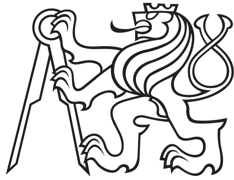
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Robotic Object Manipulation in Virtual and Physical Environment

Bc. Ondřej Švec

Supervisor: Ing. Tomáš Jochman
Field of study: Cybernetics and Robotics
May 2024

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Švec Ond ej**

Personal ID number: **483800**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Master's thesis details

Master's thesis title in English:

Robotic Object Manipulation in Virtual and Physical Environment

Master's thesis title in Czech:

Robotická manipulace s objekty ve virtuálním a fyzickém prost edí

Guidelines:

The aim of the thesis is to compare the possibilities of detection and localization of objects in a simulated and real environment, which will be implemented in a robotic cell. The simulated environment offers photorealistic real-time rendering that can be extended to include additional physical properties of the objects where the cameras will be simulated. Equivalent conditions will be prepared in the real workplace.

1. Get acquainted with the function of individual components and algorithms for object 6D position estimation.
2. Design and prepare a scene with objects, lighting, camera and other elements in the simulation environment. Select different objects to be manipulated and implement an algorithm to detect and locate them. Apply the same procedure in a real workplace.
3. Extend the simulation to include robotic manipulation of objects by using a camera to guide the robot. Implement robot guidance in the real workplace.
4. Test and compare the behavior of the algorithm in both environments.
5. Evaluate the algorithm performance for selected objects.

Bibliography / sources:

[1] Labbé, Yann, Lucas Manuelli, Arsalan Mousavian, Stephen Tyree, Stan Birchfield, Jonathan Tremblay, Justin Carpentier, Mathieu Aubry, Dieter Fox, and Josef Sivic. MegaPose: 6D Pose Estimation of Novel Objects via Render & Compare, arXiv, 2022, https://doi.org/10.48550/ARXIV.2212.06870.

[2] Hanzhi Chen, Fabian Manhardt, Nassir Navab, Benjamin Busam, TexPose: Neural Texture Learning for Self-Supervised 6D Object Pose Estimation, 2023, https://doi.org/10.48550/arXiv.2212.12902.

[3] Zhihan Lyu, Mikael Fridenfalk, Digital twins for building industrial metaverse, Journal of Advanced Research, 2023, ISSN 2090-1232, https://doi.org/10.1016/j.jare.2023.11.019.

[4] NVIDIA, Developer Guide Overview, 2024, https://docs.omniverse.nvidia.com/dev-guide/latest/index.html

Name and workplace of master's thesis supervisor:

Ing. Tomáš Jochman Testbed CIIRC

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **16.01.2024**

Deadline for master's thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Tomáš Jochman Supervisor's signature	prof. Dr. Ing. Jan Kybic Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

© VUT v Praze, Design: VUT v Praze, VIC

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt

Student's signature

© VUT v Praze, Design: VUT v Praze, VIC

Acknowledgements

I would like to dedicate a special thanks to my supervisor Ing. Tomáš Jochman for all the long evenings spent debating matters both involving and excluding the topic of this thesis. I would also like to thank all of my colleagues from the Testbed lab who made this place feel like home. I have met some of the most humble and down-to-earth people in this place. Sometimes we all wish we could appreciate the great moments before they pass. And while I am trying my best to acknowledge that these are some of the best years of my life, I know that the true realization has yet to dawn on me.

I would also like to thank my whole family for the support they have given me throughout all the years of my life and my studies. A special thanks goes to my mother, an amazing woman, who has played a pivotal role in my upbringing, always there when I need her the most, always there to keep me grounded; she truly is my rock, and my fortress. Thanks, mom.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 24 May 2024

. . . .

Abstract

Robotic manipulation using cameras is a critical advancement in the automation industry, enhancing the flexibility and efficiency of robotic systems. This thesis investigates the integration of camera-based robotic manipulation within virtual and physical environments, focusing on the capabilities of NVIDIA Omniverse to improve simulation accuracy. The study begins by choosing and implementing a 6D pose estimation algorithm using a static RGB camera. A robotic cell is designed and tested in both virtual and physical settings, with the virtual environment created using Process Simulate and KUKA.OfficeLite, and subsequently refined in NVIDIA Omniverse for high-fidelity rendering.

A major component of the thesis involves developing tools within NVIDIA Omniverse, a dataset generation tool for machine learning model training, and an interactive application serving as a Digital Model. The proposed modular architecture includes a high-level controller and a unified interface, ensuring scalability. Experimental results demonstrate that synthetic datasets can be used to effectively train object detection models. The consistency of performance across virtual and physical domains validates the digital model's authenticity. Recommendations for enhancing setup performance include using a higher zoom camera lens or redesigning the cell with a robot-mounted camera.

Keywords: robotic manipulation, industrial metaverse, digital twin, 6D pose estimation

Supervisor: Ing. Tomáš Jochman

Abstrakt

Robotická manipulace s využitím kamer představuje pokrok v automatizaci, který zvyšuje flexibilitu a efektivitu robotických pracovišť. Tato diplomová práce zkoumá integraci manipulace s objekty pomocí kamer ve virtuálním a fyzickém prostředí, přičemž se zaměřuje na schopnosti platformy NVIDIA Omniverse zlepšit přesnost simulací. Práce začíná rešerší a implementací algoritmu pro 6D odhad polohy pomocí statické RGB kamery. Robotická buňka je navržena a testována ve virtuálních i fyzických podmínkách, přičemž virtuální prostředí je vytvořeno pomocí nástrojů Process Simulate a KUKA.OfficeLite a následně rozšířené o simulaci v platformě NVIDIA Omniverse pro věrné a autentické vykreslování.

Hlavní součástí práce je vývoj nástrojů využívajících platformy NVIDIA Omniverse, nástroj sloužící ke generování datasetů pro trénink modelů strojového učení a interaktivní aplikace sloužící jako digitální dvojče. Navrhovaná modulární architektura se skládá z nadřazeného kontroleru a unifikovaného rozhraní, což zajišťuje škálovatelnost. Experimentální výsledky ukazují, že synteticky tvořené datasety mohou být využity pro efektivní trénování modelů pro detekci objektů. Konzistentní výkon napříč virtuálními a fyzickými doménami potvrzuje věrnost digitálního modelu. Doporučení pro zlepšení výkonu zahrnují použití kamery s větším přiblížením, popřípadě přepracování buňky s kamerou namontovanou na rameni robota.

Klíčová slova: robotická manipulace, průmyslový metaverse, digitální dvojče, odhad 6D pozice

Překlad názvu: Robotická manipulace s objekty ve virtuálním a fyzickém prostředí

Contents

1 Introduction 1

2 Background 3

2.1 Robot object manipulation 3

2.2 6D Pose estimation state of the art 4

2.2.1 Megapose6D 5

2.2.2 FoundationPose 5

2.2.3 CosyPose 6

2.2.4 Comparison and selection of 6D Pose algorithm 7

2.3 Digital Twin, Digital Model 8

2.4 Simulation software 9

2.4.1 Process Simulate 10

2.4.2 NVIDIA Omniverse Isaac Sim 11

2.5 Experimental Workplace Setup . 12

2.6 Testbed Dataset 13

2.7 Object detection model 15

3 Methodology 17

3.1 System setup 18

3.2 Process Simulate study 19

3.3 KUKA.OfficeLite VRC and programming of the robot 22

3.4 Scene setup in NVIDIA Omniverse 25

3.4.1 Scene lighting 27

3.4.2 Object texturing 29

3.5 Virtual camera setup 31

3.5.1 Camera intrinsic parameter calibration 33

3.6 Developing offline dataset generation tools 34

3.7 Interactive application development 38

3.7.1 Modifying the offline generator script 40

3.7.2 Main script overview 41

3.8 Camera extrinsic parameter calibration 42

3.9 Pose estimation 46

3.10 Setting up physical camera 47

3.11 Defining the gripping positions 48

4 Experiments 49

4.1 Evaluating the object detection model 49

4.2 Evaluating methods in simulated environment 50

4.3 Evaluating methods at the physical cell 52

5 Results 53

5.1 Object detection using YOLOv8 53

5.2 Pose estimation using MegaPose in simulated environment 54

5.3 Pose estimation using MegaPose at the physical cell 58

5.4 Transition from Digital Model to physical cell 59

6 Conclusion 61

A Bibliography 63

Figures

2.1 Robot object manipulation scheme 4
2.2 Coarse pose estimator and pose refiner[7] 6
2.3 Difference among the three definitions proposed by [14] 8
2.4 Comparison of the scene with and without photo realistic rendering 10
2.5 Kuka Educate dimensions [3] 13
2.6 Physical robotic cell 14
2.7 Testbed Dataset 15

3.1 Data flow used in this thesis 18
3.2 Workspace graphic containing the workspace envelope and the dimensions of the robot [30] 19
3.3 Robot kinematics setup in Process Simulate 20
3.4 Gripper kinematics setup in Process Simulate 20
3.5 Vacuum end effector setup in Process Simulate 21
3.6 Controller settings window in Process Simulate 22
3.7 Screenshot of the KUKA.OfficeLite VRC 23
3.8 Tecnomatix VRC Server window 23
3.9 Tool and base settings 24
3.10 The default NVIDIA Omniverse scene as exported from Process Simulate 28
3.11 Process of taking 3 of 51 images to perform the stitching process 29
3.12 Preview of materials with given HDRI background map 29
3.13 The created HDRI image as an image 30
3.14 Comparison of stainless steel textures 30
3.15 An object with its reflection in the glass 31
3.16 Visualization of the final version of the NVIDIA Omniverse environment 32
3.17 Two iterations of random pose generation output with one object 38
3.18 Scheme of the proposed star-shaped architecture 39
3.19 Comparison of different mesh approximations for collisions 42

3.20 Two of eighteen calibration poses from the process of extrinsic camera calibration 44

3.21 The relation between the perspective projection in the image plane and the point's location in the world coordinate frame 45

3.22 Eye-to-hand configuration schema 45

3.23 MegaPose estimation render & compare visualization 46

3.24 Difference in the camera coordinate system definition between standards of OpenGL and OpenCV 46

3.25 Absolute and gripping transformation for the object 48

4.1 One of training batches 51

5.1 Confusion matrix of labels and predictions on the virtual environment testing set 55

5.2 Confusion matrix of labels and predictions on the physical cell testing set 56

5.3 Visualization of faulty object estimation appearing correct in the image plane (magenta object is the rendered model in the estimated pose, green is the ground truth pose) . . . 57

Tables

5.1 Image detection metrics on the testing set from virtual environment 54

5.2 Image detection metrics on the testing set from physical cell 54

5.3 Comparison of 7 different methods of camera extrinsics calibration . . . 56

5.4 Object estimation errors using MegaPose 58

5.5 Object estimation errors of the whole pipeline (intrinsic, extrinsic, MegaPose) 59

Chapter 1

Introduction

Robotic manipulation of objects using cameras represents a pivotal task in the automation and robotics industry. This technology involves the use of visual data captured by cameras to guide and control robotic actions such as picking, placing, assembling, welding and other manipulation tasks. The integration of cameras in robotic systems enhances their ability to interact with dynamic and unstructured environments, leading to an increase of versatility and efficiency. In the context of various industries like automotive and aerospace, the application of robotic manipulation using cameras ranges from precise assembling of components to part inspection for the task of quality control, utilizing the ability to perceive the environment through camera lens to significantly enhance the operational capabilities and flexibility.

In the aforementioned industries, deploying new production lines typically involves a long process of designing and testing, which results in long downtimes. Introducing simulation and virtual commissioning into the process of designing such production line can help speeding up the development and reducing the downtimes to a bare minimum, while also providing tools for optimization of the processes before their physical implementation, which saves funds otherwise spent on rebuilding the physical constructions. A digital twin, a virtual replica of the physical system, plays a crucial role in this process by allowing to monitor the efficiency and make radical changes in real-time.

However, a significant limitation has been the inability to accurately simulate robotic manipulation of objects using cameras in simulation software used by industry leaders. Traditional simulation tools struggle to replicate

the complexities of visual perception and the dynamic interactions between robots and their environment. This gap makes it difficult to fully test and improve robotic systems in a virtual environment, causing possible issues when transitioning to physical implementation.

Recent advancements have started to bridge this gap, most notably with the development of NVIDIA Omniverse. Omniverse is a platform designed to allow, beside other things, integration of the ray-tracing technology allowing real-time simulation capable of high-fidelity rendering[1]. Through the concept of Connectors, it natively offers compatibility with existing industrial software used for line design and simulation enabling industrial companies to create more reliable digital twins overcoming previous limitations and accurately reflecting the performance of robotic systems in real-world conditions.

The primary objective of this thesis is to explore and evaluate the capabilities of robotic manipulation of objects using cameras within both virtual and physical environments. This thesis aims to demonstrate the potential of NVIDIA Omniverse in improving the accuracy and reliability of such simulations. The key objectives of this thesis include:

- Identifying and selecting appropriate algorithms for 6D pose estimation using an RGB camera.
- Designing a virtual robotic cell with a camera using the provided simulation software.
- Implementing the selected algorithms and systems in the digital copy of the physical cell. This involves programming a robot, developing a high-level controller, camera calibration and implementation of the pose estimation pipeline itself.
- Deploying the solution onto a physical cell.
- Testing and comparing the performance of the chosen pose estimation algorithm in both environments.

By addressing the limitations of current simulation techniques and leveraging NVIDIA Omniverse, this thesis seeks to enhance the field of virtual simulations, leading to improvement of the design, testing phases of robotic systems and physical deployment, thereby reducing costs and increasing efficiency.

Chapter 2

Background

This chapter summarizes the background information necessary to understand the thesis context around implementation and results. It starts by defining the pick-and-place task in the context of robot object manipulation in section 2.1. An explanation of the 6D poses estimation task, and a description of the MegaPose method used in the thesis follows in section 2.2. The concept and definition of Digital Twin is discussed in section 2.3 (page 8), highlighting its applications and benefits in optimizing efficiency. The next section 2.4 (page 9) describes the simulation software used in detail in the context of industrial automation. After that, the Kuka educational robot cell and the Testbed Dataset are introduced respectively in chapters 2.5 (page 12) and 2.6 (page 13).

2.1 Robot object manipulation

Robot object manipulation has become a core part of the automation industry, which relies heavily on consistency and high precision, leading to higher productivity. As a task, it consists of a robot interacting with its environment and objects within its workspace, such as picking and placing objects, welding, painting, screwing, and others[2].

In this thesis, a pick-and-place task is implemented and utilized to demonstrate and test the functionality and compare the possibilities and constraints in virtual and physical environments. As the name suggests, it consists of

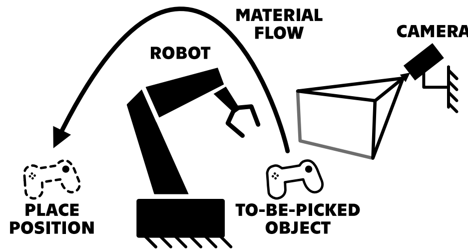


Figure 2.1: Robot object manipulation scheme

two main parts - picking an object and placing it somewhere else. This task was chosen because the manipulated objects generally do not destructively change their state and thus allow for a high number of repetitions, which does not hold for the other tasks mentioned above, such as welding and painting. For further description of the used setup, refer to chapter 2.5.

Various topics related to robotic manipulation are currently being explored, including optimization of movement planning, collaboration of robots with human workers, collaboration of multiple robots, and many more. This thesis focuses on implementing computer vision algorithms for perception into robotics, simulating robotic cells in a virtual environment, and finally testing the solution in real-world conditions.

2.2 6D Pose estimation state of the art

6D (6 Degrees of Freedom) pose of an object in 3D space is a set of values defining the object's position and rotation with respect to a given coordinate system[4]. For this, various types of data are used, such as descriptions of the geometry of detected objects, their color, and textures, which all stem from apriori knowledge of the object. These are then used with data gathered from sensors such as RGB images, RGB-D images with depth information, or point clouds[5]. Research is currently being put into developing and improving methods to make such a process as quick and precise as possible.

Learning-based methods use the given information, such as 3D CAD models of given objects during training[6]. This goes directly against the growing need for flexibility in working with newly introduced objects as training can take up to days to produce sufficient results[7]. Some methods focus on grouping objects into known classes, such as printed circuit boards or cardboard boxes, which partly solves the problem of having to learn weights each time a new object is added to the process. However, all objects must be

subject to the groups defined at the learning stage[8]. Other methods rely on components that are not learned; however, they lack the same level of potential robustness to any changes in the environment, such as lightning conditions, that models with learned weights have[7].

2.2.1 Megapose6D

In this thesis, a method called MegaPose is used for 6D pose estimation. It takes a single RGB (or RGB-D) image, a bounding box of the object's position in the image and the object's CAD model representation. Considering the object's detection in the image is not part of the method itself, the process can be divided into two separate parts, the coarse pose estimation and pose refinement[7].

The coarse pose estimator is used to give an initial pose estimation, which can be refined. It takes an image and the object detection as input. Next, it estimates distance from the camera based on the input bounding box size and creates a given number of randomly sampled poses called hypotheses of the object and renders synthetic views using the object's CAD model. Each hypothesis is scored using the model to decide whether it can be further refined in the next step, which uses the highest scoring sampled pose as its initial pose[7].

For pose refinement, it leverages a method called render & compare. Using the initial pose from the coarse pose estimator and the rendered images together with the original image, it iteratively predicts and updates the pose estimation. After a given number of iterations, it gives the best pose estimation by score[7]. The process overview can be observed in figure 2.2.

To tackle the previously mentioned problems of either needing to know used objects during training or lacking robustness, MegaPose was trained on a large synthetic dataset of images spanning a diverse set of models[7].

2.2.2 FoundationPose

FoundationPose is a state of the art 6D pose estimation method. It uses RGB-D images and, on top of model-based pose estimation, supports model-free estimation, which uses neural implicit representation when no CAD model

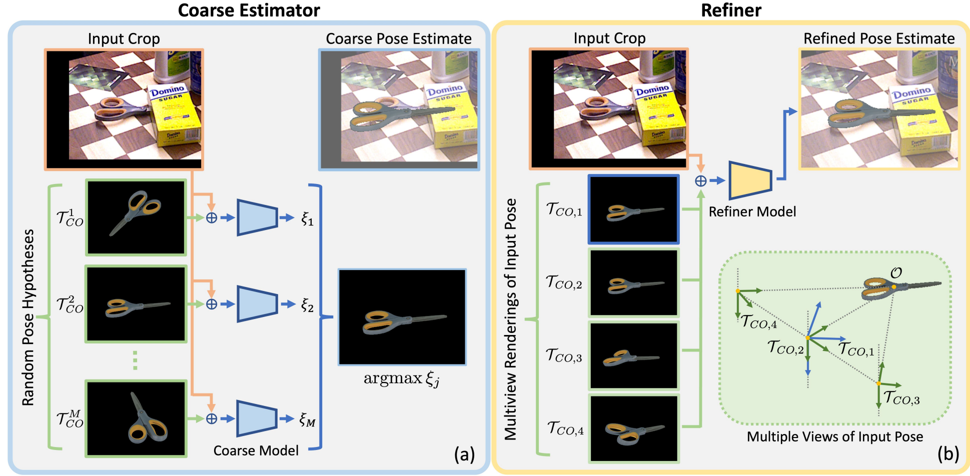


Figure 2.2: Coarse pose estimator and pose refiner[7]

but only a few images of the object are available. FoundationPose training used novel large language models (LLM) technologies and transformer-based architecture to reach better generalization capabilities [9].

After the input of an image with depth and a bounding box is passed to the model, it uses the median of the depth for translation initialization and samples rotations from an icosphere for initial pose hypotheses. Then, the model iteratively improves the pose using a pose refinement network utilizing a single view rendering method, a dynamic cropping strategy and tokenization together with a transformer encoder to predict the translation update. A hierarchical pose ranking network is used to compute scores for each hypothesis, and a pose with the highest score is selected. The ranking is done using a two-level strategy. First, each pose hypothesis' render is compared against the cropped image. Second, the context of all pose hypotheses is observed and evaluated to get more information about the pose to be estimated[9].

2.2.3 CosyPose

CosyPose introduces a robust framework for 6D pose estimation that focuses on multi-view and multi-object scenarios. CosyPose addresses the complexities associated with estimating the 6D poses of multiple objects from multiple views with unknown camera positions[6].

The approach begins with single-view 6D pose estimation using a render-

and-compare method inspired by DeepIM[10]. This method generates initial 6D pose hypotheses for each object in individual images. CosyPose then employs a robust matching procedure to correlate these hypotheses across different views. This is achieved through an object-level matching process using RANSAC[11], which optimizes the overall scene consistency by minimizing reprojection errors.

A key feature of CosyPose is its global scene refinement stage. This stage refines the poses of both objects and cameras by solving an object-level bundle adjustment problem, ensuring a consistent and accurate reconstruction of the entire scene. CosyPose explicitly handles object symmetries, is robust to missing or incorrect object hypotheses, and does not require depth measurements. It has demonstrated significant performance improvements on benchmarks such as YCB-Video and T-LESS, surpassing existing single-view and multi-view pose estimation methods[6].

2.2.4 Comparison and selection of 6D Pose algorithm

FoundationPose and MegaPose are both state-of-the-art methods for 6D object pose estimation. FoundationPose is known for its innovative use of foundational models and transfer learning techniques, which allow it to achieve high performance with relatively less training data[9]. This approach is advantageous in scenarios where data acquisition is challenging or costly. FoundationPose leverages a robust framework that integrates deep learning with traditional pose estimation methods, providing a balanced solution that performs well across a range of conditions.

CosyPose, with its strong performance in multi-view scenarios and robust handling of object symmetries, offers an excellent solution for complex scene reconstruction, which is not the main focus of this thesis. Additionally, it is not applicable in a setup with a static camera, which is the desired configuration of the cell.

While FoundationPose seems to yield better results on 6D pose estimation of unseen objects[12], it actually requires depth for its functionality, making the method not suitable for the used cell design. The only suitable method, which is both allowing for RGB estimation and using a stationary camera is MegaPose, which is actually also very competitive according to [12].

2.3 Digital Twin, Digital Model

A simulated robot cell is often called the Digital Twin. Digital Twin is a technical term with many definitions currently being used. One of the original ones defines a Digital Twin as a set of information fully describing a physical product where any information obtainable by inspecting a physical product should, in an ideal world, be also obtainable from the Digital Twin[13].

However, this proved not to be as unambiguous as many applications and research papers, which use different definitions. [14] proposes to divide the definition into three, the Digital Model, the Digital Shadows, and the Digital Twin (shown in figure 2.3), depending on whether or not changes in physical or digital objects are automatically reflected onto the other. A proposed definition of a Digital Model is a virtual representation that does not use automatic synchronization with the physical product. Any reflecting changes require manual work. This thesis proposes a methodology for developing a Digital Model representation of a physical cell.

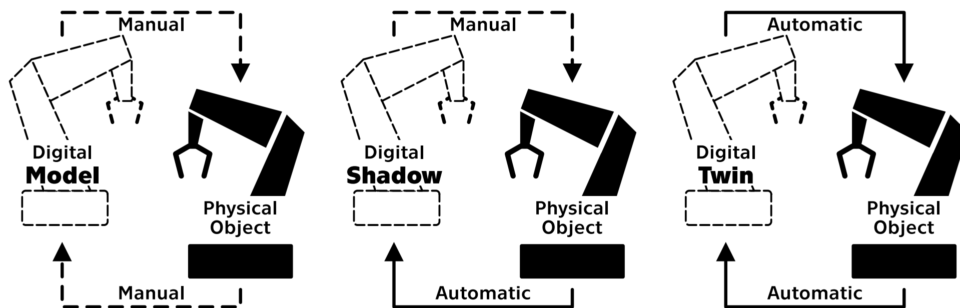


Figure 2.3: Difference among the three definitions proposed by [14]

The development of a Digital Model has many applications and can bring advantages in every step of the Product Lifecycle. For example, the first version of a robotic cell's Digital Model can be used for a case study of configurations using different robot models and tools. The second iteration can help when designing individual parts of the cell. Layout and all geometries can be fine-tuned to achieve the best efficiency and tested with, among other things, collision and reachability checks. Finally, a fully developed Digital Model can be used as a reference when a physical cell is being built and serviced.

Research in this direction is attractive to companies of any scale. For smaller companies, the difference in cost of modeling a production line instead of building it and adjusting construction afterward can be a question of future existence. For large corporate companies, it is more of an opportunity to optimize the cost of developing and implementing needed changes into

already functioning production lines while minimizing production downtimes. Currently, there is no solution to link the industrial simulation platform and the types of robots used by industry together with photorealistic camera simulation, which plays a key role in the process. This limits most companies to completing the robotic workstation with the camera system virtually. This thesis, on the other hand, presents a solution to this problem.

Although rendering software has been used for the creation of synthetic datasets for many years now, currently, one of the main limitations of any virtual representations of industrial applications is the use of rendering software to produce authentic and photo-realistic camera images in real-time. Together with the growth of popularity of using cameras as sensors in the automation industry, this proved to be a strong point of interest for companies, both from the industry of 3D rendering software and the automation industry.

2.4 Simulation software

In the industry of automation, simulation software can serve as a virtual testing ground. It allows companies to create digital representations of production lines, which results in an opportunity to test and rework parts of the system without needing the physical equipment. This process is called virtual commissioning[15]. A vast number of simulation software exists, each of which is specialized at a different set of goals. Most of this software works on the principle of rigid body simulation and allows engineers to create and solve the kinematics and dynamics of a system of bodies.

Some software is focused on emulating individual pieces of hardware, which can then be plugged into larger systems if they are supported, bringing an upside of usually letting the engineers program them using integrated development editors (IDE) used for programming their physical counterparts. This makes it a lot easier for a company to get people from the field to switch to programming digital models. Also, the transition of a program from a simulated environment to a physical device is usually a lot easier as it requires doing very little to zero changes to the code itself[16]. An example of such simulation software would be KUKA.OfficeLite virtual robot controller (VRC) which emulates the physical KUKA robot controller with everything that comes with it, such as TeachPendant, an IO device used for programming KUKA robots, as well as an option to include any add-on package.

Another area of interest is 3D visualization and simulation of the process.

Software of this type usually allows importing robots with their kinematics, writing robot programs with an option of exporting such operations to the physical controllers, checking for collisions, power consumption estimation, and many more. In this subset of simulation software, another distinction can be made to compare different software - the scope the software is made for. While RoboDK or KUKA.Sim are mainly used for virtual commissioning of robot cells usually consisting of up to three robots, Siemens' Process Simulate can be used for modelling of an entire production factory. The latter is described, together with the reasoning behind the choice, in more depth in chapter 2.4.1.

2.4.1 Process Simulate

Process Simulate is a simulation software currently being developed by Siemens. It allows for testing of an entire production process, from robot cells through assembly lines to whole factory processes[17]. In addition to simulation processes already described in the previous section 2.4, Process Simulate offers many other possibilities, such as simulating the autonomous ground vehicles (AGV) resulting in a simulation of the whole product flow or simulating human workers with very detailed information about ergonomics and evaluation of work zones.

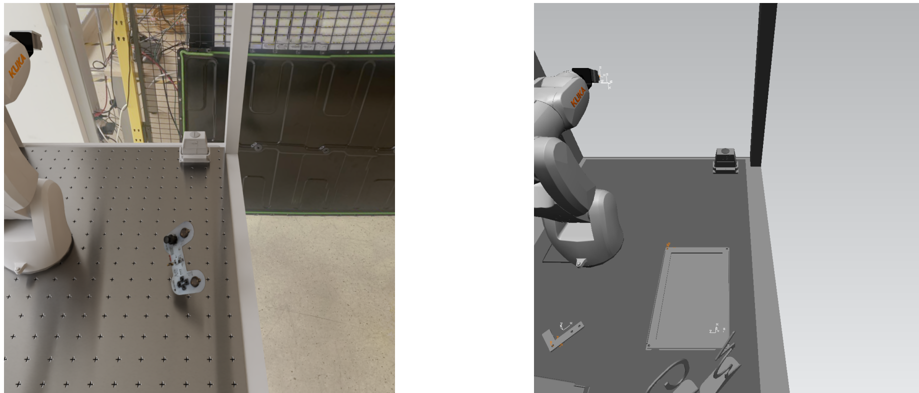


Figure 2.4: Comparison of the scene with and without photo realistic rendering

However, the main reason behind choosing Process Simulate as one of the key parts of this thesis is its native connectivity to NVIDIA Omniverse, a simulation software discussed in the next chapter 2.4.2, and KUKA.OfficeLite VRC. This proved to be a crucial fact as creating connectors between these two given simulation software programs is out of the scope of this thesis and choosing any other combination of simulation software programs would result in the thesis not using the tools used in the automation industry.

Process Simulate offers a variety of options for connecting to other platforms, including both simulation software and physical devices.

Connecting the Kuka VRC to any other simulation software can be done using one of two options mentioned in the products' documentation. One of them is KUKA.VRC Interface, which is used for connecting to other KUKA simulation software, such as aforementioned KUKA.Sim. The other is called Y200Interface, which can be used with other simulation software, such as WinMOD, or, Process Simulate. The interface is used in place of PROFINET, an industrial communication protocol based on the technology of Ethernet usually used to connect devices in the physical automation lines and cells[18]. Y200Interface sends data about the I/O signal values of a robot and also the current position of all the robot's axes in order to synchronize the robot's position.

The Process Simulate to NVIDIA Omniverse connection is natively supported since the 2307 version of Process Simulate[19]. It was introduced as a technology aimed at the Industrial Metaverse and allows transfering projects (studies) from Process Simulate to NVIDIA Omniverse' Nucleus database (see chapter 2.4.2), including geometry and simulations, together with the ability to update between the two mentioned software programs real-time, meaning changes done in Process Simulate are immediately reflected in the NVIDIA Omniverse stage[20].

2.4.2 NVIDIA Omniverse Isaac Sim

In many cases where companies want to simulate a given automation process, before-mentioned simulation (Process Simulate) software fully fits the needs as other sensors are used instead of a camera. However, there is a growing need for the use of cameras in the automation industry and so does the need for generation of authentic images taken from a simulated camera. This is where NVIDIA's Omniverse Isaac Sim (Isaac Sim) comes in. Isaac Sim is a robotics simulation toolkit built on top of the NVIDIA's platform called Omniverse which, beside other things, handles rendering using raytracing[21]. This presents a convenient way of developing Digital Models using camera sensors as Omniverse takes care of everything in terms of producing photo-realistic rendering and an engineer developing such application only has to provide textures and lighting.

NVIDIA Omniverse is a platform consisting of tools for virtual simulation of processes with a support for accurate real-time physics model. Its extensive

documentation creates an environment, in which designers and engineers can create not only large and detailed studies, but also tools and extensions, which further broaden the possibilities and create a whole of an ecosystem. The Omniverse platform is made of five main components[22]:

- **Omniverse Nucleus** serves the purpose of a central database used for collaboration of multiple users. It can be used in a form of a cloud service or installed directly on a workstation for teams of up to two users[23].
- **Omniverse Connect** gives users an opportunity to use third party software as main content creation tools while leveraging the benefits of working with the NVIDIA Omniverse[24].
- **Omniverse Kit** is a toolkit allowing to develop extensions and whole applications using the fully customizable UI engine[25].
- **Omniverse RTX Renderer** is a renderer build on top of the existing NVIDIA RTX ray-tracing technology[22].
- **Omniverse Simulation** is a collection of tools made for simulating a physical world, such as a physical simulator PhysX or robotic simulator toolkit Isaac Sim[26].

Omniverse RTX Renderer is made of two render modes for different uses. The first one is called RTX - Real-Time mode and it is used, as the name suggests, for rendering the scene view in real-time. In general, it produces a higher frame rate, which is why this method is suitable for interactive applications, such as virtual reality (VR) application. It is able to render more details as opposed to traditional rasterization methods, however, isn't as accurate as RTX - Interactive mode, which is the most accurate one. The latter mentioned mode generates a lower amount of frames per second, which suits better for creating pre-rendered scenes. In this thesis, the RTX - Interactive mode is used, as it is not required to have a high framerate, as the camera is stationary and the real camera also needs to wait a given time period for the image to get stale. On the other hand, it is a goal to come as close to the physical world visually, as possible.

2.5 Experimental Workplace Setup

KUKA ready2_educate (KUKA Educate) is a mobile robot cell made to be used as a training tool for people seeing KUKA industrial robot programming

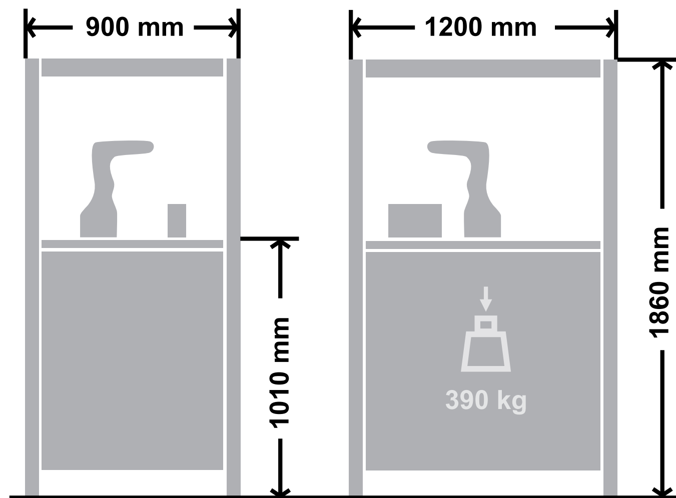


Figure 2.5: Kuka Educate dimensions [3]

for the first time. It comes with a PLC, KR C4 compact robot controller and a KR3 R560 robot preconfigured with a tool and preprogramed for various tasks, such as writing with a marker on a paper, building with plastic cubes that come with the cell, or continuous path motion tasks[3].

There is two main reasons behind the decision to choose KUKA Educate for the physical realization of the simulated environment. The whole setup comes in a chassis with windows surrounding the working space of the robot. During the planning phase of this thesis, it was not obvious, whether or not it will be possible to get the lighting setup in a simulated scene to match the ligthing conditions of a real lab. In such case, it would simply require to install strong light sources into the cell and if not sufficient enough, install tinted windows as well.

Secondly, as KUKA puts it, it can be easily moved using a pallet truck and fit into spaces of smaller size[3], the dimensions can be observed in figure 2.5, together with an image of the physical cell in figure 2.6. Combined with the first advantage, KUKA Educate becomes suitable to be used as a cell presented at various expositions, which is the planned use case of the cell together with its virtual counterpart.

2.6 Testbed Dataset

The selection of an object dataset is an important step in the development and validation of algorithms for 6D pose estimation. This chapter describes

Figure 2.6: Physical robotic cell

the reasonings and process for selecting the Testbed Dataset, modelled and manufactured in Testbed for Industry 4.0 and developed by [27], as the primary dataset for this research.

The Testbed Dataset comprises of an RC car model made from a combination of 3D printed parts and printed circuit boards (PCBs). This dataset comprises high-resolution 3D scans of these parts, which are representative of industrial parts thanks to their complex geometries and textures. The selection of this dataset tests the robustness of these algorithms and also ensures the research is applicable to real-world industrial robotics scenarios.

Additionally, using this dataset keeps continuity of the work and the research done within the Testbed lab, building upon the already established foundations and knowledge base gained during this previous work. This is beneficial because it makes setting up experiments faster and easier since the dataset is ready to use. The objects and their comparison in both the physical and virtual environment can be observed in figure 2.7.

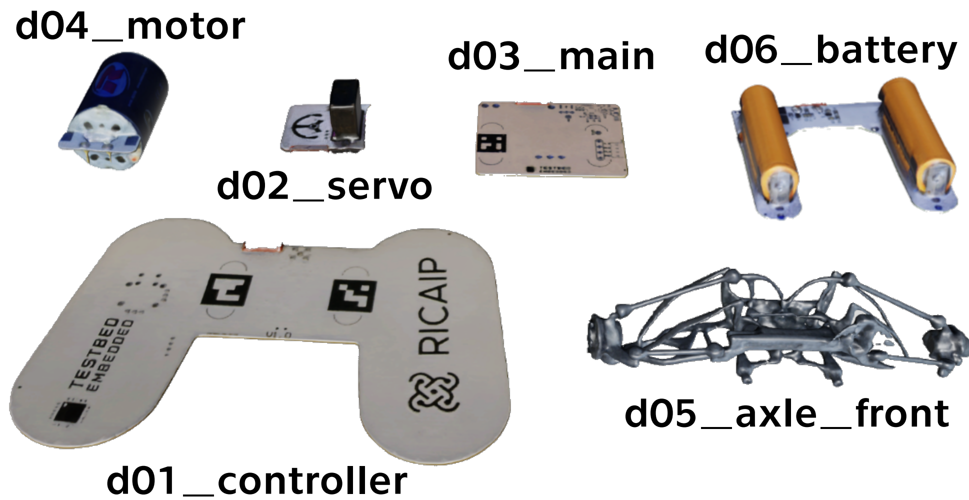


Figure 2.7: Testbed Dataset

2.7 Object detection model

In the development of an automated pick-and-place robot cell, precise and efficient object detection is critical. The methods for estimating the objects' 6D pose generally expect the detection to be done separately, which is why part of this thesis is devoted to learning weights of a detection model using the synthetic dataset.

This thesis implements the YOLOv8 model developed by Ultralytics due to its promising speed and accuracy[28][29]. Also, the model comes with a set of tools for training and evaluating the model without a need for developing such tools from scratch, and pretrained weights for general object detection, which help with reaching high performance with less training done. Using this approach, it gives an opportunity to observe the ability to create a solution with a limited dataset and aligns with the industry standard of developing working solutions as quickly as possible.

Chapter 3

Methodology

This chapter outlines a comprehensive approach to developing and simulating a digital model of a robotic cell using advanced simulation software and tools. The methodology is presented in a step-by-step manner, starting with the initial system setup and progressing through critical stages of simulation and development. The final goal is to deploy the solution onto a physical robotic cell.

1. Design a scene, define the geometries, robot's kinematics using Process Simulate.
2. Connect the simulating software to ensure future functionality.
3. Set-up a visually authentic scene in NVIDIA Omniverse.
4. Create tools for simulating the cell's behaviour consisting of image dataset generator and an interactive simulation application.
5. Develop all program parts.
6. Deploy the solution onto a physical cell.

This workflow leads to a data flow displayed in figure 3.1. The chosen object dataset serves as an input to the dataset generation tool, which is then used to train the object detection model. With MegaPose and object detection model, the application controls the virtual robot and receives its real-time axis' positions.

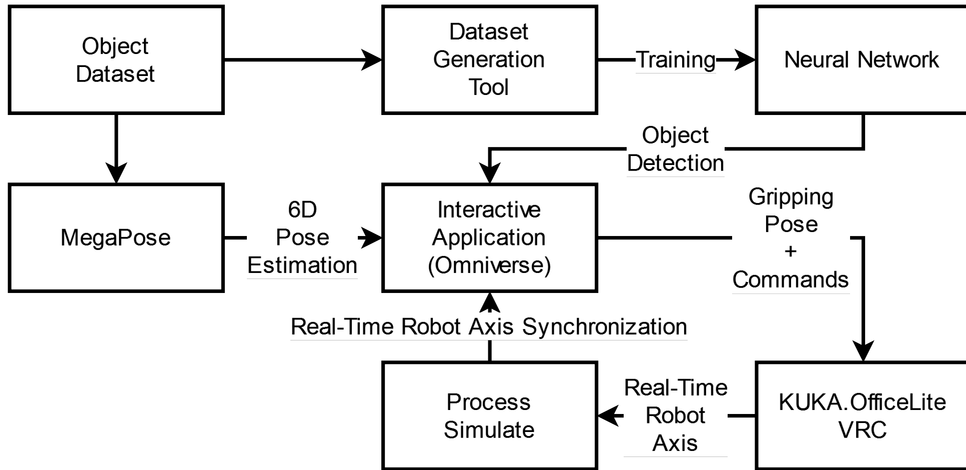


Figure 3.1: Data flow used in this thesis

3.1 System setup

The computer used for development and also running the simulation software is PC Dell Precision 3650 with a cpu 11th Gen Intel(R) Core(TM) i7-11700K @ 3.60GHz and a gpu NVIDIA GeForce RTX 3060 and 32 GB of RAM. The whole setup of using a personal computer to support a robotic cell or a production line in the industrial settings resembles a setup using an industrial computer, which is what has been used for many years. However, it is good to keep in mind that the standards are shifting more towards the cloud computing technologies, hence, the implementation is made in a scalable way. The advantage of running the system setup as an industrial computer is that it shows the flexibility of the implementation even if there is not an opportunity to use cloud computing.

In terms of the simulation software used, the Process Simulate of version 2307 released in the August of 2023 or newer needs to be used, as they introduce the Omniverse Connector, a tool for synchronizing both simulation projects real-time. It is worth noting, that the Omniverse Connector license does not come with the default license of Process Simulate and needs to be bought as an add-on. The documentation of Process Simulate does not directly state what version of NVIDIA Omniverse are supported, however, as it is still a relatively new product undergoing intensive development and functionality of different libraries are changing, should the code be ran as it is, it is recommended to use the same version of IsaacSim, 2023.1.0-hotfix.1, which was used while developing the simulation project of this thesis.

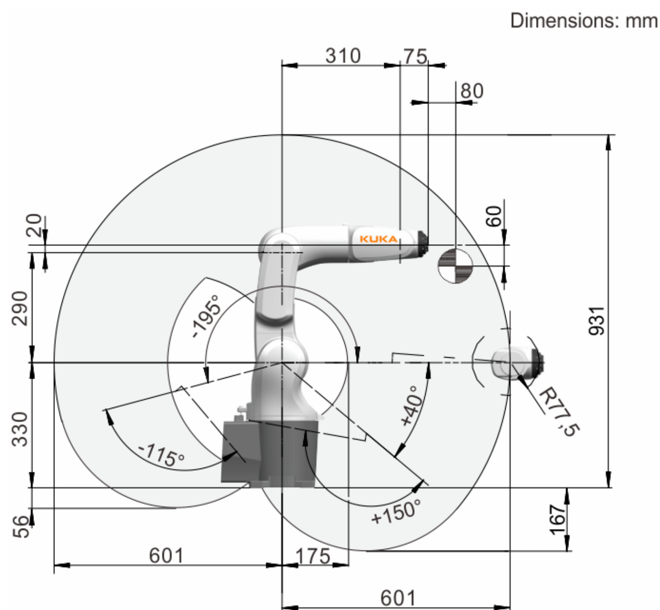


Figure 3.2: Workspace graphic containing the workspace envelope and the dimensions of the robot [30]

3.2 Process Simulate study

The foundation of the Process Simulate Study is made of a STEP file of the robotic cell distributed directly from KUKA. It can be downloaded at a site My.KUKA and contains every part of the geometry of the cell. The only requirement is to define the robot's kinematics, that means defining each individual joint axis of the robot.

The KUKA documentation for the robot contains a workspace graphic (figure 3.2) containing the workspace envelope and all the dimensions that are needed for placing the axis in their respective exact positions. The process of defining the joints' axis in Process Simulate is as follows:

1. Select the robot in the 3D viewer or the Object Tree.
2. Enter the Modeling Scope by clicking the button in the Modeling tab.
3. With robot selected, open the Kinematics Editor window in the Modeling tab.
4. Select Create Link in the Kinematics Editor window, select an individual part of the robot and confirm.

5. When at least two Links are created, they can be connected using a joint by dragging a selected link on top of another.

6. For each joint, select two points defining the axis, the limits and the joint type (in this case all joints are Revolute).

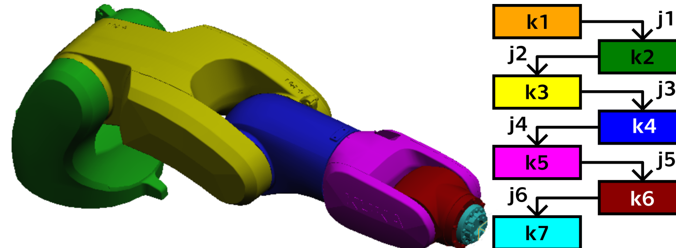


Figure 3.3: Robot kinematics setup in Process Simulate

The result of the process of defining the robot's kinematics can be observed in the figure 3.3. To confirm the right configuration of the joints, especially their orientations, clicking the Joint Jog button in the Kinematics Editor window opens up a new window letting the user to change the current values of all joints.

Same process needs to be applied to the gripper as well. In this case, the presence of the KUKA Educate cell in the lab was used to measure the gripper's open and closed positions. Also, as it can be observed from the figure 3.4, the links are connected using joints in a parallel, which would be expected as both moving parts are moving synchronously in relation to the static part of the gripper.

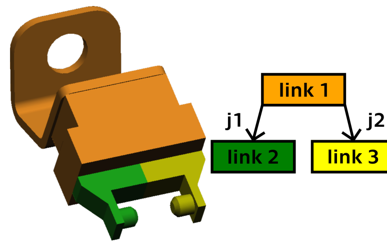


Figure 3.4: Gripper kinematics setup in Process Simulate

In the development of the physical cell, it is important to consider possibilities regarding the gripper type choice and the gripper design. After a thorough decision-making process, it was decided to use a suction gripper over a gripper with parallel fingers. The proposed solution consists of two main parts. The first part is a suction cup ESS-20-EN and its holder ESH-HCL-4-QS made by Festo together with a part designed and 3D printed in the Testbed for Industry 4.0. For generating a vacuum, the solution uses a vacuum pump MINI L14 made by Piab. The solution can be observed in figure 3.5.

The main reason for selecting a suction gripper is the characteristics of the object dataset. Multiple objects are made of printed circuit boards (PCB) which are flat. Suction grippers provide an advantage when such flat objects are lying on a flat surface as they can use bigger area for gripping the objects, hence, requiring less precision to manipulate them.

Furthermore, a question needed to be addressed of whether to develop a new gripper from scratch or to adapt to the cube-based tooling system of the KUKA Educate cell. The proposed solution uses the cube design as it, firstly, prevents from making permanent changes to the already existing setup, and secondly, offers flexibility allowing the cell to use various gripping tool types. By utilizing this design, the system can adapt to different tasks and different object datasets without a need for offline reconfiguration, which is a feature needed for the cell's future use.

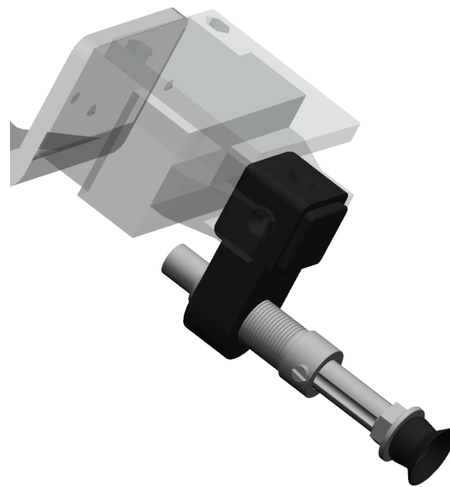


Figure 3.5: Vacuum end effector setup in Process Simulate

In this thesis, both the simulated robot controller program and the physical robot program run in the Automatic (AUT) mode which is designed for robots running without higher-level controllers (PLC)[31]. At the physical cell, the PLC is solely used for management of safety, and in the virtual environment, the safety is not modelled. Hence, defining the robot's signals can be skipped. This makes the next step of connecting the Process Simulate to KUKA.OfficeLite VRC a lot easier.

For the Process Simulate part, it is needed to, first, select the robot in the Object Tree and then go to the Robot ribbon and choose Controller Settings in the Setup part of the ribbon. A window opens up (figure 3.6 for illustration purposes) where the IP address of the VRC virtual machine and port of the VRC Server needs to be filled in, together with Motion Planner set to *VRC* and correctly filled in version of the *Controller*. After all has

Controller Settings

Validate Connection

Robot Name	Controller		Motion Planner	VRC	
	Name	Version		Host	Port
kukakr3r540PS	Kuka-Krc	v8.6.6	VRC	10.10.10.10	8523

Figure 3.6: Controller settings window in Process Simulate

been set, the *Validate connection* button will test the connection. If done correctly, the a window will pop up saying it connected successfully (the steps in the section need to be done after the starting the Tecnomatix VRC Server Kuka Real Time in 3.3). Starting the simulation in the *Line Simulation Mode* (event-based simulation) automatically connects to the VRC and the robot can be jogged to check the correct functionality.

The same way as above in the case of connecting to the VRC, connecting Process Simulate to NVIDIA Omniverse is quite straightforward and most of the steps happen on the side of NVIDIA Omniverse. After setting up an NVIDIA Omniverse' Nucleus database and connecting to it, the two software programs can be connected. In ribbon *View*, go to the section *Omniverse* and click on *Connector for Omniverse*. A window opens up, where there is a directory tree structure of the connected Nucleus servers and two buttons, *Live Connect* and *Export Scene*. *Export Scene* generates a USD file which can be used anytime and anywhere, without the Process Simulate running. In this thesis, exported scene is used for offline generation of synthetic datasets as well as testing setting up a scene in NVIDIA Omniverse. On the other hand, *Live connect* creates a file of extension `.live` and the window for Connector for Omniverse goes grey signaling an ongoing real-time connection into the file. Opening up the *.live* file in the NVIDIA Omniverse any changes done to the scene in Process Simulate are directly visible in Process Simulate.

3.3 KUKA.OfficeLite VRC and programming of the robot

KUKA distributes its VRC in the form of a system image which can be installed using a hypervisor, software used for creating and running virtual machines. In the case of KUKA.OfficeLite, it is directly recommended in the product's documentation to use Microsoft Hyper-V[16]. After performing all the steps required to correctly setup the VRC, such as licensing, the user is met with a user interface of the KUKA's teach pendant called smartPAD as

a window inside of a virtualization of Windows 10.

Figure 3.7: Screenshot of the KUKA.OfficeLite VRC

Firstly, a robot has to be set in the VRC to match the one that is used in the physical KUKA Educate cell. This can be done via KUKA.WorkVisual, an engineering suite offering tools for configuration, programming and diagnosis of robot controllers. Along with replacing the robot, an add-on package of KUKA.OPC UA was installed onto the VRC. With correct dimensions and limits set in the VRC as well as the OPC UA package installed, it is possible to connect the VRC to the Process Simulate study and start with the programming of the robot.

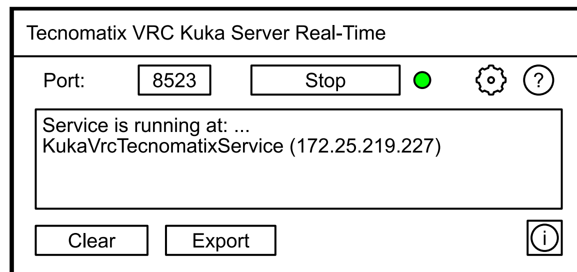


Figure 3.8: Tecnomatix VRC Server window

The KUKA VRC part of connecting to Process Simulate also consists of only a few steps. First, the correct version of the Tecnomatix VRC Server Kuka Real Time needs to be downloaded from the Siemens' download center[1]. Next, it needs to be installed in the virtual machine. As a part of the installation process a shortcut icon is created on the desktop. Starting the application opens a window (figure 3.8) where a port of the connection is set. Clicking on a button *Start* opens up the connection and an IP address is logged into the text field under the button. It is important to match these with the *Controller settings* in previous section 3.2 on page 19. After

[1] https://support.sw.siemens.com/en-US/product/288782031

openning up the connection, it is finally possible to run the Process Simulate study. If connected successfully, there is going to be a circle flashing in green color next to the button *Stop*.

As previously mentioned in section 3.2, the robot itself runs in the AUT mode. This means that the robot is not controlled using a PLC connected by a typical industrial communication bus. Instead, it is controlled using a communication protocol OPC UA which utilizes the aforementioned OPC UA package. It is a cross-platform communication protocol which is a direct response to Industry 4.0's demand for flexibility[32]. It doesn't rely on manufacturers' internal standards of communication and instead allows communicating with devices of different manufacturers using a unified interface. As such, using the OPC UA library in a robot controller does not require any additional work to be done to create and add signals and variables into the communication interface. Instead, declaring a variable in the Kuka Robot Language (KRL) code using the keywords `DECL GLOBAL <type> <varname>` automatically by default results in the variable being writable and readable from the OPC UA interface.

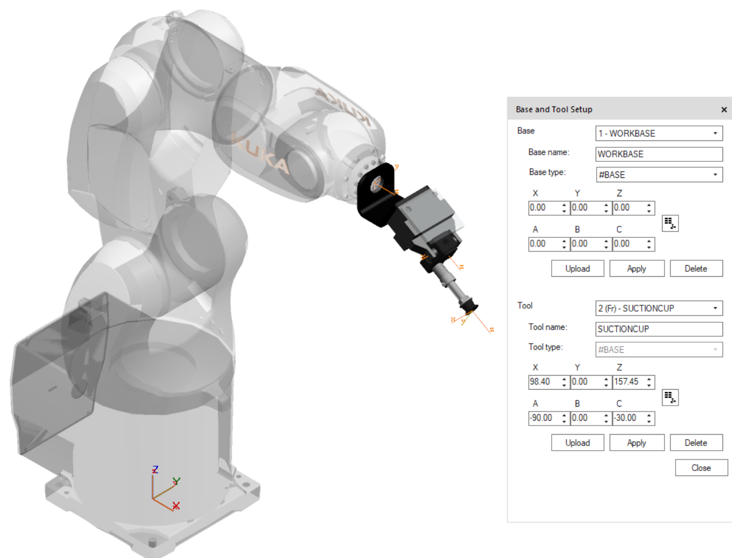


Figure 3.9: Tool and base settings

Before starting to create code for the robot, the base and tools need to be correctly defined. In the intended task, the base frame of the robot is used as the base of the coordinate system. Setting up the robot's tools is a more complex process. It involves transforming three coordinate systems: from the flange robot frame to the gripper frame and to the suction cup frame. This can be seen in 3.9.

The design and creation of the suction end-effector was done in order to be

able to grasp all objects in the dataset, since the original gripper on the robot only allows grasping objects of the cube type with a cylindrical centering hole. The gripper therefore works on the principle of a simple tool changer. The advantage of working with the simulation software is the possibility to obtain all the described transformations and entering them directly into the virtual robot controller. To compensate for the inaccuracies of the real end-effector, it is still possible to perform a four-point calibration method, which consists in moving the robot in different orientations on a fixed point in its workspace.

The robot program's pseudocode can be observed in algorithm 1. The code is quite simple and consists of a loop waiting for a rising edge of `Command_Request` variable and a switch case branching depending on the value of `Command_Number`. In case of value 2, the robot calls a sub-program called `PickOperationRoutine` (algorithm 2). It is expected that in the case of sending the robot to the picking position, the picking position is sent before the `Command_Request`.

The pick routine consists of momentarily setting the base to `PickPosition` frame and performing all operations in such base. This allows the robot to first arrive at a point `PrePickOffset`, which is defined as

$$\texttt{PickPosition} + \{z = -50.0\}\ [\text{mm}], \tag{3.1}$$

without a need to do any frame arithmetic. The robot is then moved to the gripping position using `LIN` operation, and afterwards applying a correction of a millimeter in the z-axis per cycle in a loop of defined number of corrections, until the gripper signalizes a successful gripping. Applying the correction can be done thanks to the design of the suction gripper, which allows a contraction of a total of 20 millimeters giving some margin for error of the pose estimation.

3.4 Scene setup in NVIDIA Omniverse

The setup of the scene in NVIDIA Omniverse starts with one of two options depending on the use case and the method used when exporting the study from Process Simulate. In this section, the Universal Scene Descriptor (USD) file exported from the Process Simulate study is used. For future reference, this is what is referred to when speaking about a default scene setup in this thesis.

As can be seen in figure 3.10, the default scene setup has close to no textures. The only textured parts are single colored parts, such as KUKA

Algorithm 1 A pseudocode algorithm of the robot program

```
1: INIT_MOVES                                   ▷ Base, Tool and other variables are set here
2: PTP HOME
3:
4: while TRUE do
5:     INIT_VARIABLES
6:     WAIT FOR (_/‾ of Command_Request)
7:     Command_Running ← TRUE
8:     Command_Ready ← FALSE
9:     Ack_Result
10:    switch Command_Number do
11:        case 1                                                      ▷ Go Home
12:            PTP HOME
13:            WAIT FOR $ASYNC_STATE = #IDLE
14:        case 2                                              ▷ Go to Pick position
15:            PickOperationRoutine()
16:            WAIT FOR $ASYNC_STATE = #IDLE
17:        case 3                                                ▷ Start calibration
18:            for ITER = 1 to Calibration_Length do
19:                PTP CalibrationPose[ITER]
20:                WAIT FOR $ASYNC_STATE = #IDLE
21:                Calibration_RenderRdy ← TRUE
22:                WAIT FOR (_/‾ of Calibration_RenderDone)
23:            end for
24:        default                                     ▷ Unknown Command Number
25:            Command_Error ←TRUE
26:    end switch
27:    if Command_Error = TRUE then
28:        WAIT FOR (_/‾ of Command_Ack_Result)
29:    end if
30: end while
```

logos or cubes originally used for one of KUKA Educate training scenarios. Fortunately, for the task of this thesis, most of the surfaces' colors are not needed to reach the amount of the authenticity required. Before changing the textures, however, it is reasonable to work on the environment and lighting of the scene, as any reflective materials have potential to work diametrically different in an empty default scene as opposed to a fully developed scene, where the surroundings can be reflected by shiny materials. There are two options to choose from when designing the environment setup.

Algorithm 2 Pick Routine

```
 1: def Pick_Routine()
 2:     DECL INT PreviousBase
 3:     DECL INT ITER
 4:     DECL FRAME Origin
 5:
 6:     PreviousBase ← CURRENT_BASE_IDX
 7:     Origin ← {X 0.0, Y 0.0, Z 0.0, A 0.0, B 0.0, C 0.0}
 8:     BASE_DATA[1] ← PickPosition
 9:     CURRENT_BASE_IDX ← 1
10:     if Is_Reachable(Origin) then
11:         PTP PrePickOffset
12:         WAIT FOR $ASYNC_STATE == #IDLE
13:         Gripper ← CLOSE
14:         LIN Origin
15:         WAIT FOR $ASYNC_STATE == #IDLE
16:         for ITER = 1 to NumberOfCorrections do
17:             LIN_REL Correction
18:             WAIT FOR $ASYNC_STATE == #IDLE
19:             if Gripper_Closed then
20:                 break
21:             end if
22:         end for
23:         LIN PostPickOffset
24:         WAIT FOR $ASYNC_STATE == #IDLE
25:     else
26:         Command_Error ← TRUE
27:     end if
28:     CURRENT_BASE_IDX ← PreviousBase
29: end def
```

3.4.1 Scene lighting

First uses a full-scale 3D model of the lab with all its robot cells and workstations. This approach comes with a disadvantage of requiring significantly more computational power to produce the same amount of realism as the next one due to the presence of vast geometries. Not only does this pose a problem for future potential use of a virtual reality headset or even a computer screen rendering for immersing the user into the simulation as significant drops in frames per second can start to occur making it inconvenient to interact with the simulation. It also prolongs rendering of each individual frame, which would result in great delays when creating synthetic datasets for learning weights of neural networks. The advantage is that, given enough computational power, a single NVIDIA Omniverse scene can be used for simulating

Figure 3.10: The default NVIDIA Omniverse scene as exported from Process Simulate

the whole factory as this is also supported by Process Simulate, enabling to simulate the whole production line together with the so-called material flow between separate cells.

The second method leverages a high dynamic range image map (HDRI) which is a photographic image capturing a view of a location in every direction from a single viewpoint. It is usually taken either as a panorama or a photosphere taken with a 360° camera. A standard bitmap image, sometimes also called low dynamic range image, stores brightness values as an 8-bit value ranging from 0 to 255, whereas the HDRI image will describe brightness values using 32-bit number which better describes the range in which real-word brightness levels appear. This helps capturing a consistent lighting in a scene where one picture is taken in direction of a light source and one the opposite direction. Also, it helps to identify the brightest spots on the photos and making them the light sources used in the scene.

HDRI images are easy and quick to take. Nowadays, applications for mobile phones with cameras exist on both Android and iOS. They are created by tens of pictures in different angle and the applications help with stitching

them together. This thesis follows the approach of taking an HDRI image, the process of taking the pictures and the results can be seen in figures 3.11, 3.12 and 3.13 (page 29 to 30). To import the HDRI map into the scene, a light of type *Dome* is created, which takes *Texture file* as a parameter, as well as *Intensity* and *Exposure*, which are the only two parameters tuned to portray the amount of light generated by the HDRI map.

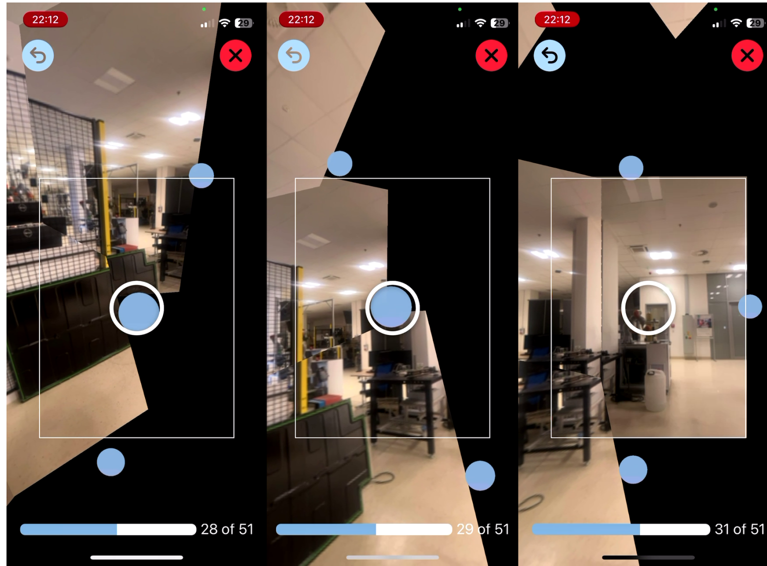


Figure 3.11: Process of taking 3 of 51 images to perform the stitching process

Figure 3.12: Preview of materials with given HDRI background map

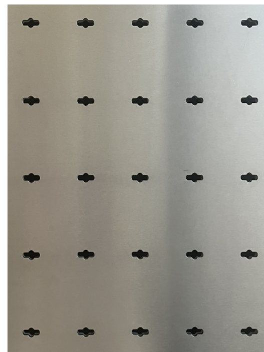
3.4.2 Object texturing

Texturing the scene consists of texturing a ground plane inside of the robot cell and adding glass windows to the sides of the cell.

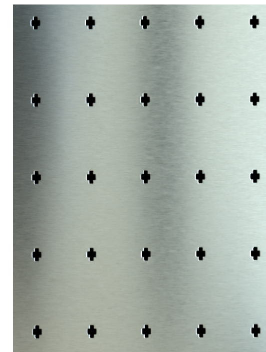
The ground plane of a physical cell has a surface of a brushed stainless steel with cross shaped holes in a grid of 5×5 cm. The almost exact same

Figure 3.13: The created HDRI image as an image

type of material is directly in the Omniverse' material asset library. The only difference being the shape of the holes which varies slightly and can be tuned to match the dimensions using following parameters: *punching grid size*, *cutout size*, *cutout roundness*, *cutout bevel width* and *cutout bevel strength*. When it comes to the texture itself, a number of parameters can be tuned for best resemblance of the physical material, such as *reflectivity*, *emissivity* or *roughness*. The last parameter is the nly parameter changed, as the *roughness* had to be raised to about double of the default value, resulting in softening the reflections. In other words, items farther away from the plane appear with softer outlines. The material texture to its physical counterpart can be observed in figure 3.14.



(a) : Photo of stainless steel plane



(b) : Stainless steel texture render

Figure 3.14: Comparison of stainless steel textures

For the glass windows, a clear glass asset is used. The only change applied to the asset is setting the *Thin Walled* attribute to `True`. This results in the incoming light not being refracted, which is not a trait needed to be simulated and results in lower computational power needed. The main reason for putting the glass windows into the scene is to simulate whether or not

are reflections of the objects going to be wrongly detected as extra objects in the scene (figure 3.15). This could potentially lead to robot crashing into the window and breaking either the gripper or the window itself. Should there be a problem with the reflections being detected as objects themselves, it will be needed to limit the area of interest (AOI) of the image of the camera so that the outside of the cell is not seen in the detection part of the pose estimation pipeline. However, it should not be cropped out right away as the camera itself is not installed directly next to the window, hence, cropping the image would result in limiting the workspace even further than it already is. The complete and finalized scene is shown in figure 3.16 on page 32.

Figure 3.15: An object with its reflection in the glass

3.5 Virtual camera setup

While the rest of the robot cell Digital Model could rely on its physical counterpart being present throughout its development, the camera parameters were not known until the moment of transition to the physical cell. That means the default camera setup was used for the rest of the chapter. This section talks about setting up the camera image rendering to mimic a real camera image with the subsection 3.5.1 introducing the process of measuring the intrinsic parameters of a camera and comparing them to calculated intrinsic parameters from the set of parameters of the simulated camera used in the Omniverse environment.

The Render Settings panel contains a number of different parameters to

Figure 3.16: Visualization of the final version of the NVIDIA Omniverse environment

set which are divided into three categories - *Common*, *Path Tracing* and *Post Processing*. This thesis changes only the settings of the post-processing section.

Firstly, an *Auto Exposure* functionality is turned on. *Auto Exposure* dynamically balances camera settings to produce a well-exposed image. This is convenient because changing the lighting conditions of the scene does not change the rendered image brightness as much as it would have without this functionality, which is also a standard functionality of industrial cameras. With the plans of using the robot cell in various lighting conditions, auto-exposure is a must for the correct functioning of the camera. Several attributes of *Auto Exposure* can be changed in the NVIDIA Omniverse.

Namely, *Adaptation speed* is set to the lowest possible value, which does not change much as even with the lowest value, the adaptation process does not take more than a second; it may, however, result in a need for waiting for a number of rendered images until the correct image is rendered. This means any scripts working with the renderer will have to implement wait periods, which are expected to be needed for working with the physical camera, as well.

The second attribute worth the attention is *Exposure Clamping* with its sub-attributes *Min* & *Max Exposure Value*. Without setting limiting values to these attributes, a scene with any lighting conditions is going to appear in

the camera view without a change, which is not how a physical camera works due to its limits of exposure times.

White Point Scale is the last tuned attribute, setting the desired brightness of the resulting image after the auto-exposure.

For the same reason as in the case of *Auto Exposure*, the *Motion Blur* effect is used with the highest value of number of samples. It might not seem to be too important to have the *Motion Blur* effect enabled, as the pictures are meant to be taken when the scene appears to be static, however, sometimes, especially when the framerate of a camera is low due to the exposure times, it is better to keep in mind that if the waits are not implemented, a camera shot might produce more of a smudge than a correct static picture.

The last effect added in the renderer's settings is *Film Grain*. *Film Grain* applies a layer of noise onto the image. This mimics the effect of gain which digitally amplifies the physical camera's sensor's signal to brighten images in low light, however, also introduces unwanted noise.

3.5.1 Camera intrinsic parameter calibration

Camera calibration is an essential part of all computer vision applications and techniques. It uses a detection of predefined patterns in images taken by a camera with some given setup. This thesis uses a chessboard pattern of predefined sizes, a commonly used pattern due to its high-contrast characteristics.

A methodology using the Digital Model is proposed. The first part of the process consists of calculating the parameters usually obtained by tools like OpenCV or Matlab consisting of calibration parameters in a from an intrinsic matrix and distortion parameters from the parameters that are set in the Omniverse in a form of an f-theta model physical units for the camera's sensor_size and focal_length, see listing 3.1.

With the camera_matrix' calculated ground-truth values, the next step is to estimate the camera intrinsic values. The process starts with taking images of a chessboard placed in the camera's view in as many configurations as possible, with the position of the chessboard changing in both translation and rotation changing with respect to the image plane. It is generally a good

practice to place the chessboard so that it occupies each of the four corners of the camera image, where the strongest effects of distortion can be observed.

```
# Convert from cm to mm
focal_length_isaac = camera.get_focal_length() * 10.0
hor_aperture_isaac = camera.get_horizontal_aperture() * 10.0
vert_aperture_isaac = camera.get_vertical_aperture() * 10.0
focus_distance_isaac = camera.get_focus_distance() # meters
lens_aperture_isaac = camera.get_lens_aperture() / 100.0

# Calculate pixel size, f-stop, and camera matrix
# assuming the pixel area is a square
pixel_size = horizontal_aperture_isaac / width
f_stop = lens_aperture_isaac
focal_length_x = focal_length_isaac
focal_length_y = focal_length_isaac

# Calculate camera matrix
fx = focal_length_x / pixel_size
fy = focal_length_y / pixel_size
cx = width / 2.0
cy = height / 2.0

# Create the camera matrix
camera_matrix_opposite = [[fx, 0.0, cx], [0.0, fy, cy], [0.0,
    0.0, 1.0]]
```

Listing 3.1: Code for retrieving OpenCV-like calibration parameters from the f-theta model

Next, the chessboard corners are detected in each image, allowing for 6D pose estimation of the chessboard target with respect to the coordinate frame of the camera. Reprojecting the chessboard corners' 3D positions back onto the image plane gives the reprojection error in a form of an Euclidean distance to the corners' initial position in the picture.

This leads to an optimization problem of fitting parameters of a chosen camera model so that the reprojection error across all the calibration images is the lowest possible. Defining and solving the problem formally is outside of the scope of this thesis and can be found together with the script for measuring the intrinsic values using the OpenCV library at [33].

3.6 Developing offline dataset generation tools

In this part of the thesis, an emphasis is put on developing a tool, which will be able to generate an image dataset of all the objects of any given

object dataset. A script called `offline_generation.py` was created, which follows the workflow of a template script proposed by [34]. With NVIDIA Omniverse and all its subsidiaries being still under heavy development, a lot of the code from the template script simply does not work in the same way it was proposed. This, together with a need for adjustments in the process of itself, is the reasoning behind going into an in-depth description of developing a new script.

The original script in the tutorial opens up an already prepared scene, which could be done by opening up the exported `.usd` scene exported from the Process Simulate. However, due to internal settings of the scene in the exported file itself, some Omniverse functionality and modules tend to work differently or not at all. Hence, the script starts with initializing the scene by, first, creating a new empty scene, and then adding the exported file as a prim object reference (listing 3.2).

References are mainly used for reusing assets from some shared database of objects so that changing the object in one place ends up updating the objects in all the projects they are used in. Leveraging this functionality allows using the reference to a `.live` object file together with its real-time updates and additionally keeping the scene set-up in the same way any other scene made directly in Omniverse is.

```
1  from omni.isaac.kit import SimulationApp
2  from omni.isaac.core.utils.stage import create_new_stage
3  from omni.isaac.core.utils import prims
4  # starts up the Isaac Sim environment
5  simulation_app = SimulationApp(
6      launch_config=config["launch_config"])
7  # create a new empty scene
8  create_new_stage()
9  # load the Kuka Educate robot cell model
10 prims.create_prim(
11     prim_path="/World/r2e",
12     usd_path=config["server_url"] + config["env_url"])
```

Listing 3.2: Creating a scene using the Omniverse Python API

The next part of the script leverages Isaac Sim Replicator, a collection of tools made specifically for synthetic data generation. To prevent the randomization process from changing the scene it is applied on, the usual workflow starts with creating a new Replicator layer and using its context (keyword `with`) for the rest of definitions using the Replicator functionality. listing 3.3 shows creating a new Replicator layer together with creating object of a camera which is positioned in a uniformly distributed volume and aimed at a coordinate mean of randomly positioned objects from the dataset. The listing also creates a lighting dome which changes its randomly chosen HDRI map texture during the creation of the dataset, and defines a set of materials

used for texturing the plane on which the objects are placed on. When creating a synthetic dataset, it is needed to change the scene's appearance in the background of the objects, so that the models used for object detection do not over-fit to the background.

Creating the first two mentioned objects is straightforward. With the limitations of the version of Isaac Sim used, referencing a material, however, comes in two steps. First, it is needed to create an empty scene with a material in question, a step where attributes of the material can be set beforehand, and exporting it as a `.usd` file. Only then it is possible to reference the file. In this instance, the USD file contains multiple of materials that will be randomly chosen with a uniform distribution.

```
import omni.replicator.core as rep
with rep.new_layer():
    cam = rep.create.camera(
        focal_length=[...], clipping_range=[...])
    render_product = rep.create.render_product(
        cam, config["resolution"])
    # [...]
    light_prim = rep.create.light(
        light_type="dome",
        texture=[...],
        rotation=(.0, .0, 190.),
        intensity=5000.0)
    plane_steel_materials = rep.create.from_usd([...])\
        .get_output_prims()['prims'][0]\
        .GetChildren()[0].GetChildren()
    with rep.trigger.on_frame(interval=1):
        with cam:
            rep.modify.pose(position=[...])
            rep.randomizer.rotation()
        rep.modify.material(
            plane_steel_materials,
            input_prims=steel_plane)
        with light_prim:
            rep.randomizer.texture([texture_url_list])
            rep.modify.pose(rotation_z=[...])
            rep.modify.attribute("intensity", [...])
```

Listing 3.3: Initializing the scene inside of the Replicator layer

Prims of objects meant to be randomized and detected are created in a similar fashion with only a few differences. When creating the prim of the detected objects, a `semantics` parameter is passed to the creator function `rep.create.from_usd`. This is later used when data needs to be annotated. Each prim has a class named the same way its CAD model is. This aligns with what the MegaPose expects as it uses mesh called the same way the class of the object is, to render hypotheses.

When it comes to the positioning, it is called in the context of `rep.trigger.on_frame`, which results in the operation of setting a pose and rotation in every frame. Furthermore, to place the object randomly in the given subspace, `rep.modify.pose` is given an argument of `rep.distribution.uniform`. This can be seen in the first part of the listing 3.4, while the second part shows how rotating between the selected objects is done.

```
with rep.new_layer():
    # [...]
    with rep.trigger.on_frame():
        with new_prim:
            rep.modify.pose(position=rep.distribution.uniform
    ([...]))
            rep.randomizer.rotation()
    with rep.trigger.on_frame(interval=[...]):
        for i, rep_item in enumerate(input_prims_rep):
            sequence = [False] * len(input_prims_rep)
            sequence[i-num_obj_in_frame:i] = [True, True, True]
            with rep_item:
                rep.modify.visibility(
                    rep.distribution.sequence(sequence))
```

Listing 3.4: Creating object prims and rotating between them

Each object is assigned an array of Boolean values of the same length as is number of objects. Then N elements of an array are set to `True`, while others are set to `False`. Every time the number of frames triggered reaches a given number, a pointer to this array is incremented and based on the Boolean value in every object's array sets the visibility of the object resulting in N objects being in the frame.

```
from MegaPoseWriter import MegaPoseWriter
rep.WriterRegistry.register(MegaPoseWriter)
with rep.new_layer():
    # [...]
    writer = rep.WriterRegistry.get(config["writer"])
    writer.initialize(
        **config["writer_config"],
        camera_prim=cam.get_output_prims()['prims'][0].
    GetAllChildren()[0],
        prims=input_prims
    )
    writer.attach([render_product])
```

Listing 3.5: Initializing writer using the Replicator layer

Finally, a writer is assigned to the replicator layer context. Writers are used for processing the sensors' values and data from annotators into a format suitable for training of neural networks and other machine learning models. It is initialized with a list of annotators to be used and an output path, which can be a local directory or a cloud storage in form of a Nucleus database. Annotators are used for generating labeled annotations of the sensor inputs,

namely camera images. The writer used in this thesis uses the `rgb` annotator to produce RGB images taken from the camera placed in the scene and the `bounding_box_2d_tight` which gives bounding box coordinates of objects' positions in the image.

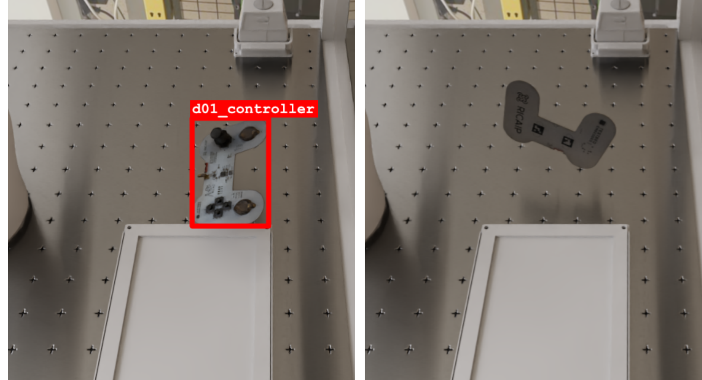


Figure 3.17: Two iterations of random pose generation output with one object

3.7 Interactive application development

The offline generator script is suitable for generating a large amount of images in short time and storing them specifically for later use in form of training machine learning models. However, running the whole simulation exclusively by using Replicator's functions is not suitable for creating a simulation which needs to mimic the real-time behaviour of the robot cell. Such simulation needs to process some form of a user input and also provide some form of visualization, both of which require asynchronous handling. On top of that, the simulation has to also show the real-time geometry of the robot which, of course, does not move in discrete steps from one position to another, since the whole simulation pipeline uses a VRC.

A decision was made early-on to make this interactive simulation in a form of a control script which can be later used to control the physical cell, as well. This ensures that the simulated cell and the physical cell can be developed in a single application and any changes made in the simulation can be immediately observed in the physical cell. However, this introduces some additional challenges, such as writing an API wrapper for the Omniverse script so that its I/O workflow works the same way as working with a physical industrial camera.

The proposed architecture can be seen in figure 3.18. It is a star-shaped architecture, where the main script starts up and communicates with every

other script, each running in a different thread. Each thread child of the main thread running a main script uses a different type of communication due to each type's advantages and limitations. The benefit of an option to automate tests of each separate part's functionality stems also stems from the modular approach. This allows multiple of people to work on the project in the future, ensuring scalability and longevity, since each part of the application only needs to be given expected input and output format and the inner logic can stay separated.

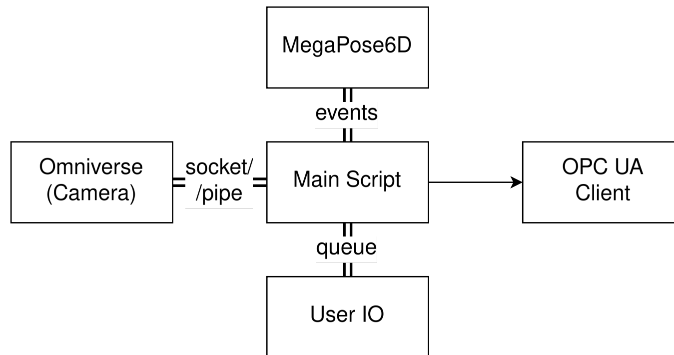


Figure 3.18: Scheme of the proposed star-shaped architecture

The Omniverse/camera script synchronizes with the main script using python's `multiprocessing.connection` socket wrapper. It provides high level messaging API which allows for bidirectional communication between two running scripts. The main advantage offered to this application, and, the main reason it is used, is the connection using an IP address, port and authentication key. Starting up a stand-alone Omniverse application requires to run a bash script which sets and modifies the environment variables, meaning, it can't be simply ran by calling the script directly in Python. Instead, it is started using the `subprocess.run` with parameter `shell` to `True`. Calling this function opens up a new shell and executes a given command. Using this setup results in the inability to pass any objects such as `Queue` to the script by memory address, hence, sockets are used.

The MegaPose script is executed in a never-ending for loop inside of its own thread. The execution is synchronized using `threading.Semaphore` and `threading.Event` objects. In early development stage of the application, semaphores were used exclusively. However, semaphores, as they are implemented in the `threading` module, come with a disadvantage. They do not provide an option to check if they are acquire-able without actually trying to acquire the semaphore. A possible solution would be to use semaphore implementation from module `asyncio`, however, this thesis proposes to use the `Event` class for its potential to create a better and more readable code.

The part of the application taking care of reading the user inputs communicates with the Main Script using the `Queue` object. Processing the inputs in the order they were inserted, or, possibly with a given priority, can be crucial for the correct functionality. Furthermore, the main script might sometimes decide to ignore the first-in-queue command and in such case, thanks to the functionality of the `queue` object, it can be put back into the queue and processed later.

The OPC UA client is imported directly as a class from a module previously described in section 3.3. Thread is created to monitor the connection and reconnect to the robot as soon as possible after a connection dropout.

3.7.1 Modifying the offline generator script

With the generator script already prepared, the Omniverse part of the interactive application is its direct iteration. Some changes need to be worked into the script to fulfill the additional requirements. First and foremost, the stand-alone Omniverse application needs to run asynchronously to allow the user to be able to control the graphical interface (GUI) while it is running. The second requirement is to simulate physics. Lastly, it is needed to setup the communication with the main script.

To achieve asynchronicity, after preparing the replicator layer, which is used in this part for it's writer and annotator support, a `World` object needs to be referenced from the running application. After that, it is needed to periodically call the `my_world.step` function with the parameter `render` set to `True`. Each time this line executes, the GUI of the Omniverse application renders a frame. It is important to ensure this line is called with high enough frequency, otherwise the user experience inside of the GUI will appear unresponsive.

```
from omni.isaac.core import World, PhysicsContext
my_world = World(stage_units_in_meters=1.0)
physics_context = PhysicsContext(set_defaults=False)
physics_context.set_gravity(-9.81e2)

while simulation_app.is_running():
    my_world.step(render=True)
```

Listing 3.6: Defining scene's physics properties

Referencing the `PhysicsContext` as well as setting the gravitational constant can be observed in the listing 3.6. It is needed to define the gravitational

constant in order to match the scene units. It is worth noting that the scene in this thesis uses units of millimetres (mm) which is the same unit the robot controller uses for its poses, hence, the gravitational constant needs to be adjusted accordingly. To apply the physics rules to the object instances created with the `replicator` module, the physics needs to be applied in the `replicator`'s layer context using the `replicator.physics` submodule (listing 3.7. First step is to define some sort of a surface with the collider that the objects can fall onto. After that, each object is given the `rigid_body` and `collider` properties. Rigid bodies allow the objects to accelerate under gravity and other applied forces. Colliders are used for calculating the objects' collisions using simplified models.

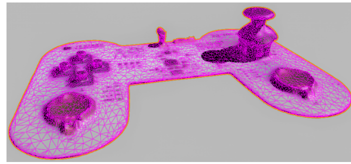
```
1 with rep.new_layer():
2     with rep.create.plane(position=(0,0,config["ground_z"]),
     scale=123, visible=False):
3         rep.physics.collider()
4
5     with new_prim:
6         rep.physics.rigid_body()
7         rep.physics.collider("convexDecomposition")
```

Listing 3.7: Defining the collider approximation and rigid body of an object using the Replicator module

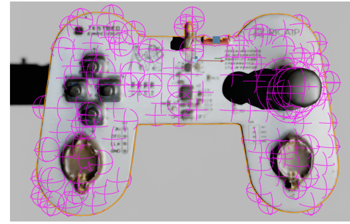
Choosing the right method for mesh approximation depends on the use case. In most cases, it is a good idea to try to lower the computational requirements as much as possible. However, some use cases require a close geometric representation for fulfilling a task, such as assembling multiple parts together, which is the case of this thesis. figure 3.19 shows the comparison of different methods and their representations of the gamepad's model from the Testbed dataset. This thesis uses a convex decomposition with variable number of convex hulls used for different objects in the Dataset.

3.7.2 Main script overview

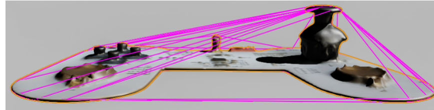
The main script works as a communication bridge between all the other parts of the application and also as a control system giving out requests and expecting responses. It starts with starting all of the threads and also takes care of connecting by sockets to the other scripts and OPC UA to the robot controller. After that, two process state machines are initiated for a process called Generation Process and a process called Calibration Process. The Generation Process switches among states `Not Running`, `Running`, `Waiting for a camera`, `Waiting for MegaPose` and `Done`. The process describes which step of generating an image is currently active. The same goes for



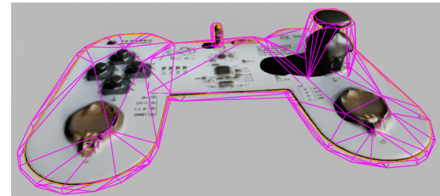
(a) : Triangle mesh approximation



(b) : Sphere approximation



(c) : Convex hull approximation



(d) : Convex decomposition approximation

Figure 3.19: Comparison of different mesh approximations for collisions

the Calibration Process, however, without the waiting states. The simplified process is described in a form of a pseudo-code in the algorithm 3.

3.8 Camera extrinsic parameter calibration

The completion of work on the interactive simulation application finally allows doing tasks that require moving the robot and processing the input from a camera. In this case, it is important to do the camera's extrinsic parameter calibration which starts with moving the robot's tool center point (TCP) in a number of locations with a pattern attached to its tool and taking pictures with the pattern in recognizable positions.

For this purpose, a robot's program was developed so that the whole process of calibration can be automated. It consists of manually created sequence of 18 poses (figure 3.20) which are rotated around all three axes and in different distances to the camera to capture the space as evenly as possible. After arriving at a pose, the robot signalizes that it is ready for an image to be taken by raising the variable called `Calibration_RenderRdy` to `TRUE`, which is acknoledged by bringing a rising edge to the variable named `Calibration_RenderDone`. In the interactive application, acknowledging is done by the main script communicating with the robot. Before the acknowledgment, it first requests an image from the camera (or Omniverse) script and, also, saves the robot's pose into a JSON-formatted file.

Algorithm 3 Main script rundown using pseudocode

```
1:  Establish connection through socket wrapper
2:  Start all threads
3:  CPS ← Not Running                              ▷ Calibration Process State
4:  GPS ← Not Running                              ▷ Generation Process State
5:  while True do
6:      Command ← Check socket message buffer
7:      if Command = Empty then
8:          Command ← Pop I/O Queue
9:      end if
10:     if Command = Generate Data and GPS = Not Running then
11:         Send Socket ← Request Image and Annotations)
12:         GPS ← Running
13:     else if Command = Generation Done then
14:         if CPS = Running then
15:             Send robot to next calibration pose
16:         else
17:             Start MegaPose script on generated data
18:             GPS ← Waiting for MegaPose
19:         end if
20:     else if Command = Start Calibration and CPS = Not Running then
21:         Start Robot Calibration Program
22:         Send Socket ← Calibration Mode
23:     else if MegaPose Event Done then
24:         Send Picking operation with new pose to the robot
25:     end if
26: end while
```

The initial step of the extrinsic calibration is to detect the pattern's features in the image and then estimate the objects pose using the knowledge about patterns dimensions.

The relation between the perspective projection in the image plane $\overline{\mathbf{x}} = (u, v, 1)$ and the point's location in the world coordinate frame $\mathcal{W}$ (figure 3.21), ${}^{\mathcal{W}}\mathbf{X} = ({}^{\mathcal{W}}X, {}^{\mathcal{W}}Y, {}^{\mathcal{W}}Z)$ is

$$\overline{\mathbf{x}} = \mathbf{K} \cdot \mathbf{\Pi} \cdot {}^{\mathcal{C}}\mathbf{T}_{\mathcal{W}} \cdot {}^{\mathcal{W}}\mathbf{X}, \text{ where} \tag{3.2}$$

$\mathbf{K}$ is the camera intrinsic parameters matrix of shape 3×3, $\mathbf{\Pi}$ is the projection matrix of shape 3×4 and ${}^{\mathcal{C}}\mathbf{T}_{\mathcal{W}}$ is the transformation matrix from the world coordinate system $\mathcal{W}$ to the camera coordinate system $\mathcal{C}$:

Figure 3.20: Two of eighteen calibration poses from the process of extrinsic camera calibration

$$
{}^{\mathcal{C}}\mathbf{T}_{\mathcal{W}} = \begin{pmatrix} {}^{\mathcal{C}}\mathbf{R}_{\mathcal{W}} & {}^{\mathcal{C}}\mathbf{t}_{\mathcal{W}} \\ \mathbf{0}_{3\times 1} & 1 \end{pmatrix}.
$$

Assuming the intrinsic calibration matrix is already known, the equation 3.2 can be simplified putting $\mathbf{x} = \mathbf{K}^{-1}\overline{\mathbf{x}}$:

$$
\mathbf{x}_i = \mathbf{\Pi}\, {}^{\mathcal{C}}\mathbf{T}_{\mathcal{W}}\, {}^{\mathcal{W}}\mathbf{X}_i. \tag{3.3}
$$

Knowing the geometry of the object, this leads to a system of equations made of 3.3. Solving this system of equations gives us ${}^{\mathcal{C}}\mathbf{X}$, meaning the objects position in camera coordinate system.

Next, the transformation matrix from the camera coordinate system to the robot base can be obtained. This can be done using OpenCV's function `calibrateHandEye`. Its use is straight-forward for the case of using a hand-eye calibration. If the assumptions given in the documentation of the library are true, the function needs to be given a transformation from the gripper coordinate system to the robot base coordinate system ${}^{\mathcal{B}}\mathbf{T}_{\mathcal{G}}$ and a transformation matrix from the target coordinate system to the camera coordinate system ${}^{\mathcal{C}}\mathbf{T}_{\mathcal{T}}$ and it returns the transformation matrix from the camera coordinate system to the gripper coordinate system ${}^{\mathcal{G}}\mathbf{T}_{\mathcal{C}}$.

However, in this case, it is needed to transform the task so that it is applicable to the eye-to-hand configuration (figure 3.22). Instead of ${}^{\mathcal{B}}\mathbf{T}_{\mathcal{G}}$ we pass to the first parameter a matrix representation of transformation from the robot base coordinate system to the gripper coordinate system ${}^{\mathcal{G}}\mathbf{T}_{\mathcal{B}}$ while the second parameter is passed correctly the matrix of ${}^{\mathcal{C}}\mathbf{T}_{\mathcal{T}}$. This leads to a system of equations

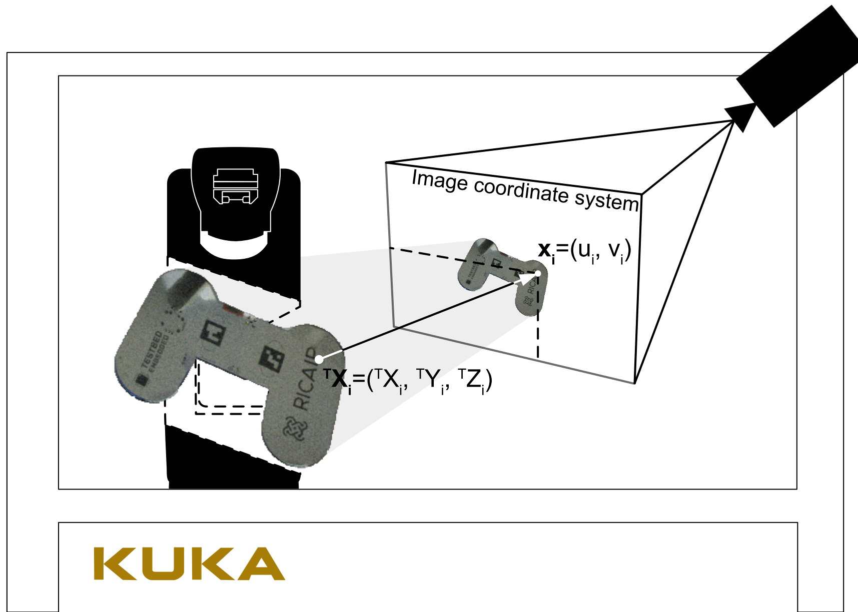


Figure 3.21: The relation between the perspective projection in the image plane and the point's location in the world coordinate frame

$$
\begin{gathered}
{}^{\mathcal{G}}\mathbf{T}_{\mathcal{B}}{}^{(1)}\,{}^{\mathcal{B}}\mathbf{T}_{\mathcal{C}}\,{}^{\mathcal{C}}\mathbf{T}_{\mathcal{T}}{}^{(1)} = {}^{\mathcal{G}}\mathbf{T}_{\mathcal{B}}{}^{(2)}\,{}^{\mathcal{B}}\mathbf{T}_{\mathcal{C}}\,{}^{\mathcal{C}}\mathbf{T}_{\mathcal{T}}{}^{(2)}, \\
({}^{\mathcal{G}}\mathbf{T}_{\mathcal{B}}{}^{(2)})^{-1}\,{}^{\mathcal{G}}\mathbf{T}_{\mathcal{B}}{}^{(1)}\,{}^{\mathcal{B}}\mathbf{T}_{\mathcal{C}} = {}^{\mathcal{B}}\mathbf{T}_{\mathcal{C}}\,{}^{\mathcal{C}}\mathbf{T}_{\mathcal{T}}{}^{(2)}({}^{\mathcal{C}}\mathbf{T}_{\mathcal{T}}{}^{(1)})^{-1}, \\
A_i X = X B_i.
\end{gathered}
$$

The output of the method, and the solution X of the system, is the desired matrix of ${}^{\mathcal{B}}\mathbf{T}_{\mathcal{C}}$.

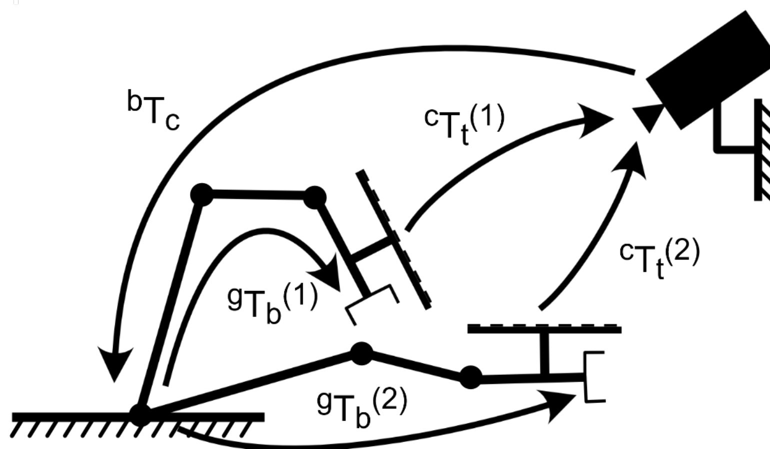


Figure 3.22: Eye-to-hand configuration schema

3.9 Pose estimation

The pose estimation pipeline starts with taking a picture of an object placed in the camera's field of view and detecting it along with its position the image.

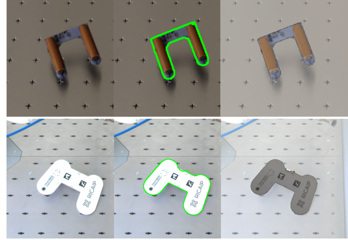


Figure 3.23: MegaPose estimation render & compare visualization

This detection output joined with the original image is then given to the MegaPose algorithm which, in turn, returns the transformation matrix ${}^{\mathcal{C}}\mathbf{T}_{\mathcal{T}}$. Visualizations of MegaPose render & compare results can be observed in figure 3.23 Depending on whether an OpenCV's or OpenGL's camera coordinate system is being used, a correction matrix is applied

$$
{}^{\mathcal{C}}\widetilde{\mathbf{T}}_{\mathcal{T}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} {}^{\mathcal{C}}\mathbf{T}_{\mathcal{T}}, \tag{3.4}
$$

which translates between the two mentioned coordinate systems visualized in figure 3.24.

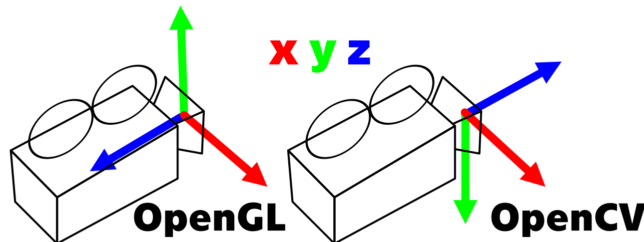


Figure 3.24: Difference in the camera coordinate system definition between standards of OpenGL and OpenCV

Next, offline-defined gripping poses of the detected object are iterated through, until a viable pose is found. The plausibility of the given gripping pose is checked in the robot controller itself using a function `Inverse`. If such position exists, the robot immediately goes to that position to pick the object, otherwise the pipeline return an error and awaits another command from the user (or planning system). The algorithm is further explained using a pseudocode 4.

Algorithm 4 Pose estimation pseudocode

1: Object is placed in front of a camera
2: User executes `Generate Data` command
3: image, object, bbox ← Camera with a detection model generates data
4:
5: ${}^{\mathcal{C}}\mathbf{T}_{\mathcal{T}} \leftarrow$ MegaPose estimates the 6D pose of the given object
6:
7: i ← 1
8: **while** i < N **do**
9: $\mathcal{GP} \leftarrow \mathcal{GP}_i \in (\mathcal{GP}_1, ..., \mathcal{GP}_N)$, $\mathcal{GP}$ is a gripping pose coord. system
10: ${}^{\mathcal{B}}\mathbf{T}_{\mathcal{GP}} \leftarrow {}^{\mathcal{B}}\mathbf{T}_{\mathcal{C}}\, {}^{\mathcal{C}}\widetilde{\mathbf{T}}_{\mathcal{T}}\, {}^{\mathcal{T}}\mathbf{T}_{\mathcal{GP}}$
11: **if** IKT solution for path to $\mathcal{GP}$ results in error **then**
12: i ← i + 1
13: **else**
14: Found suitable $\mathcal{GP}$ ← True
15: **break**
16: **end if**
17: **end while**
18:
19: **if** Found suitable $\mathcal{GP}$ **then**
20: **return** ${}^{\mathcal{B}}\mathbf{T}_{\mathcal{GP}}$
21: **else**
22: **throw** Exception
23: **end if**

3.10 Setting up physical camera

In the previous sections of this thesis, a virtual camera model was developed and tested to simulate the process of taking an image, however, without optical distortions. This section talks about the process of setting up a real imaging system to test and compare the results with the Digital Model and produce high-quality images in various lighting conditions.

The physical cell uses an industrial camera Basler acA2440-20gc equiped with Basler Lens C23-0816-2M-S f8mm. The lens allows to change the aperture stop size in range of F1.6 to F16. It is needed for the objects at the other end of the robot cell to be as sharp as the objects that are almost directly under the camera, hence, the highest F number is set-up on the lens. The downside is that this results in the lowest amount of light reaching the camera's chip and thus the images require longer exposure times. It is very important to perform the test of the F-stop setup before any of the parameter calibrations, intrinsic in particular, as changing the camera's F-stop and the focus distance changes the parameter values, as well.

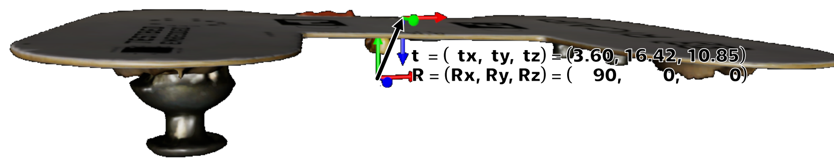


Figure 3.25: Absolute and gripping transformation for the object

Basler cameras come with pylon software development kit (SDK) offering programming interfaces for various programming languages. This thesis uses an open source Python wrapper called pypylon to comply with the code of the rest of the thesis which is also done in Python. A class was created, which connects to the camera using its serial number as an identificator. The first step after connecting to the camera is to set the Gain, Exposure time and Balance White values. The setup in this thesis uses camera's Auto Functions that set the mentioned values automatically accordingly to given constraints. The given camera has two profiles, one which minimizes gain and one minimizing exposure time. In this thesis, a profile minimizing the gain is used. This results in a lower frame rate, however, the images have close to no noise, unless the image is taken in very dim circumstances, such as in the night.

3.11 Defining the gripping positions

The reconstructed objects used in the used object dataset come in *.obj* files, which contain information about a coordinate system with an origin. Position of this origin is what the MegaPose algorithm returns estimation of. It is highly improbable that this origin together with the coordinate system's orientation would align with the gripping positions needed for gripping the objects. This stems from the fact that every gripper prefers a bit different gripping poses, depending on its geometry, as well as geometry of the object in question. In order for the robot to be able to move to the object and grasp it with the end effector, suitable positions need to be defined. This can be done opening any 3D rendering software able to open *.obj* file and creating a coordinate system with respect to the default one. With NVIDIA Omniverse, it is possible to do exactly that, the setup can be seen in figure 3.25.

Chapter 4

Experiments

4.1 Evaluating the object detection model

Assessing the object detection model's performance uses a range of metrics to provide valuable insights. These metrics give an understanding of how well the model works under given conditions, both in the simulated environment, which is the same as the synthetic image dataset was created in, and at the physical cell.

One set of metrics consists of precision and recall. Precision is defined as

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{\text{TP}}{\text{all detections}}, \text{ where}$$

TP is a number of true positive dectections and FP is a number of false positive detections. Recall is defined as

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{\text{TP}}{\text{all ground truths}}, \text{ where}$$

FN is false negative detection. Precision shows the model's ability to identify relevant objects, while recall describes the sensitivity with which the model predicts a given label. Together with these metrics comes precision-recall curve metric which plots precision and recall values concerning probability confidence threshold. Selecting such a threshold can be challenging, hence, average precision (AP) is introduced to evaluate the performance without

a dependency on the threshold. AP is the area under the precision-recall curve

$$\text{Average Precision (AP)} = \int_{r=0}^{1} p(r)dr \approx \frac{1}{11}\sum_{r=0.0}^{1.0} p(r), \text{ where}$$

p stands for precision and r stands for recall. Furthermore, AP values are computed for each class. To understand the model's performance over the whole dataset, the AP values can be averaged to get mean average precision (mAP):

$$\text{mAP} = \frac{1}{k}\sum_{i}^{k} \text{AP}_i.$$

To learn weights as well as evaluation of methods, a synthetic dataset was created using the tools created in section 3.6 on page 34. The dataset consists of 40'000 labeled images, which was split to three different groups - the training set, the validation set and the test set. Training batch can be seen in figure 4.1.

The training set is directly used for learning the weights of the used model and consists of 34'000 images which equals to 85 % of the whole dataset.

The validation set is used for evaluating the performance of the model during training, allowing for spotting the model's tendencies to overfit to training data. The phenomena of overfitting occurs when the model's weights start to fit the training data and stop working on newly introduced data[35]. The validation set consists of 6'000 images which equals to 15 % of the dataset. In the case of bad results on the validation set, a method of learning the weights, or the model itself can be changed.

To accurately evaluate the model's performance in the real use case scenario, a curated testing set is usually used once at the end of the whole process[36]. For this purpose, a set of 3'000 images of the scenario in virtual environment using the prepared scene is used, as well as 100 images from the physical cell. The performance of the model on the images from the virtual environment and the physical cell are shown separately and compared.

4.2 Evaluating methods in simulated environment

Evaluating the pose estimation process accuracy in simulated environment with a digital model comprises of comparing results of different parts of the

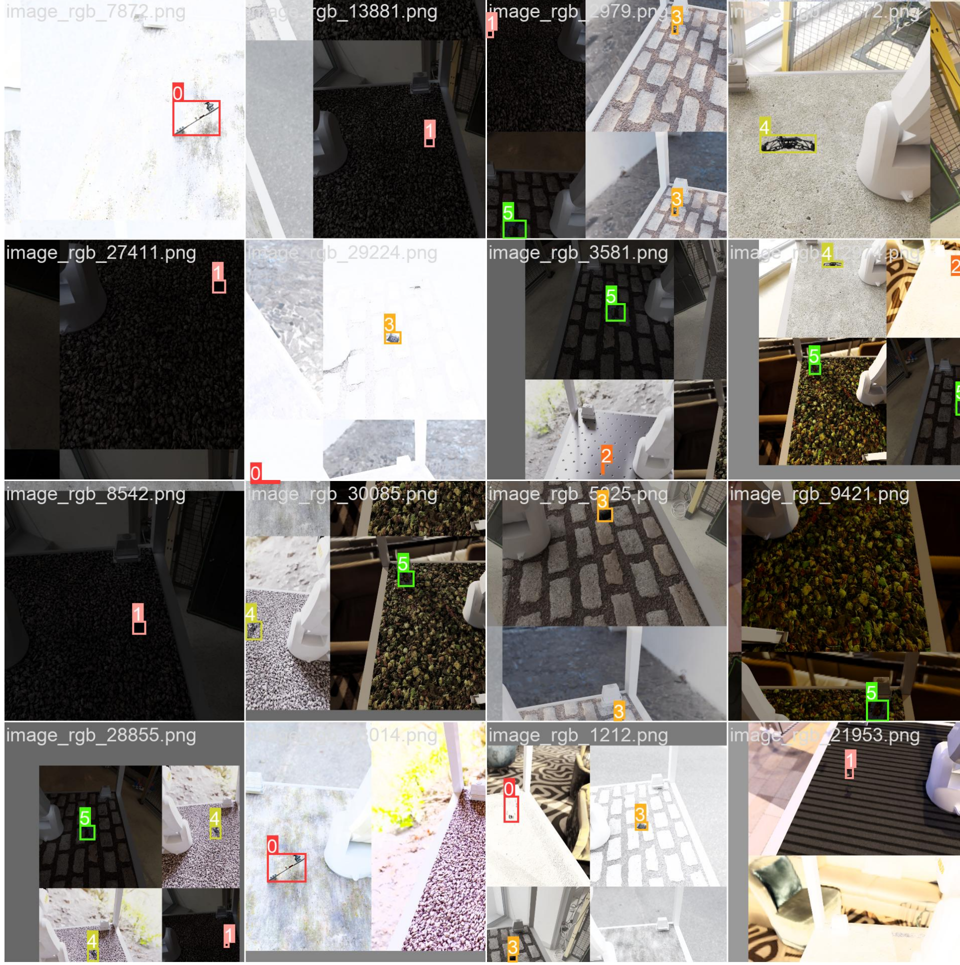


Figure 4.1: One of training batches

pipeline to the ground truth values taken from the simulated environment. Accuracy of three main areas of the pipeline are evaluated: the calibration of camera intrinsic parameters, the calibration of camera extrinsic parameters, and the performance of the MegaPose pose estimation algorithm.

Accuracy of the calibration of camera intrinsic values in the simulated environment can be directly compared to the values set in the simulating software. The comparison is quantified using relative errors

$$e = \frac{v_{\mathrm{GT}} - v_{\mathrm{meas}}}{v_{\mathrm{GT}}}$$

for each of the values, where v_{GT} represents ground truth value and v_{meas} represents the measured value.

Following the evaluation of the intrinsic calibration, accuracy of the camera's extrinsic parameters is evaluated using metrics of Euclidean distance

and angular differences to compare the measured coordinate frames against the camera's position in the object scene coordinate system. Using these metrics, 7 different methods of calibrating the camera's extrinsic parameters implemented in openCV are compared.

For performance review of the pose estimation algorithm, the same metrics as for the camera's extrinsic parameters are used. A total of 100 images is taken of the objects from the object dataset, together with their respective ground truth values. Then MegaPose algorithm is used for estimating the pose and then those values are compared.

4.3 Evaluating methods at the physical cell

The evaluation of the aforementioned methods in a physical robot cell presents an additional challenge as the ground truth values are not present anymore. In the real-word industry setting, alternative evaluation techniques are used for measuring the accuracy of different parts of the pipeline.

The accuracy of the pose estimation algorithm together with extrinsic calibration is evaluated using a two-step process. First, the robot is moved to what is estimated by the algorithm as the optimal gripping pose. Then, the robot is moved by hand to what is expected to be the correct gripping pose. The Euclidean distance between these frames is used as a metric. Evaluating the accuracy of the angle estimation by hand is challenging as matching a precise angle with round suction gripper is nearly impossible. Hence, instead of using difference in angle, only a success rate of gripping after performing the correction is recorded.

Chapter 5

Results

5.1 Object detection using YOLOv8

Performance of the object detection model on both the virtual and physical testing set is separately presented using a table (tables 5.1 and 5.2) and a confusion matrix (figure 5.1 on page 55 and figure 5.2 on page 56). The tables show the number of instances in which objects from the dataset represented by a row appear in the test set, precision, recall, mAP calculated at an IoU of threshold of 50 % and at a threshold of 50-95 %. The confusion matrices consist of rows and columns representing labels and predictions, respectively. A perfect scenario would consist of a convolution matrix with the only nonzero elements lying on the diagonal, meaning all predictions would be equal to the ground truth label.

The results of predicting on the testing set from the simulated environment are almost perfect, with only 4 of 3’000 images being predicted wrong. This, of course, hints at an underlying problem. Even though the training data set was created using randomized camera position, textures of the cell and also textures in the background, and the test set pictures being unseen, with the size of the dataset of 40’000 images, the randomization of the scene is not broad enough, resulting in the model correctly predicting vast majority of images.

More interesting results can be observed with the physical cell testing set. As can be seen in the table 5.2, the precision of the model is relatively high,

Class	Instances	P	R	mAP50	mAP50-95
all	3000	0.999	0.999	0.995	0.975
d01_controller	500	0.999	1.000	0.995	0.991
d02_servo	500	1.000	1.000	0.995	0.934
d03_main	500	0.996	0.992	0.995	0.969
d04_motor	500	0.999	1.000	0.995	0.984
d05_axle_front	500	0.999	1.000	0.995	0.985
d06_battery	500	0.999	1.000	0.995	0.989

Table 5.1: Image detection metrics on the testing set from virtual environment

with the lowest value of 93.3 % for the *d03_main* object, while recall seems to drop a bit lower, with the lowest value of 79.7 % for the object of *d04_motor*. In most cases, trying to reach a higher recall results in lowering the precision as a trade-off which would go directly against the needs of an industrial setting, where most of the applications are rather sensitive to precision, than to recall, due to safety reasons. It is much better for an object to go undetected, leading to a pipeline being stopped in the beginning and allowing to reconfigure, than miss-classifying an object as some other one, resulting in the equipment possibly colliding unexpectedly, creating all sorts of hazards.

Class	Instances	P	R	mAP50	mAP50-95
all	92	0.964	0.868	0.93	0.656
d01_controller	26	1	0.844	0.914	0.695
d02_servo	16	0.991	0.875	0.954	0.575
d03_main	12	0.933	1	0.995	0.873
d04_motor	11	1	0.797	0.943	0.612
d05_axle_front	12	1	0.89	0.938	0.656
d06_battery	15	0.862	0.8	0.836	0.522

Table 5.2: Image detection metrics on the testing set from physical cell

5.2 Pose estimation using MegaPose in simulated environment

With the ground truth values of camera matrix as well as the estimated values of camera matrix obtained through camera calibration, difference of the values can be calculated (equation 5.3). With the pinhole model projection equation

$$\Delta x = f_x \cdot \frac{x}{z} = \left| x = 0.3 \text{ m}, \ z = 1 \text{ m}, f_x = 7.47 \cdot 10^{-3} \text{ m} \right| = 2.241\text{mm},$$

where x is distance of an object from the optical axis, z is distance of the object from the camera, the error of an observed position on a given axis can be

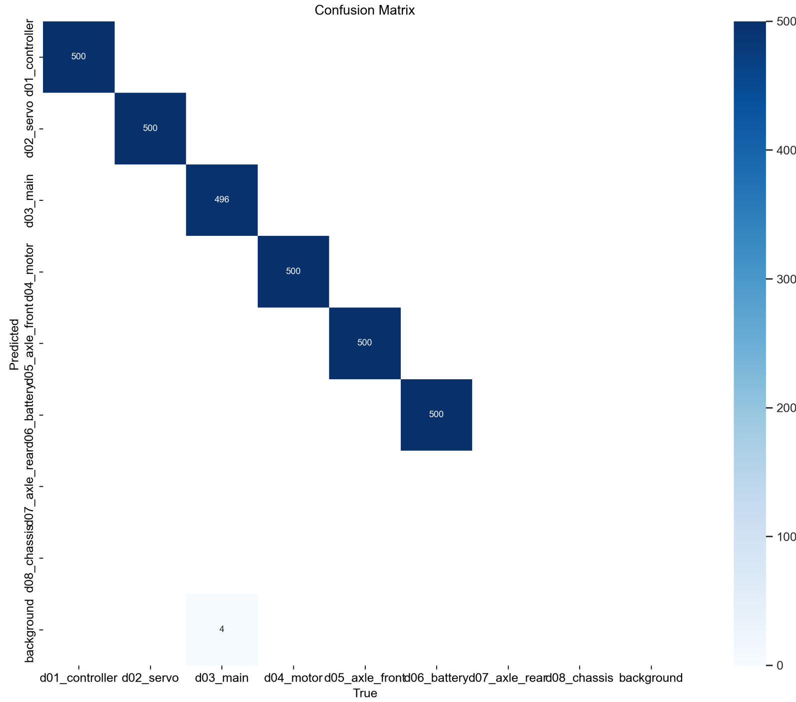


Figure 5.1: Confusion matrix of labels and predictions on the virtual environment testing set

calculated. With the objects in the scene being roughly in the distance of 1 meter from the camera and up to 0.3 metres from the optical center, we can expect an error of 2.2 millimeters due to the intrinsic calibration error only.

$$K_{\mathrm{GT}} = \begin{bmatrix} 886.93 & 0.0 & 512.0 \\ 0.0 & 886.93 & 512.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}, \tag{5.1}$$

$$K_{\mathrm{est}} = \begin{bmatrix} 879.52 & 0.0 & 515.86 \\ 0.0 & 878.65 & 527.65 \\ 0.0 & 0.0 & 1.0 \end{bmatrix}, \tag{5.2}$$

$$\Delta K = K_{\mathrm{GT}} - K_{\mathrm{est}} = \begin{bmatrix} 7.47 & 0.0 & 9.37 \\ 0.0 & 7.56 & 2.62 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \tag{5.3}$$

Calibration of the camera's extrinsics was done using a total of 7 methods, with the first 5 implemented in OpenCV's `calibrateHandEye` method and the last two implemented in `calibrateRobotWorldHandEye`. Two methods, namely method number 3 [37] and method number 6 [38], give results

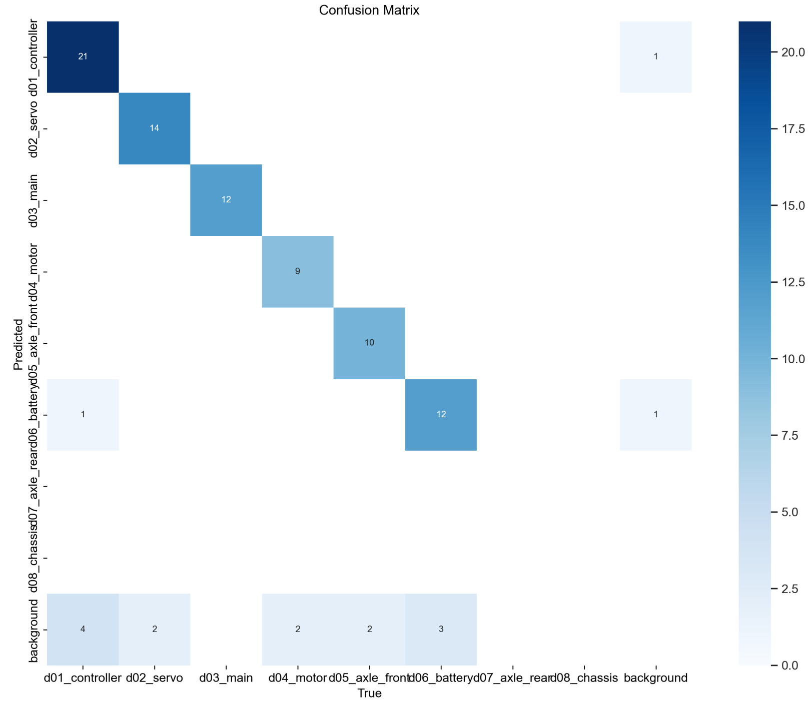


Figure 5.2: Confusion matrix of labels and predictions on the physical cell testing set

with accuracy of less than a millimeter and no angular error. For this reason, an error of the resulting pose estimation of 1 millimeter is given by the extrinsics calibration. The other methods, except for method 4 [39], give accuracy of up to 5 millimeters. Method 4 [39] is very far off with 66 milimetres of an error in the translational part of the extrinsics, which hints at a faulty implementation in the OpenCV library. The results can be observed in table 5.3.

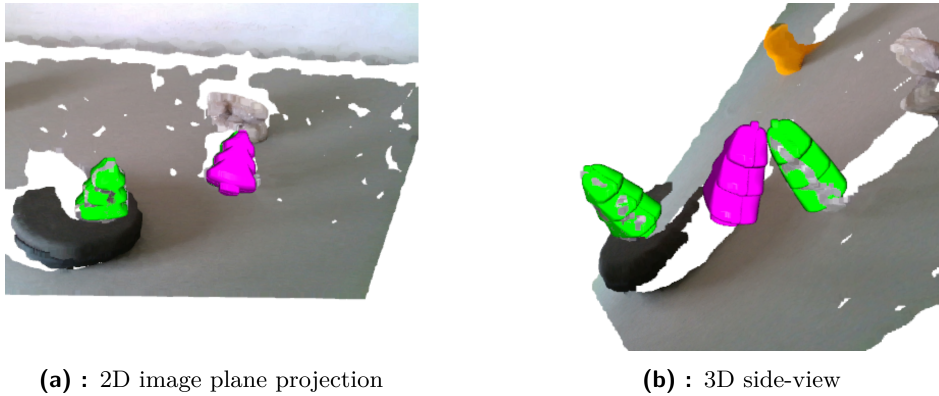
Method no.	**Δx [mm]**	**Δy [mm]**	**Δz [mm]**	**Δe [mm]**	**Δθ [°]**
0	1.30	-2.28	-4.20	4.95	0.15
1	0.05	-0.94	1.28	1.59	0.00
2	0.05	-0.93	1.30	1.60	0.00
3	**-0.18**	**0.46**	**-0.17**	**0.52**	**0.00**
4	-5.46	48.04	-45.27	66.23	0.12
5	-0.21	0.72	2.63	2.73	0.02
6	0.00	-0.73	0.62	0.96	0.00

Table 5.3: Comparison of 7 different methods of camera extrinsics calibration

MegaPose estimation errors are presented in table 5.4. First column shows average error per-axis μ and also a standard deviation value σ, while the second shows the average Euclidean distance error. The third shows the same metric

as the first one, only with an angle difference. The angles are shown in Euler intrinsic Z $\rightarrow$ Y $\rightarrow$ X format, which directly maps to the a, b, c corrections of the robot's tool center point frame in the robot's controller. The fourth column shows the average angle error around a vector.

The results of the estimation using the camera setup from the simulated environment clearly show, that the setup is not suitable for the problem. An interesting phenomena of the highest errors appearing in the z axis, which is the case for 5 out of 6 objects tested. This means that the model shows a problem with estimating the objects' distance from the camera. However, there is a possible explanation, as the probability of the rendered CAD model moving significantly out of the bounding box made by the object detector is lower than positioning the rendered model closer to (or, further from) the camera and rotating it in a way such that the outline of the object matches the outline of the object in the scene (figure 5.3). This is further supported by the high error values in the rotational part of the 6D estimation.



(a) : 2D image plane projection **(b) :** 3D side-view

Figure 5.3: Visualization of faulty object estimation appearing correct in the image plane (magenta object is the rendered model in the estimated pose, green is the ground truth pose)

The tendency to rely on the object's outline can be observed in one of two scenarios - using a CAD model without a texture or the objects being too small in the context of the picture. With 5 out of 6 used objects being textured with very distinctive features, such as April tags, it hints to the other option. In that case, opting out for a camera lens with higher zoom and smaller field of view would be advisable.

Class	$\mu_{\Delta x} \pm \sigma_{\Delta x}$ $\mu_{\Delta y} \pm \sigma_{\Delta y}$ $\mu_{\Delta z} \pm \sigma_{\Delta z}$ [mm]	$\mu_{\Delta e} \pm \sigma_{\Delta d}$ [mm]	$\mu_{\Delta R_Z} \pm \sigma_{\Delta R_Z}$ $\mu_{\Delta R_Y} \pm \sigma_{\Delta R_Y}$ $\mu_{\Delta R_X} \pm \sigma_{\Delta R_X}$ [°]	$\mu_{\Delta \theta} \pm \sigma_{\Delta \theta}$
d01_ controller	2.65 ± 4.07 4.30 ± 8.06 8.61 ± 7.67	10.60 ± 11.30	10.09 ± 26.15 7.67 ± 18.19 5.04 ± 13.43	55.38 ± 105.87
d02_ servo	2.05 ± 4.42 2.32 ± 3.95 14.68 ± 25.63	15.28 ± 26.15	1.97 ± 3.54 2.52 ± 3.57 5.25 ± 13.84	22.04 ± 44.44
d03_ main	4.74 ± 5.97 5.05 ± 6.82 12.86 ± 14.57	16.32 ± 15.54	11.65 ± 25.29 2.54 ± 3.56 9.00 ± 15.01	57.40 ± 88.40
d04_ motor	3.48 ± 6.48 2.76 ± 4.49 15.60 ± 21.04	16.54 ± 22.23	30.41 ± 63.55 1.38 ± 1.52 0.99 ± 1.07	96.99 ± 198.91
d05_ axle_ front	3.73 ± 7.38 7.18 ± 14.22 17.40 ± 13.47	20.87 ± 19.26	11.12 ± 25.75 5.05 ± 7.79 18.32 ± 45.33	70.44 ± 132.95
d06_ battery	10.29 ± 25.09 2.20 ± 3.20 8.21 ± 8.16	16.79 ± 24.55	22.07 ± 56.58 0.78 ± 0.88 23.28 ± 58.94	75.07 ± 184.64

Table 5.4: Object estimation errors using MegaPose

5.3 Pose estimation using MegaPose at the physical cell

Real-world environment experiment results are presented using the same metrics (table 5.5) as in the previous case of the simulated environment. There is one major difference, however, as in the case of table 5.4, the axes of x, y, z and their respective error values were in the coordinate system of a camera, while in this case, the experiment uses the coordinate system of the object estimation. Hence, the same conclusions cannot be drawn in terms of the z-axis, as the effect of wrongly estimating the objects' distance to the camera randomly distributes into all three values, depending on the estimated orientation of the object. It can be seen, that the magnitude of the error values is comparable to that of the simulated environment testifying the authenticity of the developed simulation setup.

Class	$\mu_{\Delta x} \pm \sigma_{\Delta x}$ $\mu_{\Delta y} \pm \sigma_{\Delta y}$ $\mu_{\Delta z} \pm \sigma_{\Delta z}$ [mm]	$\mu_{\Delta d} \pm \sigma_{\Delta d}$	Success rate
d01_ controller	0.10 ± 0.24 0.71 ± 1.75 2.15 ± 3.06	2.84 ± 3.09	100.0 %
d02_ servo	2.86 ± 3.35 4.03 ± 5.48 10.14 ± 9.67	13.06 ± 9.57	62.5 %
d03_ main	20.08 ± 21.01 19.24 ± 21.04 7.25 ± 5.92	31.12 ± 27.87	25.0 %
d04_ motor	5.29 ± 3.70 25.43 ± 15.37 27.38 ± 10.19	39.02 ± 15.99	20.0 %
d06_ battery	6.89 ± 4.19 7.94 ± 3.97 14.67 ± 7.75	18.68 ± 8.38	50.0 %

Table 5.5: Object estimation errors of the whole pipeline (intrinsic, extrinsic, MegaPose)

5.4 Transition from Digital Model to physical cell

Four separate programs were created in the simulation development part of the thesis - an industrial PC control script acting as high-level controller, a robot program and an Omniverse interactive application together with an offline dataset generation script. Three of which are meant to be deployed into the physical environment.

The control script itself allows transitioning to the physical environment and back with a flip of a switch. The modular design with which it has been developed together with the communication protocols of sockets and OPC UA allows for seamless transition between the two worlds. A possible improvement in this area would be changing the script in a way allowing for concurrent running of the simulated and physical setup.

The robot program implemented in the KUKA.OfficeLite VRC can be transferred without any changes to the real controller. However, with the current setup, it is required to manually download the program either using a flash disk or KUKA.WorkVisual and then uploading it onto the real robot using the same method. In the future, this could be improved using packages

allowing for downloading and uploading robot program files through local area network.

The interactive application is the only part, which cannot be directly transferred into the physical world, as physical world cameras require working with their respective drivers. As such, every time a camera needs to be used, a program taking care of its drivers and methods needs to be developed. However, in the case of reusing cameras from the same manufacturer, there is a possibility these drivers are all controlled in the same, or very similar, way, lowering the time spent on the deployment to the physical world. Furthermore, the main control script comes with a universal interface for any image generating module. This allows for an engineer to develop the camera control script concurrently with the development of the Digital Model, due to an already established expected behaviour.

Conclusion

This thesis has explored the process of developing a robotic pick and place cell within both virtual and physical environments. By leveraging advanced technologies and simulation platforms, most notably NVIDIA Omniverse, the study aimed to enhance the current methodology of virtual commissioning and robotic simulations in the automation industry.

The thesis begins with a review of the current state-of-the-art in 6D pose estimation algorithms, selecting MegaPose for its compatibility with using a static RGB camera without depth. This selection was followed by the development of a virtual robotic cell scene using the chosen simulation tools of Process Simulate and KUKA.OfficeLite.

After the initial setup of the scene, it was exported into NVIDIA Omniverse, where textures and lighting were applied together with the camera renderer effects to mimic the real camera view as closely as possible. With these properties set-up, the thesis then moves on to create two separate NVIDIA Omniverse tools - the image dataset generation tool used for creating a dataset for training of a machine learning model, and the interactive application which serves the true purpose of a Digital Model[1].

The latter comes with a fully developed modular architecture consisting of a high-level controller with a universal interface for working with any image producing system and an OPC UA interface allowing for a unified

[1] All of the developed tools can be obtained at https://github.com/testbedCIIRC/Robotic-Object-Manipulation-in-Virtual-and-Physical-Environment

manufacturer independent communication with the robot controller. Thanks to these properties, the proposed system is scalable, flexible and invites a workflow where multiple engineers can concurrently work on the divided modules. Furthermore, the simulation software used for the implementation enables very smooth transition from the virtual to physical environment, with some parts needing no reconfiguration and a part requiring only a transfer of files from one device to another, without a need to change any of the code.

The performance of the solution is assessed using various metrics on different parts of the pipeline. The first part of the experimentation phase evaluates the trained detection model YOLOv8. It was trained on a dataset of 34'000 purely synthetic images of objects from a selected object dataset. With the average precision of 96.4 % across the whole object dataset on images taken from the physical cell's camera, it was proved that the synthetic dataset can be used to train the object detection model.

Pose estimation performance evaluation was done both in virtual and physical environments. The pose estimation experiments revealed that the pose estimation accuracy with the current setup is far from reaching the required values set by the design of the used object dataset. However, it also shows that the developed Digital Model simulates the physical environment even with its limitations, arriving at the same conclusions across domains. This was not an expected outcome in the beginning of the development and is a success in and of itself.

Hence, this thesis proposes a number ways that could potentially enhance the performance of the setup. Firstly, using a camera lens with higher zoom and smaller field of view to prevent the estimation method from relying solely on the object's outline in the image due to a lack of details caused by the objects being too small in the captured images. Another possibility, should it be impossible to change the camera lens, is to redesign the whole cell and mount the camera to the robot's flange. This comes with an upside of being able to control how close the camera is to the estimated objects, but also a downside of limiting the robot's workspace, especially in the case of such a small robot as the one used in this thesis.

Bibliography

[1] Omniverse Platform for OpenUSD Development and Collaboration | NVIDIA. Online. Dostupné z: https://www.nvidia.com/en-us/omniverse/. [cit. 2024-05-24].

[2] Robotic Manipulation. Online. Robotics at Leeds. Dostupné z: https://robotics.leeds.ac.uk/research/ai-for-robotics/robotic-manipulation/. [cit. 2024-05-11].

[3] KUKA ready2_educate – Training cell for hands-on robotic training. Online. KUKA AG. Dostupné z: https://www.kuka.com/en-de/products/robot-systems/ready2_use/kuka-ready2_educate. [cit. 2024-05-13].

[4] MERWANSKY. What is 6D Object Pose Estimation in Computer Vision? Online. In: Medium. 2022. Dostupné z: https://just-merwan.medium.com/what-is-6d-object-pose-estimation-in-computer-vision-21e8acf9e3e2. [cit. 2024-05-11].

[5] HE, Zaixing; FENG, Wuxi; ZHAO, Xinyue a LV, Yongfeng. 6D Pose Estimation of Objects: Recent Technologies and Challenges. Online. Applied Sciences. 2021, roč. 11, č. 1. ISSN 2076-3417. Dostupné z: https://doi.org/10.3390/app11010228. [cit. 2024-05-11].

[6] LABBÉ, Yann, et al. CosyPose: Consistent multi-view multi-object 6D pose estimation. Online. ECCV 2020. 2020. ISSN 1611-3349. Dostupné z: https://doi.org/10.48550/arXiv.2008.08465. [cit. 2024-05-11].

[7] LABBÉ, Yann, et al. MegaPose: 6D Pose Estimation of Novel Objects via Render & Compare. Online. 2022. Dostupné z: https://doi.org/10.48550/arXiv.2212.06870. [cit. 2024-05-11].

[8] LIN, Yunzhi; TREMBLAY, Jonathan; TYREE, Stephen; VELA, Patricio A. a BIRCHFIELD, Stan. Single-Stage Keypoint- Based Category-Level Object Pose Estimation from an RGB Image. Online. 2022 International Conference on Robotics and Automation (ICRA). 2022, s. 1547-1553. ISBN 978-1-7281-9681-7. Dostupné z: https://doi.org/10.1109/ICRA46639.2022.9812299. [cit. 2024-05-12].

[9] WEN, Bowen; YANG, Wei; KAUTZ, Jan a BIRCHFIELD, Stan. FoundationPose: Unified 6D Pose Estimation and Tracking of Novel Objects. Online. Dostupné z: https://doi.org/10.48550/arXiv.2312.08344. [cit. 2024-05-24].

[10] LI, Yi; WANG, Gu; JI, Xiangyang; XIANG, Yu a FOX, Dieter. DeepIM: Deep Iterative Matching for 6D Pose Estimation. Online. International Journal of Computer Vision. 2020, roč. 128, č. 3, s. 657-678. ISSN 0920-5691. Dostupné z: https://doi.org/10.1007/s11263-019-01250-9. [cit. 2024-05-24].

[11] FISCHLER, Martin A. a BOLLES, Robert C. Random sample consensus. Online. Communications of the ACM. 1981, roč. 24, č. 6, s. 381-395. ISSN 0001-0782. Dostupné z: https://doi.org/10.1145/358669.358692. [cit. 2024-05-24].

[12] BOP: Benchmark for 6D Object Pose Estimation. Online. Dostupné z: https://bop.felk.cvut.cz/leaderboards/pose-estimation-unseen-bop23/core-datasets/. [cit. 2024-05-24].

[13] GRIEVES, Michael. Origins of the Digital Twin Concept. Online. 2016. Dostupné z: https://doi.org/10.13140/RG.2.2.26367.61609. [cit. 2024-05-11].

[14] KRITZINGER, Werner. Digital Twin in manufacturing: A categorical literature review and classification. Online. IFAC-PapersOnLine. 2018, roč. 51, č. 11, s. 1016-1022. ISSN 2405-8963. Dostupné z: https://doi.org/10.1016/j.ifacol.2018.08.474. [cit. 2024-05-11].

[15] LECHLER, Tobias; FISCHER, Eva; METZNER, Maximilian; MAYR, Andreas a FRANKE, Jörg. Virtual Commissioning – Scientific review and exploratory use cases in advanced production systems. Online. Procedia CIRP. 2019, roč. 81, s. 1125-1130. ISSN 22128271. Dostupné z: https://doi.org/10.1016/j.procir.2019.03.278. [cit. 2024-05-12].

[16] KUKA.OfficeLite. Online. Industrial intelligence 4.0_beyond automation | KUKA AG. Dostupné z: https://www.kuka.com/en-gb/products/robotics-systems/software/simulation-planning-optimization/kuka_officelite. [cit. 2024-05-12].

[17] Process Simulate software. Online. Siemens Digital Industries Software | Siemens Software. Dostupné z: https://plm.sw.siemens.com/en-US/tecnomatix/products/process-simulate-software/. [cit. 2024-05-12].

[18] PROFINET - The Leading Industrial Ethernet Protocol: PROFINET. Online. Dostupné z: https://www.profinet.com/. [cit. 2024-05-24].

[19] GALLAGHER, Kelly. Discover what's new in Tecnomatix 2307 (August 2023). Online. Blog Network | Siemens Digital Industries Software. 2023. Dostupné z: https://blogs.sw.siemens.com/tecnomatix/discover-whats-new-in-tecnomatix-august-2023/. [cit. 2024-05-13].

[20] GALLAGHER, Kelly. Battery pack assembly use case: the future of Process Simulate & the Industrial Metaverse. Online. Blog Network | Siemens Digital Industries Software. 2023. Dostupné z: https://blogs.sw.siemens.com/tecnomatix/battery-pack-assembly-use-case-the-future-of-process-simulate-and-the-industrial-metaverse/. [cit. 2024-05-13].

[21] What Is Isaac Sim? Online. Omniverse IsaacSim latest documentation. 2024. Dostupné z: https://docs.omniverse.nvidia.com/isaacsim/latest/index.html. [cit. 2024-05-13].

[22] Platform Overview. Online. Omniverse Platform latest documentation. 2024. Dostupné z: https://docs.omniverse.nvidia.com/platform/latest/index.html. [cit. 2024-05-13].

[23] Nucleus Overview. Online. Omniverse Nucleus latest documentation. 2024. Dostupné z: https://docs.omniverse.nvidia.com/nucleus/latest/index.html. [cit. 2024-05-13].

[24] Connect Overview. Online. Omniverse Connect latest documentation. 2024. Dostupné z: https://docs.omniverse.nvidia.com/connect/latest/index.html. [cit. 2024-05-13].

[25] Kit Architecture. Online. Omniverse latest documentation. 2024. Dostupné z: https://docs.omniverse.nvidia.com/dev-guide/latest/kit-architecture.html. [cit. 2024-05-13].

[26] Simulation. Online. Omniverse Platform latest documentation. 2024. Dostupné z: https://docs.omniverse.nvidia.com/platform/latest/simulation.html. [cit. 2024-05-13].

[27] ZEMAN, Vít. Benchmarking 6D Object Pose Estimation for the Pick and Place Task. Online, Master thesis. Prague, 2024. Dostupné z: http://hdl.handle.net/10467/113393. [cit. 2024-05-24].

[28] RANJAN, Sapkota; DAWOOD, Ahmed; MANOJ, Karkee a MANOJ. Comparing YOLOv8 and Mask RCNN for object segmentation in complex orchard environments. Online. 2023. Dostupné z: https://doi.org/10.48550/arXiv.2312.07935. [cit. 2024-05-14].

[29] CAMACHO, Jean Carlo a MOROCHO-CAYAMCELA, Manuel Eugenio. Mask R-CNN and YOLOv8 Comparison to Perform Tomato Maturity Recognition Task. Online. Information and Communication Technologies. Communications in Computer and Information Science. 2023, s. 382-396. ISBN 978-3-031-45437-0. Dostupné z: https://doi.org/10.1007/978-3-031-45438-7_26. [cit. 2024-05-14].

[30] KR 3 R540 - 0000270971_EN.pdf. Online. Dostupné z: https://www.kuka.com/-/media/kuka-downloads/imported/-8350ff3ca11642998dbdc81dcc2ed44c/0000270971_en.pdf. [cit. 2024-05-24].

[31] KUKA System Software 8.6 Operating and Programming Instructions for System Integrators. PDF. KUKA Deutschland, 2019.

[32] SCHLEIPEN, Miriam; GILANI, Syed-Shiraz; BISCHOFF, Tino a PFROMMER, Julius. OPC UA & Industrie 4.0 - Enabling Technology with High Diversity and Variability. Online. Procedia CIRP. 2016, roč. 57, s. 315-320. ISSN 22128271. Dostupné z: https://doi.org/10.1016/j.procir.2016.11.055. [cit. 2024-05-15].

[33] Camera Calibration. Online. OpenCV. Dostupné z: https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html. [cit. 2024-05-16].

[34] 10.3. Offline Dataset Generation — Omniverse IsaacSim latest documentation. Online. Dostupné z: https://docs.omniverse.nvidia.com/isaacsim/-latest/replicator_tutorials/tutorial_replicator_offline_generation.html. [cit. 2024-05-24].

[35] What is overfitting? Online. IBM. Dostupné z: https://www.ibm.com/topics/overfitting. [cit. 2024-05-24].

[36] SHAH, Tarang. About Train, Validation and Test Sets in Machine Learning. Online. In: Towards Data Science. 2017. Dostupné z: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7. [cit. 2024-05-24].

[37] ANDREFF, Nicolas; HORAUD, Radu a ESPIAU, Bernard. On-line hand-eye calibration. Online. 1999. Dostupné z: https://dl.acm.org/doi/10.5555/1889712.1889775. [cit. 2024-05-24].

[38] LI, Aiguo; WANG, Lin a WU, Defeng. Simultaneous robot-world and hand-eye calibration using dual-quaternions and Kronecker product. Online. International journal of physical sciences. 2010, roč. 5. Dostupné z:

https://www.researchgate.net/publication/268056839_Simultaneous_robot-world__and__hand-eye__calibration__using__dual-quaternions__and__Kronecker__product. [cit. 2024-05-24].

[39] The International Journal of Robotics Research. Online. 1999, roč. 18, č. 3. 1999. ISSN 0278-3649. Dostupné z: http://journals.sagepub.com/doi/10.1177/02783649922066213. [cit. 2024-05-24].