Opleiding Informatica

Developing a Digital Twin of a Food Industry Imaging Setup in Unity3D

3712478

R.C. Salden

Supervisors:
Prof.Dr. K.J. Batenburg
André Mesquita Fery Antunes MSc

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

01/06/2025

Abstract

This thesis developed a Digital Twin for the food industry imaging setup in the Physical Twin lab of SAILS. The Digital Twin is a digital counterpart to the physical setup, which simulates the same expected behaviors. A Digital Twin provides a digital platform to experiment with the setup, such as predicting behavior, gathering data, etc. The behaviors to simulate include the movement of a rotary carousel, the movement of two robot arms, and computing X-ray images. Objects are transported on a rotary carousel, of these objects we can compute X-ray images by simulating one robot arm as a source and the other as a detector. For the development of the Digital Twin we use the game engine Unity3D. The movement of the carousel is simulated by moving the "holders" across a predetermined path. The movement of the robot arms is done by training a model with reinforcement learning within Unity3D using the ML-Agents package. The simulated X-ray images are computed using a server that accepts coordinates and orientations of the source, detector, and objects. The performance of the model for the movement of the robot arms was measured at different speeds and accuracy scales, which showed the model performing poorly at a low scale, leaving room for improvement.

Contents

1 Introduction

Throughout history we have marked some of the most important changes in the industry as an Industrial Revolution. So far, we have seen three industrial revolutions. However, the fourth industrial revolution, also known as Industry 4.0, is currently underway. The main concept of Industry 4.0 is the digitization of industrial production. Essentially the idea of Industry 4.0 is to make industrial production "smart". One such implementation of making industrial production "smart" is using a Digital Twin. A Digital Twin can be seen as the digital counterpart of the Physical Twin. For example, suppose you have a conveyor belt which transports products. By using a Digital Twin, it would be possible to predict whether or not it is necessary for the speed of the conveyor belt to be adjusted according to the amount of products which are incoming. Knowing what amount of products is incoming would also be a part of the concept of Industry 4.0, as gathering such data would require the implementation of technology such as sensors or cameras.

One of the most important topics of discussions in the present day concerns the sustainability of our planet. Industry 4.0 has been mentioned to greatly support the development of more sustainable industrial production. Lasi [LFK+14], mentions how one of the important concepts of Industry 4.0 includes Corporate Social Responsibility. This concept focuses on sustainability and resource efficiency. Ghobakhloo [Gho20] describes how digital connectedness helps with resource efficiency and waste reduction. However, they also mention how the increased rate of production would also lead to a higher resource and energy consumption. Many other papers touch on the relevance of Industry 4.0 in accordance to sustainability and reducing waste emissions.

The research of this paper aims to answer the question: How do we develop a Digital Twin of a Food Industry Imaging Setup in Unity3D. The Food Industry Imaging Setup consists of a carousel and two robot arms. The carousel transports products and the two robot arms make images of the products using a similar technique as radiographs. A more in-depth explanation of the setup is provided in section 2.2. This leads us to our research question: How do we simulate the expected behaviors of the physical twin in Unity3D. To answer this, we research what the expected behaviors include, such as the movement of the carousel and robot arms, as well as the imaging for the radiographs. For the behavior of the robot arms we experiment with training an intelligent agent in Unity using reinforcement learning, which is made possible by the ML-Agents [JBT+20] toolkit. For the imaging of the radiographs, we make use of the ASTRA toolkit [vAPC+16].

1.1 Thesis overview

This bachelor thesis was conducted under the supervision of Prof.Dr. K.J. Batenburg at LIACS. Section 2 includes the necessary background knowledge and definitions; Section 3 describes the methodology of this research; Section 4 describes the experiments performed on the movement of the robot arms; Section 5 provides an overview of the functionalities of the Digital Twin; Section 6 discusses how the research went, concludes the research, and highlights the possibilities for future research.

2 Background

The goal of this project is to develop a Digital Twin of the food industry imaging setup located in the SAILS Physical Twin lab. The Physical Twin Lab of SAILS focuses on providing an industry-like setup that has a high accessibility for research. This setup features a carousel, which is able to transport objects, as well as two robot arms, which act as a "source" and a "detector" to simulate X-ray imaging. The X-ray imaging is simulated using the ASTRA toolbox, which provides a collection of tools for tomographic projection and reconstruction. The Digital Twin for this project will be developed using the cross-platform game engine Unity3D, in which we use the ML-Agents package to train a model using reinforcement learning to simulate the expected movement of the robot arms.

The term Digital Twin emerges from one of the main concepts of Industry 4.0: Cyber-physical systems. Cyber-physical systems consist of merging the physical and digital levels. In these systems, the physical and digital counterparts cannot be differentiated. The digital counterparts in these systems are known as Digital Twins. In such a system, a Digital Twin would be able to perform various tasks such as, predicting performance and maintenance of the physical counterpart, check the current status of the physical counterpart remotely, keep track of statistics, etc. In conclusion, a Digital Twin can be very useful for varying forms of application.

Aside from Cyber-physical systems, Industry 4.0 consists of many other different concepts and fields of expertise. Industry 4.0 is known as the fourth industrial revolution, which has a large focus on connecting the physical and digital world. This connection has become a necessity for manufacturing in the current industry because of increasing demands, such as the demand for a high innovation rate, as well as the demand for a high flexibility.

2.1 Industry 4.0

As mentioned in the introduction, Industry 4.0 is currently underway. It can be seen as a combination of many different fields of expertise, as well as a collection of many different concepts. In this paper, we will only elaborate on a couple of concepts, however, keep in mind that there are a lot of concepts within Industry 4.0 that are not mentioned within this paper. Lasi [LFK+14] describes how Industry 4.0 can be seen as two development directions.
Firstly, as an application-pull, which asks for changes in the operative framework conditions. Which are triggered by several changes, which include: Short Development Periods; Individualization on demand; Flexibility; Decentralization; Resource efficiency;.
Secondly, the other development direction is a technology-push. This direction focuses more on the spread of innovative technologies in the industrial field. This includes: Further mechanization and automation; Digitalization and networking; Miniaturization;.

As mentioned before, Industry 4.0 can be seen as a collection of many different concepts. Lasi [LFK+14] lists the following fundamental concepts: Smart Factory, Cyber-physical Systems, Self-organization, New systems in distribution and procurement, New systems in the development of products and services, Adaptation to human needs, and Corporate Social Responsibility. Some of these concepts are closely related to this project:

- Smart Factory. By equipping the manufacturing process with "smart" technologies such as sensors, actors, and autonomous systems, we develop a so-called "Smart Factory" which is autonomously controlled [LCW08].
- Cyber-physical Systems. By merging the physical and digital levels, we create systems where the physical and digital counterparts cannot be differentiated. From this concept the term "Digital Twin" emerges.
- Corporate Social Responsibility. There has been a significant increase in focus on sustainability and resource-efficiency of the design of manufacturing processes. These factors have become fundamental for products to succeed.

The last mentioned concept, Corporate Social Responsibility, is a concept which is mentioned across many different papers relevant to Industry 4.0. As mentioned in the introduction, Ghobakhloo [Gho20] describes how the digital connectedness which would result from Industry 4.0 would help with resource efficiency and waste reduction. Another important thing Ghobakhloo mentions is how the same technologies responsible for the increasing skill gap across all industries, can also continue to bridge it. The on-the-job training can be significantly improved by the usage of technologies such as AVR, smart apps, and data analytics tools.

2.1.1 Digital Twin

In the previous section we talked about several fundamental concepts and design principles of Industry 4.0. One of these design principles is virtualization. Virtualization involves the conversion of data from the physical world into a digital environment [Gho20]. This principle is related to the fundamental concept of Cyber-physical systems that Lasi [LFK+14] mentions. In Cyber-physical systems we have a physical and digital counterpart, the Digital Twin is described in the current section, the "physical twin" is described in section 2.2.

First let's define the term Digital Twin, Singh [SFH+21] provides the following definition of Digital Twin: "One thing that binds most definitions of DT other than being a virtual representation of a physical object is the bidirectional transfer or sharing of data between the physical counterpart and the digital one, including quantitative and qualitative data (related to material, manufacturing, process, etc.)". Singh [SFH+21] furthermore mentions some tasks a digital twin can perform with this data: Design and validation of new or existing product/process; Simulate the health conditions of physical twin; Predict the performance of physical twin;.

Now that we have the definition of the term, we will classify the Digital Twin for this project. Singh [SFH+21] describes how Digital Twins can be classified according to certain criteria. These criteria include: Creation Time of the Digital Twin; Level of Integration; Application; Hierarchy; Level of Maturity;.

To provide a better insight to the Digital Twin we will be developing in this project, we will classify the desired Digital Twin according to each of these criteria. First is the creation time of the Digital Twin. This criteria classifies a Digital Twin according to whether or not the Digital Twin was developed at the designing phase of a project or after the product is ready. In our case, the Digital

Twin is developed after the production of the product. If the Digital Twin is developed during the designing phase, it can be put through several tests to determine if the Digital Twin exhibits the expected behavior.

Second criteria is the level of integration. This criteria classifies the Digital Twin on the exchange of data between the Physical Twin and the Digital Twin. It defines three instances: Digital Model, the data is only manually exchanged and therefore any changes in either of the twins are not reflected in the other one; Digital Shadow, the Physical Twin automatically sends data to the Digital Twin meaning that any changes in the Physical Twin can be seen in the Digital Twin but not vice versa; Digital Twin, both of the twins automatically send data meaning that any changes in either of the twins are reflected in the other twin;. The Digital Twin in this project can be classified as a Digital Twin according to this criteria.

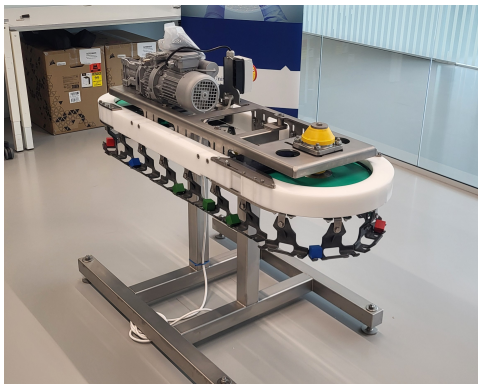
The third criteria is application. This criteria has two cases, a Predictive Digital Twin and an Interrogative Digital Twin. The first case involves a Digital Twin which predicts future behavior and performance, while the second case is used to interrogate the current or past state of the Physical Twin. In our case, the Digital Twin is Predictive.

The fourth criteria is the Hierarchy of the Digital Twin. This criteria defines three classes: Unit level, smallest participating unit in manufacturing which can be a piece of equipment, material, environmental factors; System level, amalgamation of several unit-level Digital Twins. Think of a production line, factory, etc.; System of Systems, multiple system-level Digital Twins interconnected together form a System of Systems. In this project the Digital Twin can be classified as a System level, as it involves multiple factors such as the location of the objects, the positions of the robot arms, speed and structure of the carousel, etc.

The fifth and final criteria is the Level of Maturity. This criteria classifies a Digital Twin based on the sophistication level of the Digital Twin. This accounts for both the sophistication of the data obtained, as well as the virtual representation. The data obtained is classified into: Partial, Clone, Augmented. In our case, the data of the Digital Twin is similar to a clone, where it contains all significant and relevant data. Whereas a Partial would only contain a small number of data points, and the Augmented would include all data a Clone would have as well as historical data.

2.2 Physical Twin

Now that we have an understanding of the classification of the Digital Twin of this project, we will look at the physical counterpart, also known as the Physical Twin. The Physical Twin is located in the Physical Twin Lab of SAILS (Society, Artificial Intelligence and Life Sciences). The purpose of the Physical Twin is to provide an industrial-like setup to perform research on. The issue with research on these industrial setups is that often they have already been implemented in the process of production. Performing research on a setup which is already part of production can be quite challenging, which is why this project aims to deliver a new way to easily perform research on setups without such challenges.



(a) The rotary carousel of the setup

(b) The robot arms used in the setup, ReBeL cobot 6 DOF

Now let's look at the specifications of the setup. It consists of a rotary carousel, see figure 1a, which is able to move objects by rotating. Alongside the carousel, we have two robot arms, see figure 1b, each of which serve a different purpose. One robot arm is the "Receiver" and the other is the "Sender". In an actual industrial setup, these robot arms would contain the necessary components to make X-ray images of the objects on the carousel. And thus one arm acts as the "Sender", which would send out radiation, the other arm acts as the "Receiver", which would receive the radiation sent out by the "Sender". In the Physical Twin lab, these X-ray images are simulated.

2.3 Unity3D

To simulate the expected behaviors in the Digital Twin, we require several tools and programs. For the movement of the carousel, visualization, physics interactions, as well as several other purposes, we use Unity3D. For the movement of the Robot Arms, which can be quite complex and dynamic, we use the ML-Agents toolkit to train a model using reinforcement learning within Unity. Another part of the simulation are the simulated X-ray images. Computing these simulated X-ray images is done by using the ASTRA toolbox.

Unity3D is a game engine developed by Unity Technologies. The program has been widely used as it is considered to be easy to use for beginners. Another reason it is popular is because it supports a wide variety of platforms, including desktop, mobile, console, augmented reality, and virtual reality. Aside from video games, Unity3D provides a great platform for machine learning due to the ML-Agents package. Unity3D has also been used for industries other than the gaming industry. These industries include filmmaking and automotive.

In Unity3D we divide parts of a game or the entirety of it within scenes. Scenes can be used as levels, sections, or contain the entire game. In a scene we can have GameObjects, these GameObjects can be configured using components. For example, the robot arms in our setup consist of a

hierarchy of multiple GameObjects, with each GameObject representing a part of the robot arm. The components assigned to the GameObjects can be things such as: Colliders, which allow the GameObject to check for collision; Rigidbodys, which allow for realistic physics simulations of the GameObject; Scripts, to define behavior of one or more GameObjects in a scene;.

2.3.1 ML-Agents

ML-Agents [JBT+20], which is short for Machine Learning Agents, is an open-source toolkit developed by Unity Technologies. The toolkit is available as a package in the Unity Package Manager, which is a conventional way to share code, assets, or even whole projects in Unity. The ML-Agents toolkit allows users to use Unity environments to train intelligent agents using techniques such as reinforcement learning or imitation learning. Because Unity allows for the development of complex and realistic environments, the toolkit is greatly beneficial for the research of AI.

Training the agents can be done using either one of the implemented Deep Reinforcement Learning Algorithms, or by adding your own custom training algorithm. The implemented Deep Reinforcement Learning Algorithms include PPO [SWD+17], which is used most commonly, SAC [HZAL18], and MA-POCA [CTB+22].

Another benefit of this toolkit is the support for training single-agent, multi-agent cooperative, and multi-agent competitive scenarios. As well as training using multiple concurrent Unity environment instances. Because of these benefits, the toolkit has been used by many researchers, as well as game developers.

2.3.2 ASTRA

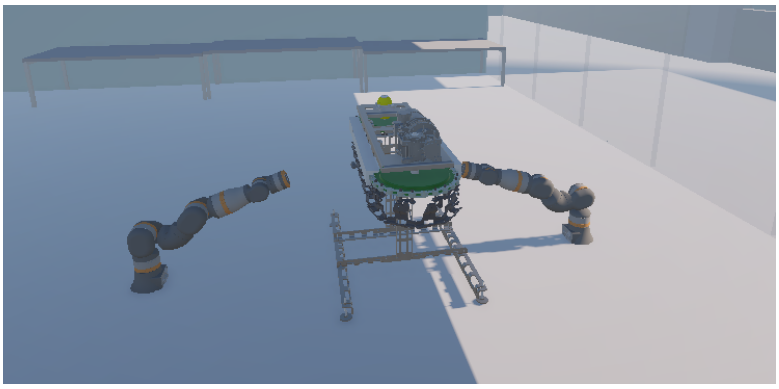
The ASTRA toolbox [vAPC+16] is an open-source software package which provides a collection of tools for tomographic projection and reconstruction. The toolbox is accessible through a Python interface, using the CUDA language to offload the core computations. This allows for fast computation of large-scale tomographic data. In our case, we use the toolbox to simulate the X-ray images for the Physical Twin. These simulated X-ray images are computed using data obtained from previously made real X-ray images. In conclusion, the ASTRA toolbox provides us a fast and efficient way of simulating the X-ray images for our setup.

3 Methodology

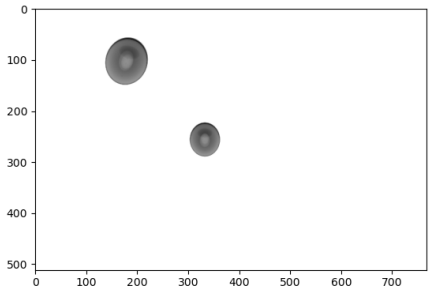
In the Background section 2 we discussed the purpose of the Physical Twin. In this section, we will discuss how we developed the Digital Twin by describing the expectations, whether or not any problems were encountered during development, and what solutions we used for our approach.

To develop the Digital Twin, we have to simulate both the environment and the behaviors of its components. Simulating the environment consists of providing an accurate visual representation of the environment in 3D. This visual representation contains the 3D modeling of the robot arms and the carousel. These 3D models both have their own requirements, as well as difficulties. For example, the 3D model for the robot arms requires to be separated into parts so it can have rigging that allows for movement similar to its physical counterpart.

The expected behaviors of the Digital Twin include the movement of the carousel, the movement of the robot arms, and computing the simulated X-ray images. The carousel is expected to be able to transport objects, so we have to ensure the interactions between the moving carousel and objects are accurate in terms of physics. The movement of the robot arms are complex and dynamic, because of these aspects we decided to train an intelligent agent using reinforcement learning for the movement of the robot arms. Lastly are the simulated X-ray images. For this, we have to decide on a sufficient system architecture for the interactions between the Physical Twin and Digital Twin. This system architecture should be able to broadcast the simulated X-ray images, so any client, such as the Physical Twin or Digital Twin, can acquire the simulated X-ray images by providing certain data. See figure 2a for a preview of the setup, as well as figure 2b for an example of the simulated X-ray images.



(a) Preview of the setup.



(b) Simulated X-ray image of two apples.

Figure 2: Preview of the setup and an example of a simulate X-ray image.

3.1 3D Modeling

For the environment of the Digital Twin, we require 3D models of the carousel and the robot arms. These models each have their own requirements and difficulties, the 3D model for the robot arms has to be rigged with certain parts attached to bones so it is able to move like its physical

counterpart. However, the obtained CAD model for the robot arms was not divided into parts and instead consisted of a single whole piece, which means it has to be separated into parts.
The 3D model for the carousel has other requirements, with its main focus on being accurate while also keeping it from being too complex. The 3D model obtained for the carousel was incredibly detailed, whereas every screw was a different part. This means that the 3D model for the carousel has to be simplified.

3.1.1 ReBeL cobot 6 DOF Model

For the 3D model of the robot arms, we require a rigging so it is able to be moved like its physical counterpart. To achieve this, we integrate bones into the model. By assigning parts of the 3D model to a bone, any movement of that bone influences the assigned parts. In addition, the bones are connected to each other in order, meaning that the movement of a bone also influences the bones that come after it. Thus, in our case these bones would represent joints, where any rotation in joint 2 means that positions of joints 3, 4, 5, and 6 are changed. This behavior can be observed in 3.

The 3D model for the ReBeL cobot 6 was obtained from the IGUS CAD Library [Gmb]. As the reference suggests, the 3D model was a CAD model, also known as a Computer-aided design. These models are typically used to create 2D or 3D models for manufacturing products, and they are widely used in the manufacturing business. Currently this 3D model was a single whole piece, while we needed the model to be divided into parts. Because of this, the model needed some adjustments to be ready to use in Unity3D.

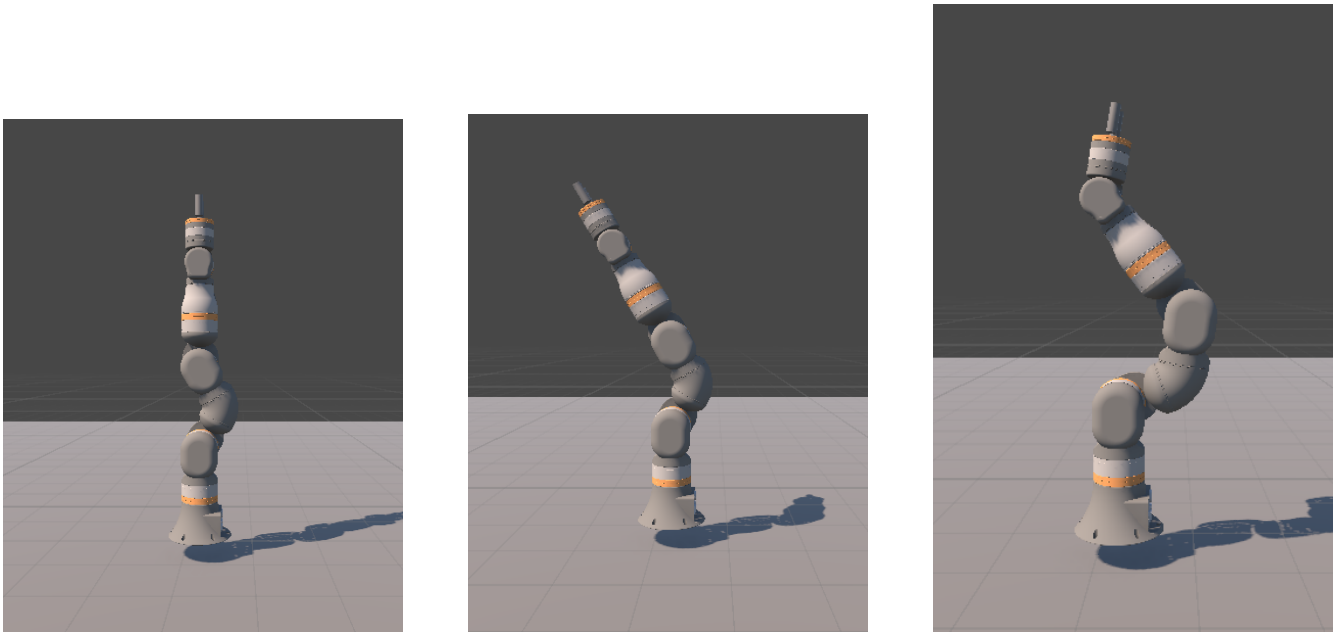
After exporting the model to the fbx format, which stands for FilmBox, we first separated the model into parts so it was ready for the rigging. For this we used the software Blender [Fou]. We separated the model into groups by assigning the polygons of each part to a vertex group. The polygons were grouped by parts and by joints, whereas parts were the sections which were connected to the joints.

Now that the 3D model was divided into parts, it was ready to be rigged. For the rigging, we added a bone for each joint. Each bone was connect to the head of the previous one, which would result in the behavior we express before, where movement of bones would influence the bones that would come after it. The previously mentioned vertex groups, which we consider to be the parts and joints, are then assigned to the correct bones.

3.1.2 Carousel Model

The 3D model for the carousel had a similar requirement as the 3D model for the robot arms. To simulate the movement of the carousel, the 3D model had to be separated into parts. Namely, the parts that would actually move and transport other objects, the "Holders", had to be separated so any movement of these parts would not influence other parts of the carousel.

For the 3D model of the carousel we were able to obtain the original CAD model provided by the manufacturers of the carousel. Due to the nature of the model being a CAD model, the original file format was not supported by Unity3D. The model was converted to the FBX format.



(a) Arm in resting position. No rotated bones.

(b) Arm with a slight rotation in bone 3.

(c) Arm with a rotation in bone 2, 3, and 5.

Figure 3: Robot arm in several configurations.

Unlike the 3D model for the robot arms where it was a single whole piece, the 3D model we obtained for the carousel was incredibly detailed and complex. However, in this project this level of detail was not necessary for all parts. Thus the 3D model was simplified by separating the parts relevant to the movement, and reducing the polygon count of these irrelevant parts, to reduce the complexity of the model.

3.2 Implementing the Digital Twin in Unity3D

The Physical Twin has multiple behaviors that we have to simulate in the Digital Twin. The first one is the movement of the carousel. The carousel has multiple "holders" on which objects can lie on and be transported. The objects on these "holders" have physics applied to them, thus the "holders" have to account for this in their movement. The speed of the movement should also be adjustable.

Second is the movement of the robot arms. The robot arms act as a source and a detector for the X-ray imaging. They are able to make X-ray images of the objects that are on the "holders" of the carousel. By keeping track of an object, which means having both of the arms look directly at the object, they are able to continuously make X-ray images of the object. This movement can be quite complex, which is why we will train an intelligent agent using reinforcement learning to perform this behavior.

Third is the X-ray imaging of the objects. The X-ray imaging is done using the ASTRA toolbox. The ASTRA toolbox provides tools for tomographic projection and reconstruction. The computing

of the X-ray images is done on a server. This server is able to take requests from any client to compute an X-ray image depending on the provided data. For future work, a different system architecture is proposed. In the proposed system architecture the server will have a persistent world state, where the clients would only provide updates regarding the data, and the resulting X-ray images would be broadcasted.

3.2.1 Carousel Movement

As mentioned before, the carousel has multiple "holders" on which objects can be transported. These objects are subject to external forces, which is important to keep in mind for the movement of the "holders". Furthermore the "holders" move the same in the Digital Twin as they would for the Physical twin, that is, they loop around the predetermined path with an adjustable speed.

One of the most common ways to move objects along a predetermined path in Unity is by using splines. A spline consists of knots which are connected to each other to form a path. The path formed between the knots is a bezier curve that is automatically calculated from the neighboring knot positions. The spline, together with the carousel, can be seen in figure 4. We used the chains of the carousel as a guidance to determine the positions of the knots.

Now that we have the path for the holders to follow, we require a script to simulate this movement. Normally with splines, each object starts the movement at the start of the spline. In our case, the objects start at different points of the spline. To account for this, we start by getting the nearest point on the spline for each holder in the Start() function. This point is then used as an offset for the holder. As you can see in figure 4, the holders rotate at certain points. Because of this, we also use an offset for the rotation.

```
1   void Start(){
2       Holder = GetHolder();
3       WorldPosition = Holder.Position.LocalToWorld();
4
5       Spline = GetSpline();
6
7       // T represent how far this point is on the progression of the spline.
8       t = GetNearestPointOnSpline(WorldPosition);
9
10      PositionOffset = Spline.length() * t;
11      RotationOffset = GetRotationAtPoint(t);
12  }
```

Listing 1: Pseudocode of Start().

To simulate the movement of the holders we require a script that handles physics correctly, loops the predetermined path, and handles forces correctly. For this, we use the Update() function. The Update() function is called once per frame in Unity3D. We used the current progress of the holders on the path, as well as the movement speed, to calculate what the progress of the holders should be for the next frame. To make the holders loop the path, we modulo this progress by the length of the spline. By normalizing the progress value, we can calculate the world position associated to this point, as well as the rotation of the holder at that point. The world position of this point is then used with the MovePosition function, which is a method function of the Rigidbody and

moves the holders while accounting for physics, and MoveRotation, which has similar functionality to MovePosition.

```
    void Update(){
        SplineLength = Spline.length();
        // TimeElapsed returns the time elapsed between the current and
    previous frame.
        DeltaDistance = MovementSpeed * TimeElapsed();

        t = (t + DeltaDistance) % SplineLength;
        NormalizedT = t.normalize();

        WorldPositionT = Spline.WorldPositionForPoint(t);
        RotationT = Spline.GetRotationForPoint(t);

        MovePosition(WorldPositionT);
        MoveRotation(RotationT);
    }
```

Listing 2: Pseudocode of Update().

Finally, to make use of the physics engine we have to assign each of the holders a Rigidbody. The Unity documentation reference for the Rigidbody component: "A Rigidbody provides a physics-based way to control the movement and position of a GameObject. Instead of the Transform properties, you can use simulated physics forces and torque to move the GameObject, and let the physics engine calculate the results.". In our Digital Twin, we decided to set the isKinematic property of the Rigidbody's of the holders to true. This makes it so that external forces do not affect the position or rotation of the holders.

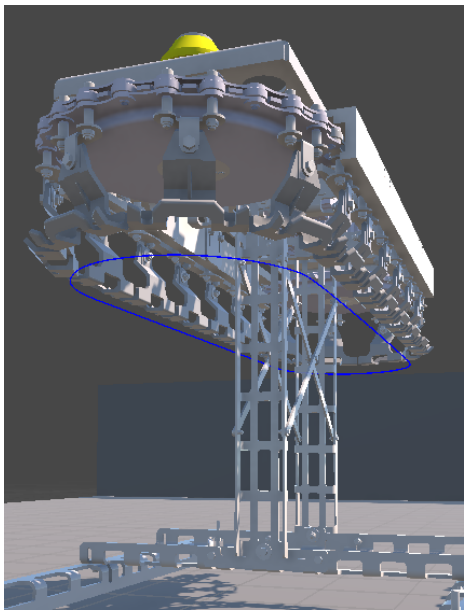


Figure 4: The spline, which is the blue line, together with the carousel. The height of the spline is irrelevant, as we only use the other coordinates to calculate the positions.

3.2.2 Reinforcement Learning for the Robot Arms

The movement required for the robot arms is quite complex and dynamic. They should be able to actively track objects, which means that both arms should look directly at the object, to make the X-ray images. Furthermore, the arms may have to switch to another object on command, as well as perform other tasks such as grabbing an object. Because of these requirements, options such as inverse kinematics are ruled out due to the complexity of these requirements. The option we opted for was to train a model using reinforcement learning.

To start with the reinforcement learning, we have to set up the environment in which the model will be trained. In this environment can be one or more Agent objects, in our case, the Agent objects are the robot arms. As mentioned before, the robot arms have to keep track of an object, which means we also have to add this object to the environment. The Agent objects have certain behavior parameters we can configure. The Vector Observation Space value is determined by the amount of observations an Agent makes about the state of the environment, which we decide ourselves. Another parameter is the Actions parameters, this determines the amount of actions the model has to return. For our robot arms, the only actions performed are the rotation of the joints, which mean the Actions parameter is set to 1 action per joint.

During the training of the model, it can request decisions either manually or automatically. These decisions determine the Actions received by the model. We can manually request a decisions every step, however, requesting a decision every step can waste computation and generate redundant data. The ML-Agents toolkit provides a script called DecisionRequester, which allows us to automatically request decisions for an instance at regular intervals. Therefore, we set the frequency of decisions, known as the DecisionPeriod in the DecisionRequester script, to 5. Meaning that the Agent will request a decision every 5 Academy steps.

The Actions the model receives have to be converted into actions in the environment. Furthermore, the observations of the environment have to be collected, and the state of the environment should be reset on the beginning of each episode. To perform these tasks, we require a class for the Agents in our environment. This class is made by overriding several functions of the ML-Agents toolkit. These include:

- OnEpisodeBegin(), which is called on the start of every episode.
- CollectObservations(), which is called for every step to collect the vector observations
- OnActionReceived(), which is called for every step to specify the Agent behavior on every step, based on the provided Action.

On the beginning of each episode, the state of the environment should be reset and prepared for a new episode. We do this by resetting the rotations of the joints of the robot arms, as well as prepare the location or movement of the object the Agents have to track. The state of the environment is used by the Agent to make decisions, this state is observed and collected using Observations. The Observations collected include: the rotation of each joint, position of the target object, position of the end effector of the robot arm, and the position of the target object relative to the end effector. The actions received by the Agent have to be converted into actions

that can be performed in the environment. For this, each Action is seen as a rotation value for a joint of the robot arm. So, we loop through the Actions and apply these values to the relative joints.

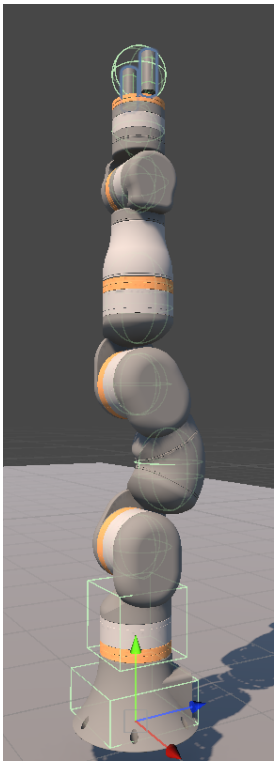
To simulate the expected behavior of the robot arms, we have to determine the reward models. These reward models determine what and when the Agents are rewarded. The two robot arms both serve a different purpose, yet their behavior is quite similar. Both of them need to accurately track the target object to make an X-ray image. When a target is moving across the carousel, the Agent should keep track of the object for as long as possible while also minimizing the movement, which could result in a smooth and efficient motion. Considering how both of the robot arms will exhibit similar behavior, only one model will be trained for the source, with the detector robot arm mirroring the movement of the source robot arm.

For the tracking of a target object, the end effectors of the robot arms should have a determined distance to the target object depending on which side of the carousel it is on. To train the model for this behavior, we attach an object to the end effector that is at the determined distance. Using the location of this attached object, we calculate the distance between the object and the target using the Vector3.Distance function, which calculates the euclidean distance between two points. We then calculate the clamped distance using Mathf.Clamp01, this returns 0 if the distance is negative and 1 if the distance is bigger than 1. After that we subtract the clamped distance from 1. The range of this reward is 0 - 1, if the attached object is close to the target, the calculated distance will be low and thus the reward will be high. Depending on what part of the training the Agent is at, it may end the episode with a big reward if the distance gets below a determined threshold.

During the tracking, the distance should remain as close as possible. To encourage this, we reward the model when the calculated distance is less than the previously calculated distance. We save the previously calculated distance to determine distanceΔ. If distanceΔ is positive, the end effector has moved closer to the target and we give a small reward such as 0.01 to reward exploring. However, we also penalize the Agent if the distanceΔ is negative, which means the end effector has moved away from the target. The movement of the robot arm should be smooth, having no unnecessary movements. Encouraging this is done by applying a small penalty of multiplying the sum of the absolute values of the Actions by 0.00001.

The Digital Twin will express behavior accurate to the Physical Twin. However, within Unity certain things are able to happen that are not possible for the Physical Twin. That is, objects are able to phase through each other. In Unity the robot arms can clip through the floor, carousel, target object, etc. The Agent is heavily discouraged of doing this by applying a large penalty whenever this happens. The detection of this is done by adding mesh colliders to the environment. These colliders should represent the objects they are attached to as accurate as possible. However, complex mesh shaders often require heavy computing, which is why developers often tend to use simpler shapes to determine the colliders of an object. For the robot arms we used capsule colliders. See figure 5a for the robot arm and its colliders.

An important aspect of the training process are the hyperparameters. The hyperparameters, as well as the training process as a whole, were heavily inspired by Matulis [MH21]. The hyperparameters are shown in figure 5b. For this agent, we decided to use PPO [SWD+17] as the trainer, with a



(a) The robot arm with the colliders highlighted in green.

```
behaviors:
  RobotArmTD3:
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 10240
      learning_rate: 2.0e-4
      beta: 1.0e-3
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    max_steps: 10000000
    time_horizon: 64
    summary_freq: 1000
```

(b) The hyperparamaters of the training process.

Figure 5: The robot arm for training and the hyperparameters.

batch size of 128 and learning rate of 2.0e-4. PPO is commonly used as a trainer as it is capable for many different tasks and environments. An alternative could have been SAC, however this trainer took significantly longer to train, which is why we did not choose it. Within ML-Agents it is possible to train multiple instances at the same time. This significantly speeds up the process as they share their experience. Because of this, we set up an environment in unity with 12 robot arms training at the same time. See figure 7 for an image of this "Robot Farm".

Due to the size of the robot arms, as well as training complexity, we assume that the robot arms will only image the objects on certain sections of the carousel, which are both of the straight sections. The training of the model consists of randomly placing a target within this section, with randomly generated offsets for all directions to make the model more robust. See figure 6 for an example of the section in which the objects are scanned.

3.3 System Architecture

An important part of the Digital Twin is the simulation of the X-ray images. The current system architecture was provided by Dr. W.J. Palenstijn. In our current system architecture, we have a server which accepts a set of coordinates, simulates a projection, and replies with the simulated X-ray image. Our Digital Twin serves as a client who sends the set of coordinates, and receives the simulated X-ray image. In this section, we propose an updated system architecture which

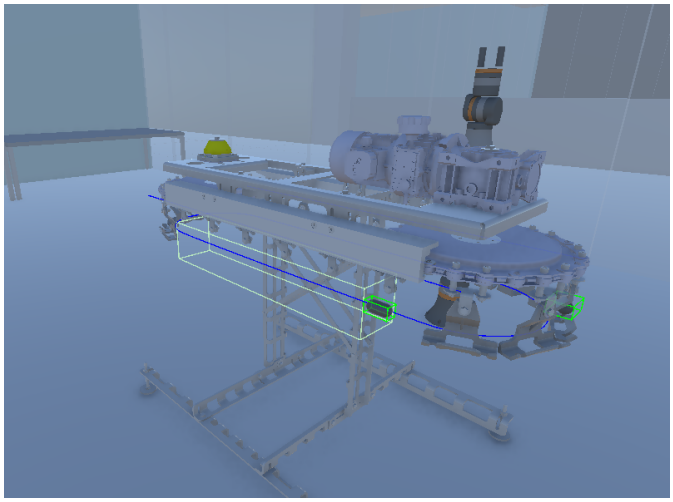


Figure 6: Segment of the carousel in which the objects are scanned. During training the "holders" of this segment are temporarily removed. Within this segment the target is randomly placed with randomly generated offsets for all directions.

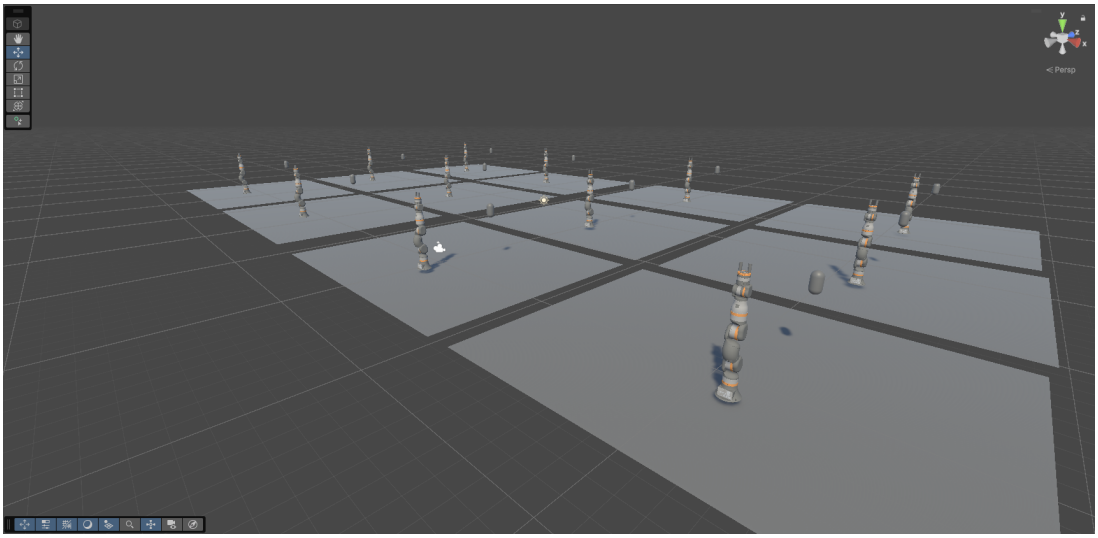


Figure 7: The "Robot Farm" used for the training.

aligns more with the requirements of the Physical Twin lab. The design of the proposed system architecture was designed in collaboration with Dr. W.J. Palenstijn. Ideally, multiple clients should be able to acquire the X-ray images. This way, we can have several clients who each serve a different purpose. For example, there can be a client which synchronizes the simulation with the Physical Twin using sensors, actuators, or other interfaces. Another client could be an automated experiment, by programmatically moving objects and running test sequences to gather various output images. It would also be possible to have a client to remotely track the current simulation state and output images. Our Digital Twin would also serve as a client, allowing for experiments with an accurate simulation of the expected behaviors of the setup. For this, we propose an architecture with a server at its core. This server will have a persistent world state, which can accept coordinate updates from

multiple clients and broadcast the resulting simulated X-ray images.

3.3.1 ASTRA

As mentioned in the background, the ASTRA toolkit provides a highly efficient and highly flexible set of tools for tomographic projection and reconstruction. One of the goals of the Physical Twin setup is to make X-ray images of the objects on the carousel. To simulate this behavior in our Digital Twin, we make use of the ASTRA toolkit. Using a template for the client code provided by Dr. W.J. Palenstijn, we wrote the client code for our Digital Twin which allowed us to interact with the current system architecture mentioned before.

The client of our Digital Twin sends the following data of objects:

- Name: The name of the objects in the Unity3D scene.
- Dimensionality: There are three types of objects in our scene, a source, detector, and food objects. The dimensionality of a source is 0D. The dimensionality of the detector is 2D, this can be seen as a plane. The dimensionality of an object is 3D.
- Position: The position of the object in the Unity3D scene.
- Bounding box: The bounding box of the object depends on the dimensionality of the object. A bounding box is a box which perfectly encloses an object. For the source, there is no bounding box as it is 0D. The detector has a bounding box of 2 vectors, as it is 2D. And finally, the food objects have a bounding box of three vectors, because it is 3D.

Currently the server is hosted on the same machine as the Digital Twin, and thus the resulting X-ray image is saved onto the file system of this machine. This image can then be used in Unity3D as a texture for a UI image element. This element has a script which checks with a certain interval if there is a new image, and if so replaces the current one.

4 Experiments with the Robotic Arm.

Developing the Digital Twin consisted of several tasks, firstly the visualization of its components, secondly the simulation of the expected behaviors. The task of simulating the robot arm movement was only one of many. Because of this, extensive experimentation was not the main focus of this project. However, the locations and orientations of the robot arms determine the resulting simulated X-ray images of ASTRA. Considering how the simulated X-ray images from ASTRA is one of the expected behaviors, the performance of the model for the movement of the robot arms is significant to the performance of the Digital Twin. In this section we perform an experiment to determine how effective the model for the movement of the robot arms is in tracking a target.

4.1 Experimental Setup

One of the most important parts of the Digital Twin is to have an accurate representation of the expected behaviors. One of these behaviors is the movement of the robot arms. The robot arms

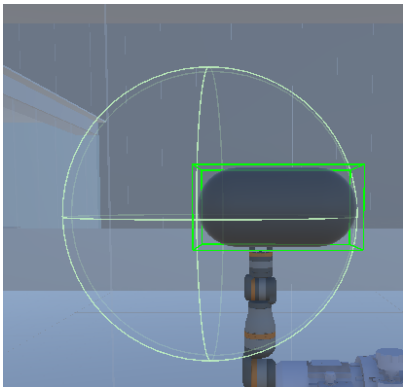


Figure 8: Scale 1 for the spherical collider.

represent a source and a detector for X-ray imaging. Thus, the locations of the source and detector determine the resulting X-ray image. To ensure accurate X-ray images, the source and detector should track the selected target. We determine the accuracy of tracking as follows:

- We move a target across the same section used during training, see figure 6.
- We use the same size target for the experiments as we did for the training.
- The speed of the target has 2 configurations, 1 is the same speed as the target would move in the Physical Twin, 2 is two times the speed of the Physical Twin.
- The accuracy is determined by setting a spherical collider at the end effector. We keep track of two timers: the total amount of time the target takes to move across the section; and the amount of time the target is within the collider.
- The size of the spherical collider has 3 configurations, scale 1 means that the radius of the spherical collider equals the length of a target. See figure 8. Scale 0.5 means the radius equals 0.5 the length of a target. Scale 2 means the radius equals 2 times the length of a target.
- The experiment for each configuration is performed 10 times, from which we calculate the mean.

4.2 Results

The results of the experiment can be seen in table 1. In the table, we can clearly see that at scale 0.5, the model performs poorly. During the experiments, as well as regular performance, the model can be observed to be moving jittery, making lots of unnecessary movements. We also observed the model having the tendency to move the end effector down significantly at irregular intervals, severely affecting its performance. However, the result also indicates the model being robust to changes in speed, seeing how the difference in performance between the two different speeds is less than expected. Nonetheless, the model leaves lots of room for improvement.

	Scale 0.5	Scale 1	Scale 2
Speed 1	10.0	42.7	71.4
Speed 2	8.6	36.8	67.2

Table 1: The percentage of time in which the target is within the collider.

5 Functionalities of the Digital Twin

This section features an overview and description of the several functionalities implemented in the Digital Twin. This includes the movement of the carousel, the movement of the robot arms and selecting a target, the astra interface, and the custom camera grid.

The Digital Twin developed by this project includes several functionalities. The first functionality is the movement of the carousel; the "holders" of the carousel move at a configurable speed along the spline. As said, this speed is configurable and can be set in the editor.

The second functionality is the robot arms. The robot arms are able to track a manually selected target, or automatically set their target. Manually setting the target is done by setting the target field in the editor. Automatically setting the target is done by using a script attached to a collider. This collider is the same as the section in figure 6. Whenever a new target object enters this collider, the script selects this target as the target object for the robot arms.

The third functionality is the ASTRA interface. At the top right of the screen is a RawImage UI element that automatically updates its texture. This texture is the resulting X-ray image from the ASTRA server. In the current setup, the ASTRA server is hosted on the same machine as the Digital Twin, therefore the resulting X-ray image is saved onto the machine, which is why the RawImage element has a script that loads an image at a predetermined path as its texture. See figure 9 for an example of the robot arms making a simulated X-ray image.

The fourth and final functionality is the camera grid. When the Digital Twin is running, it is possible to either walk around manually, or press the "Tab" key to open up a camera grid. For this camera grid, it is possible to either place a camera at a static position, or dynamically move the camera around according to a script, as well as attach the camera to an object such as a target. See figure 10 for an example of a camera grid.

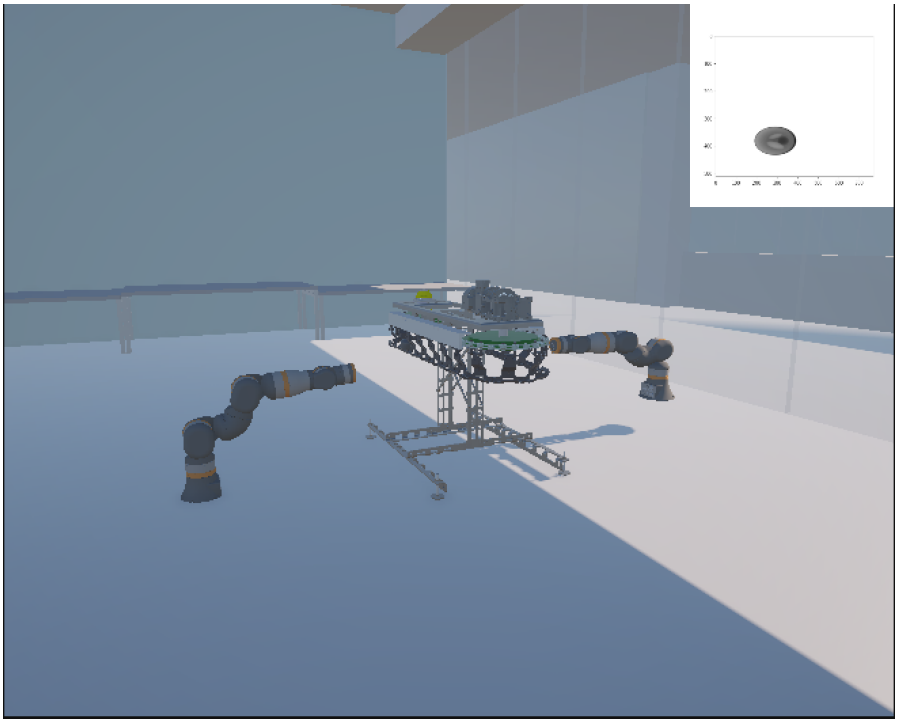


Figure 9: The robot arms tracking a target. The resulting X-ray image can be seen at the top right.

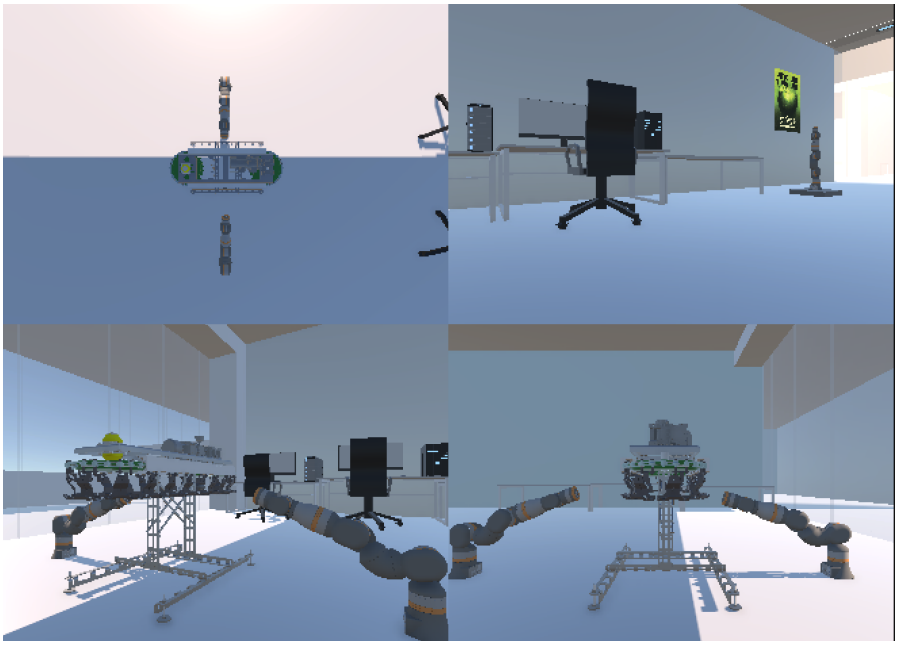


Figure 10: The custom camera grid that is opened by pressing Tab. Top-left is a top down view; Top-right is a camera attached to a target. The bottom two are statically placed cameras.

6 Discussion

Training an intelligent agent using reinforcement learning proved to be quite challenging. We observed how seemingly small changes in the reward model could significantly affect the outcome produced by the training. Even near the end of the project, we were still trying different reward models. The current reward models prove to be quite successful, but that does not mean there are no better alternatives.

Another limitation was the lack of data to train the intelligent agents on. Preferably, we would have used data generated by the Physical Twin to train them. This data would consist of the recorded paths that an object took and their speed. However, gathering this data would have been a lengthy process. One of the problems would be to convert the locations of the objects and paths from the Physical Twin to the Digital Twin.

In this work, we decided to have the robot arms move by changing the rotation value of each individual joint. However, this is not the most realistic approach. A problem with the current approach is that it does not account for the physics that would be applied to the joints and parts of the robot arms. A more realistic approach would be to assign the different parts of the model a mass value, and rotate the joints by adding velocity. Unfortunately we did not have the necessary data to make a correct assumption regarding the weights of the individual parts, therefore we decided to opt for the current approach.

There are multiple opportunities for further research. One of these include researching whether or not a different trainer could improve the performance of the model. For this, we propose either using the SAC algorithm which is already implemented in ML-Agents, or implementing a custom trainer based on the TD3 algorithm [FvHM18]. TD3, also known as Twin Delayed Deep Deterministic Policy Gradient, is an reinforcement learning algorithm that builds on the foundation of the DDPG algorithm [SLH+14]. Research by Mock, J. and Muknahallipatna, S [MM23] showed that TD3 and SAC both learn significantly quicker than PPO. However, it also demonstrated how TD3 does not perform well with minimal state spaces. Nonetheless, research on these algorithms could definitely prove to be fruitful.

Another opportunity would be to improve the realism of the Digital Twin. So, instead of the current implementation of movement for the carousel, we take an approach that would involve actually use the gears and chain on the model to simulate the movement. However, while Unity has a great physics system, this implementation can be quite difficult to get correct. But this alternative approach would allow for more realistic behavior of the carousel, such as the movement being more bumpy or snappy, giving a more accurate representation of the behavior the physical twin would exhibit.

7 Conclusion

We started this thesis by exploring the necessary background knowledge for this project. This involved the definition of the Industry 4.0 concept, as well as the sub-concept Digital Twin. Then we discussed the physical counterpart of the Digital Twin, which we call the Physical Twin. After that, we discussed the technologies used for this project. This consisted of Unity3D, the cross-platform game engine; ML-Agents, the toolkit used for training the models in Unity3D using reinforcement learning; ASTRA, the toolkit which was used to implement a simulation of the images that would be produced from the setup;.

Then we describe the methodology for this thesis. First, we discussed how we acquired and adjusted the 3D models to be ready for use in the Unity3D environment. Secondly, the implementation of the expected behaviors. There were 3 behaviors expected, the holders of the carousel were to move while making use of the physics engine; the robot arms were to keep track of the objects which were transported on the carousel; the ASTRA implementation was to provide a simulation of the expected images;. The movement of the carousel was simulated by making use of the spline package within Unity3D. The movement of the robot arms was simulated by training a model using reinforcement learning. The simulated X-ray images were computed using an ASTRA implementation where the Digital Twin sends data to a server which responds with the resulting X-ray image. Thirdly we describe the experimental setup. The experimental setup consisted of measuring the performance of the model, keeping track of its accuracy. The results of these experiments are analyzed, after which we provide an overview of the Digital Twin developed by this project.

The key contributions of this project include the designing of the Digital Twin, the experiments with reinforcement learning for the movements of the robotic arms, and providing an interface with the broader eco-system of ASTRA to simulate the X-ray images.

References

[CTB+22] Andrew Cohen, Ervin Teng, Vincent-Pierre Berges, Ruo-Ping Dong, Hunter Henry, Marwan Mattar, Alexander Zook, and Sujoy Ganguly. On the use and misuse of absorbing states in multi-agent reinforcement learning. *RL in Games Workshop AAAI 2022*, 2022.

[Fou] Blender Foundation. blender.org - Home of the Blender project - Free and Open 3D Creation Software.

[FvHM18] Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

[Gho20] Morteza Ghobakhloo. Industry 4.0, digitization, and opportunities for sustainability, 4 2020.

[Gmb] KiM GmbH. IGUS® CAD.

[HZAL18] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor, 2018.

[JBT+20] Arthur Juliani, Vincent-Pierre Berges, Ervin Teng, Andrew Cohen, Jonathan Harper, Chris Elion, Chris Goy, Yuan Gao, Hunter Henry, Marwan Mattar, and Danny Lange. Unity: A general platform for intelligent agents. *arXiv preprint arXiv:1809.02627*, 2020.

[LCW08] Dominik Lucke, Carmen Constantinescu, and Engelbert Westkämper. *Smart Factory - A Step towards the Next Generation of Manufacturing*, pages 115–118. 01 2008.

[LFK+14] Heiner Lasi, Peter Fettke, Hans Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business and Information Systems Engineering*, 6:239–242, 8 2014.

[MH21] Marius Matulis and Carlo Harvey. A robot arm digital twin utilising reinforcement learning. *Computers Graphics*, 95:106–114, 2021.

[MM23] James Mock and Suresh Muknahallipatna. A comparison of ppo, td3 and sac reinforcement algorithms for quadruped walking gait generation. *Journal of Intelligent Learning Systems and Applications*, 15:36–56, 01 2023.

[SFH+21] Maulshree Singh, Evert Fuenmayor, Eoin P. Hinchy, Yuansong Qiao, Niall Murray, and Declan Devine. Digital twin: Origin to future, 2021.

[SLH+14] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, page I–387–I–395. JMLR.org, 2014.

[SWD+17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[vAPC+16] Wim van Aarle, Willem Jan Palenstijn, Jeroen Cant, Eline Janssens, Folkert Bleichrodt, Andrei Dabravolski, Jan De Beenhouwer, K. Joost Batenburg, and Jan Sijbers. Fast and flexible x-ray tomography using the astra toolbox. *Opt. Express*, 24(22):25129–25147, Oct 2016.