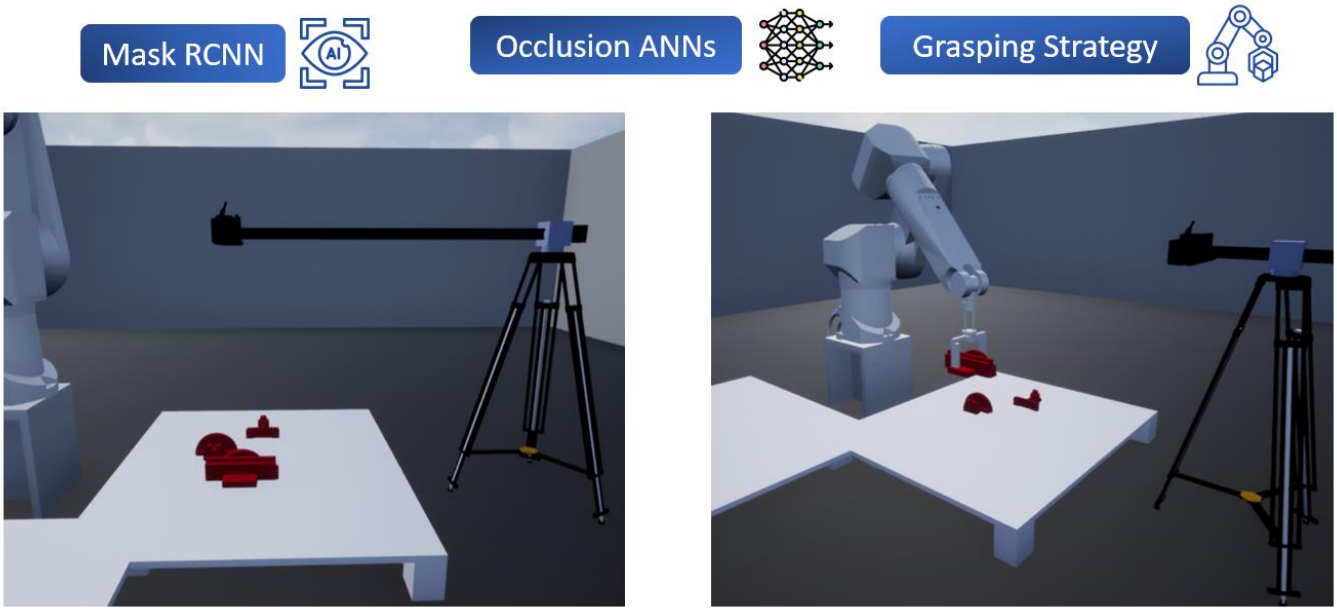
National Technical University of Athens
School of Mechanical Engineering
Section of Manufacturing Technology

Robotic Arm Manipulation for Object Detection & Grasping in Occlusion Environments Using Machine Vision & Neural Networks

Diploma Thesis

Pavlos Chionidis



Supervisor: Prof. Panorios Benardos (NTUA)

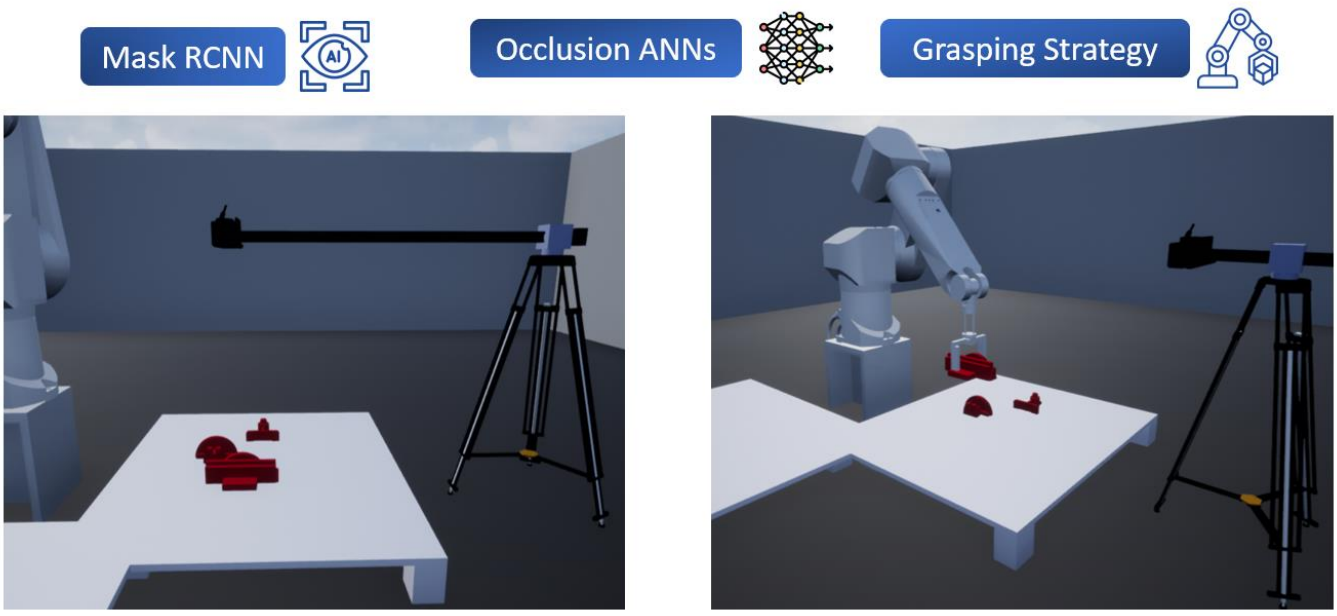
Athens 2024

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών
Τομέας Τεχνολογίας των Κατεργασιών

Ρομποτικός Βραχίονας με Σύστημα Μηχανικής Όρασης & Νευρωνικών Δικτύων για την Αναγνώριση & την Αρπαγή Αλληλεπικαλυπτόμενων Αντικειμένων

Διπλωματική Εργασία

Παύλος Χιονίδης



Υποβλέπων: Πανώριος Μπενάρδος, Καθηγητής ΕΜΠ

Αθήνα 2024

-- Blank Page --

Foreword

This diploma thesis was conducted during the academic year 2023-2024 at the Mechanical Engineering School of the National Technical University of Athens, specifically within the Section of Manufacturing Technology.

I extend my sincere appreciation to my supervisor, Mr. P. Benardos, for his guidance and support throughout the duration of this project. The section of Manufacturing Technology is also acknowledged for their trust in providing access to necessary equipment.

A heartfelt thank you goes to my family for their consistent support and upbringing that laid the foundation for my academic pursuits. Their encouragement has been a driving force behind my efforts.

I would like to thank my friends for their support and assistance in improving my writing skills throughout the academic years.

As this thesis concludes, it reflects not only individual effort but also the collective support received from various quarters. May this work contribute meaningfully to the academic discourse in the field of machine vision and robotics.

Pavlos Chionidis
Athens, February 2024

Contents

Abstract

This diploma thesis focuses on the simulation of a robotic manipulator with machine vision to grasp objects in occlusion environments. Specifically, the robotic arm in use is the Stäubli RX90L with 6 degrees of freedom, equipped with a gripper. The study delves into the assessment of inverse kinematics and trajectory control during grasping operations. A crucial augmentation to the system involves the incorporation of machine vision systems for object detection. The machine vision component encompasses the calibration of Kinect V2 RGB and depth images, along with the mapping of RGB images to depth images. Subsequently, the theoretical underpinnings of Convolutional Neural Networks are elucidated, with the chosen network for this thesis being the Mask RCNN. The methodology for training the network is explicated, followed by comprehensive testing in diverse occlusion environments.

A paramount aspect of this work involves addressing the occlusion problem in the grasping methodology, determining which objects are occluded, by implementing neural networks that utilize features extracted from the Mask RCNN network and depth images. The acquired occlusion data undergoes a logic-based algorithm, delineating the item to be grasped and generating an effective grasping strategy. The entire process is simulated within the integrated MATLAB-Simulink Unreal Engine environment, providing a holistic evaluation of the proposed methodologies. This comprehensive exploration encompasses robotic arm control, machine vision implementation, advanced neural network training, occlusion problem resolution, and subsequent simulation within a virtual environment.

Περίληψη

Η παρούσα διπλωματική εργασία εστιάζει στην προσομοίωση ενός ρομποτικού βραχίονα με μηχανική όραση για την σύλληψη αντικειμένων σε περιβάλλοντα αλληλοεπικάλυψης. Πιο συγκεκριμένα, ο ρομποτικός βραχίονας που χρησιμοποιείται είναι ο Stäubli RX90L με 6 βαθμούς ελευθερίας, εξοπλισμένος με αρπάγη. Η μελέτη εμβαθύνει στην υλοποίηση της αντίστροφης κινηματικής και του ελέγχου της τροχιάς κατά τη σύλληψη των αντικειμένων. Κρίσιμη είναι η ενσωμάτωση του συστήματος μηχανικής όρασης για την ανίχνευση των αντικειμένων. Το κεφάλαιο της μηχανικής όρασης περιλαμβάνει τη βαθμονόμηση και την αντιστοίχιση των εικόνων RGB στις εικόνες βάθους του Kinect V2. Στη συνέχεια, γίνεται μία αναφορά στην θεωρία των συνελικτικών νευρωνικών δικτύων, με το δίκτυο που επιλέχθηκε για τη διατριβή αυτή, να είναι το Mask RCNN. Έπειτα ακολουθείται η μεθοδολογία για την εκπαίδευση του δικτύου, με την δοκιμή του σε διάφορα περιβάλλοντα αλληλοεπικάλυψης.

Μια πρωταρχική πτυχή της εργασίας περιλαμβάνει την αντιμετώπιση του προβλήματος αλληλοεπικάλυψης των αντικειμένων. Για τον προσδιορισμό των αντικειμένων που επικαλύπτονται, χρησιμοποιήθηκαν νευρωνικά δίκτυα με είσοδο χαρακτηριστικά που εξάγονται από το δίκτυο Mask RCNN και τις εικόνες βάθους. Η έξοδος από τα νευρωνικά δίκτυα χρησιμοποιείται από αλγόριθμο, ο οποίος οριοθετεί το αντικείμενο που πρέπει να συλληφθεί και δημιουργεί την στρατηγική σύλληψης. Η όλη διαδικασία προσομοιώνεται στο περιβάλλον του MATLAB-Simulink-Unreal Engine, παρέχοντας μια ολική εικόνα των προτεινόμενων μεθοδολογιών. Δηλαδή η διπλωματική αυτή περιλαμβάνει τον έλεγχο ρομποτικού βραχίονα, την εφαρμογή του συστήματος μηχανικής όρασης, την εκπαίδευση των νευρωνικών δικτύων, την επίλυση των προβλημάτων αλληλοεπικάλυψης και την επακόλουθη προσομοίωση σε εικονικό περιβάλλον.

List of Abbreviations

2D: Two Dimensional

3D: Three Dimensional

ANN: Artificial Neural Network

API: Application Programming Interface

ASIC: Application Specific Integrated Circuits

BFGS: Broyden–Fletcher–Goldfarb–Shanno

CAD: Computer Aided Design

CNN: Convolutional Neural Network

CPU: Central Processing Unit

DOF: Degrees Of Freedom

FBX: Filmbox Format

GPU: Graphics Processing Unit

MS COCO/ COCO: Microsoft Common Objects in Context

PAR: Pixel Aspect Ratio

RCNN: Region-based Convolutional Neural Network

RAM: Random Access Memory

ReLU: Rectified Linear Unit

RGB: Red Green Blue

RGB-D: Red Green Blue - Depth

RPN: Region Proposal Network

STL: Standard Triangle Language

UE: Unreal Engine

URDF: Unified Robot Description Format

VOC: Visual Object Classes

VRAM: Video Random Access Memory

YOLO: You Only Look Once

1. Introduction

Automation has become a cornerstone in modern industry and daily life, playing a pivotal role in enhancing efficiency, precision, and safety. One of the primary advantages is the substantial increase in efficiency through the reduction of manual labor, as robots excel at performing repetitive tasks with unwavering precision and speed. The quest for precision and accuracy in various applications is addressed adeptly by robots, which can achieve levels of intricacy challenging for human operators.

Moreover, the economic landscape benefits significantly from automation, as robots contribute to cost reduction in the long term. Operating 24/7 with minimal maintenance needs, robots streamline production processes, resulting in increased productivity and reduced operational costs. The integration of automation fosters safer working environments by assigning hazardous or physically demanding tasks to robots, mitigating the risks of accidents and injuries to human workers, especially in industries such as manufacturing and logistics.

Machine vision alongside advanced grasping techniques in robotic manipulators finds widespread application in scenarios characterized by dynamic and varied environments. This utility extends across diverse domains, ranging from the sorting of objects in different environments [5] to the agricultural industry [6]. Consequently, the incorporation of machine vision techniques in modern robotics applications assumes substantial significance, facilitating adaptability and precision in addressing the evolving demands of dynamic operational settings.

The challenge of occlusion remains a focal point of research for numerous institutions and researchers globally. This persistent focus stems from the profound implications of solving the occlusion problem, which could propel humanity towards realizing a machine vision-robotic system with human-like perception. Such a breakthrough holds the potential to revolutionize various sectors, particularly by significantly enhancing the deployment of robotic manipulators in manufacturing facilities and everyday environments. By mitigating occlusion issues, these advanced systems would operate with heightened efficiency and adaptability, ushering in a new era of automation and integration into daily life. As a result, the pursuit of solutions to occlusion represents a crucial step forward in advancing the capabilities and impact of machine vision and robotics technologies.

While significant research is conducted in this domain, the implementation of findings often lacks clear methodologies, leaving uncertainty regarding practical application. Moreover, when methods are identified, they are typically tested on small-scale robotic arms in real-world settings to mitigate potential damages resulting from errors in the developed methodology and code. However, this approach suggests that simulating the system within a virtual environment could enhance the applicability of such methods and expedite the testing process. By leveraging simulation, researchers

can iterate more rapidly, accelerating the development and validation of methodologies while minimizing risks associated with real-world experimentation. This shift toward virtual simulation promises to streamline the advancement of robotics technologies and facilitate the translation of research into practical solutions.

This thesis endeavors to both replicate and build upon concepts introduced by previous researchers to handle the occlusion problem in a machine vision – robotic manipulator environment. To be more specific, a system is built from the ground up to simulate in a virtual environment the whole process of detecting the objects, selecting the correct one to be grasped and then simulating the whole procedure demonstrating a real-world scenario.

The implementation of this system is conducted within MATLAB-Unreal Engine environment. The methodology for training machine vision system with real world data is also illustrated with the same network being able to detect objects within the virtual setting. Additionally, the occlusion problem is handled by implementing Neural Networks and the grasping methodology is generated based on the occlusion detection and depth data. A simple but yet effective angle based grasping technique was used to grasp the objects detected.

In the realm of practical implementation, the necessary steps for calibrating and utilizing the Kinect V2 camera is outlined. This involves mapping the RGB image frame to the Depth frame using real-world data and polynomial regression. Finally, a tangible example is provided, demonstrating how this methodology can be effectively applied to a robotic arm equipped with a gripper and a Kinect V2 camera.

2. Related Work

Robotic arms have been around for decades, therefore most fundamental problems like the inverse kinematics or the trajectory to follow in restrained spaces, have been solved. In contrast, the field of machine vision, while benefiting from modern detectors, still faces substantial hurdles in achieving a level of sophistication comparable to human vision. Despite notable progress, the intricacies of interpreting visual data in diverse environments and adapting to dynamic scenarios remain areas where further developments are crucial. Bridging this gap in machine vision holds the key to unlocking enhanced robotic capabilities and expanding their applications in various sectors.

In 2013, Ramisa A. et al. [4] pioneered a model focused on garment part detection, particularly for items such as t-shirts. They extended the Bag-of-Words model, employing a sliding window technique to thoroughly analyze the entire image and extract features from each region. To enhance robustness, they incorporated rotation and scaling invariant techniques, effectively addressing variations in orientations and sizes. Notably, this approach bears similarities to the methods employed by CNNs for classification purposes.

Additionally, Răileanu S. et al. [12] developed an open-source machine vision platform tailored for manufacturing robotics. Their work involved integrating robotics, including grasping, with visual platforms in a user-friendly manner. This platform simplifies the implementation of basic machine vision algorithms, providing a comprehensive solution that bridges the gap between robotics and visual processing in manufacturing contexts.

In 2014, the introduction of R-CNN [7] marked a significant advancement in object detection, resulting in a notable 30% improvement over the prior best result on Pascal VOC 2012. This innovation enabled the detection of multiple objects within an image, reigniting interest in the integration of robotic manipulators with machine vision. Furthermore, the emergence of affordable depth sensors, such as the Microsoft Kinect in 2010 followed by Asus Xtion in 2011, prompted a surge in research focused on feature-based methods and point cloud data.

Aarth R. and Rishma G. explored the capabilities of Mask R-CNN in detecting waste objects on the ground. Their findings indicated superior performance compared to other networks, achieving an impressive accuracy of 97%. In a separate study, Zhen Li et al. [11] investigated the utilization of YOLO for object detection in a mobile robot equipped with a robotic arm and an RGB-D camera. They proposed a convolutional-based neural network for grasping, generating two "images" – one depicting grasp quality across the entire image and the other indicating the grasp angle. This example underscores the versatility of convolutional networks beyond classification, showcasing their efficacy in diverse tasks, such as object detection and robotic grasping.

Tuan-Tang Le et al. [8] used a similar method that is used in this diploma thesis where Mask RCNN was deployed to segment the objects for grasping, combined with a point pair feature voting approach to estimate the pose of the robotic arm. They showed that in real world scenarios the system can achieve a 90% accuracy. Ziyad Tareq N. conducted a comprehensive investigation into the efficacy of mask RCNN in detecting occlusion segments within heavily occluded environments. In his study, the network was trained using augmented images featuring objects, ensuring complete masks for each object. Notably, rather than distinguishing between masks for occluded and non-occluded objects, he opted to employ the full masks of all objects for network training. This approach aimed to assess the network's performance in handling occlusion scenarios without segregating mask information, achieving close to the same results if it had. To expand on that Yusuke Inagaki et al. [9] proposed a new dataset to evaluate the performance of Segmentation networks with respect to occlusion percentage.

Lastly Hongkun Tian et al [10] gathered all the data driven grasping methodologies for unknown objects, deviating from the typical analytical solutions or methods and assessed their performances on different datasets. This is substantially useful when the object detectors used has a substantial amount of object classes and there is no dataset for the grasping strategy of each one.

3. Methodology

The methodological framework pursued in this diploma thesis can be delineated into three distinct categories, each concurrently addressed to synergistically guide the progress of the others. A visual representation of this interwoven approach is presented in Figure 3-1, where a comprehensive flow chart illustrates the concurrent implementation of these methodologies. The concurrent nature of their development reflects the interdependence and mutual influence of each category in shaping the overarching methodology employed in this research.

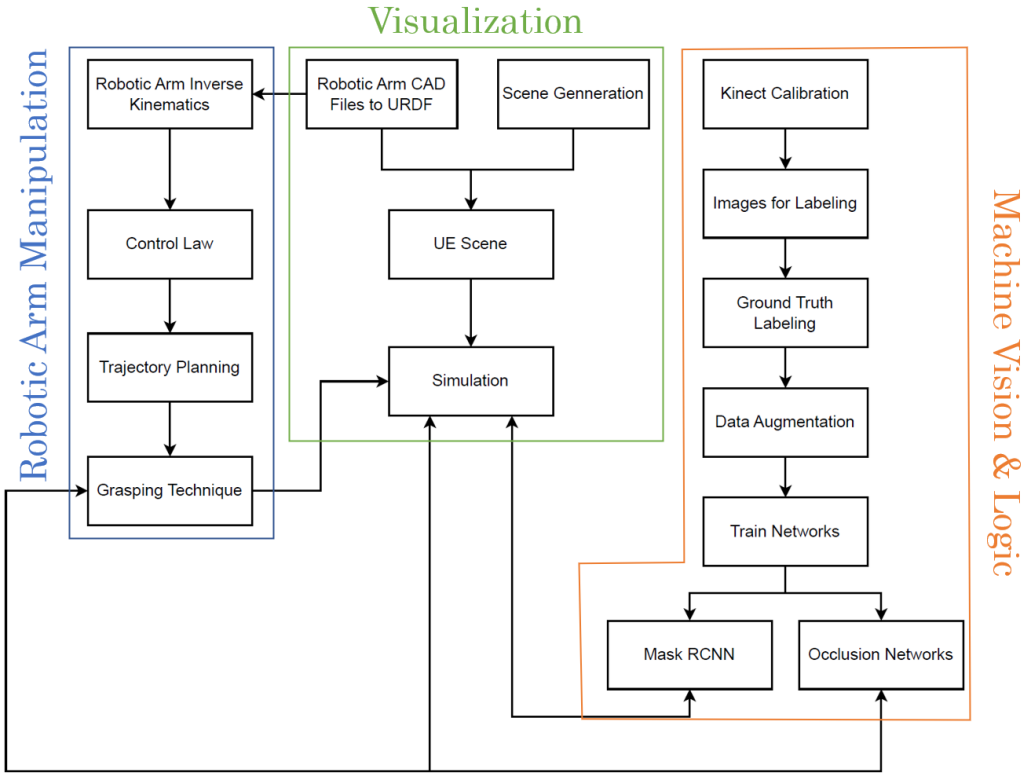


Figure 3-1: Methodology flow chart.

- **Robotic Arm CAD Files to URDF**

In SOLIDWORKS, assemble the robotic components only with concentric and coincident mates, ensuring accurate alignment. Subsequently, utilize an internal SOLIDWORKS add-on to export the assembled components as a URDF file.

- **Scene Generation**

In SOLIDWORKS, generate each component individually, and subsequently integrate them into a comprehensive assembly. Import this assembly into Blender to construct the simulation environment, carefully arranging the components within the scene. Following the scene creation, export the entirety as an FBX file format, facilitating its incorporation into the Unreal Engine environment for simulation purposes.

- **UE Scene**

Within the MATLAB-Simulink model, commence the scene setup using Simulation 3D building blocks. Integrate the robotic arm into the model by importing it as a URDF and establish a distinct transformation function to facilitate conversion between coordinate systems. Additionally, incorporate an RGB-D camera into the model, adjusting its parameters to align with those of a real-world camera.

- **Kinect Calibration**

To calibrate the RGB camera, employ a calibration paper and leverage the image calibration tool within MATLAB. Utilize this tool in conjunction with the calibration paper to ascertain and refine intrinsic camera parameters. Subsequently, perform the extrinsic calibration by capturing multiple images of known objects at various depths. This process involves associating RGB image frames with corresponding depth frames, establishing a mapping that facilitates accurate spatial referencing within the calibrated RGB-D camera system.

- **Images for labeling**

Configure the Kinect device to capture images of the parts from various orientations. Integrate lighting fixtures strategically to mitigate pronounced shadowing effects, thereby enhancing the quality and clarity of the captured images. Establish a connection between the Kinect device and a custom-written code designed to process the captured data and generate mapped images.

- **Ground Truth labeling**

Generate ground truth data for network training, specifically labels for Mask RCNN and occlusion Artificial Neural Networks, utilizing the MATLAB Image Labeler App. Subsequently, generate annotations and a datastore for training Mask R-CNN through the COCO API, ensuring compatibility and adherence to standard practices. Undertake preprocessing of labels and depth images to create input data for the occlusion ANNs.

- **Data augmentation**

Create augmented data and images featuring diverse backgrounds and item orientations derived from the original dataset, employing a custom-written code. This code is specifically designed to introduce variations in the visual context and object orientations, thereby enriching the training dataset for the Mask R-CNN network.

- **Train the Networks**

Employ stochastic gradient descent with momentum as the optimization algorithm for training the Mask RCNN network and resilient back propagation for training the occlusion ANNs, leveraging the computational capabilities of GPU

acceleration. This approach harnesses the parallel processing power of a GPU to expedite the training process, significantly reducing the computation time compared to traditional CPU-based implementations.

- **Robotic Arm Inverse Kinematics**

Generate the solution for inverse kinematics utilizing the analyticalinversekinematics() function within MATLAB, strategically selecting the appropriate joints that inherently align with the methodology's prerequisites.

- **Control Law**

Determine the end configuration for grasping by integrating machine vision data, camera information, and the inverse kinematics of the robotic arm.

- **Trajectory Planning**

Employ a polynomial trajectory within the joint space to expedite the implementation, capitalizing on the fact that precise positioning of the robotic arm during motion is not imperative. This strategic use of polynomial trajectories facilitates a rapid and efficient trajectory planning process. To determine the optimal end configuration, initiate the search from the initial configuration of the robotic arm joints, aiming to minimize the sum of the squared angle deltas required for the trajectory.

- **Grasping Technique**

Leverage occlusion ANNs in conjunction with depth data to formulate the grasping strategy for the robotic arm. Implement a strategy where each item is grasped from its shorter length while traversing through the center of area.

- **Simulation**

Integrate all the networks and generated data to simulate the entire process within the Simulink and Unreal Engine environment. Develop a comprehensive loop within the simulation framework to facilitate the robotic arm's grasping actions iteratively until there are no remaining items within the scene.

4. System Parameters

In the context of this diploma thesis, our primary focus will be on the utilization of the "Stäubli RX90L" robotic arm, complemented by a gripper developed in a preceding diploma thesis within the laboratory setting. This robotic arm is distinguished by its six joints, driven by brushless motors coupled to resolvers, thereby facilitating precise and versatile movements. The ensuing precision, speed, and limitations of each joint are meticulously detailed in Table 4-1. The repeatability of the system is notably specified as ±0.025mm under constant temperature conditions, underscoring its efficacy in repetitive tasks. Given the long-arm configuration denoted by the appended letter "L" in its name, the "Stäubli RX90L" robotic arm is characterized by a total mass of 113 kg. Notably, the load capacity of the end effector is constrained to 3.5 kg at nominal speeds and 6 kg at reduced speeds. The delineation of the robot's working area, elucidated as the reachability map, is visually represented in Figure 4-1 while an actual image of the robotic arm is shown in Figure 4-2.

Table 4-1: Joint parameters for the Stäubli RX90L robotic arm.

Joint	1	2	3	4 (1)	5	6
Amplitude (°)	320	275	285	540	225	540 (2)
Working range distribution (°)	A ± 160	B ± 137.5	C ± 142.5	D ± 270	E +120 -105	F ± 270
Nominal speed (°/s)	236	200	286	401	320	580
Maximum speed (°/s)	356	356	296	409	800	1125 (3)
Angular resolution ($°.10^{-3}$)	0.87	0.87	0.72	1	1.95	2.75

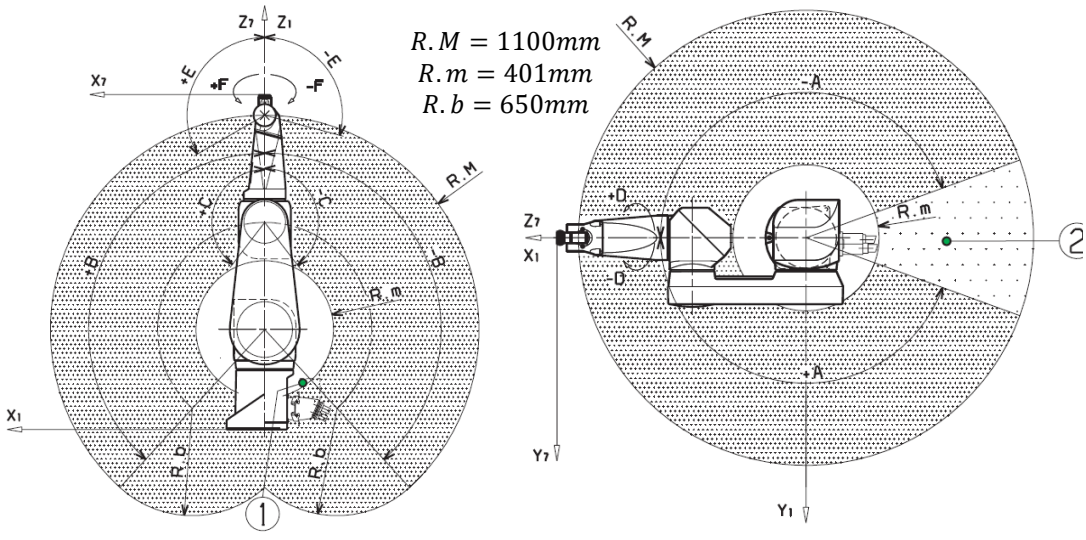


Figure 4-1: Dexterous workspace of the Stäubli RX 90L robotic arm.

Figure 4-2: Image of the Stäubli RX 90L robotic arm taken from the lab.

Moreover, an additional component, the "Kinect v2" as show in Figure 4-3, has been incorporated into the system to serve as an RGB-D camera. The low relative cost of this module combined with the high amount of information it provides, is of great significance in the machine vision system that was developed, but also in the grasping strategy by utilizing the RGB the Depth information respectively.

Figure 4-3: Image of the Kinect V2.

5. Robotics

5.1 Theory

5.1.1 Kinematics

In most robotics applications, the objective is to position the robot precisely relative to its base. The coordinates of a specific position are typically expressed in Cartesian space using a translation vector of size three (x, y, z). However, in robotics, defining the position alone does not fully characterize the end effector pose, an additional description for orientation is essential. Consequently, a 3x3 rotation matrix (as shown in Equation *5-1*) is employed for this purpose, requiring a minimum of three angles for complete specification.

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{21} \\ r_{31} & r_{31} & r_{33} \end{bmatrix} \qquad 5\text{-}1$$

Various methods exist for creating rotation matrices, and the selection depends on specific requirements and conventions within the given robotics application. One common and intuitive approach involves using Euler angles, such as 'roll, pitch, yaw' or 'ZYX', where three angles are specified as show in Figure 5-1. For each angle, a corresponding "2D" rotation matrix is associated, and by multiplying these three rotation matrices together, the overall rotation matrix is derived as shown in Equation *5-2* [16].

$$R_{ZYX} = R_Z(\alpha) \cdot R_{Y'}(\beta) \cdot R_{X''}(\gamma) \qquad 5\text{-}2$$

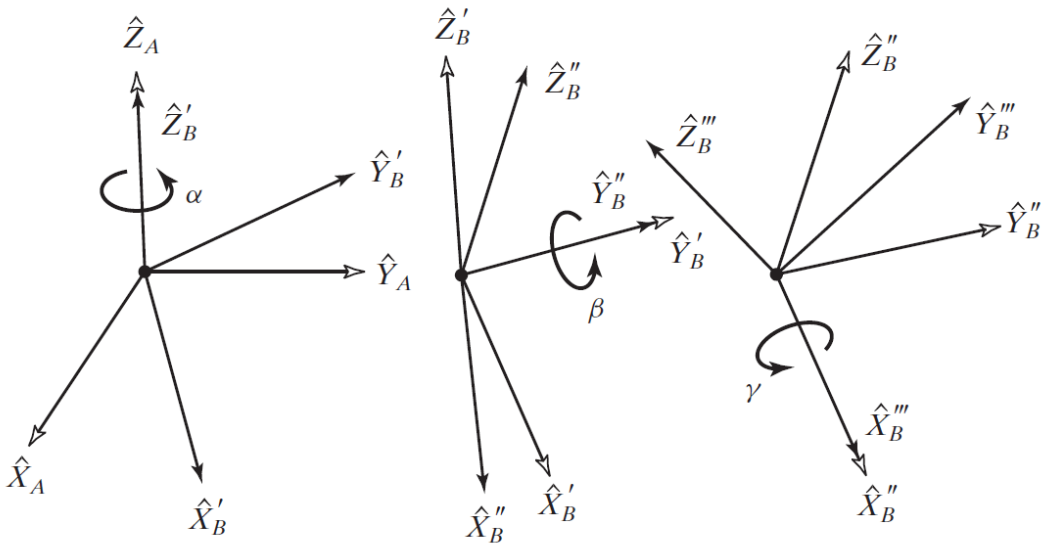


Figure 5-1: Euler Angle rotations in the form of Z-Y-X [16].

Despite the intuitiveness of Euler angles, they can encounter issues like gimbal lock (singularities). As a result, more often than not, four-parameter methods are employed to describe the end effector pose of robotic manipulators. Two prominent representations in this category are the Axis-Angle Representation and the Quaternion Representation. The former utilizes three angles to define a rotation axis and an additional angle to specify the rotation around that axis. The latter, although less intuitive, utilizes Euler parameters (ε_i) to describe the direction of the end pose that are considered more robust in mitigating the singularities. The definitions of those parameters are provided in Equation *5-3*.

$$R = (2 \cdot \varepsilon_4^2 - 1) \cdot I_3 + 2 \cdot \varepsilon_4 \cdot \boldsymbol{\varepsilon}^{\times} + 2 \cdot \boldsymbol{\varepsilon} \cdot \boldsymbol{\varepsilon}^T$$

Where:

First three Euler parameters: $\boldsymbol{\varepsilon} = \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \varepsilon_3 \end{bmatrix} = \widehat{\boldsymbol{k}} \cdot \sin\left(\frac{\theta}{2}\right) = \frac{1}{4 \cdot \varepsilon_4} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}$

Fourth Euler parameter: $\varepsilon_4 = \cos\left(\frac{\theta}{2}\right) = 0.5 \cdot \sqrt{1 + r_{11} + r_{22} + r_{33}}$ *5-3*

Skew Symmetric Matrix: $\varepsilon^{\times} = \begin{bmatrix} 0 & -\varepsilon_3 & \varepsilon_2 \\ \varepsilon_3 & 0 & -\varepsilon_1 \\ -\varepsilon_2 & \varepsilon_1 & 0 \end{bmatrix}$

The rotation matrix (R_{3x3}) and the translation vector (P_{3x1}) can derive fully the end effector position and orientation by using a homogeneous transformation matrix as described in Equation *5-4*.

$$T = \begin{bmatrix} R_{3x3} & P_{3x1} \\ 0_{1x3} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \textit{5-4}$$

However, moving the end effector to a specific position and orientation is not a straightforward task, it necessitates knowledge of the inverse kinematics of the robot or, at the very least, an approximation of it. This complexity arises from the fact that, in general, the controller of a given robotic system can only interpret angle rotations. Consequently, achieving a desired pose for the end effector requires specific rotations of all joints from the base up to that point. This is evident in the transformation matrix shown in Equation *5-5* concerning the end effector from the coordinate system of the base (0) up to it (e).

$${}^0T_e = {}^0T_1 \cdot {}^1T_2 \cdot \ldots \cdot {}^6T_e$$

Where:

Transformation matrix of coordinate system i with respect to coordinate system j: ${}^jT_i$ *5-5*

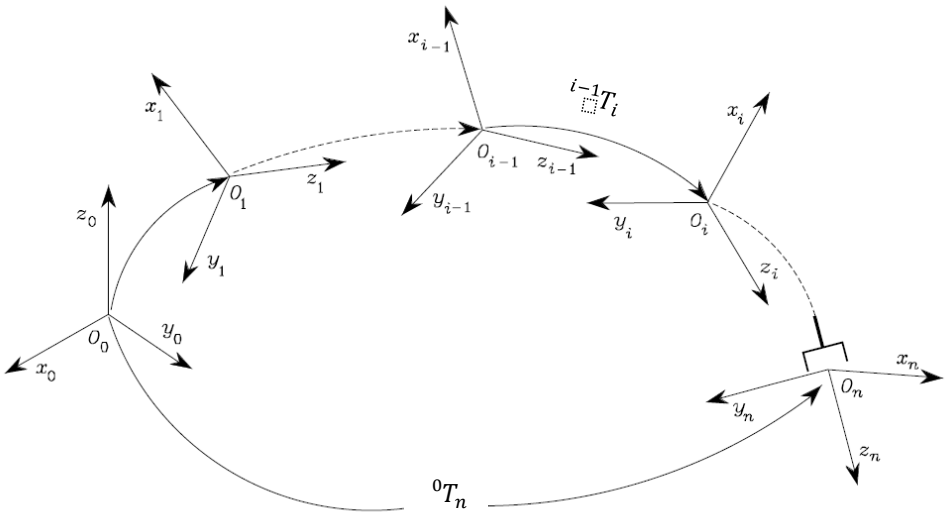


Figure 5-2: Coordinate transformation in an open kinematic chain [17].

Figure 5-2 shows an open kinematic chain, the different coordinate systems with in it and how they relate to each other using homogeneous transformations. The derivation of the transformation matrices comes from the mechanical drawing (the structure) of the robotic arm. More specifically how the joints are connected between the links of the robotic manipulator. By knowing those parameters, the homogeneous transformations (${}^{i-1}T_i$) between two consecutive joints ($i-1\ and\ i$) can be derived using various methods like the Denavit Hardenberg method explained in more detailed in [16][17]. It is noted that for rotational joints the transformation matrix is dependent on the angle of rotation of the joint ($T_i(\theta)$) where for a translational joint the transformation matrix is dependent on the translation of the joint ($T_\iota(d)$).

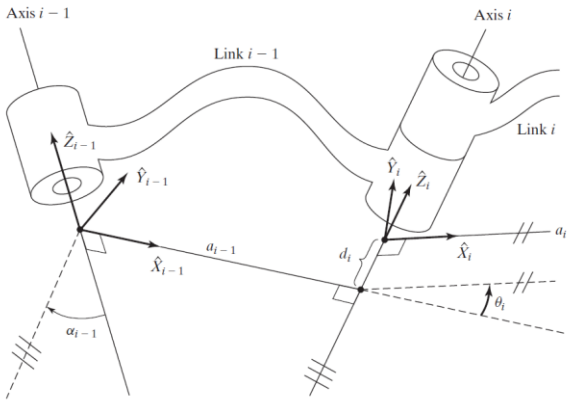


Figure 5-3: Joint frames (coordinate systems) between a link [16].

In the case of this diploma thesis, as the complete model exists in CAD files, the relations between joints as shown in Figure 5-3 (the homogeneous transformations) are automatically generated and saved in a ".urdf" file as it is described in more detail in the virtual environment implementation section.

From the aforementioned, it is derived that the objective to move the end effector to a desired position and orientation (${}^{0}T_{e,desired}$), requires a solution as show in Equation *5-6*.

$$ {}^{0}T_{e,desired} = X(q_1, q_2, \dots, q_n) \qquad 5\text{-}6 $$

Where:

Rotations or translation of the joints: q_i

The problem becomes even harder when there are joint limitations as shown in Table 4-1. Therefore, analytical solutions involve creating a solution tree for all possible configurations (where many solutions may exist for a specific configuration), while numerical methods, such as "BFGS Gradient Projection" or the "Levenberg-Marquardt" method, can be employed when an analytical solution is hard to find.

As of 2022, MATLAB has introduced a function named analyticalinversekinematics(), which the inputs are the rigid body tree of the manipulator and the desired base and end effector configuration. This function creates an object-function containing the analytical solution of the manipulator. It's crucial to specify the desired configuration, since the end effector can be defined as either the camera or the gripper center, allowing for separate control of their positions. The function is applicable to robotic arms with configurations consisting only of revolute joints, with the last three joints in a "wrist" configuration (perpendicular to each other) as shown in Figure 5-4. This is due to MATLAB using the Pieper method [18] where the problem can be split in two different three joint manipulators, which is much easier to solve analytically.

Fortunately, the robotic arm utilized in this diploma thesis, the Stäubli RX90L, conforms to the conditions required for the analyticalinversekinematics() function, enabling the calculation of its inverse kinematics with no additional inputs.

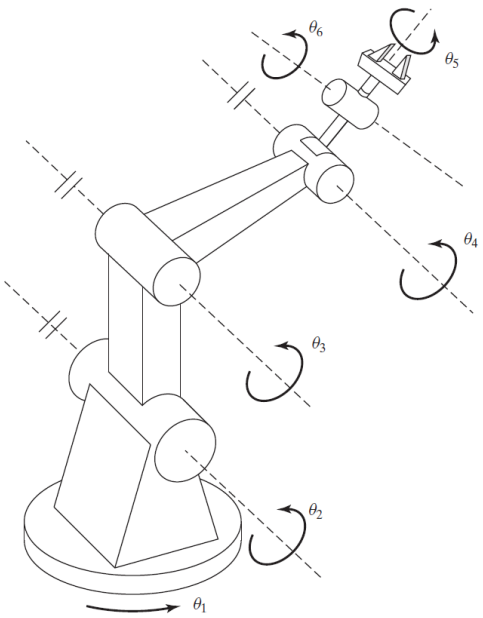


Figure 5-4: Robotic manipulator with a wrist configuration in the last three joints[16].

5.1.1 Image Based Visual Servoing

Image-based visual servoing (IBVS) is a control approach used in robotics, specifically in the field of visual servo control, where the control of a robot is based on the information extracted from images. In IBVS, cameras are used as sensors to provide visual feedback to control the robot's motion. The primary goal of image-based visual servoing is to manipulate the robot to achieve a desired visual goal or configuration, often specified in terms of features observed in the images.
In general, there are two subdivisions in a visual servoing system, the first one is the Image processing and feature recognition and the second one is the control law which moves the robot-the camera to the desired image frame. This diploma thesis handles both problems separately and joints them in the implementation process, meaning that in any case the control law or the feature recognition can be swapped for a different approach.

5.2 Implementation

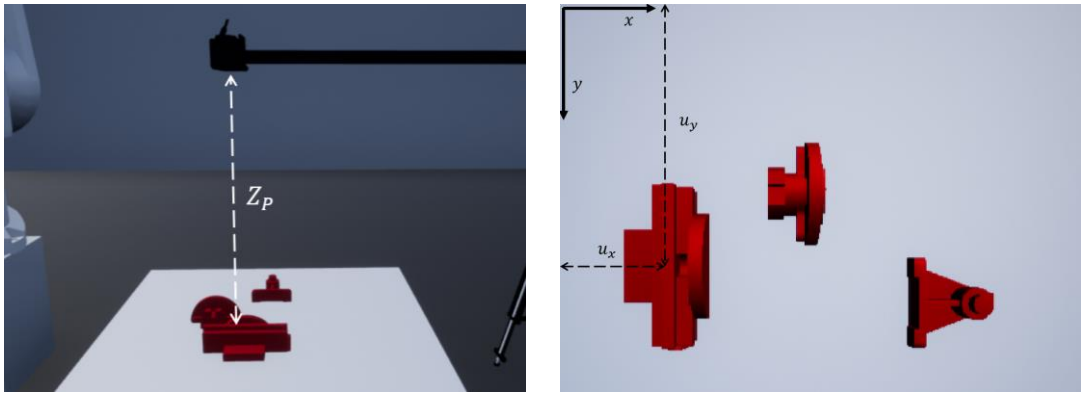
5.2.1 Robot Control Law

The implemented control law in this study leverages the features (points) derived from the integrated machine vision and grasping strategy. Ensuring precise depth readings from the camera is imperative for the effectiveness of the control system. This control scheme, integral to the thesis, is intricately woven around the foundational framework of the robot's inverse kinematics. To elaborate the methodology is initiated by calculating the translation of a point in the image frame into real-world coordinates by using Equation *5-7* [19].

$$X_P = \frac{u_x - c_x}{f_x} \cdot Z_P$$
$$Y_P = \frac{u_y - c_y}{f_y} \cdot Z_P \qquad 5\text{-}7$$

Where:
u_x, u_y: Are the x and y point coordinates in the image frame in pixels.
c_x, c_y: Are the camera centroid x and y position in pixels.
f_x, f_y: Are the focal lengths of the sensor in millimeters
X_P, Y_P, Z_P: Are the real world coordinates of the point observed by the camera relative to the camera frame.

Incorporating a depth sensor facilitates the assessment of the distance from the camera to the observed point, consequently establishing the value of Z_P. Thus, the coordinates X_P and Y_P can be evaluated, providing the position of the observed point. Figure 5-5 shows an example of how the coordinates of an object are evaluated.



(a) (b)

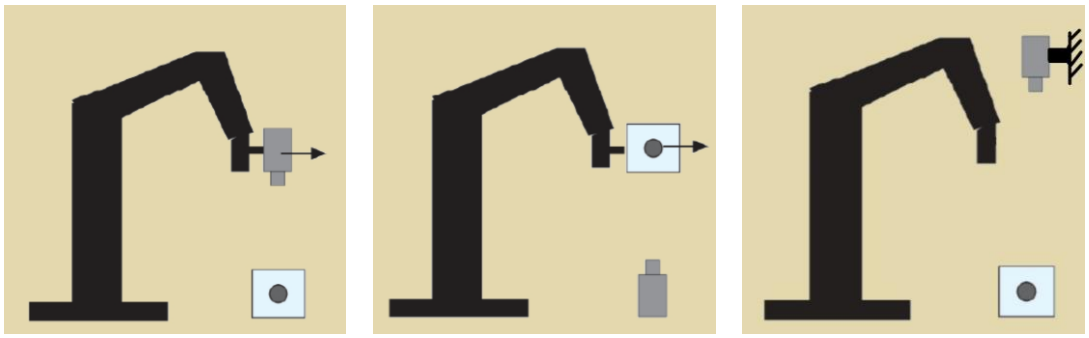
Figure 5-5:Example of coordinates, of distance from the camera Z_P(a) and object location in image frame (b).

Therefore, a simple method can be implemented to create the homogeneous transformation between the object to be grasped and the camera $\left({}^{c}T_{obj}\right)$, by using the coordinates vector $[X_P, Y_P, Z_P]$ and a predefined orientation (eg. Vertical from the ground plane, looking down). and use the inverse kinematics to move the robot to that exact pose. Then the transformation matrix of the object $\left({}^{b}T_{obj}\right)$ can be computed if the robotic arm – camera transformation is known, using Equation *5-8*. Thus, by using the analytical inverse kinematics, the configuration of the robotic arm can be evaluated.

$$ {}^{b}T_{obj} = {}^{b}T_{c} \cdot {}^{c}T_{obj} \qquad 5\text{-}8 $$

Where:
The homogeneous transformation between the base of the robotic arm and the camera relative to the base frame: ${}^{b}T_{c}$

The methodology employed in this diploma thesis is chosen for its efficacy and simplicity. Notably, this approach accommodates the generation of a trajectory both when the camera is affixed to the robotic arm (Eye in hand system) and when it is positioned statically above the robot's operational workspace (separate eye and hand system). The sole divergence in the method's implementation pertains to the transformation matrix of the camera with respect to the manipulator's base. In this thesis, the adoption of a stationary camera was favored for its practical ease of implementation in real-world scenarios. Figure 5-6 shows the differences between three different camera-robotic arm configurations.



(a) (b) (c)

Figure 5-6: Eye-in-hand system (a), Eye-to-hand system (b), Separate eye and hand system (c).

Importantly, the stationary configuration ensures the universality of applicability across diverse robotic arms, particularly those lacking specialized mounts for camera attachment. It is noted that for the implementation to be possible the camera has to move away from the working area of the robot (Position 2) as depicted in Figure 5-8. However, the term stationary still applies because every time a picture is required, the camera moves to the capturing position (Position 1) as depicted in Figure 5-7.

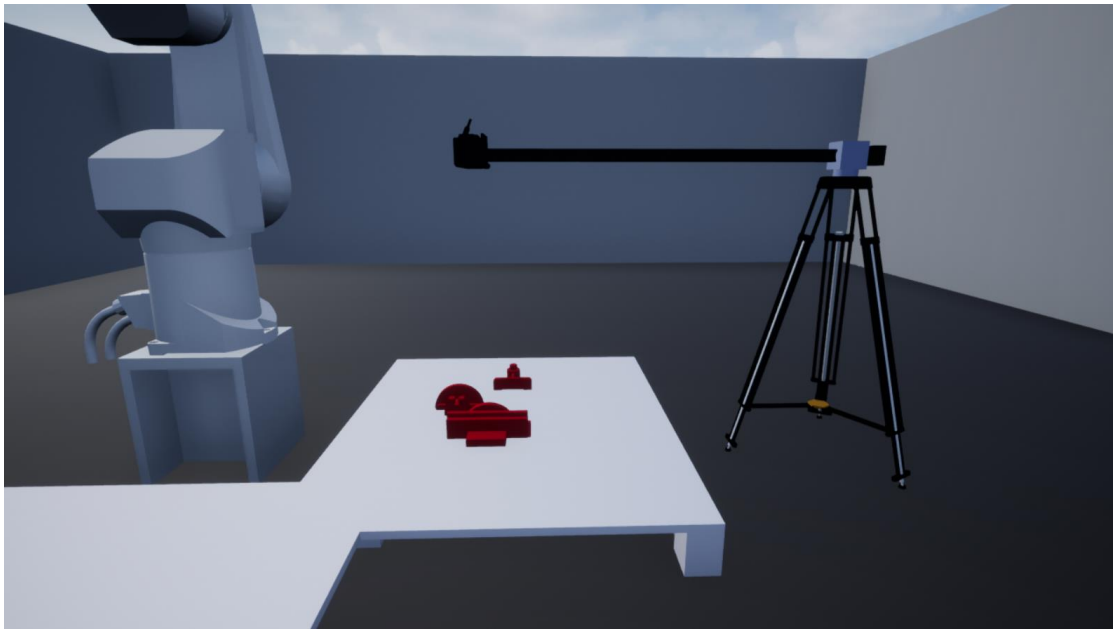


Figure 5-7: Camera position right above the table (Position 1).

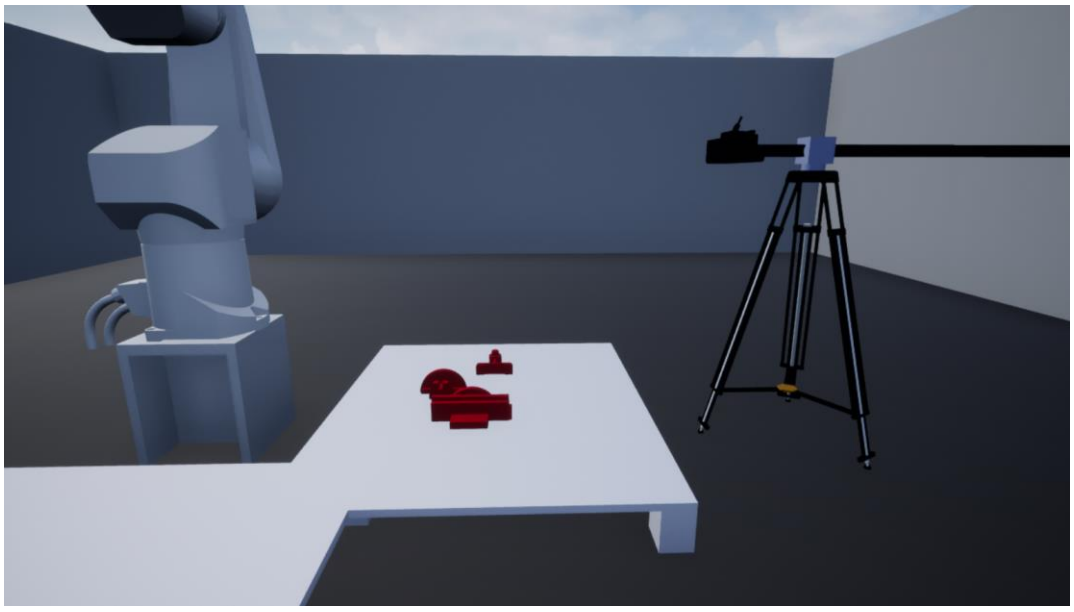


Figure 5-8: Camera position away from the table (Position 2).

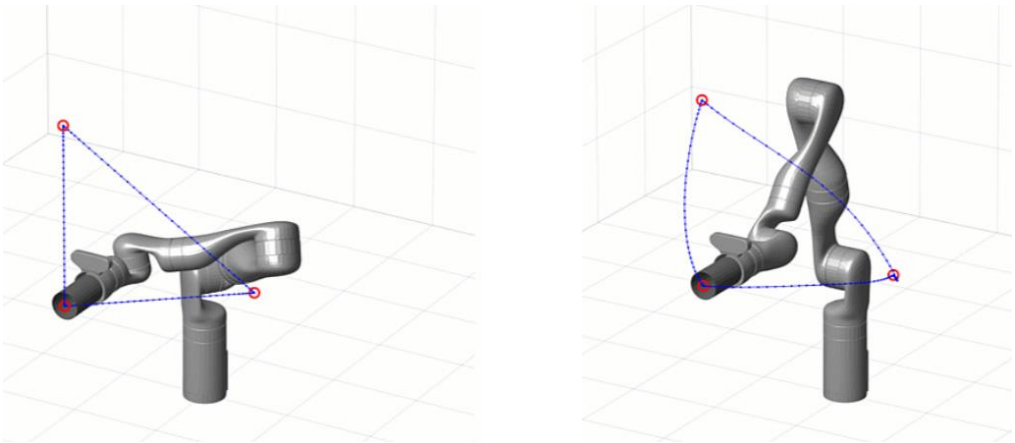
It is imperative to underscore that while the adoption of this method is characterized by its ease of implementation, a meticulous understanding of the transformation matrix of the camera in relation to the manipulator's base is requisite. This necessitates employing an eye-to-hand system and obtaining measurements from

known objects in conjunction with the manipulator's orientation. Notably, this approach demands a precise calibration process to ascertain accurate results in real world scenarios. Additionally, it is pertinent to acknowledge the existence of alternative visual servoing methods that circumvent reliance on inverse kinematics, opting instead for the utilization of the manipulator's Jacobian matrix. [19].

5.2.2 Trajectory Control

The trajectory of a robotic arm is a fundamental aspect of its motion planning, defining the path and sequence of positions that the end-effector traverses over time. In the realm of robotics, crafting an effective trajectory is crucial for achieving precise and efficient manipulation of objects in diverse applications, ranging from industrial automation to research and development. The trajectory planning process involves the careful consideration of factors such as kinematics, dynamics, and workspace constraints, with the goal of optimizing the movement of the robotic arm to fulfill specific tasks.

Robotic arms exhibit two primary types of trajectories: Cartesian trajectory control and joint trajectory control. In Cartesian trajectory control, the emphasis is on regulating the end-effector's motion through specified positions and orientations in the Cartesian space. This approach provides a more intuitive means of commanding the robot, particularly in applications where precise positioning in a global coordinate system is critical. On the other hand, joint trajectory control focuses on managing the movement of individual joints to attain desired positions. This method allows for a more direct influence over the robot's configuration, often proving advantageous in tasks that demand intricate joint-level coordination. It is also noted that in joint trajectory control, there are no possibilities of falling into gimbal lock configurations. The difference between those two types of control is show in Figure 5-9.



(a) *(b)*

Figure 5-9: Cartesian trajectory control (a) and Joint trajectory control (b).

In this dissertation, joint space control is adopted, primarily driven by the absence of a stringent requirement for precise end-effector positioning throughout the robotic arm's motion. The adoption of joint space control offers notable advantages, including

reduced computational demands, as it necessitates the calculation of inverse kinematics solely for the terminal solution rather than the entire trajectory. Furthermore, this approach enables a reduction in the overall joint rotation time. This efficiency arises from the fact that joints only need to traverse the angular delta from the initial to the final configuration, minimizing the rotational distance covered. In contrast, cartesian trajectory control may result in joint overshooting or deviations from predefined speed profiles during the robotic arm's motion. Despite such nuances, both control approaches are ultimately directed toward reaching the same endpoint, potentially incurring comparable total joint angle rotations under optimal conditions.

Within the context of path planning, an essential consideration arises from the fact that numerous potential solutions often exist for a given end effector configuration as shown in Figure 5-10. The selection among these alternative solutions is of paramount significance, as it influences the operational speed and the energy efficiency of the robotic system. In this thesis, a trajectory prioritizing speed is adopted, with preference given to the end configuration characterized by the minimal angular deltas. Equation Figure 5-8 shows the loss function to be minimized which is known as the “Summed Squared Error” function.

$$min\{L_j\} = \min\{\sum_{i=1}^{6}\left(q_{i,end} - q_{i,init}\right)^2\} \quad 5\text{-}9$$

Where:

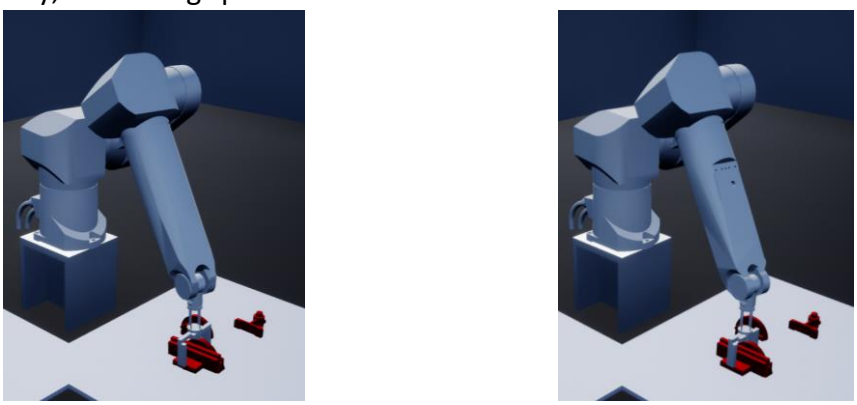
$q_{i,end}$: Is the end configuration joint angle.

$q_{i,init}$: Is the initial joint angle.

i: Denotes the i^{th} joint.

j: Denotes the j^{th} end configuration

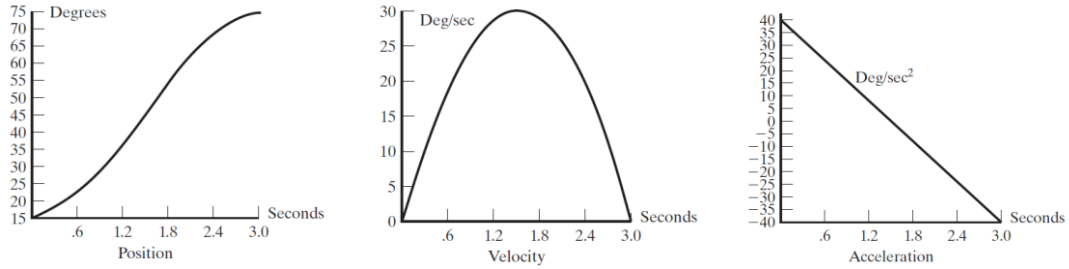
This selection process is automated through a dedicated function, which systematically calculates the angular deltas for various configurations and automatically identifies the configuration which minimizes the loss function. The meticulous choice of a speed-oriented trajectory underscores the commitment to optimizing the robotic arm's performance by minimizing angular deviations and, consequently, enhancing speed.



(a) *(b)*

Figure 5-10: Same end effector position and orientation with two configurations.

In this thesis, third-degree polynomials were utilized in the control strategy employed for the articulation of robotic arm joints. The decision to incorporate third-degree polynomials is rooted in their inherent capacity to facilitate smooth and continuous joint motion. By leveraging cubic polynomial functions with time as the parameter, the trajectories are adeptly defined, allowing for control over the joints. The four essential parameters—initial position, initial angular velocity, final position, and final angular velocity—shape the trajectory, enabling a tailored approach to achieve specific movement requirements. In all cases the angular velocity was set to 0 in all the joints for every single arm configuration to avoid any overshooting's. The mathematical model of those trajectories is described in Figure 5-11.



(a) *(b)* *(c)*

Figure 5-11: Example of polynomial trajectory of third degree, position (a), velocity (b), acceleration (c).

6. Machine Vision System

6.1 RGB-D Camera Calibration

As previously stated, image capture for object detection involved the use of the Kinect V2, an RGB-D camera. The RGB sensor in the Kinect V2 can capture images with a resolution of up to [1920x1080 pixels], while the separate Depth sensor has a resolution of [512x424 pixels][1]. The disparity in resolution, sensor positioning, and aspect ratio presented a challenge in mapping both sensors to a single image frame. Figure 6-1 shows a visual example of this disparity.

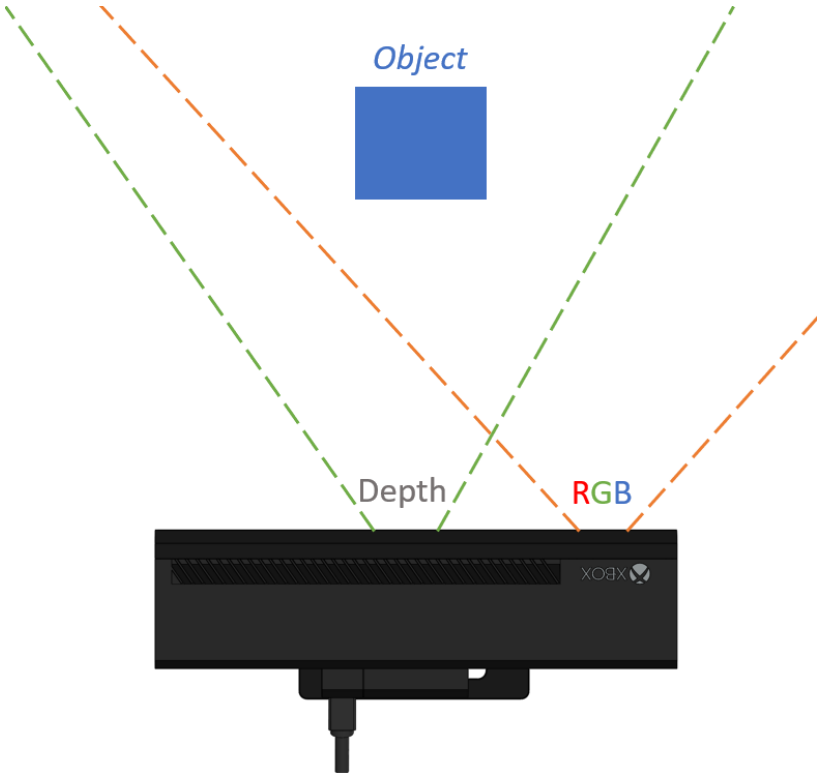


Figure 6-1: Example (not to actual scale) of sensors misalignment on the Kinect V2.

Consequently, a calibration procedure was executed to align the RGB image with the depth image. It's crucial to highlight that this mapping approach was chosen because the depth sensor has a lower resolution, which acts as a limiting factor. Mapping the depth sensor on the RGB sensor's image frame would have introduced errors in the calculated depth, requiring an additional calibration step.

The calibration started by calculating the pixel size – focal length of each sensor. Firstly, for the RGB sensor the focal length was calculated using MATLAB automatic

[1] In this thesis the phrases “depth pixels” and “depth image” are implemented rather than the most common term in the bibliography “point cloud”. This is done intentionally due to the way the depth data are handle, mimicking the way the image data are handled.

calibration tool, where 15 photos of a calibration paper were taken from different angles that were fed through the calibration tool.

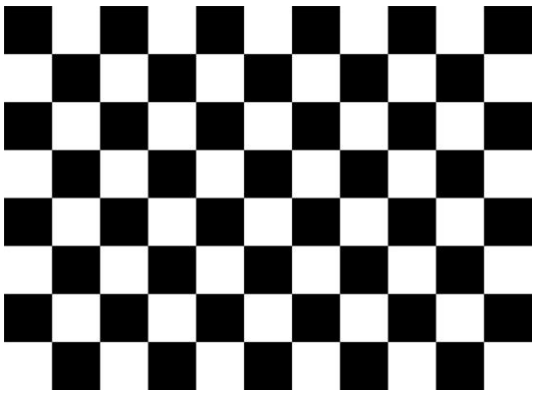
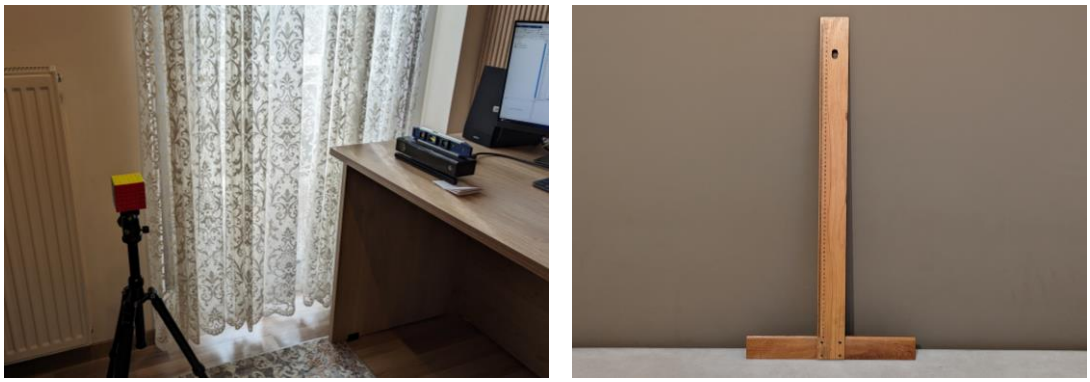


Figure 6-2: Calibration pattern with squares of size 25x25mm.

Another important parameter to determine was the distance of the sensor center from the center of the Kinect in the x direction. This measurement was accomplished by placing an object in a tripod with a mark on its center with a known and predetermined distance ($Z = 72cm$) from the kinect. This object was a Rubik's cube 6x6 where there are lines passing through its center, the tripod was leveled as well as the Kinect with a leveling apparatus, the tripod was raised to bring the center of the cube as close to the center of the Kinect in the y direction and the tripod was moved to the center of the Kinect in the x direction using a marked right angle ruler (length 72cm) ensuring that the Kinect and the face of the cube are parallel as shown in Figure 6-3. Then by taking a photo and tracing the center of the Rubik's cube, the physical distance of the sensor to the center of the Kinect can be evaluated using the formula shown in Equation *6-1*.

$$X = \frac{(u_x - c_x) \cdot Z}{f_x} \qquad 6\text{-}1$$

Where
Distance of the point in the x direction: $X[m]$
Focal length in the x direction: $f_x\ [m]$
Distance from sensor to the point: $Z\ [m]$
Distance from center of the image: $u_x - c_x\ [pixels]$



(a) *(b)*

Figure 6-3: Calibration setup (a) and right-angle ruler (b).

The computed distance aligns precisely with the distance derived from the CAD measurements of the Kinect, specifically X=95mm. Subsequently, the pixel ratio between the depth and RGB sensors was determined by selecting two points representing the identical physical position from images captured by both sensors. By calculating the pixel distance in both images and utilizing the Equation *6-2*, the pixel aspect ratio was computed equal to $PAR = 3.05$. Noteworthy is the observation that the aspect ratio was calculated independently for both the x and y axes, and the results indicated equal values. This equivalence implies that both sensors exhibit square pixels in their respective imaging systems.

$$PAR = \frac{pixel\ distance\ from\ RGB\ image}{pixel\ distance\ from\ Depth\ image} \qquad 6\text{-}2$$

Then the mapping of RGB image frame to the depth image frame ensued by taking photos from different 'Z' distances of a specific object inside an operating range of [500,1400] mm of depth and calculating the distance between the same point in the two image frames. In this case the object was a Rubik's cube 6x6. This operating range was chosen because in the physical world the camera will not be placed further or closer to the table where the items will lie on, than those bounding values. Table 6-1 shows the calculated delta's where the in general dy is constant with a value of $dy = -15$ (depth image) depth pixels which means that the RGB image needs to be translated 15 depth pixels upwards. In contrast dx was not constant, therefore polynomial regression of second order was generated from those points, the result of which is shown in Equation *6-3*.

Table 6-1: Delta's in x and y direction between RGB and Depth sensor of the Kinect V2.

Depth Value [mm]	dx [pixels]	dy [pixels]
543	25	-15
561	24	-15
622	20	-15
764	15	-15
855	13	-14
949	12	-15
1036	10	-15
1355	6	-15

$$x_{new} = x_{old} - 5 \cdot 10^{-5} \cdot d^2 + 0.1078 \cdot d + 68.46 \qquad 6\text{-}3$$

Where the depth value in [mm] at the specified position (x_{old}, y): d

Consequently, code was written to map the pixels according to their depth. Then the images are cropped by 25 pixels from the left side to avoid areas of where there are no depth values. Therefore, the resultant resolution of the images is [417,340] (width, height). A significant factor to note is that the RGB image can be mapped to the depth image with a resolution of [1273,1038] where each depth pixel is equal to 3.05 RGB

pixels. This mapping technique utilizes the high resolution of the RGB sensor which was eventually needed for increasing the performance of training the Mask RCNN network.

Figure 6-4 and Figure 6-5 show the initial and the final images. For the depth images a custom function was created to convert the depth data (uint16) to grayscale (uint8), thus the difference in brightness is due to the conversion function and not to actual change in depth values.

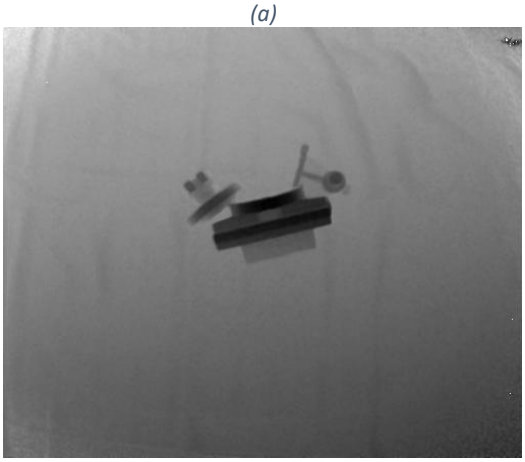
Table 6-2: RGB camera parameters

Resolution	Principal Point	Focal Length	Radial Distortion[2]
$[1920,1080]_{w,h}$	$[945,557]$	$[1053,1053]_{x,y}$	$K_1 = 0.04$

Table 6-3: Depth camera parameters.

Resolution	Principal Point	Focal Length	Radial Distortion
$[512,424]_{w,h}$	[256,212]	$[365,365]_{x,y}$	$K_1 = 0.2$

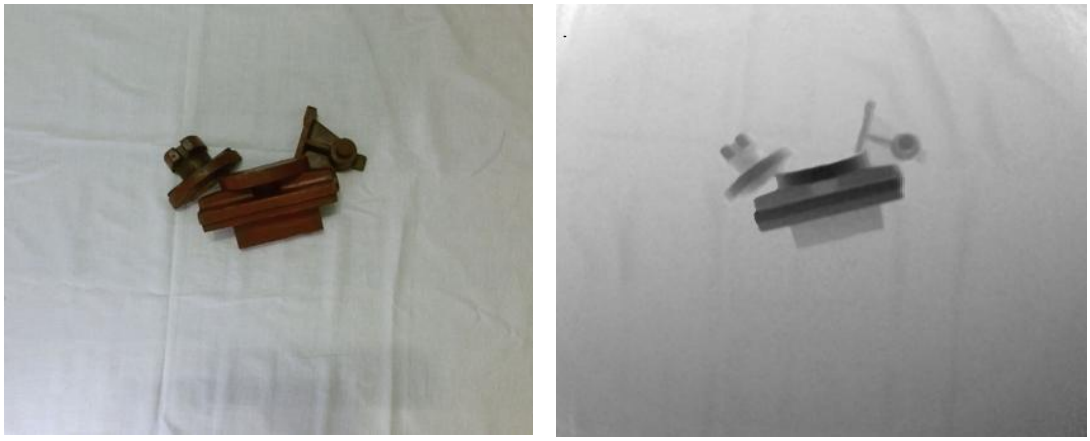
(a)



(b)

Figure 6-4:Initial Images, RGB image (a) and point cloud visualization (b).

[2] Parameter of Brown-Conrady model as described in: https://en.wikipedia.org/wiki/Distortion_(optics) . All other parameters are set to 0 (no tangential distortion).



(a) (b)

Figure 6-5:Mapped Images, RGB image (a) and point cloud visualization (b).

6.2 Convolutional Neural Networks Theory

6.2.1 Image Classifier

A CNN is a specialized type of artificial neural network designed for processing and analyzing visual data. CNNs have become a fundamental technology in computer vision applications, excelling at tasks such as image classification, object detection, and image segmentation. They are inspired by the visual processing capabilities of the human brain and are particularly effective in capturing spatial hierarchies of features. A schematic representation of a simple CNN is shown in Figure 6-6.

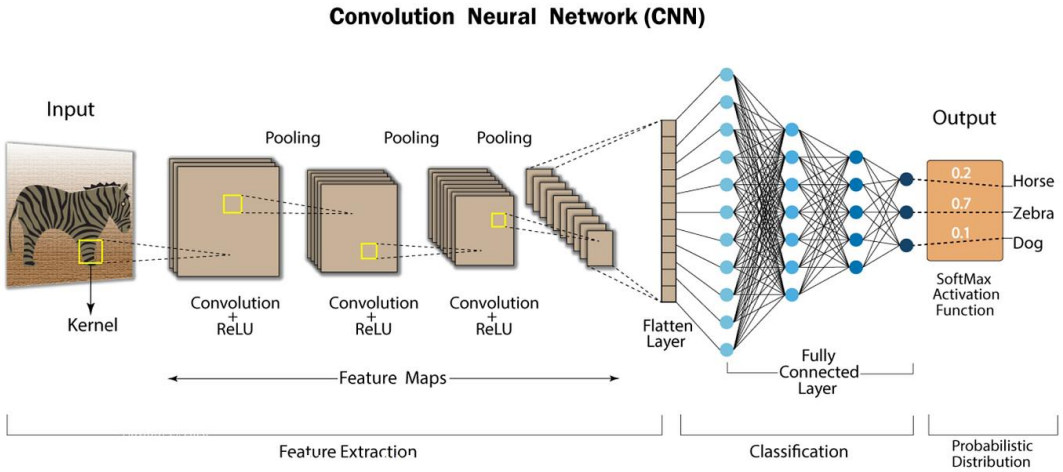


Figure 6-6: Schematic representation of a CNN.

As the name suggests the main feature of a convolutional neural networks are the convolutional layers. A CNN has to contain at least one convolutional layer to be called a CNN. These layers apply convolution operations to the input data, allowing the network to automatically learn spatial hierarchies of features by changing the weights of those filters. Convolution involves sliding a small filter (also known as a kernel) across the input data to extract local patterns. A simple example of a convolution operation is show in Figure 6-7.

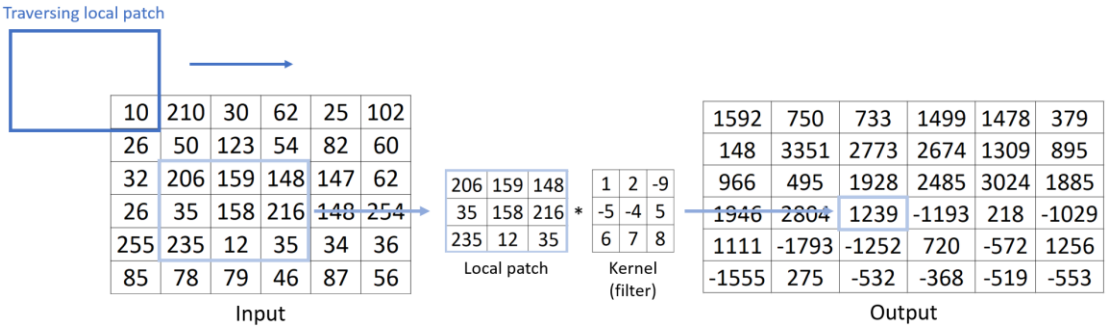


Figure 6-7: Example of a convolution with a kernel size of 3x3.

In case of an RGB image convolution occurs for every different channel of the image where the "kernel"- filter is a 3 dimensional matrix that applies convolution for every channel, outputting a two dimensional matrix (feature map) as show in Figure 6-8.

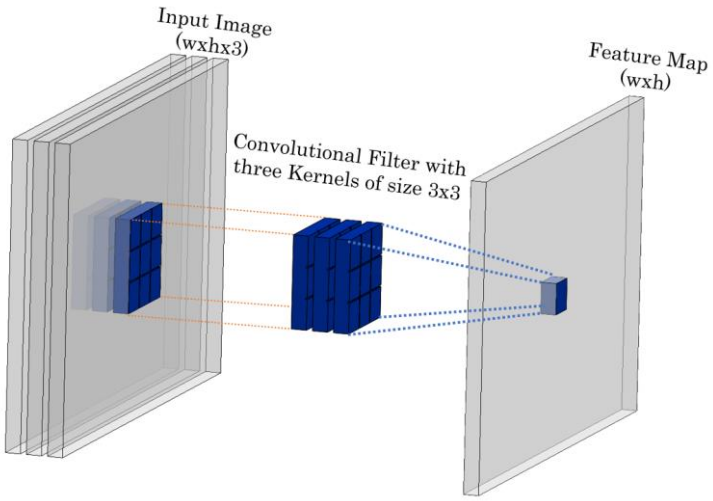


Figure 6-8: Example of convolutional filter applied to an Image.

CNN's expand on that idea to extract even more features, they use more than one filter (matrix of 2D kernels) for each set of channels, generating additional feature maps. It is noted that the weights of these filters are learned during the training process to extract the desired features. The results of the convolution operation are passed through an activation function. The activation function is applied elementwise to the output of the convolutional layer as shown in Figure 6-10. The most common activation function is the ReLU function or used more recently Leaky ReLU. The mathematical model of those functions is depicted in Equations *6-4* & *6-5*.

$$ReLU: f(x) = \max(0, x) \qquad 6\text{-}4$$

$$Leaky\ ReLU: f(x) = \max(ax, x)\,, a \leq 0.01 \qquad 6\text{-}5$$

In simple terms, if the input to a ReLU neuron is positive, it outputs the same value, if the input is negative, it outputs zero. The idea behind this step, is to introduce non linearities to the model, which in turns enables the network to learn complex relationships in the data. The main advantage of ReLU is its simplicity and effectiveness in training deep neural networks. It also helps address the vanishing

gradient problem, where gradients become very small during backpropagation, by allowing the flow of information for positive inputs. Therefore, after the activation map a pooling layer follows. The main purpose of the pooling layer is to down sample the spatial dimensions of the input volume, reducing the computational complexity of the network. Max pooling is a common pooling operation, where the maximum value in a local region is retained, helping to preserve the most important features while discarding less relevant information. It must be noted that in this way some information is always lost. Figure 6-9 depicts a simple example of applying max pooling to a 2D feature map.

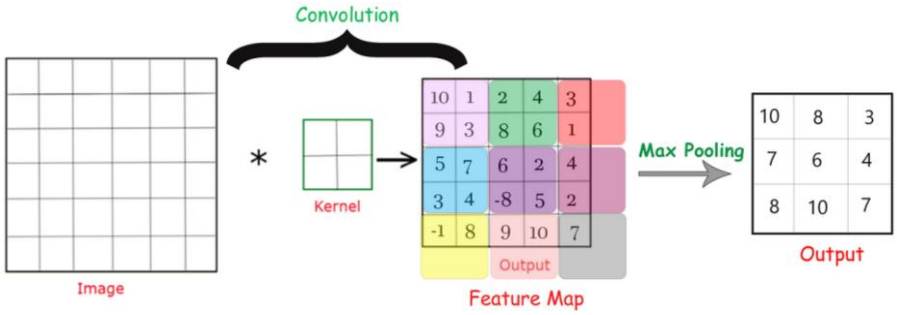


Figure 6-9: Example of a convolution and a max pooling layer with a kernel size of 2x2.

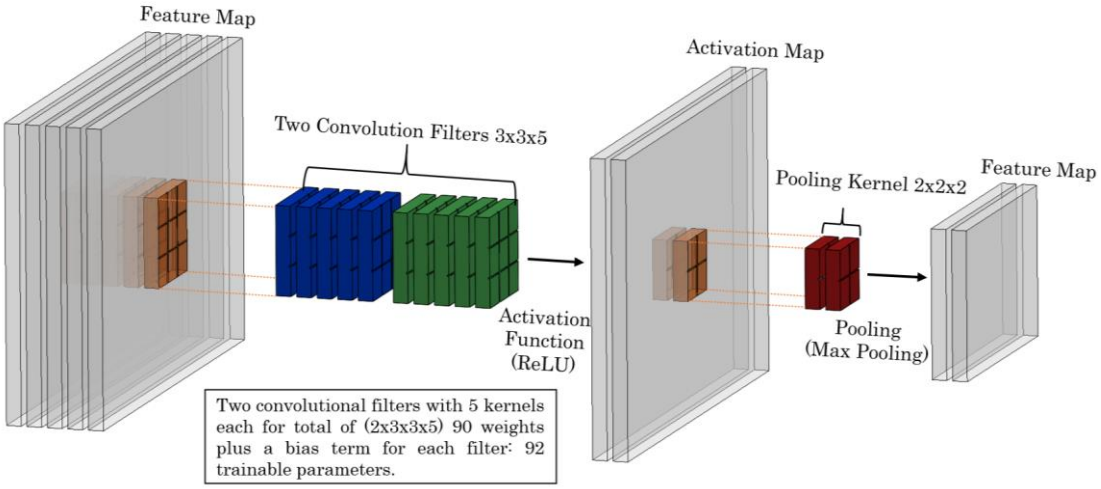


Figure 6-10: Example of a convolutional block.

All the above steps compose a convolutional block as shown in Figure 6-10. By connecting many of those convolutional blocks together the feature extractor or the "Backbone" of the network is created as shown in Figure 6-6. For the classification to be possible a classifier or a "Head" must be added. The head consist of fully connected layers, that are able to combine high-level features learned by the backbone. These layers connect every neuron in one layer to every neuron in the next layer, allowing the network to make predictions or classifications based on the learned features. Lastly the results of the classifier are passed through a SoftMax function to convert the results of the network to probabilities for each class. Figure 6-11 shows a simple example of the SoftMax function in use.

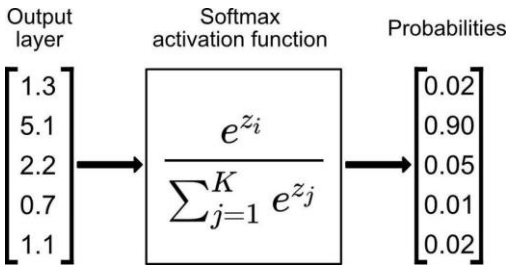


Figure 6-11: SoftMax Function.

The training process of the Convolutional Neural Network (CNN) involves a form of backpropagation akin to traditional neural networks. In this iterative process, an image is passed through the network, and the output is observed in the last layer. A loss function then quantifies the disparity between the model's predictions and the actual target values. Subsequently, a backward pass is initiated to calculate the gradient of the loss with respect to each parameter in the network. These gradients serve as guides for updating the network's parameters through optimization algorithms, such as the stochastic gradient descent with momentum (SGDM) [1] that is depicted in Equation 6-6.

$$u_{i+1} = u_i - a\nabla L(u_i) + \gamma(u_i - u_{i-1}) \quad 6\text{-}6$$

Where:
Updated value of the Parameter: u_{i+1}
Previous value of the Parameter: u_i
Gradient of the Loss Function: ∇L
Momentum value: γ
Learning rate: a

The objective is to iteratively adjust the parameters in the opposite direction of the gradient to minimize the loss. A typical loss function used in those type of networks is the Categorical Cross-Entropy loss function that is described in Equation 6-7.

$$L = -\sum_{i}^{n} y_i \cdot \log_2 \hat{y}_i \quad 6\text{-}7$$

Where:
Loss Function: L
The true probability (ground truth) of class i: y_i
The predicted probability of class i from the CNN: $\hat{y}_i$
The number of classes: n

Key parameters subject to modification during training include the convolutional filters (kernels), where the weight of each filter is adjusted. Additionally, the weights and biases of the fully connected layers are adjusted. It is worth noting that while this encapsulates the fundamental premise of the model, more advanced architectures may involve the adjustment of additional parameters during the backpropagation process.

6.2.2 Object Recognition

A CNN with the features as described above performs feature recognition to the whole image thus outputting in the last layer (of the fully connected layers) the probability of an object to exist in that image. However, it is not specified where the objects might be located within the image. For this reason, more advanced methods where created, building upon the existing CNNs. The additional capability is prediction of the position of each item in a scene (object detection) by using labeled bounding boxes (rectangles). Each rectangle is denoted with a specific label for the item contained inside it as shown in Figure 6-12.

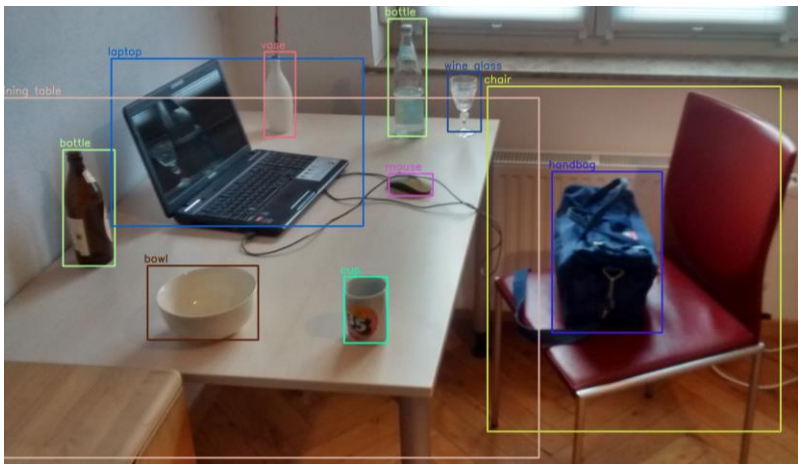


Figure 6-12: Example of image object segmentation and labeling with bounding boxes.

These networks, notably exemplified by models such as YOLO v4, RCNN, Fast RCNN, and Faster RCNN, are referred to as object detectors, distinguished by their enhanced capabilities. In this section focuses more on the implementation of the Faster RCNN network as the network used in this thesis (Mask RCNN) builds upon it. The typical architecture of these networks includes a backbone network, often based on a pre-trained CNN (such as VGG16 or ResNet in the case of Faster RCNN). This backbone network is employed to extract features from the input image as it was explained in the previous section. Those features form the foundation for subsequent region proposal and object detection stages. Figure 6-13 shows a schematic representation of the Fast RCNN network.

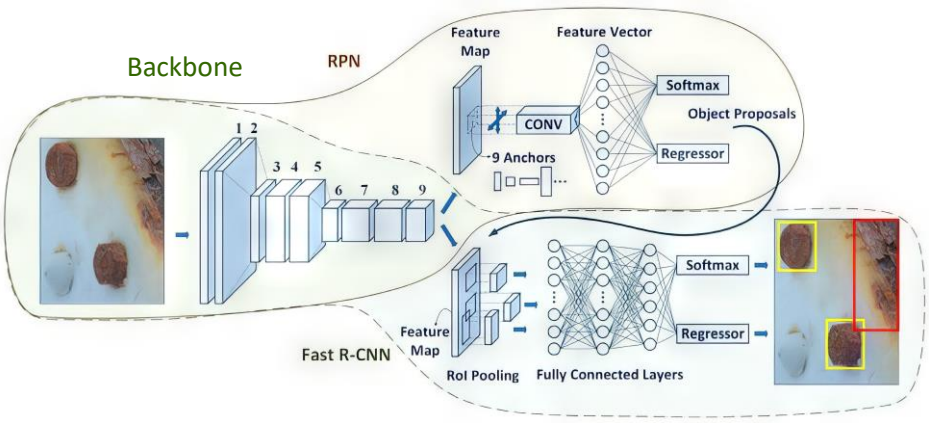


Figure 6-13: Faster RCNN schematic representation.

The process initiates with the passage of features through a Region Proposal Network (RPN), responsible for generating region proposals (bounding boxes) indicating potential object locations. Essentially, the RPN guides the classifier on where to focus its attention within the image. These proposals are generated by employing two compact networks that "slide" across the feature maps produced by the Backbone. These networks are fully connected with a sliding window (kernel) of size n×n (where in the case of Faster RCNN n=3). At each position of the sliding window k anchors boxes (reference boxes) of different sizes and aspect ratios are used for proposing multiple bounding box regions of objects (for example in Faster RCNN 3 different aspect ratios of anchor boxes were used with 3 different sizes totaling k=9 anchor boxes). It is noted that the anchor boxes are constant and are set by the user in the creation of the CNN. The networks connected to the sliding window is the box classifier (cls) that has an output of 2·k representing the probabilities of an item to be located inside the anchor box or not and the box regressor (reg) that has an output of 4·k that represent the adjusted coordinates of the anchor boxes [2]. Subsequently, the RoI Pooling layer or RoI Align layer takes these region proposals from the RPN and extracts fixed-size feature maps from the features produced by the backbone network by using a max pooling operation of varying size. This crucial step ensures that subsequent layers receive inputs of uniform size that is needed for the fully connected layers. Figure 6-14 shows a schematic representation of how the RPN network is structured.

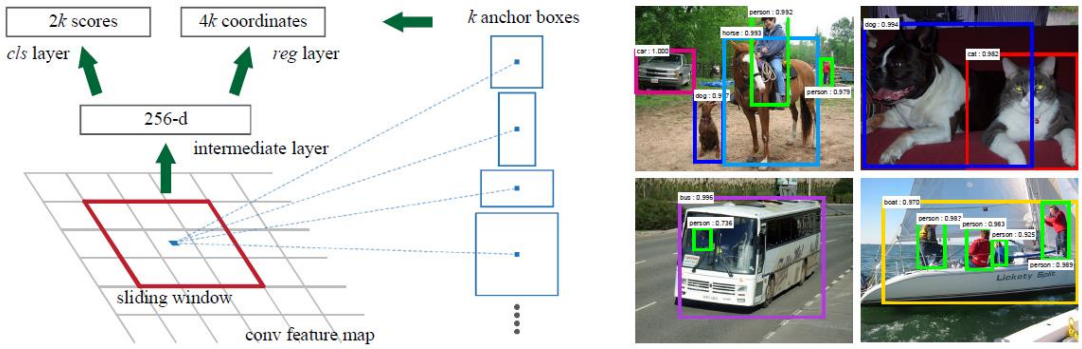


Figure 6-14: Schematic representation of the RPN structure [2].

The RoI Align layer's output undergoes processing through a series of fully connected layers, culminating in the generation of two distinct types of outputs, constituting a total size of 5·C·N. Here, C denotes the number of classes or detectable objects within the network, and N signifies the region proposals detected by the network for a given image. The initial N·C outputs correspond to the probabilities assigned to each class for every region proposal, while the subsequent 4·C·N outputs are specifically designated for bounding box coordinates. Notably, this implies that the RPN is independently trained to generate region proposals based on bounding boxes and their associated ground truth labels. An intriguing implication of this standalone training approach is that in scenarios where a pretrained network exhibits objects (classes) akin in size and shape to those intended for use by a user, a resource-

conserving strategy can be employed. Specifically, the user has the flexibility to exclusively train the classifier component of the network, opting to "freeze" the backbone CNN and RPN networks. This strategic freezing of certain components enables the preservation of computational resources while tailoring the model to specific classification objectives.

It is crucial to emphasize that the loss functions for the regression head and the classification head exhibit distinct formulations. To elaborate, the regression loss is computed through the application of the Smooth L1 Loss function, as delineated in Equation 6-8 while the classification loss is determined using the Binary Cross-Entropy Loss, as outlined in Equation 6-9. This differentiation underscores the tailored nature of the loss functions, each catering to the specific requirements of its respective task within the model.

$$L_{reg} = \sum_{i} P_i \cdot l_i$$

$$\text{With: } l_i = \begin{cases} \sum_{j}^{4} \frac{0.5\left(y_j - \hat{y}_j\right)^2}{\beta} \; if \; \left|y_j - \hat{y}_j\right| < \beta \\ \left|y_j - \hat{y}_j\right| - 0.5 \cdot \beta \end{cases} \qquad 6\text{-}8$$

Where:
Smooth L1 Loss Function: L
Anchor value (0 or 1): P_i[3]
The predicted anchor parameters (center coordinates, width and height): $\hat{y}_j$
The ground truth parameters (center coordinates, width and height): y_j

$$L_{cls} = \sum_{i=1} y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot \log\left(1 - \hat{y}_i\right) \qquad 6\text{-}9$$

Where:
Binary Cross-Entropy Loss Function: L
The true label (0 or 1) for the region i: y_i
The predicted probability from the cls (between 0-1) for the region i: $\hat{y}_i$

The total RPN Loss is calculated by the following equation.

$$RPN\ Loss = \frac{L_{cls}}{N_{cls}} + \lambda \cdot \frac{L_{reg}}{N_{reg}} \qquad 6\text{-}10$$

Where (in this work):
Normalizing factor: $\lambda = 10$
Mini batch size: $N_{cls} = 128$
Anchor boxes locations: $N_{reg} = 1250$

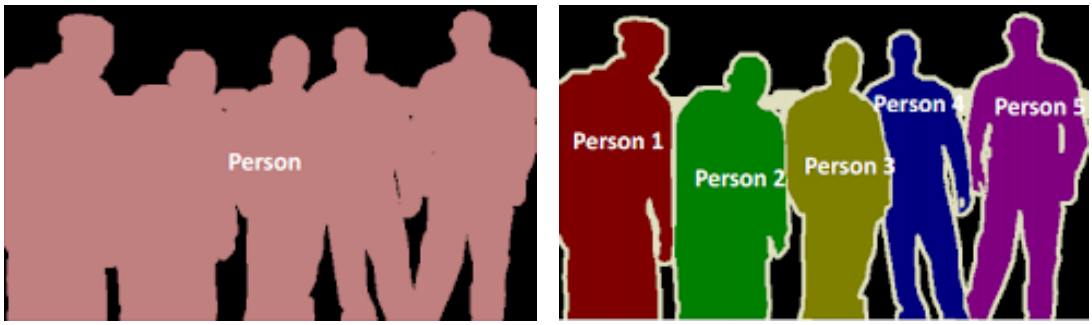
[3] $P_i = 1$: if the IoU is higher than 0.7 or if the specified anchor has the highest value of IoU in terms of all the other anchors and has an IoU higher than 0.3. For all the other cases $P_i = 0$.

6.2.1 Object Masks & Segmentation Methods

While bounding boxes serve to indicate the general position of an object, they fall short of providing a more comprehensive description. Essentially, a bounding box delineates an area where an object is likely located, yet for a more precise characterization, a finer delineation is essential. This nuanced representation is achieved through the creation of pixel-wise masks for each object in an image. A mask, in essence, is a binary image wherein ones (1) correspond to the region occupied by the object, and zeros (0) denote areas where the object is absent. This binary segmentation allows for a more granular understanding of the object's spatial extent, enabling the extraction of valuable information such as its area, perimeter, centroid, among other metrics. Consequently, the utilization of masks enhances the accuracy and efficacy of object comprehension and facilitates more sophisticated grasping techniques.

To address the intricate challenge of precise mask generation in image analysis, more sophisticated methods and CNNs have been developed. Notable examples include Mask R-CNN and YOLO v8, both representing advancements in the realm of CNNs dedicated to predicting masks for objects within an image. In essence, these CNNs extend the capabilities of traditional Object Recognition CNNs by incorporating an additional network (head) known as the Segmentation CNN. This augmented architecture empowers these networks with the ability to perform segmentation tasks, producing detailed masks for identified objects.

In general, there are two types of object segmentation methods in an image: instance segmentation and semantic segmentation. Instance segmentation involves labeling and outlining individual objects within an image, providing a detailed understanding of their boundaries. This is important in applications like medical imaging and robotics. On the other hand, semantic segmentation classifies pixels into predefined categories, giving an overall view of the scene. It is used in tasks such as scene understanding and augmented reality. The key difference lies in granularity: instance segmentation focuses on specific instances, while semantic segmentation provides a general classification of pixels, as shown in Figure 6-15. For this diploma thesis, instance segmentation was chosen for its more object-specific segmentation.



(a) *(b)*

Figure 6-15: Schematic segmentation (a) and instance segmentation (b) [20].

As mentioned in previous sections the network used in this work is the Mask RCNN that extends the foundation of the Faster RCNN. Specifically, the Mask RCNN network introduces a supplementary branch dedicated to predicting segmentation masks for each Region of Interest generated by the Region Proposal Network. The mask segmentation network (head) as depicted in Figure 6-16 is characterized by a CNN structure, exclusively comprised of convolution layers. This network operates concurrently with the classifier network and the bounding box regressor network, enhancing the comprehensive capabilities of the model. The incorporation of the mask segmentation branch facilitates the precise delineation of object boundaries within identified regions, contributing significantly to the network's capacity for detailed instance segmentation in object detection tasks. This parallel architecture demonstrates the versatility and adaptability of the Mask RCNN in simultaneously addressing classification, bounding box regression, and pixel-wise segmentation tasks. A simplified schematic representation of the Mask RCNN network is shown in Figure 6-17.

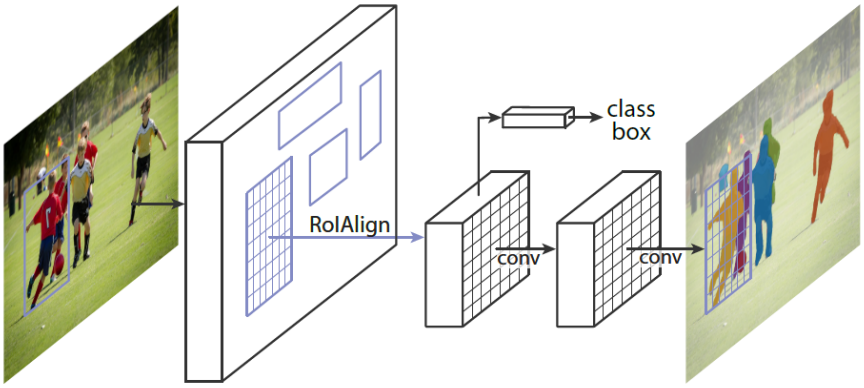


Figure 6-16: Mask R-CNN segmentation layers [3].

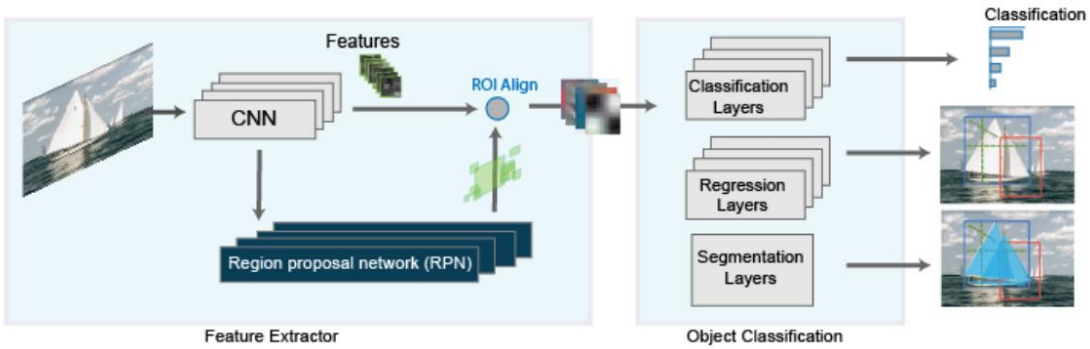
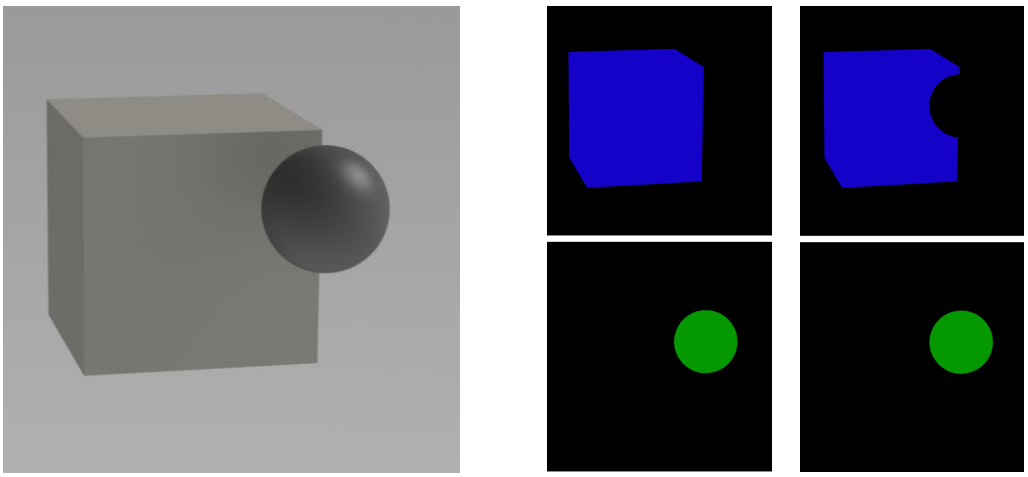


Figure 6-17: Schematic representation of Mask R-CNN [20].

The Loss function of the mask network is the already discussed binary cross-entropy loss function denoted in Equation 6-9. The sum of all the loss functions discussed so far (Box regression- Classification, RPN and Mask loss) is equivalent to the total loss of the network.

6.3 Mask RCNN Implementation

In this diploma thesis Mask R-CNN pretrained with the MS COCO dataset was employed for the machine vision part of the robotic system [20]. The focus was on predicting bounding box coordinates and segmentation masks to achieve object delineation. A common problem that arouses in those type of applications (robotic grasping of items) is the occlusion that occurs during object stacking. In simpler terms, occlusion refers to the situation where one object is in front of another, partially or completely hiding it from view. This creates a challenge in the grasping strategy that the robot has to follow due to uncertainties of the position of the objects. However, many researchers have suggested methods for predicting the masks of the occluded objects, others by training the network with the full shape masks (incorporating and the occluded areas) [14] whereas others predicting the non-occluded masks [8]. Figure 6-18 show the difference between those two methods. In this work the non-occluded masks are adopted, as in practice it isn't easy obtain clean ground truth data for the occluded parts. The full shaped masks can be created from computer generated images (from CAD etc).



(a) (b) (c)

Figure 6-18: Example image (a) full shaped masks (b) and non-occluded masks (c).

6.3.1 Data generation-Image Labeling

As described and in the beginning of this diploma thesis, in order to train the Mask RCNN model there needs to be data. There are three types of objects used in this work shown in Figure 6-19.

(a) (b) (c)

Figure 6-19: Objects used in this work. "PT1" (a), "PT2" (b), "PT3" (c).

Due to the uniqueness of the objects used for detection these data needed to be generated by hand. For this reason, 114 images were taken containing these objects in different configurations. Those images are then cut to match the depth images however they are not resized to match their resolution. To elaborate as mentioned and in the calibration section of the Kinect, the images resolution is cropped from [1920,1080] to [1273,1038] which if scaled down by the PAR (=3.05), gives (if rounded) [417,340] which is the depth image resolution. This was done to increase the performance of the network during training as higher resolution incorporates more data to process. Figure 6-20 depicts the setup used to take the required pictures.

Figure 6-20: Camera setup for acquiring images.

Consequently, using the "Image Labeler" app in MATLAB, polygon shaped masks are created by hand for each object class, where for each polygon mask there are two additional variables to be set:

- **Item (Integer)**

  This variable denotes the designated object identifier within a particular class. Its purpose is to facilitate the subsequent consolidation of masks pertaining to a singular object that may be occluded, resulting in the generation of two distinct masks. As illustrated in Figure 6-21 "PT3" has three distinct masks. Consequently, the item variable is assigned a value of 1 for all the masks corresponding to the class "PT3", signifying that the masks collectively represent a singular object.

- **Occlusion (Boolean)**

  This variable serves as an indicator for the occlusion status of the object, assuming a value of "true" if the object is occluded and "false" if it is not. Subsequently, this

information is harnessed for the training of occlusion networks. As depicted in Figure 6-21, the object labeled "PT3" is occluded, leading to the assignment of the occlusion property as "true" for all the masks representing the identical component. Additionally, "PT2" is occluded by "PT1" in the bottom edge thus the occlusion property is set to "True".

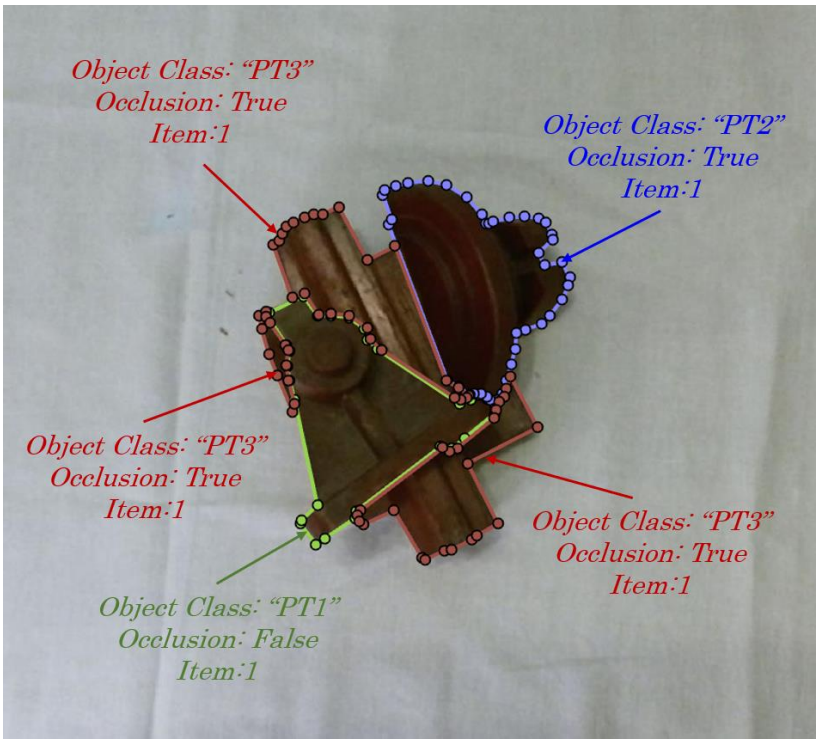


Figure 6-21:Example of occluded object "PT3" and "PT2" with labeling variables.

Then by exporting the masks and the variables as a table, the annotation for each image is created from a custom made MATLAB code that does the following:

- Transformation of the polygon points to binary images of size [1273,1038] that are then concatenated to a 3D matrix of size [1273, 1038, n] where n is the number of objects in the image to represent the masks.
- Generation of the bounding boxes [x,y,w,h] from the polygon points by taking the maximum distance in y and x direction.
- Export one annotation (.m) file containing the image name with its labels, bounding boxes and masks in the form of cell arrays. The naming of the cell arrays must be kept the same as described in the coco API for the generation of the file datastore used in training.

An "annotation.m" file is exported for every image into a folder containing only the annotation files for the training. Then a file datastore is created using the coco API function, for the folder containing the annotation files. This datastore is then used to train the Mask RCNN object. It is noted that in this way memory is preserved, because only the data for one image is loaded at a time rather than the whole datastore. Figure 6-22 shows some examples of ground truth labeled data.

Figure 6-22: Ground Truth data for training and validation purposes.

Although there are only 3 types of objects, they share similar characteristics, like color and geometries making it harder to distinguish from each other. Adding more objects (classes) would not change the general procedure of labeling and training, however it would require a larger amount of training data to be generated, increasing the manhours required for this step. For context on average 2 minutes where needed to generate the labels for each image.

6.3.2 Data augmentation

In general data augmentation is employed in image segmentation networks to enhance model generalization, robustness, and performance. By artificially expanding the training dataset through transformations like rotation, flipping, and scaling, data augmentation addresses the challenges of limited labeled data, improves the model's resistance to variations in lighting, orientation, and scale, and reduces overfitting. This regularization technique fosters the learning of generalized features, making segmentation models more adept at handling diverse object shapes, sizes, and orientations.

In the context of this diploma thesis, given the limited quantity of training images, a data augmentation algorithm was devised to increase the training dataset by a factor of 10. This algorithm utilizes pre-annotated images for the generation of new instances through random rotations, translations, and the incorporation of diverse backgrounds selected from a pool of 10 options. The random rotations and translations are independently applied to each object using the rand() function in MATLAB, and the resultant images are merged with their respective masks. Importantly, the concatenation order is randomized, leading to random object occlusion in the augmented images. This randomness introduces complexities in the masks and bounding boxes of the objects within an image, necessitating adjustments. Consequently, all bounding boxes are modified to encompass solely the non-occluded regions of the corresponding objects. If an object is significantly occluded (area < 100 pixels), its mask and bounding box are excluded from the annotation file. Subsequently, following the same procedure as before, annotation files are generated for each augmented image, and a new datastore for training is constructed. Figure 6-23 shows some examples of the augmented images.

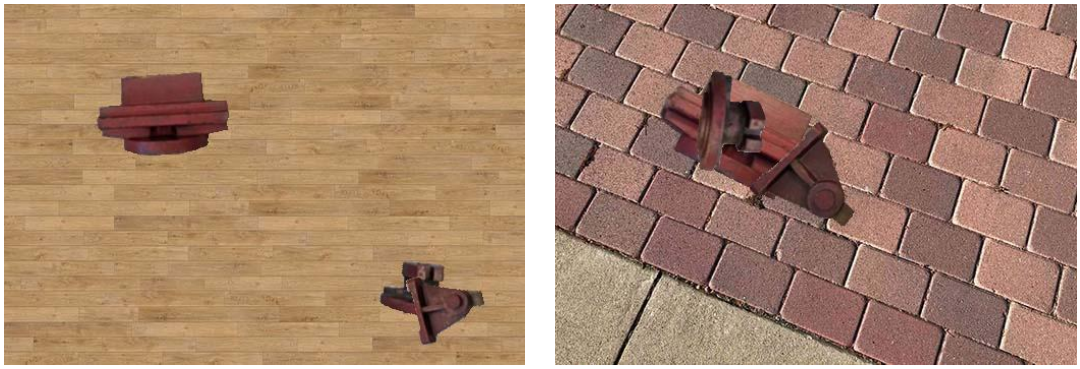


Figure 6-23: Augmented images generated for training.

6.3.3 Training

In the initiation of the network the anchor boxes (15 in total) remained the same as the pretrained network. Training encompassed the entire network, including the backbone, Region Proposal Network (RPN), and classifier, as superior outcomes were observed with this comprehensive approach. The training process was bifurcated into two stages: the initial stage involved augmented images and annotations (1140 cases), while the subsequent stage utilized real-world images (114 cases). The training was executed on a personal computer equipped with a GPU (RTX 3090), leveraging a minimum of 20GB of VRAM during the training process. Detailed specifications of the training algorithm options are provided in Table 6-4, while the Mask R-CNN training options are comprehensively outlined in Table 6-5.

Table 6-4: Training algorithm options.

Parameters	**Value**
Training Algorithm	Stochastic Gradient Decent with Momentum
Initial Learn Rate	0.001
Momentum	0.9
Learn Rate Schedule	Piecewise
Learn Rate Drop Period	1
Lear Rate Drop Factor	0.95
Max Epochs	10
Mini Batch Size	2
Bach Normalization Statistics	Moving

Table 6-5: Mask R-CNN additional options.

Parameters	**Value**
Number of Regions to Sample	128
Number of Strongest Regions	1300
Positive Overlap Range	[0.75, 1]
Negative Overlap Range	[0, 0.75]
Number of Anchor Boxes	15

Each parameter significantly influences both the training time and the overall performance of the network, with MATLAB and various sources providing valuable

insights into the purpose of each [8]. However, among these parameters, two prove particularly pivotal in achieving optimal regression for the network: the Positive Overlap Range and the Negative Overlap Range. Essentially, these values dictate whether an anchor is selected as positive sample or as negative one based on the Intersection over Union (IoU) value with the training data. To elaborate, the Positive Overlap Range determines the acceptable range of IoU values with the training data, while the Negative Overlap Range serves the opposite purpose. By elevating the threshold from 0.5 to 0.75, the network exhibited accelerated regression, accompanied by a notable reduction in total training errors from 0.7 to 0.02. Furthermore, this adjustment successfully addressed issues associated with multiple bounding boxes per item. In Table 6-6 the training performances are shown, where the RPN Loss is calculated as described in Equation 6-10, the RMSE is the mean squared error for the box regressor of the head, the Mask loss is described with 6-9 (binary cross entropy) and lastly the Total Loss is the sum of those three errors. Figure 6-24 depicts a diagram of the training process after training with the augmented data was applied.

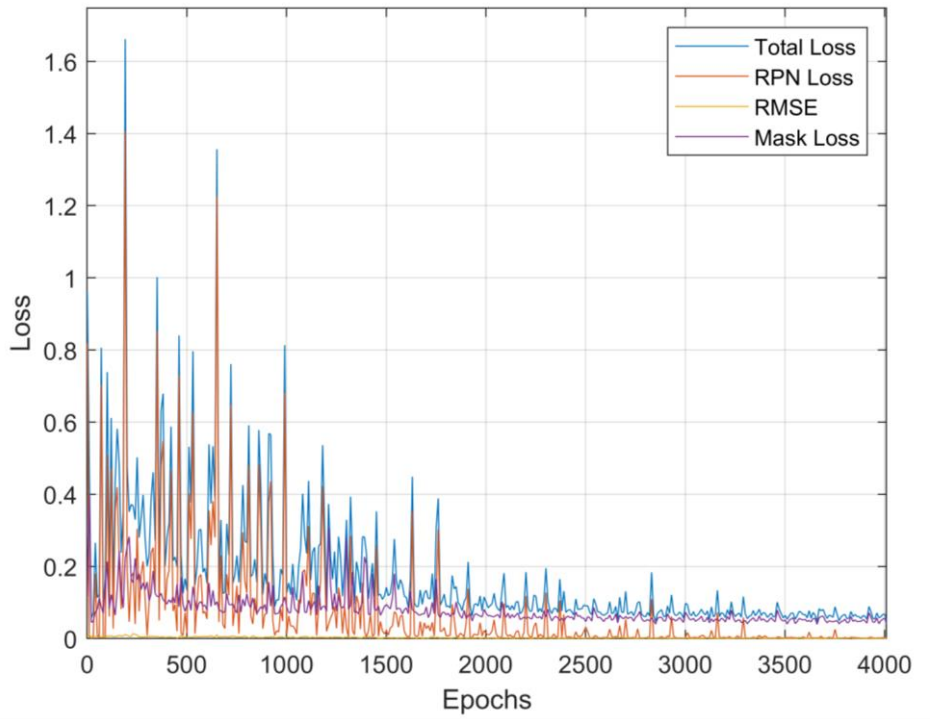


Figure 6-24: Mask RCNN training diagram.

Table 6-6: Mask RCNN training performance.

RPN Loss	RMSE	Mask Loss	Total Loss
0.0016	0.0004	0.0153	0.0223

6.3.4 Testing & Results

To assess the network's performance, three distinct datasets were curated, each comprising a total of 15 ground truth-labeled images, following the creation process of the training datasets. The initial dataset encompasses all objects within each image, without any inter-object occlusion. The second dataset introduces mild occlusion (<25% maximum area of an object is occluded) between objects, while the third dataset features images characterized by substantial occlusion (>25% minimum area of an object is occluded) between objects. This varied testing methodology aims to evaluate the model's accuracy across diverse scenarios and discern the impact of occlusion on the overall network performance. By systematically testing the model under different levels of occlusion, valuable insights into its robustness and adaptability to varying conditions are gained, contributing to a comprehensive assessment of its capabilities. Table 6-7 show the performances for each dataset while Figure 6-25, Figure 6-26 and Figure 6-27 show one example for each dataset respectively.

Table 6-7: Mask RCNN testing performances.

Case	RPN Loss	RMSE	Mask Loss	Total Loss
No Occlusion	0.0022	0.0005	0.0179	0.0258
Mild Occlusion	0.0031	0.0005	0.0251	0.0352
High Occlusion	0.0032	0.0009	0.0381	0.0520

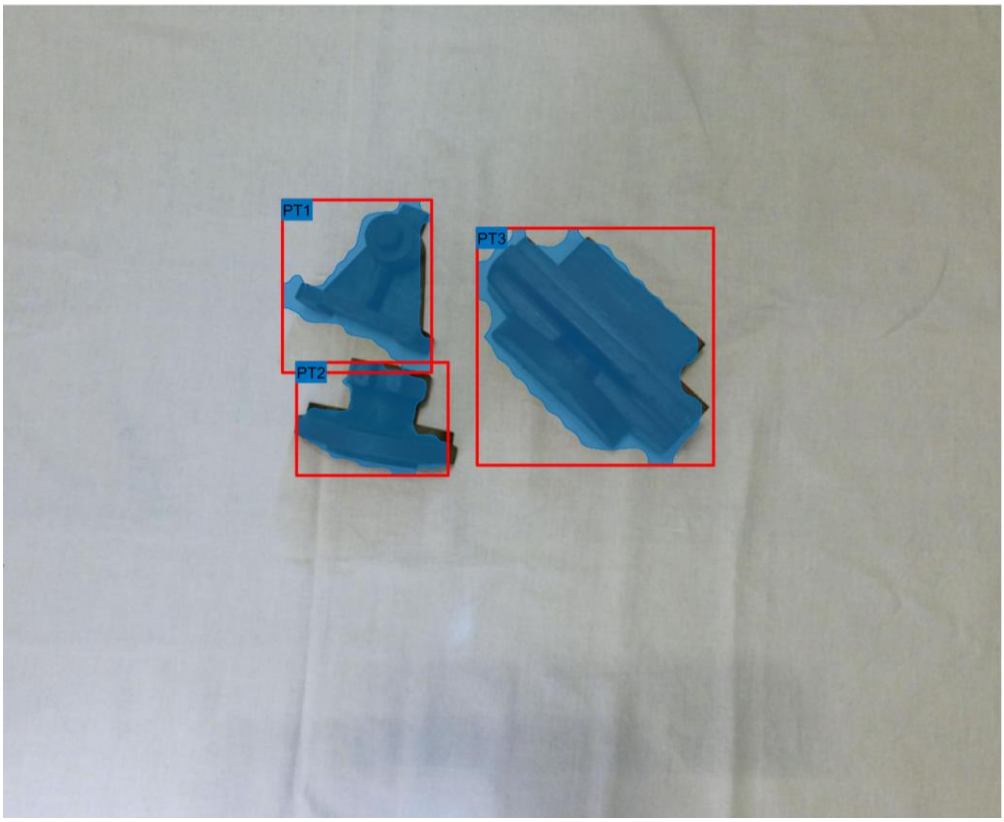


Figure 6-25: Model testing in no occlusion environment.

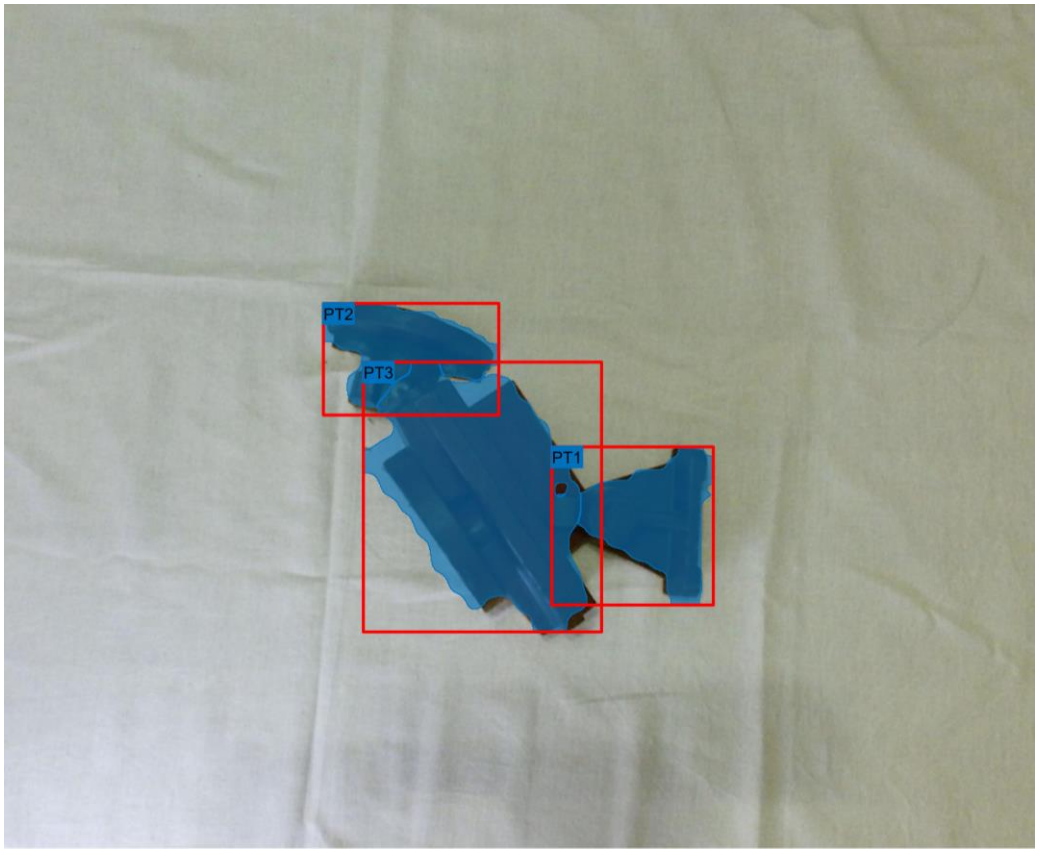


Figure 6-26: Model testing in mild occlusion environment.

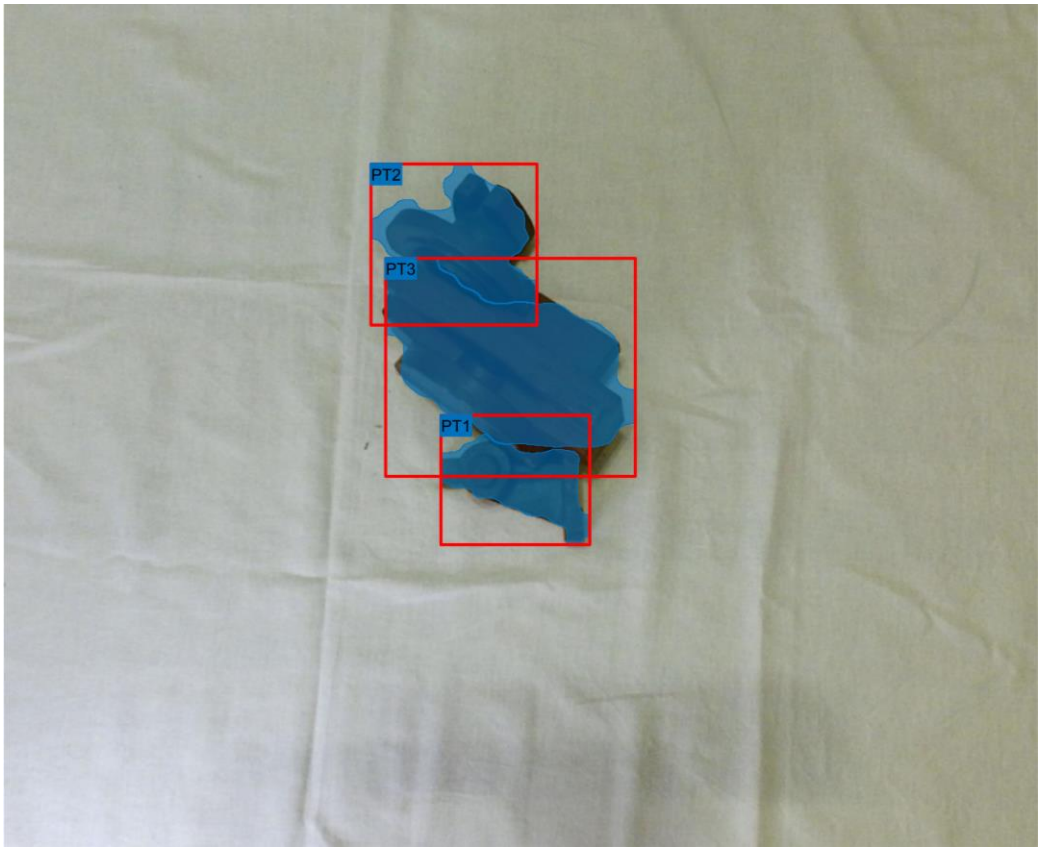


Figure 6-27: Model testing in heavy occlusion environments.

The obtained results reveal consistent values comparable to those witnessed during the network training phase, underscoring the network's capacity for generalization across diverse environments in segmentation tasks. It is essential to highlight that as the occlusion percentage increases, an increase in error occurs, indicating the heightened challenge for the network to generate precise masks. This escalation in error is predominantly attributed to the amplified mask loss in instances of increased occlusion. Notwithstanding this, the outcomes achieved in this thesis surpass those reported by [8] (total loss of 0.08), wherein depth data was also incorporated for network training. It is pertinent to acknowledge the disparity in dataset characteristics, as [8] incorporated a significantly higher number of object classes and objects per image, that should result in higher errors. In summary, the network's performance, as demonstrated in this study, proves to be sufficiently adept in addressing occlusion problems even in demanding scenarios.

7. Occlusion Detection

7.1 Parameters

In this dissertation, the implementation of feedforward neural networks is employed to address the occlusion problem. Fundamentally, these networks are designed to ascertain whether a detected object is subject to occlusion. This process facilitates the development of a straightforward algorithm that utilizes occlusion data to determine the selection of objects for manipulation. To elaborate, if an object is identified as occluded, it is precluded from being chosen. Each distinct object type is addressed by a separate artificial neural network. Although this thesis focuses on three specific object classes, a generalized approach could be devised to train k networks for k classes. The uniformity of inputs across all networks is maintained, comprising the following:

- **Background Height**
  Background height refers to the vertical distance between the background and the depth sensor of the Kinect device. The initial step involves the generation of a background mask through a logical "or" operation applied to the masks of the objects corresponding to the specific image and by taking its inverse. Subsequently, the derived mask is spatially aligned with the depth image, enabling the computation of the average depth value associated with the background mask.
- **Opened Perimeter**
  The opened perimeter is defined as the ratio of the mask perimeter of the object by the background height. This approach enables the network's applicability across varying distances of the background from the camera, accommodating different aspect ratios of the object. Consequently, this adaptability enhances the robustness of the system to changes in its configuration.
- **Opened Area**
  The opened area is defined as the ratio of the mask area of the object by the square of the background height. This approach enables the network's applicability across varying distances of the background from the camera, accommodating different aspect ratios of the object. Consequently, this adaptability enhances the robustness of the system to changes in its configuration.
- **Object Height**
  This is the difference between background height and object height, where the object height is the average distance between the object from the camera that is calculated by mapping the masks to the depth values and taking their average.

- **Distance Between Closest Object**
  This metric denotes the distance, expressed in meters and calculated through the utilization of background height and focal length parameters, between the closest object in proximity to the object under inspection, as illustrated in Figure 7-1. This deliberate incorporation of distance information enables the network to assimilate an additional discerning feature that may serve as an indicative factor for occlusion. In instances of occlusion, this distance is anticipated to be minimal, theoretically approaching zero.
- **Height Difference Between Closest Object**
  This parameter represents the disparity in height, measured in distance from the camera, between the nearest point of the closest object and the closest point to the same object within the inspected region. To mitigate the impact of potentially erroneous depth values, the average height of the respective areas of both objects at the specified points is calculated. Figure 7-1 provides an illustrative example of this procedure. The integration of this input, coupled with the distance information for these objects, empowers the network to acquire an additional reference for discerning occlusion. In practical terms, a negative value may suggest occlusion of the inspected part, while a positive value may imply potential occlusion of the nearest object.

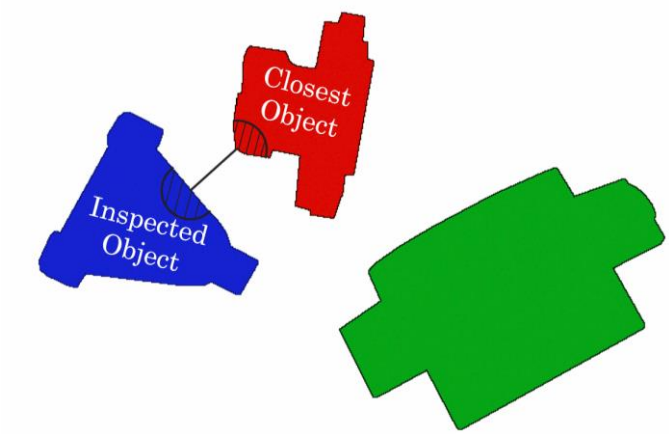


Figure 7-1: Visual representation of closest distance between objects, and areas to calculate the height difference between the objects.

The input data for the networks necessitates the inclusion of labels to discern the appropriate network, alongside the masks and the depth image. Extensive preprocessing techniques are applied to extract the specified features. The network's output is a singular Boolean value, where a result of 1 denotes occlusion of the object, and 0 signifies its absence. It is imperative to note that in scenes featuring only a solitary object, the evaluation for occlusion becomes redundant; thus, the aforementioned procedures necessitate the presence of at least two objects within an image. During the training phase, output values are derived from the ground truth table outlined in the data generation section of the Mask R-CNN, utilizing an additional variable specifically dedicated to occlusion.

7.2 Training process

For the training of the networks a total of 114 images were used that were annotated during the Mask RCNN data creation process. However, a new datastore needed to be generated, with resized masks, bounding boxes and images to match the resolution of the depth images. Furthermore, no data augmentation was applied, because the depth data are more susceptible to errors thus augmenting the errors would results in a less valid approach, making the networks less reliable in real world scenarios. The data for training each network is split automatically from specially written code exporting three different tables (for each object class) with each one containing the inputs and the output as described in the previous section.

The idea behind the training process was to implement as many hidden layers are needed to be able to learn the complexity of the problem. Increasing the hidden layers did not severely increase the training time as GPU computing was used (RTX 3090), utilizing thousands of cores to complete the operations. The optimization method that was used, was the resilient back propagation (RP) because from a selection of few it registered the best results. It is noted that while Levenberg-Marquardt (LM) method seems to get the same results as RP with less hidden layers, the implementation of the optimization algorithm is based on CPU computing increasing the computational time substantially.

Upon delineating the system's inputs and output, the data are categorized into distinct subsets to facilitate the training and assessment of the neural network. The chosen categories adhered to the training/validation/testing paradigm, with an associated allocation of 80/10/10 (%) respectively. The inclusion of the "Validation" category was deemed imperative due to its role in mitigating overfitting, notwithstanding the reduction in available data for the training phase. The deliberate selection of a specific percentage for testing and validation, was driven by the limited amount of data acquired, however they are deemed enough to reduce the possibility of a “lucky” network that cannot handle generalized cases.

The network's performance evaluation criterion is defined by the selection of the mean squared error (MSE), a method chosen for its capability to magnify substantial deviations, consequently imposing a more pronounced penalty on the model. It’ s emphasized that although it’ s common to use cross entropy for the evaluation, it was found that MSE performs better in this case (one binary output). It is imperative to note that the errors are calculated for all three subcategories separately and are summed in the end to create the total error of the network. To elaborate, all three categories have the same amount of influence in the final selection, even though the training subcategory category is containing 80% of the data. By this implementation, the chosen network would be able to handle generalized inputs, but it should also be able to handle similar cases with the ones it was trained.

Consequently, the hidden layers where set manually (increasing them in size) until no increase in performance was observed. Additionally, for each set of hidden layers a loop was generated, comprising of 100 iterations, undertaking the estimation of 100 neural networks sharing identical structures but initialized with different weights. This meticulous process ensures the preservation of the network ability, by taking the most favorable performance, as indicated by the lowest MSE. Such an approach not only enhances the statistical accuracy of the results but also serves as a guiding mechanism, enabling us to make well-informed decisions pertaining to the performance of each parameter set.

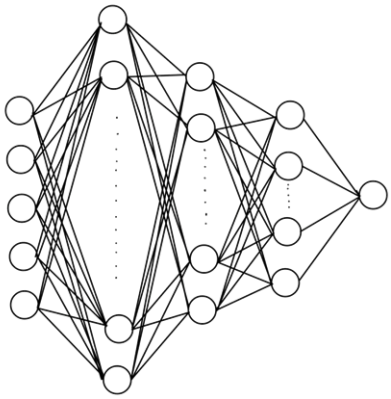


Figure 7-2: Schematic representation of the Neural Networks implemented in this work.

The activation function employed between neurons is the hyperbolic tangent function, denoted as "tansig", with a subsequent rounding function applied in the output layer. Specifically, if the output is greater than or equal to 0.5, it is assigned a value of 1; otherwise, if it is less than 0.5, it is set to 0. It is pertinent to acknowledge that an alternative activation function, namely the Rectified Linear Unit (ReLU), was also investigated as a substitution for the “tansig” function to assess potential performance improvements. However, the empirical results indicated a decrease in performance. Table 7-1 delineate the parameters for each network during both the training. Table 7-2 elucidates the results of the training and testing process.

Table 7-1: Training parameters for occlusion networks.

Maximum Epochs	Maximum Validation Increases	Regularization	Normalization	Training algorithm
1000	20	false	false	Resilient backpropagation

Table 7-2: Hidden layers and errors of each network.

Object Class	Hidden Layers	Training	Validation	Testing
PT1	[50,50,25,25,10,10,5,5]	2	1	1
PT2	[100,100,50,50,25,25,10,10,5,5]	0	0	2
PT3	[100,100,50,50,25,25,10,10,5,5]	4	1	1

7.3 Testing Results

Upon scrutiny of the network parameters, it becomes evident that the rationale guiding the design of the hidden layers was to commence with a substantial number of neurons and progressively reduce this count with each subsequent layer, culminating in the final layer. The writer's experiential insights suggest that such an implementation proves efficacious in networks where the number of inputs exceeds that of the outputs. Furthermore, this approach simplifies the strategy employed to enhance performance-namely, augmenting the number of layers and their respective sizes. In contrast, for other network architectures, the means to improve performance may be less straightforward, potentially introducing ambiguity concerning how alterations in structure could positively impact performance.

The overall performance of the networks appears commendable and aligns - surpasses other networks documented where the obtained performances achieved was 68.3% [14] whereas in this thesis the combined accuracy in testing was 90.5%. It is imperative to underscore, nonetheless, the discrepancy between data type used for training, were in [14] more objects and more object classes exist in a single image, increasing the difficulty of the problem. Additionally, the training data for the network comprises ground truth labels, wherein the implementation phase, both masks and labels are derived from the Mask R-CNN network. This underscores a critical dependency, where the efficacy of the entire model is substantially contingent on the segmentation performance of the Mask R-CNN network. Table 7-3 shows in more details the type of error found from during training and testing.

Table 7-3: Type of error of Occlusion ANNs.

Object Class	False Positive	False Negative
PT1	1	3
PT2	0	2
PT3	1	5

Although the testing results indicate an overall "good" performance, it's essential to acknowledge that the networks may underestimate the likelihood of occlusion. The disparity observed in false positive and false negative results can be attributed to the nature of inputs provided to the networks. Specifically, in instances where significant occlusion occurs between objects, and multiple object masks intersect closely or overlap, the parameters "Closest Object Distance" and "Height Difference Between Closest Objects" may not accurately identify the object responsible for occlusion. Consequently, these parameters might erroneously refer to another occluded object with a lower height than the inspected part, leading to the incorrect conclusion that the inspected part is unobstructed.

From a simple logical perspective, it's preferable for networks to err on the side of overestimating occlusion, thereby producing false positive results. This approach ensures that even in the absence of occlusion, the network would still detect parts as

occluded. Consequently, a logic-based grasping technique would disregard grasping the specific object affected by the occlusion, opting to grasp another. However, with networks that underestimate occlusion, the opposite scenario may occur. In such cases, if the networks fail to recognize the correct occlusion properties of an object, the grasping logic might lead to attempting to grasp the occluded object, potentially risking collisions with other objects obstructing it. Non the less the possibility of the later is counteracted by implementing a highest first logic in the grasping technique.

8. Grasping Technique

Within this study, the robotic arm has been outfitted with a proprietary gripper. Consequently, the present investigation will adopt a strategy centered around a gripper device, as opposed to a suction apparatus. The rationale underpinning the grasping technique is derived from the way humans naturally manipulate and secure objects. More precisely, human tendencies involve grasping items at two points situated proximate to the center of mass, positioned in close proximity to each other, as illustrated in Figure 8-1.

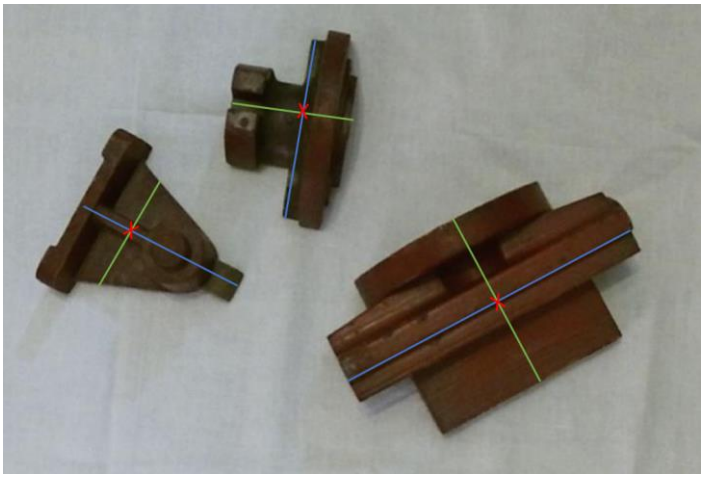


Figure 8-1: Center of areas (red crosses), Grab axis (green lines), Long Axis (blue lines).

Henceforth, an algorithm has been devised to facilitate the precise grasping of a designated object. Initial computations involve the determination of the center of area for the specified object mask, a point generally proximal to the center of mass. Subsequently, the angle of rotation of the object part is ascertained by considering the angle of its largest axis. Concurrently, the average height of the object is calculated utilizing both the mask and depth data, and a constant value of 10mm has been established below this average height to determine the optimal grasping position for each object.

Importantly, a preventive mechanism has been implemented to safeguard against potential collisions with the tabletop. Specifically, if the calculated height falls below the table surface, the grasping height is automatically adjusted to be 10mm above the table. Furthermore, within the scope of this investigation, the challenge of occlusion is addressed. A dedicated methodology has been incorporated to discern and exclude occluded objects while prioritizing those that remain visible. In instances where multiple fully visible objects are present, the algorithm is designed to make discerning choices among them. To achieve this, the occlusion predicament is distinctly managed in a preceding section, and here, the outcomes from occlusion neural networks are utilized as inputs.

The underlying rationale guiding the grasping strategy involves refraining from grasping occluded objects until they transition to a fully visible state. Subsequently,

among the fully visible objects, the algorithm is programmed to prioritize and grasp the object with the greatest height, signifying its proximity to the camera.

Derived from the aforementioned considerations, it becomes evident that for the successful implementation of this strategy, it is imperative to capture an image each time an object is extracted from the scene. Subsequently, the acquired image undergoes processing through both the segmentation network and the occlusion networks to identify and select an object for grasping. This procedural choice is motivated by its inherent resilience compared to formulating the entire grasping strategy based on a solitary image.

The efficacy of this approach is particularly pronounced in scenarios involving occlusion, where certain objects may not be initially visible within the scene. However, it is imperative to acknowledge that this method amplifies computational time by a factor of N, corresponding to the number of objects present in the scene. Despite the computational trade-off, the enhanced robustness achieved, especially in occlusion scenarios, justifies this strategic decision.

9. Virtual World Implementation

9.1 Virtual world initialization strategy

As emphasized in the introduction, the establishment of a virtual environment that faithfully replicates the real-world context holds paramount significance for the in-depth examination of control methodologies governing the robotic arm, computer vision, and object grasping. At this juncture, various approaches are available for simulation within a virtual environment with seamless integration with "MATLAB-Simulink," each offering distinct advantages and drawbacks. This work delves into three such approaches, placing particular emphasis on the selected one.

The first method initially employed involved the option provided within "Simulink," specifically the Virtual Reality package found within the Simulink 3D Animation blockset. The primary advantage of this package lies in its ease of integration, allowing the robotic arm to be effortlessly inserted into the 3D world by importing its rigid body tree and configuration (joint rotations). Additional benefits include extensive documentation due to its longer history in the realm of 3D animation. However, notable disadvantages include the inability to incorporate an RGB-D camera into the model, and the graphics may appear outdated.

The second option entails creating the environment in the external program "Gazebo" and utilizing the "Simulink-Gazebo" blockset for communication between programs. Gazebo is a well-established program in the field of robotics, offering diverse functionalities and simulations, including RGB-D cameras, sensors, and fully integrated robotic systems. However, this approach introduces an additional step in the simulation process and increases the system's complexity especially in the windows operating system. Moreover, the graphics are not superior to those of the previous method.

A more recent and advanced approach involves MATLAB's collaboration with Unreal Engine to create virtual environments for simulation within Simulink. MATLAB offers ready-to-use airspace, drone, and automotive applications in the Unreal Engine environment, with corresponding blocks available in the aforementioned categories and the 3D animation toolbox. A significant advantage of this 3D environment lies in its realistic lighting, shading, and color representation of objects within the scene. Additionally, it enables the insertion and movement of RGB-D cameras, a feature crucial for the real-world application. The primary drawback is the absence of a straightforward method for modeling robotic arm movements using a single ready-to-use block. Furthermore, for optimal scene representation, the file type must be ".fbx," which is not supported by SOLIDWORKS. However, Blender was utilized for configuring lighting and object color due to its more advanced visualization techniques, and it can output files in the ".fbx" format.

9.2 Robotic Arm Modeling

Initially, the work begins with importing graphic files of the ".step" type, which represent each segment - element of the robotic arm, into SOLIDWORKS for the assembly construction. The way in which the components of the robotic arm are defined is crucial, where the reference point (origin) of each component must be located at the corresponding joint that rotates it and have the correct orientation (z-axis in the direction of rotation, according to the right-hand rule).

In addition to incorporating the arm files, one supplementary assembly was introduced into the model: the arm's gripper (end effector). It is imperative to meticulously oversee and position the reference point, ensuring that the z-axis extends outward up to the location of grabbing. Figure 9-1 shows the robotic arm assembly within SOLIDWORKS.

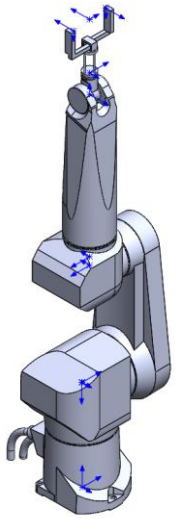


Figure 9-1: Stäubli RX90L Assembly with all joints in zero position.

A crucial step involves converting the graphical assembly into a format compatible with the control software, specifically MATLAB. Two alternatives are at one's disposal: the "multibody link" add-on designed for MATLAB, seamlessly integrated into SOLIDWORKS, and the more versatile “SW urdf exporter”, capable of producing a robot arm file in the ".urdf" format, universally readable by programs within this domain. It is imperative to emphasize that, in both instances, the arm must be positioned at its "zero" position, characterized by all joints possessing a value of zero, prior to the conversion process. Additionally, the mates between the manipulators links should be exclusively done with contact (coincident) and rotation (concentric) constraints.

The initial approach involved the application of the first method to introduce the robot into the MATLAB environment. Nevertheless, complications emerged during the transfer of graphic elements in the ".stl" format, as well as in the accurate positioning of the robot in Simulink. These challenges were effectively mitigated through the

utilization of the "SW urdf exporter". Facilitating a streamlined procedure, the "SW urdf exporter" delineates joint types, their constraints concerning torque and rotation and reference points for each "link" element. It must be noted that it is better to name the base of the robot as “base” and the end effector (gripper) of the robot as “tool0” to avoid problems during the inverse kinematics in MATLAB. It is pertinent to observe that solely constraints pertaining to rotation and angular velocity were specified due to the unavailability of precise motor torques. However, this limitation remains inconsequential for the present undertaking, as the intricacies of the arm's dynamics, particularly with respect to the application, are of nominal significance.

Following this, the option is presented for the exportation of the ".urdf" file concomitant with the graphical elements (meshes). These elements are organized within a directory encompassing comprehensive information pertaining to the generated robotic arm. It is imperative to highlight that MATLAB treats each arm as an object of the "rigidbodytree" type and not as ".urdf". Nevertheless, the transition between these two formats is seamlessly executed through the utilization of the "importrobot()" command.

9.3 Establishing 3d Environment Within Simulink

In any scenario requiring the incorporation of a 3D scene into Simulink using Unreal Engine, a "Simulation 3D Scene Configuration" block is essential to establish the initial parameters of the scene. At the time of writing, the scene must be configured as an Empty scene, and the Scene Source should be set to "Default Scenes." This configuration is chosen to enable the creation of the simulation within MATLAB rather than running an already preconfigured simulation. Subsequently, all the elements of the scene (except from the robotic arm), are exported from Blender, and are introduced using the "Simulation 3D Actor" block.

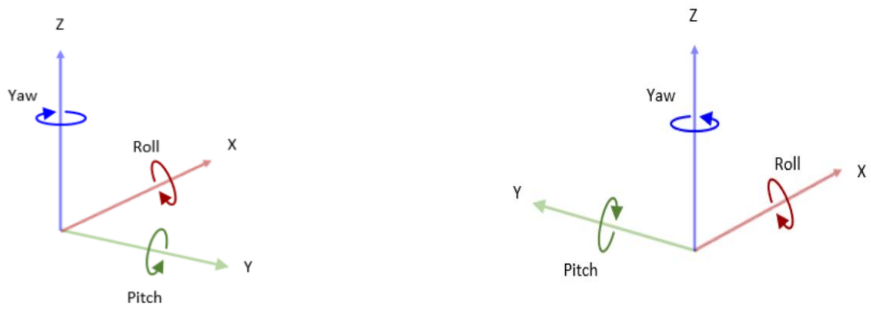
During this phase, two primary methodologies were assessed. The initial approach involved the importation of objects designated for grasping as individual ".stl" files. A random position and orientation generator were then employed to situate these objects above a specified region on the table. Subsequently, utilizing the physics engine inherent in the Unreal Engine, the items would descend and orient themselves in a randomized manner, enabling comprehensive testing of the network under diverse scenarios. However, despite the initial implementation, the rendering quality of surfaces proved suboptimal, even for high-quality ".stl" files. This issue adversely impacted the detection accuracy of objects by the Mask RCNN network, upon which the remainder of the system relies.

Subsequently, an alternative approach was adopted, entailing the insertion of objects into the scene as ".fbx" files. However, a notable limitation of this method is the inability to utilize the physics engine on a per-part basis within the ".fbx" file. Consequently, the initial method involving falling objects could not be implemented. To overcome this limitation, a combination of Blender based animation with a manual

placement strategy within the Simulink model was employed. To elaborate, the placement of the objects was completed inside Blender by using the physics engine and letting the objects fall from a distance above the table. Then the positions and orientation of the objects was captured, and it was inserted inside the model.

For simulating the camera, a "Simulation 3D Camera" block is incorporated with characteristics mirroring those of a real-world camera, such as the "Kinect v2"(Image Resolution = [417,340], Focal Length = [315,315]). Additionally, in the Ground Truth tab, "Output Depth" is enabled.

The pivotal element in the simulation was the 3D representation of the robotic arm. The STL files delineating the robot links are automatically incorporated into the scene through the introduction of a "Simulation 3D Actor," with its input specified as the ".urdf" file representing the robotic arm. The configurations, denoting the joint rotations of the robotic arm, are subsequently converted into homogeneous matrices, encapsulating the positional and orientational attributes of the links. However, a notable challenge arises from the incongruity between the coordinate systems employed by MATLAB and Unreal Engine. To address this incongruence, a specialized function was developed within Simulink to effectuate the conversion of MATLAB coordinates and orientations into the corresponding representations within the Unreal Engine framework. Figure 9-2 show the disparity between the coordinate system of MATLAB and UE, whereas Figure 9-3 shows the Simulink model that converts the MATLAB coordinates to UE coordinates for the visualization of the robotic arm.



(a) *(b)*

Figure 9-2: Unreal Engine coordinate system (a), MATLAB coordinate system(b).

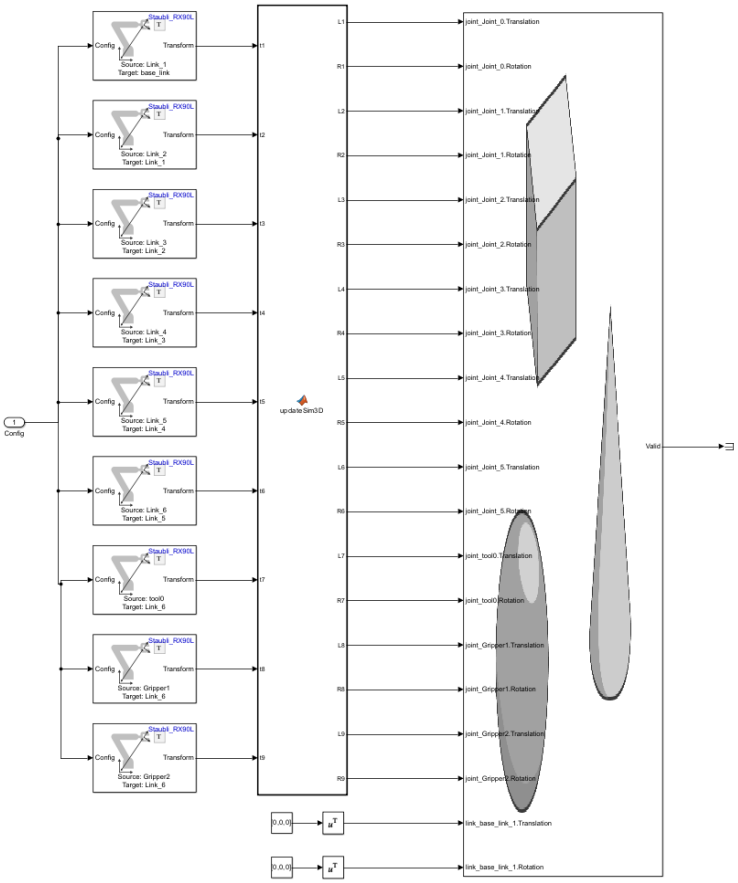


Figure 9-3:MATLAB to Unreal Engine robotic arm modeling and transformation of coordinate systems.

9.4 Grasping Simulation

To accurately represent the entire process visually, it became imperative to visualize and simulate the grasping aspect as well. While the concept of grasping an object is straightforward and easily comprehensible in the physical realm, it presents certain challenges within the 3D environment. In the pursuit of a solution to this issue, two methods were deliberated upon.

The initial approach involved leveraging the physics engine within the Unreal Engine environment to simulate the act of "grasping" through collision detection and friction, mirroring real-world mechanics. However, this approach was swiftly discarded due to its potential interference with the movements of the robotic arm joints. To elucidate, the Unreal Engine does not directly manage the forces generated in the 3D environment in conjunction with the joint movements orchestrated by MATLAB. Consequently, issues such as clipping between components or non-functionality emerged.

Subsequently, a second option was conceptualized. This alternative approach is as follows: While the grasping maneuver is computed by the system's logic, the actual coordinates of the items are retained to initialize their positions. Thus, when the logic identifies an object and calculates the end configuration of the manipulator for

grasping, we can concurrently determine the homogeneous transformation between the end effector and the object using the following expression:

$$^{e}T_{obj} = {}^{e}T_{b} \cdot {}^{b}T_{obj} \qquad 9\text{-}1$$

Where:
The homogeneous transformation between the end effector and the base relative to the end effectors frame: $^{e}T_{b} = \left({}^{b}T_{e}\right)^{-1}$
The homogeneous transformation between the base and the object relative to the base frame: $^{b}T_{obj}$

Hence, to facilitate the grasping process, it is essential to temporarily fix both the position and orientation of the object in relation to the end effector during the duration of the grasping operation. This entails maintaining a consistent homogeneous transformation between the end effector and the grasped object. We note that in order for this to be possible in all cases the homogeneous transformation is converted between the coordinate systems of MATLAB and Simulink.

To implement this, a specialized function within Simulink was devised to address this specific challenge. This function incorporates a loop mechanism designed to sustain the objects' positions (initial positions) as constant until they are designated as "grabbed" by the logical system. In this manner, the custom Simulink function ensures the stability and fixation of the object's location and orientation during the grasping phase. Lastly when the object is no longer grasped it retains in its last position by utilizing the looped coordinates.

To simulate the closing position of the gripper, primarily for visualization purposes, the development of another tailored function became imperative. The rationale behind this additional step draws from real-world scenarios where, upon grasping an item, the gripper and the objects come into contact.

To emulate this process within the simulation, a custom-made function was created. It operates by utilizing the "Hit" event as an output on the gripper hands. Consequently, when the gripper reaches the grasping position, both the left and right "fingers" of the gripper commence closing with linear velocity. By doing so, the closing of each finger can be independently halted when a hit event is registered. This method effectively achieves a visual representation of the gripping mechanism in action. Figure 9-4 shows an example of the grasping visualization.

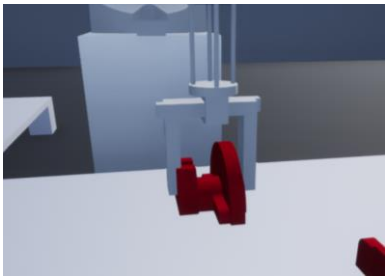


Figure 9-4: Grasping Visualization.

9.5 Full model Simulation

The final stage in the virtual world implementation involves simulating the entire process from initiation to conclusion. The significance of this simulation lies primarily in two aspects: firstly, it serves as a comprehensive demonstration of the entire process that would transpire in the real world, thereby validating the models developed; and secondly, it provides an opportunity to rectify any errors in the logic or the code. A flow diagram of the entire process is presented in Figure 9-5, illustrating how the simulation integrates all the models generated throughout this study.

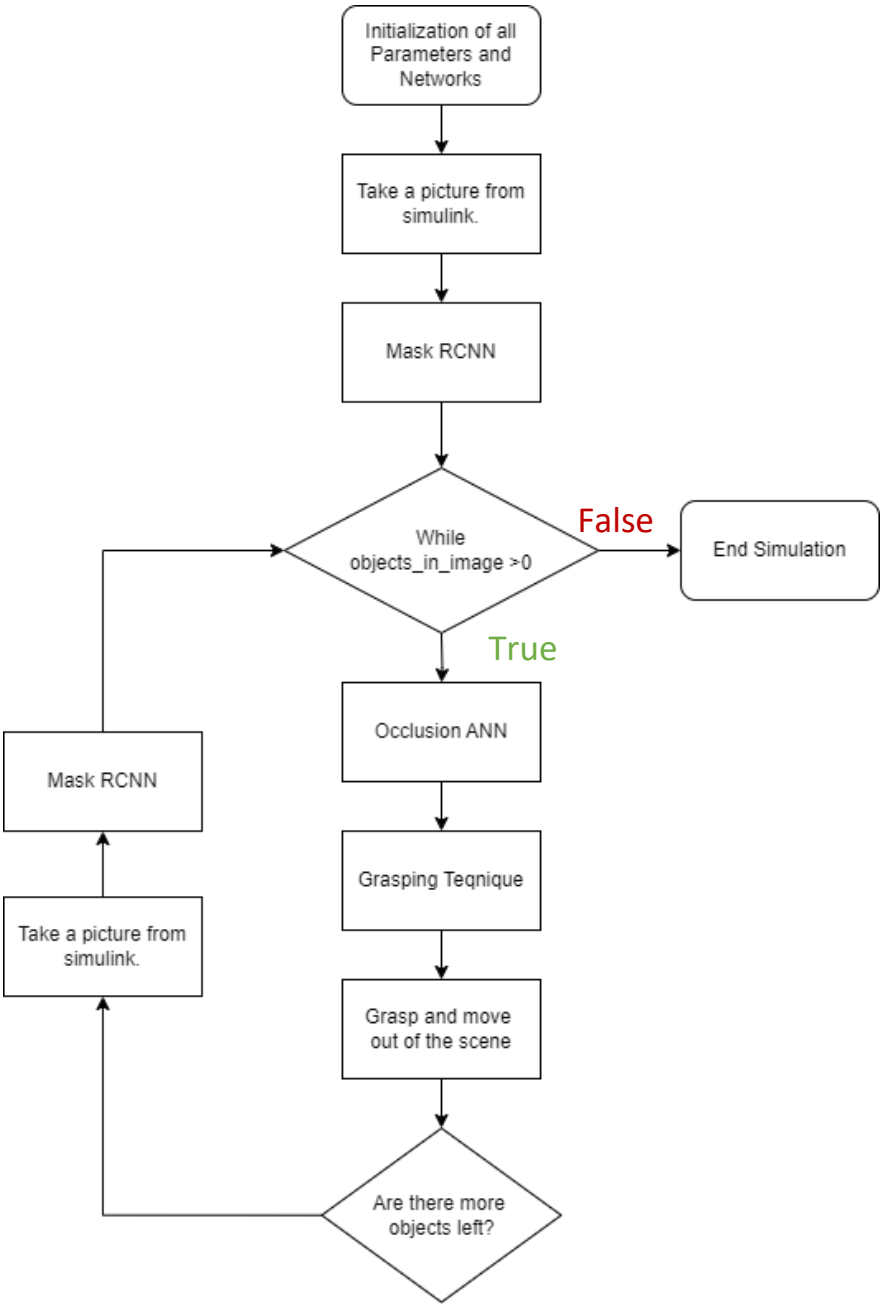


Figure 9-5: Flow diagram of the simulation model implemented.

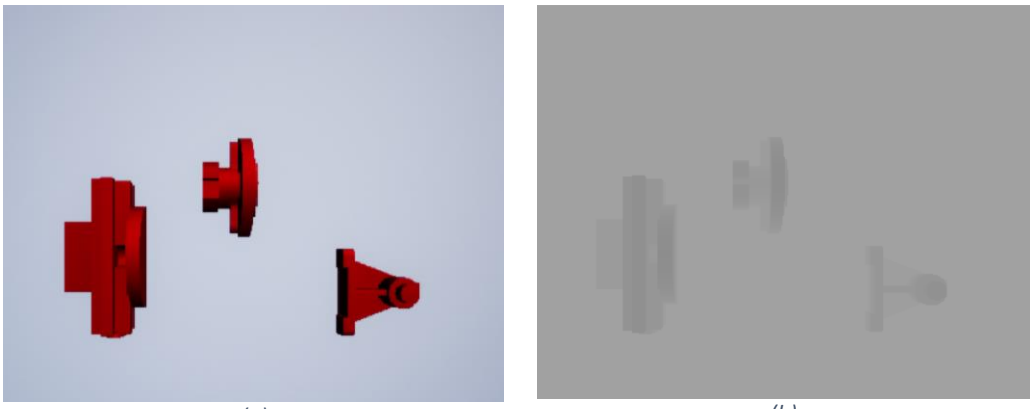
To provide a more detailed insight into the procedural intricacies, a comprehensive delineation of each step is presented below:

- **Initialization of All Parameters and Networks**

This initial step involves loading essential parameters, the robot assembly, function directories, as well as the Mask RCNN and occlusion neural networks. The completion of this step requires approximately 5 seconds.

- **Capture Image from Simulink**

Opening the Simulink model for a mere 0.1 seconds facilitates the capture of an image using the virtual camera, storing both depth and RGB data. However, this step extends to around 8 seconds due to the initiation and opening of the Unreal Engine world.



(a) (b)

Figure 9-6: Initial Pictures taken from both sensors.

- **Mask RCNN**

Employing the RGB image obtained from the virtual camera, the Mask RCNN network generates labels, bounding boxes, and masks for detected objects. Notably, a pop-up window displays the results.

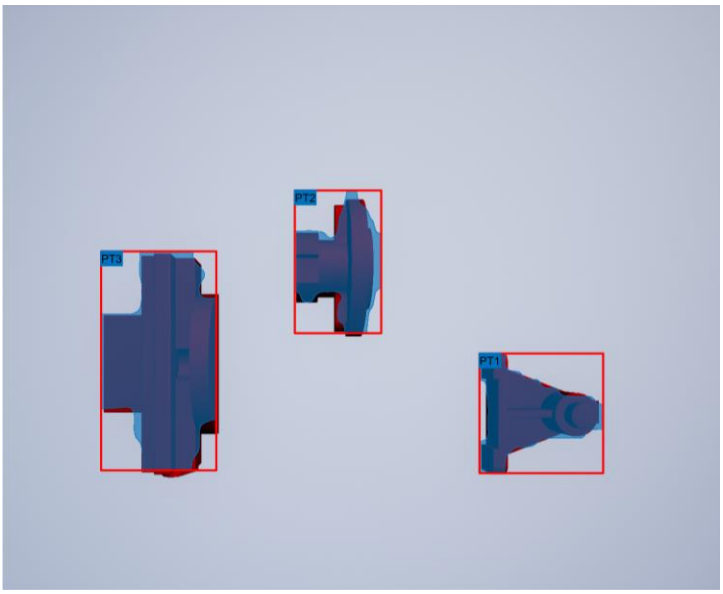


Figure 9-7: Object Detection & Segmentation by Mask RCNN.

- **Occlusion ANN**

Utilizing the masks and labels from Mask RCNN along with the depth image from the Kinect, features are created for each object, and then the Occlusion ANN generates binary indicators of occlusion for the specified objects. This step occurs nearly instantaneously (less than 0.5 seconds).

- **Grasping Technique**

Combining masks, depth image, and the output from the Occlusion ANNs, this step involves the automatic selection of objects to grasp. Subsequently, positions and orientations for grasping are generated using the most efficient path to the end solution, requiring approximately 1 second.

- **Grasp and Move Out of the Scene**

Generating robot configurations, including approach, grab, depart, approach end position, detach, depart, and initial position, is achieved through inverse kinematics, depth data, and masks. Additionally, binary indicators for grasping validity are supplied for each configuration. The simulation duration is around 20 seconds, contingent on the configured times for the robot to reach each configuration. This process repeats until no objects are detected by the Mask RCNN network, concluding the simulation with a notification to the user. Lastly it is noted that the simulation was completed on my own personal computer with characteristics depicted in the appendix. In the following Figures the procedure for one loop of the simulation is elucidated where “PT3” was selected.

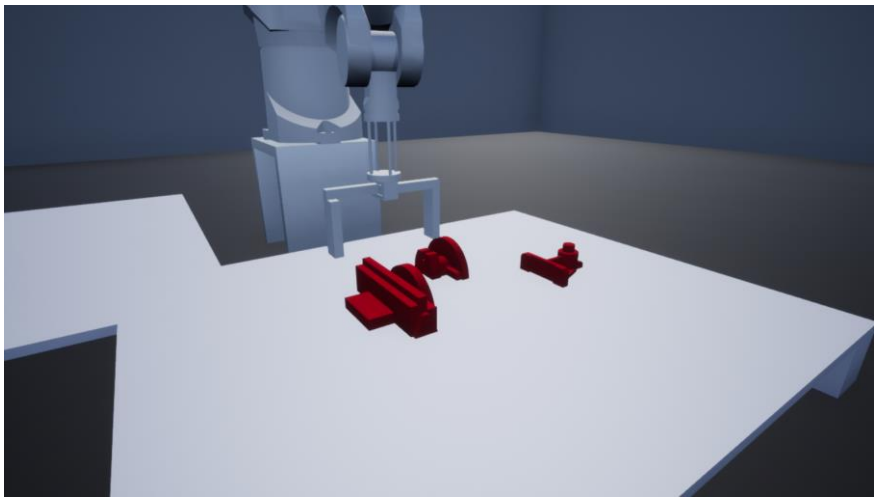


Figure 9-8: Approach position.

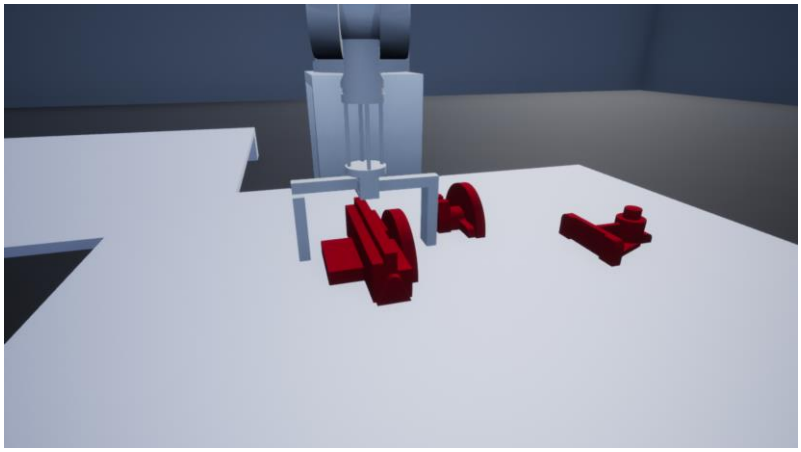


Figure 9-9: Grab Position.

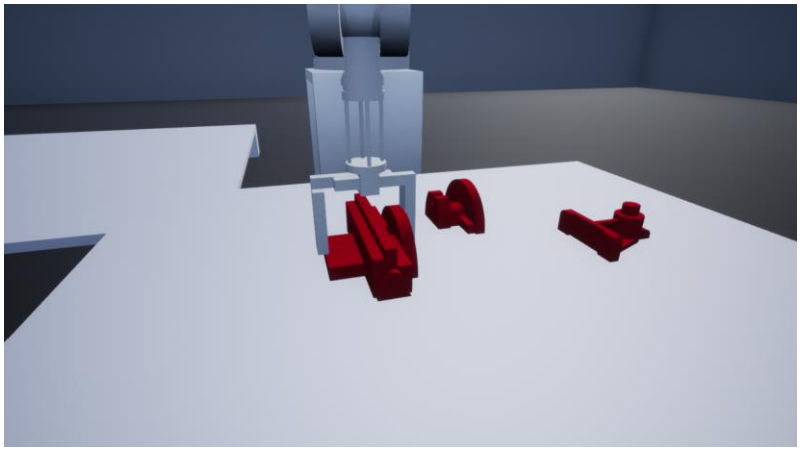


Figure 9-10: Gripper Closed.

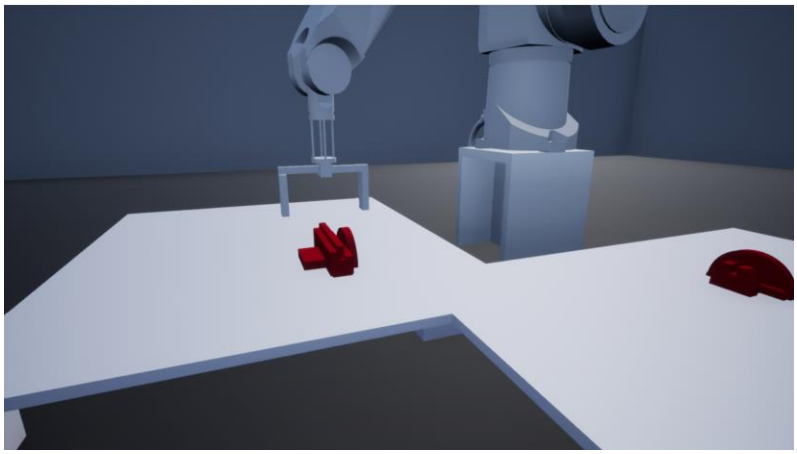


Figure 9-11: Detach Position.

A video of the simulation example can be found at the following link: https://ntuagr-my.sharepoint.com/:v:/g/personal/mc19098_ntua_gr/ET8tc6sprSdJvt_Ezgmybm8BIFP8IXQMy33FBxDUm-ZB4A?nav=eyJyZWZlcnJhbEluZm8iOnsicmVmZXJyYWxBcHAiOiJPbmVEcml2ZUZvckJ1c2luZXNzIiwicmVmZXJyYWxBcHBQbGF0Zm9ybSI6IldlYiIsInJlZmVycmFsTW9kZSI6InZpZXciLCJyZWZlcnJhbFZpZXciOiJNeUZpbGVzTGlua0NvcHkifX0&e=SmhL6m

10. Discussion

The culmination of this work necessitated expertise spanning three distinct domains: Robotic Systems, Visualization Methods, and Machine Vision. While the foundational knowledge in these domains was acquired during my academic tenure in the School of Mechanical Engineering, the completion of this diploma thesis demanded additional insights and specialized understanding acquired during its progression. It is crucial to underscore that while the developed system exhibits commendable performance, there remains room for refinement. Notably, challenges are most prominent in scenarios involving occlusion within the scene-a subject currently under global investigation by numerous researchers. The occlusion problem poses significant implications for robotics in various industries and addressing it with human-like accuracy could pave the way for a fully automated future.

As delineated in the related work section, some researchers integrate depth data with RGB images using the Mask RCNN network to enhance segmentation capabilities. Unfortunately, within MATLAB, such fusion of depth data through the network is not feasible, potentially resulting in a loss of performance. To mitigate this limitation, alternative implementations in environments like Python, where pre-existing code for these network types is available, or the creation of a custom Mask RCNN within MATLAB, were considered. However, both approaches would entail substantial implementation efforts, with uncertain performance improvements as showcased in the section testing the network.

Regarding occlusion-handling methods, while other approaches exist within the Mask RCNN network and involve additional CNNs, the method emphasized in this thesis stands out for its simplicity, directness, and speed. However, its performance is coupled with the performance of the Mask RCNN network which in turn may result in bad prediction if masking is not accurate enough. Avenue for potential enhancement (of this method) lies in incorporating additional inputs or changing the existing ones.

11. Conclusions

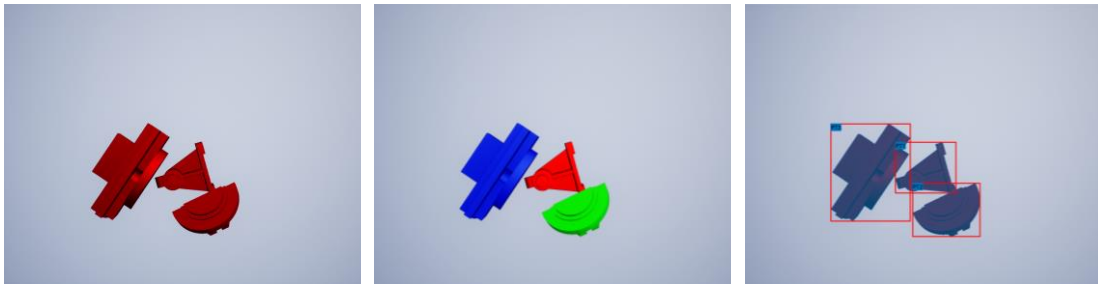
In conclusion, this diploma thesis presents a comprehensive methodology for constructing a robotic arm–machine vision system capable of addressing challenges related to occlusion and incorporating a sophisticated grasping technique. The entire process, from data generation for training to the implementation within MATLAB-Simulink and the Unreal Engine environment, is elucidated. By implementing both augmented and real world ground truth images a relatively high segmentation performance is achieved compared to [8], where in the presented model a total loss of 0.025 is achieved where as in [8] a total loss of 0.08 was achieved. Furthermore, a completely different approach to handling occlusion was used, were features extracted from the depth images and the masks are used as inputs to the occlusion ANNs. The performance in testing showed a success rate of 90.5% in finding occlusion occurrences compared to 68.3% managed by [14]. The grasping of the objects is handled by a logic-based method that is fast and effective. The same can be said about the control and the trajectory planning of the robotic arm, utilizing the depth data acquired from the depth sensor. Lastly the whole process is simulated in a virtual environment utilizing the high graphics capabilities of the Unreal Engine showcasing the system's ability in actual scenarios.

Therefore, this work contributes to democratizing access to advanced machine vision techniques coupled with robotic systems, fostering further exploration by the broader research community. A notable aspect of this thesis is the explicit demonstration of a systematic approach to training advanced neural networks, filling a potential gap in documentation for generating data required by segmentation networks. Emphasis is placed on the efficiency gained through GPU-accelerated training. The methodology for calibrating the Kinect camera and mapping RGB images to depth images is detailed, underscoring precision in the process. Importantly, the adaptability of the trained network is highlighted, as it can successfully segment both real-world and simulated images. This flexibility opens avenues for bidirectional applications, allowing a network trained with computer-generated images to identify objects in real-world images.

12. Future Work

Numerous avenues exist for extending the scope of the work presented in this diploma thesis, starting with the utilization of depth data within the Mask RCNN network to augment its performance. This approach is elaborated upon in the discussion section, where Python emerges as the preferred tool for network generation. While Mask RCNN has been widely employed by researchers for segmentation tasks, recent advancements in networks like YOLO V5 & V8 have demonstrated exceptional capabilities in similar tasks. Consequently, integrating these networks into the framework could potentially enhance segmentation performance further. Expanding the thesis to explore these possibilities could yield valuable insights and contribute to advancing the field of machine vision and segmentation methodologies.

Another aspect not addressed pertains to the generation of computer-generated images for training the Mask RCNN and occlusion networks. Based on the writer's limited knowledge of the subject, this objective could potentially be achieved within the Unreal Engine environment. Specifically, if the random position generator, as outlined in the Virtual World Implementation section, were operational (which may be the case in future iterations), or if the physics engine of Blender were utilized to generate random positions, different colors could be assigned to individual objects. Consequently, employing a color segmentation technique would enable the creation of masks. By capturing two images per object setup—one depicting the actual colors of the objects and another featuring distinct colors—masks (and consequently bounding boxes) could be generated from the latter, while the former would provide RGB and depth images for training the networks.



(a) *(b)* *(c)*

Figure 12-1: Computer Generated Images, Image for training (a), Segmentation image (b), Labeled image (c).

Another pivotal facet of this diploma thesis would have involved the practical simulation of the methodologies elucidated in a real-world scenario. Regrettably, technical constraints precluded the utilization of the robotic arm for this purpose. To provide a comprehensive overview of the prospective implementation, a simplified outline of the methodology required for this endeavor is presented below:

1. **Serial Communication**

   Establish a serial communication link between MATLAB and the CS7 module situated near the robotic arm but beyond its physical reach for control purposes.

Leveraging serial communication within MATLAB, as successfully demonstrated in prior works within the manufacturing department, is crucial for interfacing with the specific robotic arm in question.

2. Control of the Arm

Given that the Stäubli RX90L employs the V+ language for operation, control of the robotic arm can be achieved by transmitting desired coordinates in V+ code. This language facilitates the manipulation of the robotic arm in a manner analogous to its behavior within the simulation environment.

3. System Calibration

A fundamental prerequisite is the calibration of the camera-robotic arm system to determine the homogeneous transformation between them. Employing an eye-in-hand system, an object of known dimensions is positioned at the robotic arm, and a series of measurements are conducted in various robotic arm configurations to facilitate accurate calibration.

4. Implementation

After completing the aforementioned steps, the Simulink model can seamlessly transition from the virtual environment to the real-world counterpart with minimal reconfiguration. The same methodology developed in simulation can be applied to the physical robotic arm setup.

List of Tables

List of Figures

References

[1] Murphy, K. P. *Machine Learning: A Probabilistic Perspective*. The MIT Press, Cambridge, Massachusetts, 2012.

[2] Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. https://doi.org/10.48550/arXiv.1506.01497

[3] Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick. Mask R-CNN, 24 Jan 2018 https://doi.org/10.48550/arXiv.1703.06870

[4] Ramisa, A., Alenyà, G., Moreno-Noguer, F., & Torras, C. (2014). Learning RGB-D descriptors of garment parts for informed robot grasping. Engineering Applications of Artificial Intelligence, 35, 246–258. https://doi.org/10.1016/j.engappai.2014.06.025

[5] Zeng, A., Song, S., Yu, K.-T., Donlon, E., Hogan, F. R., Bauza, M., Ma, D., Taylor, O., Liu, M., Romo, E., Fazeli, N., Alet, F., Chavan Dafle, N., Holladay, R., Morona, I., Nair, P. Q., Green, D., Taylor, I., Liu, W., … Rodriguez, A. (2019). Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching. The International Journal of Robotics Research, 41(7), 690–705. https://doi.org/10.1177/0278364919868017

[6] Hongkun Tian, Tianhai Wang, Yadong Liu, Xi Qiao, Yanzhou Li, Computer vision technology in agricultural automation —A review, Information Processing in Agriculture, Volume 7, Issue 1, 2020, Pages 1-19, ISSN 2214-3173, https://doi.org/10.1016/j.inpa.2019.09.006

[7] Girshick, R.B., Donahue, J., Darrell, T., & Malik, J. (2013). Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 580-587. https://doi.org/10.48550/arXiv.1311.2524

[8] Tuan-Tang Le, Trung-Son Le, Yu-Ru Chen, Joel Vidal, Chyi-Yeu Lin, 6D pose estimation with combined deep learning and 3D vision techniques for a fast and accurate object grasping, Robotics and Autonomous Systems, Volume 141, 2021, 103775, ISSN 0921-8890, https://doi.org/10.1016/j.robot.2021.103775.

[9] Y. Inagaki, R. Araki, T. Yamashita and H. Fujiyoshi, "Detecting layered structures of partially occluded objects for bin picking," *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Macau, China, 2019, pp. 5786-5791, doi: 10.1109/IROS40897.2019.8968093.

[10] Hongkun Tian, Kechen Song, Song Li, Shuai Ma, Jing Xu, Yunhui Yan, Data-driven robotic visual grasping detection for unknown objects: A problem-oriented review, Expert Systems with Applications, Volume 211, 2023, 118624, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2022.118624

[11] Zhen Li, Benlian Xu, Di Wu, Kang Zhao, Siwen Chen, Mingli Lu, Jinliang Cong, A YOLO-GGCNN based grasping framework for mobile robots in unknown environments, Expert Systems with Applications, Volume 225, 2023, 119993, ISSN 0957-4174, https://doi.org/10.1016/j.eswa.2023.119993

[12] R Aarthi., G Rishma., A Vision Based Approach to Localize Waste Objects and Geometric Features Exaction for Robotic Manipulation, Procedia Computer Science, Volume 218, 2023, Pages 1342-1352, ISSN 1877-0509, https://doi.org/10.1016/j.procs.2023.01.113.

[13] Silviu Răileanu, Theodor Borangiu, Florin Anton, Silvia Anton, Open source machine vision platform for manufacturing and robotics, IFAC-PapersOnLine, Volume 54, Issue 1, 2021, Pages 522-527, ISSN 2405-8963, https://doi.org/10.1016/j.ifacol.2021.08.060.

[14] Ziyad Tareq Nouri Master thesis

[15] Automate Virtual Assembly Line with Two Robotic Workcells. MATLAB & Simulink. (n.d.). https://www.mathworks.com/help/robotics/ug/automate-virtual-assembly-line.html
[16] Craig, J. J. (2022). Introduction to robotics: Mechanics and control. Pearson Education Limited.
[17] Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2010). Robotics: Modelling, planning and control. Springer.
[18] Pieper, Donald. The Kinematics of Manipulators Under Computer Control. Stanford University, 1968.
[19]François Chaumette, S.Hutchinson. Visual servocontrol, PartI & II: Basicapproaches. IEEE Robotics and Automation Magazine, 2006, 13(4), pp.82-90. inria-00350283.
[20]MaskRCNN. Train Mask R-CNN network to perform instance segmentation - MATLAB. (n.d.). https://www.mathworks.com/help/vision/ref/trainmaskrcnn.html

Appendix

I. MATLAB .m Files

Matlab_Simulink_Control.m

```
clc;
clear;
%% INPUTS
%add chp funcitons
addpath('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\Mask
CNN\CHP_Functions');
%Insert the urdf file of the robotic arm
robot_urdf_dir ="B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\Cad
Models\URDF\Staubli RX90.SLDASM\urdf\Staubli RX90.SLDASM.urdf";
%Insert the initial position of the robotic arm
Initial_Config = zeros(1,8);
%Insert Detector directory (TrainedDetector)
load('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\Mask
CNN\TrainedDetectors\test7.2.mat');
imageSize = [346 512 3]; %height-width (needed for detection purposes) [346
512 3]
%Depth sensor resolution
D_res = [340,417];
%Insert Simulink 3d World Directory File
simulink_dir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Testing_Images\SIMULATION.slx';
%Insert ANNS
load('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\Mask
CNN\Best1_occlusion_net.mat');
net1 = best_net;
load('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\Mask
CNN\Best2_occlusion_net.mat');
net2 = best_net;
load('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\Mask
CNN\Best3_occlusion_net.mat');
net3 = best_net;
addpath('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect');
%Insert Kinect (camera) position and orientation,
%additional camera parameters are set inside simulink make sure to change
%them if needed!
%Destances from base to camera
xcamera = 0.75;
ycamera = -0.0;
zcamera = 0.4;
%camera Rotation, vector outwards from camera
Camera_Rotation = [180,90,0];
%Camera Transformation
Rcamera = eul2rotm(Camera_Rotation*pi/180,"YZX"); %YXZ
Camera_Position = [xcamera,ycamera,zcamera];
T_base_camera= eye(4);
T_base_camera(1:3,1:3) = Rcamera;
T_base_camera(1:3,4)=(Camera_Position)';

f = 345; %focal length of D_camera fx = fy (almost);
cx = D_res(2)/2; %center point of depth camera;
cy = D_res(1)/2; %center point of depth camera;
```

```
% Be carful optical center is in form of [x,y] while image resolution is in
% form of [h,w] in simulink camera parameters.

%% Connect to the kinect
% colorDevice = imaq.VideoDevice('kinect',1);
% depthDevice = imaq.VideoDevice('kinect',2);
%% Take a Picture with Kinect
% [im_rgb,im_d]=KinectPicture(colorDevice,depthDevice);
% [rgb,d]=Kinect_RGBtoDepthMap(im_rgb,im_d);
% d_show = uint16_to_uint8(d);
% figure(1)
% montage({rgb,d_show});

%% Initialize robot
%Use simulink for the 3d world representation
%Use matlab to solve the problem

%import the robotic arm
Staubli_RX90L=importrobot(robot_urdf_dir,"urdf");
% Set the kinematic groop to the specified links
Kinematic_group =
struct(convertStringsToChars('BaseName'),'base_link',convertStringsToChars(
'EndEffectorBodyName'),'tool0');
% Solve the inverse kinematics by creating the robotIK function
%
https://www.mathworks.com/help/robotics/ref/analyticalinversekinematics.htm
l
invkin =
analyticalInverseKinematics(Staubli_RX90L,"KinematicGroup",Kinematic_group)
;
analsol = generateIKFunction(invkin,'robotIK');
% Set the data format to row manualy due to matlab restrictions
Staubli_RX90L.DataFormat='row';
%% Initialize Item Locations
%In UE coordinate system:
Item_Initial_Locs=[0.8,-0.2,-0.227; %item 1 [x,y,z]
    0.65,0.0,-0.227; %item 2
    0.75,0.15,-0.228];%item 3
Item_Initial_Rots = zeros(3);
Item_End_Locs=[-0.2,0.7,-0.227; %item 1 [x,y,z]
    0,0.7,-0.227; %item 2
    0.2,0.7,-0.228];%item 3
Item_End_Rots = zeros(3);

%% Parameters for simultion initialization
%(these do not matter for the initialization but are needed to use the
simulink model)
Item_to_grab=1;
end_effector_part_transform=zeros(4);
Gripper_Position2=0;
Gripper_Position1=0;

%% Parameters for simulation
VEL_BC = zeros(8,2); % velocity of robot at each configuration
Grab_time=30;
Letgo_time = 70;
Configs_Time=[0,20,Grab_time,40,60,Letgo_time,80,100];
%Camera Positions for Visulization
```

```
Camera_take_picture_loc = [1.699,0,-0.3581];
Camera_Hide_Loc = Camera_take_picture_loc+[0.75,0,0];
Camera_Locations_Visulization =
[Camera_take_picture_loc;Camera_Hide_Loc;Camera_Hide_Loc;Camera_take_pictur
e_loc]';
Camera_Speed_Visulization = zeros(3,size(Camera_Locations_Visulization,2));
Camera_Times_Visulization = [5,10,90,95];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%

%% Robot Controll with matlab - simulink

%% Take a Picture with simulink
% Robot conficurations
Robot_Configs = [zeros(1,8);zeros(1,8)]';
%Velocity at each configuration
VEL_BC = zeros(8,2); %2 extra prismatic joints for grippers
%Time stamps to achieve the configurations
Configs_Time = [0,20];
% run the simulation for 0.1 sec to just get a picture
out = sim(simulink_dir,0.1);
%RGB image:
rgb = out.Image.signals.values;
%Depth image:
d = out.Depth.signals.values; % in this way we get distance in doubles
which is giving a value of meters
d=imresize(d,D_res); %change to actual depth resolution
%visualize images
figure(1)
montage({rgb,d})
% [x,y] =ginput(1)
d=d*1000; %to get the depth values in mm
%% Detection (mask RCNN)
[masks,labels, scores, bboxes] = segmentObjects(TrainedDetector,rgb);
%% visualize Detection
overlayeredImage = insertObjectMask(rgb,masks);
figure(3)
imshow(overlayeredImage)
hold on
showShape("rectangle",bboxes,"Label",labels,'LineColor',[1,0,0])
itemsonimage=size(labels,1);
% Change to match depth image resolution
[rgb,masks,bboxes]=ResizeImageMasksBoxes(rgb,masks,bboxes,D_res);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
while itemsonimage>0 %% Initiation of the loop

    %% Matlab coodrinate system transforms
    totalnumberofitems=size(Item_Initial_Locs,1);
    T_items_initial = zeros(4,4,totalnumberofitems);
    T_items_end = zeros(4,4,totalnumberofitems);
    for i=1:totalnumberofitems

T_items_initial(:,:,i)=UEtoMATLABtransform(Item_Initial_Locs(i,:),Item_Init
ial_Rots(i,:));

T_items_end(:,:,i)=UEtoMATLABtransform(Item_End_Locs(i,:),Item_End_Rots(i,:
));
```

```
    end
    %% Grasping Teqnique
    %%Gennerate Occlusion detection for all objects
    occlusion=Occlusion_Detector(net1,net2,net3,d,masks,labels);
    %Gennerate the grasping information for the object to be grabbed next

[itemcenter,itemorientation,itemz_grab,Item_to_grab,Gripper_Position2,Gripp
er_Position1] = Grab_Selection(masks,d,occlusion,f);
    %The name of the object to be grabbed next
    Item_to_grab_name = labels(Item_to_grab)
    %set the index for the simulation
    if Item_to_grab_name=="PT1"
        Item_to_grab=1;
    elseif Item_to_grab_name=="PT2"
        Item_to_grab=2;
    elseif Item_to_grab_name=="PT3"
        Item_to_grab=3;
    end

    %% Object world Transformation
    %Calculating X and Y diff from center of camera to the center of the
object
    X_item = (itemz_grab)*(itemcenter(1)-cx)/f;
    Y_item = (itemz_grab)*(itemcenter(2)-cy)/f;
    Z_item = itemz_grab;
    R_item = R_z(-itemorientation); % - sign is to change from image frame
to world frame
    % Item homogeneous transform from camera world frame to item world
frame
    T_camera_item = eye(4);
    T_camera_item(1:3,1:3)=R_item;
    T_camera_item(1:3,4)=[X_item;Y_item;Z_item];
    % Homogeneous transfrom from base to item
    T_base_item = T_base_camera*T_camera_item;

    %% Inverse Kinematics & Trajectories
    %Approach Transformation from end solution (z=10cm)
    T_approach_diff = [0,0,0,0;0,0,0,0;0,0,0,0.1;0,0,0,0];

    %Grab Approach
    T_base_item_approach=T_base_item+T_approach_diff;
    ikConfig = robotIK(T_base_item_approach);
    Robot_approach = best_sol(Initial_Config(1:6),ikConfig);

    %Grab
    clear ikConfig
    ikConfig = robotIK(T_base_item);
    Robot_Grab = best_sol(Robot_approach(1:6),ikConfig);
    end_effector_part_transform =
inv(T_base_item)*T_items_initial(:,:,Item_to_grab);% transform between the
grabbing point and the part coordinate system
%getTransform(Staubli_RX90L,Robot_Grab,'base_link','tool0')

    %Detach Approach
    clear ikConfig

T_base_letgo=T_items_end(:,:,Item_to_grab)*inv(end_effector_part_transform)
; % to get the letgo position
```

```
    T_base_letgo_approach =T_base_letgo+T_approach_diff;
    ikConfig = robotIK(T_base_letgo_approach);
    Robot_letgo_approach = best_sol(Robot_Grab(1:6),ikConfig);

    %Detach
    clear ikConfig
    ikConfig = robotIK(T_base_letgo);
    Robot_letgo = best_sol(Robot_letgo_approach(1:6),ikConfig);

    %Set the configurations and timings for simulation
    Robot_Configs =
[Initial_Config;Robot_approach;Robot_Grab;Robot_Grab;Robot_approach;Robot_l
etgo_approach;Robot_letgo;Robot_letgo_approach;Initial_Config]'; % Robot
configuration
    VEL_BC = zeros(8,size(Robot_Configs,2));
    Configs_Time= [0,20,Grab_time,40,50,60,Letgo_time,80,100];

    %Simulate
    out = sim(simulink_dir,100);
    I_rgb = out.Image.signals.values;
    I_d = out.Depth.signals.values;

    %Set new locations for items
    Item_Initial_Locs(Item_to_grab,:)=Item_End_Locs(Item_to_grab,:);
    Item_Initial_Rots(Item_to_grab,:)=Item_End_Rots(Item_to_grab,:);

    %% Take a Picture with simulink
    % Robot conficurations
    Robot_Configs = [zeros(1,8);zeros(1,8)]';
    %Velocity at each configuration
    VEL_BC = zeros(8,2); %2 extra prismatic joints for grippers
    %Time stamps to achieve the configurations
    Configs_Time = [0,20];
    %Run the simulation for 0.1 sec to just get a picture
    out = sim(simulink_dir,0.1);
    %RGB image:
    rgb = out.Image.signals.values;
    %Depth image:
    d = out.Depth.signals.values; % in this way we get distance in doubles
which is giving a value of meters
    d=imresize(d,D_res); %change to actual depth resolution
    %visualize images
    figure(1)
    montage({rgb,d})
    % [x,y] =ginput(1)
    d=d*1000; %to get the depth values in mm
    %% Detection (mask RCNN)
    [masks,labels, scores, bboxes] = segmentObjects(TrainedDetector,rgb);
    %% visualize Detection
    overlayeredImage = insertObjectMask(rgb,masks);
    figure(3)
    imshow(overlayeredImage)
    hold on
    showShape("rectangle",bboxes,"Label",labels,'LineColor',[1,0,0])
    itemsonimage=size(labels,1);
    % Change to match depth image resolution
    [rgb,masks,bboxes]=ResizeImageMasksBoxes(rgb,masks,bboxes,D_res);
```

```
end

Reply = 'No more objects where Found by the detector'
```

Kinect_Photos.m

```
%% INITIALIZATION (Run once)
% clc;
% clear;
%Directory to store High quallity mapped images
RGBHighQuallity_dir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10.
Diplomatikh\matlab code\Kinect\Testing\RGBHigh Mapped';
%Directory to store mapped RGB images with the same resolution as depth
images
RGBFolder_dir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\Testing\RGB Mapped';
%Directory to store mapped depth images
DepthFolder_dir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\Testing\Depth Mapped';
%Directory to store raw images
RGB_RAW_dir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\Testing\RGB RAW';
%Directory to store raw depth images
Depth_RAW_dir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\Testing\Depth RAW';
%Directory to store visulization of depth images
Depth_Images_RAW_dir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10.
Diplomatikh\matlab code\Kinect\Testing\Depth Images RAW';
Depth_Images_dir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\Testing\Depth Images Mapped';
%Initialization of kinect sensors
colorDevice = imaq.VideoDevice('kinect',1);
depthDevice = imaq.VideoDevice('kinect',2);

%% Get Images
[img_rgb,img_d]=KinectPicture(colorDevice,depthDevice);
[rgb_c,d_c]=Kinect_RGBtoDepthMap(img_rgb,img_d);
%high resolution mapped image:
rgb = imcrop(img_rgb,[285,39,1272,1037]);

%% Display images
d_show =uint16_to_uint8(d_c);
figure(1)
title('Montage')
montage({rgb_c,d_show});
figure(2)
imshow(rgb)
dr_show = uint16_to_uint8(img_d);
figure(3)
montage({img_rgb,dr_show})

%% Save images
rgbh_dir = strcat(RGBHighQuallity_dir,'\',num2str(i),'.jpg');
imwrite(rgb,rgbh_dir);
rgb_dir = strcat(RGBFolder_dir,'\',num2str(i),'.jpg');
imwrite(rgb_c,rgb_dir);
d_dir = strcat(DepthFolder_dir,'\',num2str(i));
save(d_dir,"d_c")
rgbr_dir = [RGB_RAW_dir,'\',num2str(i),'.jpg'];
imwrite(img_rgb,rgbr_dir);
dr_dir = [Depth_RAW_dir,'\',num2str(i)];
save(dr_dir,"img_d")
```

```
di_dir = [Depth_Images_dir,'\',num2str(i),'.jpg'];
imwrite(d_show,di_dir)
dir_dir = [Depth_Images_RAW_dir,'\',num2str(i),'.jpg'];
imwrite(dr_show,dir_dir)
i=i+1;
```

Dataset_Creation_With_Occlusion.m

```
clc;
clear;
% imageLabeler;

% Use the imageLabeler app to create polygon shaped masks and the bounding
% boxes will be created with the code bellow. export the gTrouth as "table"
% with the name "gTruth".

% In the image labler app you need to specify for each label 2 attributes:

%Item attribute with the name "Item", is a numeric value that identifies
%what number of object is the label for the same class of objects. This is
%needed in case one object was two or more polygon shaped masks (due to
%occlusion). So is two areas are fo the same object then both areas should
%have the same value for the attribute item. Eg 1,2,3,4....

% Occlusion Attribute with the name "Occlusion", is a logical value that
% specifies occlusion of the spesific object if an object is occluded then
% this value should be true, if it is not (or if it occludes other object)
% this value should be falase. If an object has multiple areas it should
% you should manually set the occlusion to all the areas (set to value
"true"
% because an item that is not occluded should not have many areas)
%% INPUTS
% Add path to custom made CHP Functions
addpath('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\Mask
CNN\CHP_Functions')
% Add the source directory to the path (add maskrcnn main functions)
addpath('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\Mask
CNN\mask-rcnn-main\src')
% Set the root directory for COCO API:
cocoAPIDir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Mask CNN\MatlabAPI';
%Set the Image Training Folder (needs to contain images as described in
imageFile above):
trainImgFolder = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\Testing\RGBHigh Mapped'; % where the images are stored
%load the groud truth data
load("B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\Testing\OCCLUSION CLASSES\gTruth_High_Occlusion.mat");%load the
gTruth table
h= 1037; %height of the images
w=1272; %width of the images
%Set the annotation folder where all the .mat files will be stored
AnnotationFolder = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\Testing\OCCLUSION CLASSES\Anottations_High_Occlusion';
%Set the occlison annotation folder where all the .mmat files will be
%stored
OcclusionFolder = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\Testing\OCCLUSION CLASSES\Anottations_High_Occlusion -
Occlusion';
% Set image size for training (you can change that if you need to lower the
res for faster training)
imageSize = [h w 3]; %height-width
mask_image_save_location = "B:\OneDrive - CHP\Σχολή μαθήματα\10.
Diplomatikh\matlab code\Kinect\Testing\OCCLUSION CLASSES\Masks_HO\";
```

```
%% Data Manipulation
Image_Number = size(gTruth,1); % number of images in the table
Object_Number =size(gTruth,2); % number of objects in the table

varnames = gTruth.Properties.VariableNames; % the names are set in the
gTrouth table
varnames= varnames(1,2:end);
blds_cell = cell(Image_Number,1); % Bounding box cell array. Working with
cell is easier (at least for me)
mask_cell = cell(Image_Number,1);
variable_names_cell=cell(Image_Number,1);
occlusion_cell = cell(Image_Number,1);
% the image cell is not needed for the datastore it is only cept in here
% for legacy reasons. The datastore needs only the image file names.
%image_cell = cell(Image_Number,1); %open images inside matlab and store
them inside image_cell for concatenation later
for n=1:Image_Number
    i=1; %goes from [1 to Total_items]
    clear masks_on_image var_name_categorical% delete the last atempts
    Total_items=0;
    for ni=2:Object_Number
        if isempty(gTruth{n,ni}{1,1})==0
            Total_items=Total_items+gTruth{n,ni}{1,1}(end).Item;
        end
    end
    masks_on_image=false([h,w,Total_items]);
    %total_var_names = 1:Total_items; %to create a categorical array of
size Total_items
    var_name_categorical =
cell(1,Total_items);%categorical(total_var_names);
    occlusion_property = false(1,Total_items);
    bboxes = zeros(Total_items,4);
    for nn=2:Object_Number% object classes are Object_Number - 1
        num_of_object_segments = size(gTruth{n,nn}{1,1},2);
        if isempty(gTruth{n,nn}{1,1})==0
            Objects_in_Image = gTruth{n,nn}{1,1}(end).Item;% find how many
objects of a specific type exist (e.g 2 rubiks cubes)
        else
            Objects_in_Image=0;
        end
        if Objects_in_Image > 0
            BD= zeros(Objects_in_Image,4); %zero-out the matrix and change
it's dimensions per iteration
            %
            Item=1; % to distinguish items of the same type [1 -
Objects_in_Image]
            nnn=1;
            while nnn<=num_of_object_segments

                    mask = false([h,w]); % create the mask for the specific
object
                    if gTruth{n,nn}{1,1}(nnn).Item==Item % every object has
1 or more segments (because of occlusion) therefore we have to create one
mask for all the segments together (of that item)
                        gT_polygon = gTruth{n,nn}{1,1}(nnn).Position; %
find the polygon coordinates inside the ground truth table
                        x = gT_polygon(:,1); % find the x coordinates of
the polygon points
```

```
                        y =gT_polygon(:,2); %find the y coordinates of the
polygon points
                        mask=poly2mask(x,y,h,w); % crete the polygon by
inserting ones (1) where there is an object
                        masks_on_image(:,:,i)=mask+masks_on_image(:,:,i);

occlusion_property(i)=gTruth{n,nn}{1,1}(nnn).Occlusion;
                    end
                    if nnn<num_of_object_segments %generate for every
objects its properties
                        if gTruth{n,nn}{1,1}(nnn+1).Item ~= Item
                            var_name_categorical(i) = varnames(nn-1);
                            bboxes(i,:) =
BoundingBox_From_Mask(masks_on_image(:,:,i)); %Create the vertical
rectangle (Bounding box) that encapsulates all the polygon points

occlusion_property(i)=gTruth{n,nn}{1,1}(nnn).Occlusion;

imwrite(masks_on_image(:,:,i),strcat(mask_image_save_location,num2str(n),nu
m2str(nn-1),num2str(nnn),'.png')) % write the image to a file
                            Item=Item+1;
                            i=i+1;
                        end
                    else %nnn==num_of_object_segments
                            var_name_categorical(i) = varnames(nn-1);
                            bboxes(i,:) =
BoundingBox_From_Mask(masks_on_image(:,:,i)); %Create the vertical
rectangle (Bounding box) that encapsulates all the polygon points

occlusion_property(i)=gTruth{n,nn}{1,1}(nnn).Occlusion;

imwrite(masks_on_image(:,:,i),strcat(mask_image_save_location,num2str(n),'_
',num2str(nn-1),'_',num2str(nnn),'.png')) % write the image to a file
                            Item=Item+1;
                            i=i+1;
                    end
                    nnn=nnn+1;

            end

            %blds_cell{n,nn-1} = BD; % instert bounding boxes in the
bounding box cell array
        end
    end
    %image_cell{n}= imread(gTruth{n,1}{1,1});
    occlusion_cell{n} = occlusion_property;
    blds_cell{n}=bboxes;
    mask_cell{n}=masks_on_image;
    variable_names_cell{n}=var_name_categorical';
end


[~,image_names,ext] = fileparts(gTruth{:,1});
image_names = strcat(image_names,ext);%get the name and the type (.jpg)

TrainingData = [image_names,blds_cell,variable_names_cell,mask_cell];

%% Create a different .mat file for every image
```

```
%In the annotation folder there need to be only .mat files the names does
%not matter but in this case we save them as 1.mat,2.mat,3.mat,4.mat.. etc
%for mask rccn training:
for n=1:Image_Number
 mat_to_save = TrainingData(n,:);
 %save averything in a different variable so as when the mat files are
 %loaded the open up 4 different variables with the names specified in the
 %save function:
 imageFile = mat_to_save{1}; % it is the image file name eg. '0001.png'
(string)
 boxes = mat_to_save{2}; % it is a 2d array (double) containing bounding
boxes in the form of: Mx4 (where M is the objects in image '0001.png')
 labels = categorical(mat_to_save{3}); % its is a categorical array
containing the categories for each bounding box Mx1
 masks = mat_to_save{4};% its a 3d array containing the masks as
HeightxWidthxM (logical) for each object

save(strcat(AnnotationFolder,'\',num2str(n)),"imageFile","boxes","labels","
masks"); % the names play a very big role check: cocoAnnotationMatReader.m
and put the correct names there
 occlusion= occlusion_cell{n};
 save(strcat(OcclusionFolder,'\',num2str(n)),"occlusion");
end


%% Save Variable Names
item_names = varnames(1:end);
save('item_names','item_names');
%% Using the coco api to create the datastores (because matlab
imagedatastore does not support multi image masks)
%% Using code from MaskRCNNTrainingExample.mlx
% COCO-MATLABAPI: https://github.com/cocodataset/cocoapi
% Add the API directory to the path
addpath(cocoAPIDir);
% Create the training datastore to read image and ground truth data from
% the unpacked annotation MAT file
ds =
fileDatastore(AnnotationFolder,'ReadFcn',@(x)helper.cocoAnnotationMATReader
(x, trainImgFolder));
%% OUTPUT
trainDS = transform(ds, @(x)helper.preprocessData(x, imageSize));
save('trainDS','trainDS'); % save the training datastore to use for the
training m file
%% Test the datastore!
trainDS.shuffle();
%preview the datastore needs to be in form of 1x4 cell array like this:
%{512×512×3 uint8}    {7×4 double}    {7×1 categorical}    {512×512×7
logical}
data = preview(trainDS) % done
%% visualise an image with its bounding box
%load("1.mat")
% overlayeredImage = insertObjectMask(test_img,masks);
% figure(2)
% imshow(overlayeredImage)
% hold on
% showShape("rectangle",boxes,"Label",labels,'LineColor',[1,0,0])
```

Image_Augmentation.m

```
clc;
clear;
%% INPUT
addpath('CHP_Functions') %Add the custom made functions
% Add the source directory to the path (add maskrcnn main functions)
addpath('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\Mask
CNN\mask-rcnn-main\src')
% Set the root directory for COCO API:
cocoAPIDir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Mask CNN\MatlabAPI';
% Set the directory where the images to be augmented are located
Images_location = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\CleanImages\HighQuallity\RGBHigh\';
% Set the location of the initial dataset to be augmented
DS_location = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\CleanImages\HighQuallity\trainDS.mat'; % insert the Dataset
created with Datase_Creation.m file
%Set all the item names contained in the images
Item_Names = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Item images\v2\Testing\item_names.mat';
%Set the directory to save the augmented images
trainImgFolder= 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\CleanImages\HighQuallity\Augmented\Images'; % location where
the images will be saved
%Set the directory where the annotatios are going to be stored
Augmented_Images_Annotation_Folder = 'B:\OneDrive - CHP\Σχολή μαθήματα\10.
Diplomatikh\matlab
code\Kinect\CleanImages\HighQuallity\Augmented\Annotations'; % location
where the annotations will be saved
%Set the directory where the Backround images are stored
Background_Images_Location = 'B:\OneDrive - CHP\Σχολή μαθήματα\10.
Diplomatikh\matlab
code\Kinect\CleanImages\HighQuallity\Augmented\Background\JPEG'; % location
of imagedatastore containing the backround images for the augmentation
%Set the ammount of augmented images created from 1 image.
Augmentation_Number = 10; %Ammount of images to create with random
rotations and translations of the Items per image using a random backround
%Set the image  sizes
h= 1042; %height of the images
w=1353; %width of the images
Image_Size = [w,h];%width,height
imageSize=[h,w,3];%height,width
% Set the directory for intermediate images to be stored.
Augmented_Images_Intermediate_Location= 'B:\OneDrive - CHP\Σχολή
μαθήματα\10. Diplomatikh\matlab
code\Kinect\CleanImages\HighQuallity\Augmented\Intemediate\';

%%
% Find the image center
Image_Center= Image_Size/2;
% Create an imagedatastore for the backround images (just to get their
names)
Background_IMDS = imageDatastore(Background_Images_Location);
% Number_of_Backround_Images = size(Background_IMDS,1);
```

```
DS = load(DS_location); % Load the datastore to be augmented
DS=DS.trainDS; % the datastore name should be trainDS for this to work

Number_of_Images = size(DS.UnderlyingDatastores{1,1}.Files,1); % Number of
images in the datastore

for n =1:Number_of_Images % for every image
load(DS.UnderlyingDatastores{1,1}.Files{n,1}) % load masks,
labels,boundingboxes and image name orresponding to an image from the
annotation folder
image_location = strcat(Images_location,imageFile); % directory of the
image
image = imread(image_location); %load the image to be augmented
%save the initial parameters of the annotations corresponding to the image
labels_n=labels;
boxes_n = boxes;
imageFile_n = imageFile;
masks_n=masks;
num_of_items_in_image = size(labels,1); %number of items in the image


    for nn=1:Augmentation_Number
        clear msk mask img boxes labels % clear the anotation parameters
for the augmented images
        masks = masks_n;
        boxes=boxes_n;
        labels=labels_n;
        imageFile = imageFile_n;
        number_of_bg_images = size(Background_IMDS.Files,1); % number of
backround images
        bg_image=floor(rand*number_of_bg_images+1); %pick a random
background image
        IMG = imread(Background_IMDS.Files{bg_image}); %initiate the
backround image

        %i is like nnn but it has randomly sorted the number (for random
occlusion purposes)
        i = 1:num_of_items_in_image;
        i = i(randperm(length(i))); %create an array of length num of items
in image but to have randomly distributed numbers. This number indicates
the occlusion order the i(end) item occludes all the others the i(1) is
occluded by all the other (it is placed first in the image)

            for nnn=1:num_of_items_in_image
                props = regionprops(masks(:,:,i(nnn)),'Centroid'); % find
the centroid of the mask (masks need to have only 1 region)
                centroid= props.Centroid;
                img = image;
                msk = masks(:,:,i(nnn));
                Translation = Image_Center - centroid;
                img = imtranslate(img,Translation,'FillValues',[0,0,0]);
                msk=imtranslate(msk,Translation,'FillValues',0);
                %image augmentation part
                theta = rand*360;
                img=imrotate(img,theta,"bilinear","crop");
                msk=imrotate(msk,theta,"bilinear","crop");
                translate= (rand*Image_Center-rand*Image_Center);  %random
translation
```

```
                img = imtranslate(img,translate,'FillValues',[0,0,0]);
                msk=imtranslate(msk,translate,'FillValues',0);
                saveloc = strcat(Augmented_Images_Intermediate_Location,
num2str(n),'_',num2str(nn),'_',num2str(nnn),'.png');
                Alpha = double(msk);
                imwrite(img,saveloc,'alpha',Alpha)
                masks(:,:,i(nnn)) = msk;
                img_intermediate = imread(saveloc);
                IMG = CHP_Blend(IMG,img_intermediate,msk);
                props=regionprops(masks(:,:,i(nnn)),'BoundingBox');
                boxes(i(nnn),:)=props.BoundingBox;% wrong
            end
        imageFile = strcat(num2str(n),'_',num2str(nn),'.jpg');%mask rcnn
does not support png!
        save_loc = strcat(trainImgFolder,'\',imageFile);
        imwrite(IMG,save_loc);

        [masks,boxes,labels]=Occlusion_Correction(masks,boxes,labels,i,50);

save(strcat(Augmented_Images_Annotation_Folder,'\',num2str(n),'_',num2str(n
n)),"imageFile","boxes","labels","masks");
    end
end


%% Using the coco api to create the datastores (because matlab
imagedatastore does not support multi image masks)
%% Using code from MaskRCNNTrainingExample.mlx
% COCO-MATLABAPI: https://github.com/cocodataset/cocoapi
% Add the API directory to the path
addpath(cocoAPIDir);
% Create the training datastore to read image and ground truth data from
% the unpacked annotation MAT file
ds =
fileDatastore(Augmented_Images_Annotation_Folder,'ReadFcn',@(x)helper.cocoA
nnotationMATReader(x, trainImgFolder));
trainDS = transform(ds, @(x)helper.preprocessData(x, imageSize));
save('trainDS','trainDS'); % save the training datastore to use for the
training m file
%% Test the datastore!
trainDS.shuffle();
%preview the datastore needs to be in form of 1x4 cell array like this:
%{512×512×3 uint8}    {7×4 double}    {7×1 categorical}    {512×512×7
logical}
data = preview(trainDS) % done
%% visualise an image with its bounding box
% im_pos = string(gTruth{1,1});
% figure(1)
% im = imread(im_pos);
% imshow(im)
% hold on
% rectangle("Position",TrainingData{1,2})
```

Resize_Datastore.m

```
clc;
clear;
%% Input
% Set the resolution to resize to
resize_res= [340,417];%height width
% Add the source directory to the path (add maskrcnn main functions)
addpath('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\Mask
CNN\mask-rcnn-main\src')
% Set the root directory for COCO API
cocoAPIDir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Mask CNN\MatlabAPI';
% Set the directory of the RGB images to be rescaled
Highres_rgb_dir = "B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\UE Generated Pictures\High Res\RGB";
%Set the directory of the datastore to be rescaled
trainDS_dir = "B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\UE Generated Pictures\High Res\trainDS.mat";
%Initial Occlusion Folder
Occlusion_dir = "B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\UE Generated Pictures\High Res\Occlusion Annotations";

%The annotation folder to save the new annotation files
AnnotationFolder = "B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\UE Generated Pictures\Low Res\Annotations";
%The rescaled image dirirectory
Rescaled_rgb_dir = "B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\UE Generated Pictures\Low Res\RGB";
%Occlusion Annotations resized
Occlusion_dir_resized = "B:\OneDrive - CHP\Σχολή μαθήματα\10.
Diplomatikh\matlab code\UE Generated Pictures\Low Res\Occlusion
Annotations";

%% Output
%load the datastore
load(trainDS_dir)
%get the file directories
Files_dir = trainDS.UnderlyingDatastores{1,1}.Files;
%get the number of files in the datastore
num_of_files= size(Files_dir,1);

for i=1:num_of_files
    %load one annotation file at a time
    load(Files_dir{i}) %masks,labels,imageFile,boxes
    %get the directory of the image to be resized
    rgb_dir = strcat(Highres_rgb_dir,'\',imageFile);
    rgb=imread(rgb_dir);
    res = size(rgb,1,2);
    resize_ratio=resize_res./res;
    boxes_new = boxes;
    %find the amount of different objects in the image
    items_on_image = size(labels,1);
    masks_new = false([resize_res,items_on_image]);
    %Rescale the boxes and the masks
    for ii=1:items_on_image
        boxes_new(ii,1)=round(boxes(ii,1)*resize_ratio(2));
        boxes_new(ii,3)=round(boxes(ii,3)*resize_ratio(2));
```

```
        boxes_new(ii,2)=round(boxes(ii,2)*resize_ratio(1));
        boxes_new(ii,4)=round(boxes(ii,4)*resize_ratio(1));
        masks_new(:,:,ii)=imresize(masks(:,:,ii),resize_res);
    end
    %rescale the image
    rgb_new = imresize(rgb,resize_res);
    %save the data to a new annotation file
    imageFile = strcat(num2str(i),'.jpg');
    rgbwrite_dir = strcat(Rescaled_rgb_dir,'\',imageFile);
    imwrite(rgb_new,rgbwrite_dir)
    boxes = boxes_new;
    masks=masks_new;


save(strcat(AnnotationFolder,'\',num2str(i)),"imageFile","boxes","labels","
masks"); % the names play a very big role check: cocoAnnotationMatReader.m
and put the correct names there
    %for occlusion
    [n,name,ext]=fileparts(Files_dir{i});
    occlusion_dir = strcat(Occlusion_dir,'\',name,'.mat');
    load(occlusion_dir);
    save(strcat(Occlusion_dir_resized,'\',num2str(i)),"occlusion")
end
%% Using code from MaskRCNNTrainingExample.mlx
% COCO-MATLABAPI: https://github.com/cocodataset/cocoapi
% Add the API directory to the path
addpath(cocoAPIDir);
% Create the training datastore to read image and ground truth data from
% the unpacked annotation MAT file
trainImgFolder=Rescaled_rgb_dir;
clear trainDS
ds =
fileDatastore(AnnotationFolder,'ReadFcn',@(x)helper.cocoAnnotationMATReader
(x, trainImgFolder));
%% OUTPUT
imageSize = [resize_res,3];
trainDS = transform(ds, @(x)helper.preprocessData(x, imageSize));
save('trainDS','trainDS'); % save the training datastore to use for the
training m file
%% Test the datastore!
 trainDS.shuffle();
%preview the datastore needs to be in form of 1x4 cell array like this:
%{512×512×3 uint8}    {7×4 double}    {7×1 categorical}    {512×512×7
logical}
 data = preview(trainDS);
%% Visualise
% load('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\Testing\Resized to best net\Annotations\1.mat');
% test_img=imread('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\Testing\Resized to best net\RGB\1.jpg');
% overlayeredImage = insertObjectMask(test_img,masks);
% figure(2)
% imshow(overlayeredImage)
% hold on
% showShape("rectangle",boxes,"Label",labels,'LineColor',[1,0,0])
```

Occlusion_Data_Creation.m

```
clc;
clear;
%% Inputs
%Add CHP Functions
addpath('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\Mask
CNN\CHP_Functions');
%load the training ds created by the dataset creation
TrainDS_dir = "B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\CleanImages\HighQuallity\Downscaled\trainDS.mat";
%Input the occlusion mat files for training
Occlusion_dir = "B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\CleanImages\HighQuallity\Occlusion";
%Input the depth images folder
Depth_dir = "B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\CleanImages\HighQuallity\Depth";
%load the names of the parts
load("B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\Images\Mapped\item_names")
%give a location to visualize the intersection areas
intersection_dir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\CleanImages\HighQuallity\Intersection_occlusion_dir';
%intersection circle radius (in pixels)
r_intersection = 10;
%% Main
load(TrainDS_dir);

TrainDS = trainDS.UnderlyingDatastores{1,1}.Files;
total_images = size(TrainDS,1);
id = 1;

for i=1:total_images
    %load the masks etc from annotation folder...
    load(TrainDS{i,1});
    % get the name of the annotation .mat file to open the corresponting
    % .mat file containing the depth image and the occlusion labels
    [folder,name,ext]=fileparts(TrainDS{i,1});
    load(strcat(Occlusion_dir,"\",name,'.mat')) % load the corresponding
mat file
    load(strcat(Depth_dir,'\',name,'.mat'));
    if size(labels,1)>=2
        [INPUT,lab] = Occlusion_Features(d_c,masks,labels);
        objects =size(occlusion,2);
        for ii=1:objects
            DATA(id,1:6)=INPUT(ii,:);
            DATA(id,7)=occlusion(ii);
            DATA_labels(id,1)=lab(ii,1);
            DATA_labels(id,2)=name;
            id=id+1;
        end
    end
end

i1=1;
i2=1;
i3=1;
%Split the data for each network
```

```
for i=1:size(DATA_labels,1)
    if DATA_labels(i,1)=='PT1'
        DATA1(i1,:) =DATA(i,:);
        i1=i1+1;
    elseif DATA_labels(i,1)=='PT2'
        DATA2(i2,:) =DATA(i,:);
        i2=i2+1;
    else
        DATA3(i3,:) =DATA(i,:);
        i3=i3+1;
    end
end
```

Occlusion_ANN_Training.m

```
clear;
clc;
%% INPUTS
load("B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\UE
Generated Pictures\Low Res\DATA2.mat")
DATA = DATA2;
%The raw that the output values are located
output_ind = 7;
input_ind = 4;
datasize = size(DATA,1);
%Set the validation and testing indicies
validation_ind = 4:10:datasize;
test_ind = 6:10:datasize;
%% Input manipulation
test_datasize = size(test_ind,2);
input = DATA(:,1:input_ind);
output = DATA(:,output_ind);
%The remaining indices go to the training data = 80%
training_ind= 1:datasize;% create a matrix with all the indices

index = 1;
for n =1:test_datasize
    training_ind(validation_ind(n)+1-index)=[]; %remove validation indices
    index = index+1;
    training_ind(test_ind(n)+1-index)=[];%remove test indices
    index = index+1;
end

% Separation of training,validation and testing data
training_data=input(training_ind,1:input_ind);
training_output = output(training_ind);

validation_data = input(validation_ind,1:input_ind);
validation_output = output(validation_ind);

testing_data = input(test_ind,1:input_ind);
test_output = output(test_ind);
%% training
HiddenSizes = [100,100,50,50,25,25,10,10,5,5]; %400,400,200,200,100,100,
net = feedforwardnet(HiddenSizes,'trainrp');%to use gpu you have to change
from trainlm, trainscg, trainrp
%%Set ANN training parameters
net.trainParam.epochs = 10000; % maximum epochs
net.trainParam.goal = 0;  % We want 0 error
net.trainParam.max_fail = 20; % validation failures before rejection
net.divideFcn = 'divideind'; % User defined training/testing/validation
data
net.divideParam.trainInd = training_ind;
net.divideParam.valInd = validation_ind;
net.divideParam.testInd = test_ind;
net.performParam.regularization = 0;
net.performParam.normalization = 'none';
% net.layers{1}.transferFcn ='poslin';
%net.layers{end}.transferFcn ='softmax';
%net.performFcn = 'crossentropy';%'mse'; %Using mean absolute error
```

```
net.trainParam.showWindow = 0; % close window pop up for lowering
computation time
% net.trainFcn = 'trainscg';
perf=1000;
for i=1:100
    net.trainParam.showWindow = 0;
    [trained_net,tr] = train(net,input',output'); % train the network
%'useParallel','yes' for cpu // 'useGPU','yes'
    y = trained_net(input');% find the outputs for all the inputs
    %p = crossentropy(trained_net,output',y,{1});%  %Define the error as
described by the performace function (sse)
    p = perform(trained_net, output', y);
    if perf>p
        perf=p; %best performing network
        best_net=trained_net; %keep the best perfoming net
        y_best=y; %keep the relative error
        best_tr =tr; %keep matrix tr
    end
    i
end


%Check best solution
y=best_net(input');
y = round(y,0);
error = abs(y-output');
totalerrors = sum(error);

error_validation=sum(error(validation_ind));
error_testing = sum(error(test_ind));
dif = y-output';
false_positive=0;
false_negative=0;
for i=1:datasize
    if dif(i)>0
        false_positive=false_positive+1;
    elseif dif(i)<0
        false_negative = false_negative+1;
    end
end
```

Mask_RCNN_Training.m

```
clc;
clear;
%% INPUTS
% Add the source directory to the path (add maskrcnn main functions)
addpath('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\Mask
CNN\mask-rcnn-main\src')
% Set the root directory for COCO API:gTruth
cocoAPIDir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Mask CNN\MatlabAPI';
% Add the API directory to the path
addpath(cocoAPIDir);
imageSize = [1042 1353 3]; %height-width
%load training datastore (trainDS)
load('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\CleanImages\HighQuallity\trainDS.mat')
%load cell array with item_names
load('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\Images\Mapped\item_names.mat')

%% Create the network
detector = maskrcnn('resnet50-
coco',item_names,'InputSize',imageSize,PoolSize=[14
14],MaskPoolSize=[14,14]);%https://www.mathworks.com/help/vision/ref/maskrc
nn.html

%% Train the network
options = trainingOptions("sgdm",  ...
    InitialLearnRate=0.002, ...
    Momentum=0.9, ...
    LearnRateSchedule="piecewise", ...
    LearnRateDropPeriod=1, ...
    LearnRateDropFactor=0.99, ...
    Plot="none", ...
    MaxEpochs=12, ...
    MiniBatchSize=2, ...
    BatchNormalizationStatistics="moving", ...
    ResetInputNormalization=false, ...
    ExecutionEnvironment="gpu", ...
    VerboseFrequency=10);
    % GradientThresholdMethod='l2norm',...
    % L2Regularization=10e-5,...

[TrainedDetector,info] =
trainMaskRCNN(trainDS,TrainedDetector,options,'NumRegionsToSample',128,'Num
StrongestRegions',1250,'PositiveOverlapRange',[0.75,1],'NegativeOverlapRang
e',[0  0.75]); %
https://www.mathworks.com/help/vision/ref/trainmaskrcnn.html
%FreezeSubNetwork="backbone"

%% test network for one image
% Read the image for inference
test_img = imread('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\Testing\Resized to best net\RGB\19.jpg');

% Define the target size of the image for inference
targetSize = imageSize; % same as the network image size
```

```
% Resize the image maintaining the aspect ratio and scaling the largest
% dimension to the target size.
imgSize = size(test_img);
[~, maxDim] = max(imgSize);
resizeSize = [NaN NaN];
resizeSize(maxDim) = targetSize(maxDim);

test_img = imresize(test_img, resizeSize);
% detect the objects and their masks for more info:
https://www.mathworks.com/help/vision/ref/maskrcnn.segmentobjects.html#mw_a
9899d3b-d637-4834-85c9-fe5cfae7f8af_sep_mw_c42f62bd-bea4-474a-96a3-
f99843001dde
[masks,labels, scores, boxes] = segmentObjects(TrainedDetector,test_img);

%% visualize test
overlayeredImage = insertObjectMask(test_img,masks);
figure(2)
imshow(overlayeredImage)
hold on
showShape("rectangle",boxes,"Label",labels,'LineColor',[1,0,0])
```

Mask_RCNN_Testing.m

```
clc;
clear;
%this file is named validation but it is more for testing purposes of the
%network
%% INPUTS
% Add the source directory to the path (add maskrcnn main functions)
addpath('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\Mask
CNN\mask-rcnn-main\src')
% Set the root directory for COCO API:
cocoAPIDir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Mask CNN\MatlabAPI';
% Add the API directory to the path
addpath(cocoAPIDir);
h= 1038; %height of the images
w=1353; %width of the images
imageSize = [h w 3]; %height-width
%insert trained detector directory
Trained_detector_dir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10.
Diplomatikh\matlab code\Mask CNN\TrainedDetectors\test7.2.mat';
%inert validation datastore directory
ValidationDS_dir ='B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Kinect\CleanImages\HighQuallity\trainDS.mat';
%load cell array with item_names
load('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\Item
images\v2\Testing\item_names.mat')
%load trained detector
load(Trained_detector_dir);
%load validation datastore
load(ValidationDS_dir);

% Annotations = trainDS.UnderlyingDatastores{1,1}.Files;
% image_number = size(Annotations,1);

%% Validating-Testing
options = trainingOptions("sgdm",  ...
%https://www.mathworks.com/help/deeplearning/ref/trainingoptions.html %
    InitialLearnRate=0.00000001, ... %for testing set Initial Learn Rate to
10^-10 and epochs to 1 and verbose frequency to 1 thet take the average of
the results
    Momentum=0.9, ...
    LearnRateSchedule="piecewise", ...
    LearnRateDropPeriod=1, ...
    LearnRateDropFactor=0.95, ...
    Plot="none", ...
    MaxEpochs=1, ...
    MiniBatchSize=2, ...
    BatchNormalizationStatistics="moving", ...
    ResetInputNormalization=false, ...
    ExecutionEnvironment="gpu", ...
    VerboseFrequency=1);
%use the same properties in trainMaskRCNN as used in actual training
[TrainedDetector,info] =
trainMaskRCNN(trainDS,TrainedDetector,options,'NumRegionsToSample',128,'Num
StrongestRegions',1000,'PositiveOverlapRange',[0.8,1],'NegativeOverlapRange
',[0.1  0.8]); %
```

```
https://www.mathworks.com/help/vision/ref/trainmaskrcnn.html
%FreezeSubNetwork="backbone"
%total error is the summ of errors from last epoch devided by the total
%amount of evaluations
TotalLoss = 0;
RPNLoss = 0;
RMSE = 0;
MaskLoss = 0;

testing_size = size(info,1);
for n =1:testing_size

TotalLoss=info(n).TrainingLoss/testing_size+TotalLoss;
RPNLoss = info(n).TrainingRPNLoss/testing_size+RPNLoss;
RMSE = info(n).TrainingRMSE/testing_size+RMSE;
MaskLoss=info(n).TrainingMaskLoss/testing_size+MaskLoss;

end
```

UE_Data_Creation.m

```
clc;
clear;
%CHP Functions
addpath('B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\Mask
CNN\CHP_Functions');
%Initiate image numbering
% i=1;
%Add Simulink Path
simulink_dir='B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\Testing_Images\UE_Pictures.slx';
%Initial Color of Items
InitialColor = [0.5,0,0.009];
%Segmentation Color of Items
Color1_seg =[1,0,0];
Color2_seg=[0,1,0];
Color3_seg=[0,0,1];
%Depth sensor resolution
D_res = [340,417];
%RGB Save Location
RGB_dir = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab code\UE
Generated Pictures\High Res\RGB';
%Depth Save Location High res
Depth_dir_H = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\UE Generated Pictures\High Res\Depth';
%Depth Save Location low res
Depth_dir_L='B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\UE Generated Pictures\Low Res\Depth';
%Annotation Folder
AnnotationFolder = 'B:\OneDrive - CHP\Σχολή μαθήματα\10. Diplomatikh\matlab
code\UE Generated Pictures\High Res\Annotations';

%% Take initial Pictures to save
Color1 = InitialColor;
Color2= InitialColor;
Color3= InitialColor;

out = sim(simulink_dir,0.1);
%RGB image:
RGB = out.RGB.signals.values;
%Depth image:
D = out.D.signals.values*1000;

% figure(1)
% imshow(RGB);
% figure(2)
% imshow(D)
imageFile = strcat(num2str(i),'.png');
rgb_h_dir = strcat(RGB_dir,'\',imageFile);
imwrite(RGB,rgb_h_dir);

d_c = imresize(D,D_res);
d_l_dir = strcat(Depth_dir_L,'\',num2str(i));
save(d_l_dir,"d_c");

%Segmentation
```

```
Color1 = Color1_seg;
Color2= Color2_seg;
Color3= Color3_seg;

out = sim(simulink_dir,0.1);
%RGB image:
rgb_seg = out.RGB.signals.values;
figure(3)
imshow(rgb_seg);

Mask1 = Image_to_Mask(rgb_seg,Color1*256);
Mask2 = Image_to_Mask(rgb_seg,Color2*256);
Mask3 = Image_to_Mask(rgb_seg,Color3*256);
% figure(4)
% montage({Mask1,Mask2,Mask3});

labels = categorical(["PT1";"PT2";"PT3"]);
masks = false([size(Mask1),3]);
masks(:,:,1)=Mask1;
masks(:,:,2)=Mask2;
masks(:,:,3)=Mask3;
boxes = zeros(3,4);
boxes(1,:)=BoundingBox_From_Mask(Mask1);
boxes(2,:)=BoundingBox_From_Mask(Mask2);
boxes(3,:)=BoundingBox_From_Mask(Mask3);

% visualise an image with its bounding box
%load("1.mat")
overlayeredImage = insertObjectMask(RGB,masks);
figure(2)
imshow(overlayeredImage)
hold on
showShape("rectangle",boxes,"Label",labels,'LineColor',[1,0,0])
%Save Annotation File
save(strcat(AnnotationFolder,'\',num2str(i)),"imageFile","boxes","labels","
masks");
i=i+1;
```

II. MATLAB Custom Made Functions

best_sol.m

```
function [Final_Config] = best_sol(Initial_Config,Final_Configs)
%This function finds the "best" solution for the end configuration of a
% robotic arm based on angle rotations.
[num_of_sol,~] = size(Final_Configs);
minimum_movement=10000;
for n=1:num_of_sol
    movement = sum((Initial_Config-Final_Configs(n,:)).^2); %summed squered
error
    if movement<minimum_movement
        minimum_movement=movement;
        best_ik_sol=n;
    end
end
Final_Con = Final_Configs(best_ik_sol,:);
Final_Config = [Final_Con,0,0];
end
```

BoundingBox_From_Mask.m

```
function [Box] = BoundingBox_From_Mask(mask)
%This function creates a bounding box from a mask
props=regionprops(mask,'BoundingBox');
bboxes=cell2mat(struct2cell(props)');
bboxes(:,3)= bboxes(:,1)+bboxes(:,3);
bboxes(:,4)= bboxes(:,2)+bboxes(:,4);
x=min(bboxes(:,1));
y=min(bboxes(:,2));
w=max(bboxes(:,3))-x;
h=max(bboxes(:,4))-y;
Box=[x,y,w,h];
end
```

BoundingBox_From_Polygon.m

```
function [BD] = BoundingBox_from_Polygon(x,y)
%This function create a bounding box from a polygon
upper_y = floor(min(y)); %as it is in images y is from top to buttom
lower_y = ceil(max(y));
left_x = floor(min(x));
right_x = ceil(max(x));
w = right_x-left_x;
h=lower_y-upper_y;
BD = [left_x,upper_y,w,h];
end
```

BW_circle.m

```
function [out] = BW_circle(c,r,res)
% This function creates a binarry image of size res =[h,w] with a circle of
% radius r in pixels and a center at c = [y,x]
```

```
out = false(res);
for i=1:res(1)
    for ii=1:res(2)
        dist = sqrt((ii-c(2))^2+(i-c(1))^2);
        if dist<=r
            out(i,ii)=1;
        end
    end
end
end
```

CHP_Blend.m

```
function [out] = CHP_Blend(Background,img,mask)
%This function inserts an image on top of the background image as specified
%by the mask (alpha channel).
if size(Background)==size(img)
    H=size(Background,1);
    W = size(Background,2);
    out=Background;

    for n=1:H
        for nn=1:W
            if mask(n,nn)==1
                out(n,nn,:)=img(n,nn,:);
            end
        end
    end

else
end

end
```

DrawBW_line.m

```
function [BW1] = DrawBW_line(c1,c2,res)
%This function creates a binary image containing a line prom point c1 to
%point c2
%initial point c1 = [x,y] and ending point c2=[x,y];
%res = [h,w] the resolution of the image
c1=[ceil(c1(1)),ceil(c1(2))];
c2=[ceil(c2(1)),ceil(c2(2))];
dp = c2-c1;
a=dp(2)/dp(1);% dy/dx
BW=false(res);
x_pixels = dp(1);
if x_pixels>=0
for x=1:abs(x_pixels)
    y=a*x+c1(2);
    BW(ceil(y),x+c1(1))=1;
end
else
for x=1:abs(x_pixels)
    y=a*x+c2(2);
    BW(ceil(y),x+c2(1))=1;
end
end
se=strel('rectangle',[2,2]);
BW1 = imdilate(BW,se);
end
```

DrawBW_lineA.m

```
function [BW] = DrawBW_lineA(c,a,res)
% This function creates a binary images with a line  passing from point
% c=[x,y] with an angle of a.

BW=false(res);
A=tan(a);

b=c(2)-A*c(1);
if A>1 %if the angle is greater than 45deg use the form x=(y-b)/A
for i=1:res(1)
    xx=round((i-b)/A);
    if xx>0 && xx<=res(2)
        BW(i,xx)=1;
    end
end
else%if the angle is smaller than 45deg use the form y=Ax+b
for i=1:res(2)
    yy=round(A*i+b);
    if yy>0 && yy<=res(1)
        BW(yy,i)=1;
    end
end
end
```

Grab_selection.m

```
function [center,orientation,simplez,Pick_Item_Index,Gripper1,Gripper2] =
Grab_Selection(masks,depth,occlusion,f)

%% Select a part (the one that is not occluded but also is the tallest)
[height,background_height] = Height_of_Objects(masks,depth);
items_in_image = size(masks(1,1,:),3);
BiggestHeight =-10;
for i =1:items_in_image
    if occlusion(i)==0 && height(i)>BiggestHeight
        Pick_Item_Index=i;
        BiggestHeight=height(i);
    end
end

%% Get orientation of the part
mask = masks(:,:,Pick_Item_Index);
props = regionprops(mask,"Centroid","Orientation");
center = props(1).Centroid;
orientation=props(1).Orientation/180*pi; % angles in degs from x axis [-
90,90] positive with right hand rule with z towards us from the screen
simplez = (background_height-BiggestHeight+10)/1000; % to get the z
distance from item

%% Get Grpipper positions
res=size(mask);
%Get the perimeter of the mask
perim = bwperim(mask);
se = strel('rectangle',[2,2]);
%dilate it by 2
```

```
perim = imdilate(perim,se);
%Daw a line where the gripper is going to grab the part
Line =DrawBW_lineA(center,orientation-pi/2,res);
%Find the two grasping positions
Grab_BW =and(Line,perim);
% To get the correct gripper to move we proceed with the following:
R=R_z(pi-orientation);
T=eye(4);% tranfrorm from image coordinate system to item coordinate system
T(1:3,1:3)=R;
T(1:2,4)=center';
%b-> image coordinate system, %c-> item coordinate system
T_c_b = inv(T);
Grab_props=regionprops(Grab_BW,"Centroid");
Grab_locs = struct2array(Grab_props); %[x1,y1,x2,y2]
c1=Grab_locs(1:2);
c2=Grab_locs(3:4);
c1_new=T_c_b*[c1,0,1]'; %y coordinate indicates the distance needed to move
from the center of the item to the grabbing location
c2_new=T_c_b*[c2,0,1]';
Dist_of_gripper_from_center=0.085;
if c1_new(2)>0 %Gripper is located in the positive direction while gripper
2 is in the negative
Gripper1=Dist_of_gripper_from_center-c1_new(2)/f*simplez;%distance that
gripper needs to move for visulization
Gripper2=Dist_of_gripper_from_center-abs(c2_new(2)/f*simplez);
else
Gripper1=abs(Dist_of_gripper_from_center-c2_new(2)/f*simplez);%distance
that gripper needs to move for visulization
Gripper2=abs(Dist_of_gripper_from_center-abs(c1_new(2)/f*simplez));
end
end
```

Height_of_Objects.m

```
function [height,background_height] = Height_of_Objects(masks,depth)
%This function calculates the average height of an object specified by a
%mask and calculates the background height (where there are no objects).
res = size(masks(:,:,1));
items_in_image = size(masks,3);
background_mask= false(res);

for ii = 1:items_in_image % find backround height
        background_mask = or(masks(:,:,ii),background_mask);
end

background_mask = imcomplement(background_mask);
background_mask_height = double(background_mask).*double(depth);
background_area = sum(background_mask,"all");
background_height = sum(background_mask_height,"all")/background_area;
height = zeros(items_in_image,1);
for ii =1:items_in_image
    Area = sum(masks(:,:,ii),"all");
    mask_depth = double(masks(:,:,ii)).*double(depth);
    height(ii) =background_height-sum(mask_depth,"all")/Area;
end

end
```

Occlusion_Correction.m

```
function [masks,bboxes,labels] =
Occlusion_Correction(masks,bboxes,labels,i,min_occlusion_pixel_area)
loop = size(i,2);

for n =1:(loop-1)
    for nn=1:n
        masks(:,:,i(end-n)) = masks(:,:,i(end-
n)).*imcomplement(masks(:,:,i(end-nn+1)));
    end

end
ii=0;
while n <= loop-ii
    S = sum(masks(:,:,n),'all');
    if S<min_occlusion_pixel_area % it can be altered
        masks(:,:,n)=[];
        labels(n)=[];
        bboxes(n,:)=[];
        n=n-1;
        ii=ii+1;
    else
        stats = regionprops(masks(:,:,n),'BoundingBox');
        bb= stats.BoundingBox;
        if size(bb,1)==1
            bboxes(n,:)=bb;
        else
            bboxes(n,1)=min(bb(:,1));
            bboxes(n,2)=min(bb(:,2));
            % find the maximum x from all the regions that represent the
item
            xmax = max(bb(:,1)+bb(:,3));
            ymax = max(bb(:,2)+bb(:,4));
            bboxes(n,3) = xmax-bboxes(n,1);
            bboxes(n,4) = ymax-bboxes(n,2);
        end
    end
n=n+1;
end

end
```

Occlusion_Detector.m

```
function [occlusion] = Occlusion_Detector(net1,net2,net3,d_c,masks,labels)
%this function utilizes the occlusion networks to find occlusion properties
num_of_objects_detected = size(labels,1);
if num_of_objects_detected==1
    occlusion=0;
elseif num_of_objects_detected>1
    [Features,Features_labels] = Occlusion_Features(d_c,masks,labels);
    occlusion = false(num_of_objects_detected,1);
    for i=1:num_of_objects_detected
        if Features_labels(i,1)=='PT1'
            occlusion(i,1)=round(net1(Features(i,:)'),0);
        elseif Features_labels(i,1)=='PT2'
            occlusion(i,1)=round(net2(Features(i,:)'),0);
        else
            occlusion(i,1)=round(net3(Features(i,:)'),0);
        end
    end
else
    occlusion = [];
end

end
```

Occlusion_Features.m

```
function [DATA,DATA_labels] = Occlusion_Features(d_c,masks,labels)
%This functionextracts the features for the detection of occlusion
%d_c is the depth image
%mask is the masks on the image
%labels are the labels of the masks

%radius of intersection circle for local area height
r_intersection = 10;

%the ammount of objects-items in an image is equal to the ammount of
%labels
items_in_image = size(labels,1);
% find the resolution of the images
res = size(masks(:,:,1));
%% Generate the background mask and find its height
background_mask= false(res);
if items_in_image<=1 % if there is only one item then there is no occlusion
    %do not create any data
else
    for ii = 1:items_in_image % find backround height
        background_mask = or(masks(:,:,ii),background_mask);
    end
    background_mask=imcomplement(background_mask);
    background_mask_height = double(background_mask).*double(d_c);
    background_area = sum(background_mask,"all");
    background_height = sum(background_mask_height,"all")/background_area;
    %% Calculate the distance between all the items-objects in an image
    % Find minimum distance between objects
```

```
    %valuable info from:
https://nl.mathworks.com/matlabcentral/answers/91046-how-to-find-distance-
in-binary-image
    % gennerate a matrix with very high diagonal values
    %(maximum distance between objects is set to 100 pixels)
    min_distance_matrix = eye(items_in_image)*100;
    %matrix of cells containint [x,y] coordinates of pixels, to keep track
    %of the pixel positions that have the smallest distance between the
objects
    min_distance_index = cell(items_in_image);
    %find the local height at the closest points between objects
    local_height_matrix = eye(items_in_image)*1000;
    pixeldistance_mask=zeros(res(1),res(2),items_in_image);
    mask_perimeter = false(res(1),res(2),items_in_image);
    for ii = 1:items_in_image
        pixeldistance_mask(:,:,ii) = bwdist(masks(:,:,ii));
        mask_perimeter(:,:,ii) = bwperim(masks(:,:,ii));

    end
    for ii= 1:items_in_image
        for iii =1:items_in_image
            if ii~=iii
                %Distance of item iii from item ii
                perimeter_distance_ii_iii =
pixeldistance_mask(:,:,ii).*mask_perimeter(:,:,iii);
                % create a matrix with very high values where there is no
perimeter
                % (to add it to the distance so when I take the minimum I
take the distance and not 0)
                perimeter_inverse =
inverse_BW_CHP(mask_perimeter(:,:,iii))*1000000;
                Distances_matrix =
perimeter_distance_ii_iii+perimeter_inverse;
                %this gets the minimum from all columns
                [a,I]=min(Distances_matrix,[],1);
                %this gets the mimimum from the row (gets index as well).
                [min_distance_matrix(ii,iii),II]=min(a);
                %get the indexes to a matrix (pixel where there is the
closes distance)
                min_distance_index{ii,iii} = [I(II),II];
                % Gennerate a binary image with resolution res witch
contains a
                % circle with radius r and center at the closest point
between two objects
                circle =
BW_circle(min_distance_index{ii,iii},r_intersection,res);
                % find the local height of item ii in the closest point to
item iii.
                circle_intersection= and(masks(:,:,ii),circle);
                %find the area of intersection
                circle_intersection_area = sum(circle_intersection,'all');
                % gennerate a metrix containing the local heights between
objects
                local_height_matrix(ii,iii)=background_height-
sum(double(circle_intersection).*double(d_c),"all")/circle_intersection_are
a;

            end
```

```
        end
    end
    local_height_difference_matrix = eye(items_in_image)*1000;
    for ii=1:items_in_image
        for iii=1:items_in_image
            if iii~=ii
                local_height_difference_matrix(ii,iii)=
local_height_matrix(ii,iii)-local_height_matrix(iii,ii);
            end
        end
    end
    for ii = 1:items_in_image

        [closest_object_distance,closest_object_index] =
min(min_distance_matrix(ii,:));
        props =
regionprops(masks(:,:,ii),'Area','MajorAxisLength','MinorAxisLength','Perim
eter');

        Area = props(1).Area;
        Perimeter = props(1).Perimeter;
        aspect_ratio = props(1).MajorAxisLength/props(1).MinorAxisLength;
        mask_depth = double(masks(:,:,ii)).*double(d_c);

        avg_height =background_height-sum(mask_depth,"all")/Area;

        %hiest_point = background_height-min(smothed_mask_depth,[],'all');
        Openned_Area = Area/background_height^2;
        Openned_Perimeter = Perimeter/background_height;
        %find the minimum height ddifference of object ii in terms of all
        %other objects (for occluded objects it might be negative)
        min_height_difference =
local_height_difference_matrix(ii,closest_object_index);%min(local_height_d
ifference_matrix(ii,:));
        if isnan(min_height_difference)
            min_height_difference=0;
        end
        DATA(ii,:) =
[Openned_Area,avg_height,Openned_Perimeter,aspect_ratio,closest_object_dist
ance,min_height_difference];
        DATA_labels(ii,1)=labels(ii,1);
    end
end
end
```

R_z.m

```
function [R] = R_z(t)
%Rotation matrix when a coordinate system its rotated by t rad in the z
%axis
c=cos(t);
s=sin(t);
R=[c,-s,0;
    s,c,0;
    0,0,1];
end
```

ResizeImageMasksBoxes.m

```
function [im_new,masks_new,boxes_new] =
ResizeImageMasksBoxes(im,masks,boxes,newres)
%Resize the image,masks and boxes to new resolution specified by newres

res=size(im,1,2);
resize_ratio=newres./res;
boxes_new = boxes;
items_on_image = size(masks,3);
masks_new = false([newres,items_on_image]);
for ii=1:items_on_image
    boxes_new(ii,1)=round(boxes(ii,1)*resize_ratio(2));
    boxes_new(ii,3)=round(boxes(ii,3)*resize_ratio(2));
    boxes_new(ii,2)=round(boxes(ii,2)*resize_ratio(1));
    boxes_new(ii,4)=round(boxes(ii,4)*resize_ratio(1));
    masks_new(:,:,ii)=imresize(masks(:,:,ii),newres);
end
    im_new = imresize(im,newres);
end
```

UEtoMATLABtransfrom.m

```
function T = UEtoMATLABtransform(L,eul)
%Ureal Engine to MATLAB coordinate system tranfromation
lsgn = [1 -1 1];
rsgn = [1 -1 -1];
T=eye(4);
T(1:3,4)=(L.*lsgn)';
eul=eul.*rsgn;
T(1:3,1:3)=eul2rotm(eul,"XYZ");
end
```

Kinect_RGBtoDepthMap.m

```
function [rgb_cropped_fix,d_cropped_fix] =
Kinect_RGBtoDepthMap(RGB_im,D_im)
%This function maps the rgb image to the depth image
%resolutions
Pixel_Ratio = 3.05; %depth pixels are 3 times larger than rgb pixels
Scale_Ratio = 2; %How much bigger ratio should the rgb image have
%the center distance value in the x direction changes!!
c_x = 0; %pixels in depth image plane(move it towards the depth plane)
c_y = -15;%pixels in depth image plane
f = 365; %focal length of D_camera fx = fy (almost);
cx = 256; %center point of depth camera;
cy = 212; %center point of depth camera;
K_depth = [f,0,cx;
    0,f,cy;
    0,0,1];
RadialDistortion_depth = [0.2,0]; % it specifies an elipse(rx ,ry)
side_cut = 35; % is an additional cropping in pixels from each side that
needs to be done in order to compensate for the distortion correction;
% x = ii-0.0000500604*bg_depth^2-0.1078012098*bg_depth+68.4597461493;
%% Fix distortion of depth image
depth_params =
cameraParameters('K',K_depth,'RadialDistortion',RadialDistortion_depth);
D_im = undistortImage(D_im,depth_params);
res_d = size(D_im);
boundingBox = [side_cut,side_cut,floor(res_d(2)-2*side_cut),res_d(1)-
2*side_cut];
D_im = imcrop(D_im,boundingBox);
res_d = size(D_im);

%principal points
% principal_rgb = res_rgb/2;
% principal_d = res_d/2;
%
% res_ratio = res_rgb./res_d;
RGB_im = imresize(RGB_im,1/Pixel_Ratio);

res_rgb = size(RGB_im,[1,2]);
RGB_im = imtranslate(RGB_im,[c_x,c_y]); % move the rgb image to the depth
center
boundingBox = [0,0,floor(res_rgb(2)-abs(c_x)),floor(res_rgb(1)-abs(c_y))];%
Becarefull!!!! only if c_x and c_y are negative this works
% Crop the translated image to remove empty areas
RGB_im = imcrop(RGB_im, boundingBox);
res_rgb = size(RGB_im,[1,2]);
new_image_res = [min(res_rgb(1),res_d(1)),min(res_rgb(2),res_d(2))];
%355x512
x_diff = ceil((res_rgb(2)-res_d(2))/2);
y_diff = ceil((res_d(1)-res_rgb(1))/2);

rgb_cropped = imcrop(RGB_im, [x_diff,1,new_image_res(2)-
1,new_image_res(1)]);
d_cropped = imcrop(D_im, [1,y_diff,new_image_res(2),new_image_res(1)-1]);
% d_cropped = imgaussfilt(d_cropped,2);

%% Fix depth pixels to rgb mapping issue in x direction
cropped_res = size(d_cropped);
```

```
d_cropped_f = d_cropped;
for i=1:cropped_res(1)
    for ii = 1:cropped_res(2)
        if d_cropped(i,ii)==0
            %do nothing (should probably fill the area with the average of
the perimeter)
        else
            d_at_pos = double(d_cropped(i,ii));
            dx = (0.0000500604*d_at_pos^2 - 0.1078012098*d_at_pos +
68.4597461493);
            x = ii-dx;
            if x>=1 && x<=cropped_res(2)
                d_cropped_f(i,ceil(x))=d_cropped(i,ii);
                d_cropped_f(i,floor(x))=d_cropped(i,ii);
            end
        end
    end
end
%crop again both images
rgb_cropped_fix = imcrop(rgb_cropped,[1,1,cropped_res(2)-
27,cropped_res(1)]);
d_cropped_fix = imcrop(d_cropped_f,[1,1,cropped_res(2)-27,cropped_res(1)]);
end
```

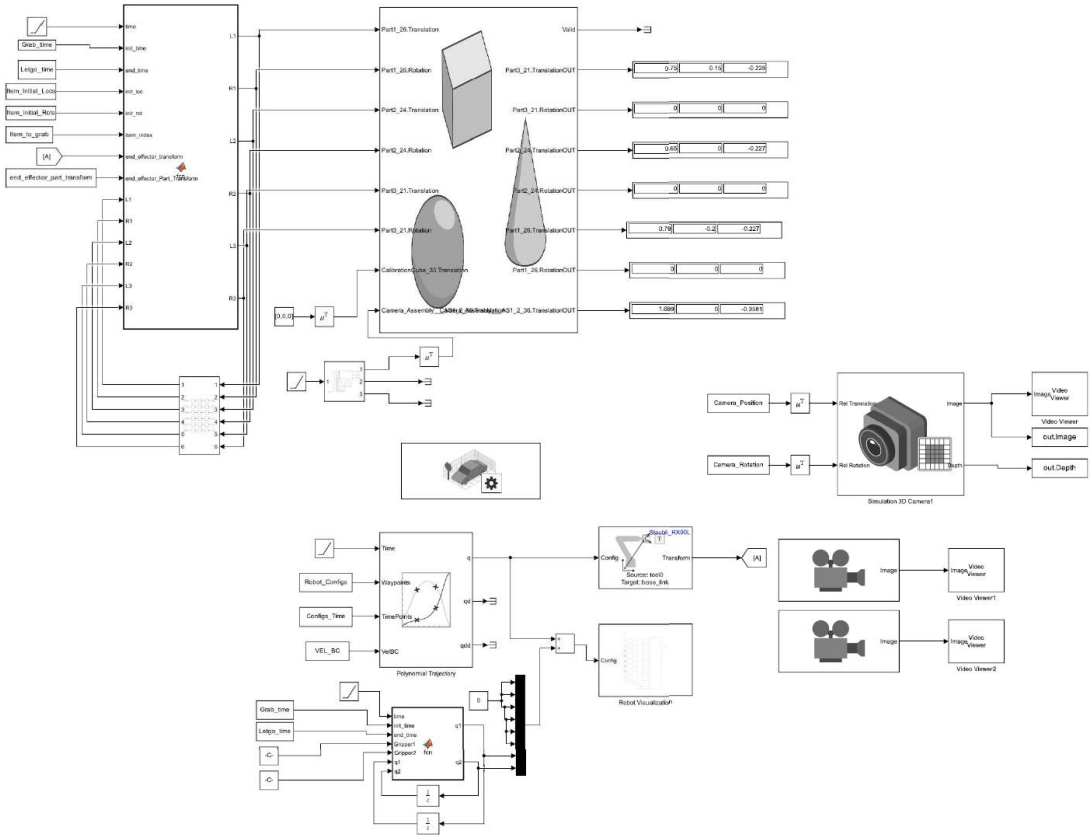
KinectPicture.m

```
function [img_rgb,img_d] = KinectPicture(colorDevice,depthDevice)
%This function takes a picture using the Kinect
step(colorDevice);
step(depthDevice);
step(depthDevice);
img_rgb = step(colorDevice);
img_d=step(depthDevice);
img_rgb = flip(img_rgb,2);
img_d = flip(img_d,2);

end
```

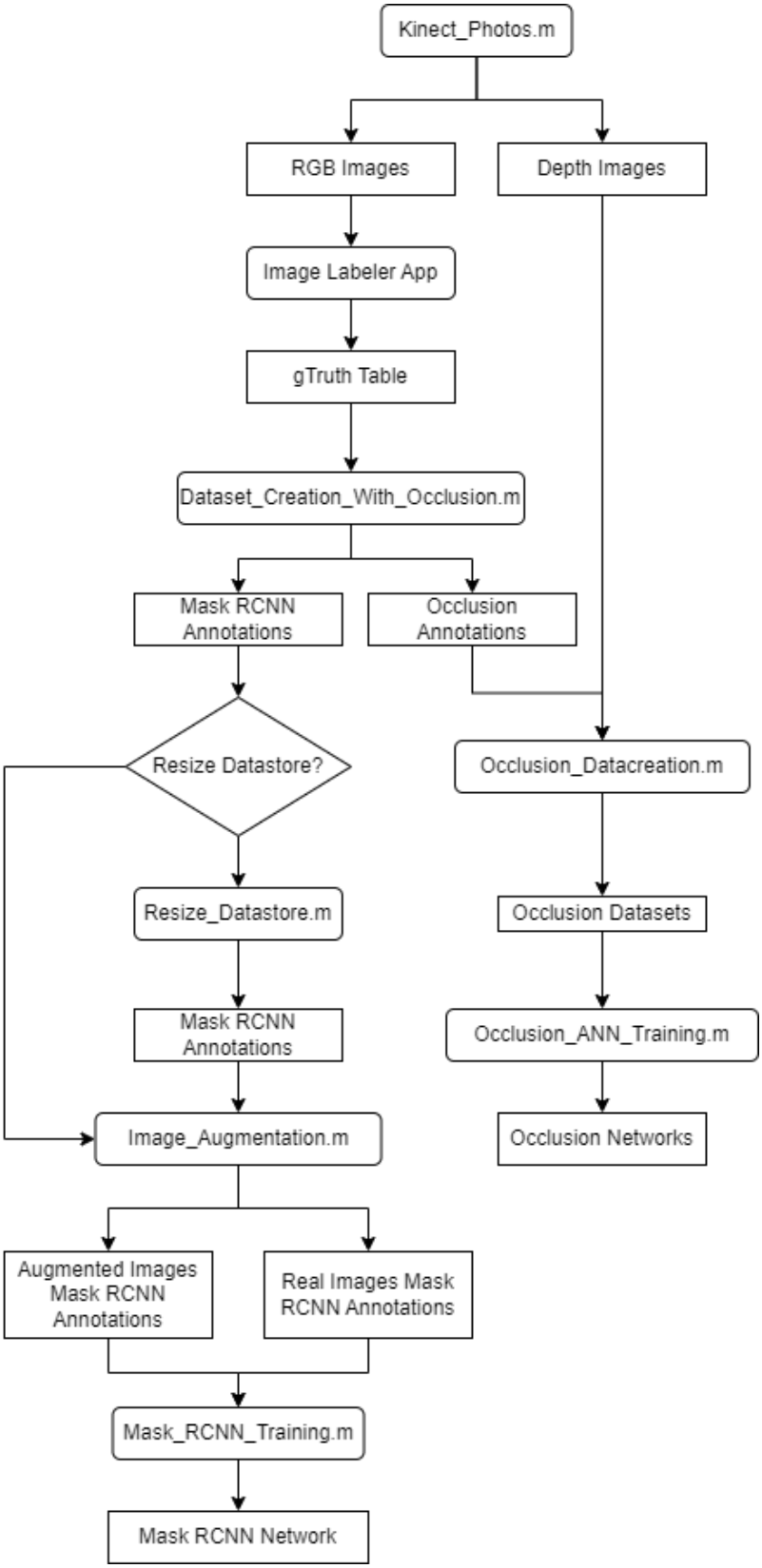
Uint16_to_uint8.m

```
function [out] = uint16_to_uint8(d)
%This function converts a depth image for visulization
max_z = max(d,[],'all');
dd = d;
dd(dd==0)= max_z;
min_z = min(dd,[],'all');

out = uint8(ceil((double(dd-min_z))*255/double(max_z-min_z)));
end
```

III. Simulink Model



IV. Mask RCNN & ANNs Training Workflow



V. Computer & Software Specifications

CPU	AMD Ryzen 9 5900X @4.2GHz
RAM	64GB ddr4 @3600MHz
GPU	NVIDIA GeForce RTX 3090

MATLAB Version 2023a, update 3.

SOLIDWORKS 2023 SP3.

Blender 2.93.4.