university of groningen

faculty of science and engineering

MASTER'S THESIS

Target Driven Object Grasping in Highly Cluttered Scenarios through Domain Randomization and Active Segmentation

Author:
Ivar MAK

First Supervisor:
Dr. Hamidreza KASAEI

Second Supervisor:
Prof. Dr. Raffaella CARLONI

Artificial Intelligence
Computational Intelligence and Robotics

August 10, 2022

Abstract

Robots nowadays are demanded to perform tasks that divert from controlled environments. With regards to robotic grasping systems, dynamic situations pose problems in multiple ways. Grasping a target object from a pile might be impeded by other objects that obscure the robot's view, or restrict the grasping motion. In this work, a deep learning approach is presented that is capable of flexible object manipulation in highly cluttered environments. Using a Neural Network (NN) for object detection and -segmentation, and a second NN for grasp synthesis, a system is built that handles robust object grasping in human-centric domains. These consist of household objects located on a table, which are manipulated by a robotic arm with a two-fingered grasping hand in order to single out a target object. To evaluate the performance of the proposed approach, four sets of experiments are performed, with the objects being in *isolated*, *packed*, *piled*, and *cluttered* scenarios. An experiment is considered a success when the target object is placed in its corresponding tray, located next to the table. Experimental results show accuracy scores of 94% for isolated, 89% for packed, 85% for piled, and 80% for cluttered environments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With our society's present-day technical advances, automated jobs and tasks are a burgeoning part of our lives. The functionality of 'smart' electronical systems is becoming prevalent in day-to-day acts, but with their expanding rate of deployment come challenges that need conquering.

Robots have been widely utilized for industrial purposes over the past decades. With the advancing of industrialization, and their increasing competence and effectiveness with regards to object manipulation, intelligent robots play an increasingly important role in intelligent manufacturing, intelligent transportation systems, the Internet of things, and intelligent services (Hu et al., 2019; Wang, Tao, and Liu, 2018).

Furthermore, the desire to deploy robots has grown in other sectors. An assistant robot could alleviate the workload of medical staff, help out with domestic tasks, or be of service in the hospitality sector (Robinson, MacDonald, and Broadbent, 2014; Wilson et al., 2019; Bowen and Morosan, 2018). We are striving towards robotic applications that are capable of performing increasingly complex tasks. These tasks include circumstances that divert from controlled environments in which solely repetitive and preprogammed operations are to be performed. With this transition arises the need for dynamic and adaptive systems.

In order to deploy robots in human-centric environments, systems need to account for unstructured settings. When imagining a service robot performing household tasks or a factory robot working a production line, these settings may consist of packed items or piles of objects. In these, motion planning is a challenging task due to the high demand of real-time and accurate responses for a vast number of objects. Also, these objects come with a wide variety of shapes and sizes under various clutter and occlusion conditions (Kasaei and Kasaei, 2022). Furthermore, realistic environments pose high uncertainty with regard to properly grasping objects, either through sensory occlusion or the actuator being prohibited of reaching the object (Kiatos et al., 2022).

Grasping is in many cases the most effective way of moving an object from one place to another. It comprises of gripping and moving actions, in which the basic elements are localization of the object and its environment. This requires visual accuracy, robust sensing, and fine control. Performing a successful grasp consists of three stages: the inital grasp, carrying or manipulation, and finally the release or placement. Grasping can be performed in a target-agnostic manner, coming with a higher level objective, for example bin picking or cleaning a dinner table. When a specific target is to be localized and manipulated, we refer to it as a target-oriented problem. Besides that, a crucial part of grasping is acquiring perception, which is called the precondition of grasping (Marwan, Chua, and Kwek, 2021). This is particularly important for a robot facing a complex environment with multiple objects.

Vision is the main channel in which humans receive all types of information. In order to acquire robotic applications capable of complex manipulation tasks, we need to equip them with vision systems that have high accuracy and robustness, similar to human beings. Vision sensors are mostly applied in robotics using cameras, which provide input data for the system component responsible for object detection. This component will apply image processing in order to analyze the input and make inferences about the situation. Accurate and fast object recognition based on vision is a basic element of robot applications in both industry and real-life scenarios (Bai et al., 2020).

In many situations, objects do not appear in isolation. Realistic environments are often highly cluttered, which leads to challenging tasks with regard to grasping. In cluttered environments, grasping a target object might not be feasible as a result of occlusion caused by other objects. Furthermore, the cluttered presence might impair the detection accuracy of an object recognition system as well. When performing targeted grasping on objects in cluttered environments, the key challenges are object detection and finding appropriate grasp poses (Jo and Song, 2020).

When a robotic object manipulation system operates, an accurate object detection module aids in constructing a segmented representation of the scenario at hand. Since cluttered environments might impair an accurate execution of the manipulation step, this might not be sufficient for reliable performance. Pre-grasp manipulation policies can substantially increase the grasp success rates in complex environments (Kiatos et al., 2022). Grasping requires a certain amount of free space around an object, which means in dealing with clutter the robot might need active segmentation steps in order to achieve a stable grasp opportunity. When a collision-free grasp does not exist, pushing operations can singulate objects in clutter, enabling future grasping of these isolated objects (Tang et al., 2021).

Before the object detection and grasping components of a robotic manipulation system can be employed, they need to be familiarized with the objects that need manipulation. In most deep learning approaches, this demands training on labeled data until a point of convergence is reached, and the task can be performed accurately. Training on real life scenarios might not be desired, it could take a lot of time, or random exploration might be employed, which can be dangerous on physical hardware. Performing robotic learning in a simulator could accelerate the process, make it more scalable, and lower the cost of data collection (Tobin et al., 2017). When this trained system is then applied to a real life scenario, we talk about a transfer of domains. This comes with its challenges, since the discrepancies between a simulation and real life scenario can be large, even if the task at hand is the same. This where domain randomization comes into play. It is a popular technique for improving domain transfer. By randomizing parameters in the training phase that might vary in the target environment, the hope is that the agent will view the target domain as just another variation (Mehta et al., 2020).

1.1 Research Questions

The main objective of this research is adding to the current state-of-the-art of object manipulation by applying and combining a number of deep learning techniques that have been shown to be individually successful. We continue research into object detection and grasping synthesis by focusing on highly cluttered scenarios. We explore the possibilities of our approach and evaluate its performance by answering the following questions:

- Is it possible to implement a model capable of grasping a target object in a highly cluttered scenario?
- Does the obtained model perform well with regard to accuracy of targeted grasping?
- What kind of scenarios pose limitations on the approach, and what are these limitations?
- How does the visibility of the object influence the models performance in highly cluttered scenarios?
- Does the application of domain randomization improve the performance of the model?

1.2 Contributions

In this research, we propose to solve the problem of grasping a target object in a highly cluttered scenario by implementing deep learning methods combined with active segmentation. The key aspects of the approach are:

- A target-oriented model that is able to operate in a highly cluttered scenario of 15 human-centric objects, with limited visibility of the target object.
- Increased performance on object recognition in isolated and cluttered scenarios is obtained, by incorporating a novel approach of domain randomization in the training phase of the object detection module.
- The implementation of an algorithm applying both passive- and active segmentation, which aids in the performance of the grasping module.

1.3 Thesis Outline

The thesis consists of five chapters, and is organized as follows: Chapter 2 discusses the related work with regard to the components of this research. Chapter 3 describes their functionality in depth and the methods utilized in the implementation of the approach. Chapter 4 provides all details of the experiments that were performed and presents the results. These are evaluated and discussed in Chapter 5, followed by our conclusions and directions for future work.

Chapter 2

Related Work

An in-depth review is beyond the scope of this work, and has been written extensively regarding intelligent robots (Wang, Tao, and Liu, 2018), grasping (Marwan, Chua, and Kwek, 2021; Mohammed, Chung, and Chyi, 2020; Wang et al., 2020), and object detection (Bai et al., 2020; Bharati and Pramanik, 2020). However, in order to provide a theoretical background, in this section a few recent efforts are discussed and summarized for each of the subjects encompassed by this research.

2.1 Object Detection

There are several possible approaches that can be used for object detection, most of them based on deep learning methods with convolutional properties.

Liu et al., 2016 proposed the Single Shot MultiBox Detector (SSD). The method detects objects in images using a single deep neural network, which means the entire process of SSD requires only one step. It uses a multiscale feature map to a priori detect and set a box for target detection. It is a relatively simple algorithm compared to methods that require object proposals because it eliminates proposal generation, and uses a single network for the computation. This makes it easy to train and straightforward to integrate (Bai et al., 2020).

The You Only Look Once (YOLO) algorithm was proposed by Redmon et al., 2016. By transforming the object detection problem into a regression problem, they were also able to use a single neural network to perform object detection in one evaluation. This is done by to spacially separating bounding boxes and associated class probabilities, by using a Convolutional Neural Network (CNN) structure. High detection accuracies and fast computation speeds can be achieved by using this method. The SSD algorithm is much better than YOLOv1 in accuracy and speeds, and SSD directly uses the convolution layer in the last layer to extract the detection results of different feature maps (Bai et al., 2020). Recently an improved version of the SSD has been released by Zhai et al., 2020, showing better accuracy than the previous version, even with a smaller input image size.

However over the years, the YOLO algorithm has been subject to multiple improvements, leading to the release of newer versions. YOLOv3 which was released in 2018 integrated some advantages of similar algorithms such as the feature pyramid network (FPN) and the Fast-Region-based CNN (RCNN) and runs as accurate as SSD but three times faster (Redmon and Farhadi, 2018; Bai et al., 2020).

The Mask Region-Based Convolutional Neural Network (Mask-RCNN) proposed by He et al., 2017 is different from the two previously mentioned algorithms in the fact that it is a two- instead of 'one-shot' approach. This means that instead of one single network used for the detection of bounding boxes and class probabilities, it utilizes two different models. It is an extended version of the Faster R-CNN algorithm Ren et al., 2017, which is combined with a fully connected network (FCN) used for semantic segmentation. Region proposal networks (RPN) are used to generate candidate regions and train an RPN and Fast RCNN to share a convolutional layer, which greatly improves the detection speed of the network. However it takes a considerable amount of time to generate candidate regions, which also affects the detection performance (Bai et al., 2020). Recently, an improved version of the algorithm has been released by Zhang et al., 2021, which was tested on x-ray images and showed promising results.

Although it has been shown that the YOLO algorithm is faster with regard to execution speed, previous research has also shown that the Mask-RCNN algorithm has a higher detection accuracy (Bharati and Pramanik, 2020; Dorrer and Tolmacheva, 2020). Furthermore, Buric, Pobar, and Ivasic-Kos, 2018 found that YOLO has more trouble with the recognition of occluded objects. Another reason for choosing the Mask-RCNN algorithm is the fact that it is able to generate segmentation masks for the detected objects. Based on these conclusions and the additional functionality, we choose to implement the Mask-RCNN algorithm in this research.

2.2 Grasping

Knowing how to grasp an object is an important part in the process of manipulating objects in an efficient and productive way. Most state-of-the-art grasping approaches addressed four degrees-of-freedom (DoF) object grasping, where the robot is forced to grasp objects from above based on grasp synthesis of a given top-down scene, although methods using six DoF also exist.

Morrison, Corke, and Leitner, 2018 presented an object-independent grasp synthesis method that can be used for closed-loop grasping. The Generative Grasping Convolutional Neural Network (GG-CNN) predicts the quality and pose of grasps at every pixel, improving computation times and grasp accuracy in non-static environments. The grasp success achieved ranged from 83% to 88%, with also scoring a high frequency of grasp pose generation at a rate of 50 Hz.

The Volumetric Grasping Network (VGN) predicts six DoF grasps from 3D scene information (Breyer et al., 2021). The network uses a Truncated Signed Distance Function (TSDF) as a representation of the visual input, making it suitable for feature learning with deep neural networks. After training a FCN, the experiments show a computationally fast grasping network that enables six DoF grasp synthesis in real-time. Apart from that, it is able to learn collision-free grasp proposals due to the full 3D scene information that is used in the network.

Another deep learning approach to handle real-time object grasping in more human-centric domains is MVGrasp, proposed by Kasaei and Kasaei, 2022. Multi-view depth images are generated from a partial point cloud input, subsequently, the best view is selected by a view function. This is then fed into a deep network to estimate a pixel-wise grasp synthesis, which encloses grasp quality, orientation, and width of the grasp. The evaluation is performed on highly cluttered scenarios, with the experimental results showing that the proposed system outperforms the state-of-the-art (amongst others, GG-CNN) in packed and piled situations.

Another approach is presented by Kumra, Joshi, and Sahin, 2020 in the form of a Generative Residual Convolutional Neural Network (GR-ConvNet). It consists of a modular solution for grasping novel objects, using n-channel input data to generate images that can be used to infer grasp rectangles for each pixel in an image. The system obtained high accuracy grasp scores for two datasets, respectively 95.6% and 93%. Showing the ability to predict and perform accurate grasps for previously unseen objects, with low inference time.

The GR-ConvNet outperforms the GG-CNN with regard to grasp accuracy, and shows good performance in different scenarios (Kumra, Joshi, and Sahin, 2020). It needs less complex input and preprocessing compared to the VGN (Breyer et al., 2021) and MVGrasp (Kasaei and Kasaei, 2022) algorithm, which is why in this research we choose to implement an approach using GR-ConvNet. This means we will approach the problem with a top-down grasping policy.

2.3 Cluttered Environments and Active Segmentation

In a highly cluttered scenario, both object detection and grasping might be impaired due to objects occluding the robot's perception, or blocking its grasping trajectory. In these cases, an active segmentation step can be executed in order to singulate objects from the clutter by means of pushing or other movements. This aims at improving the situation, optimally resulting in a clear view of the object and a collision-free grasp.

With regard to occluded objects (Yang, Liang, and Choi, 2020) propose a deep learning approach, trained by self-supervision in simulation, to find a block that is buried in clutter. Using two policies to tackle the problem, the first is aimed at explorational target-oriented pushing of the blocks based on a Q-learning policy. Once the target is visible, a classifier-based policy that takes the clutteredness around the target as an input, coordinates decision making in pushing or grasping. The deep learning approach in a critic-policy format shows high success rates in both simulation and real robot experiments.

Another approach based on deep reinforcement learning is discussed by Kurenkov et al., 2020. A novel procedure combining three algorithmic strategies allows for sample efficient and effective learning of the problem of uncovering a target object occluded by a heap of unknown objects. A robot is tasked with uncovering a target object from occluding objects on top using pushing actions based on the images acquired by an RGB-D sensor. By means of teacher-aided exploration, a privileged information critic, and mid-level representations, the experiments show faster training and more efficient converging uncovering solutions compared to baselines.

Six DoF pushing and grasping policies can also be learned by making use of the Q-learning framework Tang et al., 2021. A robot is trained to learn joint planar pushing, and grasping motions using two separate networks with three dimensional visual input. The system is able to deal with cluttered scenes in the form a messy dinner table with a wide variety of objects. Using pregrasp pushing actions for declutttering, the approach is able to outperform the state-of-the-art baseline model (Zeng et al., 2018), in terms of action efficiency and grasp success rate.

Research done by Kiatos et al., 2022 is aimed at planning a stable grasp in densely cluttered environments by means of employing a single push-grasping action. Based on a visual input a function is learned that can output multiple strategies for the robotic hand, such as inwards and outwards rolling movements, and multi-fingered pushes. The policy that is proposed aims at creating enough space for the fingers to wrap around an object using a FCN for optimal pose prediction and a CNN for robotic hand aperture prediction. Decoupling these components yields efficient learning, because the networks focus on individual parts of the operation. High success rates in the simulation and a robust transfer to a real environment is achieved for the learned policy.

The implementation in this research is different from the previously mentioned approaches, we choose to inject object segmentation into the object grasping pipeling to get rid of unnecessary pushes. Operating on a highly cluttered scenario, it is aimed at improving the performance of the designed model with a clear-cut single object isolation motion. Most cluttered approaches are either: not target driven, or operated on simple objects, or not highly cluttered. The approach taken in this research combines all these properties: resulting in a method aimed at target driven grasping for relatively complex objects in highly cluttered scenarios.

2.4 Domain Randomization

Domain randomization (DR) is a technique mostly used to improve performance of appliances bound for a transfer of domains. The randomization of parameters in the training stage aims at desensitizing the system for changes in environment, possibly making its performance more robust towards unencountered conditions. Furthermore, DR can be convenient with regard to data generation, which is helpful in situations with sparsely available training data.

The divide that exists between robots operating in simulated environments and experiments performed on hardware is called the 'reality gap'. Bridging this gap could accelerate robotic research (Tobin et al., 2017). This research uses randomized rendering in a simulation environment with non-realistic textures to train a model, before transfering it to real images. It is focused on object localization, a main component of robotic manipulation systems. They were able to train an object detector solely using simulated images, that displayed accurate real-world object detection to 1.5 cm, robust to distractors and partial occlusions.

Generalization of robotic grasping models is a challenge, due to the amount of training data that is necessary. Tobin et al., 2018 explore a novel data generation approach used for training a deep neural network. The network is used for grasp planning, and DR is applied to object synthesis. This is done by generating millions of unrealistic objects used for training the network to perform grasp planning. Despite having only been trained on random objects, a high grasp success rate is achieved at unseen realistic objects in simlated and real-world tests.

Dehban et al., 2019 showed that by training on a dataset constructed with DR, object detection methods can show substantial improvements in accuracy. Models that were pretrained on standard datasets and fine-tuned with domain-relevant images are compared to models that were trained using synthetic datasets. The conclusions are that the latter, while the images not being photo-realistic, can be a better alternative with regard to mean average prediction scores than fine-tuning a pre-trained model.

Research done by Mao et al., 2021 also shows that object detection can be improved by using DR, in this case applied to the automatic detection of birds. Due to lack of training data and difficulties in extracting fine-grained features used to differentiate bird species, the accuracy of existing ornithological analysis models is limited. By using DR in the training phase, the accuracy of the deep learning model is improved. This is done by training on virtual birds with variations in different environments, leading to a model that tends to focus more on the fine-grained features of a bird, achieving higher accuracy scores.

As in the two studies lastly mentioned, in this research domain randomization is employed in order to improve the performance of the object detection algorithm. By randomizing parameters in the training data used for the Mask-RCNN, the aim is to improve detection accuracy on the objects that are to be segmented and grasped respectively.

Chapter 3

Methodology

In this section the three main components of this research are described in detail. Multiple varieties of these components can be used, which is why we focused on a modular approach. This means each of these elements can be interchanged with a counterpart with comparable functionality.

3.1 Mask-RCNN

3.1.1 Background

The Mask-RCNN framework proposed by He et al., 2017 is capable of accurate object detection as well as generating a segmentation mask for each instance, and is an extension of Faster R-CNN (Ren et al., 2017). Faster R-CNN is an algorithm that is able to classify objects in an image, but unable to locate the pixels associated to the specific object. The latter functionality is called instance segmentation. The Faster R-CNN algorithm has two outputs for each candidate object, a class label and a bounding box. Instance segmentation in Mask-RCNN is performed at pixel level by adding a separate branch in the architecture which operates in parallel with the Faster R-CNN classifier. The Mask-RCNN algorithm is a procedure composed of two stages, of which the architecture is depicted in Figure 3.1.

The first stage consists of a feature extractor with ResNet-101 (He et al., 2016) and a Feature Pyramid Network (FPN) (Lin et al., 2017) as a backbone. The latter utilizes a top-down architecture with lateral connections to build high-level semantic feature maps at all scales. These feature maps are able to detect objects with different sizes. The backbone is connected to a Region Proposal Network (RPN) as described by Ren et al., 2017. The RPN generates a batch of the region of interests (RoIs) according to the anchors provided by the FPN. Subsequently, features of these RoIs are extracted by using the RoIAlign layer. This is an improved version of the RoI pooling layer, used in Faster R-CNN. The improvement is necessary because pixel level segmentation needs much more fine-grained alignment than bounding box generation. Bilinear interpolation is used for avoiding misalignment of location due to coarse spatial quantization. This quantization-free layer is called RoIAlign, and faithfully preserves exact spatial locations (He et al., 2017; Bharati and Pramanik, 2020). This results in a set of bounding box proposals for the candidate objects.

The second stage consists of a Faster R-CNN classifier and a binary mask prediction branch. The detected features are fed into the classifier, which performs object detection and classification. It produces softmax probability estimations-, and uses bounding box regression to output bounding box coordinates for each of the detected objects. In parallel to this, the RoIs are fed into the mask prediction branch for semantic segmentation. The segmentation masks are predicted by using a Fully Connected Network (FCN) made up of four convolution layers and one deconvolution layer. By applying the FCN to each RoI, segmentation masks are predicted in a pixel-to-pixel manner (He et al., 2017), resulting in a binary mask for each candidate box.

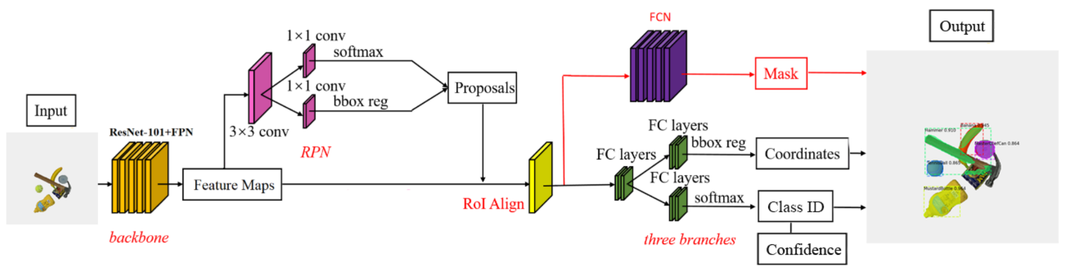


FIGURE 3.1: Mask-RCNN architecture, adapted from Yu et al., 2019.

According to He et al., 2017, the decoupling of mask and class prediction was essential with regard to accurate performance of the approach. The binary mask is predicted for each class independently, without competition among classes, and the classification branch uses the RoIs to predict the category. This in contrast to the the per-pixel multi-class categorization usually performed by FCNs, coupling segmentation and classification. This works poorly for instance segmentation according to experiments performed by He et al., 2017.

The Mask-RCNN used in this research is based on the implementation[1] by Matterport (Abdulla, 2017). It is build in Python using a TensorFlow[2] and Keras[3] framework as backend.

3.1.2 Training

The Mask-RCNN has been specifically trained on the custom data used in this research. The custom data used is part of the YCB dataset (Calli et al., 2017), which contains simulated objects and is described in more detail in Chapter 4. Training is performed using the GPU hardware accelerated runtime that is made available by Google[4]. Since the used backbone is the ResNet-101 network, the training stage consists of retraining the network heads on the custom data in order to finetune the network.

This training data consists of 100 images for each of the 16 objects, adding up to a total of 1600 training images. These images have been obtained by collecting RGB data from the simulated objects, separately spawned in positions that are initiated randomly, resulting in a set of images in which the object is captured from multiple angles. The images are saved with metadata containing the class label, and binary mask values. The exact training parameter configuration is disclosed in Appendix A.1.

[1] `https://github.com/matterport/Mask_RCNN`
[2] `https://www.tensorflow.org/`
[3] `https://keras.io/`
[4] `https://colab.research.google.com/`

Two different variations of training data have been used for training the network: one using standard data, and one using data that has been subject to a form of Domain Randomization. The former consists of an object with a plain background, whereas the latter consists of an object on a custom background. Examples of the images used as training data are depicted in Figure 3.2. Apart from being subject to pre-training Domain Randomization, Data Augmentation is performed on the images over the course of the training phase.

Domain Randomization

In an attempt to improve the object detection performance of the Mask R-CNN algorithm, a form of Domain Randomization is implemented and executed on the training data. The objects images are enriched by adding a custom background. This underlay is randomly selected from a set of 252 textures during the data generation phase. The set of reference textures is courtesy of The Vision and Modeling Group of the Massachusetts Institute of Technology Media Laboratory and belongs to the VisTex database[5]. By randomizing the background in the training data used for the Mask-RCNN, the aim is to improve detection accuracy on the objects that are to be segmented and grasped respectively. Examples of these training images are depicted in Figure 3.2b.

Data Augmentation

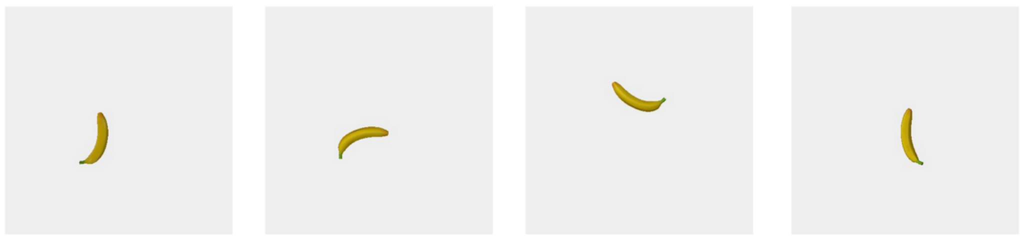
Deep convolutional neural networks can perform very well on computer vision tasks. However, they are heavily reliant on large amounts of data to prevent overfitting behaviour. Overfitting occurs when a network learns a function with very high variance, perfectly modelling the training data. This is at the expense of the accuracy of the model on test data, which is novel to the network and therefore the main interest with regard to performance. Generalizability refers to the difference in performance of a model evaluated on training data versus testing data. A means of improving generalizability and decreasing the chance of overfitting is data augmentation (Shorten and Khoshgoftaar, 2019).

Augmentation	Details	Probability
Horizontal Flip	-	0.5
Rotation	One of (90°, 180°, 270°)	1
Scale X	Between (-20, +20 %)	1
Scale Y	Between (-20, +20 %)	1
Random Crop	Between (0, 20 %)	1
Brightness Shift	Between (-20, +50 %)	1
Contrast Shift	Between (-25, +50 %)	1
Gaussian Noise	Between (0, 5 %)	Pixelwise
Gaussian Blur	Between (0, 0.25)	0.5

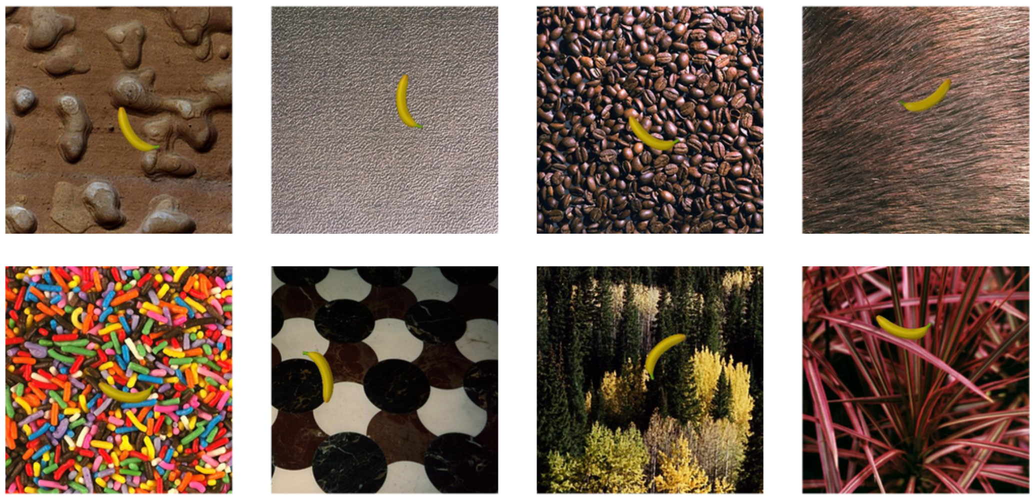
TABLE 3.1: Implemented data augmentations, with detailed information, and probability of execution.

The augmentation algorithm[6] we use is courtesy of Jung et al., 2020. It is is applied during the training phase, and converts a set of input images to a new, larger set of slightly altered images. The augmentation is also applied to the binary mask. In the case of this research we chose for a nine augmentations which are applied

[5] `https://vismod.media.mit.edu/vismod/imagery/VisionTexture/vistex.html`
[6] `https://github.com/aleju/imgaug`



(a) Standard



(b) With Domain Randomization

FIGURE 3.2: Examples of training data: (a) shows images of the banana object from the standard data set, with a plain background; (b) shows images of the banana objects from the data set constructed with domain randomization, with texturized backgrounds.

sequentially in a random order. For each training image, the probability of the augmentation sequence being applied is $\frac{2}{3}$. Furthermore, within this sequence each augmentation has its own probability of application, and parameters that describe the intensity of the transformation. The augmentations and their details are described in Table 3.1, and an example for each of them is depicted in Figure 3.3.

3.2 GR-ConvNet

3.2.1 Background

The GR-ConvNet is a generative residual convolutional neural network architecture that is able to predict suitable antipodal grasp configurations for objects within the field of view of a fixed camera (Kumra, Joshi, and Sahin, 2020). Antipodal points are a pair of points on an object whose normal vectors are collinear and in opposite direction, effectively meaning grasps with two-fingered actuators.

It uses an n-channel input image provided by the camera, and generates pixel-wise grasps in the form of three images. Subsequently the algorithm is able to convert these from image space to robot space, and instruce a robot to execute a predicted

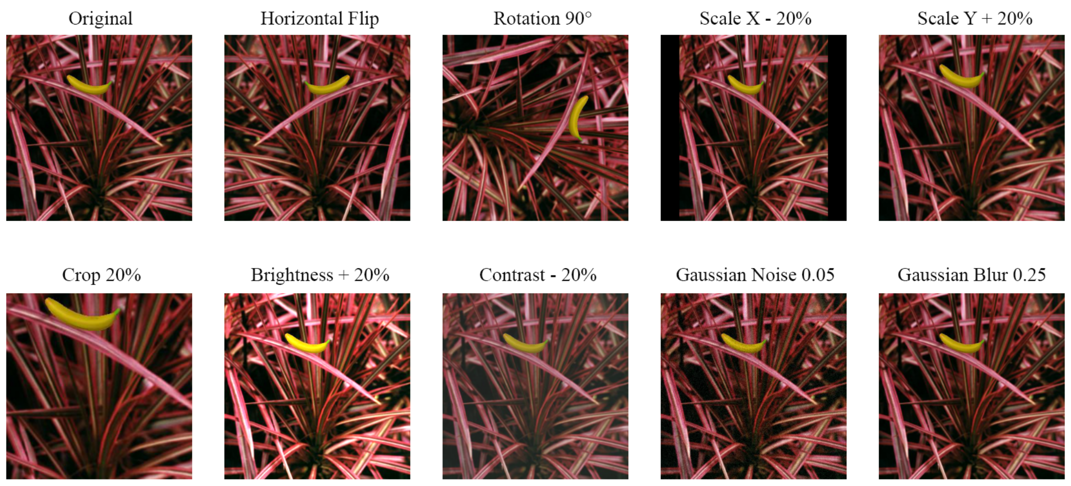


FIGURE 3.3: Augmented training images with one example for each augmentation.

grasp. The latter is done by applying transformations to convert the image coordinates to robot's frame of reference. The camera used in this research is able to provide both an RGB- as a Depth image, which are inpainted to obtain a depth representation as described by Xue, Zhang, and Cai, 2017. This input image is 224x224 pixels and is fed into the network.

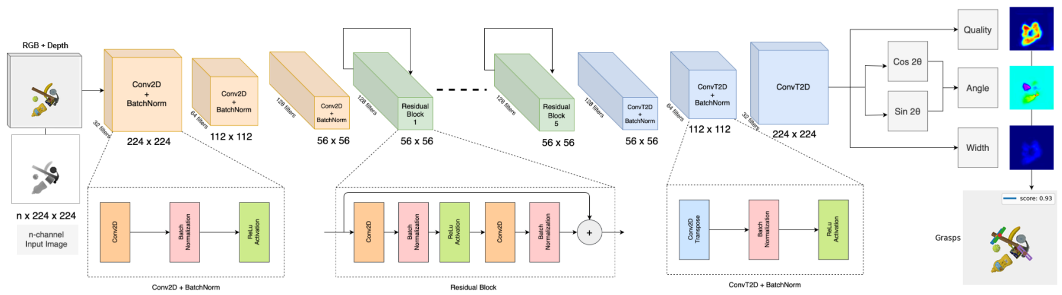


FIGURE 3.4: GR-ConvNet architecture, adapted from Kumra, Joshi, and Sahin, 2020.

The inference module consists of three parts. The first is a pre-processing component which is used for resizing and cropping if necessary, and normalizing of the input. The second part is an encoder-decoder architecture that uses convolution, deconvolution, and batch normalization for feature extraction and generating three output images as grasp angle, grasp width, and grasp quality score. The achitecture is depicted in Figure 3.4, and starts with three layers of convolutions, followed by five residual layers, and concluded with three layers of deconvolutions, forming an autoencoder network.

The convolutional layers extract the features from the input image. Each convolutional block consists of a two-dimensional convolutional layer with batch normalization and a Rectified Linear Unit (ReLu) activation function. The output of this is fed into the residual layers, which are used for retaining accuracy in a deep neural network. These layers are enabling the model to better learn identity functions and skip connections, while surpassing the problem of vanishing gradients and dimensionality errors (Kumra, Joshi, and Sahin, 2020). After these layers the size of the image has shrunk to 56x56 pixels due to the applied convolutions. In order to make interpretation of the image easier, it is upsampled using three layers of transposed convolutions, obtaining an output with the same size as the input image.

This output consists of three images of a score for the grasp quality, the required width for the end effector, and the angle at which it is required to operate. Smoothing is then applied to these images by means of a Gaussian function. These images contain the grasp quality score, grasping width, and angular rotation for all pixels in the input image. The third component of the model is then used to infer one or multiple grasps from the obtained features. A cascaded transformation is then used to convert the predicted grasp from image- into robot space. This is done by using the intrinsic camera parameters to first transforming from image space into three-dimensional camera space. Subsequently, the camera position and base position of the robot are used to transform camera space into robot space, after which the grasp is ready for execution.

The GR-ConvNet algorithm is implemented in Python using a PyTorch[7] framework as backend.

3.2.2 Training

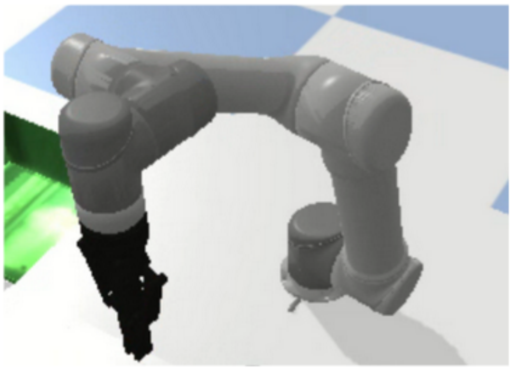
For this research a GR-ConvNet model implementation is employed with pre-trained network weights[8], as was also used in research done by Santhakumar and Kasaei, 2022. It has been trained on the Cornell grasping dataset, which contains 1035 RGB-D images of 240 objects. In this dataset, 5110 valid- and 2909 invalid grasps have been annotated, consisting of several grasp rectangles representing grasping possibilities per object. Kumra, Joshi, and Sahin, 2020 applied data augmentation in the form of random crops, zooms, and rotations on the images, in order to increase the amount of training data. This resulted in 51k grasp examples, of which only positive labeled i.e. successful grasps were used for training the network.

3.3 Active Segmentation

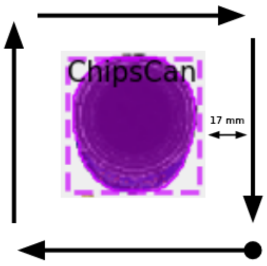
The Mask R-CNN network is capable of applying a combination of object detection and instance segmentation. The information about the location of a possible target object is then transferred to the GR-ConvNet algorithm for the grasping stage, the details of this process are described in the next section. But in a highly cluttered scenario grasping might be unfeasible due to objects occluding the robot's perception, or blocking its grasping trajectory. In these cases the passive segmentation technique employed is not sufficient for the model's operational success, and the situation can be improved by executing an active segmentation protocol, aimed at singulating an object from the clutter.

We have devised a hand crafted policy capable of isolating an object with an exact movement trajectory. Many datadriven mehtods can generalize the dynamics of pushing, but they require explicit object modeling and are highly sensitive to parameters. Furthermore, complex pushing procedures can be complicated to perform in a highly cluttered workspace, which is why we chose a more elementary implementation. The trajectory that is to be taken, is based on the output of the Mask-RCNN, with one specific object targeted for singulation. First off, the two fingered gripper

[7]https://pytorch.org/
[8]https://github.com/skumra/robotic-grasping



(a) The orientation of the gripper, the fingers are closed.



(b) The motion executed with regard to the object targeted for segmentation.

FIGURE 3.5: The main components of the implemented segmentation step in this approach.

is closed, as depicted in Figure 3.5a making the actuator more agile, and its coordinated movements more precise. Subsequently, the coordinates of of the bounding box are passed to the segmentation module. The bounding box provided by the Mask-RCNN is often close-fitting around the object, which is why a buffer is initiated. This is done by padding the bounding box on all sides with 17 pixels in grasping space, aimed at decreasing the chance of hitting the targeted object. The gripper is then moved to the bottom right corner of the bounding box, and follows the contours of all four sides of the box, resulting in a rectangular movement which is depicted in Figure 3.5b. Optimally, performing the segmentation step results in a clear view of the object and a collision-free grasp. An overview of the full process of the implemented segmentation step, including the events leading up to it, and a resulting succesful targeted grasp and drop are depicted in Figure 3.6. The relatively uncomplicated segmentation step aims at trying to achieve an approach that utilizes synergistic behaviour between the grasping- and segmentation module.

3.4 Model

This section describes an overview of the model, the implementation of which is available on GitHub [9]. It is comprised of the three components that were described in the previous sections, which can be regarded as separate modules. A overview of the model architecture is depicted in Figure 3.7. The method we take is a State Machine approach (Schneider, 1990) in which during execution, decisions are made depending on the current state of the machine. These states are altered, depending on multiple factors regarding succeeded or failed (sub)tasks, and the condition of the environment. An overview of the possible states is reported in Table 3.2.

The principal architecture of the algorithm is described in Algorithm 1, it takes a list of target objects, runs the requested number of experiments for each object, and saves the result in a dataframe. The model is put together by a main loop that connects the three modules described in the previous sections. This is the inner while loop described in Algorithm 1, within this loop, the model operates and information is exchanged.

[9] https://github.com/ivarmak/clutter-grasping

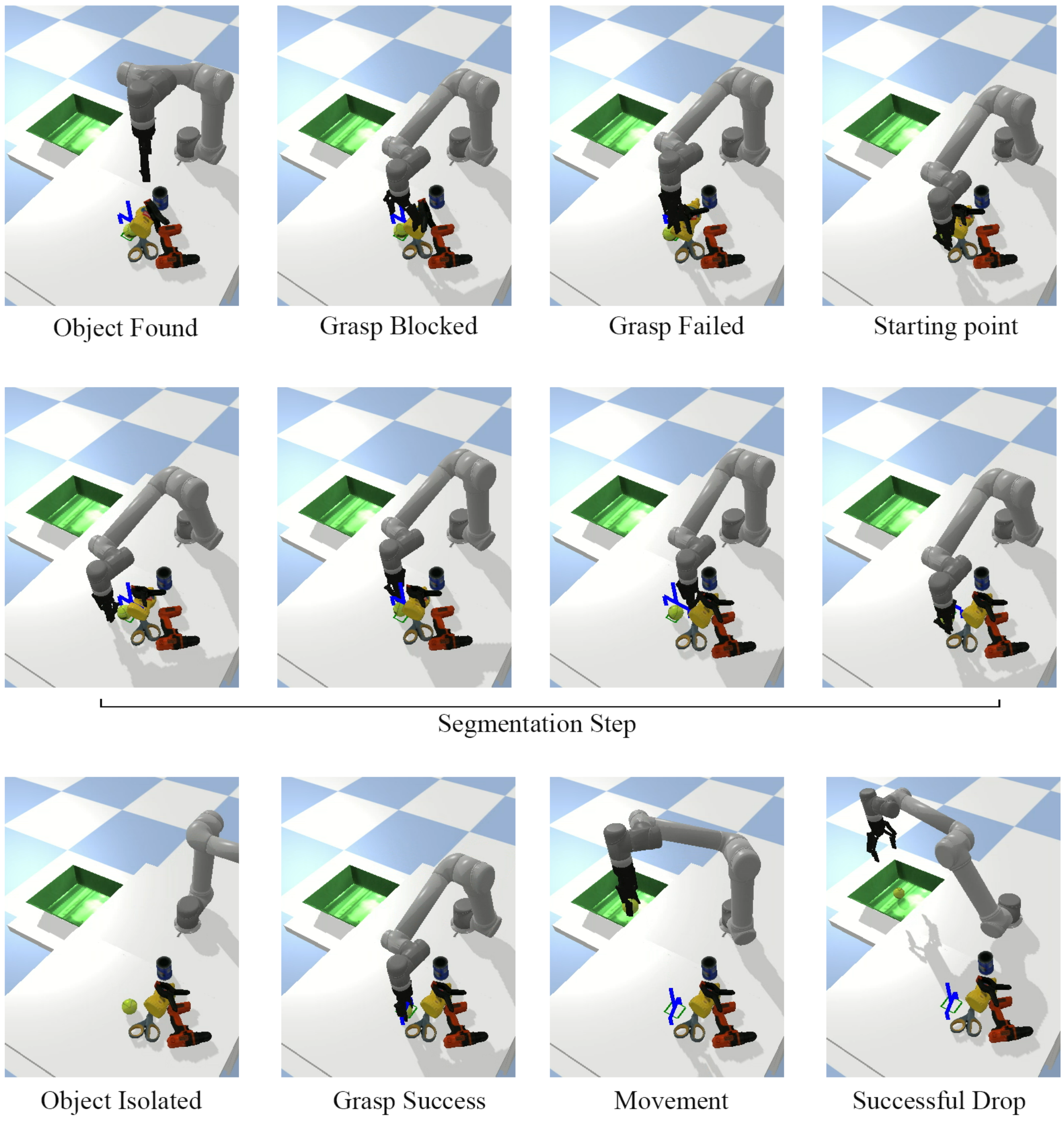


FIGURE 3.6: Overview of the segmentation process, with the Tennis-Ball as target object, its bounding box obtained from the Mask-RCNN in depicted in green, and the grasp predicted by the GR-ConvNet depicted in blue.

The main loop consists of three phases: the Recognition phase, the Grasping phase, and the Analysis phase. The Recognition phase, described in detail in Algorithm 2, is where the object detection module resides. The Mask-RCNN algorithm is ran on RGB data of the workspace provided by the camera, and the output of the network determines the model state. The *State* is determined by whether the target object T has been detected, and forthcoming actions are dependent on its value.

Depending on the output of the Mask-RCNN there are two options, either the algorithm has detected one or more objects, or it has not detected any objects. In the case of the former, the grasping phase will be commenced aimed at a specific object. In the latter situation, the grasping phase will have the freedom to grasp any object possible. In both cases there is a specific drop-off destination, one of the trays located on the side of the workspace, or the recognition area which is located in the center of the workspace directly below the camera. If the algorithm has had an iteration

State
Idle
TargetFound
NonTargetFound
NothingFound
TargetGrasp
NonTargetGrasp
MovedToRecogArea

TABLE 3.2: Overview of possible model states.

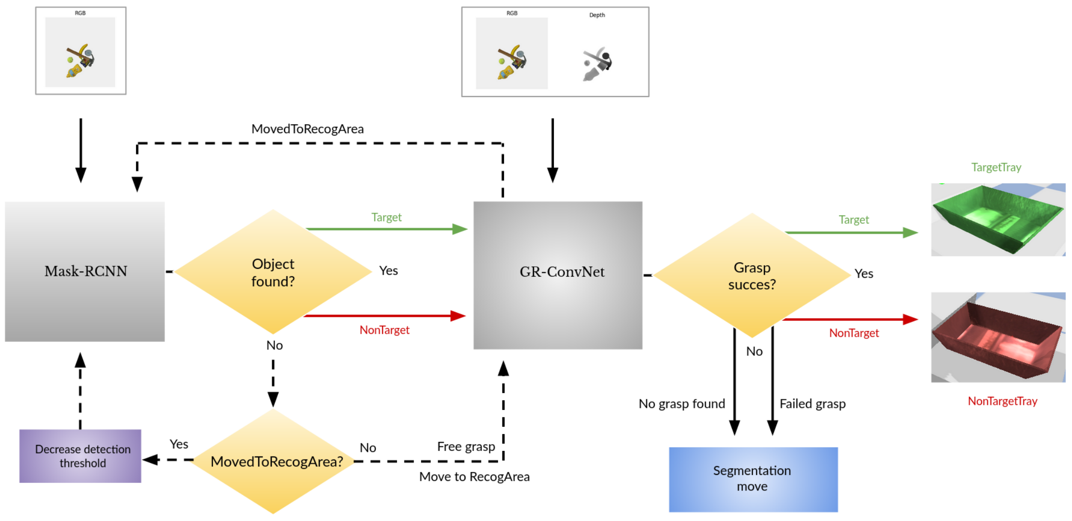


FIGURE 3.7: Overview of the model architecture: the object recognition module initiates the pipeline, where the Mask-RCNN determines whether an object is recognized in the detection area. If no object is recognized, the model will move an object to the recognition area and if no object is detected again, decrease its detection threshold. If one or more objects are detected, the targeted graping phase is entered. The GR-ConvNet will detect possible grasp configurations, and move the object to its corresponding location. If no grasps are detected-, or the execution of the grasp has failed, a segmentation move is performed and the model sequence is restarted.

where no object was found, an object was placed in the recognition area, and it was again unable to detect any objects, the Mask-RCNN's detection threshold is lowered with 0.1.

The Grasping phase is described in Algorithm 3, and it contains the grasping module as well as a link to the segmentation module. First off, the GR-ConvNet algorithm is ran on the RGB and Depth data combined with parameters that were transferred from the recognition phase. In the case of a grasped aimed at an object, the grasping region *GR*, in which the algorithm is able to detect grasps, has been dictated by the detected mask and bounding box of this object. In the case of a free grasp, the *GR* contents include the complete workspace. Running grasping algorithm will generate a set of possible grasps, of which the grasping index *Gidx* determines which will be performed. The results with regard to success or failure of the grasp and of the placement at the destination are stored. When no grasps are detected the algorithm will increment a counter which will eventually determine a model halt when too

Algorithm 1: Main Loop

```
Input: Set of target objects $ObjectList$, Number of experiments to be performed
       per object $NumberOfExperiments$
Output: Experimental results saved in dataframe
Initialize RGB-D camera;
for $T$ in $ObjectList$ do
    for $exp$ in $NumberOfExperiments$ do
        $Gidx \leftarrow 0$;
        $FailFindGrC \leftarrow 0$;
        $SegStepC \leftarrow 0$ $Thr \leftarrow 0.85$;
        while Experiment Success is False AND Experiment Fail is False do
            Run Recognition: Algorithm 2;
            Run Grasping: Algorithm 3;
            Run Analysis: Algorithm 4;
            $TargetOutsideDetectionArea \leftarrow$ Check if $T$ is outside boundaries of
              workspace;
            if TargetOutsideDetectionArea is True then
                $ExperimentFail \leftarrow True$;
                Break;
            end
            else if $SegStepC > 10$ then
                $ExperimentFail \leftarrow True$;
                Break;
            end
        end
        Save results;
    end
end
```

high. Furthermore, in the case of an aimed grasp, it will commence a segmentation step with the aim of singulating the object. Afterwards the model is returned to the Recognition phase.

After a detected and performed grasp, the Analysis phase is entered, which is described in Algorithm 4. In this phase, the results of the grasping move are analyzed, and decisions are made accordingly. If an aimed grasp was unsuccessful, the model will determine whether the next grasp in the set collides with another object. In that case, a segmentation step is commenced. If not, or in the case of a free grasping procedure, the grasp index *Gidx* is incremented leading to another grasp to be performed in the next iteration of the algorithm. In the case of a successful grasp, a number of checks take place with regards to the drop-off destination. These checks will determine whether the model needs to halt, or can continue to a new recognition phase.

Algorithm 2: Recognition Phase

```
Input: Camera data $RGB$, Targeted object $T$, Mask-RCNN detection threshold
       $Thr$
Output: Machine state $State$, Grasping region $GR$, Destination $D$, Set of object
        masks $Masks$, Bounding box $B$
$State$, $ObjectsInfo$ ← run Mask-RCNN algorithm with image $RGB$, threshold
  $Thr$, and target $T$;
$Masks$ ← from $ObjectsInfo$ take detected object masks;
if $State$ is $TargetFound$ then
    $State \leftarrow TargetGrasp$;
    $D$ ← coordinates of $TargetTray$;
    $B$ ← from $ObjectsInfo$ take coordinates of $TargetBox$;
    $M$ ← from $Masks$ take binary mask $TargetMask$;
    $GR$ ← binary operation $M$, $B$ to achieve grasping region of Target;
end
else if $State$ is $NonTargetFound$ then
    $State \leftarrow NonTargetGrasp$;
    $D$ ← coordinates of $NonTargetTray$;
    $B$ ← from $ObjectsInfo$ take coordinates of $NonTargetBox$;
    $M$ ← from $Masks$ take binary mask $NonTargetMask$;
    $GR$ ← binary operation $M$, $B$ to achieve grasping region of NonTarget;
end
else if $State$ is $MovedToRecogArea$ then
    ;    /* No object found, state remained from previous iteration */
    $Thr \leftarrow (Thr - 0.1)$;
    if $Thr < 0.45$ then
        $ExperimentFailed \leftarrow True$;
        Break;
    end
end
else
    $State \leftarrow NothingFound$;
    $GR$ ← Complete workspace;
    $D$ ← coordinates of $RecognitionArea$;
end
Go to Algorithm 3;
```

Algorithm 3: Grasping Phase

```
Input: Camera data $RGB$ and $Depth$, Machine state $State$, Grasping Region $GR$,
       Destination $D$, Bounding Box $B$, Index for choosing from grasp
       alternatives $Gidx$, Counter for number of times no grasp has been
       detected $FailFindGrC$, Counter for number of times segmentation step
       has been performed $SegStepC$
Output: Machine state $State$, Set of detected grasps $Grasps$, Booleans
        $SuccesGrasp$ and $SuccessDestination$, Counter for number of times no
        grasp has been detected $FailFindGrC$, Counter for number of times
        segmentation step has been performed $SegStepC$
$Grasps \leftarrow$ run GR-ConvNet algorithm with $RGB$, $Depth$ images on $GR$;
if Grasps is not empty then
    $FailFindGrC \leftarrow 0$;
    if State is TargetGrasp or NonTargetGrasp then
        $SuccessGrasp, SuccessDestination \leftarrow$ Perform grasping move
          $Grasps[Gidx]$ towards destination $D$;
    end
    else if State is NothingFound then
        $SuccessGrasp, SuccessDestination \leftarrow$ Perform grasping move
          $Grasps[Gidx]$ towards destination $D$;
        $State \leftarrow MovedToRecogArea$;
    end
end
else
    ;                                            /* No grasps are detected */
    if $FailFindGrC > 3$ then
        $ExperimentFail \leftarrow True$;
        Break;
    end
    if State is TargetGrap or NonTargetGrasp then
        Run $SegmentationStep$ on edges of $B$;
        $SegStepC \leftarrow SegStepC + 1$;
    end
    $FailFindGrC \leftarrow (FailFindGrC + 1)$;
    Go to Algorithm 2;
end
Go to Algorithm 4
```

Algorithm 4: Analysis Phase

Input: Machine state *State*, Bounding Box *B*, Set of object Masks *Masks*, Target object *T*, Set of detected grasps *Grasps*, Index for choosing from grasp alternatives *Gidx*, Booleans *SuccesGrasp* and *SuccessDestination*, Counter for number of times segmentation step has been performed *SegStepC*

Output: Index for choosing from grasp alternatives *Gidx*, Counter for number of times segmentation step has been performed *SegStepC*

```
NumberOfGrasps ← length(Grasps);
if SuccessGrasp is True then
    Gidx ← 0
end
else if SuccessGrasp is False then
    if State is TargetGRasp or nonTargetGrasp then
        if Gidx < NumberOfGrasps then
            Intersect ← Check if next grasp Grasps[Gidx+1] intersects with other
              object masks Masks;
            if Intersect is True then
                ;        /* Next grasp in set collides with other object,
                 isolate object aimed for.  */
                Run SegmentationStep on edges of B;
                SegStepC ← SegStepC + 1;
                Gidx ← 0
            end
        end
    end
    if Gidx < NumberOfGrasps then
        ;    /* Another detected grasp is available, increment index */
        Gidx ← (Gidx + 1)
    end
end
if SuccessDestination is True then
    TargetObjectGrasped ← check if target object T is grasped;
    if State is TargetGrasp then
        if TargetObjectGrasped is True then
            ExperimentSuccess ← True;
            Break;
        end
        else
            ;                              /* NonTarget in TargetTray */
            ExperimentFail ← True;
            Break;
        end
    end
    else if State is NonTargetGrasp then
        if TargetObjectGrasped is True then
            ;                              /* Target in NonTargetTray */
            ExperimentFail ← True;
            Break;
        end
    end
end
```

Chapter 4

Experiments and Results

The performance of the model in real-time object detection, classification, and grasping is evaluated using a simulated robot environment. In this environment multiple objects are spawned at predetermined or semi-random locations after which the intended tasks are performed. Different scenarios determine the number of objects and their subsequent location on the workspace.

4.1 Simulation

The simulated environment used in this research is designed in Python using the PyBullet[1] engine, developed by Coumans and Bai, 2016–2021. The contents of the simulation are adapted from work produced by Vrielink and Kasaei, 2021. Its components are shown in Figure 4.1, consisting of: a UR5e based robotic arm with a two-fingered gripper actuator attached, a workspace in the form of a table, one tray on either side of the table meant for object placement, and a RGB-D camera aimed at the workspace with a corresponding detection area.

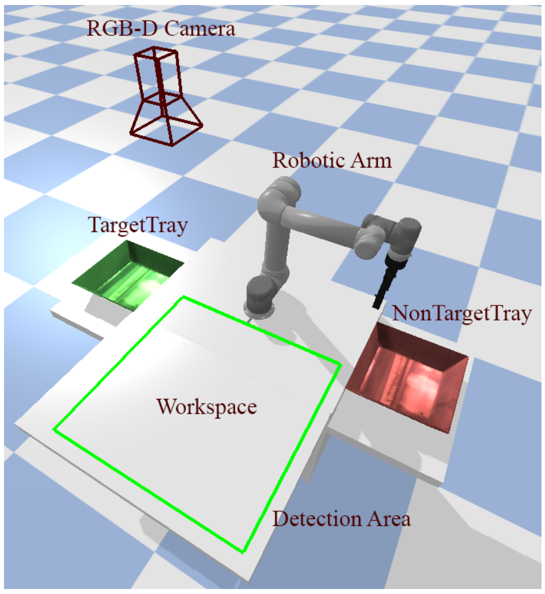


FIGURE 4.1: Environmental setup in the PyBullet simulated environment, with labeled components.

[1]https://pybullet.org/wordpress/

4.2 Dataset

A selection of 3D objects from the YCB-library[2] are used to create the dataset (Calli et al., 2017). It comprises of 16 daily life object instances with different shapes, sizes, textures, weight, and rigidity. They are based on physical objects, and form a database of mesh models and high-resolution RGB-D scans that can be imported into the simulation environment. An list of the objects that were used is provided in Table 4.1.

Objects			
Banana	ChipsCan	CrackerBox	FoamBrick
GelatinBox	Hammer	MasterChefCan	MediumClamp
MustardBottle	Pear	PottedMeatCan	PowerDrill
Scissors	Strawberry	TennisBall	TomatoSoupCan

TABLE 4.1: Overview of YCB-objectst that are used.

4.3 Experiments

The simulated robot experiments consist of using the model for combined object detection and segmentation in order to perform target driven object grasping. The implementation described, and proposed architecture, as described in the previous chapter are used with as main objective: place the target object in the target tray. If the algorithm succeeds in this objective, the experiment is finished and marked as a success.

Experiment Failure Reason
Target in NonTargetTray
NonTarget in TargetTray
Failed To Find Grasp Point (> 3)
Detection Confidence Too Low (< 0.45)
Too Many Isolation Steps (> 10)
Target Out of Detection Area

TABLE 4.2: Reasons for a failed experiment, including numerical values if applicable.

There are six conditions that result in a failed experiment, these are described in Table 4.2. The first two failure reasons are due to incorrect placement of an object during an aimed grasp. Placing a Target in the NonTargetTray as well as placing a NonTarget in the TargetTray has been decided to be considered a failed experiment. The three succeeding failure reasons are due to situations that are encountered during experiments that impair the model's functionality. These are implemented as a fail-safe, mainly to prevent the model from developing perpetual behaviour. Failing to find a grasp point more than three times-, as well as making more than ten isolation steps are indicators of a situation from which the model can not progress. A detection threshold of less than 0.45 results in the Mask-RCNN algorigthm functioning badly, with many false positive object detections, and therefore unwanted. The last item on the list of reasons to abort an experiment also operates as a fail-safe.

[2]`https://www.ycbbenchmarks.com/`

When the target object is outside of the detection area, it can no more be detected or grasped and therefore the experiment is finished.

Four scenarios are used for testing the performance of the model, which are distinguished by the location and distribution of the objects: Isolated, Packed, Piled, and Cluttered. Each of these will be initialized with a target object and specific number of randomly sampled objects in an orientation specific to that scenario. An example setup of each scenario can be seen in Figure 4.2. One of the objects is predetermined to be the target object, which is the main manipulation objective for that experiment.

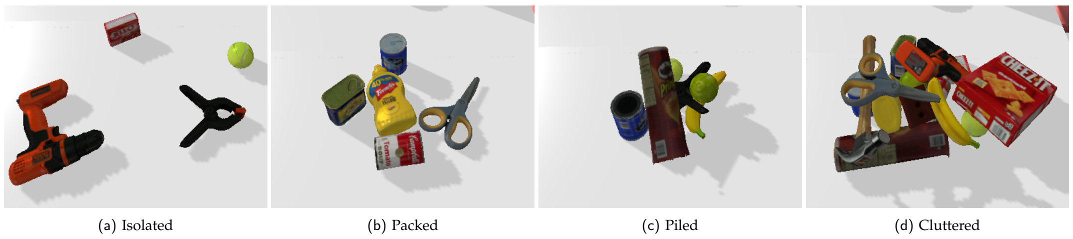

(a) Isolated (b) Packed (c) Piled (d) Cluttered

FIGURE 4.2: Example setup of each scenario.

The list of targeted objects varies per scenario. The Isolated and Packed scenarios include all 16 objects in the performed experiments. The Piled scenario does not feature experiments in which the CrackerBox is the target object. The reason is that if this object is lying face up or down, both its width and length are wider than the maximal width of the gripper. This means practically, in this orientation the CrackerBox is ungraspable and the experiment is guarenteed to fail. The random initialization of the Piled scenario can lead to this situation, which is why the CrackerBox is disregarded as a target and the Piled scenario is tested with 15 objects. With regard to the Cluttered scenario, experiments take large amounts of computation and time to perform. These experiments have been carried out with the three objects on which the model achieved the highest performance in the Piled scenario: FoamBrick, MasterChefCan, and PowerDrill. With ten experiments per object, this amounts to a total of 160 experiments for the Isolated and Packed scenario, 150 experiments for the Piled scenario, and 30 experiments for the Cluttered scenario.

4.4 Scenarios

The performance of the experiments is assessed by calculating the success rate of grasping and placing the target object in the correct tray. The success rate will determine the performance accuracy of the model with regard to target driven grasping. It is calculated by summing the total number of successful experiments and dividing by the number of experiments performed for that scenario.

$$\text{Success rate} = \frac{\text{\# of successes}}{\text{\# of experiments}}$$

Note that the experiment will only count as a success when the target object is placed inside the TargetTray. The experimental results per scenario are summarized in Table 4.3.

From Table 4.3 can be observed that the performance of the model is best in the Isolated scenario, and decreases for each the the succeeding scenarios. This is an expected result, considering the fact that the situation is more complex with each

Scenario	Number of Experiments	Success Rate
Isolated	160	94%
Packed	160	89%
Piled	150	85%
Cluttered	30	80%

TABLE 4.3: Summary of the experimental results per scenario.

subsequent scenario. The model achieves a high performance accuracy on each scenario, and an accuracy of 80% on the highly cluttered scenario.

4.4.1 Isolated

The isolated scenario is initialized with the target object, and three objects chosen randomly from the remaining set of objects, summing to a total of four. The objects are placed at predetermined coordinate locations on the workspace. The coordinates are selected with space in between each object, making their locations spread out and the objects isolated from their surroundings, thus relatively easy to grasp. An example of an isolated scenario is depicted in Figure 4.3.

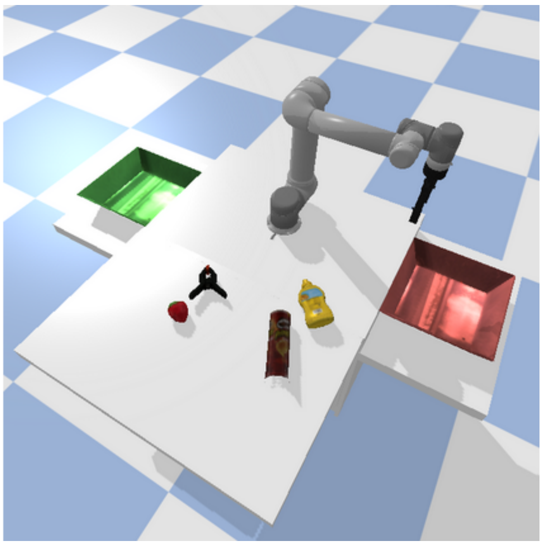


FIGURE 4.3: Example of an isolated scenario setup.

The overall accuracy the model achieves while performing the isolated scenario experiments is 94%. The performance of the model per object is described in Figure 4.4. The model performs poorly while targeting the TomatoSoupCan, with an accuracy of 50%. Figure 4.5 shows the reasons of experiment failure per object. The bad performance is caused by objects placed in the wrong tray, meaning incorrect object recognition. In the case of the CrackerBox, there is one failed experiment which is due to too many isolation steps.

4.4.2 Packed

The packed scenario is initialized with the target object, and four objects chosen randomly from the remaining set of objects, summing to a total of five. The objects are placed at predetermined coordinate locations on the workspace. After placing the objects, a procedure is started that congregates the objects into a pack. This results

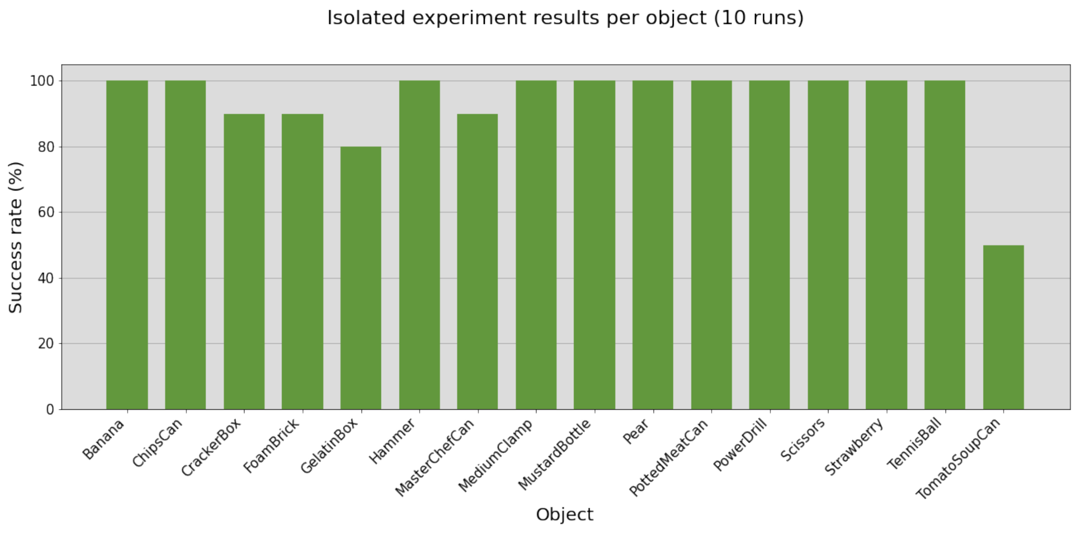


FIGURE 4.4: Results of the isolated scenario experiments.

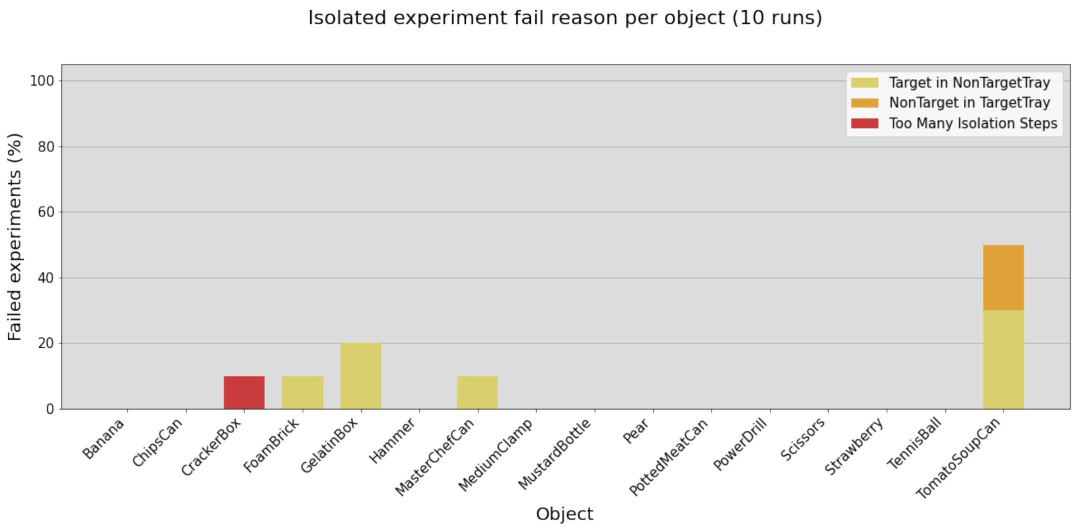


FIGURE 4.5: Failure reasons of the isolated scenario experiments.

in a scenario were object grasping and -detection are more complicated compared to the isolated scenario. An example of the packed scenario is depicted in 4.6.

The model performs the packed scenario experiments with an accuracy of 89%. The performance of the model per object is shown in Figure 4.7. The model performs poorly while targeting the TomatoSoupCan, with an accuracy of 40%. Also, the CrackerBox and GelatinBox show performance below the average of this scenario with an accuracy of 70% and 80% respectively. In Figure 4.8 the reasons for experiment failure can be seen per object. The bad performance for the TomatoSoupCan is entirely caused by placing the Target in the NonTargetTray. In the case of the CrackerBox, one of the failed experiments is caused by too many isolation steps, and another is due to the target object winding up outside the detection area.

4.4.3 Piled

Initializing the piled scenario is accomplished as follows: place the target object in the center of the workspace, position a temporary box around the target, and place

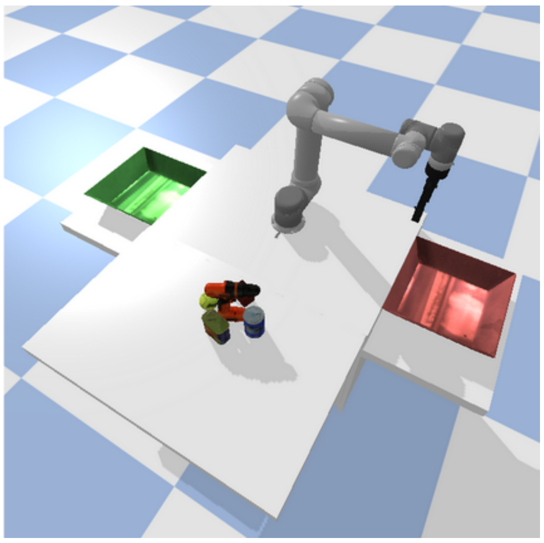


FIGURE 4.6: Example of a packed scenario setup.

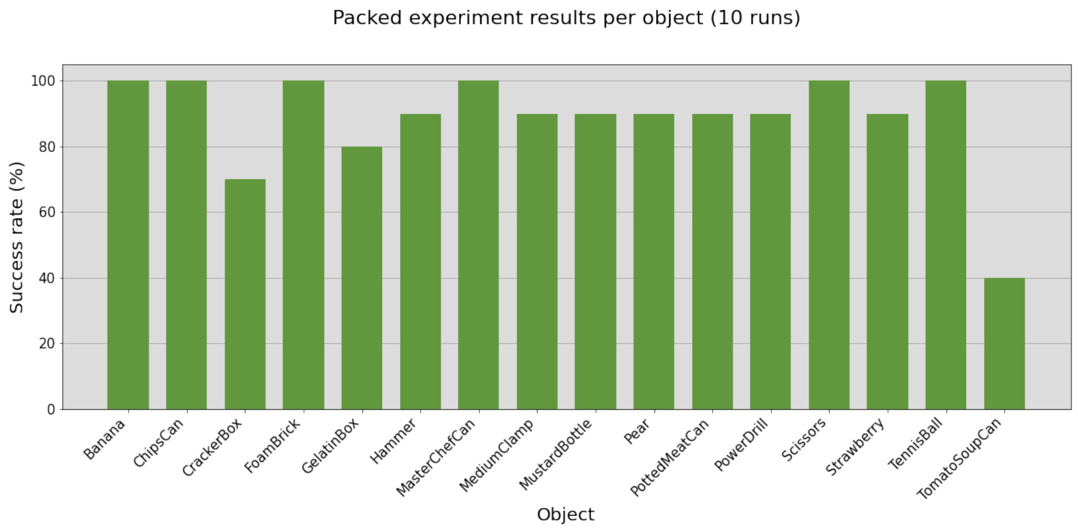


FIGURE 4.7: Results of the packed scenario experiments.

five randomly chosen objects on top of the target. This sums up to a total of six objects. The temporary box is placed to support the established pile of objects, and is removed once the objects have stopped moving. The box- and object placement is shown in Figure A.1 in Appendix A. This results in a pile of objects, with the target object located at the bottom. The target object is placed first, with the aim of increasing the probabilty of an obscured view, and creating a more complex situation. An example of a piled scenario is depicted in Figure 4.9.

The model achieves an accuracy of 85% performing the experiments in the piled scenario. Model performance is shown per object in Figure 4.10. The FoamBrick, MasterChefCan, and Powerdrill are achieving perfect accuracy in the piled scenario experiments, most of the other objects are performing between 80-90%. With the GelatinBox and Hammer as target object, the model achieves poor performance accuracy. The failure reasons of the piled scenario experiments per object can be found in Figure 4.11. For multiple targets the object culminated outside of the detection area, twice in the case of the Hammer. In the case of the GelatinBox, most of the

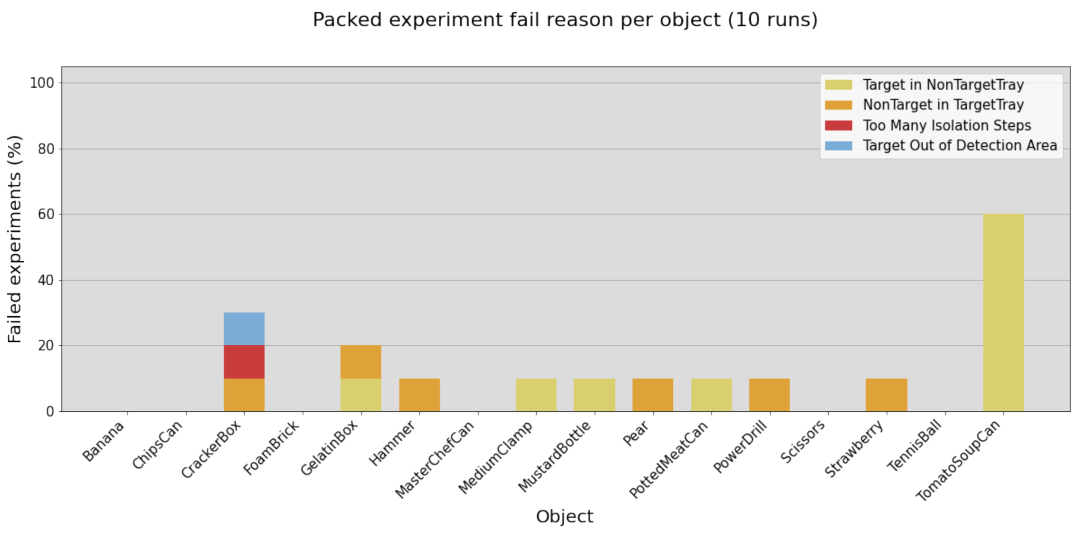


FIGURE 4.8: Failure reasons of the packed scenario experiments.

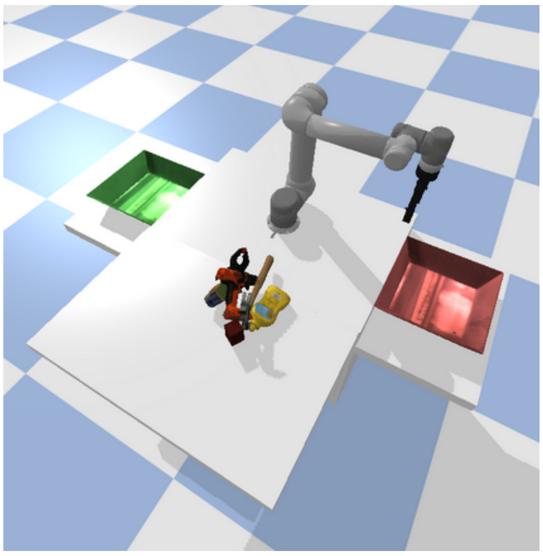


FIGURE 4.9: Example of a piled scenario setup.

experiments failures were caused by the detection confidence of the Mask-RCNN reaching the lower bound threshold.

4.4.4 Cluttered

The cluttered scenario is the most challenging scenario for the target driven grasping model. It consists of 15 objects that are piled together in a heavy clutter. The initialization is very similar to the piled scenario, the target object is placed first, a temporary box is added, and subsequently the remaining 14 objects are placed on top. Thereupon the box is removed, resulting in a large pile of objects in a heavily cluttered arrangement. An example of this scenario is depicted in Figure 4.12. Note that the cluttered scenario experiments are performed on three target objects, the objects that achieved the highest performance in the piled scenario experiments. The tested objects: FoamBrick, MasterChefCan, and PowerDrill.

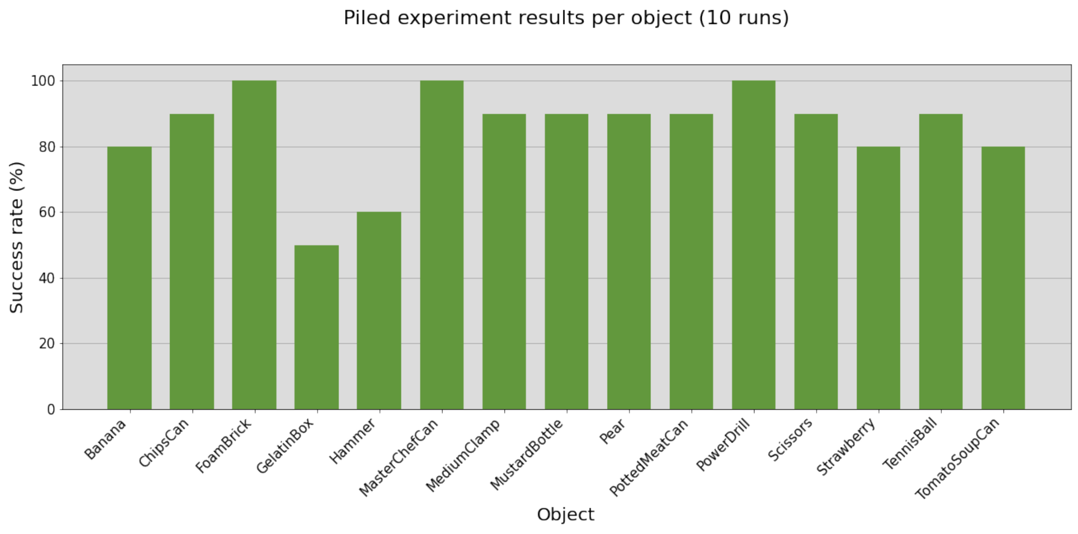


FIGURE 4.10: Results of the piled scenario experiments

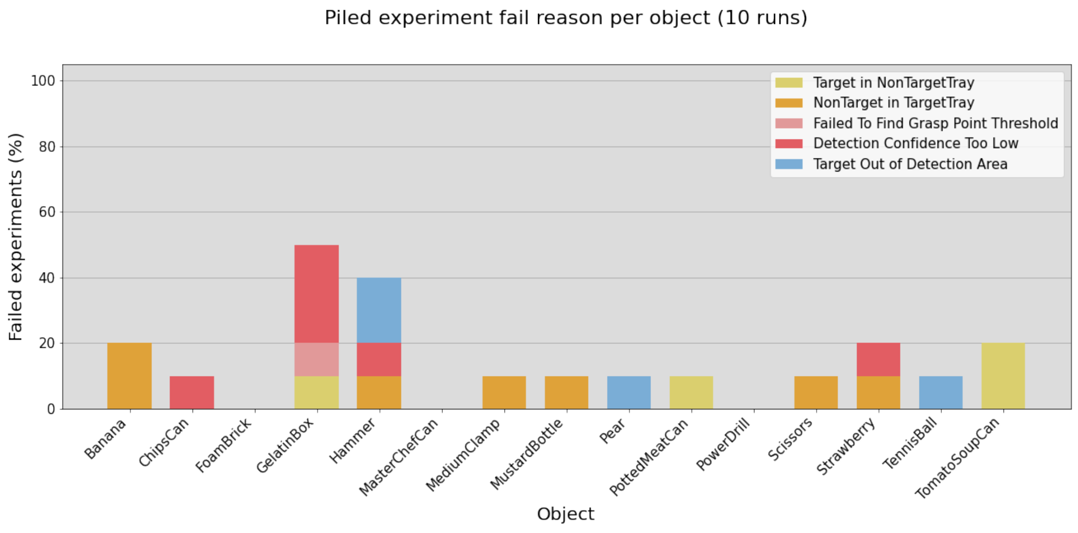


FIGURE 4.11: Failure reasons of the piled scenario experiments.

The model reaches a performance accuracy of 80% on the experiments in the cluttered scenario. This means on average, eight out of ten times it is capable of correctly singulating-, and grasping a target object from a heavily cluttered environment, and place it in the designated TargetTray. The experiment results and failure reasons per object are shown in Figure 4.13. One of the failures is caused by the FoamBrick ending up outside the detection area. The model had the most trouble with the experiments featuring the PowerDrill as a target, achieving an accuracy of 70%.

4.5 Object Visibility

In order to study the influence of target visibility on the performance accuracy of the target driven grasping model, we gathered results for both the piled and cluttered scenario. For each experiment, the visibility of the target object is calculated by means of the segmentation mask pixel value. The pixel value of the target object after scenario initialization is divided by a ground truth mask pixel value for that

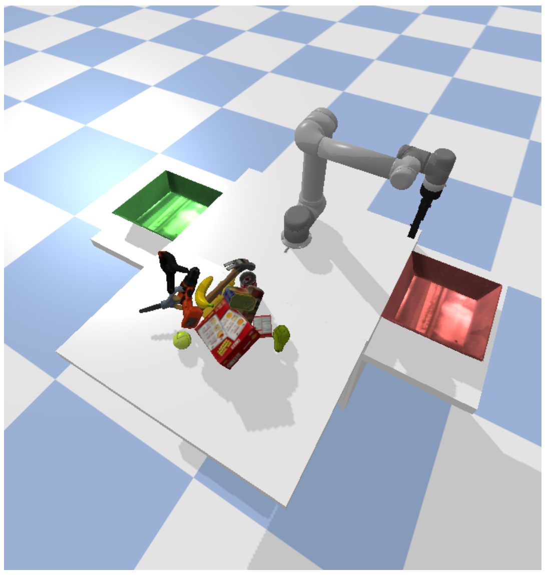


FIGURE 4.12: Example of a cluttered scenario setup.

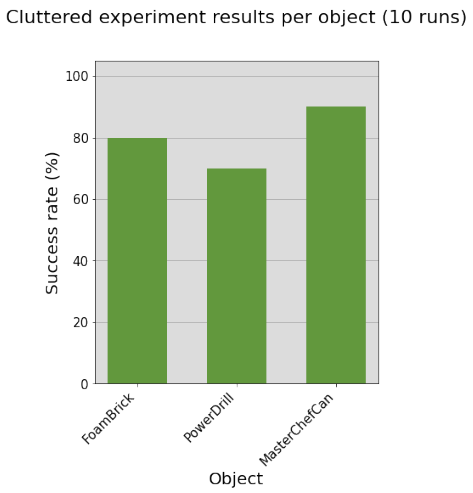
specific object. This results in a percentage that indicates object visibility after being placed in a pile or clutter of other objects.

Figure 4.14 shows the performance of the model compared to the visibility of the target object in the piled scenario experiments. In the graph can be observed that the model achieved perfect accuracy in the 0-10 and 20-30 percentile ranges, albeit with a relatively low amount of experiments. The accuracy with 10-20% visibility is zero, but only one experiment was performed in that range. Furthermore, the graph shows that most experiments have been performed with a high object visibility of 90-100%. There is no obvious trend linking higher visibility to higher accuracy. The majority of experiments have been performed with a target object visibility higher than 50%.

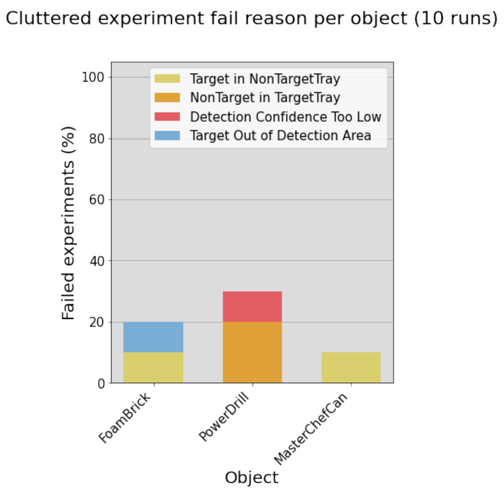
Figure 4.15 depicts the performance of the model compared to the visibility of the target object in the experiments for the cluttered scenario. Here we can see that the majority of experiments have been performed with a target object that was between 40-60% visible after initialization. The performance of the model with an object visibility of 60% and higher is perfect, with almost a third of the number of experiments performed in this range. Furthermore, with a target visibility between 0-20% the model achieves 100% accuracy as well. It is the mid range visibility experiments where the model performs worst, with the most experiments performed in this range.

4.6 Domain Randomization

The influence of domain randomization on the accuracy of the object detection module is investigated by running tests on the Mask-RCNN separately. We take model weights that have been trained on standard data, as shown in Figure 3.2a, and model weight that have been trained on texture data, as can be seen in Figure 3.3. The latter have been subject to domain randomization, where the object is provided with a texturized background.



(a) Results of the cluttered scenario experiments.



(b) Failure reasons of the cluttered scenario experiments.

FIGURE 4.13: Cluttered scenario experiment results and failure reasons.

We run the Mask-RCNN algorithm with both sets of weights, and compare the object detection accuracy of the model. The model is evaluated using all 16 objects from the dataset, with 13 randomly chosen instances for each of the objects. This adds up to a total number of 208 object detection tests. Both models utilize a confidence threshold of 0.8 for object detection.

Table 4.4 summarizes the test results for both sets of weights. It can be observed that the weights trained using Domain Randomization achieve a performance of 78%. This is 18% higher than the weights trained on standard data. The amount of wrongly predicted instances is low for both the standard-, and the DR trained weights. Most of the bad performance is due to false negative in the form of empty predictions, meaning undetected objects.

Data	**Number of Instances**	**Wrongly Predicted**	**Empty Prediction**	**Performance**
Standard	208	4	77	61%
With Domain Randomization	208	6	40	78%

TABLE 4.4: Results of tests performed on Mask-RCNN weights trained with- and without Domain Randomization.

4.7 Active Segmentation

In order to quantify the influence of the active segmentation module, we investigate the number of segmentation moves. In this comparison, the isolated scenario experiments are disregarded, since segmentation moves do not add any value in this environment. For each of the remaining three scenarios, the number of successful experiments is compared to the total number of segmentation moves. Dividing the latter by the number of successful experiments results in a metric that gives the average number of segmentation moves per successful experiment. Table 4.5 shows these metrics for each scenario. Note that the total number of experiments performed is much lower for the cluttered scenario. The packed and piled scenario experiments show comparable results, with an average of around 0.3 segmentation

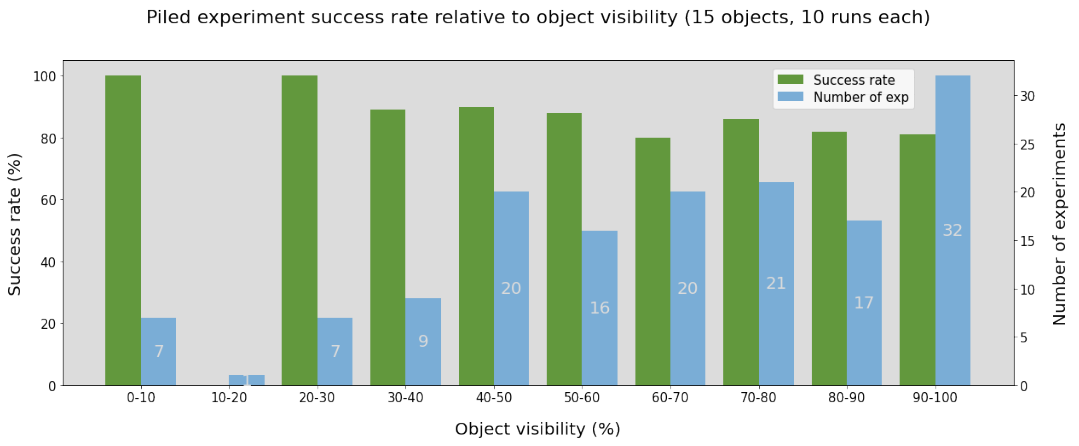


FIGURE 4.14: Piled scenario experiment success rate compared to target object visibility percentage. The green bars indicate the performance of the model, the blue bars report the number of experiments performed in that percentile range. Total number of performed experiments: 150.

steps per successful experiment. The cluttered scenario however, shows an average of 2.6 segmentation moves per successfully performed experiment, which is more than 8 times as high.

Scenario	Total Experiments	Successful Experiments	Total Segmentation Moves	Average per Success
Packed	160	142	38	0.3
Piled	150	128	55	0.4
Cluttered	30	24	60	2.5

TABLE 4.5: For each applicable scenario, an analysis of the influence of segmentation moves on the success rate of an experiment.

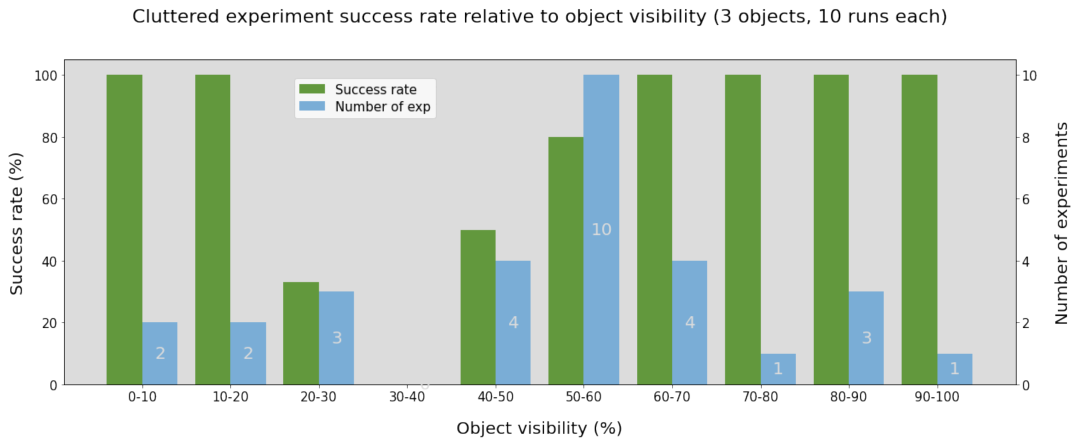


FIGURE 4.15: Cluttered scenario experiment success rate compared to target object visibility percentage. The green bars indicate the performance of the model, the blue bars report the number of experiments performed in that percentile range. Total number of performed experiments: 30.

Chapter 5

Discussion

The approach taken in this research shows adequate performance in target driven object grasping. In isolated scenarios the accuracy is very high, as expected since this environment poses few challenges. The performance of the model decreases for each subsequent scenario which is also expected, because with each scenario the complexity of the situation increases. With an accuracy of 80% on the highly cluttered scenario we have implemented a model that is capable of target driven object grasping in highly cluttered environments. The performance of the model could have been better, several factors are influencing the functionality of our approach.

First of these is the presence of the CrackerBox. During the testing phase it was noticed that the CrackerBox, when facing up- or downwards, is ungraspable by the robotic arm, due to the limit of the gripper width. We suspect that in the failed isolated and packed experiments caused by too many isolation steps, the CrackerBox has fallen down and is ungraspable. This might have happened with the robot arm passing by, and is backed up by the high number of failed grasps that have been noted for these experiments.

A design choice with regard to the CrackerBox was made, that in hindsight could have been done differently. The object is disregarded as a target object for the piled and cluttered experiments, but is still spawned as one of the secondary objects, if chosen. Considering the model might be unable to grasp the CrackerBox, the decision was made to completely ignore its existence when detected by the Mask-RCNN. This can lead to troublesome situations, for example when the CrackerBox is blocking the view of the target object. It would have been better to completely remove the CrackerBox from the dataset, retrain the Mask-RCNN, and run the model with the remaining objects. Another approach could include pushing the CrackerBox instead of trying to grasp it, when suspected that it is blocking the target object.

Another property influencing experimental results is the fact that objects fall from the table, and culminate outside of the detection area of the model. This can occur due to misfortune during scenario initialization, in the form of an object bouncing away after hitting another object. It can also occur during the segmentation step, when the arm pushes the object off the table. This happened at least once for a target object in three out of four scenarios, and influences the resulting performance accuracy. It might be solved relatively easy by adjusting the simulated table setup, implementing borders on the edges of the table aimed at keeping the objects within the detection area.

The biggest improvement that can be realized in order to decrease the number of failed experiments would be a procedure that is aimed at failure recovery. Currently, when a Target object is placed in the NonTargetTray, or when a NonTarget object is placed in the TargetTray, the experiment is terminated and finished as a failure. Summing up all experiments in the four scenarios, these account for more than 70% of the failed experiments. A solution could be implementing a phase that is aimed at verification of an object that is placed in either of the trays. If the model concludes the placed object is incorrect, responsive action can be taken, re-entering the grasping phase in order to remove and relocate the object.

The performance of the Mask-RCNN is something that greatly impacts the model, since its functionality is one of the fundamental parts. Although the implementation of Domain Randomization in the training phase improved performance on object detection, the network could be trained better, resulting in higher accuracy. The training stage of the Mask-RCNN transpired troublesome, mostly caused by the requirement of outdated versions of software packages. The training stage was interrupted due to the limited amount of time and computation memory provided by Google. Attempting to get the algorithm running on the Peregrine computercluster, similar results were not achieved. Training on a personal computer was infeasable to due to the long training times. In the graph A.2 can be observed that the validation error is still decreasing, which means a better version of the Mask-RCNN can be realized. The accuracy of the Mask-RCNN is central to the performance of the model in many ways, therefore improvements will positively affect the approach on multiple levels. Accurate object detection is essential for correctly identifying both target and non target objects, and accurate bounding box and object mask generation plays a major role in the functionality of the grasping and segmentation module.

Considering the implementation of the segmentation step in this research is relatively uncomplicated, improvements are be achievable. Although the results show that segmenation moves play a role in successful experiments, its mechanics are now very dependent on the output of the Mask-RCNN. If the bounding box transferred to the segmentation module is inaccurate, the resulting motion might be undesired. Increasing the amount-, or using different metrics for coordinating the active object segmentation would make the approach more robust. Related research has shown that intelligently coordinated pushing behaviours can lead to models achieving good performance at grasping in cluttered environments (Kiatos et al., 2022; Tang et al., 2021).

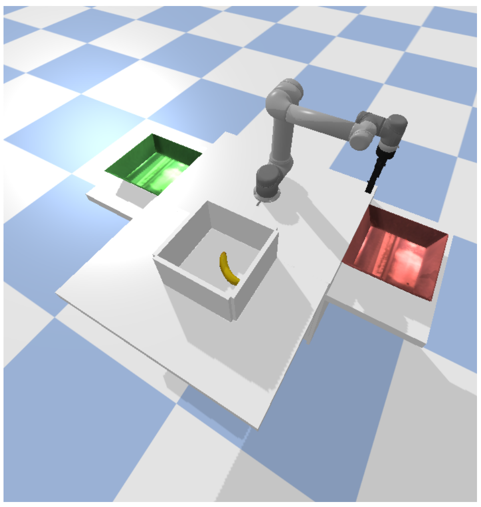
The data gathered with regard to target object visibility could be increased. Although the model shows good accuracy scores with low target visibility, most of the data resides in the middle and high ranked percentiles. Having more data in the lower percentiles would enable stronger conclusions about the systems performance in highly complex scenarios.

5.1 Conclusion

In this research we proposed an approach using deep learning methods for target driven object grasping in highly cluttered scenarios. By combining the functionality of an object detection network Mask-RCNN, and a network aimed at grasping GR-ConvNet, a model is realized. The approach allows a robot to interact with the environments in four different scenarios, isolated, packed, piled, and highly cluttered. By the implementation of a segmentation step, the model is capable of applying active segmentation. This improves the functionality of the method in scenarios where object detection and -grasping are impaired, by being able to singulate a specific object with robotic locomotion. A form of domain randomization has been applied to the training data of the Mask-RCNN, improving its performance on detecting unseen objects. Extensive sets of experiments have been performed in four everyday scenarios: isolated, packed objects, pile of objects, and highly cluttered scenes. Experimental results showed that the proposed method worked very well in all four scenarios. In the continuation of this work, it would be interesting to test the approach in real world scenarios, possibly in combination with the domain randomization approach taken in Tobin et al., 2017 to improve the transfer to a real world robot. Furthermore, the system can be testing more vigorously, focusing on different datasets of objects, or gathering results on target objects with low visibility. An interesting property this research possesses, is the fact that both the segmentation-, as well as the object detection-, and the grasping module are easily interchangeable with equivalent counterparts, making it a highly versatile approach. Using different networks for object detection and grasping could lead to interesting research. For instance implementing VGN (Breyer et al., 2021) for grasp synthesis generation in six DoF. Object detection might be improved by using both RGB as Depth images as input for the Mask-RCNN, and implementing the improved version of the algorithm proposed by Zhang et al., 2021. Implementing a different segmentation module is an option, using learning algorithms leading to different moves for specific objects can improve its functionality. This might solve the problem of pushing objects of the workspace.

Appendix A

Appendix


(a) Placement of temporary box around target object.

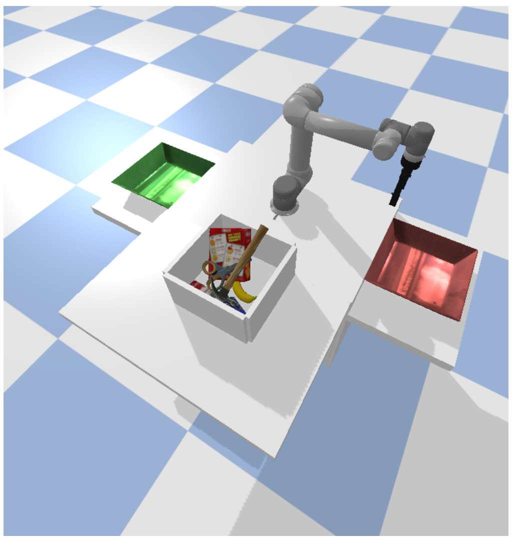

(b) Placement of five objects on top, to create pile.

FIGURE A.1: Cluttered scenario experiment results and failure reasons.

A.1 Mask-RCNN Training Parameter Configuration

```
Configurations:
BACKBONE                       resnet101
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     4
BBOX_STD_DEV                   [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE         None
DETECTION_MAX_INSTANCES        100
DETECTION_MIN_CONFIDENCE       0.9
DETECTION_NMS_THRESHOLD        0.3
FPN_CLASSIF_FC_LAYERS_SIZE     1024
GPU_COUNT                      1
GRADIENT_CLIP_NORM             5.0
IMAGES_PER_GPU                 4
IMAGE_CHANNEL_COUNT            3
IMAGE_MAX_DIM                  448
```

```
IMAGE_META_SIZE               29
IMAGE_MIN_DIM                 448
IMAGE_MIN_SCALE               0
IMAGE_RESIZE_MODE             square
IMAGE_SHAPE                   [448 448   3]
LEARNING_MOMENTUM             0.9
LEARNING_RATE                 0.001
LOSS_WEIGHTS                  {'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0,
'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0, 'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE                14
MASK_SHAPE                    [28, 28]
MAX_GT_INSTANCES              1
MEAN_PIXEL                    [123.7 116.8 103.9]
MINI_MASK_SHAPE               (56, 56)
NAME                          object
NUM_CLASSES                   17
POOL_SIZE                     7
POST_NMS_ROIS_INFERENCE       1000
POST_NMS_ROIS_TRAINING        2000
PRE_NMS_LIMIT                 6000
ROI_POSITIVE_RATIO            0.33
RPN_ANCHOR_RATIOS             [0.5, 1, 2]
RPN_ANCHOR_SCALES             (32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE             1
RPN_BBOX_STD_DEV              [0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD             0.7
RPN_TRAIN_ANCHORS_PER_IMAGE   256
STEPS_PER_EPOCH               1600
TOP_DOWN_PYRAMID_SIZE         256
TRAIN_BN                      False
TRAIN_ROIS_PER_IMAGE          200
USE_MINI_MASK                 True
USE_RPN_ROIS                  True
VALIDATION_STEPS              100
WEIGHT_DECAY                  0.0001
```

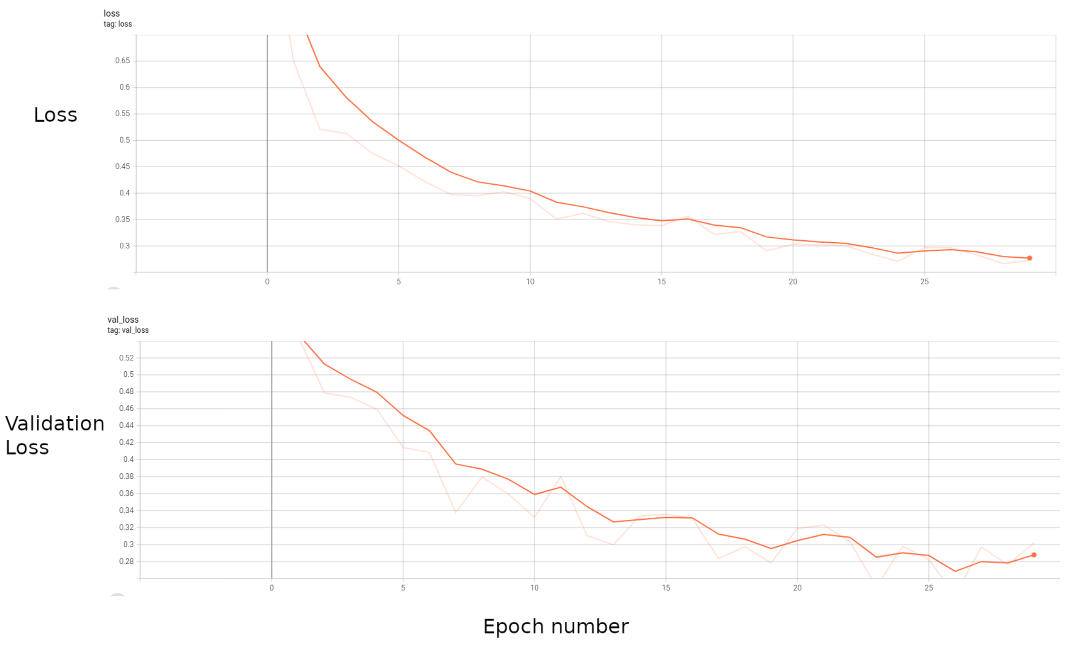


FIGURE A.2: Graphs of training stage Mask-RCNN, depicting loss and validation loss.

Bibliography

Abdulla, Waleed (2017). *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*. https://github.com/matterport/Mask_RCNN.

Bai, Qiang et al. (2020). "Object Detection Recognition and Robot Grasping Based on Machine Learning: A Survey". In: *IEEE Access* 8, 181855–181879. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3028740.

Bharati, Puja and Ankita Pramanik (2020). "Deep Learning Techniques—R-CNN to Mask R-CNN: A Survey". en. In: *Computational Intelligence in Pattern Recognition*. Ed. by Asit Kumar Das et al. Vol. 999. Advances in Intelligent Systems and Computing. Singapore: Springer Singapore, 657–668. ISBN: 9789811390418. DOI: 10.1007/978-981-13-9042-5_56. URL: http://link.springer.com/10.1007/978-981-13-9042-5_56.

Bowen, John and Cristian Morosan (2018). "Beware hospitality industry: the robots are coming". In: *Worldwide Hospitality and Tourism Themes* 10, pp. 726–733. DOI: 10.1108/WHATT-07-2018-0045.

Breyer, Michel et al. (2021). "Volumetric Grasping Network: Real-time 6 DOF Grasp Detection in Clutter". en. In: *arXiv:2101.01132 [cs]*. arXiv: 2101.01132. URL: http://arxiv.org/abs/2101.01132.

Buric, Matija, Miran Pobar, and Marina Ivasic-Kos (2018). "Ball Detection Using Yolo and Mask R-CNN". en. In: *2018 International Conference on Computational Science and Computational Intelligence (CSCI)*. Las Vegas, NV, USA: IEEE, 319–323. ISBN: 978-1-72811-360-9. DOI: 10.1109/CSCI46756.2018.00068. URL: https://ieeexplore.ieee.org/document/8947818/.

Calli, Berk et al. (2017). "Yale-CMU-Berkeley dataset for robotic manipulation research". en. In: *The International Journal of Robotics Research* 36.3, 261–268. ISSN: 0278-3649, 1741-3176. DOI: 10.1177/0278364917700714.

Coumans, Erwin and Yunfei Bai (2016–2021). *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. http://pybullet.org.

Dehban, Atabak et al. (2019). "The Impact of Domain Randomization on Object Detection: A Case Study on Parametric Shapes and Synthetic Textures". en. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Macau, China: IEEE, 2593–2600. ISBN: 978-1-72814-004-9. DOI: 10.1109/IROS40897.2019.8968139. URL: https://ieeexplore.ieee.org/document/8968139/.

Dorrer, M G and A E Tolmacheva (2020). "Comparison of the YOLOv3 and Mask R-CNN architectures' efficiency in the smart refrigerator's computer vision". en. In: *Journal of Physics: Conference Series* 1679.4, p. 042022. ISSN: 1742-6588, 1742-6596. DOI: 10.1088/1742-6596/1679/4/042022.

He, Kaiming et al. (2016). "Deep Residual Learning for Image Recognition". en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, 770–778. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.90. URL: http://ieeexplore.ieee.org/document/7780459/.

He, Kaiming et al. (2017). "Mask R-CNN". In: *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969. DOI: 10.1109/ICCV.2017.322.

Hu, Long et al. (2019). “iRobot-Factory: An Intelligent Robot Factory Based on Cognitive Manufacturing and Edge Computing”. en. In: *Future Generation Computer Systems* 90, 569–577. ISSN: 0167739X. DOI: 10.1016/j.future.2018.08.006.

Jo, HyunJun and Jae-Bok Song (2020). “Object-Independent Grasping in Heavy Clutter”. en. In: *Applied Sciences* 10.3, p. 804. ISSN: 2076-3417. DOI: 10.3390/app10030804.

Jung, Alexander B. et al. (2020). *imgaug*. https://github.com/aleju/imgaug. Online; accessed 01-Feb-2020.

Kasaei, Hamidreza and Mohammadreza Kasaei (2022). “MVGrasp: Real-Time Multi-View 3D Object Grasping in Highly Cluttered Environments”. en. In: URL: http://arxiv.org/abs/2103.10997.

Kiatos, Marios et al. (2022). “Learning Push-Grasping in Dense Clutter”. en. In: *IEEE Robotics and Automation Letters*, 1–8. ISSN: 2377-3766, 2377-3774. DOI: 10.1109/LRA.2022.3188437.

Kumra, Sulabh, Shirin Joshi, and Ferat Sahin (2020). “Antipodal Robotic Grasping using Generative Residual Convolutional Neural Network”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 9626–9633. DOI: 10.1109/IROS45743.2020.9340777.

Kurenkov, Andrey et al. (2020). “Visuomotor Mechanical Search: Learning to Retrieve Target Objects in Clutter”. en. In: arXiv:2008.06073. arXiv:2008.06073 [cs]. URL: http://arxiv.org/abs/2008.06073.

Lin, Tsung-Yi et al. (2017). “Feature Pyramid Networks for Object Detection”. en. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI: IEEE, 936–944. ISBN: 978-1-5386-0457-1. DOI: 10.1109/CVPR.2017.106. URL: http://ieeexplore.ieee.org/document/8099589/.

Liu, Wei et al. (2016). “SSD: Single Shot MultiBox Detector”. en. In: *Computer Vision – ECCV 2016*. Ed. by Bastian Leibe et al. Vol. 9905. Lecture Notes in Computer Science. Cham: Springer International Publishing, 21–37. ISBN: 978-3-319-46447-3. DOI: 10.1007/978-3-319-46448-0_2. URL: http://link.springer.com/10.1007/978-3-319-46448-0_2.

Mao, Xin et al. (2021). “Domain randomization-enhanced deep learning models for bird detection”. en. In: *Scientific Reports* 11.1, p. 639. ISSN: 2045-2322. DOI: 10.1038/s41598-020-80101-x.

Marwan, Qaid Mohammed, Shing Chyi Chua, and Lee Chung Kwek (2021). “Comprehensive Review on Reaching and Grasping of Objects in Robotics”. en. In: *Robotica* 39.10, 1849–1882. ISSN: 0263-5747, 1469-8668. DOI: 10.1017/S0263574721000023.

Mehta, Bhairav et al. (2020). “Active Domain Randomization”. In: *Proceedings of the Conference on Robot Learning*. Ed. by Leslie Pack Kaelbling, Danica Kragic, and Komei Sugiura. Vol. 100. Proceedings of Machine Learning Research. PMLR, pp. 1162–1176. URL: https://proceedings.mlr.press/v100/mehta20a.html.

Mohammed, Marwan Qaid, Kwek Lee Chung, and Chua Shing Chyi (2020). “Review of Deep Reinforcement Learning-Based Object Grasping: Techniques, Open Challenges, and Recommendations”. en. In: *IEEE Access* 8, 178450–178481. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.3027923.

Morrison, Douglas, Peter Corke, and Jürgen Leitner (2018). “Closing the Loop for Robotic Grasping: A Real-time, Generative Grasp Synthesis Approach”. en. In: *arXiv:1804.05172 [cs]*. arXiv: 1804.05172. URL: http://arxiv.org/abs/1804.05172.

Redmon, Joseph and Ali Farhadi (2018). “YOLOv3: An Incremental Improvement”. en. In: arXiv:1804.02767. arXiv:1804.02767 [cs]. URL: http://arxiv.org/abs/1804.02767.

Redmon, Joseph et al. (2016). “You Only Look Once: Unified, Real-Time Object Detection”. en. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, USA: IEEE, 779–788. ISBN: 978-1-4673-8851-1. DOI: 10.1109/CVPR.2016.91. URL: http://ieeexplore.ieee.org/document/7780460/.

Ren, Shaoqing et al. (2017). “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks”. en. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6, 1137–1149. ISSN: 0162-8828, 2160-9292. DOI: 10.1109/TPAMI.2016.2577031.

Robinson, Hayley, Bruce MacDonald, and Elizabeth Broadbent (2014). “The Role of Healthcare Robots for Older People at Home: A Review”. en. In: *International Journal of Social Robotics* 6.4, 575–591. ISSN: 1875-4791, 1875-4805. DOI: 10.1007/s12369-014-0242-2.

Santhakumar, Krishnakumar and Hamidreza Kasaei (2022). “Lifelong 3D object recognition and grasp synthesis using dual memory recurrent self-organization networks”. en. In: *Neural Networks* 150, 167–180. ISSN: 08936080. DOI: 10.1016/j.neunet.2022.02.027.

Schneider, Fred B (1990). “Implementing Fault-Tolerant Services Using the State Machine Approach: a Tutorial”. In: *ACM Computing Surveys (CSUR)* 22.4, pp. 299–319.

Shorten, Connor and Taghi M. Khoshgoftaar (2019). “A survey on Image Data Augmentation for Deep Learning”. en. In: *Journal of Big Data* 6.1, p. 60. ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0.

Tang, Bingjie et al. (2021). “Learning Collaborative Pushing and Grasping Policies in Dense Clutter”. en. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. Xi’an, China: IEEE, 6177–6184. ISBN: 978-1-72819-077-8. DOI: 10.1109/ICRA48506.2021.9561828. URL: https://ieeexplore.ieee.org/document/9561828/.

Tobin, Josh et al. (2017). “Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World”. en. In: arXiv:1703.06907. arXiv:1703.06907 [cs]. URL: http://arxiv.org/abs/1703.06907.

Tobin, Joshua et al. (2018). “Domain Randomization and Generative Models for Robotic Grasping”. en. In: arXiv:1710.06425. arXiv:1710.06425 [cs]. URL: http://arxiv.org/abs/1710.06425.

Vrielink, Jeroen Oude and Dr Hamidreza Kasaei (2021). “Learning to grasp objects in highly cluttered environments using Deep Convolutional Neural Networks”. en. In: p. 13.

Wang, Chao et al. (2020). “Feature Sensing and Robotic Grasping of Objects with Uncertain Information: A Review”. en. In: *Sensors* 20.1313, p. 3707. DOI: 10.3390/s20133707.

Wang, Tian-Miao, Yong Tao, and Hui Liu (2018). “Current Researches and Future Development Trend of Intelligent Robot: A Review”. en. In: *International Journal of Automation and Computing* 15.5, 525–546. ISSN: 1476-8186, 1751-8520. DOI: 10.1007/s11633-018-1115-1.

Wilson, Garrett et al. (2019). “Robot-enabled support of daily activities in smart home environments”. en. In: *Cognitive Systems Research* 54, 258–272. ISSN: 13890417. DOI: 10.1016/j.cogsys.2018.10.032.

Xue, Hongyang, Shengming Zhang, and Deng Cai (2017). “Depth Image Inpainting: Improving Low Rank Matrix Completion with Low Gradient Regularization”. en. In: *IEEE Transactions on Image Processing* 26.9. arXiv:1604.05817 [cs], 4311–4320. ISSN: 1057-7149, 1941-0042. DOI: 10.1109/TIP.2017.2718183.

Yang, Yang, Hengyue Liang, and Changhyun Choi (2020). "A Deep Learning Approach to Grasping the Invisible". en. In: arXiv:1909.04840. arXiv:1909.04840 [cs]. URL: http://arxiv.org/abs/1909.04840.

Yu, Yang et al. (2019). "Fruit detection for strawberry harvesting robot in non-structural environment based on Mask-RCNN". en. In: *Computers and Electronics in Agriculture* 163, p. 104846. ISSN: 01681699. DOI: 10.1016/j.compag.2019.06.001.

Zeng, Andy et al. (2018). "Learning Synergies Between Pushing and Grasping with Self-Supervised Deep Reinforcement Learning". en. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Madrid: IEEE, 4238–4245. ISBN: 978-1-5386-8094-0. DOI: 10.1109/IROS.2018.8593986. URL: https://ieeexplore.ieee.org/document/8593986/.

Zhai, Sheping et al. (2020). "DF-SSD: An Improved SSD Object Detection Algorithm Based on DenseNet and Feature Fusion". In: *IEEE Access* 8, 24344–24357. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2020.2971026.

Zhang, Jicun et al. (2021). "X-Ray Image Recognition Based on Improved Mask R-CNN Algorithm". en. In: *Mathematical Problems in Engineering* 2021. Ed. by Shanglei Jiang, 1–14. ISSN: 1563-5147, 1024-123X. DOI: 10.1155/2021/6544325.