Year 2023

MASTER EU4M in Mechatronic Engineering

Domain: Science - Technology - Health

Mention: Complex Systems Engineering

Mechatronics - Processes and Microtechnologies

Master's thesis

Development of AI-based control system for 6-axis robotic arm

Andrews Björn

Sistemas Multisensor y Robótica (Universidad de Oviedo)

C. Pedro Puig Adam, 33204 Gijón, Asturias, Spain

École Nationale Supérieure de Mécanique et des Microtechniques

26, chemin de l'Épitaphe – 25030 BESANÇON – France - Tél. 03 81 40 27 31 - Fax 03 81 80 98 70 - www.ens2m.fr

Abstract

Development of AI-based control system for 6-axis robotic arm

Beside specialized high-volume handling tasks in the industry robotic systems nowadays need also to react fast and flexibly to different object shapes. The increasing computation power and low-cost sensor devices make computer vision-based systems a viable solution for solving such requests through fast and high resolution object detection with artificial intelligence.

In this work, a pick and place system with AI-based computer vision is developed. It consists of a station with an UR3e robot by Universal robots with an attached RGI14 gripper by DH-Robotics and a RealSense L515 RGB-D camera by Intel RealSense. Before establishing the real setup, a simulation model of the system is created to proof the functionality. Both the simulation model and the control system for the real setup are designed in the environment of RoboDK (v5.6.0) and MATLAB (Version R2023a). The use of the RoboDK API for MATLAB and the RealSense MATLAB wrapper allows the communication and data handling between the MATLAB environment and the RoboDK software as well as the RealSense L515 camera. For the object detection, a tiny YOLOv4 object detection network trained with synthetic image data is applied. The pose estimation is performed with a PCA-based algorithm proposed by MathWorks.

The functionality of computer vision implemented in the pick and place system allows to detect and classify plastic bricks randomly placed with the color image data of the RealSense L515 camera. The processing of the camera's depth data enables the pose estimation of the detected bricks. By controlling the integrated RGI14 gripper, the localized bricks can be picked and placed in different boxes according to their predicted classification. The motion control and trajectory planning towards the computed target position is executed by the use of generic movement instructions of RoboDK and its built-in post-processor for Universal Robots.

The experimental results show that the simulation model demonstrates robust functionality by detecting randomly generated bricks and estimating their poses. The trained object detection model effectively classifies various bricks, enabling sorting into color-coded bins. Transitioning the simulation model to a real setup validated its practicality, detecting most plastic bricks. However, using solely synthetic data led to occasional inaccuracies, particularly in detecting orange bricks and misclassifying them as yellow. Some non-detected bricks affect the subsequent pose estimation due to missing position data.

Table of Contents

List of Figures

List of Tables

Keywords

AI	Artificial Intelligence
API	Application Programming Interface
CAD	Computer Aided Design
L515	camera model of Intel RealSense
LiDAR	Light Detection and Ranging
PC	Personal Computer
PCA	Principal Component Analysis
RGB-D	Red Green Blue – Depth
RGI14	gripper model of DH-Robotics
TCP	Tool Center Point
UR3e	robot model of Universal robots
YOLOv4	version 4 of object detection network „You Only Look Once“

1 Introduction

The present work has been done in the laboratory of the "Grupo de Sistemas Multisensor y Robótica" of the Universidad de Oviedo. The research activities of the group focus on the interplay between Human Movement Science, Sensor Systems, and Robotics. Specifically, it focuses on the development of enabling technologies for the monitoring and understanding of human activity [1]. For the present work, the equipment available in the laboratory of the robotics section is used.

1.1 Motivation and objectives

Robotic arms in combination with grippers are well suited to perform repetitive handling tasks, sparing human workers from monotonous work and avoiding mental boredom and physical fatigue. Handling tasks are the largest robot application field in all branches of manufacturing and logistics. A significant challenge in designing robotic handling systems involves creating effective grippers and grasping strategies that account for factors like the object's properties, and uncertainties in geometry and location. Currently, most objects are delivered to robot stations in a consistent manner, often via specialized carriers, magazines, or vibrating mechanisms that align parts predictably for robot manipulation. However, in the context of manufacturing automation, the need for cost-effectiveness and adaptability is leading to a shift from custom parts magazines to more versatile carriers, containers, or conveyor belts. When parts are randomly oriented on a conveyor belt or within a carrier, effective identification and precise localization become essential for the robot to perform collision-free grasping. The complex task of enabling robots to grasp partially ordered or randomly arranged components is commonly referred to as bin-picking. [2]
With the addition of artificial intelligence, the pre-programmed movements can be made more flexible and adaptive to a changing environment. Computer vision-based systems can react to the objects they work with, and, for example, perform different movements depending on the handled object. With the help of AI, the system can rapidly adapt to unknown and randomly placed objects to handle.

The goal of this work is to develop a handling system like a bin-picking or pick and place application and to extend the station with AI-based object detection in order to handle randomly placed parts and to implement a sorting algorithm in the system.

The Universal Robot UR3e available in the robotics laboratory of the Universidad de Oviedo has not yet been used for pick and place applications, since no functioning gripper has been installed. At this point, the UR3e has only been used to move to variable objects with markers using a motion capture system (Figure 1). This application requires multiple cameras to capture the position in the three-dimensional space, and different software

applications that need to communicate with each other.
So, the objective is to make the existing UR3e robot capable for a pick and place application by installing a gripper. In addition, a computer vision capability of the system should be enabled so that the robotic system can react to a random and changing environment by integrating a camera. Additional necessary parts to assemble the different components are to be designed and manufactured. The hardware and software used are to be reduced to a minimum and the entire system is to be developed and controlled within the programming environment of MATLAB in order to be able to expand it with further functional blocks in the future.

For this application, a robotic simulation model is planned before setting up the final handling station. With this, the robot's movements can be tested virtually before the real use and all programs can be checked for functionality without the necessary hardware. This way, resources and time are saved for setting up a functional system and possible collisions of the robot damaging the equipment of the laboratory can be prevented from the beginning.
To react to new and unknown products in the industrial manufacturing context, the AI-based object detection algorithm should only use synthetically generated object or product data for training to simulate the early stage of product and manufacturing design.

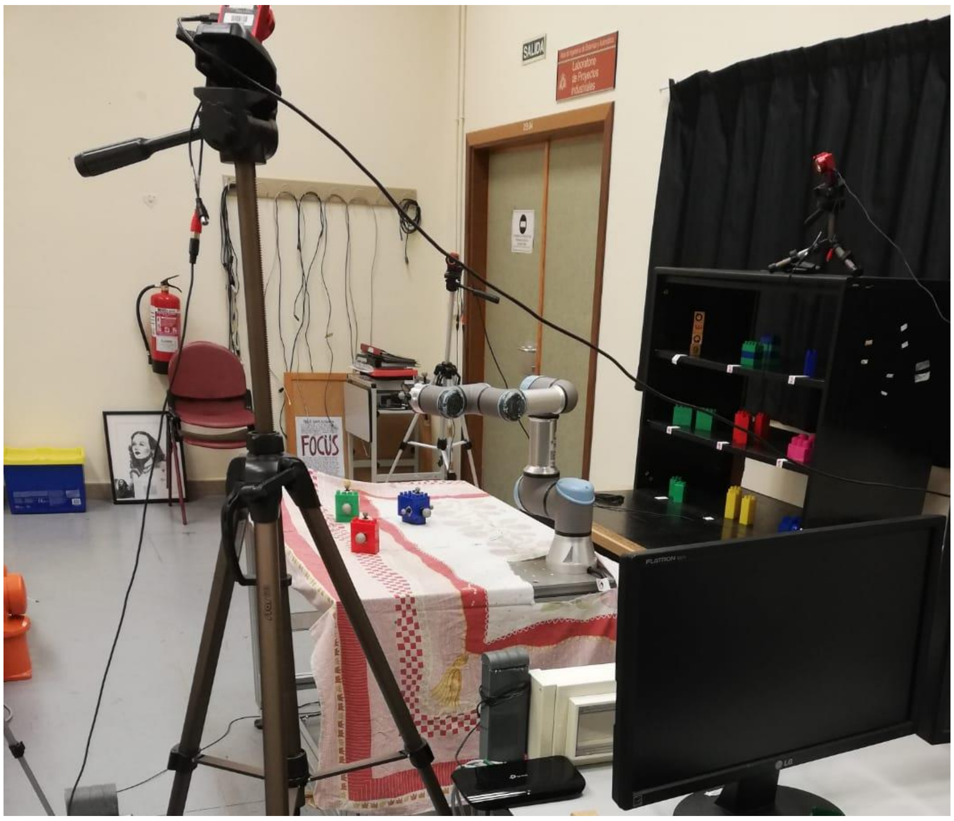


Figure 1: Initial state of laboratory

1.2 Content of the report

Chapter 2 introduces the reader to the theoretical foundations of this work. It also shows related work in the field of pick and place applications with computer vision.

Chapter 3 describes the development of the AI-based pick and place system. It begins with an overview of the system architecture, outlining the key components and their integration, both in terms of hardware and software. Furthermore, a simulation model of the system is developed and the used algorithms for object detection and pose estimation are explained. The pick and place programming is performed take advantage of the simulation model and the algorithms. The chapter concludes with details about the final setup of the system.

Chapter 4 describes the experiments for testing and validating the robotic simulation model and the final physical setup. The results are discussed analyzing the findings and displaying the limitations of the system in order to highlight potential areas for improvement.

Chapter 5 concludes the work and provides an outlook for possible follow-up work or extensions to the developed pick and place system.

2 Theoretical background and state of the art

In the following, the reader will be introduced to the technical principles that are applied in this work. These serve the better understanding of the solution presented in the following chapters.

2.1 Computer vision in robotics

Almost every animal use eyes, and the experience shows that eyes are effective sensors for recognition, navigation, obstacle avoidance and manipulation. Cameras mimic the function of eyes and can therefore be used to create vision-based abilities for robots. The computing power of nowadays and the existence of new algorithms and cheap sensors make vision a practical sensor for technical systems. [3]

This vision of technical system is called computer vision and consists in a set of techniques for extracting information from images, videos and point clouds. The most common techniques are image recognition, object detection, pose and motion estimation, and video tracking. [4]

For this work, the understanding of object detection is necessary. With object detection, e.g., persons or objects can be detected within a scene or bar codes can be read. To perform object detection, deep learning networks are trained with labeled image data. The label consists of a rectangle around the object, called bounding box, with its position data in the image, and its label name so that the object belongs to a specific class. After training and validating the deep learning network, the trained network can be applied as a detector on unknown image data. Objects which appear like the objects in the trainings data can be detected and its class is identified with a certain confidence score. This process is displayed in Figure 2. The confidence score is the probability given by the network that the detected object is of the predicted class. The detected position in the scene and the predicted class can then be used for further processing and decision-taking within algorithms.

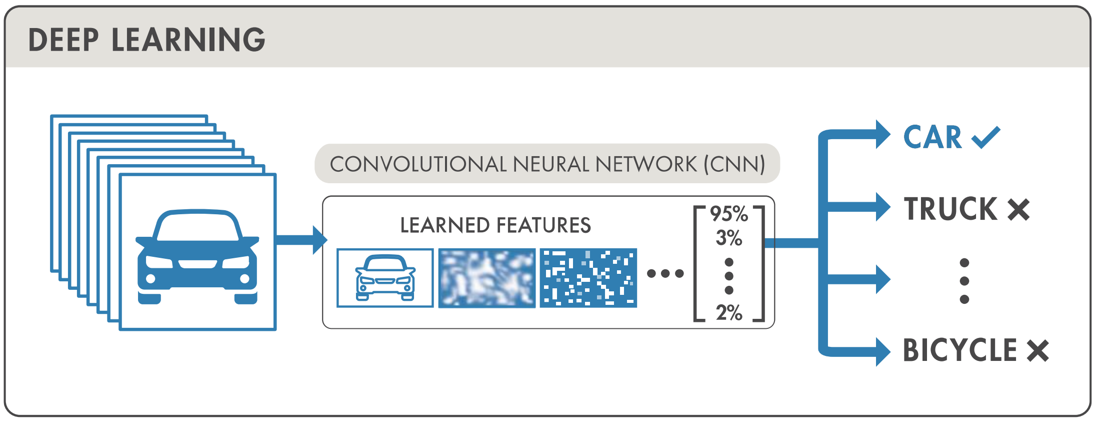


Figure 2: Deep learning technique for object recognition [5]

2.2 Simulation in robotics

Simulation in robotics involves the creation of computerized models that replicate real-world scenarios, enabling the user to analyze, validate, and optimize various aspects of robot behavior, control, and performance. By using simulators, complex robotic systems, sensor data, algorithms, and control strategies can be investigated without the need for physical prototypes or real-world experimentation. By accurately simulating the dynamics, kinematics, and sensory feedback of robots, simulators aid in understanding robot capabilities, enhancing algorithm design, testing navigation strategies, and validating safety protocols. Moreover, they facilitate the development of robot control algorithms, sensor integration, and human-robot interaction techniques.
There exist a variety of software for robotic simulators. The most common robotic simulators are Gazebo, RoboDK, Webots, CoppeliaSim and OpenRave [6].

2.3 Applications of robotic systems with computer vision

One paper [7] presents the development of a vision-based robot system aimed at playing the game Connect Four against a human player which is shown in Figure 3. The system utilizes a lightweight collaborative robot (Universal Robot UR3e) equipped with a visual sensor (a low-cost USB camera) mounted on its arm. The robot is also equipped with an adhesive gripper for grasping objects. The main goal is to create a portable and intelligent game-playing robot that can be used for entertainment, education, and potentially rehabilitation purposes. An algorithm known from game theory is used to enable rational decision-making by the robot. The proposed system involves image processing techniques for detecting the game board's status, recognizing and distinguishing disks, and detecting opponent moves. The paper discusses the hardware setup, the software architecture based on the Robot Operating System (ROS), the implementation of the Connect Four game, and the integration of different components. The robot's movements, disk detection, path planning, grasping, and dropping are explained.

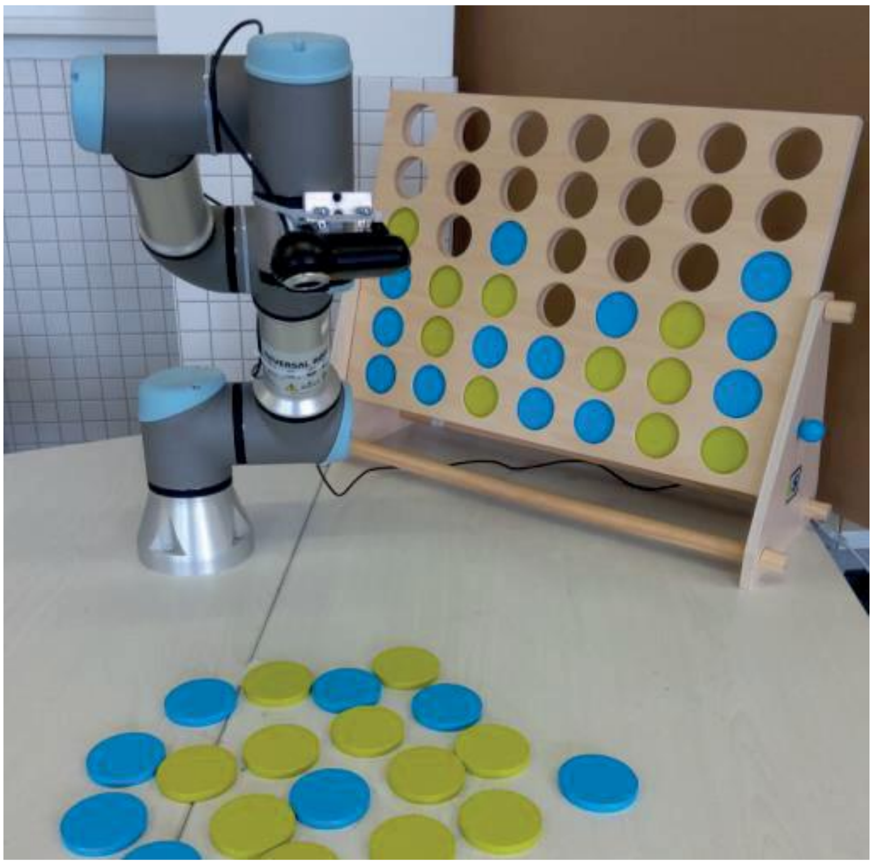


Figure 3: robot system for entertainment puposes [7]

Another application is developed by MathWorks to showcase the capabilities of MATLAB and Simulink. The application involves an Intelligent Bin Picking system that utilizes both perception and motion planning algorithms [8]. It starts by detecting the positions and orientations of objects within a bin through computer vision using the Deep Learning capabilities of MATLAB's Computer Vision Toolbox. The calculated object poses are then used as input for a motion planning algorithm, which generates a trajectory path for a Universal Robots UR5e collaborative robot to pick up the objects from the bin. The Robotics System Toolbox Support Package for Universal Robots UR Series Manipulators is employed to establish ROS communication with the UR5e robot, allowing the generated trajectory to be sent to the robot for picking up and placing the objects at a designated destination.

The object detection is done by a pre-trained YOLOv4 network. The YOLOv4 network can be used in MATLAB by installing the Computer Vision Toolbox Model for YOLO v4 Object Detection support package. The version "tiny-yolov4-coco" of the YOLOv4 is used as the detector for the object detection in this work. It has fewer network layers than the full YOLOv4 network, so it's possible to train the network and run the application with less computing power.

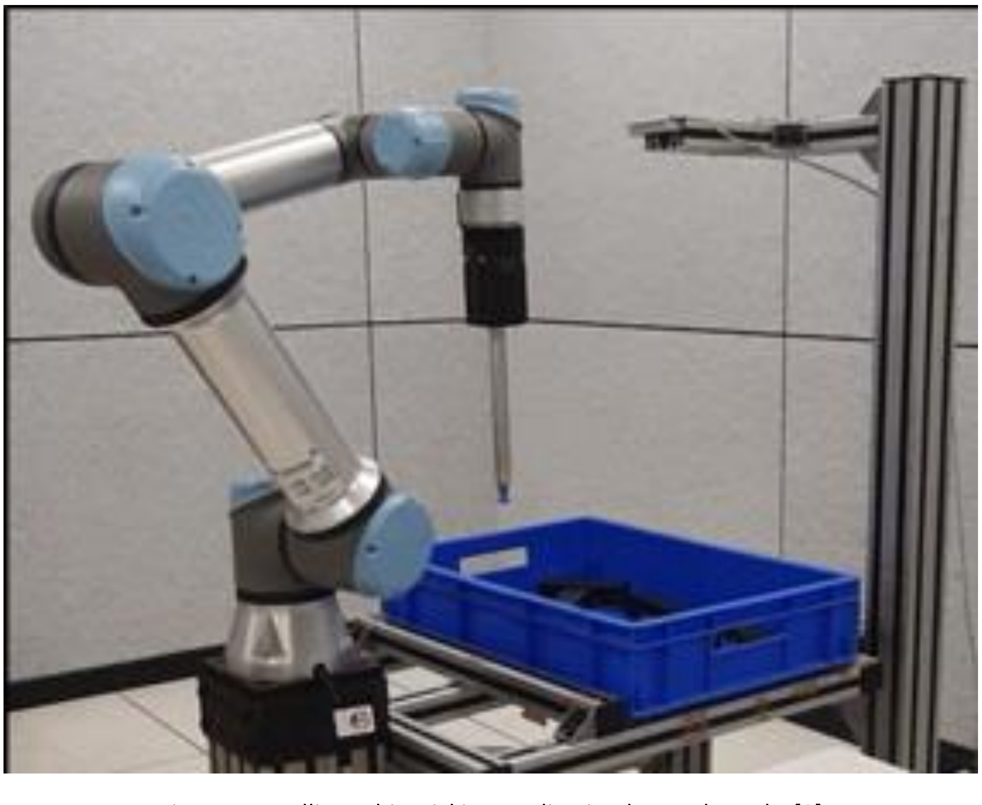


Figure 4: Intelligent bin-picking application by MathWorks [8]

The calculation of the object poses is done with a PCA-based algorithm. PCA stands for principal component analysis, and it is a statistical technique used for dimensionality reduction and data compression while retaining the most significant information within a dataset. The PCA transforms the original dataset by projecting it onto a new coordinate system defined by its principal components. These components are linear combinations of the original variables and are chosen to maximize the variance captured by each component. The first principal component is the direction in which the data varies the most. Subsequent principal components are orthogonal to the previous ones and capture the remaining variance in decreasing order. [9] Figure 5 shows an example of the two-dimensional results of a PCA.

In terms of the algorithm for pose estimation, the point cloud data of an object is used as the dataset to analyze. Due to the three-dimensional expansion of the object and its cuboid shape, the PCA-based algorithm detects the three principal components as the x-, y- and z-axis of the object.

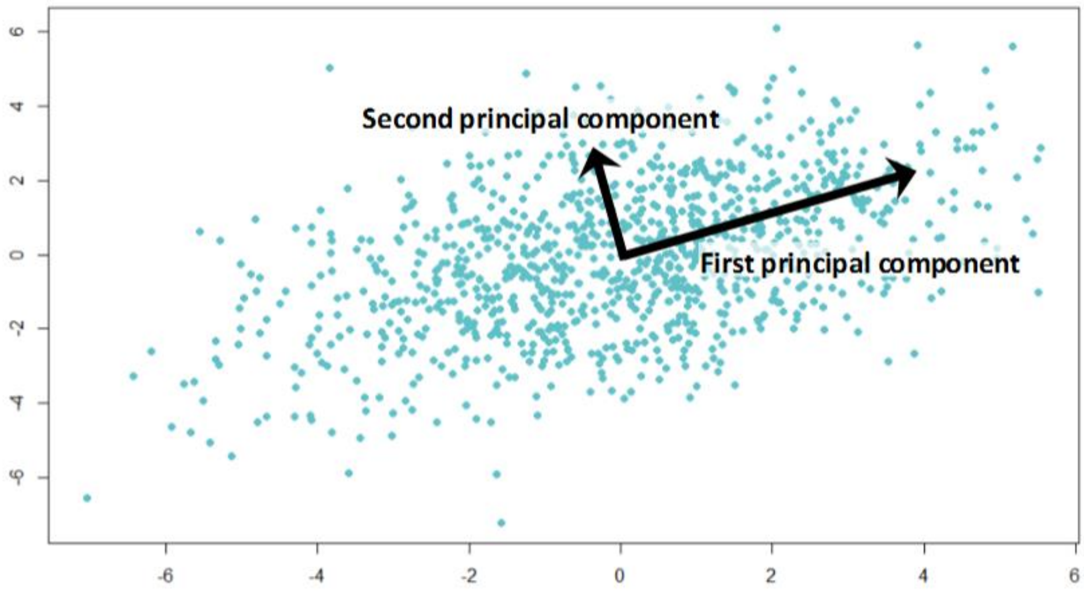


Figure 5: Principal Component Analysis (PCA) of scatter plot [10]

3 Development of AI-based pick and place system

In this main chapter, the development of the pick and place system with computer vision is detailed by the system architecture with the integration of needed hardware and software to create a simulation model and the real-world setup. The solution of the computer vision task of object detection and pose estimation for the AI-based motion control of the robot is based on the intelligent bin-picking application by MathWorks [8] analyzed in chapter 2.3.

3.1 System architecture

In this section, the components selected for the implementation of the task are described in detail. Furthermore, the communication between the components within the experimental setup are explained.

3.1.1 Overview of components

There are three main tasks of a pick and place application: the perception of the object to grasp, the planning of the robot's and the gripper's motion for reaching the object, and the motion's control to finally reach and grasp the object. In order to be able to fulfill these tasks the following components as the main hardware as well as the following software are available in the laboratory and are selected for the application:

<u>Universal Robot UR3e</u>

Figure 6: Robot model UR3e Universal Robots [11]

The Universal Robot UR3e is a collaborative robotic arm manufactured by Universal Robots. With its 6-axis articulated arm, it offers a payload capacity of up to 3 kilograms and a maximum reach of 500 millimeters. [11]
To interact with the environment, the UR3e can be equipped with various end effectors, including grippers, suction cups, or custom-made tools. The UR3e has an eight-pinned connector that provides power and control signals for these end effectors. In this work, the DH-Robotics RGI14 gripper is chosen as the end effector, which adds grasping capabilities to the robot. Universal Robots provides a programming interface called teach pendant, which allows users to program the UR3e robot for simple tasks directly and without additional hardware or software. The robot can be also programmed through offline programming software like the robotic simulator software RoboDK to process the moving instructions.

RGI14 gripper by DHrobotics

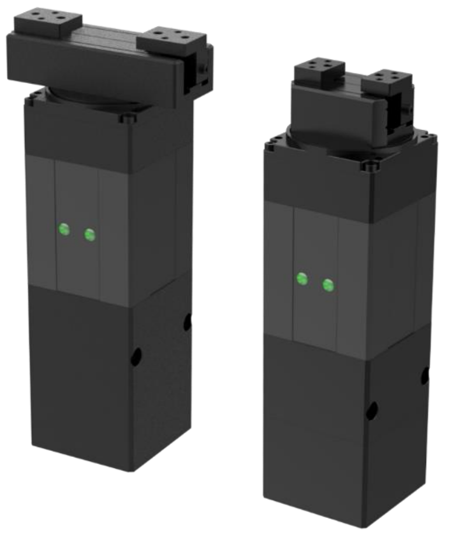


Figure 7: Gripper model RGI14 by DH-Robotics [12]

The RGI14 gripper is a robotic gripper designed and manufactured by DHrobotics (Figure 7). It offers a parallel jaw design, where the two jaws move in parallel to each other, and infinite rotation of the fingers because of a rotary unit. The gripper jaws can be adjusted to accommodate objects of different widths, making it versatile for handling a wide range of workpieces. For the rotation and gripping tasks, there are 2 sets of servo motor systems and drive controller inside the housing. The communication interface of the gripper provides a RS485 and digital Input-Output communication. In addition, the gripper has the following characteristics listed in Table 1 [13]:

Table 1: Characteristics of gripper RGI14 by DH-Robotics

characteristic	value
weight	1 kg
nominal voltage	24V DC
stroke	14mm
angle of rotation	infinite
repetion accuracy	0,02° / 0,02mm

Realsense L515 camera

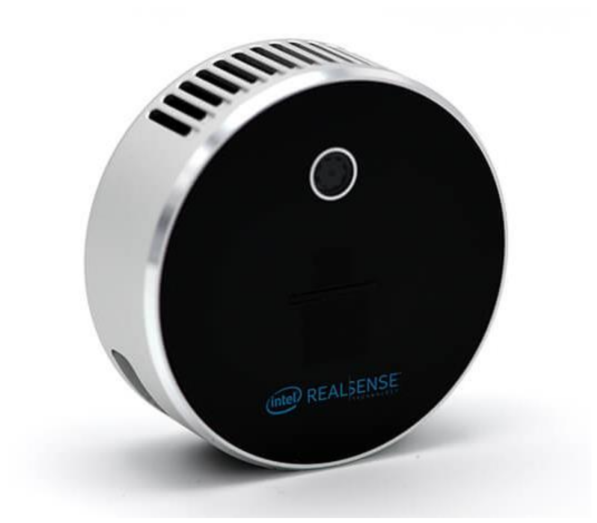


Figure 8: Camera model RealSense L515 by Intel [14]

The RealSense L515 is a LiDAR depth camera developed by Intel's RealSense division. It is designed for applications that require depth perception and 3D vision. The L515 camera utilizes Time-of-Flight (ToF) technology to capture depth information, making it suitable for a wide range of robotics, augmented reality, virtual reality, and computer vision applications. The ToF technology measures the time it takes for light to travel from the camera to the target object and back. The RealSense L515 emits an infrared laser onto the complete field of view, and the built-in sensor calculates the distance to the objects based on the processed data from the reflected beam captured by the photodiode. The aggregation of the depth points will generate a point cloud depth data representing the full scene. [15]

The RealSense L515 camera offers the capability of producing depth images with a depth resolution of up to 1024x768 pixels at 30 frames per second. The L515 camera also captures RGB color data. By combining depth and color information, the camera generates synchronized depth and color frames, allowing for the creation of full-color 3D point clouds. This fusion of depth and RGB data enhances the camera's perception capabilities, enabling the recognition and tracking of objects in the scene.

Intel provides a software development kit (SDK) for the RealSense cameras. The SDK allows developers to access and utilize the camera's depth and color streams, apply real-time filters, perform 3D mapping, and implement computer vision algorithms. The camera is compatible with programming languages such as C++, Python, MATLAB, and ROS (Robot Operating System), so that the RealSense L515 can be integrated into different applications.

<u>RoboDK (v5.6.0 for Windows)</u>

RoboDK as a robotic simulator software serves as a tool for creating simulation models of robotic systems. Its simulation and programming capabilities empower users to design, validate, and optimize robotic operations virtually, minimizing errors and maximizing efficiency when deploying robots in real-world applications. By emulating physical interactions and movements, RoboDK enables users to identify and address potential issues, ensuring error-free real-world execution.
RoboDK offers the compatibility with various industrial robot brands and models. With its in-built robot library, the robot 3D model of different manufacturers like ABB, KUKA and Universal Robots can be loaded, including the model of the UR3e used in this work. Figure 9 displays an exemplary station of the RoboDK library.
RoboDK's graphical interface allows users to design and manipulate robotic cells using a drag-and-drop approach. The robotic cell is created and saved as a station. The loaded 3D objects, in RoboDK called items, are positioned in respect to reference frames which can be added in any pose to the station while the base reference frame is always of the imported robot. A tool model can be attached to the robot’s tool flange and be customized by the user, e.g., by changing the tool center point (TCP). The coordinate system of the TCP influences the movement of the robot. By moving to specific target reference frames defined by the user, the robot’s movement ends with the TCP coordinate system at the target reference frame.

A robot program is designed with RoboDK by adding instructions. For movements, mainly the instructions *MoveL* for linear movement, *MoveJ* for joint movement and *MoveC* for circular movement are used. While computing the movements, RoboDK's collision detection feature alerts users to potential clashes between robot components, tools, and the environment, facilitating the identification and resolution of interference issues.

The computer vision utility allows the user to simulate a 2D-camera from the perspective of the reference frame set as the parent. Different parameters like the sensor size, the pixel size and the field of view can be specified in order to obtain the view of a real camera with these characteristics in the virtual environment. The camera can be kept stationary or attached to the robot, depending on the related reference frame in the station tree.

RoboDK also features the possibility to directly communicate with a real robot via a TCP/IP connection. Program instructions are sent to the connected robot already translated into the needed language for the robot controller by the selected post-processor in RoboDK.
Robot programs can also be created completely outside of RoboDK by using the RoboDK API for different programming languages like Python, C++, and MATLAB.

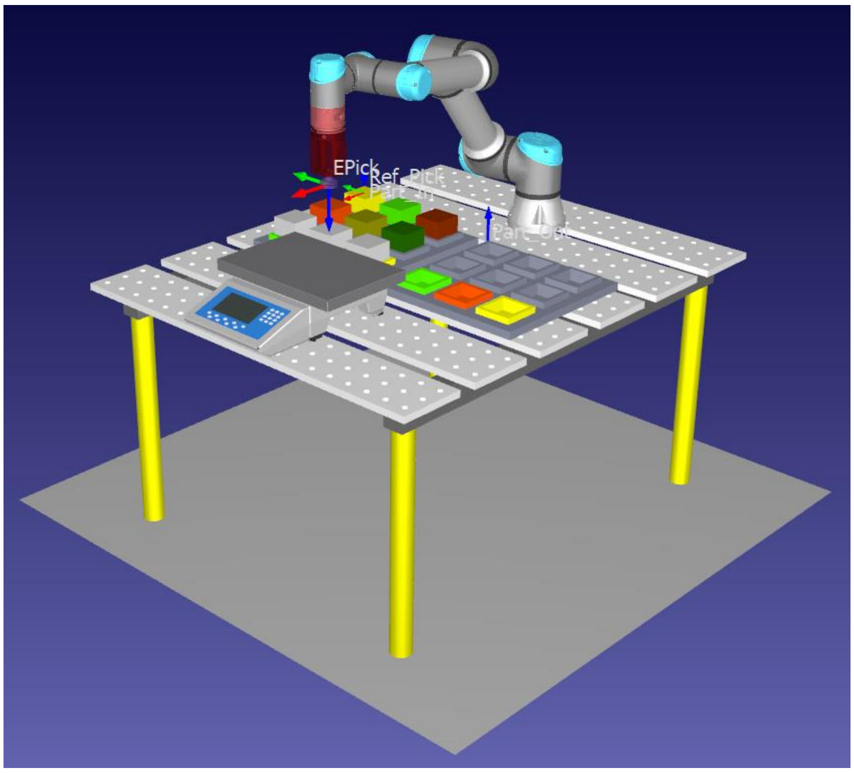


Figure 9: Example station of RoboDK's robot library [16]

MATLAB (R2023a)

MATLAB is a high-level programming environment widely used in various fields, including engineering, mathematics, and scientific research. It provides a platform for numerical computation, data analysis, visualization, and algorithm development. With its extensive library of built-in functions and toolboxes, MATLAB enables users to perform various mathematical operations, manipulate data, and create advanced visualizations. Moreover, MATLAB supports integration with various hardware devices, enabling real-time data acquisition and control in applications ranging from signal processing to robotics. In case of this work, the RealSense MATLAB wrapper and the RoboDK MATLAB API are used in order to integrate the communication with RoboDK and the RealSense L515 camera into the MATLAB environment.

The RealSense MATLAB wrapper is a package that wraps some of the core functions of the realsense2.dll with which the RealSense depth cameras can be controlled [17]. The library offers the functionality to enable a communication with the camera device by the construct the class object *realsense*. It's possible o start a color and depth stream with the camera object's method *pipeline().start* and extracting a single frame of the streamed scene with the method *wait_for_frames()*. The specific color and depth data can be obtained from the extracted frame with the methods *get_color_frame()* and *get_depth_frame()*. With the *pointcloud* class of the wrapper, the depth data can be converted into points with three-dimensional information with the method *calculate(depth)*.

The RoboDK API for MATLAB is provided as a group of m files [18]. The *Robolink* class is used to setting up the interface between MATLAB and RoboDK. With the Robolink class, any object of the loaded station tree can be handled in the MATLAB environment represented as a *RobolinkItem* object. Any operation done with the graphical interface of RoboDK can be also performed by the code in MATLAB, like adding 3D objects, targets or reference frames or change the position of these with the methods of the *RobolinkItem* object. Furthermore, programs can be created or run as well as a simulated camera can be handled.

In addition, the following MATLAB toolboxes are used to perform the needed data handling tasks and use specific functions:

- Robotics Systems Toolbox
- Computer Vision Toolbox
- Deep Learning Toolbox
- Image Processing Toolbox
- Lidar Toolbox
- Optimization Toolbox
- Statistics and Machine Learning Toolbox
- Computer Vision Toolbox Model for YOLO v4 Object Detection support package

3.1.2 Integration of hardware

In this section, the interconnections among the selected components of the robotic system are outlined. Figure 10 shows the overview of how the components are connected to each other. In the following, each connection is described in detail.

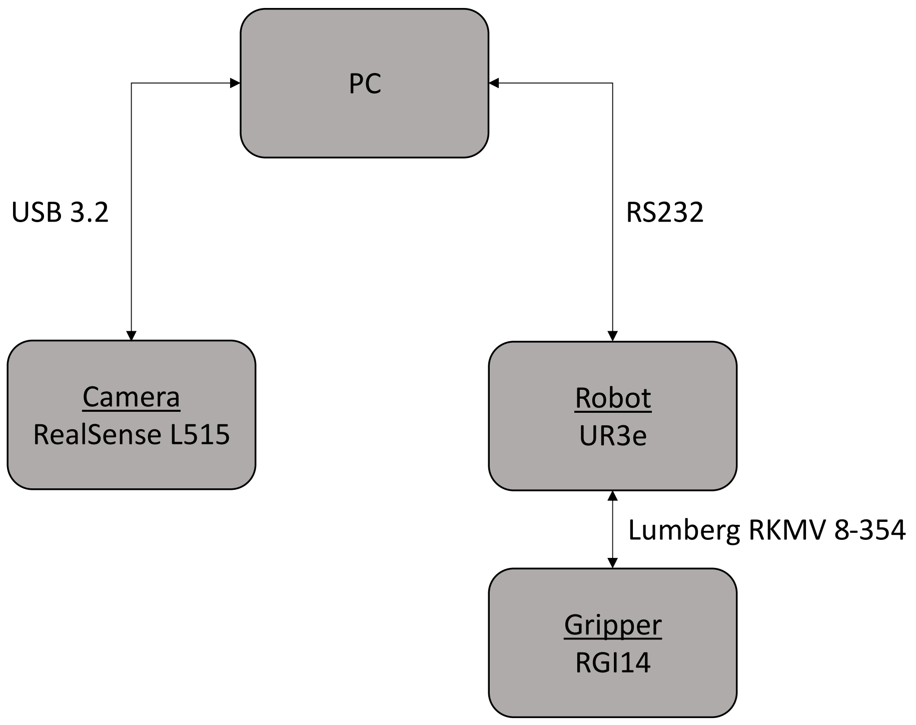
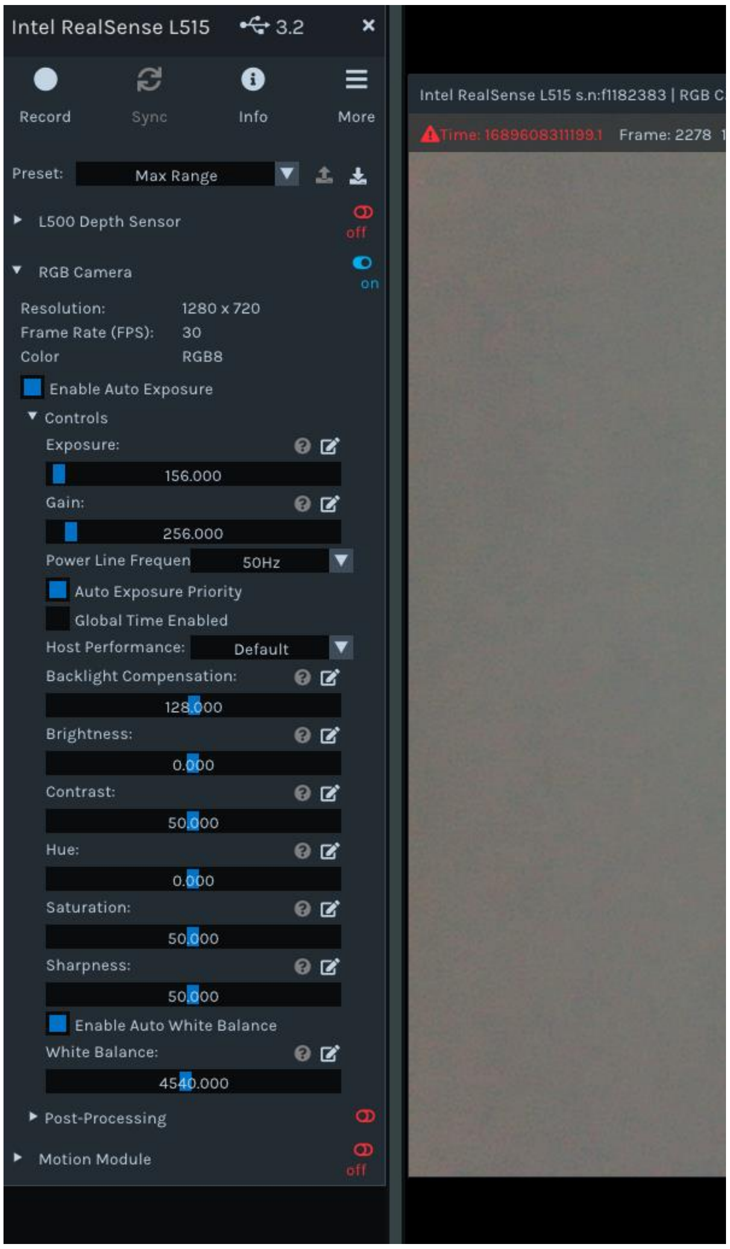


Figure 10: Interconnections of system's components

PC to camera RealSense L515

The RealSense L515 camera is connected via USB 3.2 cable to the PC in order to get the commands as well as to send the image data to the PC. The camera can be connected also with a previous version of the USB A type like USB 2.0 but it will decrease the possible image solution and frame rate. With the Software *Intel RealSense Viewer* [19] the connection can be checked and several parameters for the color and depth data can be changed in order to get the best settings for the individual application.



Intel RealSense L515 3.2

Preset: Max Range

L500 Depth Sensor off

RGB Camera on

Resolution: 1280 x 720
Frame Rate (FPS): 30
Color RGB8

Enable Auto Exposure

Controls

Exposure: 156.000
Gain: 256.000
Power Line Frequen 50Hz
Auto Exposure Priority
Global Time Enabled
Host Performance: Default
Backlight Compensation: 128.000
Brightness: 0.000
Contrast: 50.000
Hue: 0.000
Saturation: 50.000
Sharpness: 50.000
Enable Auto White Balance
White Balance: 4540.000

Post-Processing

Motion Module off

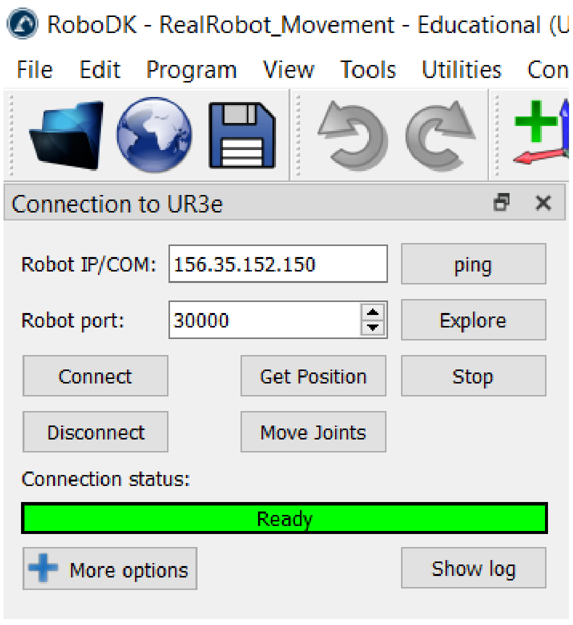
Intel RealSense L515 s.n:f1182383 | RGB C

Time: 1689608311199.1 Frame: 2278

Figure 11: Settings in Intel RealSense Viewer

<u>PC to robot UR3e</u>

The communication between the PC and the robot UR3e of Universal Robots is established via RS232 within the same network [20]. With the Software *RoboDK*, the user connects to the robot defining its IP address and port, see Figure 12. In case of this UR3e application the IP address is 156.35.152.150 with port number 30000. On the teach pendant of the UR3e, the control mode must be set to “Remote”. *RoboDK* translates the movement instructions into the necessary commands for the robot controller of the UR3e with the selected post processor for “Universal Robots”.



Robot IP/COM: 156.35.152.150
Robot port: 30000
Connection status: Ready

Figure 12: Connection between PC and UR3e

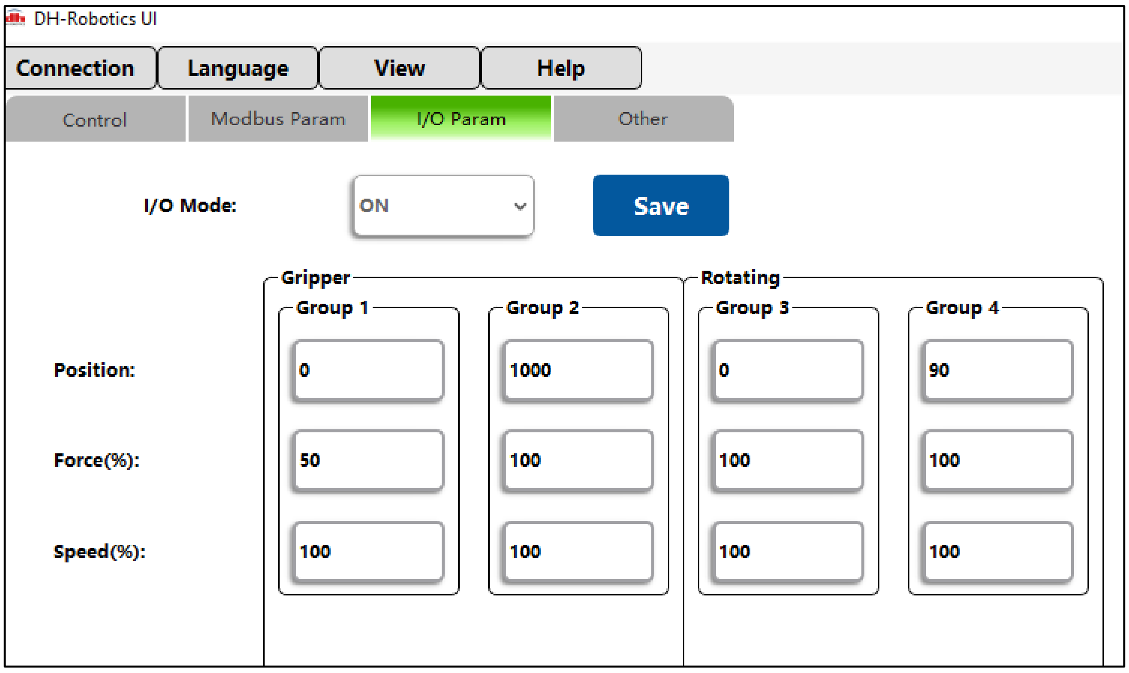
<u>Robot UR3e to gripper RGI14</u>

The eight power and signal lines of the gripper RGI14 are connected to the eight-pinned connector of the robot UR3e via a Lumberg RKMV 8-354 cable, but only the power lines and the signal lines for the digital input and output are used in this application. Table 2 shows the colors of the eight wire pairings between the used commercial Lumberg RKMV 8-354 cable connected to the robot UR3e and wires of the gripper RGI14. In order to provide the gripper with the necessary power supply, the output voltage of the tool connector of the robot UR3e must be set to 24V in the teach pendant in the control mode “Local”: tab *Installation* → *General* → *Tool I/O* → *Voltage of tool output:* 24V.

The gripper RGI14 is programmed to run in digital Input-Output mode for this application with the provided software by DH-Robotics [21] according to the settings in Figure 13. The group 1 and group 2 parameters are the values for the digital input *Input 1* set to 0 or 1 and the group 3 and group 4 parameters are the values for the digital input *Input 2* set to 0 or 1. The positions are set to 0 (fully closed) and 1000 (fully open) for the gripper input as well as 0 (initial position) and 90 (rotated by 90°) for the rotating input. All speeds and forces are set to 100% to use the full potential of the gripper, except of the closing force. It’s set to 50% in order to stop the close movement earlier when grasping an object so that it’s not damaged. Table 3 describes the different states depending on the digital input values of the gripper which depend on the digital output values of the robot.

Table 2: Wire pairings of connection between robot UR3e and gripper RGI14

No.	UR3e Lumberg RKMV 8-354		RGI14	
	Color	Description	Color	Description
1	Red	GND	Black	GND
2	Gray	POWER (24V)	Red	24V
3	Blue	Digital Output 0	White	Input 1
4	Pink	Digital Output 1	Brown	Input 2
5	Yellow	Digital Input 0	Yellow	Output 1
6	Green	Digital Input 1	Orange	Output 2
7	White	RS485+	Green	485_A
8	Brown	RS485-	Blue	485_B



I/O Mode: ON

	Gripper		Rotating	
	Group 1	Group 2	Group 3	Group 4
Position:	0	1000	0	90
Force(%):	50	100	100	100
Speed(%):	100	100	100	100

Figure 13: Settings of gripper RGI14

Table 3: States of the gripper RGI14

Input 1	Input 2	State Description
0	0	Gripper closed and in initial rotating position
0	1	Gripper closed and rotated by 90°
1	0	Gripper open and in initial rotating position
1	1	Gripper open and rotated by 90°

3.1.3 Integration of software

For the developed pick and place application, only the software of MATLAB 2023a and RoboDK v5.6.0 are used. The concept of the data flow within the integrated pick-and-place system has the following progression according to Figure 14: Starting with the MATLAB main program, the RealSense L515 camera gets the command to take a snapshot with which depth and RGB data are captured and relayed to the RealSense MATLAB wrapper. This information is then transmitted to the main MATLAB program for preprocessing, object detection and motion planning. The MATLAB program uses the RoboDK MATLAB API, which bridges to the RoboDK main program. Within the RoboDK simulation, movement instructions are validated, and the Universal Robot post processor generates robot-specific code for the robot UR3e. This code is passed to the UR3e, guiding its movement. Throughout this process, digital I/O signals enable communication with the DHrobotics RGI14 gripper to open, close and rotate the gripper.

The detailed functions of the pick and place algorithm in MATLAB and RoboDK are described in chapter 3.3 and 3.4.

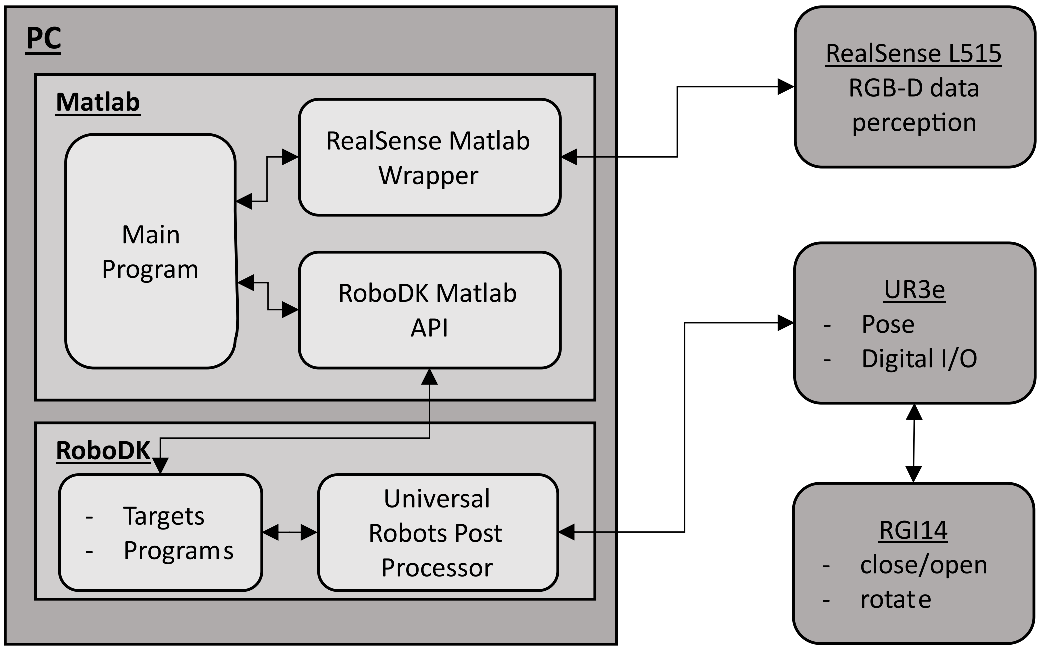


Figure 14: Concept of dataflow in system

3.1.4 Objects for pick and place application

Due to the opening width of the available gripper RGI14 of 14mm, commercial plastic bricks of the type shown in Figure 15 are used for the pick and place application. The following plastic bricks with the colors and dimensions shown in Table 4 are available in the laboratory. The different bricks allow to test the functionality of the AI-based control algorithm regarding the classification accuracy due to its different shapes and colors.

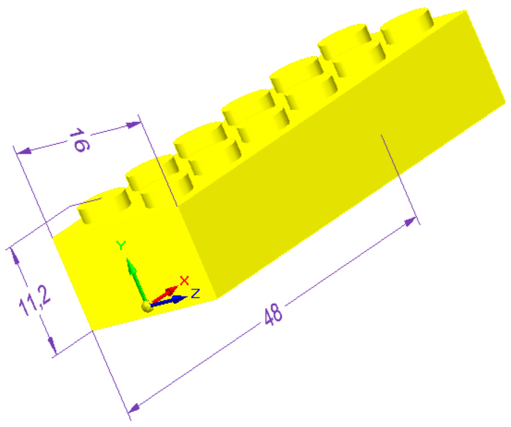


Figure 15: Model of yellow plastic brick

Table 4: dimensions of available plastic bricks

Brick type	Color	x [mm]	y [mm]	z [mm]
2x2	black	16	11,2	16
2x3	white	24	11,2	16
2x4	black	32	11,2	16
2x4	orange	32	11,2	16
2x6	yellow	48	11,2	16

3.2 Robotic simulation model development using RoboDK

As described in chapter 2.2, the development of a robotic simulation model before the setup of the final application has several advantages. In the following, the creation of a simulation model of the robot UR3e in within the pick and place application using the software *RoboDK* is detailed.

The simulation model of the pick and place application contains the following parts (Figure 16):

- Table (import of RoboDK's "open robot library")
- model of robot UR3e (import of RoboDK's "open robot library")
- CAD model of camera Intel RealSense L515 [22]
- CAD model of gripper DH-Robotics RGI14 [23]
- CAD models of designed flanges (described in chapter 3.5)
- CAD models of designed boxes (one for each brick color)
- CAD models of several plastic bricks [24]

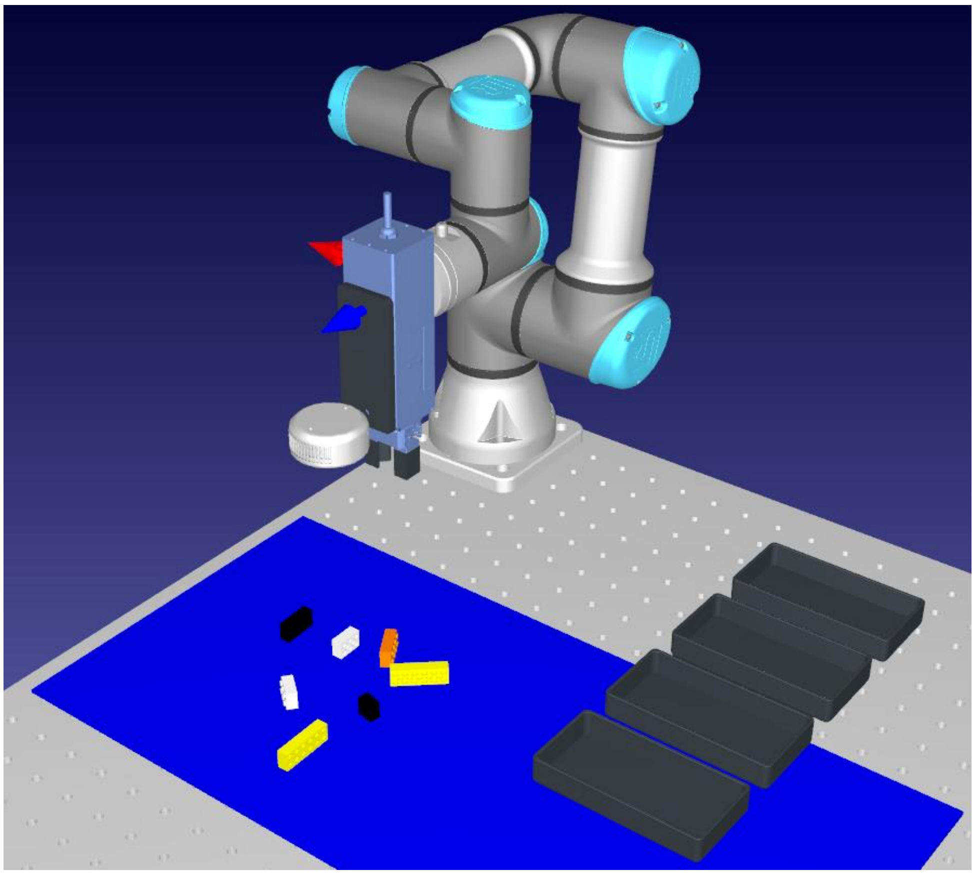
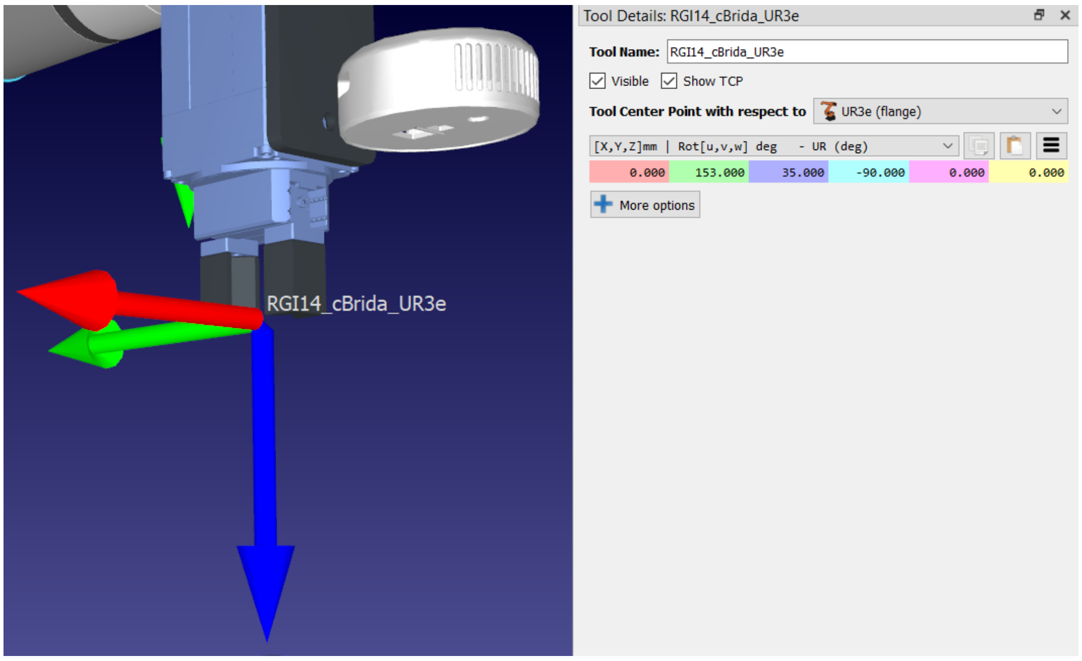


Figure 16: Simulation model in RoboDK

The model of the gripper is added twice to ensure the functionality of the real world the best in which the fingers can have the state of two different positions. In the second model, the fingers are rotated by 90°. If the gripper's functionality of rotating the fingers is executed, one model is set visible and the other one invisible and vice versa. The TCP is set in the middle of the fingers with the following values relative to the tool flange of the robot UR3e (Figure 17):
x = 0mm; y = 153mm; z = 35mm; u = -90°; v = 0°; w = 0°



Tool Details: RGI14_cBrida_UR3e

Tool Name: RGI14_cBrida_UR3e

☑ Visible ☑ Show TCP

Tool Center Point with respect to UR3e (flange)

[X,Y,Z]mm | Rot[u,v,w] deg - UR (deg)

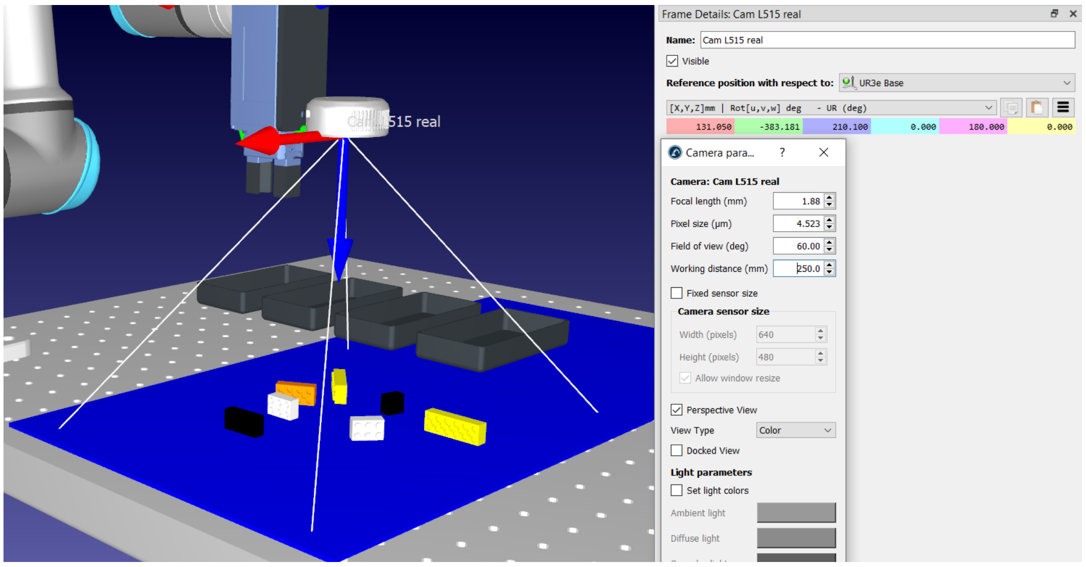
0.000	153.000	35.000	-90.000	0.000	0.000

More options

Figure 17: TCP of simulation model

The camera view can be simulated with RoboDK: tab *Connect* → *simulate 2D camera*. It has the following settings which are partly oriented at the real values of the camera RealSense L515:

- Focal length: 1.88mm
- Pixel size: 4.523µm
- Field of view: 60°
- Working distance: 250mm



Frame Details: Cam L515 real

Name: Cam L515 real

☑ Visible

Reference position with respect to: UR3e Base

[X,Y,Z]mm | Rot[u,v,w] deg - UR (deg)

131.050 | -383.181 | 210.100 | 0.000 | 180.000 | 0.000

Camera para...

Camera: Cam L515 real

Focal length (mm)	1.88
Pixel size (µm)	4.523
Field of view (deg)	60.00
Working distance (mm)	250.0

☐ Fixed sensor size

Camera sensor size

Width (pixels)	640
Height (pixels)	480

☑ Allow window resize

☑ Perspective View

View Type: Color

☐ Docked View

Light parameters

☐ Set light colors

Ambient light

Diffuse light

Figure 18: Simulated camera in RoboDK

The working distance could be also set higher. But with a lower value, the contrast for the depth image is higher what simplifies the post processing for the 3D point cloud information. Figure 19 shows the camera view of the simulated cameras, the left one for the color image and the right one for the depth image.

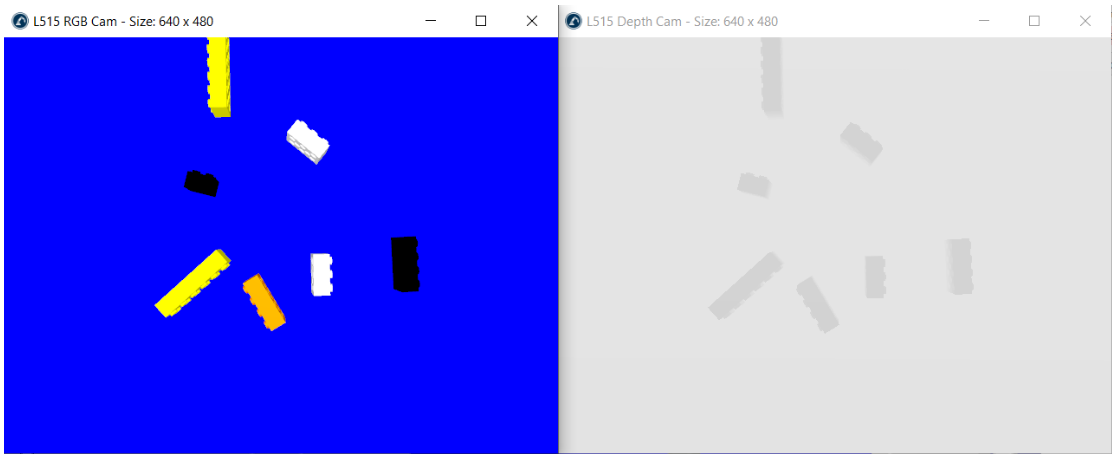

L515 RGB Cam - Size: 640 x 480
L515 Depth Cam - Size: 640 x 480

Figure 19: simulated camera view

3.3 Object detection and recognition

This chapter focuses on the object detection and recognition within the context of the developed pick-and-place system. It shows the implementation of a YOLOv4 object detection network in MATLAB for identifying the different plastic bricks with the use of computer vision. The network is trained with generated synthetic data. The chapter also introduces a PCA-based algorithm for accurately estimating the poses of these.

3.3.1 Training of YOLOv4 detector network

Training an object detection network requires extensive data, so that it will reliably detect and correctly classify the desired objects. With the help of the simulation model created in chapter 3.2, the image creation of any number of plastic bricks in various poses and colors can be automated and randomized. Then, these are used for the training of the YOLOv4 network. The script *BrickPositioning_DataGeneration* programmed in MATLAB uses the RoboDK MATLAB API for communicating with the simulation model and has the following input parameters which can be changed by the user:

- Path to the folder in which the created images should be saved
- Number of images
- Number of bricks per image
- Minimum distance between each brick

There are additional parameters that have to be changed if the script should be used for other objects than the plastic bricks used in this work:

- CAD models of objects (path to the data in STEP format)
- Color of loaded CAD models (if not specified, they're loaded to RoboDK in color grey)
- Outer dimensions of loaded CAD models

Figure 20 shows the process of the automated trainings data generation with the MATLAB script and the simulation model in RoboDK. The code initiates by connecting to the RoboDK station with the Robolink object of the RoboDK MATLAB API and creating the necessary Robolink items like camera handle and reference frames to use in the RoboDK station. After initializing all file paths and defining object properties, the robot is positioned for image capture, and within a loop for each image, another loop simulates random brick placement while ensuring spatial separation between all placed bricks. The minimum distance set to 20mm ensures that the training data reflects the real situation in which the gripper still should have enough space to grasp the objects. If the randomly generated combination of position and angle causes an interference of a new brick with the surrounding minimum distance area of already positioned bricks, the new brick won't be positioned but a new random position and angle is generated until the new brick fits in the area. The CAD model of the brick is loaded into RoboDK with the method *AddFile* of the MATLAB API. After reaching the set number of bricks, a snapshot for a color image is taken in the format of 640 x 480 pixels and the RoboDK environment is cleaned of added items. If the specified number of generated images isn't reached another iteration starts. Otherwise, the script stops.

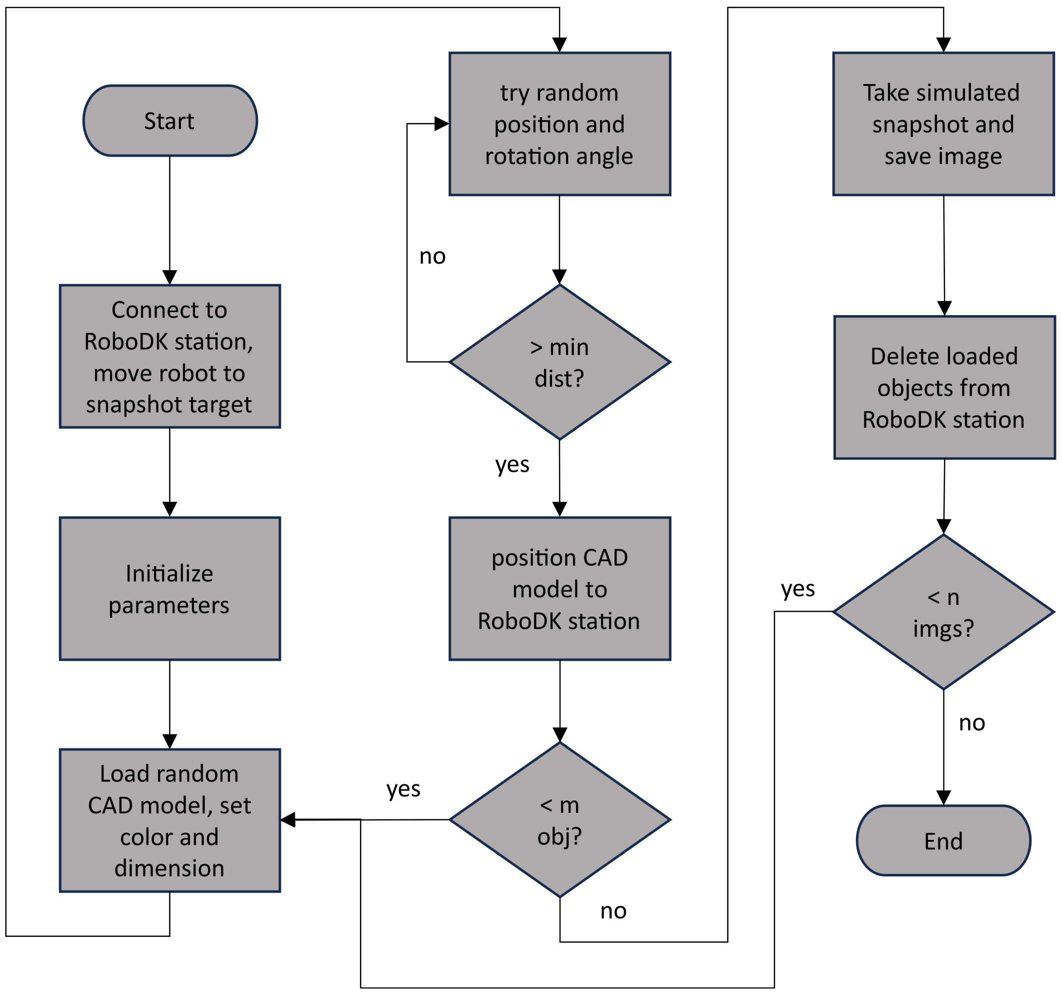

Start
Connect to RoboDK station, move robot to snapshot target
Initialize parameters
Load random CAD model, set color and dimension
try random position and rotation angle
no
> min dist?
yes
position CAD model to RoboDK station
yes
< m obj?
no
Take simulated snapshot and save image
Delete loaded objects from RoboDK station
yes
< n imgs?
no
End

Figure 20: Flowchart of synthetic trainings data generation

So that the YOLOv4 object detection network can recognize and classify the desired objects it has to be trained with image data which is labeled with a bounding box around the object and its category. With the help of the Image Labeler, an in-built app of MATLAB, the synthetic image data created with RoboDK is labeled manually with five different labels which define the color and the size of the brick:

- black_2x2
- white_2x3
- black_2x4
- orange_2x4
- yellow_2x6

Figure 21 shows the interface of the Image Labeler app with the generated data.

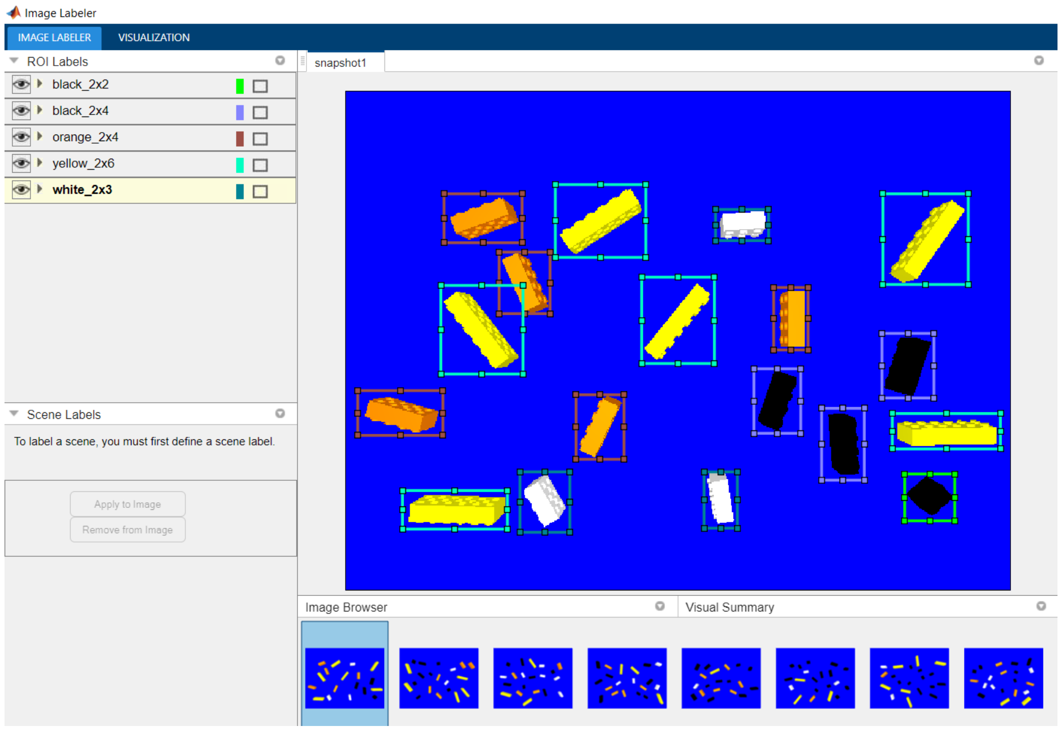


Figure 21: MATLAB Image Labeler

The output of the Image Labeler App is a ground truth struct containing the file paths of all labeled images, the definition of the different labels including name and color as well as all positions and sizes of the corresponding bounding boxes.

The object detection network is trained in 200 Epochs with the MATLAB script *TrainYoloV4ForBricks* with the ground truth struct of 25 labeled synthetic images which are used for MATLAB’s in-built data augmentation before training.

3.3.2 Integration of RealSense L515 in MATLAB

The camera Intel RealSense L515 connected via USB 3.2 with the PC can be completely controlled with MATLAB using the RealSense MATLAB wrapper released by Intel [17]. It enables the camera's functionalities to be utilized within the MATLAB environment.

This wrapper establishes a connection between the RealSense camera and MATLAB by creating a RealSense camera object.
With the camera object's methods, the camera settings for the pick-and-place application in this work are configured in the following way:

- Resolution: 640 x 480 pixels (like the input for the trained object detection network)
- Color sensor’s white balance: 2800 (lowest value to increase color intensity of image)
- Depth sensor’s Laser Power: 93 (lower than maximum value for scanning near objects)
- Depth sensor’s minimum distance: 100mm (ensures to detect near objects)

The programmed MATLAB script *takeSnapshot* using the functions of the RealSense MATLAB wrapper translates the data streams provided by the camera, including RGB images and depth maps, into MATLAB matrices. It’s important to use the align method of the RealSense camera object in order to map the depth data to the color data in the same size. Otherwise, there arises a misalignment in the further processing and the detected objects are estimated at the wrong position.
For the use in the pick-and-place application the point cloud data is needed. With the camera object’s methods and MATLAB’s functions of the Computer Vision toolbox, the depth maps are converted into dynamic point clouds. In this way, the depth values are translated into XYZ coordinates of each point.
The output of the function *takeSnapshot* are the 640x 480x3 matrix of the color image and the converted point cloud object of the current scene in the RealSense L515’s field of view when calling the function.

3.3.3 Object detection and pose estimation algorithm

The algorithm for detecting the bricks and estimating their poses is adopted from the PCA-based algorithm presented by MathWorks [25]. At first, the necessary parameters defined by the user in the MATLAB script *InitializeParameters* are initialized. The most important are the position and rotation of the camera coordinate system in respect to the robot’s coordinate system as well as the used pre-trained network, in the case of this work it’s the tiny YOLOv4 network trained with the synthetic images of bricks described in chapter 3.3.1.

Then, the color image and depth data of the scene with the bricks is taken, either with the simulated 2D camera in the simulation model described in chapter 0, or with the RealSense L515 camera as described in chapter 3.3.2. With the methods of the RealSense MATLAB wrapper, the taken depth data can be converted directly into a point cloud object which stores the information of the x, y and z dimension of each point. In case of the depth image of the simulated 2D camera, the image has to be converted into a point cloud object in the MATLAB environment. Each pixel of RoboDK's depth image (see Figure 19) has a grey value between 0 and 255 depending on the distance of the pixel. The grey value of 0 (or black) is equivalent to the distance 0 and the grey value of 255 (or white) is the working distance in the settings of the simulated 2D camera. So, the distance *z* to the camera of one pixel in the depth image is:

$$z = \frac{pixel's\ value}{255} \cdot working\ distance$$

The x and the y value of the point cloud object is calculated with the intrinsic values of the camera: the focal length, the principal point, and the pixel size.

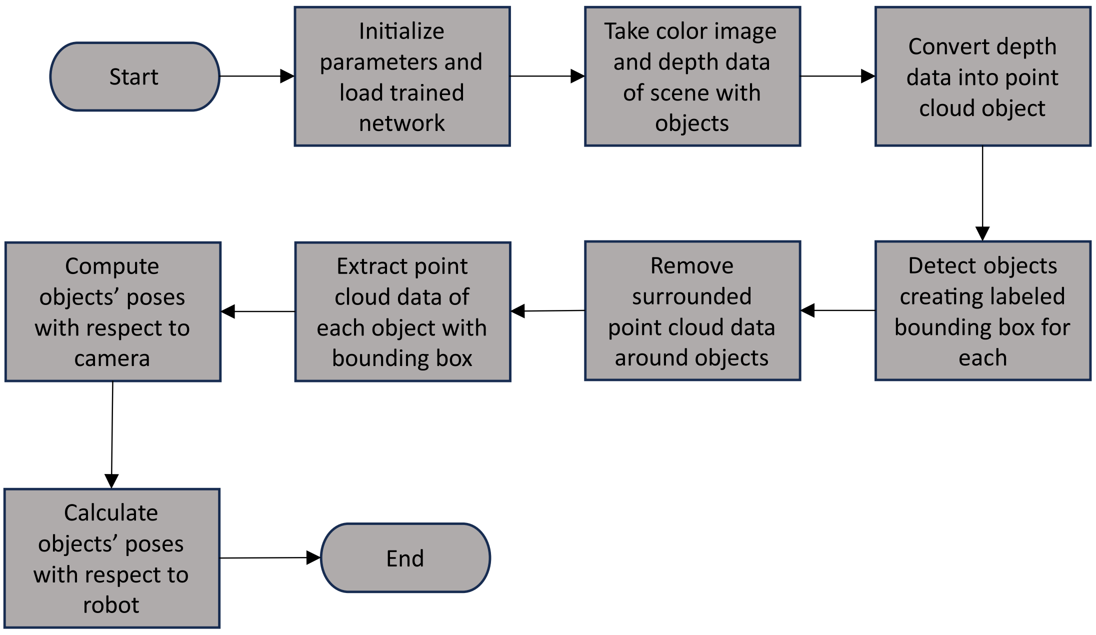

Start
Initialize parameters and load trained network
Take color image and depth data of scene with objects
Convert depth data into point cloud object
Detect objects creating labeled bounding box for each
Remove surrounded point cloud data around objects
Extract point cloud data of each object with bounding box
Compute objects' poses with respect to camera
Calculate objects' poses with respect to robot
End

Figure 22: Process of object detection and pose estimation

MATLAB's function *detectObjectsYoloNet* used with the pre-trained tiny YOLOv4 detector has as outputs the coordinates of the bounding boxes, the label with the classification with the highest confidence score and the value of the confidence score. Figure 23 shows a color image of the simulated 2D camera of RoboDK annotated with the output of the object detection network.

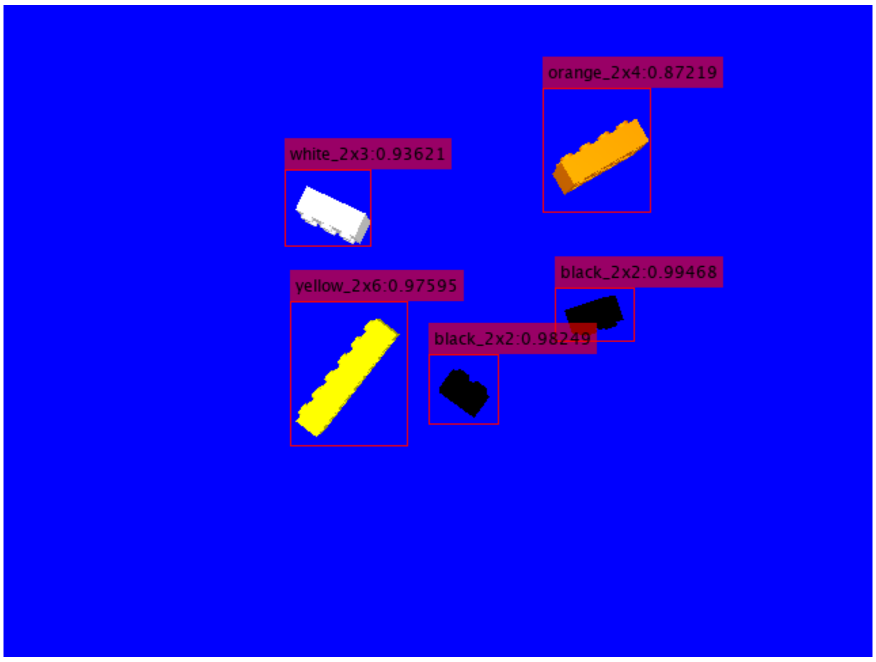


Figure 23: simulated camera image of bricks with object detection annotations

In the next step, the point cloud data of the bricks is separated from the point cloud data of the surrounding plane due to the difference of distance by obtaining a plane mask with MATLAB's function *pcfitplane* [26]. In this work, the threshold for the distance difference is set to 5mm which has enough tolerance to the bricks' height of 16mm and showed the best results. Figure 24 shows an example of the separated point cloud data.

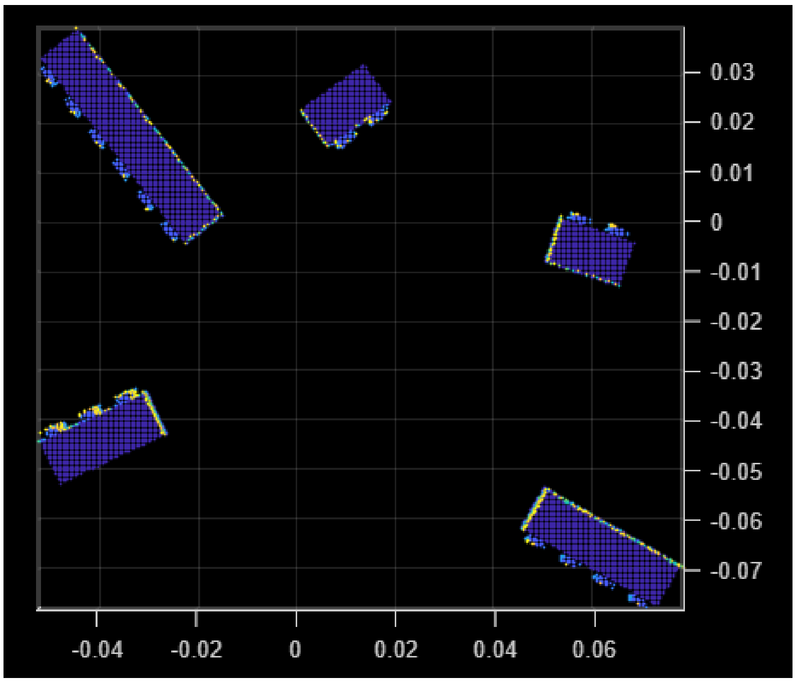


Figure 24: separated point cloud data of bricks

The separated point cloud data of the different bricks is compared with the position of the detected bounding boxes so that each brick is extracted in order to estimate its pose individually. With the PCA-based pose estimation of each brick's point cloud data, the coordinates of the center of gravity as well as the three principal axes of the object's points are computed in respect to the point of origin which is the position of the camera (Figure 25). The rotation angle of the object in respect to the camera's coordinate system can be calculated with the information of the main axis. With the rotation matrix from the camera coordinate system to the base coordinate system of the robot, the poses of the bricks are obtained in the robot coordinate system. This results in the final coordinate value of the object detection function. This then is used by the Pick and Place algorithm which will be explained in the following chapter.

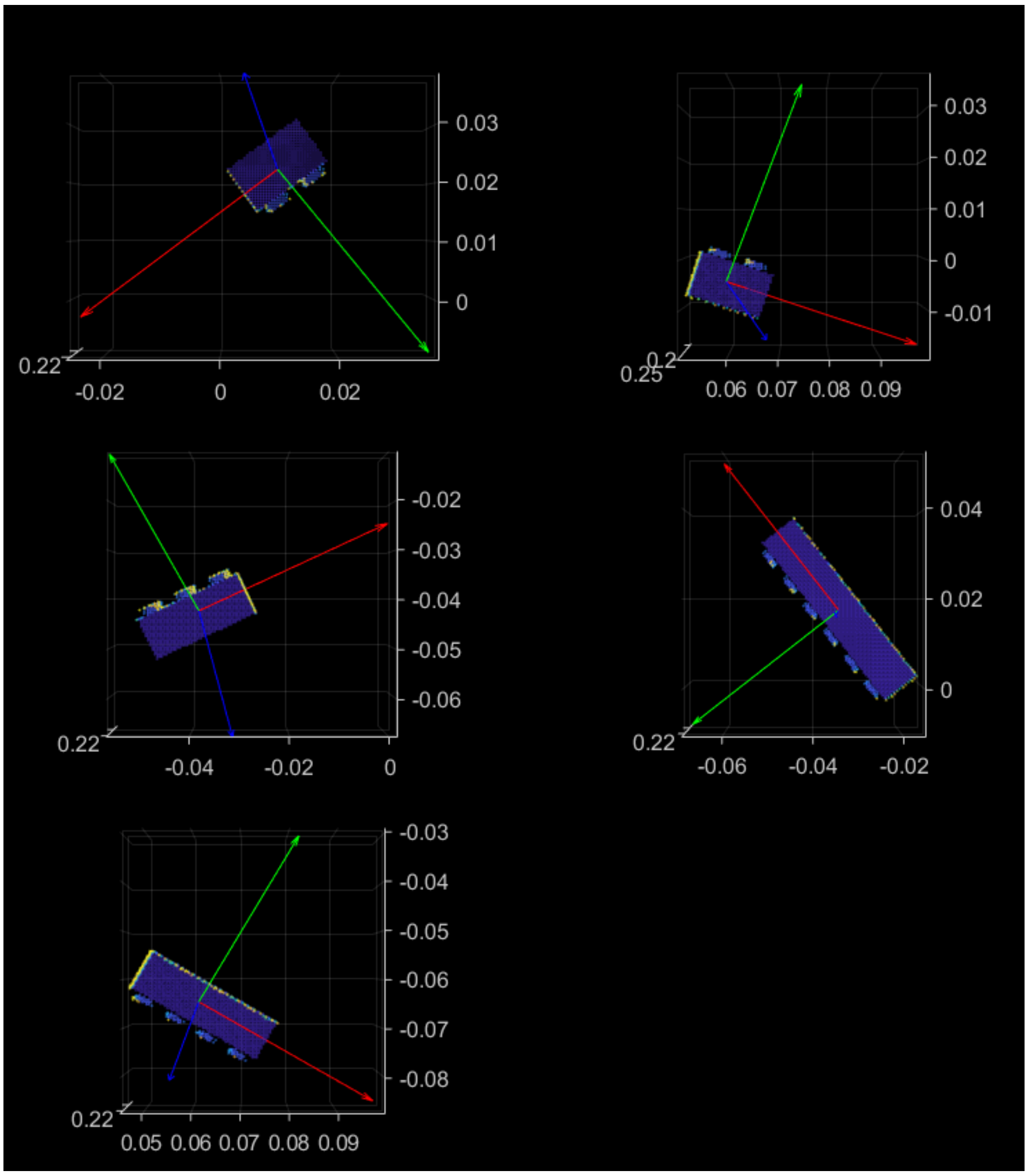


Figure 25: coordinates and orientation of point cloud data of bricks

3.4 Pick and Place Programming

In this chapter, the developed simulation model as well as the results of the object detection and pose estimation algorithm of the previous chapters are used for planning the movements of the robot.

3.4.1 Programs in RoboDK

The built-in robot controller and the post-processor for *Universal Robots* of RoboDK are used for controlling the movements of the simulation model and the real robot UR3e. Therefore, several instructions are combined in within different programs which can be called by MATLAB using the objects and methods of the RoboDK MATLAB API. After defining the different programs of the open RoboDK station as *Robolink items* in MATLAB, the item's method *RunProgram* causes that all instructions inside of the program are executed. It's taken advantage of this functionality by defining joint move instructions to the different targets which are defined in the station of the simulation model (Figure 26). All the defined programs and their effect after running them are listed in Table 5.

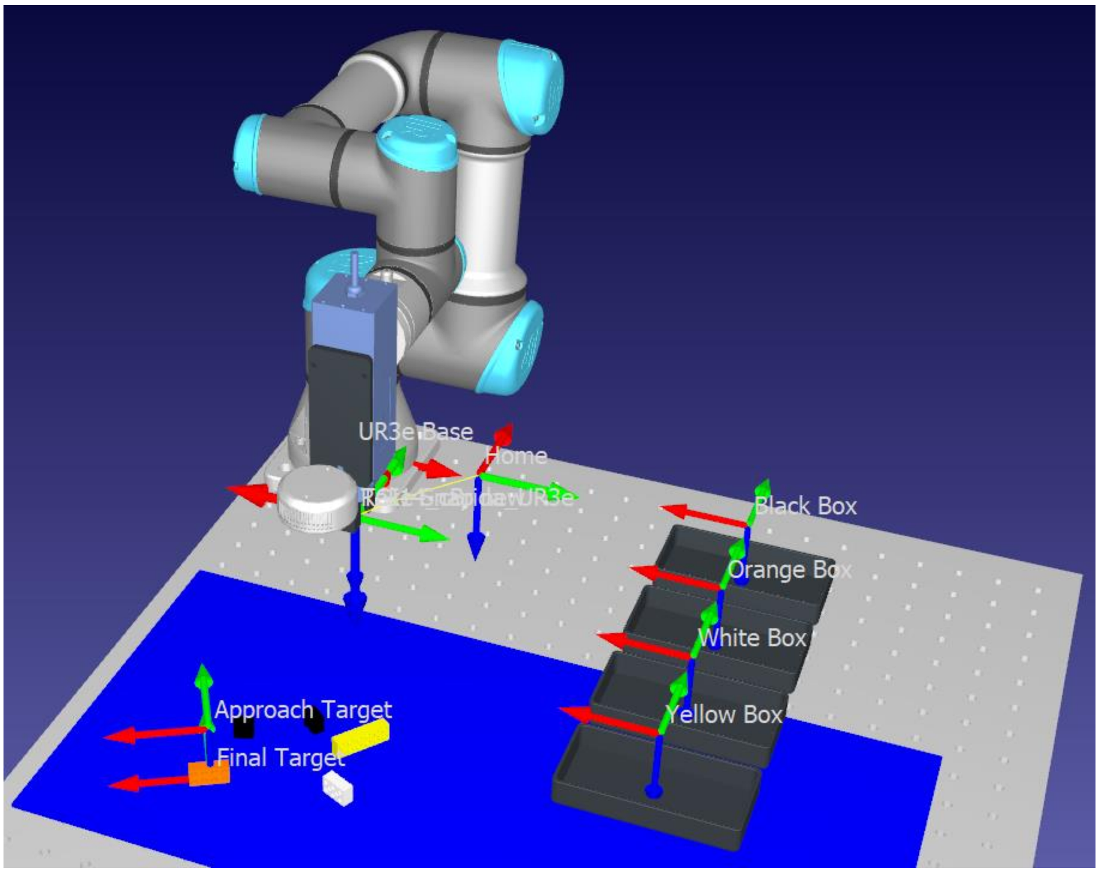


Figure 26: Targets of simulation model in RoboDK

Table 5: programs in RoboDK station of simulation model

program	instructions	effects
Move to Home	MoveJ (Home)	TCP moves to *Home* position with joint movement
Move to Snap-shot	MoveJ (TakeSnapshot)	TCP moves to position in which the camera is positioned for taking a snap-shot
Approach Tar-get	MoveJ (Approach Tar-get)	TCP moves to position 50mm above brick already correctly orientated with rotation
Move to Target	MoveJ (Final Target); Pause 500ms	TCP moves to *detected center of gravity of brick* with joint movement and waits 0,5s
TurnPos 1	Set IO_11 = 0; Pause 1s	Rotate the gripper's fingers to initial position and wait 1s
TurnPos 2	Set IO_11 = 1; Pause 1s	Rotate the gripper's fingers by 90° and wait 1s
Gripper closed	Set IO_10 = 0; Pause 1s	Close fingers of gripper until force is higher than threshold and wait 1s
Gripper open	Set IO_10 = 1; Pause 1s	Open fingers of gripper and wait 1s
Move to Yellow	MoveJ (Yellow Box)	TCP moves above box for yellow bricks with joint movement
Move to White	MoveJ (White Box)	TCP moves above box for white bricks with joint movement
Move to Orange	MoveJ (Orange Box)	TCP moves above box for orange bricks with joint movement
Move to Black	MoveJ (Black Box)	TCP moves above box for black bricks with joint movement

Since by the *MoveJ* instruction to a specific target only the final pose of the TCP is de-fined, the robot controller could compute any trajectory for the inverse kinematics which could cause a collision with the environment or twisting the cable of the RealSense L515.

Theoretically, the UR3e can perform infinite joint rotations because the default joint angle limits in RoboDK are set from -360° to 360°. Therefore, the movements of the robot are restricted with the joint angle ranges in Table 6. By the restriction, the robot moves only in within the needed workspace and doesn't perform joint rotations which could lead to a collision between the gripper or camera and the robot, or which could damage the cables between the devices.

Table 6: Joint angle restriction of UR3e in RoboDK

Joint axis of UR3e	Lower limit	Upper limit
Θ_1	-45°	180°
Θ_2	-150°	90°
Θ_3	0°	160°
Θ_4	-150°	90°
Θ_5	-150°	165°
Θ_6	-200°	-160°

3.4.2 Pick and Place Algorithm in MATLAB

As described before, the computation of the motion trajectory and the translation into the robot language is done by the software RoboDK. MATLAB is used for planning the sequence of the different movements and for defining the target positions for which the trajectory is computed, based on the results of the object detection. Figure 27 shows the flowchart of the pick and place planning with MATLAB. At first, the connection to the RoboDK station is established creating a *Robolink* object. All programs and targets of the station are initialized as *Robolink item* objects in MATLAB. If the position values of the bricks aren't already in the workspace, the positions of the bricks in the current scene are computed as described in chapter 3.3.3. The estimated center of gravity of a brick transformed into the base coordinate system of the robot is the reference for the cartesian values of the final target of the brick. Entering the loop with an iteration for each brick, the target *Final Target* is set with the values of the computed pose with the method *setPose* of the *RoboLink item* object. 50mm above this target (-z direction) an additional target *Approach Target* is set so that the tool can be positioned above the brick before grasping for avoiding collisions. The computed rotation of the brick around the z-axis leads to the decision if the gripper's fingers are rotated by 90° in order to simplify the robot's movement. Table 7 lists the angles for which the fingers stay in the initial position or are rotated by 90°. The output of the PCA-based pose estimation has values between -180° and 180° as an output. Then, the target brick is approached and the TCP aligned with the final target pose.

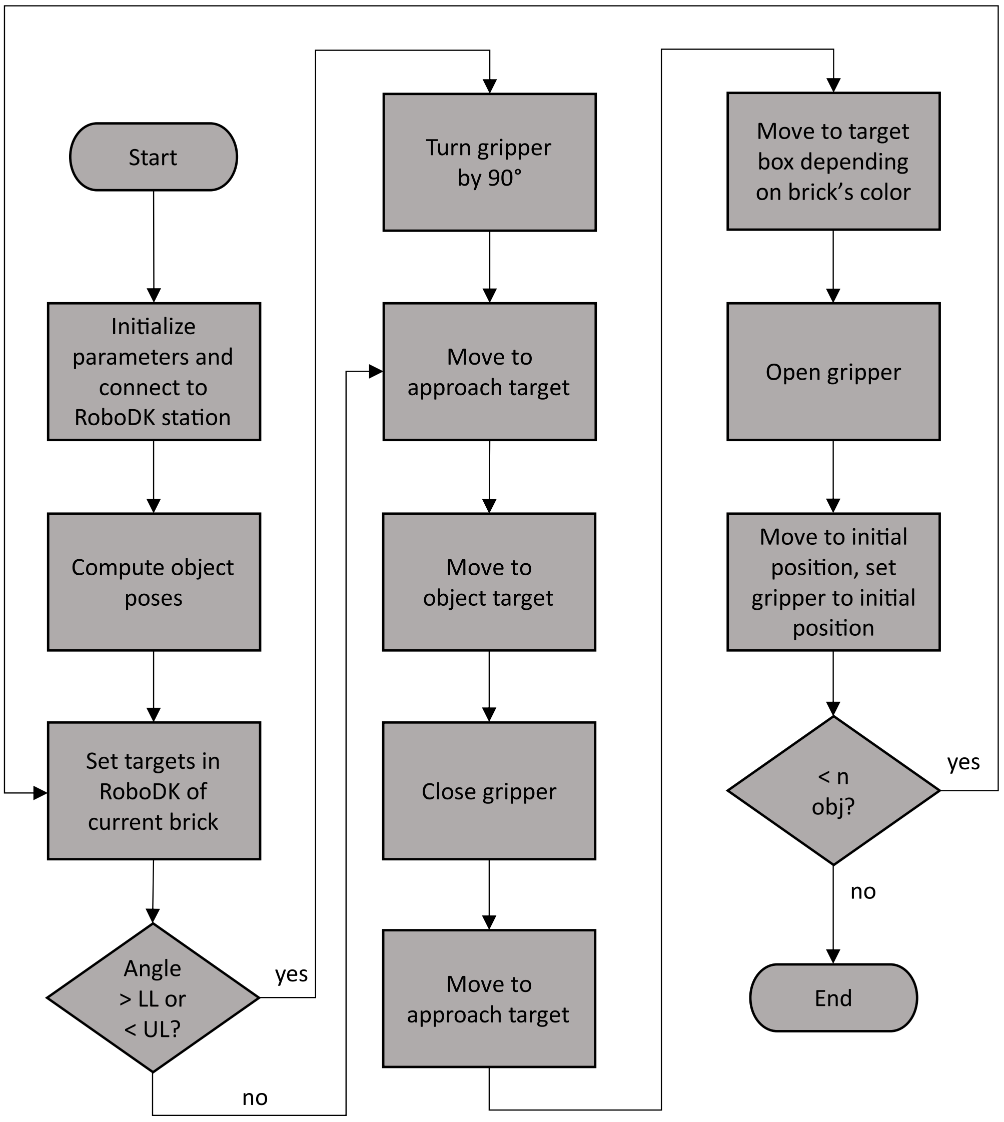

Start
Initialize parameters and connect to RoboDK station
Compute object poses
Set targets in RoboDK of current brick
Angle > LL or < UL?
yes
no
Turn gripper by 90°
Move to approach target
Move to object target
Close gripper
Move to approach target
Move to target box depending on brick's color
Open gripper
Move to initial position, set gripper to initial position
< n obj?
yes
no
End

Figure 27: process of pick and place detected bricks

The gripper closes its fingers to grasp the brick and the TCP is moved back to the approach position for avoiding collisions with other bricks. The robot performs a joint movement to the target above the corresponding box according to the color in the label name of the bounding box which was the output of the object detection (chapter 3.3.3). When the gripper with the grasped brick is located above the box according to the brick color, the gripper opens so that the brick falls into the box. The robot moves to the initial position and if there aren't any more bricks left the pick and place process stops. All the movements are performed by running the programs listed in Table 5 with the method *RunProgram* of the RoboDK MATLAB API.

Table 7: gripper's finger position according to brick orientation

brick's rotation angle	Position of fingers
<= -125°	Turned by 90°
> -125° & < -55°	Initial position
>= -55° & <= 55°	Turned by 90°
> 55° & < 125°	Initial position
>= 125°	Turned by 90°

3.5 Final setup

This chapter details the final setup of the developed pick-and-place application with computer vision, involving the robot UR3e with the integrated gripper RGI14 and the RGB-D camera RealSense L515.

In order to also establish robust mechanical connections between the selected components, a total of three distinct flanges are designed and manufactured in aluminum (Figure 28):

1. flange for the connection of robot UR3e to the table plate
2. flange to connect gripper RGI14 to robot UR3e
3. flange for connection of camera L515 to gripper RGI14 and robot UR3e

The flanges each are designed with holes with fine tolerances so that the flange between the robot and the table and the flange between the robot and the gripper can be assembled with dowel pins. This ensures a repeatable setup. The hole diameters and distances are designed according to the predefined dimensions of the robot UR3e [27].

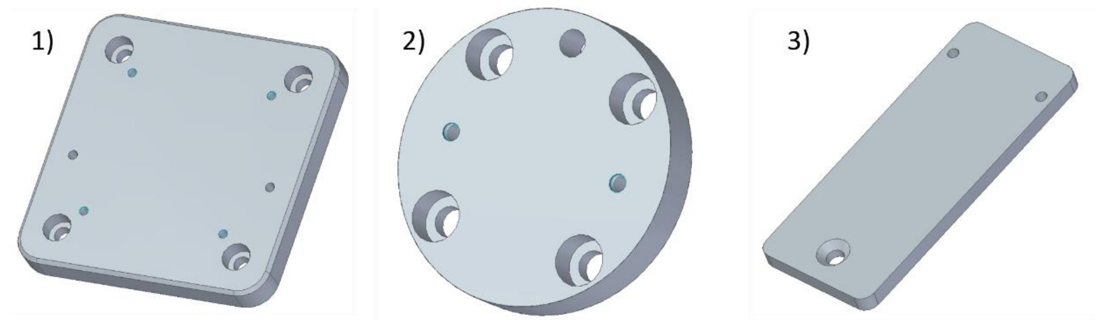


Figure 28: CAD models of flanges

The real station is assembled according to the setup of the simulation model in RoboDK (chapter 3.2). Figure 29 shows the robot UR3e with the integrated camera and gripper in the laboratory. The camera is pre-fixed on a threaded screw and, in the snapshot target position, aligned parallel to the table surface with an electronic protractor (Figure 30). It is then fixed in place with a lock nut. This ensures the correct estimation of the bricks' poses relative to the camera position. By connecting from RoboDK to the real robot and activating for each program of the virtual station the option "Run on robot", the instructions in within a program are run parallelly in the simulation as well as on the real robot.

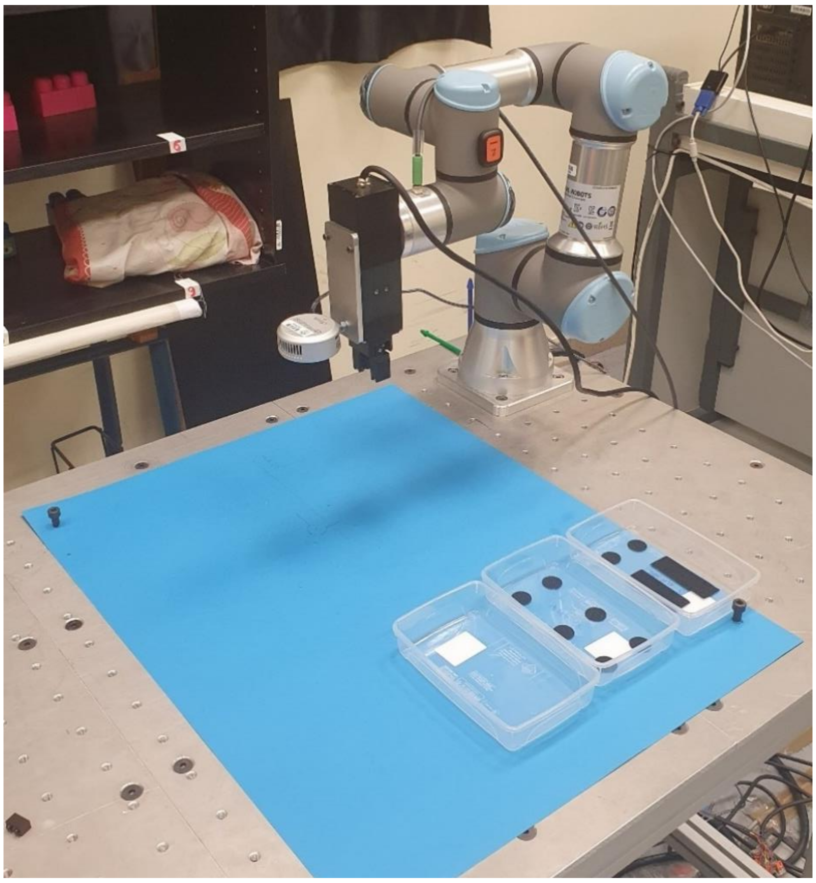


Figure 29: real setup of pick-and-place station

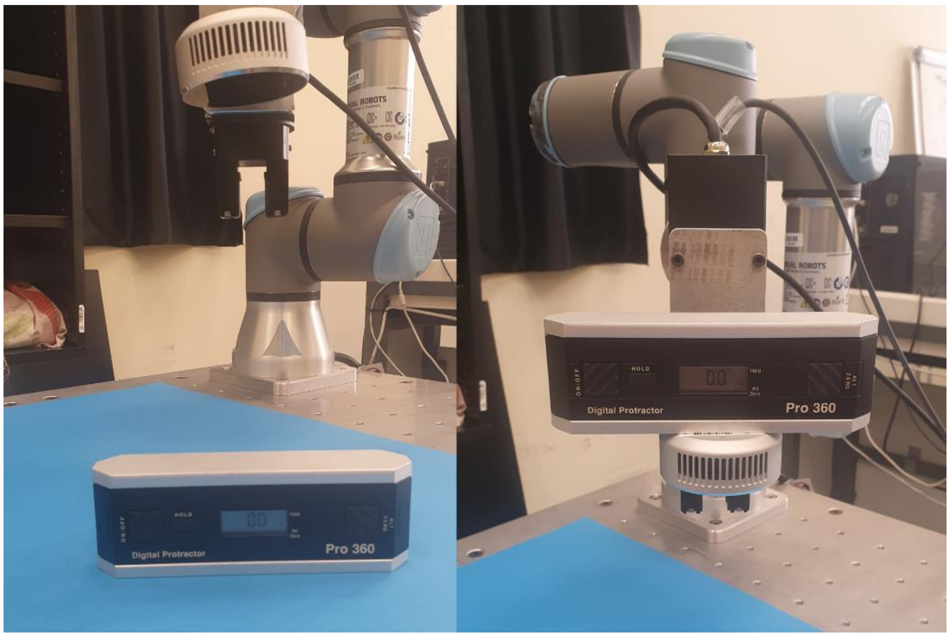


Figure 30: Camera alignment in real setup

The first tests show a difference in the position estimation between the simulation model and the real robot. While the object detection algorithm always estimates the correct center of gravity of the brick with the image data of the simulation model, there is a deviation from the center of gravity to the estimated one of the bricks in the real setup. But the orientation is already estimated correctly so that the fingers of the gripper are positioned parallel to the principal axis of the brick. A remeasurement of the real camera position shows that there is a small difference in comparison to the simulation model assembly with the nominal CAD models. Table 8 compares the values of the camera position in respect to the base coordinate system in RoboDK of the simulation model setup and the final real setup. With the correction of the camera position of 2,8mm in the y-axis, the real setup works and estimates the position of the detected bricks correctly.

The steps of the workflow for the pick-and-place station of the simulation model and of the real setup of the robot UR3e with the integrated camera and gripper are listed in Table 9.

Table 8: camera position of simulation model and real setup in robot base coordinate system

	Simulation model	Real setup
X [mm]	131,1	131,1
Y [mm]	-397,2	-400,0
Z [mm]	210,1	210,1
u [°]	0	0
v [°]	180	180
w [°]	0	0

Table 9: Workflows of pick and place application

Simulation model	Real setup
	Connect RealSense L515 to PC
Open Station *Simulation_Movement in RoboDK*	Open Station *RealRobot_Movement in RoboDK*
	Connect RoboDK with real robot UR3e
Run MATLAB script *brick_positioning*	Place bricks randomly below the *Take Snapshot* position
Run MATLAB script *RoboDK_BrickSorting*	Run MATLAB script *UR3e_BrickSorting*
Delete added bricks from station	Empty boxes of sorted brick

4 Results and discussion

The functionality of the simulation model is proven, all randomly generated bricks are correctly detected as shown exemplary in Figure 31. The confusion matrix in Figure 32 shows the result of an object detection test for ten simulated images with eight randomly chosen and placed bricks per image. It displays that every class of a detected brick is predicted as the correct ground truth class as true positive. The pose estimation is accurate enough and within the tolerance so that there are no collisions between the gripper with an opening width of 14mm and the bricks with a width of 11,2mm during the grasping process (Figure 33). The trained object detector classifies the different bricks correctly so that they're sorted into the correct boxes according to their color.

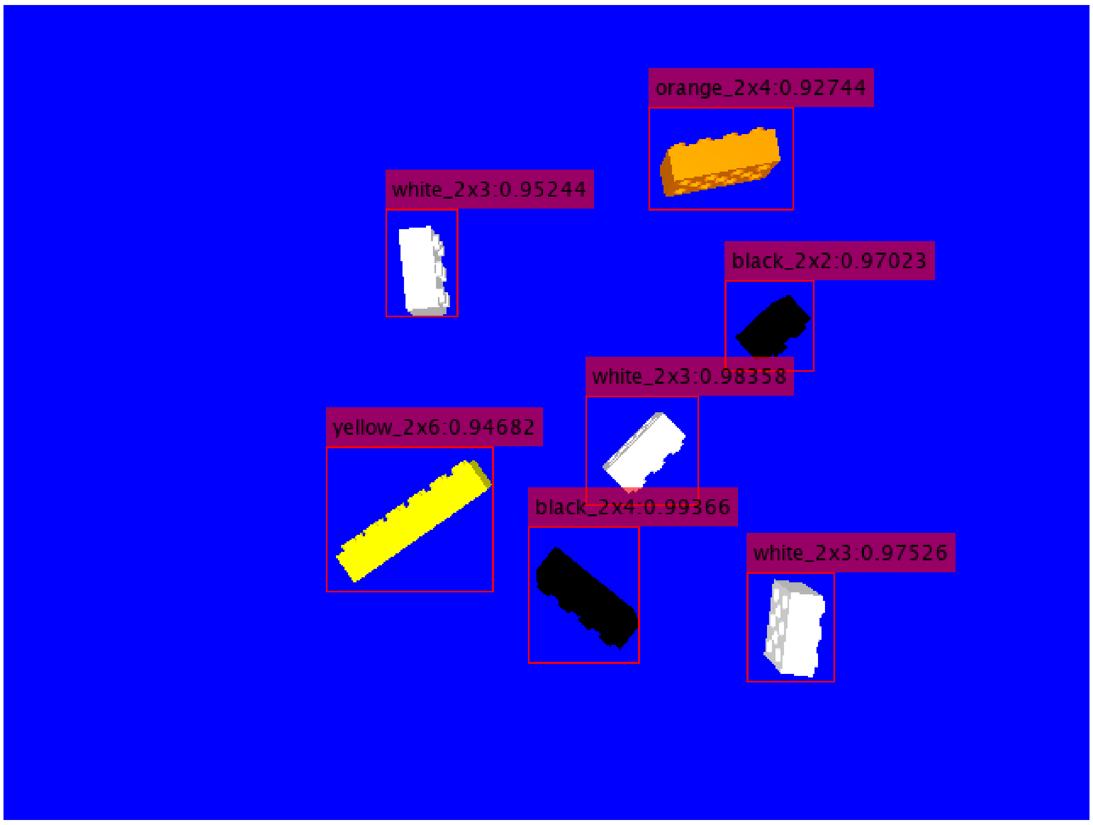


Figure 31: Object detection test of simulated camera image

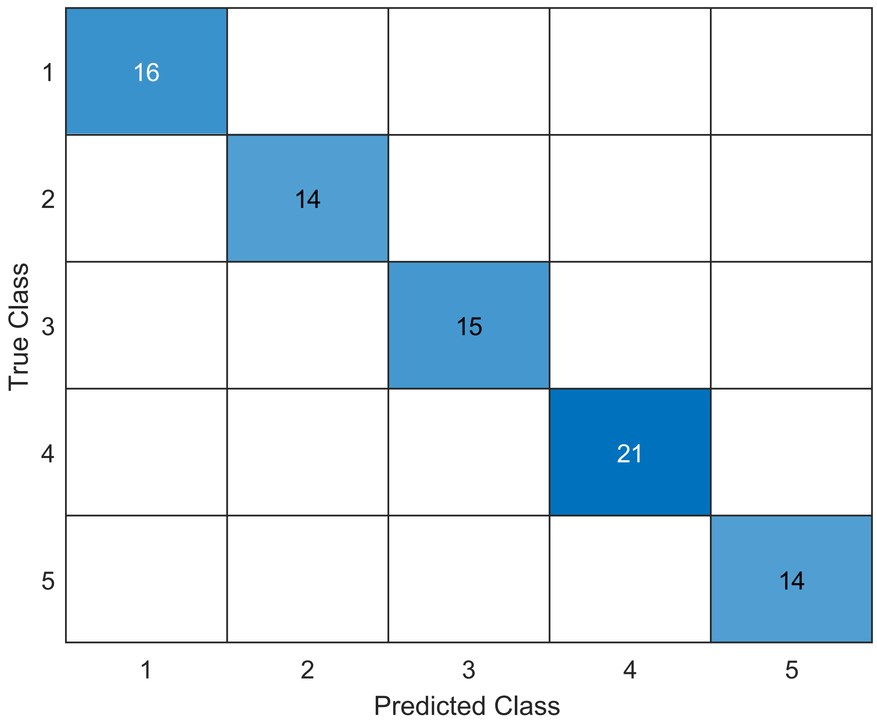


Figure 32: Confusion matrix for object detection of brick images with simulated camera; classes: 1 = “black_2x2”; 2 = “white_2x3”; 3 = “black_2x4”; 4 = “orange_2x4”; 5 = “yellow_2x6”

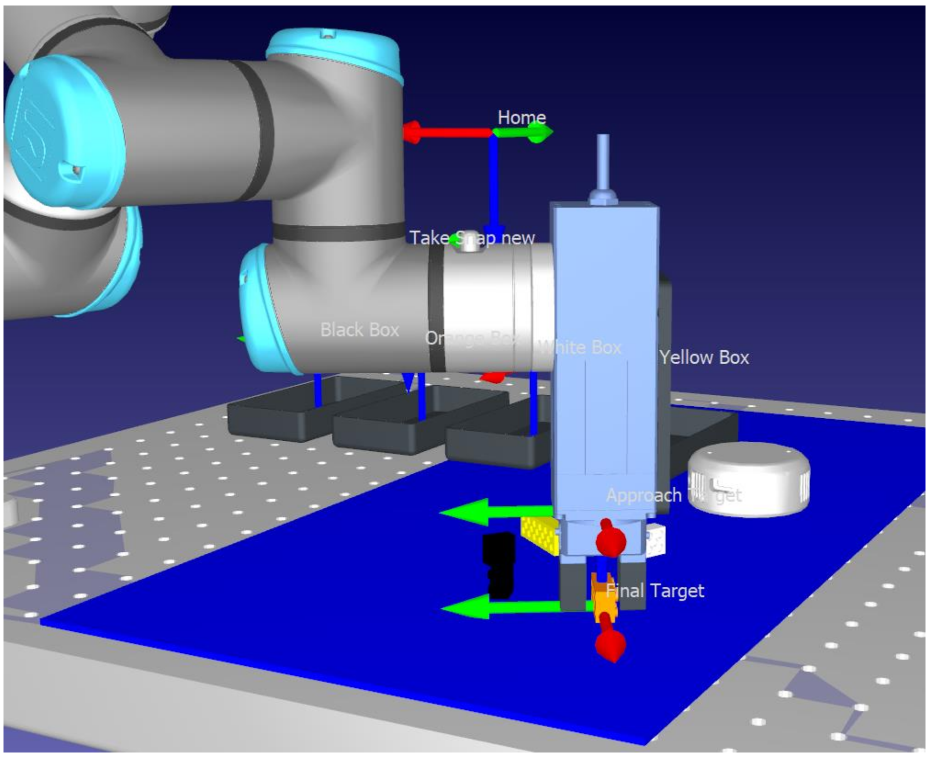


Figure 33: Functionality of simulation model

The general functionality of the simulation model can be also transferred to the real setup. The majority of the plastic bricks are detected by the pre-trained network with the image date of the RealSense L515. The position is estimated correctly of the detected bricks so that the gripper can approach and grasp them (Figure 34).

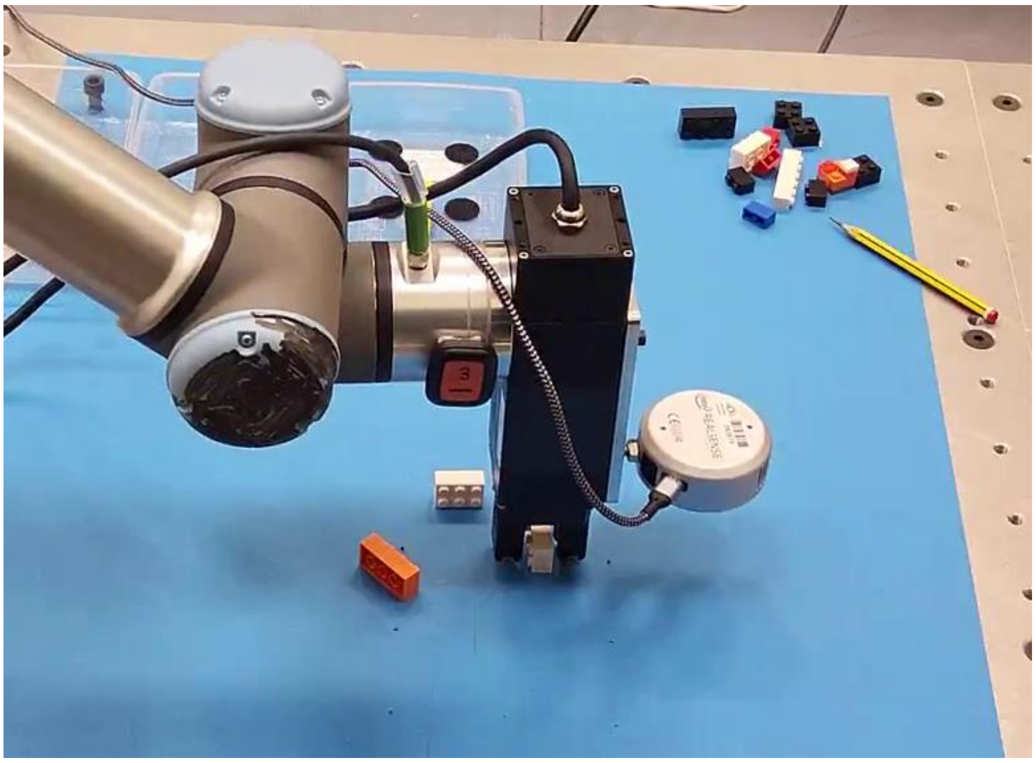


Figure 34: Functionality of real setup

If the tiny YOLOv4 object detection network trained only with the synthetic brick data is used for the object detection of the real plastic bricks, the accuracy of detecting the bricks isn’t 100%. Randomly, there are bricks which aren’t detected and therefore the PCA-based pose estimation doesn’t have a bounding box for this brick as an input. Without the bounding box the point cloud data of this brick won’t be extracted, and no target for grasping the brick will be generated. Furthermore, there are repetitive cases of wrong classification between the yellow and the orange bricks: orange bricks are often classified as yellow bricks by the trained object detector. Figure 35 shows the missing detection of the white brick and the misclassification of the orange brick as a yellow brick. For other colors and if the brick was detected, the classification is correct with a confidence score higher than 0,5.

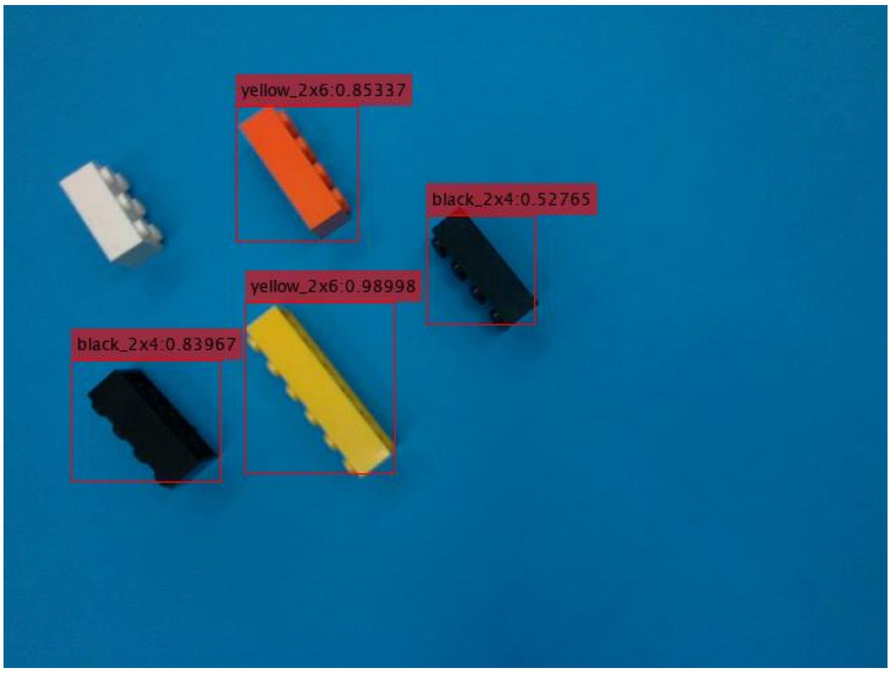


Figure 35: missing bounding box of white brick

Figure 36 displays that it shows also positive results of the recognition of bricks and computing the bounding box at the correct position testing the object detection on another surface with another background color. But still, the orange brick is misclassified as yellow.

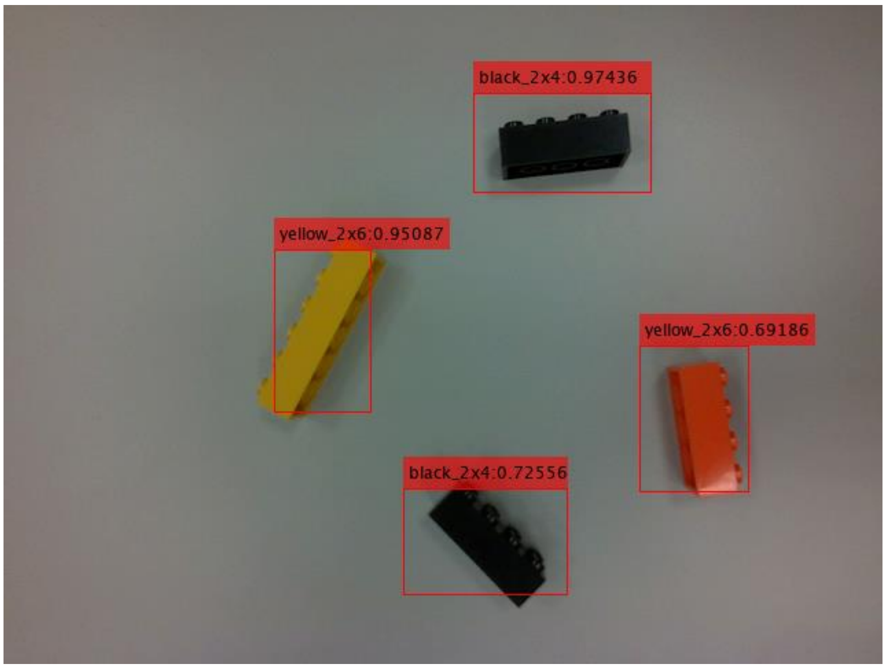


Figure 36: Object detection of real bricks with another background

For estimating the detected brick’s position with the point cloud data of the RealSense L515 camera, the PCA-based algorithm is accurate enough to correctly compute the center of gravity and the orientation. Figure 37 shows the results of the pose estimation of five detected bricks.

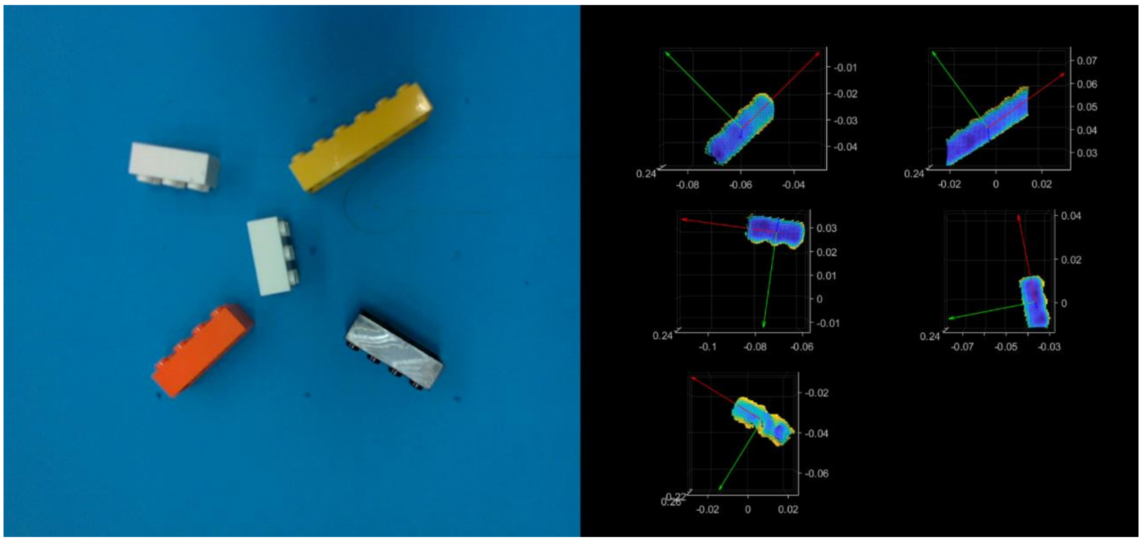


Figure 37: correct pose estimation of RealSense L515 data

After several movements of the robot, the added gripper and camera are still in the initially position when mounted with the designed flanges. There is no clearance between the parts and there are no loose screws.

The outcomes of the pick and place application, both with the simulation model and the real setup, reflect the accuracy and limitations of the implemented system. In the case of the simulation model, it is evident that the integration in RoboDK has yielded positive results. The object detection module, utilizing the pre-trained network, exhibits robustness in accurately classifying different brick types in simulated image data. The pose estimation, a critical aspect to ensure collision-free interactions, demonstrates satisfactory accuracy, thereby enabling sorting into designated boxes based on color.

Translating the functionality to the real setup showcases encouraging results. Plastic bricks can be successfully detected using the RealSense L515 image data, affirming the real-world viability of the object detection process. Accurate position estimation of detected bricks empowers the gripper to approach and grasp them seamlessly, validating the system's adaptability to physical interactions.
However, it is worth noting that certain challenges emerged when utilizing the tiny YOLOv4 object detection network trained solely with synthetic brick data for real plastic bricks. The detection accuracy isn't consistently perfect, leading to instances where bricks

remain undetected. In order to improve this issue, the network could be trained additionally with real data. The final solution of such applications could be an object detection network majorly trained with synthetic data, e.g., during the design phase of a product, and then additionally trained with real image data of the first samples or the final product. This way, the influence of the lighting and textured backgrounds in real-world applications is considered as well.
The additional training data of real-world scenes could also improve the issue with the misclassification between yellow and orange bricks. As the colors yellow and orange are side by side in the color chart small effects due to the room light or sun light could lead to a misclassification already.

Despite these challenges, the PCA-based pose estimation algorithm displays noteworthy accuracy in determining the position of detected bricks using the RealSense L515 camera's point cloud data detected with its LiDAR sensor.

5 Conclusion and future Prospects

In conclusion, an AI-based object detection pick and place application can be realized in a fast and robust manner, as shown in this work. The primary objective was to establish a system that employs a robotic arm and gripper for pick and place tasks, with the added capabilities of adaptive motion control through AI and object detection. By the design and manufacturing of aluminum flanges, the UR3e robot, the RGI14 gripper and the RealSense L515 camera are robustly attached to each other.

The achieved results in the pick and place application with a simulation model attest to the robust functionality of the system. The simulation model successfully detects randomly generated bricks and executes accurate pose estimations, so that collisions between the gripper and bricks are avoided. Furthermore, the trained object detection model exhibits the ability to classify diverse bricks accurately, thereby enabling their sorting into the corresponding color-coded bins.

Translating the simulation model's capabilities to a real setup validates the system's practicality. The majority of plastic bricks are reliably detected by the with synthetic data pre-trained tiny YOLOv4 object detection network utilizing color image data of the RealSense L515 camera. Moreover, the PCA-based algorithm showcased its precision in estimating the positions of detected bricks through the RealSense L515's point cloud data. This allows successful gripping by the integrated RGI14 gripper by DH-Robotics, contributing to the overall efficiency of the process.

However, when employing object detection network trained solely with synthetic brick data, the detection accuracy for real-world images of plastic bricks is not flawless. Instances of undetected bricks lead to missing bounding boxes, affecting the subsequent PCA-based pose estimation. Notably, classification errors between yellow and orange bricks were identified, primarily due to the trained object detector misclassifying orange bricks as yellow. Further investigation is required to mitigate these issues and enhance detection accuracy, particularly for different backgrounds and lighting conditions.

Overall, a flexible station has been developed which can be used in different environments because the camera is fixated directly to the robot. So only the robot would have to be mounted on another surface and the pick and place algorithm would run the same way as in this work. The accomplished objectives highlight the potential of the developed pick and place system but underscore the need for ongoing refinement to optimize object detection accuracy and classification reliability.

For refining this work or adding new functionalities to the developed pick and place system, several proposals can be made for going into detail in one of the fields of object detection, motion control or the use of the gripper.

The trained tiny YOLOv4 network could be improved by adding more or other training data, e.g., with image data of real bricks or with textured backgrounds and simulated lighting conditions for the synthetic data. Also, the object detection network could be trained with other objects than bricks so that the developed system can not only sort the same object type with similar shape and different colors but also classify different types of objects and sort them into different boxes. In this work, the object detection and pick and place process are performed in the two-dimensional space. A future work could concentrate on bricks that are placed on planes in different heights or even inclined bricks touching each other in order to create a 3D-problem.

The trajectory planning of the motion control in this work is carried out by the software RoboDK. An improvement could be to use the integrated camera for collision detection of randomly placed obstacles during the robot's movements targeting the objects to grasp. Then, a trajectory replanning could be developed for still reaching the target object.

Furthermore, a future work could focus on an improvement of the gripper's functionality. With a new finger design for the gripper, also bigger objects could be handled. A new design could also make the gripper more flexible so that different types of objects can be grasped with the same gripper. The rotary function of the gripper was only used for two specified positions in this work. For new applications like screwdriving, the infinite motion range of the gripper's rotary movement could be employed.

Bibliography

[1] SiMuR, "SiMuR research group," 2023. [Online]. Available: https://simur.dieecs.com/.

[2] M. Hägele, K. Nilsson, J. Norberto Pires and R. Bischoff, "Industrial Robots," in *Handbook Robotics*, Berlin Heidelberg, Springer-Verlag , 2016.

[3] P. Corke, "Robotics, Vision, Control," Springer Cham, 2017.

[4] MathWorks, "MathWorks - What Is Computer Vision?," 2023. [Online]. Available: https://www.mathworks.com/discovery/computer-vision.html.

[5] MathWorks, "Image Processing and Computer Vision - Object Recognition," 2023. [Online]. Available: https://www.mathworks.com/solutions/image-video-processing/object-recognition.html.

[6] J. Colombel , J.-L. Charles and R. Fabre, "ROS4.pro - Comparison of robotic simulator," 19 11 2021. [Online]. Available: https://learn.e.ros4.pro/en/robotic_simulators/comparison/.

[7] M. E. Matour and A. Winkler, "A Portable Vision-based and Force Controlled Collaborative Robot System for Entertainment Purposes," in *ISR Europe 2022; 54th International Symposium on Robotics*, 2022, pp. 1-6.

[8] MathWorks, "Intelligent Bin Picking in MATLAB® for Universal Robots," 2023. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/125240-intelligent-bin-picking-in-matlab-for-universal-robots?s_tid=answers_rc2-2_p5_BOTH.

[9] Data Base Camp, "Principal Component Analysis – einfach erklärt!," 9 2 2022. [Online]. Available: https://databasecamp.de/statistik/principal-component-analysis.

[10] Analytics Vidhya, "PCA | What is Principal Component Analysis & how it works?," 3 2016. [Online]. Available: https://www.analyticsvidhya.com/blog/2016/03/pca-practical-guide-principal-component-analysis-python/.

[11] Universal Robots, "UR3e Roboter," 2023. [Online]. Available: https://www.universal-robots.com/de/produkte/ur3-roboter/.

[12] Schweikert Automation, "DH Robotics RGI-14 Drehgreifer," 2023. [Online]. Available: https://schweikert-automation.de/product/dh-robotics-rgi-14-drehgreifer/.

[13] DH-Robotics, "RGI-14 User Manual," 2020. [Online]. Available: https://www.trossenrobotics.com/Shared/DH/RGI-14UserManual.pdf.

[14] Intel RealSense, "Intel® RealSense™ LiDAR Camera L515," 2023. [Online]. Available: https://www.intelrealsense.com/lidar-camera-l515/.

[15] Intel, "Intel RealSense LiDAR Camera L515 Datasheet," 2021. [Online]. Available: https://dev.intelrealsense.com/docs/lidar-camera-l515-datasheet.

[16] RoboDK, "RoboDK - Station Library," 2023. [Online]. Available: https://robodk.com/stations#.

[17] Intel RealSense, "Intel Realsense Matlab Wrapper," 2023. [Online]. Available: https://dev.intelrealsense.com/docs/matlab-wrapper.

[18] RoboDK, "RoboDK Matlab API," 2023. [Online]. Available: https://robodk.com/Matlab-API.

[19] Intel, "Intel RealSense SDK2.0," 2023. [Online]. Available: https://www.intelrealsense.com/sdk-2/.

[20] RoboDK, "Robot Calibration - Connect Robot," 2023. [Online]. Available: https://robodk.com/doc/en/Robot-Calibration-LaserTracker-Connect-robot.html.

[21] DH-Robotics, "RGI Series Robot Plug-in," 2023. [Online]. Available: https://www.dh-robotics.com/wp-content/uploads/2022/12/RGI.zip.

[22] Intel RealSense, "Intel RealSense CAD Files," 2023. [Online]. Available: https://dev.intelrealsense.com/docs/cad-files.

[23] DH-Robotics, "DH-Robotics Download Center," 2023. [Online]. Available: https://en.dh-robotics.com/service/download.

[24] TraceParts, "TraceParts LEGO bricks," 2023. [Online]. Available: https://www.traceparts.com/en/search/legor-bricks?CatalogPath=LEGO%3ALEGO.010.

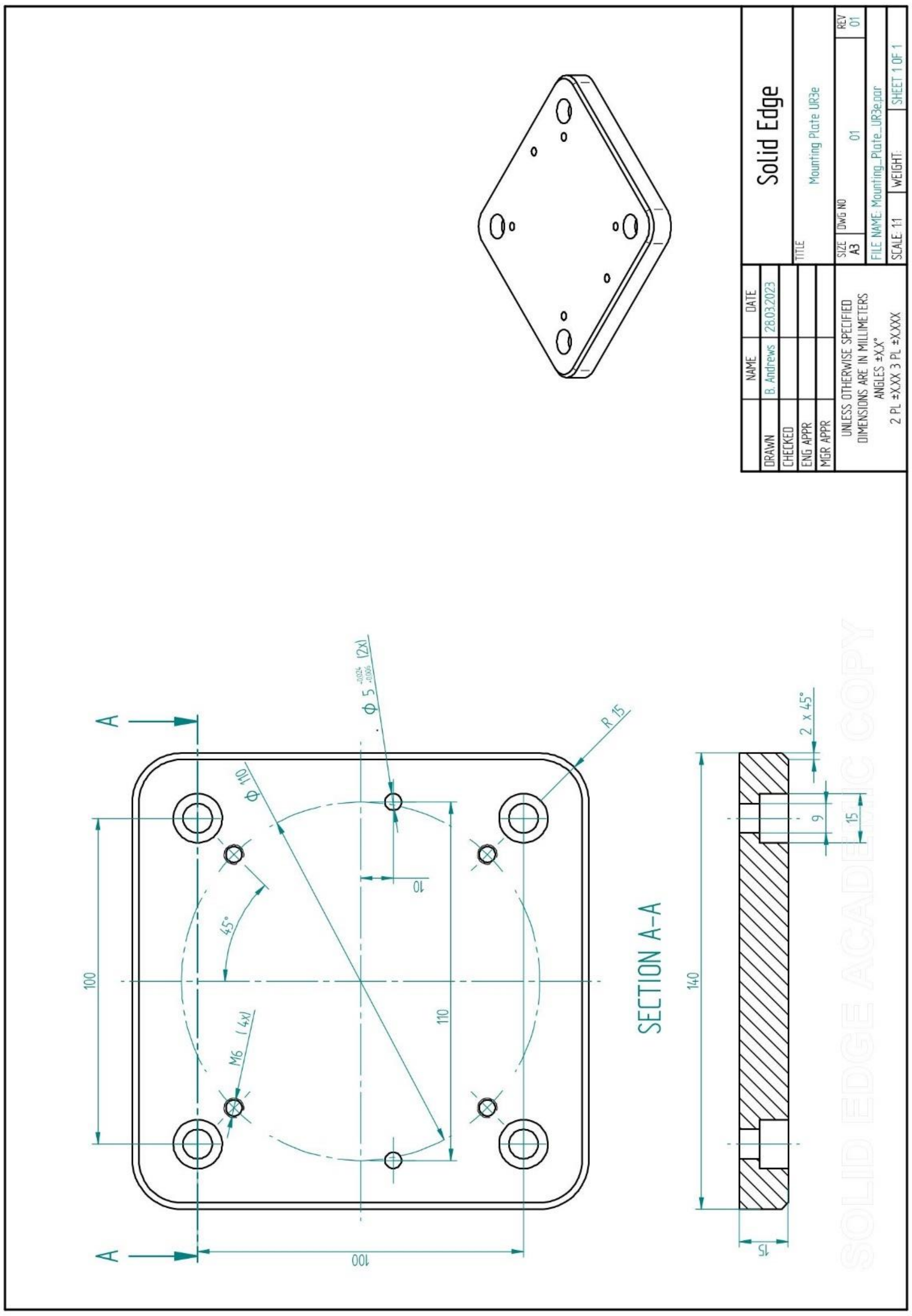
[25] MathWorks, "Gazebo Simulation of Semi-Structured Intelligent Bin Picking for UR5e Using YOLO and PCA-Based Object Detection," 2023. [Online]. Available: https://www.mathworks.com/help/supportpkg/urseries/ug/gazebo-simulation-ur5e-semistructured-intelligent-bin-picking-example.html.

[26] MathWorks, "pcfitplane," 2023. [Online]. Available: https://www.mathworks.com/help/vision/ref/pcfitplane.html.

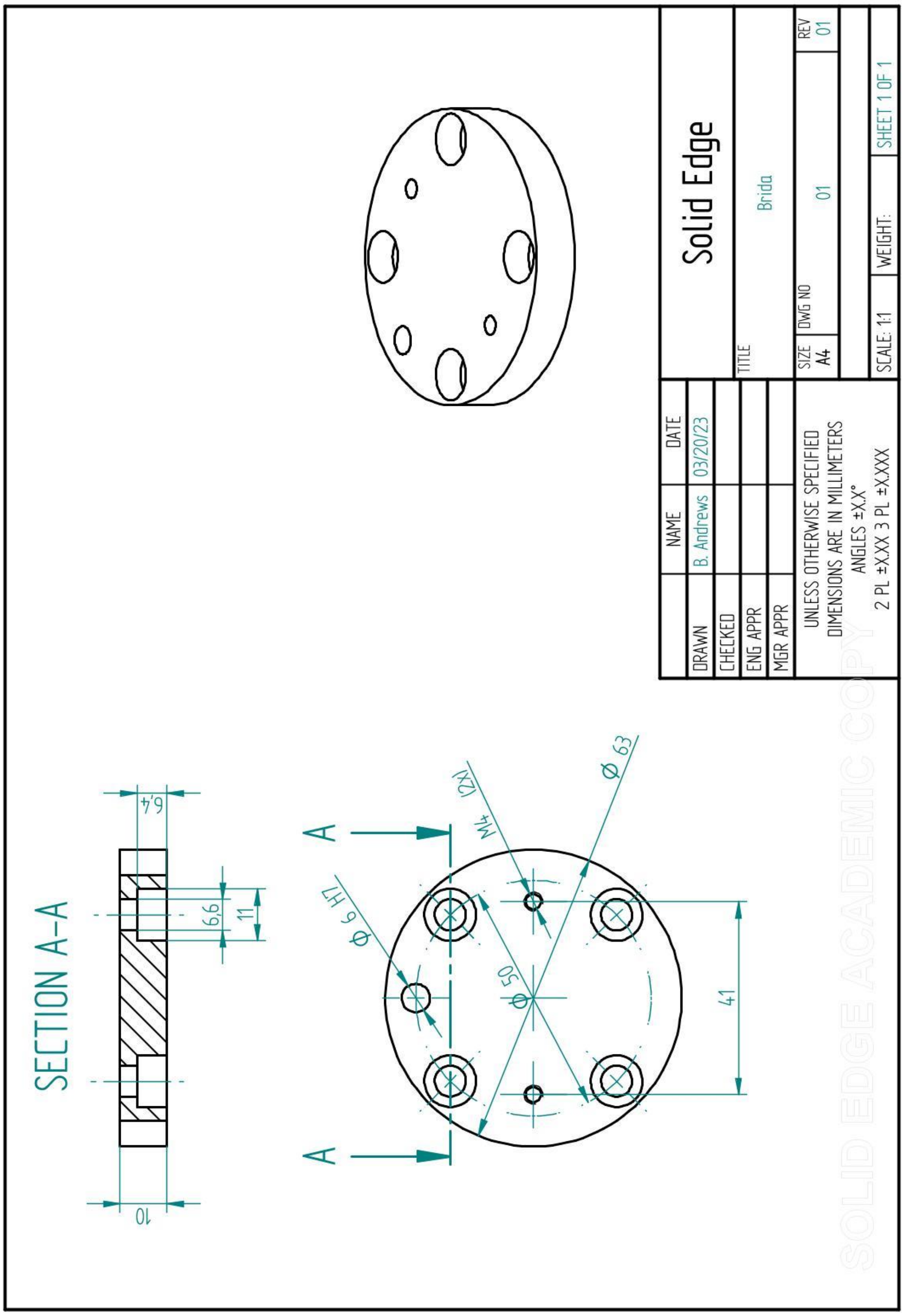
[27] Universal Robots, "Universal Robots e-Series User Manual Version 5.0.2," 2023. [Online]. Available: https://s3-eu-west-1.amazonaws.com/ur-support-site/41166/UR3e_User_Manual_en_Global.pdf.

A Appendixes

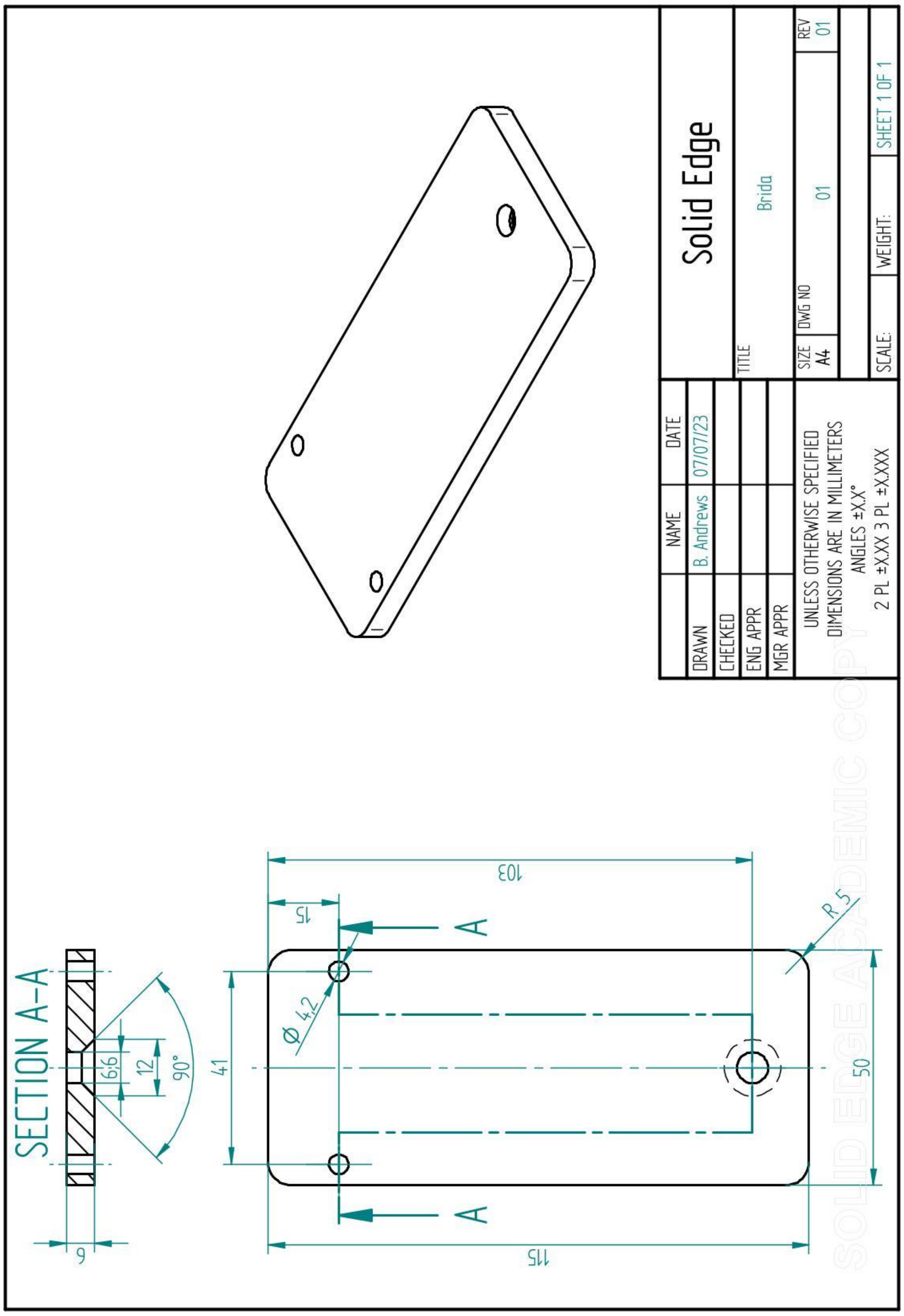
A.1 Drawing of flange robot UR3e – table



A.2 Drawing of flange robot UR3e – gripper



A.3 Drawing of flange gripper – camera


Solid Edge
TITLE
Brida
SIZE
A4
DWG NO
01
REV
01
SCALE:
WEIGHT:
SHEET 1 OF 1
NAME
DATE
DRAWN
B. Andrews
07/07/23
CHECKED
ENG APPR
MGR APPR
UNLESS OTHERWISE SPECIFIED
DIMENSIONS ARE IN MILLIMETERS
ANGLES ±X.X°
2 PL ±X.XX 3 PL ±X.XXX
SECTION A–A
6:6
12
90°
41
9
103
15
A
A
Ø 4,2
R 5
50
115
SOLID EDGE ACADEMIC COPY