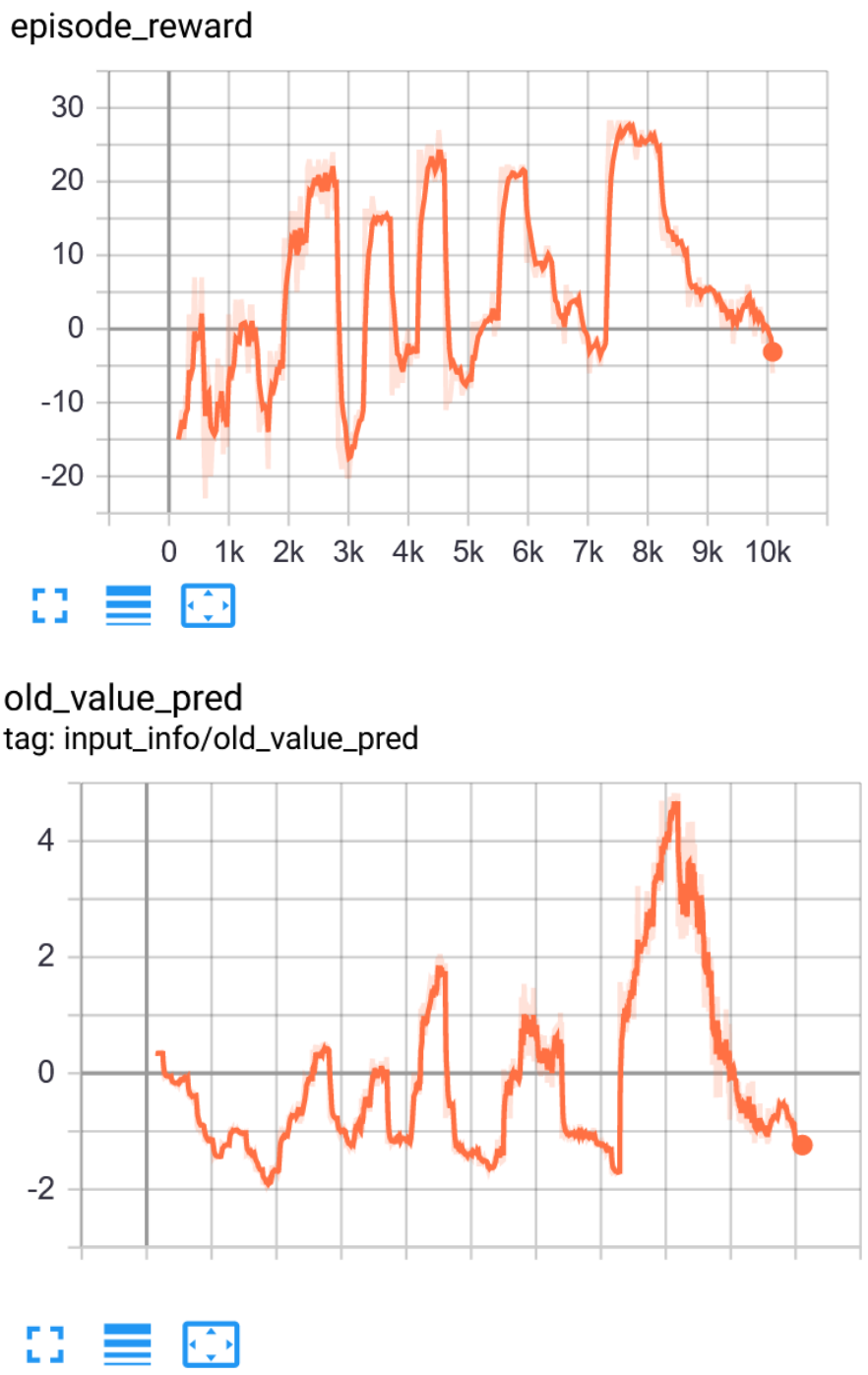

Figure 4.26 Reward tracking on TensorBoard

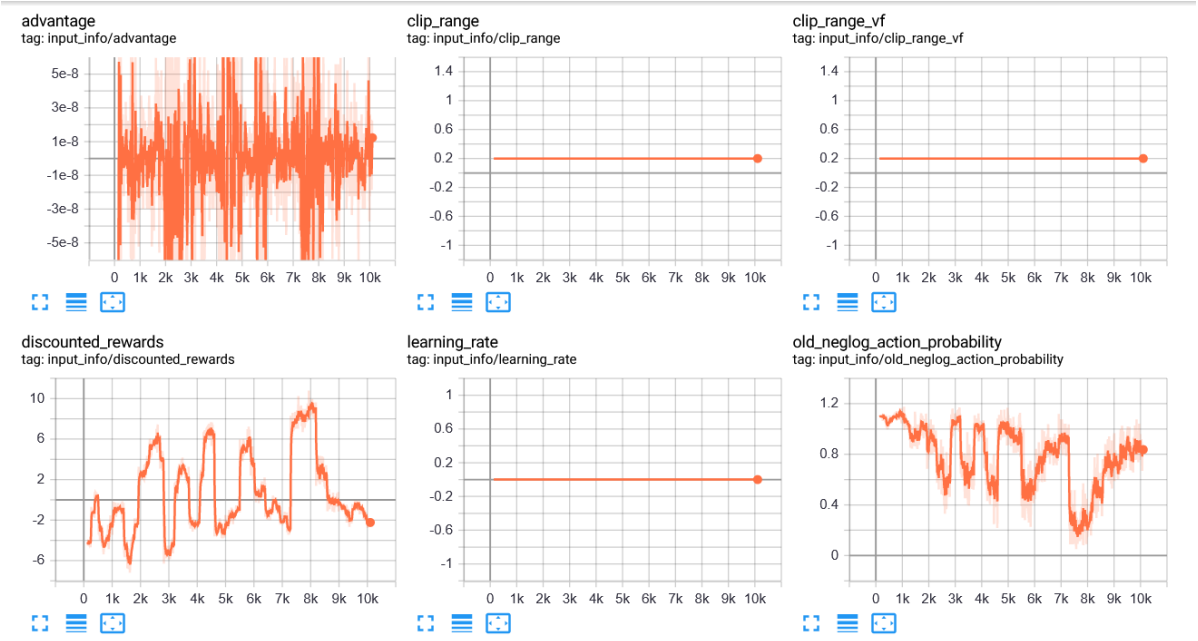

Figure 4.27 Traceable extra features

A loaded model will produce an observation message where we can track how our model is acting. Here, we can see that our actor is moving towards the destination. On arriving, a new destination is assigned and the actors starts reacting immediately.

```
Obs:[18.  5.] Obs:[9. 5.]   Obs:[ 9. 35.]   Obs:[6. 5.]
Obs:[17.  5.] Obs:[8. 5.]   Obs:[10. 35.]   Obs:[5. 2.]
Obs:[17.  5.] Obs:[9. 5.]   Obs:[11. 35.]   Obs:[4. 2.]
Obs:[17.  5.] Obs:[8. 5.]   Obs:[12. 35.]   Obs:[5. 2.]
Obs:[16.  5.] Obs:[7. 5.]   Obs:[13. 35.]   Obs:[4. 2.]
Obs:[15.  5.] Obs:[8. 5.]   Obs:[14. 35.]   Obs:[3. 2.]
Obs:[15.  5.] Obs:[7. 5.]   Obs:[15. 35.]   Obs:[3. 2.]
Obs:[14.  5.] Obs:[7. 5.]   Obs:[16. 35.]   Obs:[ 2. 35.]
Obs:[13.  5.] Obs:[8. 5.]   Obs:[17. 35.]   Obs:[ 3. 35.]
Obs:[12.  5.] Obs:[7. 5.]   Obs:[18. 35.]   Obs:[ 4. 35.]
Obs:[11.  5.] Obs:[8. 5.]   Obs:[19. 35.]   Obs:[ 5. 35.]
Obs:[10.  5.] Obs:[7. 5.]   Obs:[18. 35.]   Obs:[ 6. 35.]
Obs:[9. 5.]   Obs:[6. 5.]   Obs:[19. 35.]   Obs:[ 7. 35.]
Obs:[8. 5.]   Obs:[7. 5.]   Obs:[20. 35.]   Obs:[ 8. 35.]
Obs:[9. 5.]   Obs:[8. 5.]   Obs:[21. 35.]   Obs:[ 7. 35.]
Obs:[10.  5.] Obs:[7. 5.]   Obs:[22. 35.]   Obs:[ 8. 35.]
              Obs:[6. 5.]   Obs:[21. 35.]
```

Figure 4.28 Trained model output

5. DISCUSSION

In this work, we worked on importing CAD models, deciding on a suitable reinforcement learning library, connection between the library and the game engine. Finally, we have created a generic reinforcement learning framework for Unreal Engine. Along the way, there have been several challenges.

1) Importing CAD files are not as easy as it seems. The materials on the model may cause importing problems, may not be visible in Unreal Engine and may require another conversions.

2) CAD model origins must be corrected in a CAD software carefully. Unreal Engine accepts a model as a whole, it is not possible to select edges, vertices or surfaces. Aligning objects or rearranging axis is not possible as it is in a CAD software.

3) Unreal Engine encourages people to use blueprints and blueprints works just fine. As we have done for this project, it is also possible to create custom blueprints from scratch. From user's side, blueprints looks identical but a native blueprint to native blue print connection is quite different than a native blueprint to custom blueprint connection. A custom blueprint block should agree with the preceding block, hence the logical structure of the custom block requires adjustments.

4) Unreal Engine uses C++ on its own way. Sometimes, it is not possible to use general C++ techniques for it would not agree with Unreal Engine. Simply, you should code by Unreal Engine's own rules.

5) There are several reinforcement learning libraries available, be it individual or co-working libraries. With new researches and developments, every publisher updates their libraries and publishes under different versions. It is very important to find correct versions of those libraries in the same project in order to keep them working together, which is quite cumbersome.

6) There are also various reinforcement learning policies and each has its own rules, structures and working style. Trying to include all of them in one framework expends the scope of the framework and it takes it away from being a "generic" framework.

All the challenges we faced during this work showed us that the framework we created can make many people's life easier in their researches. We were able to solve the problems along the way but not yet perfected it. This is where we set our new goals for the future. Starting with the framework we compiled in this work following improvements will take place in the future;

1) All possible Stable Baselines reinforcement learning policies will be implemented to the framework.

2) Hyper parameters for Stable Baselines policies will be added as adjustable properties.

3) Keras library will be available after Stable Baselines is fully implemented. This is because the model saving/loading way of Keras library is close to that of Stable Baselines.

4) At the moment, trained model requires a TCP connection to transfer data. In the future work, trained model will be imported directly. This will allow users to create standalone applications and run those applications without needing an external library support.

5) And on top of all, there are still many reinforcement learning solutions out there. This framework should have an open end that would allow users to use any library they want via TCP connection, as long as they stick to the format of the framework.

6. SUMMARY

In this work, we tried to draw a frame around the term 'Digital Twin', hoping that it would be useful to distinguish digital twins from simulations. The motivation in digital twins is "Can we go beyond the simulation and get it as close as possible to the real model?". We can always go further, that is a valid purpose for simulations as well, but we may not have the exact tools for that. For a digital twin design, we might need different simulations and tools to work together. We took it from there and decided to create a new tool to combine the power of game engines and reinforcement learning into one project, Conscious Actors (CONSAC).

Along the way, we repeatedly mentioned the benefits of digital twins, reinforcement learning and their use cases. The term "Digital twins" is still too new, there is still too much to try and find out what digital twins can offer to us. Regarding reinforcement learning, it is exciting to let a system to discover its own potential, even small possibilities we wouldn't take into account, and see it evolving in the right direction.

As promising as reinforcement learning and digital twins sound, it takes time, hard work and rigorous study to create a robust model. We do believe, this work will offer new tools and possibilities to users owing to the features that come with game engines; and hopefully make the process easier for them.

7. KÕKKUVÕTE

Selle töö käigus prooviti luua raamistikku terminile "digitaalne kaksik", lootes et see tuleb kasuks digitaalse kaksiku ja simulatsioonide vahel eristamisel. Digitaalsete kaksikute põhimõte on "Kas on võimalik saavutada järgmine aste simulatsioonist, mis oleks võimalikult sarnane tegelikule mudelile?". Alati on võimalik simulatsioone täiendada, kuid nende jaoks ei pruugi leiduda õigeid vahendeid. Digitaalse kaksiku loomise puhul võib vaja minna erinevaid simulatsioone ja vahendeid, mis töötaksid üheskoos. Seega otsustati luua uus platvorm, mis kombineeriks mängumoototire võimekuse ja stiimulõppe ühte projekti, Conscious Actors (CONSAC).

Töö käigus toodi korduvalt välja digitaalse kaksiku ja stiimulõpe kasulikkus ning nende erinevad kasutusalad. Digitaalset kaksikut on siiski alles liiga vähe käsitletud ja selle teema kohta on veel palju uurida, et teada saada, kuidas see võiks inimestele abiks olla. Stiimulõppe algoritmid on rakendatud nii, et süsteem avastab iseseisvalt enda potentsiaali, arvestab ka kõige vähemolulisi nüansse ning areneb maksimeeritud tulemuse saavutamise suunas.

Ehkki stiimulõpe ja digitaalne kaksik kõlavad väga paljulubavalt, võtab robustse mudeli loomine palju aega, uurimist ja tööd. On tugev uskumus, et käesolev töö loob uue platvormi ja erinevad võimalused kasutajatele, mis muudab kogu protsessi lihtsamaks , võimaldades kasutada mängumootorite erinevaid funktsioone.

LIST OF REFERENCES

[1] GHOBAKHLOO, Morteza. Industry 4.0, digitization, and opportunities for sustainability. *Journal of Cleaner Production*, 2020, 252: 119869.

[2] LEWIS, M. A.; BEKEY, G. A. Automation and robotics in neurosurgery: Prospects and problems. *Chapter 6 in Neurosurgery for the Third Millineum*, 1992, 65-79.

[3] LOUHISALMI, Yrjö; LEINONEN, Tatu. On research of directly programmable surgical robot. In: *Proceedings of 18th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*. IEEE, 1997. p. 229-230.

[4] BEJCZY, Anta1 K. Towards development of robotic aid for rehabilitation of locomotion-impaired subjects. In: *Proceedings of the First Workshop on Robot Motion and Control. RoMoCo'99 (Cat. No. 99EX353)*. IEEE, 1999. p. 9-16.

[5] HAAG, Sebastian; ANDERL, Reiner. Digital twin–Proof of concept. *Manufacturing Letters*, 2018, 15: 64-66.

[6] BOSCHERT, Stefan; ROSEN, Roland. Digital twin—the simulation aspect. In: *Mechatronic futures*. Springer, Cham, 2016. p. 59-74.

[7] PANG, Linyong, et al. Making digital twins using the Deep Learning Kit (DLK). In: *Photomask Technology 2019*. International Society for Optics and Photonics, 2019. p. 111480B.

[8] ALASADI, Suad A.; BHAYA, Wesam S. Review of data preprocessing techniques in data mining. *Journal of Engineering and Applied Sciences*, 2017, 12.16: 4102-4107.

[9] SHAFAEI, Alireza; LITTLE, James J. Real-time human motion capture with multiple depth cameras. In: *2016 13th Conference on Computer and Robot Vision (CRV)*. IEEE, 2016. p. 24-31.

[10] FANG, Bin, et al. A novel data glove using inertial and magnetic sensors for motion capture and robotic arm-hand teleoperation. *Industrial Robot: An International Journal*, 2017.

[11] MATTHIAS, Bjoern, et al. Safety of collaborative industrial robots: Certification possibilities for a collaborative assembly robot concept. In: *2011 IEEE International Symposium on Assembly and Manufacturing (ISAM)*. Ieee, 2011. p. 1-6.

[12] EL ZAATARI, Shirine, et al. Cobot programming for collaborative industrial tasks: An overview. *Robotics and Autonomous Systems*, 2019, 116: 162-180.

[13] Robotiq, Cobots EBook, 2018, [Online]. Available: https://blog.robotiq.com/collaborative-robot-ebook.

[14] Robots and robotic devices – Collaborative robots, ISO Standard ISO/TS 15066 (2016) 2016.

[15] LAUD, Adam Daniel. *Theory and application of reward shaping in reinforcement learning*. 2004.

[16] Reinforcement Learning: An Introduction; Richard S. Sutton and Andrew G. Barto 2014, 2015

[17] IEEE standard glossary of software engineering terminology. IEEE Std. 610.12-1990, page 72.

[18] BEKRAR, Sofia, et al. Finding software vulnerabilities by smart fuzzing. In: *2011 Fourth IEEE International Conference on Software Testing, Verification and Validation*. IEEE, 2011. p. 427-430.

[19] ESSICK, John. *Hands-on introduction to LabVIEW for scientists and engineers*. Oxford University Press, 2013.

[20] PAIVA, Sofia Costa; SIMÃO, Adenilso. An Experimental Study for Complete-IOCO Theory. In: *Proceedings of the 5th Brazilian Symposium on Systematic and Automated Software Testing*. 2020. p. 107-116.

[21] BENVENISTE, Albert, et al. *Contracts for system design*. 2012. PhD Thesis. Inria.

[22] KOUSI, Niki, et al. Digital twin for adaptation of robots' behavior in flexible robotic assembly lines. *Procedia manufacturing*, 2019, 28: 121-126.

[23] BEJARANO, Ronal, et al. Implementing a Human-Robot Collaborative Assembly Workstation. In: *2019 IEEE 17th International Conference on Industrial Informatics (INDIN)*. IEEE, 2019. p. 557-564.

[24] RONZHIN, Andrey; RIGOLL, Gerhard; MESHCHERYAKOV, Roman (ed.). *Interactive Collaborative Robotics: First International Conference, ICR 2016, Budapest, Hungary, August 24-26, 2016, Proceedings*. Springer, 2016.

[25] NETO, Pedro; MENDES, Nuno. Direct off-line robot programming via a common CAD package. *Robotics and Autonomous Systems*, 2013, 61.8: 896-910.

[26] LE, Dong. Application for the ABB IRB 14000 YuMi robot using Integrated Vision and 3D printing. 2020.

[27] ROZO, Leonel, et al. Learning physical collaborative robot behaviors from human demonstrations. *IEEE Transactions on Robotics*, 2016, 32.3: 513-527.

[28] KONG, Linghe, et al. AdaSharing: Adaptive data sharing in collaborative robots. *IEEE Transactions on Industrial Electronics*, 2017, 64.12: 9569-9579.

[29] XIAO, Juliang, et al. Collision detection algorithm for collaborative robots considering joint friction. *International Journal of Advanced Robotic Systems*, 2018, 15.4: 1729881418788992.

[30] ALEBOOYEH, Morteza; URBANIC, R. Jill. Neural network model for identifying workspace, forward and inverse kinematics of the 7-DOF YuMi 14000 ABB collaborative robot. *IFAC-PapersOnLine*, 2019, 52.10: 176-181.

[31] GHADIRZADEH, Ali, et al. Human-centered collaborative robots with deep reinforcement learning. *IEEE Robotics and Automation Letters*, 2020.

[32] KORMUSHEV, Petar; CALINON, Sylvain; CALDWELL, Darwin G. Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, 2013, 2.3: 122-148.

[33] HARRIS, Adam; CONRAD, James M. Survey of popular robotics simulators, frameworks, and toolkits. In: *2011 Proceedings of IEEE Southeastcon*. IEEE, 2011. p. 243-249.

[34] FREUND, Eckhard; ROKOSSA, Dirk; ROßMANN, Jürgen. Process-oriented approach to an efficient off-line programming of industrial robots. In: *IECON'98. Proceedings of the 24th Annual Conference of the IEEE Industrial Electronics Society (Cat. No. 98CH36200)*. IEEE, 1998. p. 208-213.

[35] MATTINGLY, William A., et al. Robot design using Unity for computer games and robotic simulations. In: *2012 17th International Conference on Computer Games (CGAMES)*. IEEE, 2012. p. 56-59.

[36] VALLS, Francesc, et al. Videogame technology in architecture education. In: *International Conference on Human-Computer Interaction*. Springer, Cham, 2016. p. 436-447.

[37] CRESPO, Raul; GARCÍA, René; QUIROZ, Samuel. Virtual reality simulator for robotics learning. In: *2015 International Conference on interactive collaborative and blended learning (ICBL)*. IEEE, 2015. p. 61-65.

[38] GANONI, Ori; MUKUNDAN, Ramakrishnan. A framework for visually realistic multi-robot simulation in natural environment. *arXiv preprint arXiv:1708.01938*, 2017.

[39] DELP, Scott L., et al. OpenSim: open-source software to create and analyze dynamic simulations of movement. *IEEE transactions on biomedical engineering*, 2007, 54.11: 1940-1950.

[40] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, Wojciech Zaremba, OpenAI GYM, arXiv:1606.01540v1,2016

[41] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., ... & Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems, software available from tensorflow. org (2015). *URL https://www. tensorflow. org*.

[42] Ketkar, Nikhil. "Introduction to keras." In *Deep learning with Python*, pp. 97-111. Apress, Berkeley, CA, 2017.

[43] Hill, Ashley and Raffin, Antonin and Ernestus, Maximilian and Gleave, Adam and Kanervisto, Anssi and Traore, Rene and Dhariwal, Prafulla and Hesse, Christopher and Klimov, Oleg and Nichol, Alex and Plappert, Matthias and Radford, Alec and Schulman, John and Sidor, Szymon and Wu, Yuhuai, Stable Baselines , GitHub,2018, GitHub repository. Available from: https://stable-baselines.readthedocs.io/en/master/

[44] Schulman, John, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. "Proximal policy optimization algorithms." *arXiv preprint arXiv:1707.06347* (2017).