Reasoning for Improved Capacity in Robotic Pick-and-Place Tasks

Karin van Schooten

Master of Science Thesis



TU Delft

VANDERLANDE

Reasoning for Improved Capacity in Robotic Pick-and-Place Tasks

by

Karin van Schooten

to obtain the degree of Master of Science
at the Delft University of Technology,
Faculty Mechanical, Maritime and Materials Engineering (3mE),
to be defended publicly on Friday March 20, 2020 at 1:00 PM.

Student number:	4369289	
Project duration:	June 17, 2019 – March 20, 2020	
Thesis committee:	Dr. C. Hernandez Corbato	TU Delft, supervisor
	Dr. ing. J. Kober	TU Delft, chair
	Dr. D. Dodou	TU Delft
	M. Imre	TU Delft

This thesis is confidential and cannot be made public until March 20, 2025.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

As e-commerce continues to grow, warehouses face the challenge of keeping up with product flows. Vanderlande provides a pick-and-place robot (SIR) that helps in improving capacity. However, SIR's capacity is limited by its vacuum gripper and its inability to solve failures. The aim of this thesis is to improve SIR by adding a second gripper and reasoning capabilities. To this end, a grasp synthesis approach is developed that complements the vacuum gripper, based on existing approaches in literature and the devices available to SIR. Furthermore, an ontology is developed according to Methontology that allows the robot to reason about item properties and failures, while providing operators with insight into its thought process. The improved system is referred to as RSIR.

The grasp synthesis approach, applicable to parallel-jaw grippers, synthesizes grasps by detecting parallel edges in depth data. Tests on real depth data showed that the approach synthesized reasonable grasps in reasonable time for a variety of items, some of which are not graspable by SIR's vacuum gripper. It did not need a segmentation of the items to do so. Furthermore, RSIR was tested on a virtual robotic set-up. The results showed that RSIR reasoned how best to complete a task when provided with information about a robot's components and possible item properties. RSIR also drew conclusions based on the execution of the task, and reasoned what actions should be performed to prevent failures from re-occurring.

The above results show RSIR's potential for autonomously estimating item properties and selecting the best approach for grasping an item. In practice, this decreases downtime and thus improves capacity. Because RSIR is designed to be expandable to related tasks, its benefits could be useful in more of Vanderlande's applications than just the SIR system.

Contents

1 Introduction 1

2 Specifications of the Pick-and-Place Robot 5
2.1 Components . . . 5
2.2 Operational Pipeline . . . 7

3 Requirements for the Reasoner and Grasping Approach 9
3.1 Functional Requirements . . . 9
3.2 Non-functional Requirements . . . 10
3.3 Requirements for the Grasping Approach . . . 10

4 System Design and Verification 13
4.1 Knowledge Requirements . . . 13
4.1.1 Justification for Ontologies . . . 13
4.1.2 Ontology Development Method . . . 14
4.1.3 Ontology Requirements: Competency Questions . . . 14
4.2 Achieving Robustness . . . 16
4.2.1 Human Interaction . . . 16
4.2.2 Adaptive Behaviour . . . 16
4.3 Evaluation Method . . . 17
4.3.1 Materials . . . 17
4.3.2 Grasping Approach . . . 18
4.3.3 Ontology and Reasoning . . . 18

5 Grasping Approach 19
5.1 Vacuum Performance . . . 19
5.2 Requirements . . . 21
5.3 Gripper Technology . . . 22
5.3.1 Available Technologies . . . 22
5.3.2 Selecting the Best Technology . . . 27
5.4 Synthesis Approach . . . 28
5.4.1 Available Approaches . . . 28
5.4.2 Summary of Approaches . . . 34
5.4.3 Selecting the Best Approach . . . 37
5.4.4 Adjustments . . . 38

6 Ontology and Reasoning 41
6.1 Required Knowledge . . . 41
6.1.1 Failures . . . 41
6.1.2 Tasks . . . 44
6.1.3 Items . . . 48
6.2 Existing Ontologies . . . 50
6.2.1 Mid-level Ontology . . . 50
6.2.2 Low-level Ontologies . . . 50
6.2.3 Selection of Ontologies . . . 52
6.3 Conceptualization . . . 53
6.3.1 Main Idea behind the Ontology . . . 53
6.3.2 Reasoning about Action Plans . . . 55
6.3.3 Reasoning about Item Properties . . . 57
6.3.4 Reasoning about Failures . . . 59

6.4 Resulting Ontology and Reasoner . . . 63
6.4.1 Ontology . . . 63
6.4.2 Reasoner . . . 67

7 Implementation 69
7.1 Software Architecture . . . 69
7.2 State Machine . . . 70
7.3 Ontology . . . 71
7.4 Reasoner . . . 73
7.5 Grasp Synthesis . . . 73

8 Grasping Approach Evaluation 75
8.1 Tuning Parameters . . . 76
8.2 General Performance . . . 77
8.2.1 Method . . . 77
8.2.2 Results . . . 78
8.2.3 Discussion . . . 80
8.2.4 Conclusion . . . 84
8.3 Performance for Segmentations . . . 85
8.3.1 Method . . . 85
8.3.2 Results . . . 85
8.3.3 Discussion . . . 89
8.3.4 Conclusion . . . 90

9 Ontology and Reasoner Evaluation 93
9.1 Competency Questions . . . 93
9.1.1 Answers to Competency Questions . . . 93
9.1.2 Limitations . . . 95
9.2 Ontology and Reasoning . . . 95
9.2.1 Exploitation . . . 97
9.2.2 Prior Knowledge . . . 99
9.2.3 Exploration . . . 100
9.2.4 Robustness . . . 101
9.3 Demonstration . . . 103

10 Conclusion 105

11 Future Work 107

Appendices 109

A GS1 Product Categories 111

B Implementations of Related Tasks 113

C Details on the Resulting Ontology 123
C.1 Introduction to Ontology Terminology . . . 123
C.2 Definitions of Concepts in the Resulting Ontology . . . 124
C.3 Example Ontology for Robot Kevin . . . 126

D Details on the Grasp Synthesis Algorithm 133
D.1 Implementation of the Algorithm . . . 133
D.2 Grasp Synthesis Algorithm Parameters . . . 138

E Grasp Synthesis Results 149
E.1 Grasps for Box Segmentations . . . 149
E.2 Segmentation Masks . . . 171
E.3 Grasps for Partial Segmentations . . . 176
E.4 Grasps for Complete Segmentations . . . 198

F Ontology and Reasoner Results 205
F.1 Queries for Answering Competency Questions 205
F.2 Ontology Used for Evaluation 210
F.3 Estimated Item Properties using Exploitation 214
F.4 Estimated Item Properties using Prior Knowledge 221
F.5 Estimated Item Properties using Exploration 225
F.5.1 Exploring 1 in 5 cycles 225
F.5.2 Exploring 1 in 2 cycles 228
F.6 Estimated Item Properties for Robustness. 231
F.6.1 Exploring 1 in 5 cycles 231
F.6.2 Exploring 1 in 2 cycles 234

Bibliography 239

Acknowledgements

First and foremost, I would like to show my gratitude to my supervisor Stijn de Looijer for his feedback and other valuable input, as well as the freedom he gave me during this thesis. My thanks go out to Carlos Hernandez Corbato for his feedback, his help in navigating the university's organisational structure, and his efforts in organizing meetings with graduate students. These meetings gave me insight into knowledge and reasoning in robotics outside the scope of this thesis. I would also like to thank Hugo Taams for providing me with everything I needed to know about FM-GtP, and Huub Sanders for his help in gathering data on the FM-GtP set-up. Furthermore, I would like to thank all of the Robotics and Algorithms teams at Vanderlande for their advice, their interest, and their contribution to my becoming a better table football player.

This work would not have been possible without KnowRob. I wish to show my gratitude to the researchers behind KnowRob for their research and for making their framework available. Finally, my thanks go out to the researchers behind Protégé for making their tool available and providing excellent documentation for it.

Karin van Schooten
Veghel, March 2020

1 Introduction

As e-commerce continues to grow, warehouses must keep pace with the large number of items ordered and the customers' growing demand for service, including fast delivery [46][142]. Because of this, the warehousing sector has a need for fast and high capacity order fulfilment with near 24/7 operation. With e-commerce growing at a staggering rate - approximately an 18% increase in 2019 - this need is larger than ever [142].

Vanderlande provides for this need with the Functional Module: Goods to Picker (FM-GtP). Its goal is to provide an efficient and high capacity picking solution. It does this by picking items from an input carrier and placing them in an output carrier. The input carrier is part of the storage system, whereas the output carrier is destined to leave the system, for example to a customer. FM-GtP is intended to be combined with other Vanderlande modules, such that the whole warehouse process can be automated. Multiple FM-GtPs can then be used depending on the required picking capacity. Furthermore, an FM-GtP can be manual or automated. The goal of the automated variant is to reduce the labour cost and the dependency on labour. An FM-GtP should have as high a capacity as possible to reduce number of Fm-GtPs required in the application. Additionally, the fill rate of the output carrier should be as high as possible (that is, fitting as many items into it as possible), as this reduces transportation costs in some applications [24].

To achieve as high a capacity as possible on an automated FM-GtP, it is important for the system to handle as many different items as possible. This is because every time FM-GtP encounters an item it cannot handle, an operator has to walk over to the robot and finish the picking manually. This costs a relatively large amount of time and therefore greatly reduces the system's capacity [24]. The subsystem of Fm-GtP that provides the automated picking is called Smart Item Robotics (SIR). It uses a robotic arm as shown in Figure 1.1. However, as of yet there exists no robotic solution that can handle every type of item for every task [135]. Additionally, the set of items encountered by FM-GtP is large and continually changing, such that keeping a database of item properties is impractical. Therefore in practice nothing about the item is known and any information about it can only be derived through sensors and algorithms. This diversity of items with unknown properties, combined with the need for high capacity, is the main challenge in engineering SIR.

Related Research

There are several aspects of SIR which can be improved to achieve a higher capacity. So far, research at Vanderlande has focussed on researching grippers and selecting the best one for SIR, as well as testing the system [36][76]. This has resulted in SIR being equipped with a vacuum gripper, as well as a good idea of the system's limitations. More recent research has looked into a subsystem that allows SIR to switch between grippers. This could greatly improve the number of items the system can handle, as one gripper's strengths could complement another gripper's limitations. However, this raises new questions: Which gripper should be added, and how should a grasp pose be determined? When presented with an item, which gripper should the robot select? When grasping with one gripper fails, should it try again with the same gripper or try another gripper?

Several areas of scientific research have attempted to enable robots to answer questions similar to those above. There is plenty of finished research available that covers gripper selection, gripper limitations, determining the grasp pose (grasp synthesis), and giving a confidence score for how likely a grasp is to succeed



Figure 1.1: A typical automated FM-GtP workstation [125] [128].

[135]. However, less research has been done for the latter type of question. Some robotics applications answer this question by defining rules at design-time and hard-coding them into the software. However, defining such rules quickly becomes impractical as the system becomes more complex [19], the resulting rule-based system cannot easily model uncertainty in the world [16], the rules are not easily re-usable [114], and the systems are often brittle from a control perspective [114]. Given the uncertainty in item properties in FM-GtP and Vanderlande's wish to have the result of this thesis also be applicable to related systems, it is not favourable to implement a rule-based system in SIR.

An alternative to rule-based systems is making the system adaptable using knowledge. Adaptability is defined by the Cambridge English dictionary as: "an ability or willingness to change in order to suit different conditions" [105]. In robotics, the definition of an adaptable robot ranges from robots able to handle different scenarios using pre-defined actions (such as grasping an item from different directions, depending on the item shape) [40], to robots developing new types of behaviour not specified at design time (such as re-learning to walk when a limb gets damaged) [66][136]. One of the most prevalent examples of knowledge-based systems in robotics is KnowRob [130]. KnowRob is an action-centred knowledge processing system that combines combines encyclopedic and common-sense knowledge with the information sensors provide about the robot's surroundings [130]. This allows the robot to do complex reasoning. For example, when presented with a partially set table, the robot can reason about what type of meal is likely to be had, which items are still missing, and where it can find those items [102].

KnowRob's implementation of knowledge uses ontologies [130]. An ontology can be defined as a set of concepts and categories in a subject area or domain that shows their properties and the relations between them [77]. Other research in robotics in which ontologies have proven useful is: helping the process of (automatically) designing the structure of a modular robot [107]; determining grasping areas through semantic knowledge (i.e. for grasping the handle of a mug) [137], learning grasp poses based on the item properties and task [16], achieving robustness against failures occurring in the system [55], inferring human intentions during surgical operations [87], providing the robot with common sense information about indoor environments [48], and reasoning about failures [37]. Because of this success, Vanderlande is interested in implementing reasoning in SIR.

Problem Statement

The goal of this thesis is to improve SIR by adding a second gripper and reasoning capabilities to improve SIR's capacity. The improved capacity comes from the larger variety of items SIR could handle due to the additional gripper and its increased robustness against failures due to the reasoning. The improved version of SIR is henceforth referred to as Reasoning Smart Item Robotics (RSIR, pronounced "arser"). Hence the main research question of this thesis is: *How can a pick-and-place task of a robotic arm with access to multiple grippers be modelled using knowledge representation and reasoning such that the system is robust against failures, such that it can handle a larger variety of unknown items, and such that it can be expanded to other related tasks?* To answer this, the question is split up into subquestions:

RQ1. What knowledge and reasoning capabilities should RSIR contain?

RQ2. What method best provides this knowledge and these reasoning capabilities?

RQ3. What software architecture best integrates this knowledge and reasoning with the robot?

RQ4. What is RSIR's performance on selecting the correct actions to be executed after the model and software architecture have been implemented?

To answer these questions, chapter 2 first provides background information on SIR. The requirements for RSIR are then defined in chapter 3. Based on the requirements, chapter 4 presents RSIR's approach to developing an ontology and achieving robustness, as well as the method for evaluating its performance. Chapter 5 then develops the grasping approach that is added to RSIR, after which chapter 6 presents the ontology and reasoning framework. The implementation of said framework is presented in chapter 7. The grasping approach and framework are then evaluated in chapters 8 and 9, respectively. Finally, chapter 10 concludes this thesis and chapter 11 presents topics for future work.

2

Specifications of the Pick-and-Place Robot

This chapter provides details on the design of the pick-and-place robot that is currently in use by Vanderlande (SIR) in order to provide background knowledge for designing RSIR. The chapter considers SIR's components in section 2.1 and its operational pipeline in section 2.2.

2.1. Components

The goal of SIR is to provide an efficient and high capacity picking solution. To achieve this, Vanderlande designed the automated FM-GtP workstation as shown in Figure 2.1. SIR's task is to move items between two carriers, whereas the higher-level system takes care of providing the correct carriers [125][128]. SIR is equipped with the components indicated in Figure 2.1. The components are:

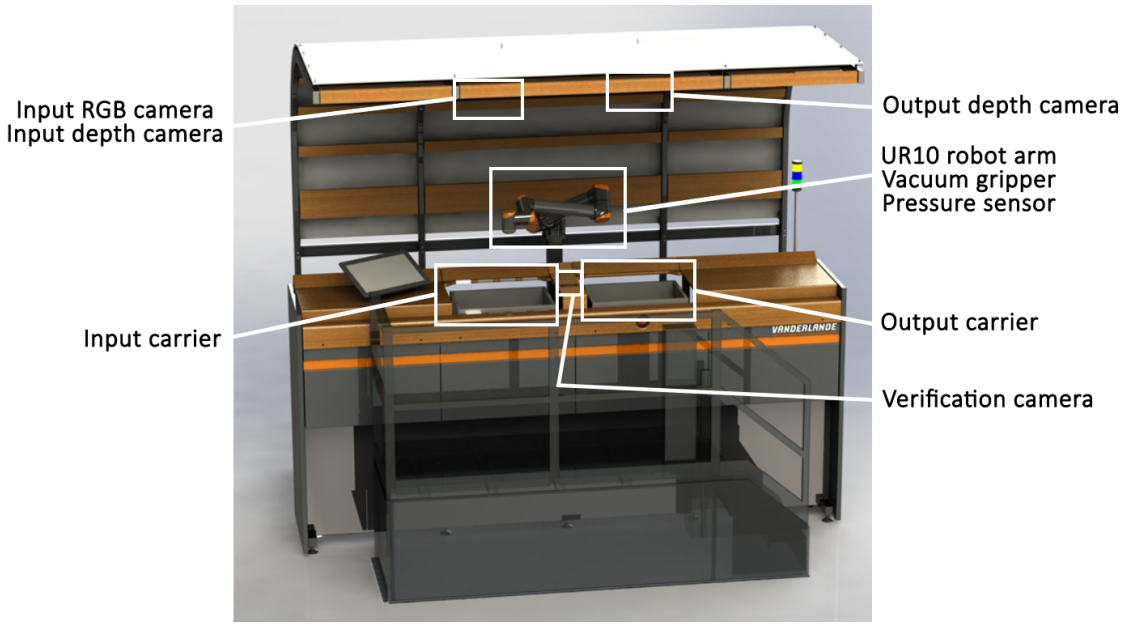


Figure 2.1: A typical SIR workstation [125] [128].

1. **Input carrier.** The input carrier provides items that SIR must pick. This carrier can have one of the following dimensions (length x width): 600x400mm, 650x450mm or 800x600mm, and one of the following heights: 270mm, 320mm, 370mm or 420mm. The maximum weight of the carrier is 50kg, with an average of 15kg. A carrier can have 1, 2, 4 or 8 compartments, as visualized in Figure 2.2a. The type of divider between the compartments can differ. Additionally, carriers can differ in colour, shape, and material. Figure 2.2b shows some examples of carriers. In practice, for a specific customer, the carriers can be assumed to not vary in shape, colour, material, and divider type [126]. Finally, there can be

one or multiple items of the same type per compartment, and the items can be in any random pattern. Figure 2.2c shows examples of item patterns in carriers.



(a) The types of compartments an input carrier can have [24].



(b) Examples of carrier designs [126].



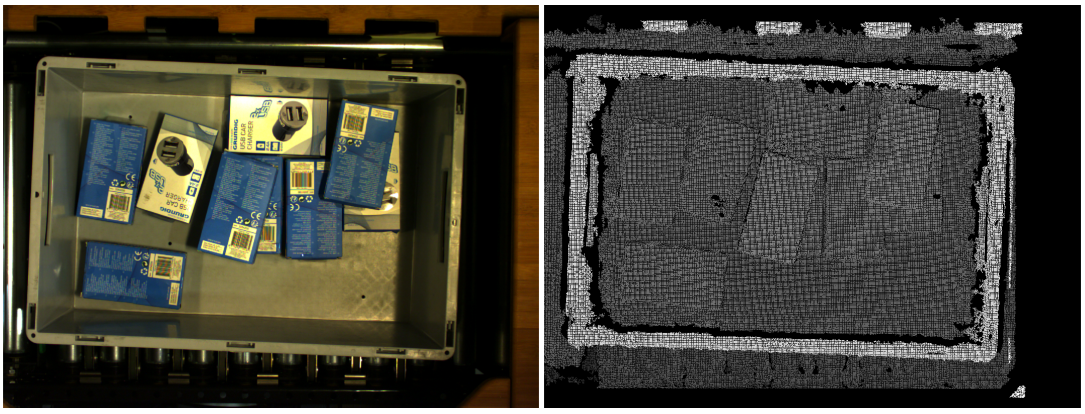
(c) Examples of how items are arranged in input carriers [126].

Figure 2.2: Types and examples of input carriers that can be encountered by RSIR.

2. **Output carrier.** The output carrier is where SIR places the items it picks. It has the same dimensions as the input carrier, but has a maximum weight of 23kg. Additionally, there may be a carton located in the carrier in which the item must be placed. There may be multiple (types of) items that must be placed in one output carrier.

3. **Input RGB camera.** SIR has an RGB camera with a resolution of 1600x1200 pixels, looking down into the input carrier. A light is located next to this camera to improve the image quality. Figure 2.3 shows an example of an image taken by the camera. Note that the carrier is not in the centre of the image.

4. **Input depth camera.** This depth camera looks down into the input carrier. It is calibrated to match the RGB image of the input carrier, as shown in Figure 2.3.



(a) The RGB image of the input carrier.

(b) The grey-scale image of the corresponding point cloud.

Figure 2.3: An example of an RGB image and the corresponding point cloud of the input carrier in SIR.

5. **Output depth camera.** This camera is of the same type as the input depth camera, but looks down into the output carrier. Again, the carrier is not guaranteed to be in the centre of the image.

6. **Verification camera.** This RGB camera is located between the two carriers and looks up. The roof of the workstation blocks surrounding light that otherwise reduces the quality of the image.

7. **UR10 robot arm.** This robot arm is compatible with SIR's vacuum grippers and several other common grippers. The arm can be augmented with a mechanism that allows the system to automatically switch between grippers. So far, this switching system has only been implemented for pneumatic grippers.

8. **Vacuum gripper.** The robot can be equipped with several vacuum grippers, differing in diameter, shape, and material. Each gripper comes with its (dis)advantages. Multiple grippers can be used if the switching system is implemented. SIR includes the vacuum system that powers the vacuum grippers.

9. **Pressure sensor.** This sensor is located inside the vacuum gripper. After an attempted grasp, the pressure inside the gripper is measured to determine whether the grasp was successful. If the measured pressure is sufficiently low, the algorithm concludes that an item is in front of the suction cup, and thus that the grasp was successful.

10. **Computer.** SIR's centre of control is a computer running on Ubuntu. The control algorithm uses ROS to communicate between the components of the system.

This concludes the background on SIR's components. The next section explains how these components are used to complete a pick-and-place task.

2.2. Operational Pipeline

This section discusses the operational pipeline of RSIR. Since RSIR is an improved version of SIR, RSIR must have all the capabilities that SIR has. Hence looking into SIR's pipeline gives a better idea of the requirements for RSIR. SIR's pipeline is visualized in Figure 2.4 and consists of the following steps [125][128]:

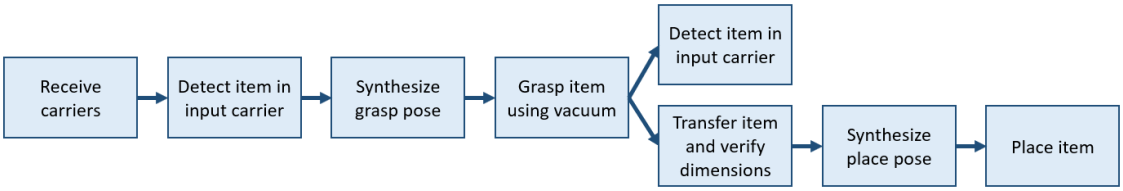


Figure 2.4: The pipeline SIR uses to complete the pick-and-place task.

1. **Receive carriers.** FM-GtP determines which order must be fulfilled and provides SIR with an input carrier, an output carrier, and the number of items to be displaced from the input to the output carrier.

2. **Detect item in input carrier.** The input RGB camera takes an image (Figure 2.5a), which is segmented by feeding it to a neural network. The network returns the segmented image with confidence values per segmented object (Figure 2.5b). The design and training of the neural network is outsourced.

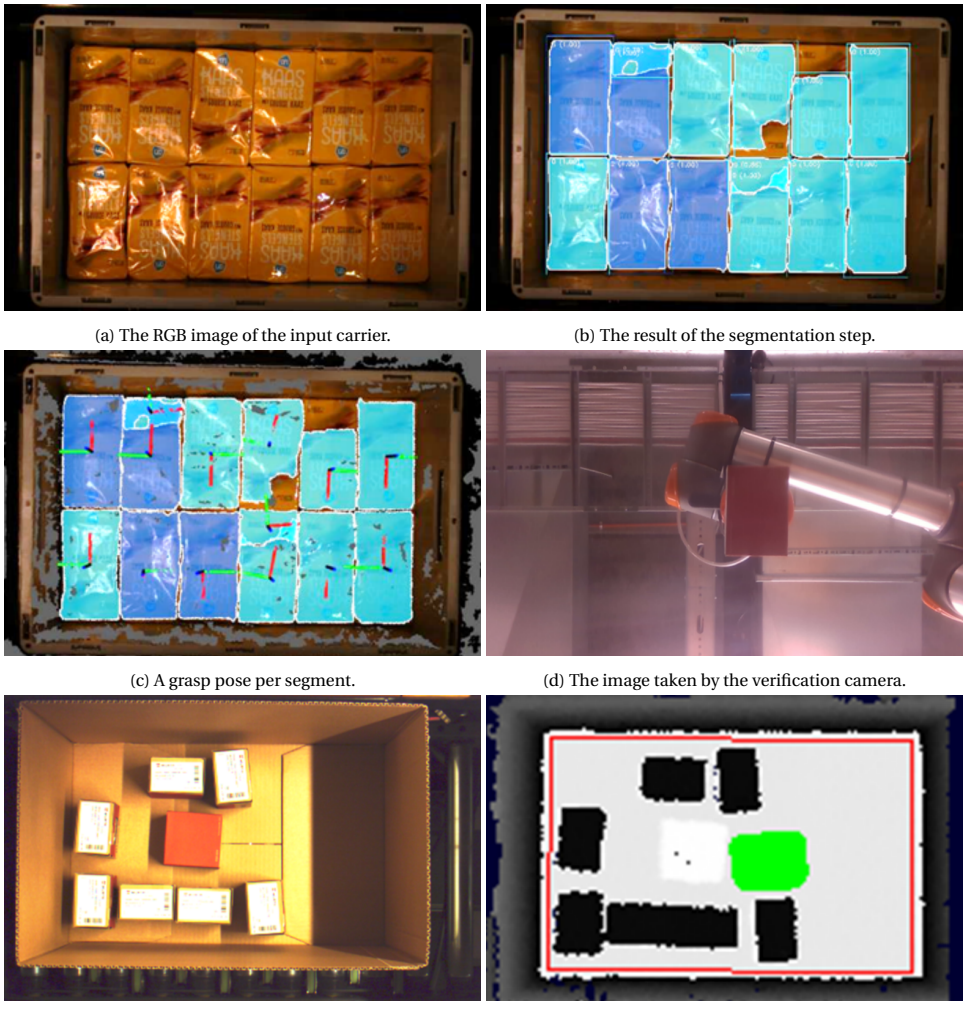
3. **Synthesize grasp pose.** The input depth camera takes an image, which is used for synthesizing the grasp for the vacuum gripper. This synthesis is based on finding flat areas in the resulting point cloud (Figure 2.5c). Five factors determine which grasp pose is the best: the confidence score of the segmentation; the height of the object; the size of the flat surface; the levelness of the surface with respect to the carrier; and how close the object is to the centre of the carrier. The best pose is selected to be executed. The specifics of selecting the best pose are unknown, since this step is outsourced.

4. **Grasp item using vacuum.** The motion planning algorithm moves the vacuum gripper to the selected grasp pose. The robot presses the suction cup onto the item and creates a vacuum. If the grasp fails, the robot moves up a little and retries the picking for the second-best grasp pose. A failed grasp is detected with the pressure sensor inside the gripper.

5. **Transfer item and verify dimensions.** After grasping, the motion planner directs the robot to hold the item above the verification camera. This camera takes an RGB image to verify that an item has been picked and to determine its footprint (Figure 2.5d). If more items must be picked from the input carrier, a new picture of the input carrier is taken at this step to speed up the process as a whole.

6. **Synthesize place pose.** A place pose in the output carrier is found using the footprint of the item and a depth image of the output carrier (Figures 2.5e and 2.5f). The algorithm for determining the place pose is outsourced, but the initial design minimized a cost function based on a preferred location (the centre of the carrier, for example) and the height of items already in the carrier at that location. While the place pose algorithm is running, the robot moves to a fixed point near the carrier.

7. **Place item.** Once the place pose is found, the motion plan is updated to reach said pose. This parallel process prevents the robot from wasting time by waiting for the place pose to be determined. Once at the place pose, the robot disables the vacuum and thus the item is placed in the output carrier. If more items must be grasped from the input carrier, the robot continues with the segmentation in step 2. Otherwise, the robot returns to its initial position and the pipeline starts again from step 1.



(a) The RGB image of the input carrier.

(b) The result of the segmentation step.

(c) A grasp pose per segment.

(d) The image taken by the verification camera.

(e) The RGB image of the output carrier.

(f) The computed placement position, indicated in green.

Figure 2.5: Visualization of steps in the pipeline of SIR [125].

Now that SIR is known in more detail, the next chapter defines the requirements for RSIR.

3

Requirements for the Reasoner and Grasping Approach

This chapter presents the requirements for RSIR, which are the starting point for determining what knowledge RSIR should contain and how this knowledge should be implemented. Section 3.1 provides the functional requirements and section 3.2 provides the non-functional requirements. Furthermore, section 3.3 presents requirements specific to the grasping approach. The MoSCoW method is used to provide a clear prioritisation amongst the requirements.

3.1. Functional Requirements

Recall that the goal of SIR is to provide an efficient and high capacity picking solution. In designing SIR, Vanderlande specified a list of requirements to make this goal more concrete. Since RSIR is an improved version of SIR, it must meet all the requirements for SIR, so most of the requirements defined by Vanderlande are relevant for RSIR [24][99]. Also recall that RSIR handles only the pick-and-place task, and therefore not higher-level system tasks such as providing the correct carriers and keeping a database of which items have proven to (not) be handleable by RSIR. The requirements for RSIR are:

Musts:

R1. RSIR must be able to implement an approach that moves items from the input to the output carrier.

R2. RSIR must provide information about the decisions it takes and failures that occur, as well as its reasoning steps. For example, it must provide information about why it thinks it is unable to handle an item.

R3. RSIR must be expandable to other related tasks, such as scanning the barcodes on items, or using a different type of gripper.

R4. RSIR must provide an improved level of robustness with respect to SIR, without requiring human intervention.

Shoulds:

R5. RSIR should handle a large collection of items, as specified in Table 3.1.

R6. RSIR should select a subset of components that leads to the highest predicted performance out of a larger set of redundant components. For example, RSIR should select the gripper out of multiple available grippers that leads to the highest predicted performance.

Coulds:

R7. RSIR could move to a specified position when a failure occurs that it cannot solve.

R8. RSIR could automatically start up all required components when it is restarted.

R9. RSIR could stop the execution of task upon request of the operator or FM-GtP.

Won'ts:

R10. RSIR won't have a GUI.

R11. RSIR won't be integrated with FM-GtP.

R12. RSIR won't have implementations for every related task.

R13. RSIR won't be tested for every related task.

Table 3.1: Requirements on an item to be handleable by the SIR system [24][99][125].

Characteristic	**Requirement**
Market segments	Food, General merchandise
Maximum applied force on item	10N
Maximum item dimensions	LxWxH: 300x300x200mm
Minimum item dimensions	LxWxH: 40x40x2mm
Maximum item mass	2000g
Maximum item mass (2020 goal)	6000g
Minimum item mass	20g
Preferred item orientation	Best pickable surface upwards
Items that cannot be handled	Fluid, inflammable, chemical, interlocking, clamped, and fragile items

3.2. Non-functional Requirements

Besides the functional requirements, that are non-functional requirements that RSIR must meet. The performance constraints are based on runtime, because the main goal of RSIR is to achieve a high capacity and runtime hinders this. Additionally, there are constraints regarding the resources available to develop RSIR.

Performance Constraints

R14. RSIR must complete its reasoning within a timeframe that is small with respect to the time it takes for SIR to complete a pick-and-place cycle. This is relevant because Vanderlande's goal is for SIR move items from the input to the output carrier at a rate of at least 600 items/hour [99].

R15. RSIR must do its reasoning on an ordinary computer.

Development Constraints

R16. RSIR must be developed within 9 months.

R17. RSIR must be developed on an ordinary computer.

R18. RSIR should have a reasonable cost with respect to SIR.

R19. RSIR should be designed to work on the UR10 robot in Vanderlande's lab in Eindhoven.

R20. RSIR should be developed using mainly components that are available at Vanderlande.

R21. RSIR could be designed to work on the UR10 robot in Vanderlande's lab in Veghel.

R22. RSIR won't be designed to work on any robotic set-up.

3.3. Requirements for the Grasping Approach

The grasping approach is one specific aspect of RSIR and has additional requirements. Besides the general requirements for RSIR, the following requirements apply to the grasping approach:

Musts:

R23. The approach must grasp as many items as possible that SIR cannot grasp due to limitations of SIR's vacuum gripper and/or SIR's grasp synthesis approach.

R24. The approach must select a grasp pose with a large chance of the pose resulting in a successful grasp.

R25. The approach must grasp an item from an input carrier with only one compartment if there are no other items in the carrier and the segmentation of the item is given.

Shoulds:

R26. The approach should make use only of components available to SIR (the input depth camera, for example). This includes not assuming any knowledge about the item, since no such knowledge is available to SIR.

R27. The approach should handle the items gently, that is, without damaging the items or leaving marks.

R28. The approach should grasp an item in clutter from an input carrier with only one compartment if the segmentation of the input carrier is given, and only one type of item is in the carrier.

Coulds:

R29. The approach could grasp an item from an input carrier with more than one compartment.

R30. The approach could grasp an item an input carrier for which more than one type of item is within a compartment.

R31. The approach could grasp an item from an input carrier if the segmentation of the item(s) is not given.

R32. The approach could be tested on the physical set-up of the UR10 robot in Eindhoven.

Won'ts:

- **R33**. The approach won't be generalizable across different gripper technologies. That is, the synthesis approach won't be generalizable across fingered grippers, vacuum grippers, magnetic grippers, etc.
- **R34**. The approach won't be tested on a full set of items using a non-changing set of parameters. Therefore the result of this thesis won't include a complete overview of the capabilities of the approach.

In summary, the functional requirements focus on answering research questions Q1 and Q2, whereas the non-functional requirements focus on performance and thus answer research questions Q3 and Q4. The next chapter discusses several high-level decisions about RSIR's design based on these requirements. In the remainder of this thesis, the requirements are referred to by their number.

4

System Design and Verification

This chapter discusses the high-level decisions about RSIR's design and presents the method for evaluating RSIR's performance. Section 4.1 introduces the concept of ontologies and the method for developing RSIR's ontology. Afterwards, section 4.2 decides which approach RSIR applies for achieving robustness. Finally, section 4.3 defines the methods used to evaluate RSIR. Based on these decisions, the following chapters can design RSIR's new grasping approach and ontology.

4.1. Knowledge Requirements

This section addresses the problem of determining the knowledge RSIR should contain and the type of representation that RSIR uses to capture this knowledge. Thus this section answers the research question *"What knowledge and reasoning capabilities should RSIR contain?"* Additionally, by selecting the representation, this section takes a first step in answering the research question *"What method best provides this knowledge and these reasoning capabilities?"*

Section 4.1.1 provides background information about capturing knowledge in ontologies and justifies the use of ontologies in RSIR. As will become clear in chapter 6, there does not yet exist an ontology that captures all knowledge required by RSIR. Hence this thesis develops its own ontology, for which the method is explained in section 4.1.2. Finally, section 4.1.3 creates the requirements for the knowledge RSIR must contain. Based on these requirements, the remainder of this thesis develops the ontology.

4.1.1. Justification for Ontologies

This subsection justifies the use of ontologies in RSIR. An ontology can be defined as a set of concepts and categories in a subject area or domain that shows their properties and the relations between them [77]. The strength of ontologies comes from the fact that they are formally encoded and highly expressive, yet suited for real-time processing by a robot [87]. Ontologies' high expressiveness allows the definition of complex concepts, which help in defining concepts such as tasks and how to complete them (R3) [107]. Furthermore, the formal encoding provides researchers and robots with a common language: if two robots define their knowledge in the same ontology, communication between these robots is simple. This is in contrast with two robots that define their knowledge in their own database, as the latter raises problems due to different definitions of the same knowledge [42]. The formal encoding allows for robots to re-use knowledge developed for other robots, thus simplifying the implementation of new behaviour (R3) [131]. Furthermore, the real-time processing aspects of ontologies keeps runtime reasonable (R14). In previous research, ontologies have proven useful in grasping (R1) [16] and in achieving robustness against failures (R4) [37][55]. Finally, an important advantage is that ontologies provide insight into the robot's thought process (R2) [130].

However, the use of ontologies also has disadvantages, some of which depend on design decisions. One design decision is the ontology language. Examples of languages are OWL, SWRL and SHACL [49][65][87]. A common trade-off in ontology languages is between expressiveness and decidability [87]. More expressive languages allow to define more complex concepts, but may end up being unable to derive the correct answer to a question (that is, the language is undecidable). A second design decision is which reasoner to use. The ontology language provides facts, but the reasoner is the algorithm that derives conclusions from these facts. Other decisions are which method and tool to use to develop the ontology, what knowledge to contain in the

ontology, and which existing ontologies to base the new ontology on. The next section describes the method by which decisions about these aspects are made for RSIR.

Besides being limited by the trade-off in expressiveness and decidability, ontologies are limited by knowledge representation in general. The main problem with knowledge representation is the sheer complexity of reality. Capturing this complexity takes skill and time, and is an ongoing process [96][98][106]. Another limitation specific to ontologies is the grounding problem. This problem refers to the difficulty in connecting abstract structures to concrete objects [87]. For example, it may be complex to convert the point cloud that a robot observes to the abstract idea of objects that the robot has (for example, when a robot defines objects by their function). However, despite these disadvantages, ontologies have proven useful for robots [16][37][48][55][87][98][107][137]. Because of the advantages of ontologies and their success in previous research, ontologies are selected as the basis for RSIR.

4.1.2. Ontology Development Method

This subsection outlines the methodology used for developing RSIR's ontology. One methodology for this is METHONTOLOGY, henceforth referred to as Methontology. It provides a list of activities to perform in order to develop an ontology [42]. Out of the existing methodologies, Methondology was chosen because of its structured steps and its role in the development of several respected ontologies [13][59][106]. This structured approach helps in designing an ontology that meets all of RSIR's requirements from chapter 3.

Methondology considers seven phases. Note that these phases do not need to be sequential. In fact, the authors behind Methondology argue that ontologies grow by evolving (as opposed to incrementing), and designing the ontology is thus an iterative process. The phases in Methontology are [42]:

1. Specification. In this phase the ontology's purpose, level of formality, and scope are determined. One way to do this is through competency questions: questions that the ontology must be able to answer.

2. Knowledge acquisition. This phase is mostly done simultaneously with the specification phase. The knowledge acquisition phase provides the domain knowledge required to specify the ontology.

3. Conceptualization. The aim of this phase is to structure the domain knowledge based on the results of the specification phase. The knowledge is split into concepts and verbs.

4. Integration. Integrating the ontology with existing ontologies comes in two steps. The first is to select one or multiple meta-ontologies that fit the conceptualization of the new ontology. Meta-ontologies are concerned with the nature of the ontology [26], which is often the highest level of the ontology. These types of ontologies describe general aspects such as space, time, and events [59]. The second step is relating the new ontology to existing ontologies, for example by using the same terms as existing ontologies.

5. Implementation. In this phase, the ontology language, reasoner, and tools are selected, and the ontology is implemented.

6. Evaluation. The aim of this phase is to confirm that the ontology is correct, and to confirm that the ontology meets the requirements from the specification phase.

7. Documentation. Although there exists no formal way to document ontologies, Methontology aims to provide formal documents as the output of each of the other phases.

This thesis goes through all seven steps to develop the ontology for RSIR. The Specification step is considered in the next subsection.

4.1.3. Ontology Requirements: Competency Questions

This subsection addresses the problem of defining the knowledge RSIR must contain (RQ1). Following Methontology, this knowledge is defined through competency questions. The questions are constructed based on the requirements from chapter 3 and are split up into three categories: tasks, failures, and items.

Task Questions

These questions are constructed to meet requirements R1, R3, R5, R6, and R14. To implement SIR's pick-and-place approach (R1) and to be expandable to related tasks (R3), RSIR must know what capabilities it has access

to and which of these it should use to complete a task. Furthermore, RSIR must know what performance each approach has (R5, R14) and it must select the approach with the best performance (R6). The task questions are:

Q1. What capabilities does the system have access to?

Q2. How are the capabilities constrained?

Q3. What is the predicted performance of each capability?

Q4. What actions must the system do to complete a task?

Q5. How confident is the system that it can complete a task?

Q6. What is the predicted performance for completing a task?

Q7. If there are multiple approaches to completing a task, which approach is optimal?

Failure Questions
The failure questions are constructed to meet requirements R2, R4, R5, and R23. The main goal in answering these questions is to achieve robustness (R4) while providing insight into the robot's thought process (R2). Furthermore, knowledge about the item's properties helps in selecting the best grasping approach (R5, R23). The questions are:

Q8. What kinds of failures can occur?

Q9. Which of these failures are likely to occur?

Q10. When a failure has occurred...

(a). ...what type of failure is it?

(b). ...what is the cause behind the failure?

(c). ...what can be concluded about...

(i). the properties of the item?

(ii). the capabilities of the robot?

(iii). the actions the system should take to prevent this failure from re-occurring?

Item Questions
The item questions are constructed to meet requirements R3, R5, and R23. First and foremost, knowledge about the item's properties helps in reasoning how to complete a task (R3), an example of which is selecting the best grasping approach (R5, R23). Second, RSIR must be able to estimate item properties, which is one of the related tasks from requirement R3. The questions are:

Q11. Which item properties are relevant for the task at hand?

Q12. Which item properties are independent of the scenario?

Q13. Which item properties are dependent on the scenario?

Q14. Which item properties are known, and with what accuracy?

Q15. Which item properties can be estimated?

Finally, there is one competency question that RSIR must be able to answer, but for which the algorithm is not designed in this thesis. This question is: *"What tasks should the system do to complete a goal?"* For example, if the goal is to grasp an item, the system must decide whether to immediately try to grasp it, or to first try to gain more knowledge about the item's properties. When answering this question, at least the following must be taken into account: which item properties are relevant for the goal, the uncertainty in the system's knowledge of the properties, which capabilities are constrained by the property, with what accuracy the property can be estimated, what the advantage is of knowing the property, and what the cost is of estimating the property. Clearly, an algorithm that properly answers this question is quite complex. Therefore,

the design of such an algorithm is outside the scope of this thesis. However, RSIR must be designed to allow for such an algorithm to be implemented in future work.

This concludes the specification of the knowledge RSIR must contain. This list of competency questions is used in chapter 6 when the ontology is developed. But before this is done, several other decisions about RSIR must be made. The next section considers RSIR's approach to achieving robustness.

4.2. Achieving Robustness

This section addresses the problem of which approaches to use for achieving robustness against failures (R4). There are two main aspects to this: the type of human interaction and the type of adaptive behaviour.

4.2.1. Human Interaction

The type of human interaction is relevant in achieving robustness because for the current state of SIR, the human operator consistently outperforms SIR in completing the pick-and-place task. Hence a version of RSIR that cooperates with a human operator could rely more heavily on the operator to detect and handle failures, rather than on its own robustness. There are two main categories in human-robot interaction: traded control and shared control. SIR is an example of traded control, which is when the robot completes some parts of the task and the human completes other parts of the task [71]. For SIR, the tasks that the human completes are handling failures and tasks that the robot cannot do (yet), such as confirming that the correct items are in the output carrier. The most prominent disadvantage of this type of control is that the robot loses situational awareness when the human takes control, and vice versa [71]. This is an especially pressing problem for applications such as autonomous vehicles, where the human can be too late to take over from the robot, resulting in an accident.

The type of control commonly considered the counterpart of traded control is shared control. This type of control stresses the fact that human and machine share control over a system together [43]. This allows the robot and human to keep situational awareness [10]. Research into driving shows that the control is faster and more accurate, costs less effort, and demands less visual attention than if the human were to control the system without help from the robot [10]. However, the main disadvantage for shared control is that a human operator is still required. Because of this, shared control does not provide the type of robustness that is defined by requirement R4.

From this inspection, it becomes clear that traded control is the better option for RSIR, because shared control does not meet requirement R4. The lack of situational awareness is less of a problem for SIR, as the task is quite simple from a human perspective and not as time-critical as, for example, autonomous driving. On top of that, subsystems are in place to let the human operator update the system's state (such as the number of items to be displaced) [24]. Hence the remainder of this thesis only considers traded control.

4.2.2. Adaptive Behaviour

The second aspect in achieving robustness is the type of adaptive behaviour. This behaviour ranges from the robot selecting from a pre-defined list of actions [40] to the robot developing qualitatively new behaviour [66][136]. It was concluded previously that selecting from pre-defined actions based on rules defined at design time is not sufficient for RSIR because of its time-consuming design process and its limited re-usability (which goes again requirement R3). That leaves two types of adaptive behaviour RSIR could implement: combining existing capabilities into new actions and developing qualitatively new behaviour.

A robot developing qualitatively new behaviour can be considered resilient. Resilience goes beyond robustness because it introduces adaptation and emergence of new solutions [136]. As an example, consider a four-legged robot pre-programmed with four types of walking (gaits). For several types of terrain, this might be sufficient. However, it is likely that all four gaits are unusable if the robot loses one leg. A resilient robot in such a scenario would develop a new type of gait that works for three legs, commonly through some form of reinforcement learning [66]. However, the introduction of reinforcement learning in RSIR raises several concerns. First, machine learning approaches generally give little insight into the robot's though process [135], thus violating requirement R2. Secondly, the reality gap poses limitations in applying machine learning to RSIR [66]. For one, some of the most challenging items to grasp are porous or deformable items [135]. These are especially time-consuming to simulate [135] and thus significantly increase the reasoning time the robot requires, violating requirement R14. Additionally, the reality gap forces the robot to attempt at least some

solutions on the physical robot (as opposed to simulation). This can lead to the robot exhibiting unexpected behaviour, especially if the reality gap is large. Such behaviour is undesirable and potentially unsafe in an environment where human operators are expected to work near the robot.

The alternative to developing quantitatively new behaviour is applying known capabilities in new ways [55][66][143]. For example, the robot might decide that the vacuum gripper is broken, and thus will only rely on other grippers from that point onwards. This type of adaptive behaviour is what KnowRob is primarily occupied with: using basic actions (moving, grasping) to complete a high-level task (setting the table) [19]. This is no surprise, as ontologies are better at capturing capabilities (and their limitations) than at capturing machine-learned behaviour [16][102]. However, this type of adaptive behaviour is clearly less robust than the behaviour that develops qualitatively new behaviour. Returning to the example of the four-legged robot: if no gaits are specified at design time that work with three legs, the robot will never recover if it loses a leg.

Given the limitations of developing qualitatively new behaviour, it can be concluded that the best approach for RSIR is to reason about applying known capabilities in new ways. The limited resilience that results from not developing qualitatively new behaviour is unfortunate. However, it should be noted that so far, resilience has mostly been applied in systems that are outside of human reach (such as in outer space) and systems for which it is hard to diagnose the failure (such as controller tuning) [66]. Neither of these aspects is particularly relevant for RSIR: at least one human operator is supervising a group of RSIRs, and several failures that are difficult for RSIR to handle can be easily diagnosed by a human. Examples of the latter are obscured cameras and broken grippers. Although resilience in such scenarios might be desirable, for a first version of SIR it is sufficient if detecting and solving such failures is left to operators and engineers.

It should be noted that a key factor in making this approach work is for RSIR to have a redundancy in components that fulfil a capability. For example, two grippers that fulfil the capability of grasping an item, or one synthesis approach that can be run with two different sets of parameters. If there exists no redundancy in the system, a lot of the advantages of reasoning are lost, as there are no alternative pipelines the robot can choose. Because RSIR is to be equipped with multiple grippers, cameras, and synthesis approaches, this requirement is fulfilled. Even so, the more redundancy RSIR has, the more value its reasoning can provide.

To conclude, the remainder of this thesis focusses on providing RSIR with the reasoning capabilities to apply pre-defined capabilities in new ways, and not on learning quantitatively new behaviour. Additionally, RSIR communicates with human operators via traded control.

4.3. Evaluation Method

This section considers the evaluation methods used to test RSIR. It considers the available materials and two key evaluations: evaluating the grasping approach and evaluating the ontology and reasoning framework. Additionally, RSIR is implemented for a picking task on a physical robot.

4.3.1. Materials

Because RSIR is to be tested on the robot in Eindhoven (R32), several components that are normally part of SIR are not available during testing. The components that are available are:

- UR10 robot
- Input and output carriers
- A vacuum gripper
- An assortment of other grippers
- The input RGB and input depth camera
- A computer running ROS Melodic on Ubuntu 18.04 (R15, R17)

The most notable missing components are the segmentation algorithm, the vacuum grasp synthesis algorithm, the verification camera, and several components required for placing. As a result, it is impossible to test all of RSIR on a physical set-up. Instead, only the grasping approach is evaluated using images collected from the set-up in Eindhoven, and the reasoning of RSIR is tested virtually. Additionally, RSIR is implemented for a grasping task on the set-up in Eindhoven to prove that its reasoning transfers to a physical robot.

4.3.2. Grasping Approach

The grasping approach developed in this thesis consists of two aspects: the gripper technology and the synthesis approach. Both could be evaluated in detail, to provide RSIR with accurate knowledge about their limitations. However, limitations of gripper technologies is a well-researched topic. To reduce its scope, this thesis does not evaluate the gripper technology but instead derives any knowledge about it from literature. On the other hand, the grasp synthesis approach is evaluated, as its limitations are less general and dependent on the implementation. Even so, the synthesis approach evaluation is not meant to be a thorough testing scheme, but rather a brief test to get a general idea about its limitations (R34).

The synthesis approach is evaluated in chapter 8 based on the variety of items it can synthesize reasonable grasps for (R23), the quality of the grasp the algorithm concludes to be the best grasp (R24), and the time it takes to synthesize a grasp (R14). This performance is evaluated for different types of clutter (R25, R28), different segmentations (R25, R31), and carriers containing one or more types of items (R25, R28, R30). Additionally, an evaluation is done to investigate the influence of the tuning parameters on the algorithm's performance.

For all evaluations of the grasping approach, RGB and depth data is used that is obtained from the set-up in Eindhoven (R19, R32). The quality of a grasp is determined by a human. A more appropriate evaluation would be to attempt the grasps and compute what percentage of them were successful. However, this was not possible because of time constraints (R16) and the limited components available in Eindhoven (R19). Although a human evaluation is more subjective, it should be valid because humans are masters of grasping [95]. The evaluation method is explained in more detailed in chapter 8.

4.3.3. Ontology and Reasoning

The ontology and reasoning are evaluated in two ways. The first way is by investigating whether the competency questions (section 4.1.3) are answered. This is relatively straightforward, as the competency questions are simple enough to easily be answered by a human. They are answered in section 9.1.

The second way RSIR is evaluated is by testing its performance on a robotic set-up. Due to the limited components available in Eindhoven, this test is done virtually. The robot is presented with an item of unknown properties and its task is to grasp it. RSIR is evaluated based on how quickly it finds the optimal way to grasp the item (R6) and how long its reasoning takes (R14). Additionally, RSIR is evaluated based on how well it estimates the properties of the items it is trying to grasp, because this is helpful in selecting the best grasping approach (R6) and provides insight into its through process (R2). This performance is evaluated for four scenarios: the baseline scenario, a scenario where RSIR has access to prior knowledge, a scenario where RSIR occasionally explores, and a scenario where RSIR has imperfect knowledge about its capabilities. The full details on these evaluations are provided in chapter 9.2.

This concludes the high-level decisions about RSIR's design and evaluation. The next step in designing RSIR is Methontology's knowledge acquisition phase. Because one of the dominant choices in RSIR is the grasping approach, the next chapter investigates available approaches and selects the best one for RSIR.

5

Grasping Approach

This chapter addresses the problem of analysing suitable gripper technologies and grasp synthesis approaches, and selecting the best gripper and approach for RSIR. Furthermore, this chapter investigates the limitations of said synthesis approach, and investigates how to negate these limitations in RSIR. This research is required for the knowledge base, because the available grasping approaches (and their limitations) is knowledge RSIR must have. Additionally, requirement R5 demands that RSIR can handle a large variety of items, but currently RSIR's performance is heavily limited because it only has access to one type of gripper. Figure 5.1 illustrates this: the vast majority of failures is due to failed grasps or failed synthesis attempts (as opposed to inaccurate motion planning, for example). Hence the addition of a second gripper is critical in improving RSIR (R23).

The problem is addressed as follows: section 5.1 provides details about the limitations of RSIR's vacuum gripper, after which section 5.2 defines the requirements for the new gripper. Section 5.3 then investigates available gripper types and selects the best one for RSIR. Finally, section 5.4 addresses the problem of selecting the best synthesis approach and suggests improvements.

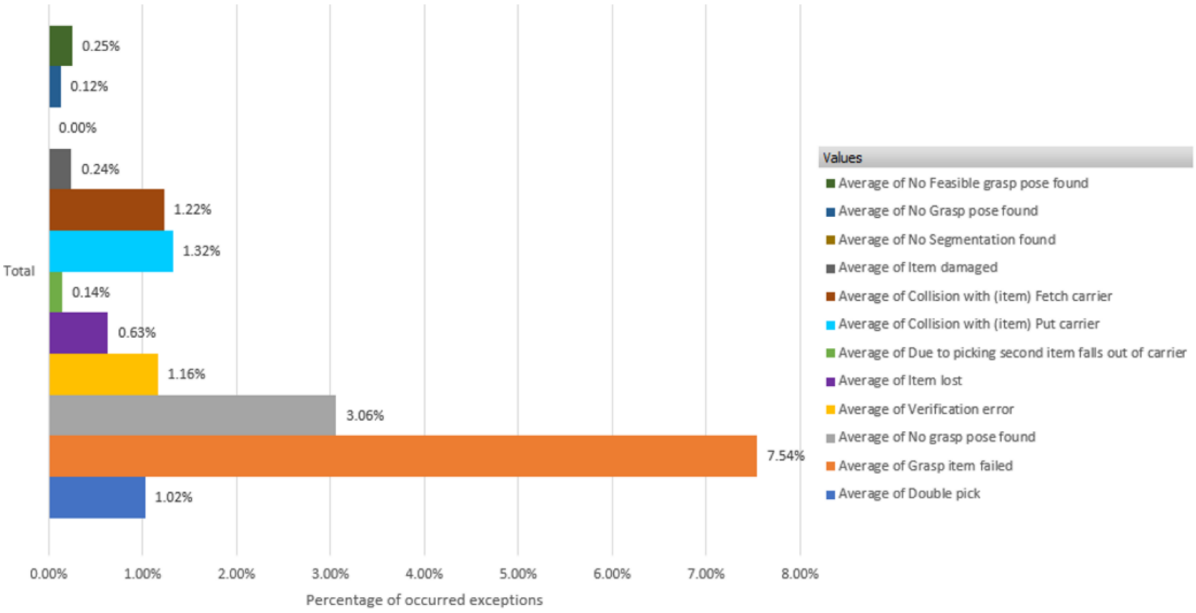


Figure 5.1: Failures observed for SIR [36]. Listed percentages are percentages of the total number of pick-and-place attempts performed.

5.1. Vacuum Performance

This section investigates the limitations of SIR's vacuum gripper. Recall that section 3.1 put limits on the items that SIR must be able to handle, such as the minimum and maximum item mass. However, there are

still items within these limits that cannot be grasped by SIR. In general, these are items that are too porous, have too little surface contact with the suction cup, deconstruct easily (such as items with paper wraps), or items that are too densely packed [125]. Items that are relatively heavy can also pose problems, often due to their eccentric centre of mass [125]. This causes the item to swing when the gripper picks it up, which breaks the air seal between item and gripper, resulting in the item being dropped. Some typically ungraspable items are shown in Figure 5.2. The findings of Vanderlande are consistent with those from literature, which also mention vacuum grippers' disadvantages for porous items and items with eccentric centres of mass [52].



Figure 5.2: Examples of items that are typically not graspable with a vacuum gripper [125].
From left to right: a hair brush, irregularly shaped items in packaging, a belt, a pack of sodas, and a bag of dog food.

Vanderlande makes use of the GS1 item categories for testing SIR's performance. The GS1 standard provides a common language for item packaging [97], using packaging types such as "box", "net", and "not packed". Figure 5.3 summarizes SIR's performance for the categories relevant for Vanderlande. The results clearly show in which categories (and their mass and size ranges) there is room for improvement. However, not all failures shown in the figure are due to grasping failures. Other failures also lower the system's capacity, such as failures due to collisions and dropping the item. The gripper hardware has a relatively small influence on these failures compared to other components such as motion control. Hence the percentages in Figure 5.3 do not directly represent how much the system can be improved by adding a gripper.

		0 - 500 g			500 - 1000 g			1000 - 1500 g			1500 - 2000 g		
		small	medium	large	small	medium	large	small	medium	large	small	medium	large
CT	tray		93%	83%		95%	90%			50%			83%
CU	cup,bowl	99%	95%	100%	100%	95%			95%				
CY	container	90%	99%		93%	100%			90%				
BPG	blister	70%	92%	100%			100%						
BX	box	89%	94%	75%	100%	98%	95%					90%	88%
BO	bottle	90%	100%		40%	88%			70%			85%	
CS	case		100%										
SY	sleeve	92%	85%	83%			70%						
CM	card	60%	60%	65%									
SW	shrink wrapping	93%	90%	98%		90%	55%		100%	100%		50%	90%
BK	basket												
MPG	multipack	90%	94%	98%		85%	100%		85%	70%			
NT	net		45%			40%							40%
BRI	pack	83%							95%			100%	
GTG	pack with point		100%		98%	88%			88%			100%	
JR	jar	99%			100%	100%		95%	98%			30%	
TU	tube	88%											
BG	bag	93%	95%	98%	88%	98%	93%		88%	75%			83%
BBO	boxblister open	93%	88%										
BXP1	boxplastic1		100%	100%		100%	95%	95%					
BXP2	boxplastic2	60%	78%			87%							
NE	not packed	40%	50%	0%		0%							
NEF	not packed fresh												

Figure 5.3: Success rates of tests for SIR, adapted from [36]. The success rate is defined as the number of successful grasp attempts divided by the total number of grasp attempts. A grasp attempt is considered successful if the first or second grasp succeeds. Note that the results for the "not packed" category are based on only a few tests, as it was found that the category gives little information on the system's performance because of the large variety in item properties.

Instead, the potential for improvement depends on both the success rate (as depicted in Figure 5.3) and the fraction of failures that are due to grasping (both in grasp synthesis and grasp execution). The product of the fraction and the success rate gives a measure for the improvement that can be realized by adding a gripper. Table 5.1 lists this measure per item category. It shows that the not packed, net, card, boxplastic2,

and blister categories are the five most promising categories to improve on. The GS1 definitions and examples for these categories are included in Appendix A. The reader is referred to the GS1 documentation [97] for the definitions of all categories.

Table 5.1: Potential improvement of the FM-GtP system when adding an extra gripper. The table lists the item category, the success rate of the system using only a vacuum gripper, the percentage of failures that are due to grasping (with respect to all failures), and the potential improvement. Numbers are based on data from tests performed by Vanderlande [36].

Category	Explanation	Success rate (%)	Grasp failures (%)	Potential (%)
All	all categories	83	70	12
NE	not packed	32	88	60
NT	net	33	73	49
CM	card	61	93	36
BXP2	boxplastic2	77	82	19
BPG	blister	82	91	16
CY	container	84	87	14
TU	tube	88	100	12
BO	bottle	81	60	11
CT	tray	83	56	10
BBO	boxblister open	89	76	8.4
JR	jar	89	73	8.0
SY	sleeve	87	61	7.9
BX	box	90	56	5.6
BG	bag	91	61	5.5
MPG	multipack	89	40	4.4
BRI	pack	93	61	4.3
SW	shrink wrapping	86	23	3.2
BXP1	boxplastic1	98	25	0.50
CU	cup, bowl	96	8.2	0.33
CS	case	100	0	0
GTG	pack with point	100	0	0
BK	basket	no data	-	-

Now that the limitations of the vacuum gripper are known, the requirements can be defined for the gasping approach that is added to RSIR.

5.2. Requirements

This section defines the requirements for the grasping approach that is to be added to RSIR. These requirements are derived from the requirements for RSIR as described in chapter 3, and mostly the limitations of SIR's vacuum gripper (R23). The requirements are:

1. **Item variety**. The variety of items the grasping approach can (theoretically) grasp should be as large as possible for the items that the vacuum gripper cannot handle (R5, R23). For grasping, items are often defined by five properties: shape, size, mass, texture, and deformability [135]. Based on the analysis in the previous section, it can be concluded that shape, texture, and deformability pose the largest problems for SIR.

2. **Computational cost**. This cost should be as low as possible so that the system can reach as high a capacity as possible (R14).

3. **Repeatability.** The grasping approach's repeatability is important because it allows the robot to consistently perform successful grasps (R5, R23). Additionally, a lower variability in grasp success allows for better reasoning and learning about which grasps are successful for certain items (R6).

4. **Lifetime**. The grasping approach's lifetime should be long to reduce maintenance and repairing costs (R18).

5. **Cost**. The grasping approach should have a low cost with respect to SIR (R18).

6. **Modularity**. The grasping approach must be (made) compatible with a switching system that allows RSIR to switch between grippers and synthesis approaches (R6).

7. **Gentleness** The grasping approach should handle items without damaging them or leaving marks (R27).

8. **Dexterity**. A grasping approach's dexterity refers to the variety of tasks that it can complete, as well as how well it performs in those tasks [85]. A high dexterity is preferred as it allows for generalizing the system to similar tasks (R3).

9. **Availability**. A grasping approach that is already available (at Vanderlande) is preferred to one that is not, because of the reduced costs, time, and effort required (R16, R18, R20, R26).

The next section presents gripper technologies and determines which gripper technology best meets the requirements defined in this section.

5.3. Gripper Technology

This section presents the types of gripper technologies available and selects the one best suited for RSIR. To do so, this section researches the available gripper technologies and ranks them based on the requirements from the previous section.

5.3.1. Available Technologies

There are many gripping technologies available besides vacuum grippers. This subsection elaborates on the ones relevant for RSIR. A previous study within Vanderlande found the technologies in Figure 5.4 to be applicable for the type of grasping required in their material handling systems [76]. However, out of these only two are suitable for SIR: vacuum and clamping. The remainder is not applicable because of the limited item masses, dimensions or materials that can be handled (R5, R23), or because the item gets damaged (R27). Supporting is the exception, but is not applicable because the design of the carrier does not allow approaching from the bottom [76][134]. Since vacuum is already implemented, a clamping gripper is selected for RSIR.

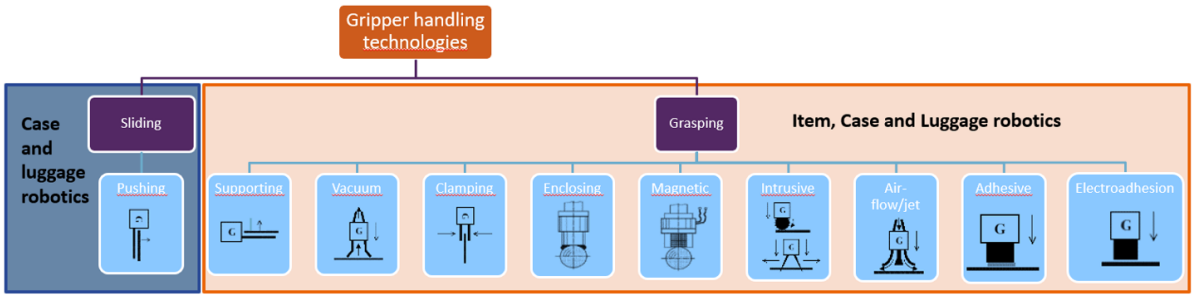


Figure 5.4: [134]

However, within the category of clamping grippers there is still a lot of variety. The study by Vanderlande identified mechanical grippers with two to five fingers, soft grippers, and mechanical suction grippers. The latter is a combination of a suction cup with mechanical fingers. In addition to those, this section also considers parallel-jaw grippers. The gripper types and their (dis)advantages are now discussed

Parallel-jaw Gripper

Parallel-jaw grippers are among the most commonly used grippers in robotics applications [135]. Some examples are shown in Figure 5.5. These grippers feature two parallel plates, which move towards each other when the gripper closes. They have been shown to be capable of grasping objects of different sizes, shapes, masses, textures, and deformabilities [135]. Other advantages are their simple and computationally cheap synthesis approaches, the available datasets applicable for machine learning, and their superior grasping success rate compared to grippers with three or more fingers [135]. They are also significantly less expensive than grippers with three or more fingers [76]. Finally, it is relatively simple to physically connect them to the robot, making them a good candidate when it comes to modularity.

Disadvantages of parallel-jaw grippers are their limited compliance, limited dexterity, and limited stroke [76][85]. The latter puts an upper bound on the dimensions of items the gripper can handle. Parallel-jaw grippers with larger strokes are available, but are usually more bulky and heavy, which poses a problem given the limited payload of the robot arm [63]. It should be noted, however, that the dexterity of a simple gripper combined with a dexterous robot arm is sufficient for many manipulation tasks [85]. Even so, added dexterity may be beneficial in reducing execution time and power consumption [85]. Another note is that the grasping approaches that have proven successful on a large variety of items do not consider a limitation in grasping force (thus potentially damaging the item) and occasionally pick up several items at once [135].

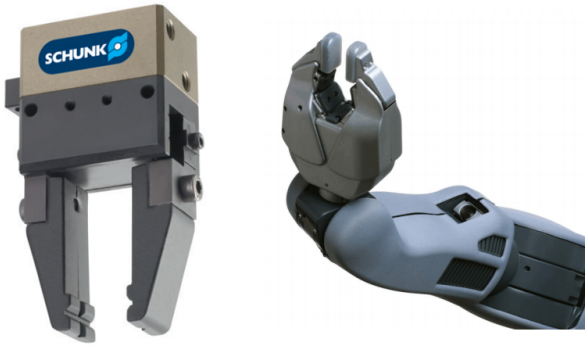


Figure 5.5: Examples of parallel-jaw gripper: a Schunk gripper (left) [31] and the gripper of a PR2 robot (right) [23].

Rotational Joint Gripper

Rotational joint grippers are similar to parallel-jaw grippers, but have a different way of closing. Some examples are presented in Figure 5.6. Instead of having plates close parallel to each other, a rotational joint gripper has joints that make it close like a human finger would. This allows for more self-solving behaviour [76], especially if the gripper is back-drivable. Depending on the gripper and item shapes, the gripper may provide form-closure grasps. Other than these aspects, rotational joint grippers have the same (dis)advantages as parallel-jaw grippers.



Figure 5.6: Examples of rotational joint grippers: a GearWurx gripper (left) [60] and a Robotiq gripper (right) [9].

Three-finger Gripper

Grippers equipped with three or more fingers are commonly referred to "multi-fingered grippers" [135]. Some examples are shown in Figure 5.7. Their main advantage compared to other grippers is the increase in dexterity. This increase is especially useful in handling arm singularities, obstacles, and clutter [85]. The three types of clutter commonly referred to in literature are illustrated in Figure 5.8. Additionally, the added degrees of freedom allow the robot to change its grasp pose without moving the arm, thus improving speed, energy consumption, and safety [85].

Despite their advantages, multi-fingered grippers are challenging to apply in practice. Because of their increased number of degrees of freedom, the overall grip strength is lower than that of parallel-jaw and two-finger grippers, and it is difficult to include high-quality, robust sensors for appropriate feedback [85]. Most

recent approaches in literature using multi-fingered hands have underwhelming results, as they either use a small set of test items, reach a low success rate, or take well over 5 seconds to find a grasp pose [135]. This is mainly due to the many degrees of freedom, which make it so that blindly searching the solution space for good grasps is infeasible [135]. Machine learning also faces problems due to the large solution space and the lack of available training data. Additionally, multi-fingered grippers are more expensive [76] and their small components result in more frequent failures [85] than is the case with previously discussed grippers. Finally, three-fingered grippers are more bulky than two-finger or parallel-jaw grippers.

However, it should be noted that the weight of some of these disadvantages depends heavily on how a grasp pose is determined. For example, some three-fingered grippers limit their grasps to be symmetric (that is, the angles between the fingers are 120°). This significantly simplifies the grasping algorithm, but also negates most of the gripper's dexterity.



Figure 5.7: Examples of three-fingered grippers: a Robotiq adaptive gripper (left) [12] and a Barrett hand (right) [1].



(a) Isolation: only one item is present in the robot's area of interest.

(b) Sparse clutter: multiple items are present, but they do not touch or overlap.

(c) Dense clutter: multiple items are present and they touch and/or overlap.

Figure 5.8: The three types of clutter commonly referred to in literature.

Dexterous Hand

A special case of multi-fingered grippers is those with four or five fingers, often referred to as "dexterous hands" or "human-like hands". Figure 5.9 shows some examples. Their advantages and disadvantages are more extreme versions of those of the three-fingered grippers [85]. However, their likeness to human hands comes with an additional advantage: it allows the robot to learn from human demonstrations [135]. Human grasping is a widely researched area which has resulted in (amongst others) grasping taxonomies [135]. These taxonomies can be used as a basis for determining a dexterous hand's grasp pose, thus removing a large part of the problems caused by its many degrees of freedom. Additionally, since these hands are based on the human hand, and human operators can handle all items that appear in FM-GtP, a dexterous hand can theoretically handle all items with the proper gentleness.

However, in practice the dexterous hands are not as successful as they are in theory. This is due to the same reasons as for three-fingered hands. An additional challenge is that for learning from human grasps, the system requires an algorithm to select a grasp from the taxonomy based on the presented item. In the case of machine learning, this requires a significant amount of training data that is not readily available.

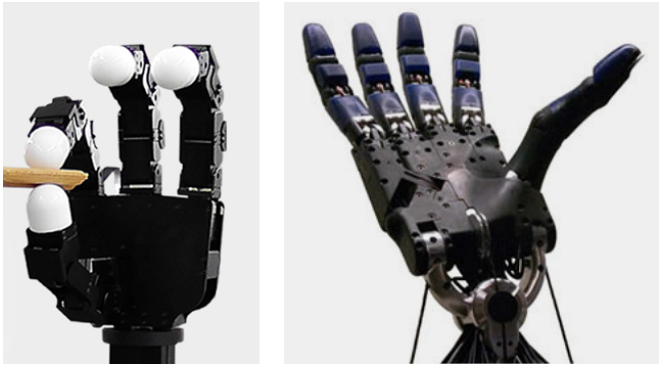


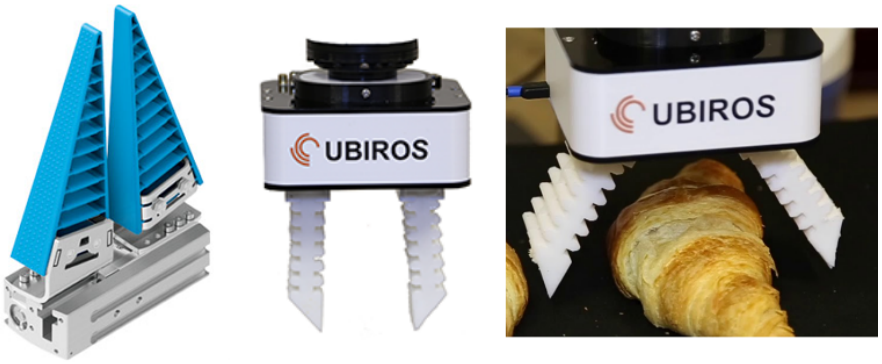
Figure 5.9: Examples of dexterous hands: the Allegro hand (left) [111] and the Shadow dexterous hand (right) [110].

Soft Gripper

Another type of gripper is the soft gripper. These grippers achieve high adaptiveness and safe cooperation with humans because they are made of materials that are soft, flexible and compliant [58]. Most of them are actuated pneumatically or through cables [58], although electric actuation also exists [133]. Pneumatic actuation would simplify the switching mechanism, as all grippers used by RSIR would then be pneumatic. Soft grippers can roughly be split into fingered grippers and non-fingered grippers, shown in Figures 5.10a and 5.10b, respectively. Both types feature several advantages: they are relatively simple in design and actuation [76], they have an adaptable manufacturing process [58], their compliance provides some self-solving abilities that can make up for limited accuracy of the grasp synthesis algorithm [58][76], and the softness allows them to handle delicate items [58]. Additionally, they can work with a large range of item dimensions and can have a good lifetime [76].

It should be noted that there is a continuous spectrum between soft and rigid grippers [58]. A gripper with soft paddings is more soft than a purely aluminium gripper, but more rigid than a pneumatic silicon gripper. A gripper's softness is a trade-off between the number of degrees of freedom and its precision, and also between the force it can exert and its compliance [58]. Hence soft grippers typically have a lower payload and less precision than rigid grippers. Other disadvantageous are the difficulty in modelling soft grippers, the complex integration of sensors, and their lack of repeatability [58]. Additionally, they can be bulky in practice. For example, Figure 5.10a shows how wide the Ubiros gripper opens. Navigating such a large gripper is challenging, especially in clutter, and it makes the gripper less dexterous. Rigid fingered grippers do not face this problem because nearly all of them can be put in any position (such as half open). However, since soft grippers are often pneumatic, most of them only have two positions: open and closed [58].

Fingered soft grippers are soft grippers with elements that can be considered fingers. These grippers combine the (dis)advantages of two-fingered grippers with those of soft grippers. On the other hand, non-fingered grippers are unlike any of the fingered grippers. Although such technologies are interesting, a recent survey concluded that it is still a widely unexplored research area, and that the state of the art is not sufficient to be applied in robotics [58]. A theoretical study by Vanderlande also concluded that some soft grippers are not applicable [76]. Hence the remainder of this report considers only fingered soft grippers.



(a) Examples of fingered soft grippers: the Festo parallel long-stroke gripper (left) [27] and the Ubiros Gentle Duo gripper (centre, right)[119].



(b) Examples of non-fingered soft grippers: the Festo Octopus Gripper (left) [70] and the Empire Robotics Versaball gripper (right) [141].

Combi-Gripper

A final category of grippers is that of the combi-grippers. This type of gripper combines a suction cup with fingers, an example of which is shown in Figure 5.11. The gripper can use only the suction cup, only the fingers, or a combination of both. A combination of both generally consists of the robot placing the suction cup onto the item, and then closing its fingers around the item [52]. Thus the combi-gripper theoretically combines the best of vacuum grippers and fingered grippers. An important advantage is that the use of such a gripper can eliminate the need for a device that switches between grippers. Additionally, this a combi-gripper was shown to be able to grasp more items than only a suction cup [52]. The fingers closing around the item make the grasp more robust than if only the suction cup were used.

However, it is unclear what exactly the advantages of the combi-gripper are regarding the diversity of items it can handle compared to a system that switches between a fingered gripper and a vacuum gripper. The multi-gripper provides the most advantages when it uses both its suction cup and its fingers, as this makes the grasp more robust. However, only 0.63% of SIR grasps fail due to a lack in robustness and the item falling out of the gripper [36]. On the other hand, 7.54% fails because the vacuum gripper cannot handle the item [36], in which case the multi-gripper performs no better than a fingered gripper.

Additionally, the multi-finger has the disadvantage that it is more bulky than a vacuum or fingered gripper (with the same number of fingers). Finally, the envisioned future for RSIR might include several vacuum grippers to handle a larger variety of items than is possible with only one gripper. In that case, the use of a combi-gripper (with one suction cup) does not eliminate the need for a switching mechanism, and it is likely equally effective to use separate vacuum and fingered grippers instead.

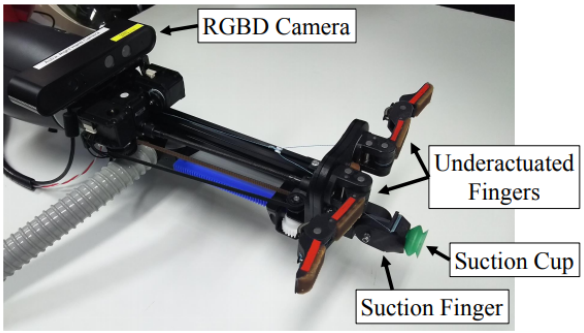


Figure 5.11: Example of a combi-gripper, used in the research by Hasegawa et al. [52].

5.3.2. Selecting the Best Technology

This subsection selects the best gripper technology for RSIR. It does so by giving a weighed score for each gripper technology per requirement. Weights and scores range from 1 (unimportant, low score) to 5 (most important, high score). Table 5.2 shows the scores. The weights and scores are determined as follows:

1. **Item variety** is split up into the five properties often used for grasping: shape, size, mass, texture, and deformability [135], as well as the level of clutter. Because vacuum grippers were found to perform poorly for textured (porous), deformable, and complex-shaped items, these properties get a higher weight. Less bulky grippers score higher for clutter.

2. **Computational cost** gets a high weight because speed is one of the main goals of RSIR. Although the computational cost mainly depends on the grasp synthesis algorithm, there is a general trend for the cost per gripper type. Grippers with synthesis approaches that generally have a lower computational costs score higher.

3. **Repeatability** gets a mid-level weight because better repeatability allows for better consistency and reasoning. Grippers that consistently achieve grasps of similar quality get a higher score.

4. **Lifetime** gets a mid-level weight because RSIR must stay operational. Note that simpler grippers generally have a longer lifetime and thus get a higher score.

5. **Cost** gets a mid-level score because a great gripper justifies a higher cost only to a certain extent.

6. **Modularity** gets a low weight because the switching mechanism is still under development and might be adapted to a new type of gripper. Grippers with less physical connections score higher because less cables must be switched. Additionally, soft grippers score high because most of them are pneumatic and thus compatible with the current switching system.

7. **Gentleness** gets a low score because all fingered grippers can theoretically gently grasp any item as long as the appropriate grasping force is used (done by Pijnacker, for example [103]). However, it is still easier if the gripper has some passive compliance that prevents the item from being damaged. Thus the less a gripper relies on friction (and thus normal force) to grasp, the higher it scores.

8. **Dexterity** gets the lowest weight because most tasks do not require a high level of dexterity, or the lack of dexterity in the gripper can be compensated by dexterity in the arm. Although dexterity can be useful in reducing execution time and energy consumption, this requires more advanced control and is left as potential future work.

9. **Availability** scores high because of the limited time available for this thesis. Grippers that are available at Vanderlande score high, whereas others score low. Note that availability is almost irrelevant when viewing RSIR outside the scope of this thesis, as it may well be worth the time and money for Vanderlande to investigate expensive grippers. Hence Table 5.2 also lists the total score per gripper without considering availability.

Table 5.2: Ranking the potential gripper technologies for RSIR. The total score is calculated as the sum of the products between the weight and the score. The total score is listed twice: once including and once excluding the scores for Availability. Total scores are highlighted to improve clarity.

		Weight	Parallel-jaw	Rotational joint	Three-finger	Dexterous hand	Soft finger	Combi
Item variety	Shape	2	3	4	5	5	4	1
	Size	1	4	3	4	3	2	5
	Mass	1	5	5	3	2	1	5
	Texture	2	5	5	5	5	5	3
	Deformability	2	4	4	5	5	4	2
	Clutter	2	5	5	4	3	2	3
	Computational cost	4	5	5	3	1	5	4
	Repeatability	3	4	3	3	2	2	4
	Lifetime	3	5	4	3	1	3	3
	Cost	3	5	5	2	1	5	3
	Modularity	2	4	4	3	1	5	3
	Gentleness	2	1	2	3	3	5	4
	Dexterity	1	2	2	4	5	1	1
	Availability	5	5	1	5	1	5	1
	Total (without Availability)		*117*	*114*	*97*	*70*	*104*	*89*
	Total (with Availability)		*142*	*119*	*122*	*75*	*129*	*94*

As shown in Table 5.2, the parallel-jaw gripper has the highest score. As a result, a parallel-jaw gripper is used in the remainder of this thesis. The next section considers synthesis approaches for this type of gripper. Note that outside the context of this thesis, the most promising grippers are the parallel-jaw and rotational joint grippers. The rotational joint gripper is not considered in the remainder of this thesis because it is not available (R20). However, it may be a relevant gripper for future work.

5.4. Synthesis Approach

In the previous section, the parallel-jaw gripper was selected to be used in combination with the vacuum gripper on RSIR. The next step in implementing a parallel-jaw gripper is to define an algorithm that computes a grasp pose when presented with an item. This section presents a literature study into such algorithms for the two-fingered gripper, and selects the best one to be used in this thesis.

A brief recap of the literature study accompanying this thesis is presented [135]. Grasp synthesis refers to the problem of finding a grasp configuration that satisfies a set of criteria relevant for the task. The main challenges in grasping are the diversity in objects, tasks, and grippers, as well as the complex physics involved. A metric is a measure of how good a grasp is. Both metrics and synthesis approaches can be analytical (based on object models) or data-driven (based on previous experiences and often machine learning). The main advantages of analytical approaches is that they are easy to design, but due to the assumptions they make, they often do not work well in practice. Approaches that represent an object by a primitive geometry (for example, its principal axis) are generally faster than those that represent it as its full geometry. Data-driven approaches generally achieve better performance, but require training data and generalize poorly across grippers and tasks.

5.4.1. Available Approaches

Since the literature study focussed on grippers with at least three fingers [135], it does not provide a sufficient overview on grasp synthesis for parallel-jaw grippers. Hence this subsection presents a literature study into grasp synthesis approaches for parallel-jaw grippers. Note that most of these approaches would also work with a rotational joint gripper or a soft (two-)fingered gripper.

Analytical Approaches

First the analytical synthesis approaches are considered. These are split up into approaches that synthesize grasps based on an object's primitive geometry (such as primitive axis), and those that synthesize a grasp based on the object's surface.

Primitive Geometry

The first type of analytical synthesis approaches is based on an object's primitive geometry. Suzuki et al. model an object based on its principal axis, which they obtain from analysing the point cloud [124]. The gripper positions itself above the object's centroid, rotates to be perpendicular to its principal axis, lowers

until it touches the plane, and closes. Figure 5.12 illustrates the gripper orientation. This simple approach achieves a 86% success rate with an inexpensive depth camera. Failures are likely due to the object shape (e.g. spherical objects) and weight (e.g. a hammer). It takes 4.2 seconds for the algorithm to output a grasp pose, although this runtime includes segmenting the scene.

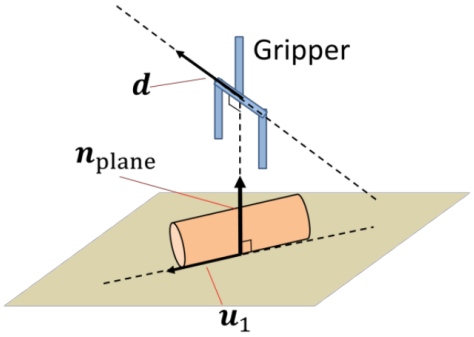


Figure 5.12: Visualization of the grasp synthesis approach by Suzuki et al. [124]. The gripper approaches the object normal to the plane and perpendicular to the object's principal axis.

Stuckler et al. also use an object's principal axis, but consider its bounding box as well [121]. For grasping the object from the side, its silhouette is obtained and simplified into a bounding box. Grasps are sampled in an ellipse around this bounding box. Top grasps are sampled along both principal axes through the centre of the bounding box. The resulting set of grasps is filtered based on collisions and reachability, and ranked based on four properties. These properties are: their distance to the object centre, grasp width, grasp orientation, and distance from the robot. The latter two are due to the robot arm geometry. A 99% success rate is achieved for a limited set of objects while requiring only 0.16 seconds to synthesize a grasp. An advantage of this approach is that it automatically determines whether a side or top grasp is more appropriate. Figure 5.13 shows some successful grasps. The authors assume an item's centre of mass coincides with its centroid, but the algorithm also performs well for (some) items that violate this assumption.

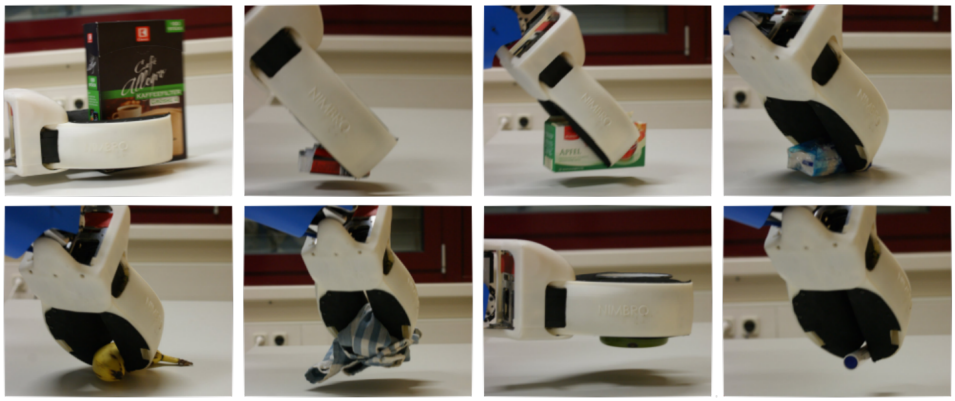


Figure 5.13: Examples of successful grasps by Stuckler et al. [121]. Note that the "top" grasps are not normal to the plane.

As opposed to Stuckler et al., Stoyanov et al. [120] consider collisions before sampling grasps. They argue that pre-computed grasps are often infeasible at runtime due to collisions with the environment, especially in clutter. They employ a constraint-based grasp formulation, which has as an additional advantage that grasp synthesis can then be integrated with the motion planning step. A primitive grasp envelope is defined from heuristics, from which grasps in collision with the environment are removed (Figure 5.14). The approach achieves an 84% success rate in sparse clutter. Computing the grasp envelopes takes 0.06 seconds, but this relies on an efficient model of the environment. Creating this model requires the depth camera to move around the scene, and takes 40 seconds for the scene considered by the authors.

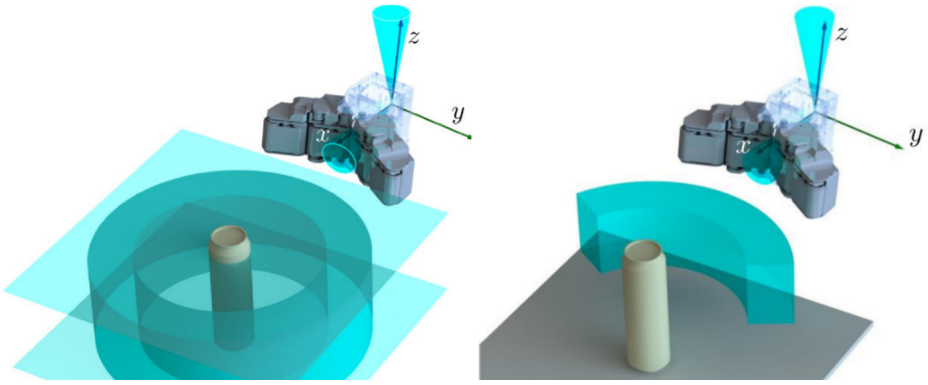


Figure 5.14: The constraint-based grasp formulation by Stoyanov et al. [120]: the primitive grasp envelope (left) and the resulting grasp envelope after removing grasps that are in collision with the environment (right).

Surface

The previous three approaches arguably work well, but the question remains how well their primitive object representation works for any given object. Some approaches avoid this simplification step and directly synthesize grasps based on the object's geometry. The approach by Baumgartl and Henrich synthesizes grasps based on pairs of parallel lines, obtained from the object's edges [17]. The pairs are ranked based on the line lengths and whether the object fits between the jaws of the gripper. Examples of grasps are presented in Figure 5.15. This simple approach achieves a success rate of 79% with grasp synthesis taking only 0.03 seconds. Note that the approach assumes that the objects have already been segmented.

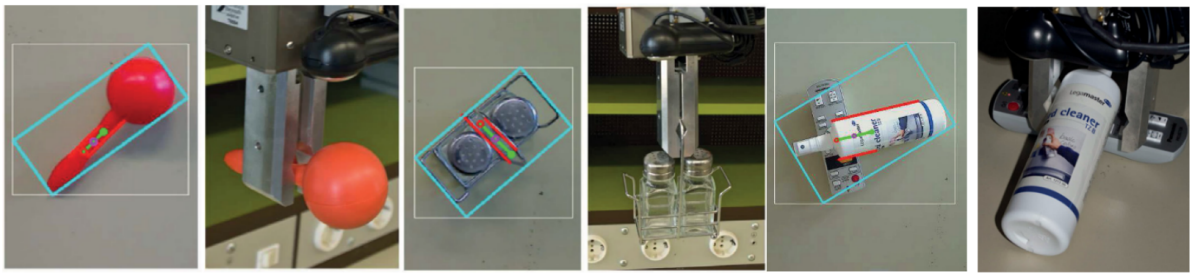


Figure 5.15: Examples of successful grasps by Baumgartl and Henrich [17]. The object's bounding box is indicated in blue, the edge segments in red, and the centre of the grasp pose in green.

Similarly to Baumgartl and Henrich, the approach by Pas et al. finds grasps by exploiting parallel surfaces [129]. It relies fully on sampling, rather than using, for example, an edge detector. To keep this feasible, it uses geometry to reduce the size of the sample space. The space is thus defined as grasps that are collision-free, have part of the item between the fingers, and have the item's principal curvature matching the direction of the hand. The latter is a heuristic that discards otherwise good grasps, but it speeds up the synthesis process by looking only at grasps that are likely to be successful. The resulting sample space contains tens to hundreds of grasps, after which it is reduced by clustering similar grasps. Grasps that have no inverse kinematic solution are discarded, and the remainder is ranked based on how easy the robot can reach them. Advantages of this approach are the simple interpretation, that it requires only the point cloud and gripper geometry, and its success rate of 73% for unseen items. Additionally, the clustering allows to select a region of interest, thus allowing the robot to grasp a specific part of an item. Finally, the ROS code is available. Disadvantages are the complex mathematics involved, the drop of success rate for dense clutter, and that no runtimes are reported. The full approach also includes machine learning, which is discussed later in this section.

Klingbeil et al. [64] take a more complex approach, using the basic idea that a good grasp is when the object shape matches the gripper shape. A 2D slice of the depth map is taken, in which shapes similar to the gripper's geometry are found (Figure 5.16). This process is repeated for multiple slices and rotations of the depth map. Although the approach achieves a 90% success rate for objects in dense clutter, it is very computationally expensive. The authors consider only few rotations to keep the approach feasible. Even so, the time from when the object is presented until it is picked up is 83 seconds on average. This number is not listed

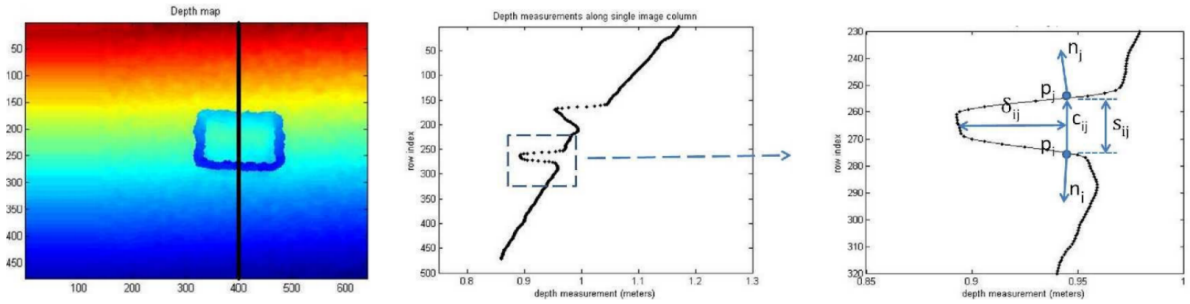


Figure 5.16: Visualisation of the approach by Klingbeil et al. [64]. A 2D slice of the depth map (left) is taken, here defined as a column. The resulting image (middle) is inspected for shapes similar to the gripper's shape (right).

in the paper, but estimated from the accompanying video. It is not clear at which point during this process the synthesis is complete, but considering that the pick-and-place cycle at Vanderlande is approximately 9 seconds [127], it is likely that the majority of the 83 seconds runtime stems from grasp synthesis.

Finally, Ala et al. present an approach based on boundary and convex segments, called grasplets (see Figure 5.17) [11]. The grasplets are obtained from the object's point cloud, and a pair of grasplets forms a grasp. Feasible grasps are obtained from analysing pairs of grasplets, realizing that the grasplets need to be opposite each other for a parallel-jaw gripper to successfully grasp the object. An advantage of this approach is that specific grasp areas can be specified, for example only the top part of an object. The authors show an example of grasping the handle of a mug, when provided with an algorithm that detects the handle in the image. Overall, this approach is a more sophisticated but also more complex variant of the simple edge-detection approach by Baumgartl and Henrich. It achieves an 88% success rate on isolated objects, requiring 12 seconds to synthesize a grasp.

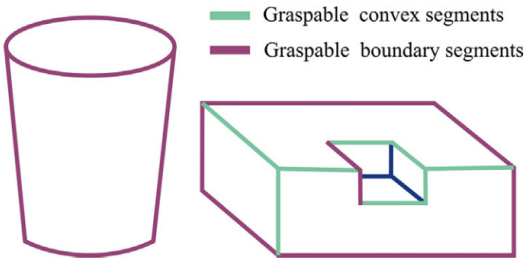


Figure 5.17: Boundary and convex segments as defined in the approach by Ala et al. [11]

Data-driven Approaches

Data-driven approaches generally achieve better performance than analytical approaches. The remainder of this section thus considers data-driven approaches. Most of these were already considered in the literature survey corresponding to this thesis [135], but they are included here for completeness.

Learning Evaluation Metrics

Approaches that learn evaluation metrics work as illustrated in Figure 5.18: they require sensory data and a grasp pose as input, and output the probability of the grasp pose resulting in a successful grasp. Grasps are sampled and the grasp resulting in the highest probability is selected.

Mahler er al. use this type of approach with a convolutional neural network to predict grasp success given a depth image and a gripper pose [86]. When presented with an image, the system samples grasps through the cross-entropy method (CEM) to find a grasp with a high predicted success. The network is trained on Dex-Net 2.0: a synthetic dataset of 6.7 million data points created in simulation. The trained network achieves a 94% success rate on a physical robot for novel objects, requiring 2.5 seconds to synthesize a grasp. Figure 5.19 shows some planned grasps. CEM is the main cause for the long runtime, as the approach only requires 0.8 seconds without it, but then the success rate also drops to 80%. The approach works well even for some

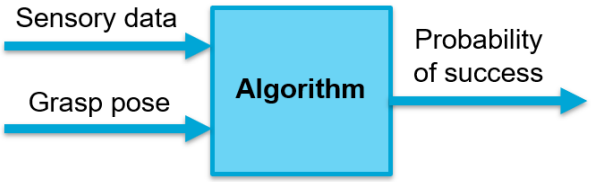


Figure 5.18: The general inputs and output of an algorithm that learns an evaluation metric.

smooth or deformable objects. It only grasps objects from the top, which might be a limitation for some applications.

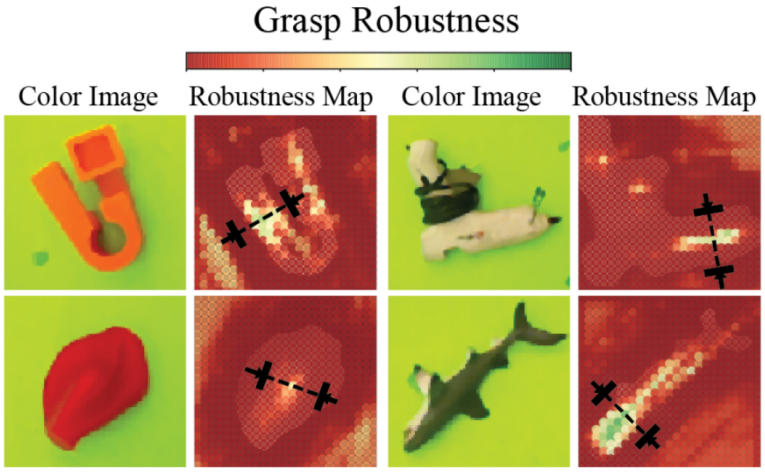


Figure 5.19: Examples of color images and the grasp success predicted by the Dex-Net 2.0 neural network in the approach by Mahler et al. [86]. The planned grasp using the CEM approach is shown in black.

Pas et al. extended their previously mentioned approach by including a Support Vector Machine (SVM) to predict whether a grasp is successful or not. This reduces the sample space, but otherwise the algorithm remains the same. To obtain training data, they gather depth data from two directions for 36 items, and automatically label it. The labelling is done based on antipodal grasps. The interpretation of such as grasp is that the robot can hold the object by applying sufficiently large forces along the line connecting the two contact points. Whether a grasp is antipodal can be derived from the gripper contact points and friction. The SVM improves the success rate from 73% to 88% for isolated items, and to 73% for dense clutter. The advantage of the SVM is the increase of improvement, though it should be noted that without the SVM, the approach still achieves a remarkable success rate and saves the time otherwise used for data acquisition and training.

Learning Optimal Grasp Poses

The previous data-driven approaches have as a disadvantage that they need to sample grasp pose, which is usually computationally expensive and thus time-consuming. To avoid this, some approaches apply an algorithm that directly transforms an input image to a grasp pose, as shown in Figure 5.20.

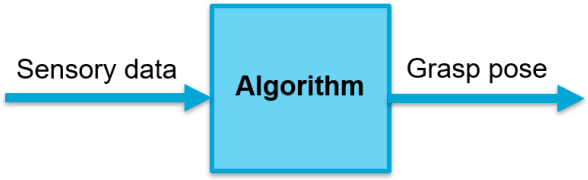


Figure 5.20: The general inputs and output of an algorithm that learns an optimal grasp pose.

Pinto and Gupta select 18 grasp poses (defined by their angle with respect to the object), and use a neural

network to determine which of those poses is most likely to result in a successful grasp [104]. They point out the problems with human-annotated data, which is incomplete and biased by semantics. Instead, their approach obtains training data from trial and error during 700 hours. The top 10 grasps are re-ranked based on how successful nearby data points are, to account for actuator inaccuracies. The approach achieves a modest 66% success rate for unseen objects in dense clutter, and 73% for seen objects in different conditions. This approach considers only grasps from the top.

Instead of predicting grasp success for a given grasp pose, the neural network by Morrison et al. [92] directly outputs a grasp pose and quality measure for every pixel in an input image. This avoids the computationally expensive CEM. The network is trained on an adjusted version of the Cornell dataset, and its output is filtered to remove outliers. The system achieves a success rate of 92% on isolated objects. However, the main contribution of this work is that the synthesis takes only 19 milliseconds, allowing the authors to create a closed-loop system. This system continuously observes the environment and computes the optimal grasp pose ("visual servoing"), allowing it to handle dynamic scenes. For example, it has an 81% success rate for grasping dynamic objects in clutter. A big disadvantage, however, is that the system requires a camera mounted on the robot's wrist. Otherwise, the bulky robot arm can occlude the objects and the system cannot find a grasp pose.

Learning Optimal Motions

Previous approaches, with the exception of Morrison et al., all relied on the sense-plan-act pipeline: sense the environment, synthesize a grasp, and execute the grasp. However, as Morrison et al. showed, continuous feedback makes an approach more robust against disturbances. One approach for this is to learn optimal motions, that is, have an algorithm that directly outputs the motion control given the sensory data. Levine et al. do just this [75]. They use a convolutional neural network and an overhead camera that observes both the objects and the robot arm. The system achieves a 90% success rate for picking the first 10 objects out of a bin, including smooth and deformable objects. An advantage is the system's robustness to differences in the scene, such as poor camera calibration and gripper wear. However, to achieve the impressive performance, the model was trained for two months, using over 800,000 grasp attempts and up to 14 robots (see Figure 5.21).

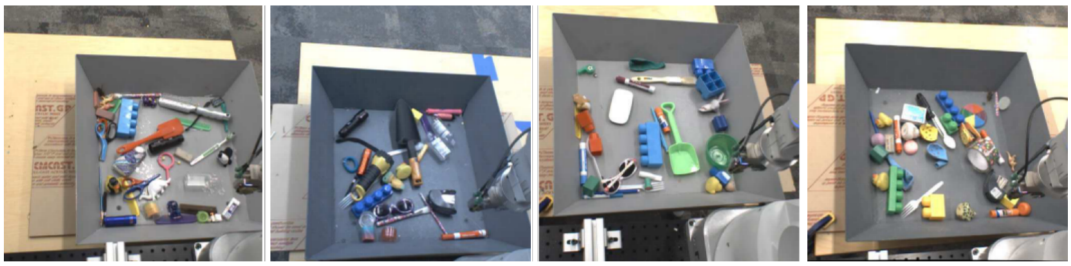


Figure 5.21: Four of the robots used by Levine et al. during training, adapted from [75]. All four robots are in the same joint configuration.

Kalashnikov et al. continue the work by Levine et al. and focus on off-policy training to decrease the required training data by one third [61]. For the same bin emptying task, the approach by Levine et al. has a 76% success rate and the approach by Kalashnikov et al. 88%. The improved approach also shows corrective behaviours, such as pregrasping motions to handle otherwise ungraspable objects (see Figure 5.22). Despite the improvement, the approach still needs a large amount of training data.

On the other hand, Yan et al. criticise deep learning approaches that predict grasp success from an image and a grasp pose, as the additional optimization step scales exponentially with the action space during runtime [144]. Their neural network architecture directly learns the distribution of good grasp poses from self-supervised grasping trials. This architecture keeps a constant inference time for increasing action spaces, at the cost of requiring more training data. Note that a shorter inference time does not directly translate to faster picking. Instead, it allows the feedback loop to run at a higher frequency. Whether the picking also becomes faster depends on the hardware and software constraints, such as safety limits on joint velocities. The architecture shows an increase in average grasp success up to 3 million data points. It achieves an 83% success rate on a physical robot for unseen objects in sparse clutter. Although the constant inference time is promising, the increased need for training data is a big disadvantage.

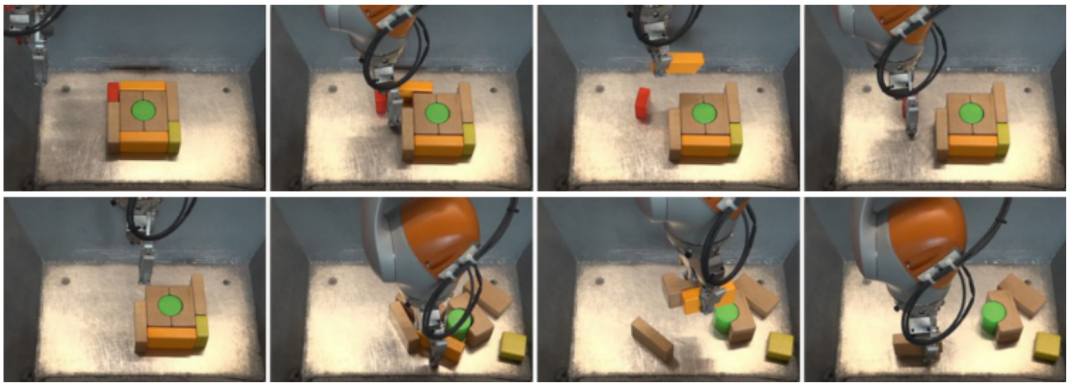


Figure 5.22: An example of pregrasping motions by the approach by Kalashnikov et al. [61].

Berscheid et al. combine grasping with shifting to improve grasping success rate for items in bins and clutter. Shifting refers to pressing the gripper on the item and moving it parallel the supporting surface. Thus the item is moved to a less cluttered location, allowing the robot to grasp it. The deep reinforcement learning approach revolves around 5 motion primitives: 3 that define the gripper's location and angle for three different opening widths; and 2 for shifting objects along the gripper's two axes. Figure 5.23 shows the basic control scheme. The robot decides whether to shift or not by comparing the predicted grasp success before and after shifting. This is more efficient than pure reinforcement learning. After training on blocks and cylinders for 110 hours (100 for grasping and 10 for shifting), the robot can completely empty densely cluttered bins at 274 picks per hour. For other items such as toothpaste and markers, the approach achieves a success rate of 92% when the items are isolated. Advantages of this approach are the inclusion of shifting, the high success rate, good runtime, and the availability of the source code and trained model. However, a big question is how well the system performs for densely cluttered items that are not blocks or cylinders. Additionally, it is likely that the system requires some level of retraining when applied on a different set-up.

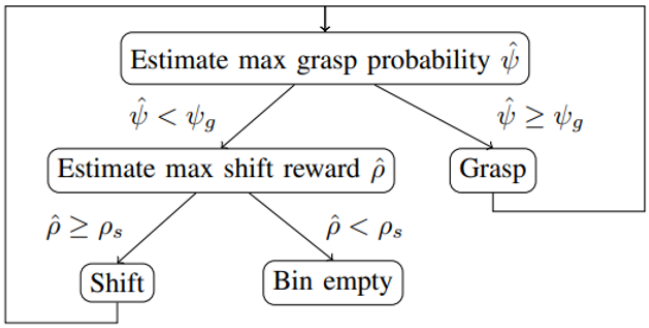


Figure 5.23: The basic control scheme in the approach by Berscheid et al. [22]. If the predicted success for a grasp is sufficiently high, the robot grasps; if not, the robot checks if shifting results in better grasps. If this is the case, the robot shifts and tries to grasp again; if not, the robot concludes the bin is empty.

5.4.2. Summary of Approaches

Table 5.3 summarizes the parallel-jaw synthesis approaches considered in this thesis. Furthermore, Table 5.4 presents a complete summary of the synthesis approaches considered in the thesis and the corresponding literature study. This table might be useful for future work. In summary, analytical and data-driven approaches have similar success rates for parallel-jaw grippers. The data-driven approaches can handle a larger variety of items, but require training and do not generalize well across tasks.

Table 5.3: Summary of grasp synthesis approaches considered in this thesis that are applicable to a parallel-jaw gripper. "-" = no data available. [a]Estimated from the video accompanying the paper. [b]Approach requires the camera to be mounted on the robot's wrist. [c]For the first 10 objects in dense clutter, without replacement. [d]For the first 10 grasping attempts, estimated from the video accompanying the paper by Kalashnikov et al. [61].

					Required Properties					Object Variety					Success Rate				Runtime		Grasp Type			
		Literature	Fingers	Clutter	RGB Image	Point Cloud	Geometry	Mass Centre	Friction	Shape	Size	Mass	Texture	Deformability	Rate (%)	Testing Environment	Compliance	Tactile Feedback	Time (s)	Process	Object Area	Binary Type	Preshape	Required Training Data
Analytical	Primitive Geometry	Suzuki et al. [124]	2	None		X				X	X	X	X		86	Physical	None	No	4.2	Synthesis				
		Stuckler et al. [121]	2	Sparse	X	X				X	X		X	X	99	Physical	Active	No	0.16	Synthesis	X			
		Stoyanov et al. [120]	2	Sparse			X			X	X		X	X	84	Physical	Passive	No	0.06	Synthesis	X			
				Sparse		X				X	X		X	X	84	Physical	Passive	No	40	Picking	X			
	Surface	Baumgartl et al. [17]	2	Sparse	X					X	X	X	X		79	Physical	None	No	0.03	Synthesis	X			
		Pas et al. [129]	2	None		X				X	X		X	X	73	Physical	None	No	-	-	X			
		Klingbeil et al. [64]	2	None		X				X	X				98	Physical	None	No	-	-				
				Sparse		X				X	X				95	Physical	None	No	-	-				
				Dense		X				X	X				90	Physical	None	No	83^a	Picking				
		Ala et al. [11]	2	None		X				X	X		X		88	Physical	None	No	12	Synthesis	X	X		
Data-driven	Learning Evaluation Metrics	Mahler et al. [86]	2	None		X				X	X	X			80	Physical	None	No	0.8	Synthesis				+++
				None		X				X	X	X	X	X	94	Physical	None	No	2.5	Synthesis				+++
		Pas et al. [129]	2	None		X				X	X		X	X	88	Physical	None	No	-	-	X			++
				Dense		X				X	X		X	X	73	Physical	None	No	-	-	X			++
	Learning Optimal Grasps	Pinto et al. [104]	2	None	X					X	X	X			76	Physical	None	No	-	-				+++
				Dense	X					X	X	X			66	Physical	None	No	-	-				+++
		Morrison et al. [92]	2	None		X^b				X	X	X	X	X	92	Physical	Passive	No	0.02	Synthesis	X			++
				Dense		X				X	X	X	X	X	81	Physical	Passive	No	-	-	X			++
	Learning Optimal Motions	Levine et al. [75]	2	None	X					X	X	X	X	X	80	Physical	Passive	No	-	-				+++
				Dense	X					X	X	X	X	X	90^c	Physical	Passive	No	28^d	Grasp Cycle				+++
		Kalashnikov et al. [61]	2	None	X					X	X	X	X	X	96	Physical	Passive	No	-	-				+++
				Dense	X					X	X	X	X	X	88^c	Physical	Passive	No	22^d	Grasp Cycle				+++
		Yan et al. [144]	2	Sparse	X					X	X	X	X	X	83	Physical	Passive	No	-	-				+++
		Berscheid et al. [22]	2	Dense		X				X	X	X	X	X	92	Physical	None	No	13	Pick-and-place				++

Table 5.4: Summary of grasp synthesis approaches considered in this survey. N/A = not applicable, "-" = no data available. [a]Successful grasps were created for all presented objects, but no success rate is reported. [b]Approach requires images from multiple views. [c]Success rate when selecting the correct preshape. [d]Approach requires the object to be represented by shape primitives. [e]Property is not required for creating grasp candidates, but might be required to calculate the evaluation metric(s). [f]For perfectly predicting the grasp stability on a 5-level scale. [g]For the first 10 objects in dense clutter, without replacement. [h]For the first 10 grasping attempts, estimated from the video accompanying the paper by Kalashnikov et al. [61]. [bb]Approach requires the camera to be mounted on the robot's wrist. [cc]Estimated from the video accompanying the paper.

					Required Properties					Object Variety					Success Rate				Runtime		Grasp Type			
		Literature	Fingers	Clutter	RGB Image	Point Cloud	Geometry	Mass Centre	Friction	Shape	Size	Mass	Texture	Deformability	Rate (%)	Testing Environment	Compliance	Tactile Feedback	Time (s)	Process	Object Area	Binary Type	Preshape	Required Training Data
Analytical	Primitive Geometry	Lei et al. [73]	3	None		X				X					-[a]	Physical	Passive	No	<1	Synthesis	X			
		Maldonado et al. [89]	4	Sparse		X				X	X	X			80	Physical	Active	No	-	-				
		Suzuki et al. [124]	2	None		X				X	X	X	X		86	Physical	None	No	4.2	Synthesis				
		Eppner and Brock [39]	3	None		X				X	X	X	X	X	62[c]	Physical	Passive	No	-	-	X	X		
		Miller et al. [90]	3	Dense			X[d]	X[e]	X[e]	X	X				-[a]	Simulation	N/A	N/A	262	Synthesize List	X	X		
		Stuckler et al. [121]	2	Sparse	X	X				X	X		X	X	99	Physical	Active	No	0.16	Synthesis	X			
		Stoyanov et al. [120]	2	Sparse			X			X	X		X	X	84	Physical	Passive	No	0.06	Synthesis	X			
				Sparse		X				X	X		X	X	84	Physical	Passive	No	40	Picking	X			
	Surface	Roa and Suárez [108]	≥2	None			X	X	X	X	X				-[a]	-	-	-	182	Synthesis	X			
		Lippiello et al. [79]	2-5	None			X	X	X	X	X				-[a]	Simulation	N/A	N/A	<1	Synthesis	X			
		Hang et al. [50]	3	None		X		X	X	X	X				90	Simulation	N/A	N/A	21	Synthesis	X			
		Lippiello et al. [80]	≥3	None	X[b]			X		X					-[a]	Physical	None	No	1.5	Object Modelling				
		Baumgartl et al. [17]	2	Sparse	X					X	X	X	X		79	Physical	None	No	0.03	Synthesis	X			
		Pas et al. [129]	2	None		X				X	X		X	X	73	Physical	None	No	-	-	X			
		Klingbeil et al. [64]	2	None		X				X	X				98	Physical	None	No	-	-				
				Sparse		X				X	X				95	Physical	None	No	-	-				
				Dense		X				X	X				90	Physical	None	No	83[c][cc]	Picking				
		Ala et al. [11]	2	None		X				X	X		X		88	Physical	None	No	12	Synthesis	X	X		
Data-driven	Human Demonstration	Brahmbhatt et al. [25]	≥3	None			X			X	X				-[a]	Simulation	N/A	N/A	-	-		X	X	+++
		Liu et al. [81]	3	None		X				X	X	X			90	Physical	None	No	4	Synthesis		X	X	++
		Hillenbrand and Roa [56]	≥2	None			X				X				85	Simulation	N/A	N/A	-	-		X	X	++
		Kopicki et al. [67]	4,5	None		X				X	X	X	X		86	Physical	Active	No	>5	Synthesis	X	X	X	+
		Kopicki et al. [68]	5	None		X				X	X	X	X		78	Physical	Active	No	23	Synthesis	X	X	X	+
		Detry et al. [34]	3	Sparse		X				X	X	X			84	Physical	Active	No	90	Synthesis	X	X	X	++
	Reasoning	Vezzani et al. [138]	5	None		X				X	X	X	X	X	83	Physical	Active	Yes	-	-	X			++
		Lu and Hermans [82]	4	None		X				X					98	Physical	None	No	-	-		X	X	++
	Learning Evaluation Metrics	Hang et al. [51]	3	None		X		X	X	X	X	X			90	Physical	None	No	5-7	Synthesis	X			++
		Morales et al. [91]	3	Sparse			X			X					51[f]	Physical	None	No	0.2	Synthesize List	X			++
		Mahler et al. [86]	2	None		X				X	X	X			80	Physical	None	No	0.8	Synthesis				+++
				None		X				X	X	X	X	X	94	Physical	None	No	2.5	Synthesis				+++
		Zeng et al. [145]	2	Dense	X	X									-	-	-	-	0.8	Synthesis				+++
		Lu et al. [83]	4	None	X	X				X					87	Physical	None	No	2-3	Synthesis		X	X	+++
		Saxena et al. [115]	3	None	X	X				X	X	X	X		76	Physical	None	No	-	-	X			-
				Dense	X	X				X	X	X	X		75	Physical	None	No	-	-	X			-
		Pas et al. [129]	2	None		X				X	X		X	X	88	Physical	None	No	-	-	X			++
				Dense		X				X	X		X	X	73	Physical	None	No	-	-	X			++
	Learning Optimal Grasp Poses	Song et al. [118]	4	None	X	X				X	X		X		88	Physical	None	No	-	-				+++
		Choi et al. [29]	4	None		X				X	X	X	X		87	Physical	Passive	No	-	-				+++
		Schmidt et al. [116]	5	None		X				X	X				55	Simulation	-	-	-	-				+++
		Pinto et al. [104]	2	None	X					X	X	X			76	Physical	None	No	-	-				+++
				Dense	X					X	X	X			66	Physical	None	No	-	-				+++
		Morrison et al. [92]	2	None		X[b][bb]				X	X	X	X	X	92	Physical	Passive*	No	0.02	Synthesis	X			++
				Dense		X				X	X	X	X	X	81	Physical	Passive*	No	-	-	X			++
	Learning Optimal Motions	Levine et al. [75]	2	None	X					X	X	X	X	X	80	Physical	Passive	No	-	-				+++
				Dense	X					X	X	X	X	X	90[g]	Physical	Passive	No	28[h]	Grasp Cycle				+++
		Kalashnikov et al. [61]	2	None	X					X	X	X	X	X	96	Physical	Passive	No	-	-				+++
				Dense	X					X	X	X	X	X	88[g]	Physical	Passive	No	22[h]	Grasp Cycle				+++
		Yan et al. [144]	2	Sparse	X					X	X	X	X	X	83	Physical	Passive	No	-	-				+++
		Berscheid et al. [22]	2	Dense		X				X	X	X	X	X	92	Physical	None	No	13	Pick-and-place				++

5.4.3. Selecting the Best Approach

This subsection selects the best grasp synthesis approach to be applied in RSIR. The best approach is selected in a manner similar to selecting the best gripper technology. The same requirements and weights are used. However, four of the requirements are defined slightly differently for synthesis approaches:

- **Computational cost** is reported as runtime in literature, either for only the synthesis process or the complete grasping cycle.

- **Repeatability** is reported as grasp success rate in literature. Although there is no consensus on what a successful grasp is [135], most approaches consider a grasp successful if the robot can lift the item and hold it for several seconds.

- **Dexterity** is reflected in approaches allowing to specify a certain grasp type: object area, binary type, or preshape. Object area refers to selecting a specific part of the object to (not) be grasped. Binary type allows to specify a precision or a power grasp, where precision grasps use only the fingertips and power grasps use all of the gripper [41]. An approach does not score on binary type if it requires retraining between tasks. Preshape allows to define a specific preshape, such as one from a human taxonomy, and is typically only used for grippers with more than two fingers. Hence preshape is not considered in this evaluation.

- **Availability** considers the amount of time required to implement the algorithm for RSIR. Simpler algorithms and algorithms that need little training data score higher because they require less time for implementation.

Since lifetime and cost are not applicable to a synthesis approach, they are not considered. Additionally, the approach must be able to work with only RGB and depth data from a camera located above the robot arm during runtime, as this is what is currently available for RSIR (R19, R20). Approaches that require more data are not considered. Finally, only approaches that report a runtime and success rate on a physical robot are included, because these metrics are an indication of the approach's performance (R5, R14, R23).

Table 5.5 shows exactly how the requirements are mapped to a score. Table 5.6 shows the resulting scores. In the table, the most promising approaches are found to be those by Baumgartl and Henrich and Berscheid et al.. Recall that the former synthesises grasps based on detected edges, whereas the latter uses reinforcement learning to combine grasping and shifting. The approach by Baumgartl and Henrich is selected to be implemented, as I am more familiar with analytical approaches than with transfer learning. However, the other high-scoring approaches are also promising and might be considered in future work.

Table 5.5: Score definitions for the synthesis approaches. Success rates are rounded to the nearest multiple of 10.

		Scores				
		1	2	3	4	5
Requirements	Item variety	No items with significantly different properties are used	-	-	-	At least two items with significantly different properties are used
	Clutter	None	-	Sparse	-	Dense
	Computational cost (Runtime [s])	>5 (synthesis)	1-5 (synthesis)	0.5-1 (synthesis)	0.1-0.5 (synthesis)	<0.1 (synthesis)
		>60 (cycle)	45-60 (cycle)	30-45 (cycle)	15-30 (cycle)	<15 (cycle)
	Repeatability (Success rate [%])	<60	70	80	90	100
	Dexterity (Grasp types)	Neither object area nor binary type	-	Object area or binary type	-	Object area and binary type
	Availability	Requires training, and no training data is available	Requires training, but training data or pre-trained model is available; or a very complex algorithm	Complex algorithm	Somewhat simple algorithm	Simple algorithm

Table 5.6: Ranking of the synthesis approaches suited for a parallel-jaw gripper for RSIR. The total score is calculated as the sum of the products between the weight and the score. Total scores are highlighted to improve clarity.

			Analytical approaches						Data-driven approaches				
		Weight	Suzuki et al.	Stuckler et al.	Stoyanov et al.	Baumgartl and Henrich	Klingbeil et al.	Ala et al.	Mahler et al. (1)	Mahler et al. (2)	Levine et al.	Kalashnikov et al.	Berscheid et al.
Item variety	Shape	2	5	5	5	5	5	5	5	5	5	5	5
	Size	1	5	5	5	5	5	5	5	5	5	5	5
	Mass	1	5	1	1	5	1	1	5	5	5	5	5
	Texture	2	5	5	5	5	1	5	1	5	5	5	5
	Deformability	2	1	5	5	1	1	1	1	5	5	5	5
	Clutter	2	1	3	3	3	5	1	1	1	5	5	5
	Computational cost	4	2	2	3	5	1	1	3	2	4	4	5
	Repeatability	3	4	5	3	3	4	4	3	4	4	4	4
	Dexterity	1	1	3	3	3	1	5	1	1	1	1	1
	Availability	5	5	3	4	5	3	2	2	2	1	1	2
	Total		80	83	86	95	62	61	58	73	84	84	93

5.4.4. Adjustments

This section explains the synthesis algorithm by Baumgartl and Henrich in more detail and proposes some improvements. Figure 5.24 illustrates the steps in the algorithm as proposed by Baumgartl and Henrich. A line represents a gripper jaw, thus two parallel lines represent the position of a parallel-jaw gripper. The algorithm consists of the following steps:

1. Take an RGB image of the scene. The items are assumed to be located on a flat surface with known height, although the paper also shows a successful grasp for overlapping objects.
2. Detect the edges in the RGB image using a Hough transformation.
3. Detect the lines in the edges. This is part of the Hough transformation.
4. Create pairs of parallel lines. This is also part of the Hough transformation.
5. Rank the line pairs based on a quality metric. The longer the lines are, the higher the metric is. Lines that are not long enough for the gripper jaws are filtered out, as well as line pairs that are further apart than the maximum opening of the gripper jaws.
6. The best line pair is used for computing the gripper position. The angle and location of the line pair provides the gripper location and rotation, using the known height of the supporting surface. The distance between the lines provides the gripper opening. While moving the gripper, force feedback is used to detect (unexpected) collisions.

The algorithm works quite well, with the paper reporting a 79% success rate and 0.03 seconds runtime. However, there are three limitations in implementing it in RSIR. First and foremost, the flat surface assumption is invalid for some scenarios encountered by RSIR (R26, R28, R29). This assumption is mainly used to determine the height of the grasping position. Baumgartl and Henrich propose force feedback to handle uncertainty, and this could be used to detect the supporting surface and obstacles in RSIR. However, this is unlikely to be a good solution, as lowering the gripper quickly creates impact on the obstacles, thus damaging the items (R27), and lowering the gripper slowly takes up too much time (R14). Since RSIR has access to a depth camera, the obvious alternative is to use the available depth information to determine the grasping height. This depth information is included in the algorithm for RSIR.

The second limitation is the validity of the metric. Using the length of the lines as a metric is a poor choice for some items, as illustrated in Figure 5.25. The longest line pairs clearly do not result in the best grasps.

Instead, a human would reason that it's better to grasp the plush by an arm or foot, or by its head. These grasps are based on the thought that a good grasp encloses the item. For the algorithm, this can be implemented as a large volume between the paired lines, when depth information is available. Figure 5.25 shows that this new metric (enclosed depth) results in more reasonable grasps. Enclosed depth is a logical successor to the metric by Baumgartl and Henrich and is simpler than most metrics commonly used in literature, which often consider force balancing and thus require item properties such as centre of mass and friction coefficient [109].

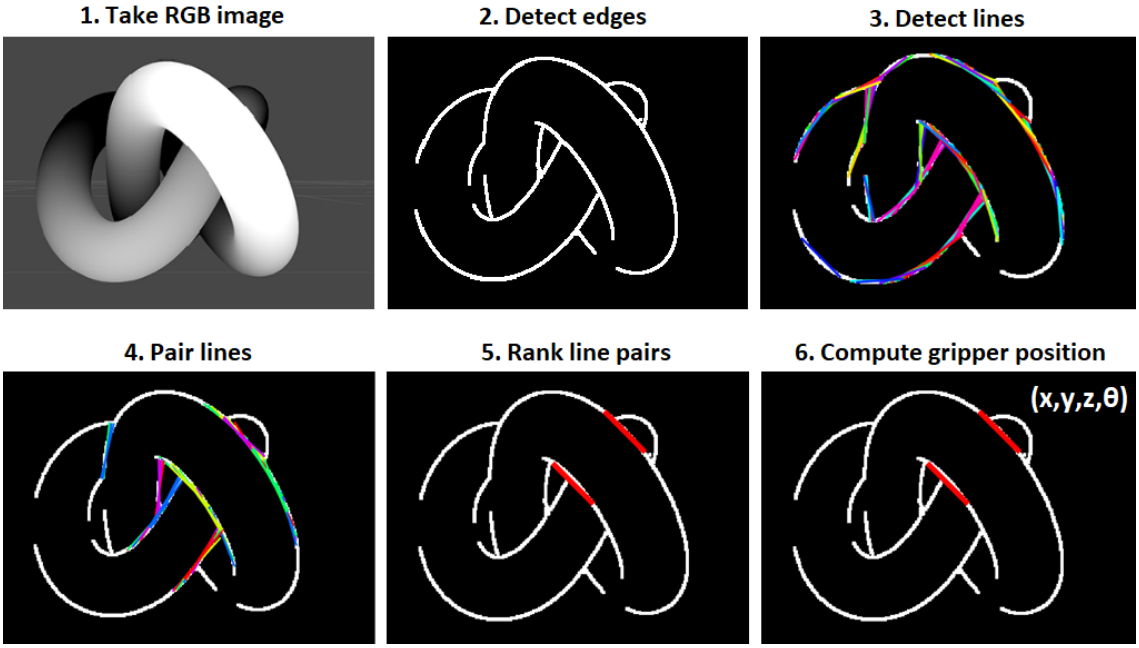
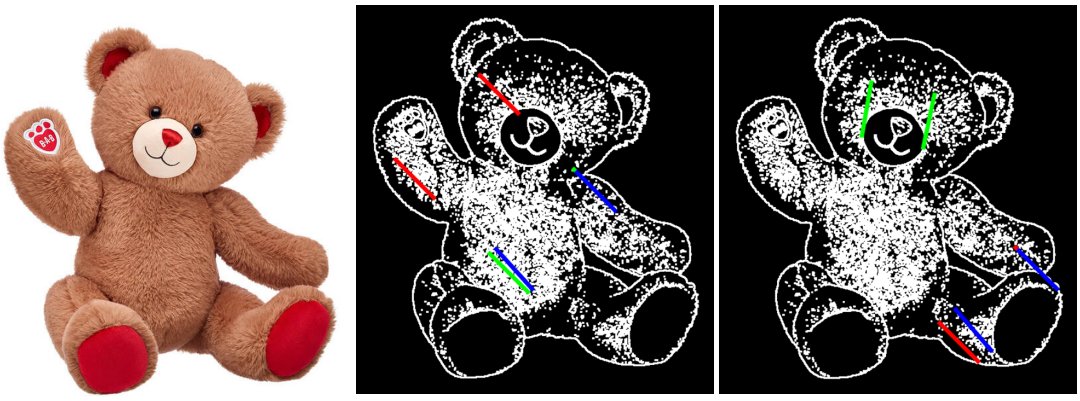


Figure 5.24: Steps in the synthesis algorithm by Baumgartl and Henrich.



(a) The RGB image of an example item [78].

(b) The three best grasps according to the metric by Baumgartl and Henrich.

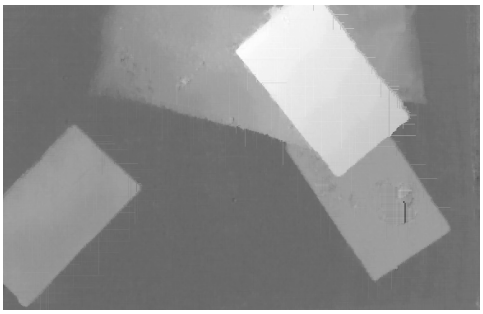
(c) The three best grasps according to the metric of enclosed depth.

Figure 5.25: An example item and the resulting grasps using two different metrics.

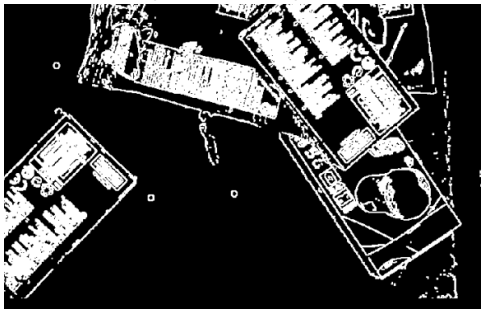
The final limitation comes from detecting edges in an RGB image. It was found during initial tests that the algorithm gets confused by the packaging of items. This is illustrated in Figure 5.26. The example contains several boxes, which are relatively easy to grasp. However, the straight lines on the packaging take the attention away from the edges of the boxes. Figure 5.26c shows there are many detected edges that are not relevant for grasping, and thus the algorithm is slow. Increasing the threshold for an edge detection improves runtime, but also ignores important edges, as is visible in Figure 5.26d. One way to solve this is to instead detect edges in the depth image. Figure 5.26e shows that this improves the edge detection. Hence from here on the algorithm in RSIR uses the depth image to detect edges. The exact implementation of the grasp synthesis approach is discussed in chapter 7.



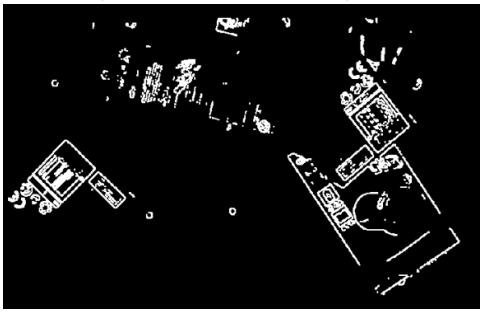
(a) The segmented RGB image of the scene.



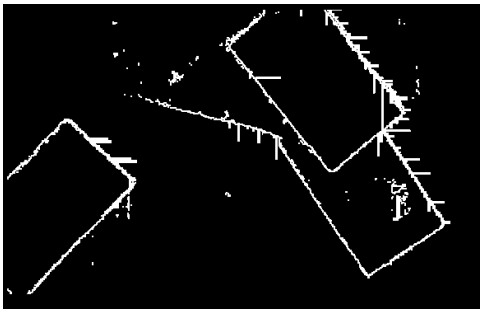
(b) The segmented and filtered depth image of the scene.



(c) Edge detections based on the RGB image, low threshold.



(d) Edge detections based on the RGB image, high threshold.



(e) Edge detections based on the depth image.

Figure 5.26: An example of a carrier with boxes, and the edge detections based on the RGB and depth images.

6

Ontology and Reasoning

This chapter develops the ontology and reasoning used in RSIR, thus answering the question: *"What method best provides the knowledge and reasoning capabilities RSIR should contain?"*. The previous chapter provided some important background information on grasping. This chapter builds on top of that by researching knowledge for failures, tasks, and items in section 6.1 (the knowledge acquisition phase in Methontology). The background on grasping will prove especially useful in specifying knowledge on item properties. Afterwards, section 6.2 considers Methontology's integration phase by investigating which existing ontologies RSIR's ontology can be based on. The development of RSIR's ontology and reasoning is then considered in section 6.3, with the final models being presented in section 6.4.

6.1. Required Knowledge

This section completes the knowledge acquisition phase of Methontology. Recall that the aim of this phase is to obtain the domain knowledge required to specific the ontology. To this end, subsection 6.1.1 discusses failures, subsection 6.1.2 discusses tasks, and subsection 6.1.3 discusses knowledge of items. Note that these three categories are the same categories used for specifying the competency questions.

6.1.1. Failures

This subsection considers the knowledge RSIR must contain about failures. To this end, the failures that can occur in SIR are investigated. These failures are derived from tests done on SIR and failures in robotics in general. Figure 6.1 summarizes those failures. The figure features three types of failures: component failures, derived failures, and critical failures. Component failures occur due to unavailable or inaccurate sensors, actuators, or (software) modules [55]. An example of an inaccurate component is when there is dust on a camera's lens. Derived failures are caused by other failures, such as inaccurate components or FM-GtP sending an empty input carrier. Finally, critical failures are failures that cause SIR's pipeline to fail.

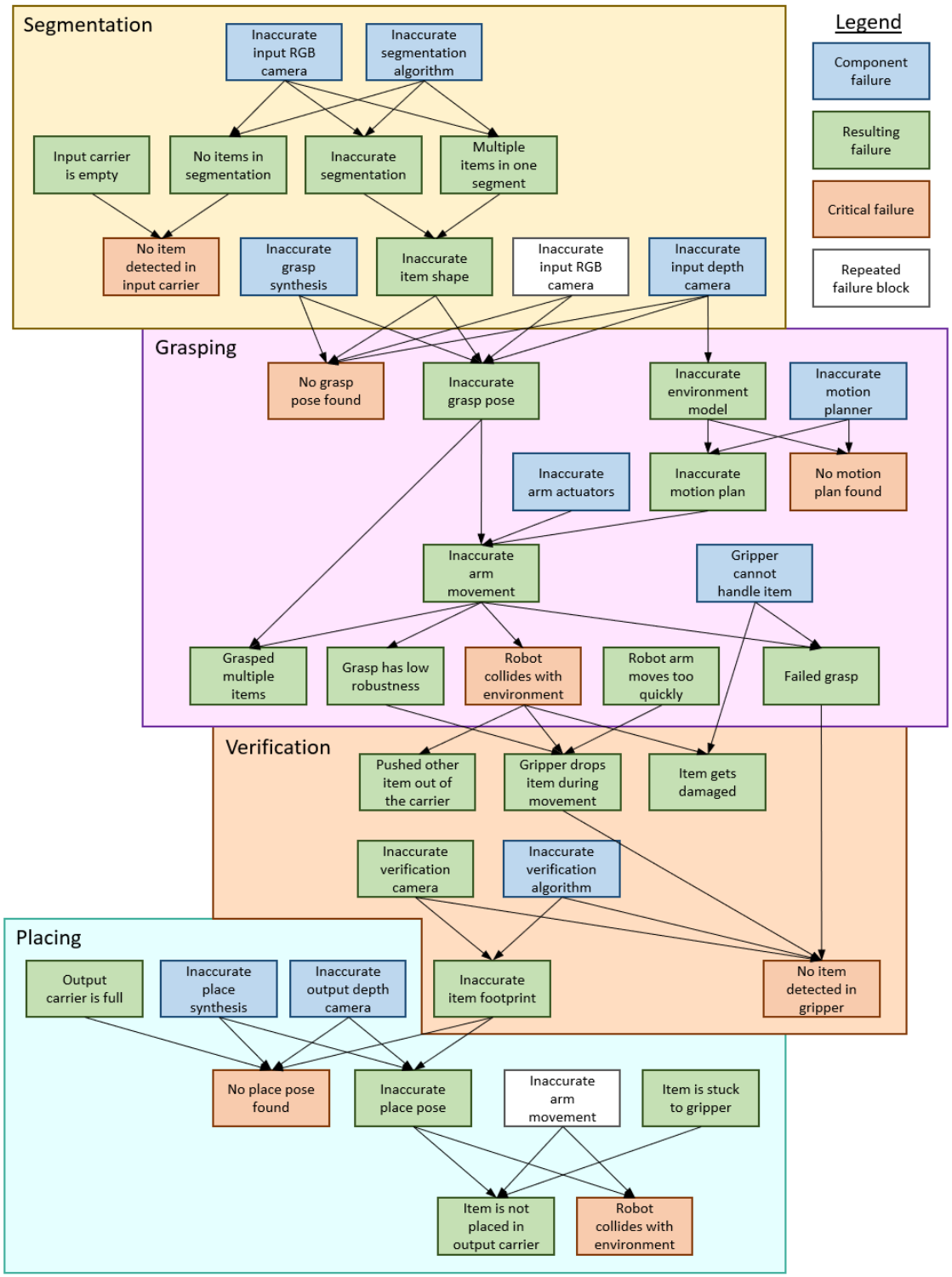


Figure 6.1: Causes and failures that can occur in SIR. The "repeated failure blocks" are blocks that are added twice in the diagram for clarity.

Note that Figure 6.1 is not meant to be a complete list of failures, but rather a basis for RSIR's ontology and reasoning. RSIR should also be robust against failures not foreseen at design, as it was concluded previously that RSIR is not meant to have hard-coded rules for every possible failure scenario. However, providing RSIR with information as depicted in Figure 6.1 likely improves its reasoning capabilities.

Segmentation Failures
Segmentation failures come from inaccuracies in sensing and processing. The camera images are noisy due to the limited camera resolution and items moving in the carrier [35]. The segmentation algorithm also has limited accuracy and it can merge multiple items together, split one item into multiple segments, or miss an item [35][93][129]. Figure 6.2 provides an example of this. These failures result in an incorrect estimation of the item's shape and position. Missing items is also a problem on the hardware level, as depth information cannot be measured or is measured incorrectly for specific items, such as items that are transparent, reflective, light-absorbing, too flat, or too small [35]. In such cases, the robot might incorrectly conclude that the carrier is empty. However, it is also possible that FM-GtP made an error and sent an empty input carrier [24].



(a) The RGB image of three milk cartons. (b) The corresponding segmentation.

Figure 6.2: An example of a segmentation failure in SIR. The segmentation merges the three items into one, misses some parts of the items, and labels part of the carrier as item.

Grasping Failures
Grasping failures are mostly caused by the limitations of the gripper and grasp synthesis approach. This was discussed in more detail in chapter 5. The resulting grasp pose can be inaccurate, or it might be that no grasp pose is found at all. An inaccurate grasp pose likely results in a failed grasp, which is when the item is not in the robot's gripper after the grasp has been attempted. Other causes for failed grasps are inaccuracies in the RGB camera, depth camera, or actuation of the robot arm [35][93][120]. A related case is when a grasp is barely stable, an example of which is shown in Figure 6.3. Although the grasping was successful, the resulting grasp is likely to cause failures later in the pipeline. Hence it might be preferred to put the item back and grasp it in a way that leads to a more robust grasp.



Figure 6.3: An example of a grasp that could be considered poor for RSIR, adapted from [69]. The cup swings around as the gripper lifts it. This grasp is not robust and the gripper is likely to drop the item when the arm moves.

Another type of grasping failure is grasping multiple items at once [35][64][75]. This can happen due to inaccuracies in previous steps in the pipeline, but also because items are stuck to each other due to lids, hooks,

glue, tape, or stickers [35]. Grasping multiple items at once is undesirable for RSIR, as RSIR currently does not detect the number of items in the gripper. In this case, the order is completed incorrectly. A final set of failures that occur during grasping stems from motion control. The motion planner might not find a motion plan, or might take too long trying to find one. When it does find a plan, it might be inaccurate, for example when the endpoint is not the desired robot pose, or when it collides with the environment. Such collisions can not only result in a failed grasp, but can also cause damage to the item [36].

Verification Failures
After grasping, the robot lifts the item to the verification camera. However, the robot might drop the item because the grasp was not robust enough to withstand the accelerations [129]. Alternatively, the robot can collide with the environment due to actuator inaccuracies or unmodelled obstacles [129]. When verifying whether a grasp was successful, the robot works with an inaccurate RGB camera and an imperfect verification algorithm. Especially the camera can be a limiting factor, due to the item moving when the image is taken, the item being deformable, or the item being too large for the camera view [35][128]. RSIR might thus be unable to detect the item that is in the gripper. When it does detect the item, the inaccuracies can still cause the algorithm to incorrectly estimate the item's footprint.

Placing Failures
The final group of failures stems from the placing task. An incorrect size estimation of the item or incorrect depth information can cause the item to not fit at the computed place pose [35]. It might also be that the output carrier is full, in which case it is not useful to attempt placing at all. Finally, it can be that the item is stuck to the gripper and thus does not end up in the output carrier.

In summary, there is a large variety of failures that can occur in RSIR, most of which can only be detected through other failures. The inaccuracies in RSIR's hardware and software components are the main cause behind failures. However, isolating the cause behind a failure might be difficult in practice, because many causes can be behind one failure.

6.1.2. Tasks
The next aspect RSIR must contain knowledge about is tasks, which is the subject of this subsection. The knowledge about tasks is required to have RSIR reason about how it can complete a task, which in turn is useful for generalizing RSIR to other related tasks. An example of such a related task is picking an item from the input carrier, scanning its barcode, and sorting it by placing it onto one of three moving belts.

This subsection attempts to capture knowledge of tasks by going through three steps: determining the list of related tasks, researching implementations for said tasks, and converting those implementations into a list of capabilities. A capability can be defined as: "a generic term referring to an action the robot has been programmed to perform, a skill the robot provides, or a behavior the robot has available" [96]. A capability can thus be considered the lowest-level behaviour of a robot, such as moving an arm, or activating a gripper. Capabilities are often considered in modelling tasks because knowledge of capabilities allows a robot to construct more complex behaviour [19][37]. Hence using capabilities allows the robot to do a vast variety of tasks without having hard-coded knowledge about said tasks.

Related Tasks
The first step in obtaining knowledge about related tasks is to construct the list of related tasks RSIR should be expandable to. This is done based on the envisioned future for RSIR, applications in Vanderlande's market segments, ideas and experience from Vanderlande employees, own ideas, and robotics applications encountered in literature. The knowledge RSIR contains should allow it to:

- **Item Variety**

  1. Handle a large variety of items [125][134].
     (a) Switch between grippers [24]. Having access to multiple grippers allows RSIR to select the gripper that is most suited for grasping the presented item.
     (b) Do preshaping. A preshape refers to the pose of a gripper before grasping [135], some examples of which are presented in Figure 6.4. Especially for grippers with three or more fingers, it is common to select a preshape, move to the item, and then close the fingers. This is because

such approaches generally take less computing time and are less susceptible to modelling errors than approaches that compute the desired fingertip locations [135]. RSIR should allow for preshaping if it is to be equipped with a gripper with three of more fingers in the future.

(c) Shift items. Shifting refers to moving an item in the carrier without grasping it, for example by pushing against its side. This behaviour allows the robot to reposition the items such that previously ungraspable items become graspable [22][61].

(d) Limit the grasp force. This prevents items from getting damaged [24]. Since the maximum allowed force can depend on aspects such as the type of gripper, grasp pose, and fragility of the item, this capability relates to the knowledge base.



Figure 6.4: Preshapes for a three-fingered hand [90]. From left to right: spherical, cylindrical, precision-tip, and hook grasps.

2. Handle different types of clutter. Besides the three types of clutter using for grasping, there are two types of clutter that depends on the location of the item as illustrated in Figure 6.5. Both the grasping and the placing approach should handle the following types of clutter [24][134]:
   (a) Item in isolation: only one item in the carrier.
   (b) Item in sparse clutter: multiple items in the carrier, but not touching each other.
   (c) Item in dense clutter: multiple items in the carrier, touching and/or overlapping.
   (d) Densely packed item: multiple items in the carrier, arranged in an organized way.
   (e) Item in narrow spaces: (multiple) item(s) arranged near non-moveable surfaces, such as items close to the carrier's edge, or boxes located in a truck.



(a) Densely packed items. (b) Items in narrow spaces.

Figure 6.5: Additional types of clutter encountered by RSIR.

3. Handle different item flows. RSIR may be applied in warehousing applications that have moving items. Vanderlande defines the following item flows, visualized in Figure 6.6 [24][134]:
   (a) Singulated flow: single items not lying against or next to each other, either in fixed windows or with fixed aps between them.
   (b) Single flow: items not lying against or next to each other, with varying distance between them.
   (c) 2D flow: items possibly lying against to or next to each other, with varying distance between them, but not stacked on top of each other.
   (d) Bulk flow: items possibly on top of each other, lying against or next to each other, in any configuration.
4. Handle different supporting surfaces. This goes for both grasping and placing actions. The following supporting surfaces can be discerned:

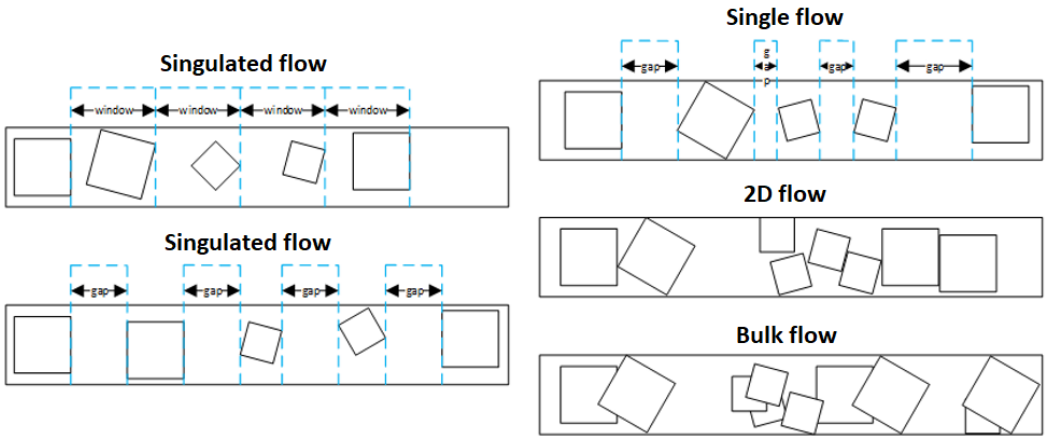


Figure 6.6: Types of item flows defined by Vanderlande [134].

(a) Flat surface, such as the bottom of a carrier.

(b) Uneven surface, such as an item located on top of several other items, or on a tilted belt. For grasping, this can be a problem because some grasp synthesis approaches (primarily those that do not use depth information) simply lower the gripper to a predefined height [135]. This does not work if the height of the supporting surface is unknown or varying.

(c) Vertical stack [134], which is when the robot's approach direction is perpendicular to the direction of gravity. The difficulty in this type of stack is that the robot cannot freely push against the item, or there is a risk of the robot pushing the item off the stack.

(d) Point support
   i. Taking from an item from a robot or human
   ii. Handing an item to a robot or human

(e) No support
   i. Dropping an item
   ii. Throwing an item
   iii. Catching an item

- **Grasping**

1. Synthesize a grasp for each type of gripper RSIR has.
2. Grasp selectively. This is for future scenarios where multiple item types are in one carrier compartment [24]:
   (a) Recognize a specific item in the carrier.
   (b) Synthesize a grasp for one specific item in the carrier.
   (c) Search for a non-visible item. It may be that the target item is occluded by other items. In that case, the robot must search for the target item.
   (d) Grasp an item in a specific location. For example, when items are stacked on top of each other, it is preferred to grasp the item on top, because grasping an item near the bottom might cause the stack to topple over.

- **Placing**

1. Place the item stably [24]. Stable placement of items prevents them from getting damaged, as well as allowing other items to be stacked on top of them.
2. Place an item such that it can be picked up again. This might be a future scenario where the placing algorithm is more complex, and has a preferred orientation to place the item in. In that case, it might be required that the robot grasps the item, places it differently, then grasps it again.
3. Estimate the item's pose. This is required for placing the item in a specific orientation, such as placing plants such that they are standing on their plant pots.

- **Robot Movement**

  1. Avoid static obstacles [24], such as the edges of the carrier or other items.
  2. Avoid dynamic obstacles [24], such as obstacles located on a conveyor belt that move with an unpredictable velocity.
  3. Scan barcodes on items.
  4. Move with different velocities. This may be useful since some failures in RSIR are due to the gripper dropping the item during movement [36]. In such a scenario, it might be beneficial to try the same grasp but move the robot arm with a lower velocity.

- **Human Interaction**

  1. Allow the operator to determine whether an action was successful. This is useful when RSIR cannot (confidently) determine whether an action was successful, or when it incorrectly concludes an action was (not) successful.
  2. Allow the operator to edit the knowledge base. Again, this is useful for correcting wrong conclusions, or providing the robot with more knowledge about the item.
  3. Inform the operator of failures that occur [24]. This way the operator can take over when the robot cannot solve the failure. Non-critical failures (such as failing to grasp an item with a specific gripper) can be useful for better understanding the limitations of each component.

- **Reasoning**

  1. Reason whether RSIR can handle an item before the input carrier arrives [24][125]. FM-GtP will not delegate carriers to RSIR if it knows that the carrier contains items that RSIR cannot handle. Hence it is useful for RSIR to answer queries from FM-GtP whether it can handle a specific item. For example, RSIR can reply that the item cannot be handled if the item properties are outside the gripper specifications, or when a previous attempt with said item failed.
  2. Implement learning to predict whether an item can be handled [125]. Future versions of SIR may include machine learning algorithms to determine when to use which gripper.
  3. Estimate item properties. Before an item is encountered by RSIR, nothing about its properties is known. However, the robot could make better decisions about what components to use (gripper type, synthesis approach) if it knew more about the item. Hence it may be favourable to estimate the properties of items that are common. The properties should then also be saved in the knowledge base.

- **Failure Detection.** These tasks are derived from section 6.1.1 and help in detecting failures:

  1. Detect whether there is an item in the gripper. This can verify that a grasp was successful.
  2. Detect whether a specific item is in the gripper. This can verify that the correct item was grasped from the carrier, for future scenarios when there are multiple item types in one compartment.
  3. Detect how many items are in the gripper. If this is possible, grasping multiple items is no longer a failure, but an advantage that allows the system to complete orders faster.
  4. Detect the stability of a grasp. This could prevent the robot from moving too quickly when it does not have a firm hold on the item.

Implementations of Tasks
A literature study is done to find implementations for each task in the list of related tasks. Each implementation is then broken down into capabilities. The implementations and break-downs are included in Appendix B. To summarize, to be expandable to every related task in the list, RSIR must be able to implement all of the capabilities in Table 6.1. Note that most of these capabilities are processing capabilities.

Table 6.1: The capabilities required for the implementations of related tasks that RSIR must be expandable to.

Actuating capabilities
Activate gripper
Activate pneumatic link
Apply known vertical force
Apply vertical force
Deactivate gripper
Deactive pneumatic link
Knock on item
Move gripper
Move gripper horizontally
Move robot arm
Orientate item for placing
Push item
Rotate item in-hand
Slide over item
Tap item
Touch item
Communicating capabilities
Ask operator for confirmation
Knowing capabilities
Know conveyor velocity
Load corrrect item property
Load drop location
Load environment model
Load item model
Load surface height
Recall positions of previous target items
Select gripper with position sensors
Select gripper with preshape
Select gripper with tactile sensors
Select motion primitive
Select vacuum gripper
Sensing capabilities
Record sound
Sense acceleration
Sense arm load
Sense deformation
Sense depth
Sense force
Sense grasping force
Sense gripper position
Sense mass of carrier
Sense pressure in gripper
Sense robot arm torque
Sense strain
Take depth image
Take RGB image

Processing capabilities
Add item to model
Compare sensory data to item model
Compute angular displacement from RGB images
Compute deactivation moment
Compute expected motion
Compute grasp location
Compute grasping force
Compute motion command
Compute placement pose
Create environment model
Create item model
Create motion plan
Detect barcode
Detect denting from position measurements
Detect empty areas
Detect grasp on human
Detect item
Detect item footprint
Detect item in depth image
Detect item in RGB and depth image
Detect item in RGB image
Detect material in RGB image
Detect robot arm in image
Detect semantic item part
Detect slip from gripper position measurements
Detect slip from tactile measurements
Detect supporting surface via force
Detect target item
Detect toppling in force measurements
Detect toppling in position measurements
Detect ungraspable item
Determine conveyor velocity
Determine item height
Determine item material from tactile measurements
Determine item position
Determine item shape
Determine item shape from depth image
Determine item shape from RGB image
Determine item shape from tactile measurements
Determine item's primitive shape
Determine item's principal axis
Determine local item shape from tactile measurements
Determine maximum grasp force
Determine shift direction
Determine stiffness from acceleration measurements
Determine stiffness from tactile measurements

Processing capabilities (cont'd)
Determine surface roughness from measurements
Determine target item position
Estimate item shape
Estimate surface height
Find free spot in gripper rack
Find gripper in gripper rack
Find suitable placement area
Isolate target item from segmentation
Map grasp from known model to sensory data
Predict item position
Run placement simulation
Segment carrier into rectangles
Segment depth image
Segment image
Segment RGB and depth image
Segment RGB image
Select rectangle segment for placement
Set maximum grasp force
Set robot arm velocity
Synthesize grasp
Synthesize grasp on specific area of item
Synthesize grasp via simulation
Synthesize place pose
Track belt
Track item on belt
Update environment model
Verify grasp validity
Verify item by comparing arm loads
Verify item by comparing carrier masses
Verify item by comparing images
Verify item through gripper position
Verify item through pressure
Verify items in gripper by comparing segmentations
Verify specific item by comparing properties

6.1.3. Items

This subsection considers the knowledge RSIR must contain about items. The specifications of SIR from chapter 2 provide a first set of item properties RSIR should contain: **dimensions**, **mass**, **orientation**, **point cloud**, **footprint**, and **segmentation mask**. Because the item cannot be damaged, its **fragility** is also impor-

tant. Furthermore, the vacuum gripper was found to have limitations based on item **friction**, **porosity**, mass, **mass eccentricity**, **deformability**, and the presence of **removable features**. On top of these, item **shape** and **texture** can result in a small contact area between the suction cup and item, thus resulting in a failed grasp. Tests were done at Vanderlande based on the **GS1 category** of an item.

The gripper technologies in chapter 5 were found to have limitations based on the item's dimensions, **level of clutter**, grip strength (that is, mass and friction coefficient), and fragility. The synthesis approaches were found to be limited by an item's **primitive shape**, **size**, mass, texture, deformability, and level of clutter. The properties used by the synthesis approaches are shape, primitive shape, mass, **principal axis**, **bounding box**, **centroid**, centre of mass, and presence of **parallel lines or surfaces**.

The research into failures in section 6.1.1 shows that an item's **transparency**, **reflectivity**, **light absorbency**, **height**, and size are limiting factors for the depth camera. Inaccurate segmentation can be due to a visual similarity between the item and carrier, for example when an item's **colour** or **visual texture** matches that of the carrier. An item's **interlockability** can cause the robot to grasp multiple items at once, whereas an item's **stickiness** might cause it to stick to the gripper.

Finally, although not currently used, two more properties might be useful in the future: an item's **material** and **EAN code**. The former can be used to estimate other properties, such as stiffness. The latter could be used to look up an item's category. For example, if an item's EAN code is known, the robot could look up more information using EAN-based search engines. There are several such database available [5][6][7][8][33][38][47][72], some of which can be accessed via APIs [6][7][8][38][72]. For example, the EAN code 8713439227062 belongs to the computer mouse in Figure 6.7a. When searching for this EAN code, the information in Figure 6.7 is found. Although this search capability is not further researched in this thesis because RSIR has no access to an item's EAN code, it could be the subject of future work.



(a) A computer mouse that is used for testing SIR.

Product name for EAN 8713439227062:

Trust Computer Products Trust Zelo Silent Click Draadloze Muis Zwart

Issuing country: NL

8 713439 227062

(b) The results obtained using EAN-search [7]. In English: "Trust Computer Products Trust Zelo Silent Click Wireless Mouse Black". The text is linked to an Amazon listing for the product.

GS1 Company Prefix	8713439
Last Change Date	Feb. 25, 2020
Party Data Language	nl
GS1 Company Prefix Licensee	
Party Name	Trust International B.V.
Role	BUYER
Role	CARRIER
GS1 Key Licensee	
Party Name	Trust International B.V.
Role	BUYER
Role	CARRIER

(c) Part of the results obtained using GS1global [47].

Figure 6.7: An example of a product and the information gained when searching for it in EAN code databases.

This concludes the knowledge acquisition phase of Methontology. The next section handles the integration phase by looking into existing ontologies that contain (some of) the knowledge RSIR should contain.

6.2. Existing Ontologies

This section researches existing ontologies and the possibilities for integrating RSIR with one or multiple of them. Recall that the first step in Methontology's integration phase is to select a meta-ontology. A meta-ontology is the most general type of ontology, describing general aspects such as space, time, and events [59]. However, meta-ontologies are of little use to robotics because they are too high-level [130]. For example, whether a robot's device is a Physical Entity or an Abstract Entity is of little importance in practice. Hence this thesis instead selects which mid-level ontology to base RSIR on. Mid-level ontologies are of more use to robotics because they describe domains, tasks, and activities [59]. Additionally, a decision must be made on which low-level ontologies RSIR is integrated with. These are ontologies specific to an application [59]. The first and second subsections of this section consider mid-level and low-level ontologies, respectively. Afterwards, RSIR's ontology is developed in section 6.3.

6.2.1. Mid-level Ontology

There are two mid-level ontologies relevant for RSIR: CORA and KnowRob. **CORA** (Core Ontology for Robotics and Automation) is the 2015 IEEE standard ontology that "allows for the representation of, reasoning about, and communication of knowledge in the robotics and automation domain" [59]. The ontology covers robot systems, robot parts, and robot poses, and is integrated with several other ontologies (SUMO, CORAX, RPARTS, POS). **KnowRob**, on the other hand, is inspired by Cyc [130]. Cyc is a common-sense ontology in development since 1984, originally designed by a team of professional ontologists and logicians [87][130]. KnowRob was published in 2009, before IEEE's ontology standards were created [59][96], and thus KnowRob is not integrated with those standards. A second version of KnowRob is in development and was first published about in 2018 [21], but its ontology is not yet available.

Whether RSIR should be based on CORA or KnowRob comes down to three aspects: level of standardization, ease of integration, and integrated ontologies. The level of standardization of CORA is obviously high. KnowRob, in contrast, leans more towards application than standardization, and with a second version of KnowRob coming up it is likely that the ontology changes. Regarding ease of integration, KnowRob is already implemented, but lacks documentation and following the installation instructions results in several errors. CORA, on the other hand, is not implemented, but has excellent documentation. The final aspect is the ontologies that are integrated with CORA or KnowRob. If an ontology based on CORA is relevant for RSIR, it could be preferred to select CORA over KnowRob, or vice versa. Hence the next subsection discusses lower-level ontologies that could be relevant for RSIR.

6.2.2. Low-level Ontologies

This subsection considers low-level ontologies that are relevant for RSIR. They can be divided into two categories: those containing task knowledge and those containing item knowledge. Out of the ontologies presented in this subsection, several are chosen for integration with RSIR.

Task Knowledge

One of the main aspects of RSIR's ontology is the knowledge about tasks. There is a handful of ontologies that attempt to capture this knowledge. **K-CoPMaN** (Knowledge-enabled Cognitive Perception for Manipulation) connects sensory information to abstract symbolic knowledge. This allows the robot to answer new types of queries, such as inferring which items are missing on a partially set table. Based on this inference, the robot can determine what task it should perform next. For example, if the milk is missing, the robot reasons it must drive to the fridge to get it. The focus of K-CoPMaN seems to be on matching sensory information with symbolic knowledge, rather than reasoning about tasks.

RoboEarth is an extension of KnowRob that, as the authors describe it, is "a Wikipedia-like platform that robots can use for sharing knowledge about actions, objects, and environments" [131]. Action recipes are computed in the cloud and must be locally executed by an engine based on CRAM [4]. However, CRAM attempts to achieve robustness by hard-coding rules [40], which was concluded to be insufficient for RSIR. Furthermore, it might be undesirable for Vanderlande to have RSIR's reasoning dependent on the cloud: RSIR's powerful computer might be faster at computing action recipes than the cloud, and can keep working even if the cloud is not available. Finally, it is unclear how much of RoboEarth is applicable to RSIR, as the examples of RoboEarth only show robots moving around a hospital room and serving drinks.

One of IEEE's study groups is currently developing the **Task Ontology**: an ontology to describe robot task structures and reasoning [96]. The ontology is integrated with CORA, and considers concepts such as tasks,

capabilities, and constraints. Although it is unfinished, it seems to be the most complete ontology regarding tasks so far. Figure 6.8 shows a fragment of the current version of the ontology.

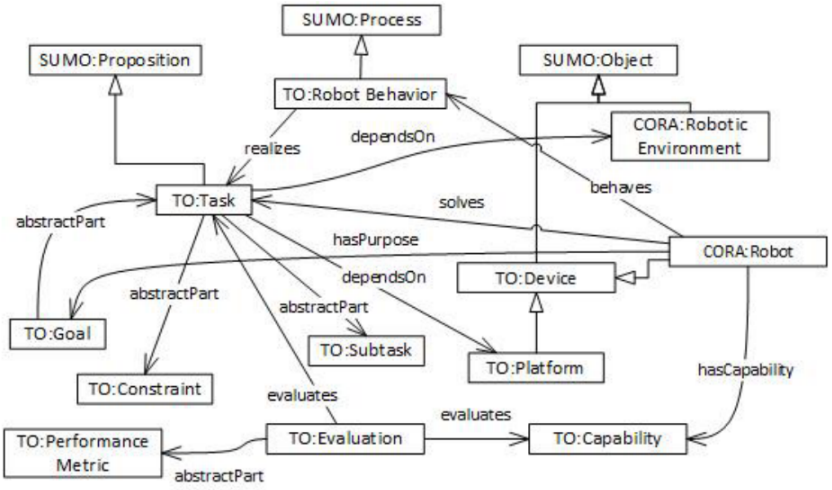


Figure 6.8: A fragment of the Task Ontology [96]. The solid arrows represent sub-class relations.

Closely related to the Task Ontology is the ontology developed in the thesis of Vredeveldt [139]. This ontology aims to assess the capabilities of autonomous underwater vehicles (AUVs) and is henceforth referred to as the **AUV Ontology**. It is presented in Figure 6.9. The ontology is integrated with the Task Ontology and introduces several new concepts, such as Components requiring Inputs and providing Outputs, and Capabilities being split into three categories based on their function (Planning, Execution, and Evaluation). The AUV Ontology provides a good example of how to extend the Task Ontology.

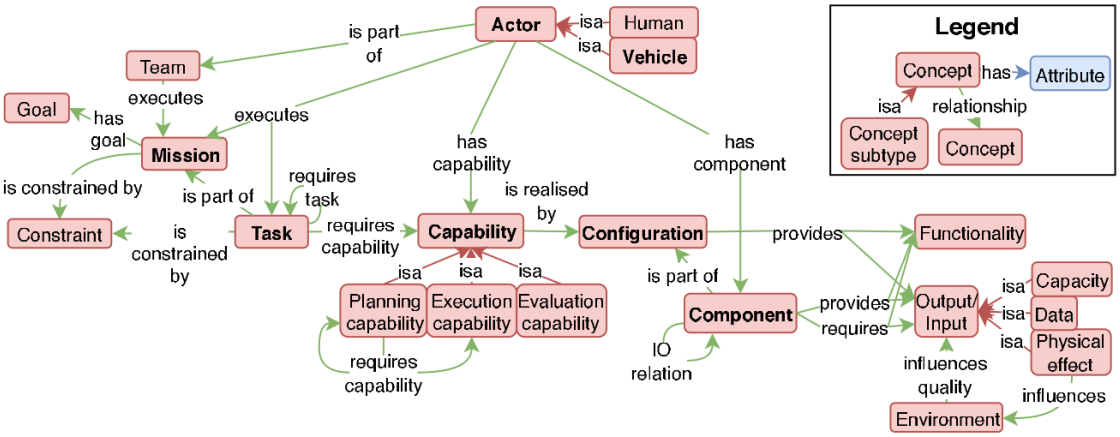
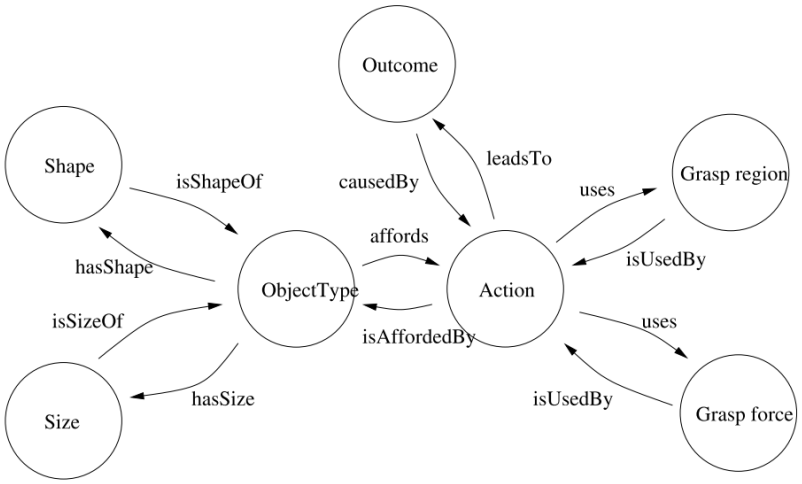


Figure 6.9: The AUV Ontology [96].

A final ontology considering tasks is the one by Barck-Holst et al., henceforth referred to as the **Grasping Affordance Ontology**. This ontology focusses on grasping actions, different item shapes and sizes, and the success or failure of an attempted grasp. Figure 6.10 shows this ontology. One unique aspect of the Grasping Affordance Ontology is that it considers item properties (Shape and Size), which is relevant for RSIR. The authors do not mention having integrated their ontology with other ontologies.



(a) The ontology.

$$
\begin{aligned}
Shape &\equiv Cylinder \sqcup Box \\
Size &\equiv LargeSize \sqcup MediumSize \sqcup SmallSize \\
ObjectType &\equiv Coke \sqcup Homer \\
GraspRegion &\equiv FromTop \sqcup HighRegion \sqcup MiddleRegion \sqcup LowRegion \\
GraspForce &\equiv HighForce \sqcup MiddleForce \sqcup LowForce \sqcup NoForce \\
Action &\equiv Grasp \\
Outcome &\equiv Success \sqcup PositionError \sqcup GraspError \sqcup LiftError \sqcup SimulationError
\end{aligned}
$$

(b) The class structure of the ontology as used during the experiment by Barck-Holst et al..

Figure 6.10: The Grasping Affordance Ontology [16].

Item Knowledge

There are several other ontologies that might provide useful concepts or structures, despite being in a (slightly) different domain than RSIR. For example, the **OMICS** (Open Mind Indoor Common Sense) ontology contains knowledge on concepts such as human desires and items and their locations [48]. It is used by KnowRob to determine where to look for items [130]. For example, OMICS allows the robot to reason that fridges are commonly found in the kitchen. RSIR needs no knowledge of fridges, but the reasoning algorithm behind OMICS could still be used, for example if rectangular items are more often in the corner of a carrier than spherical items. When RSIR detects an item in a corner, using OMICS it could infer knowledge about the item's shape.

The **Good Relations** ontology describes stores, products and services found on the internet. For example, it contains information about store opening hours, product prices, and product numbers [54]. According to Good Relations, their ontology is used by Google, Yahoo, Best Buy, and others [2]. Figure 6.11 shows a part of the ontology that is relevant for RSIR. Because many shops use Good Relations, RSIR could use the ontology to look up missing information. For example, if RSIR has the EAN code of a product, RSIR could scan the internet for webshops that sell that product and specify more information, such as weight and size.

Finally, there are two ontologies based on Wikipedia. Because Wikipedia contains a large amount of knowledge, using an ontology based on it could provide RSIR with general information, such as the fact that apples are a type of food. When presented with an apple, the robot could decide to handle it carefully because food is commonly fragile. The two ontologies are DBpedia and YAGO. **DBpedia** extracts structured information from Wikipedia [87][137]. **YAGO** does this too, but also considers some other sources [137]. Unfortunately for RSIR, the knowledge they contain is on subjects such as sports, people, animal species, countries, and historical events [14][15][44][122]. This is not the type of knowledge RSIR can work with and thus these ontologies are not considered.

6.2.3. Selection of Ontologies

Based on the research into existing ontologies, it can be concluded that CORA is the most promising mid-level ontology to base RSIR on. This is because it is an IEEE standard, has excellent documentation, and is integrated with the most promising task ontology: the IEEE Task Ontology (and the closely-related AUV On-

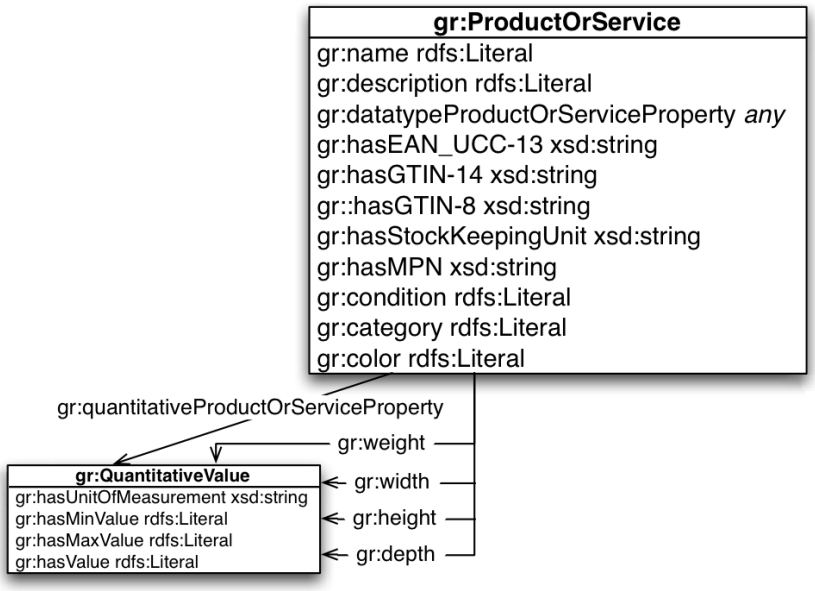


Figure 6.11: Part of the Good Relations ontology regarding products, adapted from [54].

tology). These advantages outweigh the disadvantage of CORA not having an off-the-shelf implementation like KnowRob does. Furthermore, the Grasping Affordance Ontology and Good Relations ontology provide good examples of how to include item properties. Because they are not integrated with CORA, they will not be integrated with RSIR. However, the two ontologies are used as an inspiration for RSIR.

6.3. Conceptualization

This section considers the conceptualization phase of Methontology: the general structure of RSIR's ontology and reasoning is iteratively developed based on the acquired background knowledge and existing ontologies. Subsection subsection 6.3.1 explains the main idea behind the ontology and reasoning. Afterwards, several iterations of the ontology and reasoning are considered. Subsection 6.3.2 discusses two iterations of the algorithm for combining capabilities to achieve a task. Similarly, subsection 6.3.4 discusses two iterations of the ontology for reasoning about failures. The next section in this chapter presents the final version of the ontology.

6.3.1. Main Idea behind the Ontology

This section explains the main idea behind RSIR's ontology. The ontology starts with a Robot. A Robot is equipped with one or multiple Devices, where each Device provides one or multiple Capabilities. Each Capability is defined by its prerequisites (Inputs) and its outcomes (Outputs), as well as its Constraints. The Robot must perform Capabilities one after another to complete a Task. An ordered list of Capabilities the Robot executes is called an Action Plan. Additionally, RSIR knows the concept of Target Item to keep track of the knowledge of the item.

Figure 6.12 illustrates this main idea. In this simplified example, the Task is to obtain a grasp pose. The Robot has four Devices it can choose from: two cameras and two algorithms. In this case, each Device provides one Capability. Given this knowledge, RSIR should be able to reason that the correct way to combine these Capabilities is as shown in Figure 6.13a. Putting the Capabilities in a chronological order results in the Action Plans shown in Figure 6.13b. Note that there are three Action Plans because it makes no difference when *Sense Input Carrier Depth* is executed, as long as it is executed before *Synthesize Grasp*. RSIR should choose one of these Action Plans to execute.

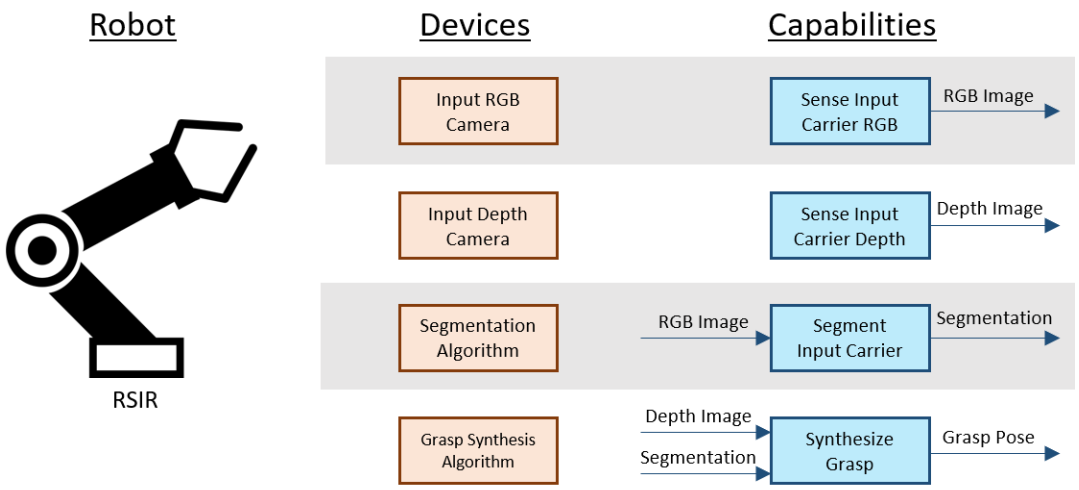
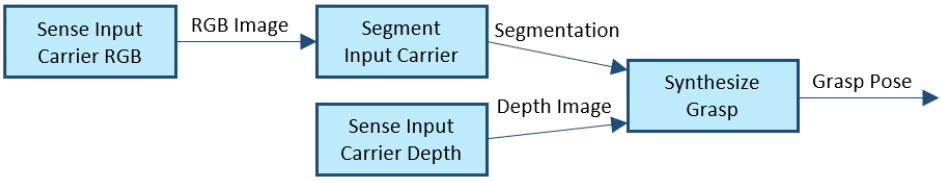
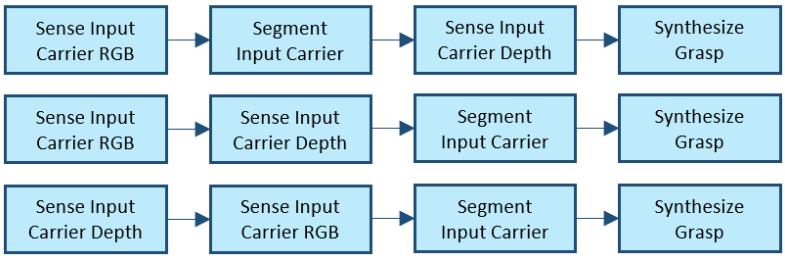


Figure 6.12: An example to illustrate the concepts of Robot, Device, and Capability. In this example, each Device provides one Capability. Robot icon adapted from [45]



(a) The combination of Capabilities that leads to successful completion of the Task.



(b) The three Action Plans that complete the Task.

Figure 6.13: The correct combination of capabilities for the example in Figure 6.12 when the Task is to obtain a grasp pose.

It follows that the seven main concepts behind RSIR's ontology are: Robot, Device, Task, Constraint, Action Plan, Capability, and Target Item. The first four concepts are part of CORA and the Task Ontology. Capability is inspired by the UAV Ontology. Including the concept of Capability allows one Device to provide multiple Capabilities. For example, the motion execution Device provides all Capabilities that include motion, such as moving to a grasp pose and moving to a place pose. If the motion execution Device is unavailable, the Robot immediately knows that all Capabilities provided by that Device are now unavailable. Because a Capability is defined by its Inputs, Outputs, and Constraints, the capabilities from section 6.1.2 can easily be included in the ontology.

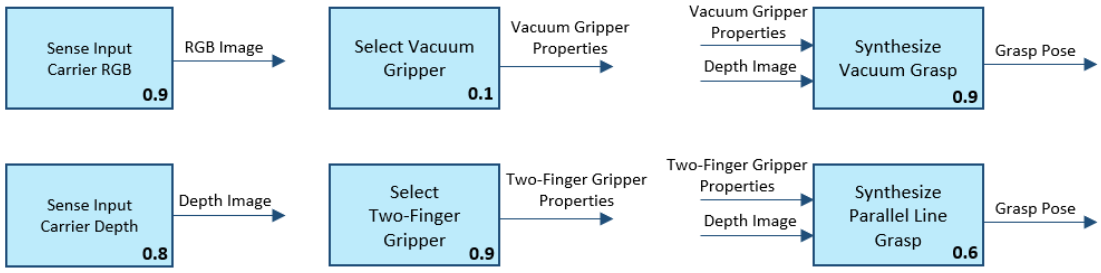
The concepts of Target Item and Action Plan are unique to RSIR. The concept of Target Item contains all knowledge of the item, similar to the Grasping Affordance Ontology and the Good Relations ontology. The concept of Action Plan allows RSIR to combine and order Capabilities in a way that completes a Task. Additionally, Action Plans can be saved so they can be re-used later, which saves computation time.

The three aspects of RSIR's ontology that were iterated upon are the algorithm for creating Action Plans, the representation of item properties, and the reasoning about failures. The next three subsections discuss these aspects, respectively.

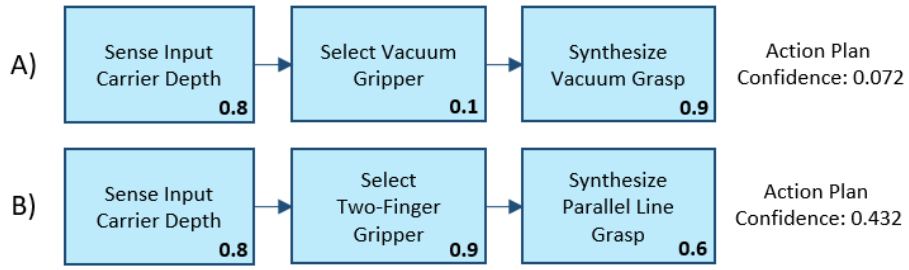
6.3.2. Reasoning about Action Plans

This subsection discusses the algorithm for creating an Action Plan out of a set of Capabilities. Two iterations of this algorithm are discussed, using the example in Figure 6.14a. The Task in the example is to obtain a grasp pose. However, from the figure it becomes clear that the Robot has two options to do so: by using a vacuum gripper and a vacuum synthesis approach, or by using a two-finger gripper and a parallel line synthesis approach. Let's assume the target item is a sponge. Based on this knowledge, RSIR can determine a confidence score for each Capability. The confidence score can be considered the probability of the Capability succeeding. For example, the *Select Vacuum Gripper* Capability has a score of 0.1 because a sponge is porous, and a vacuum gripper is nearly always unable to handle porous items. One the other hand, the vacuum synthesis algorithm has a high confidence score because the sponge's shape makes it easy to detect flat surfaces. More on calculating confidence scores is discussed in later in this section.

Figure 6.14b shows the two Action Plans that RSIR can find, along with their confidence scores. The confidence score of an Action Plan is the product of the confidence scores of its Capabilities. It is clear that selecting Action Plan B, which uses the two-finger gripper, is the preferred option for this example.



(a) The Capabilities the Robot has access to, including their confidence scores.



(b) Two Action Plans that complete the Task, and the confidence score per Action Plan.

Figure 6.14: A.

For both iterations of the planning algorithm, the algorithm starts at the "end" of the Action Plan. That is, the algorithm first looks for Capabilities that provide a grasp pose, since this is what defines the task. If this Capability requires any Inputs, the algorithm looks for other Capabilities that provide those Inputs. This process continues until no more Inputs are required.

Iteration 1: Greedy Plans

The first iteration of the algorithm could be considered greedy. When presented with multiple Capabilities that provide the same Output, the algorithm selects the Capability with the highest confidence score. Figure 6.15 illustrates the reasoning steps for the example. Because the vacuum synthesis has a higher confidence score than the parallel line synthesis, the algorithm selects the vacuum synthesis. Based on this, it then selects the vacuum gripper. Hence this approach results in the suboptimal Action Plan A. This is because the algorithm does not take into account the low confidence of the vacuum gripper when selecting the vacuum synthesis approach.

From this simple example, it can be concluded that the greedy algorithm does not result in the optimal action plan. In the example, RSIR selects an action plan that is six times less likely to succeed than the optimal action plan. Hence this algorithm must be improved.

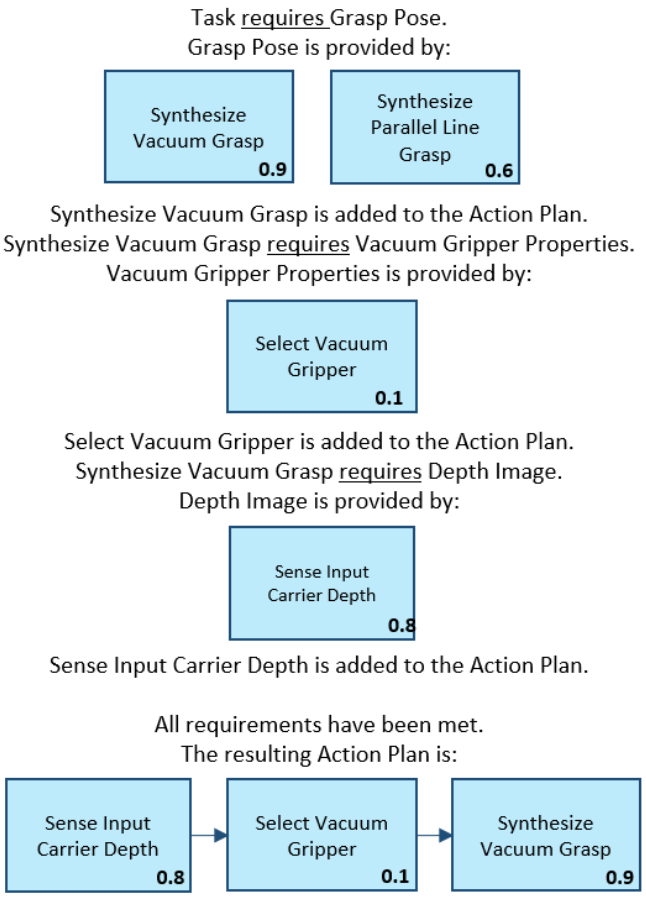


Figure 6.15: Steps of the greedy algorithm for creating action plans.

Iteration 2: Complete Plans
The second iteration of the algorithm for creating action plans computes all possible action plans. This is illustrated in Figure 6.16. The improvement over the greedy algorithm is that this complete algorithm always finds the optimal action plan. An additional advantage is that having access to all possible action plans creates the possibility for exploration. It will become clear in chapter 9 that executing a suboptimal action plan is sometimes favourable because it provides the robot more information about the item.

The disadvantage of the complete algorithm is that it is more computationally expensive than the greedy algorithm (R14). This is especially true when the robot has many redundant Capabilities. There are two reasons why this increased computation time is not critical. The primary reason is that action plans rarely need to be computed. As an example, consider a robot that does the same pick-and-place task one hundred times. If the components of the robot are not changed during this time, the action plans need only be computed once: before the first pick-and-place cycle. For every pick-and-place cycle afterwards, only the confidence scores need to be updated, which is far less computationally expensive than computing the action plans. The second reason is that the current version of RSIR has relatively few redundant components, thus the computation time for RSIR is likely to be reasonable.

The complete algorithm for computing action plans is included in the final version of RSIR.

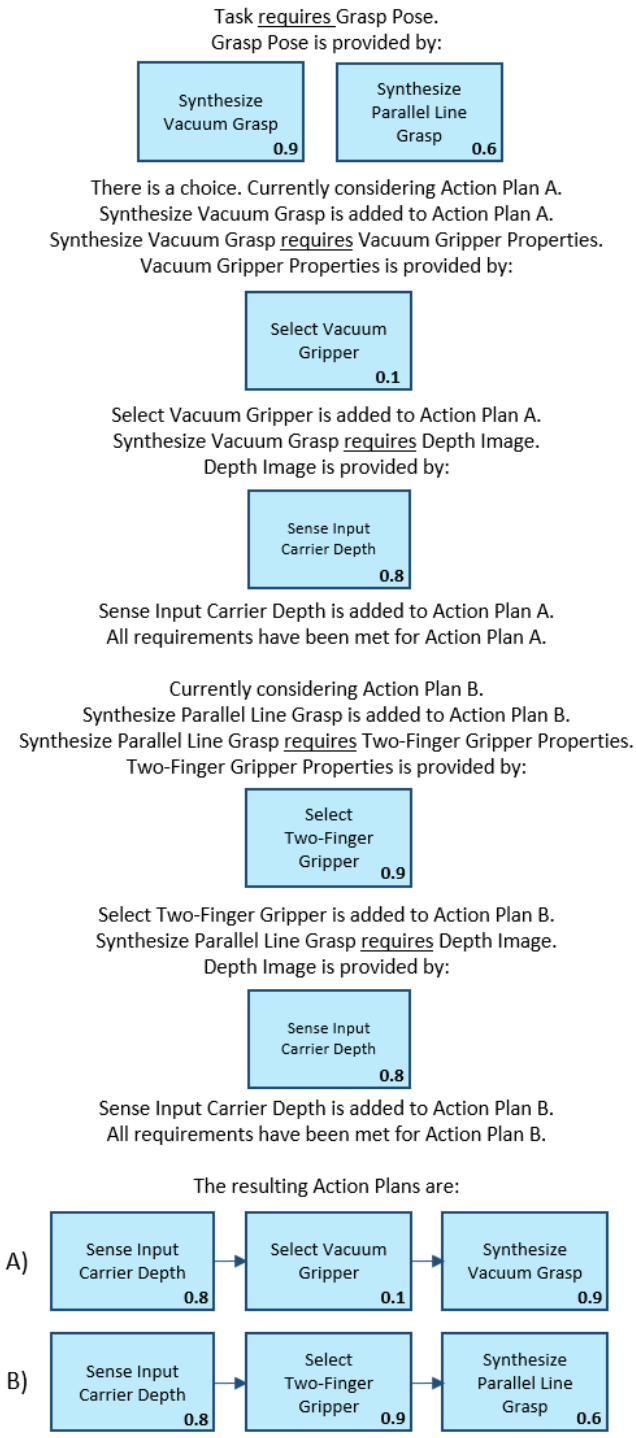


Figure 6.16: Steps of the complete algorithm for creating action plans.

6.3.3. Reasoning about Item Properties

The second aspect of RSIR that underwent iterations is the representation of item properties. To illustrate these iterations, consider an item of unknown mass and a robot with access to three grippers. Each gripper can handle a mass as shown in Figure 6.17. Let's assume that gripper 2 has a confidence score of 0.8 and the other grippers have a confidence score of 0.6. Let's furthermore assume that RSIR attempts a grasp with gripper 2 but fails. The conclusions RSIR can draw about the item's mass are the subject of this subsection.

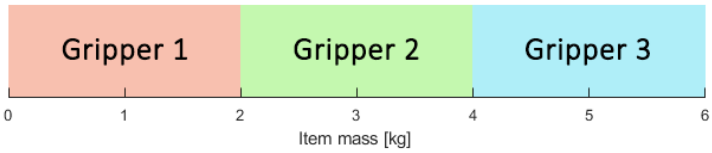
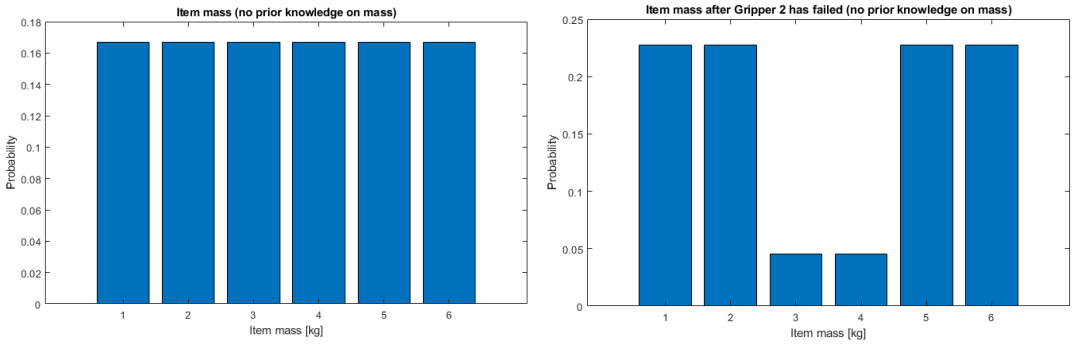


Figure 6.17: An example of three grippers, each capable of handling masses within a range of 2kg.

Iteration 1: Deterministic Properties
The first iteration of RSIR regarding item properties considers item properties as deterministic. Given the failure of gripper 2, RSIR concludes that the item is either lighter than 2kg or heavier than 4kg. Based on this conclusion, gripper 1 and gripper 3 are equally likely to succeed during the next attempt.

This type of reasoning is too simple for two reasons. The first reason is that it does not consider the probabilistic nature of the capabilities. Since gripper 2 has a confidence score of 0.8, there is a 20% chance that gripper 2 fails even if the item has a mass between 2 and 4kg. If this is indeed the case, RSIR incorrectly concludes that the item's mass is not within this range and henceforth selects the wrong grippers. The second reason is because this reasoning cannot include knowledge about the distribution of masses in the system. For example, it might be that there are more light items than heavy items in the system. Then it would be more reasonable to select gripper 1 over gripper 3. However, RSIR would still consider gripper 1 and gripper 3 equally good, because it does not consider this knowledge of mass distributions.

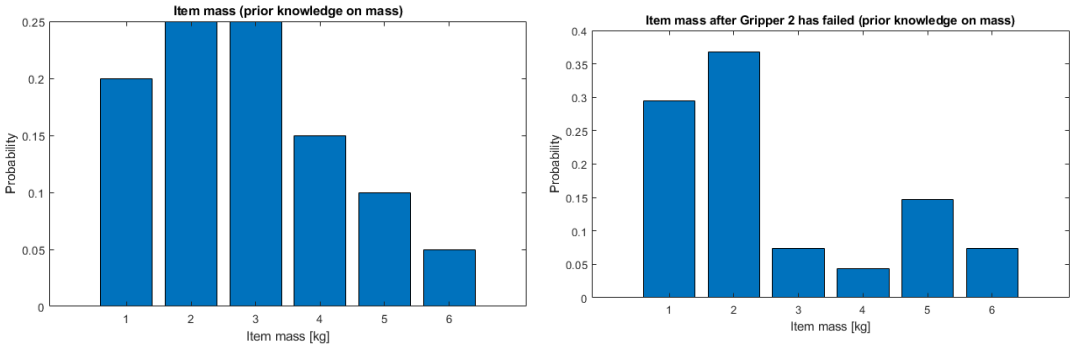
Iteration 2: Probabilistic Properties
The second iteration regarding item properties models properties as discrete probability distributions. Figure 6.18 shows this distribution before and after gripper 2 has failed. The latter distribution is computed using Bayes' rule. Note that the probability of the mass being between 3 and 4kg is non-zero. When using information about the mass distribution in the system, the item mass looks as shown in Figure 6.19. From the distribution after gripper 2 has failed it can be concluded that gripper 1 is more likely to succeed than gripper 3, as would be expected.



(a) The mass distribution before attempting gripper 2.

(b) The mass distribution after gripper 2 has failed.

Figure 6.18: Item mass modelled as a discrete probability distribution, using no knowledge of mass distributions in the system.



(a) The mass distribution before attempting gripper 2.

(b) The mass distribution after gripper 2 has failed.

Figure 6.19: Item mass modelled as a discrete probability distribution, using knowledge of mass distributions in the system.

This method of modelling item properties is used in the final version of RSIR. One important aspect in modelling continuous variables using discrete probability distributions is the resolution. In the examples of Figure 6.18 and 6.19, the resolution is 1kg. However, different resolutions might give different results.

6.3.4. Reasoning about Failures

The final aspect of RSIR that underwent iterations is the reasoning about failures. There are three iterations of this. To illustrate the iterations, consider the example when RSIR attempts to validate a successful grasp with the verification camera, but the verification algorithm concludes that there is no item in the gripper. RSIR's goal is then to determine the (most likely) cause of the failure and to create a new action plan that prevents this failure from occurring again. For example, the cause could be that the verification camera is inaccurate, or that the item cannot be handled by the gripper.

Iteration 1: Deterministic Reasoning
The first iteration of failure reasoning is based on the research into failures from section 6.1.1. When considering only the example failure, the scheme from section 6.1.1 is simplified into Figure 6.20. Based on tests done at Vanderlande, conditional probabilities can be computed or derived. For example, it can be derived from Figure 6.21 that P(Inaccurate verification|No item detected in gripper) = 1.16/16.7 = 0.07. When all conditional probabilities are known, RSIR can compute what the most likely cause is behind a failure. For example, the probability that the gripper could not handle the item is $0.8 \cdot 0.6 = 0.48$, and the probability that the depth camera was inaccurate is $0.8 \cdot 0.4 \cdot 0.7 \cdot 0.3 = 0.07$. Once the most likely cause has been found, RSIR selects an action plan that does not include the capability behind that cause. If this is not possible, RSIR moves on to the second most likely cause, then the third most likely cause, etcetera.

The main disadvantage of this approach is that modelling the system as shown in Figure 6.20 takes time and that and the model must be updated every time a capability is added to or removed from the system. Furthermore, the conditional probabilities can be difficult to obtain and are item-dependent.

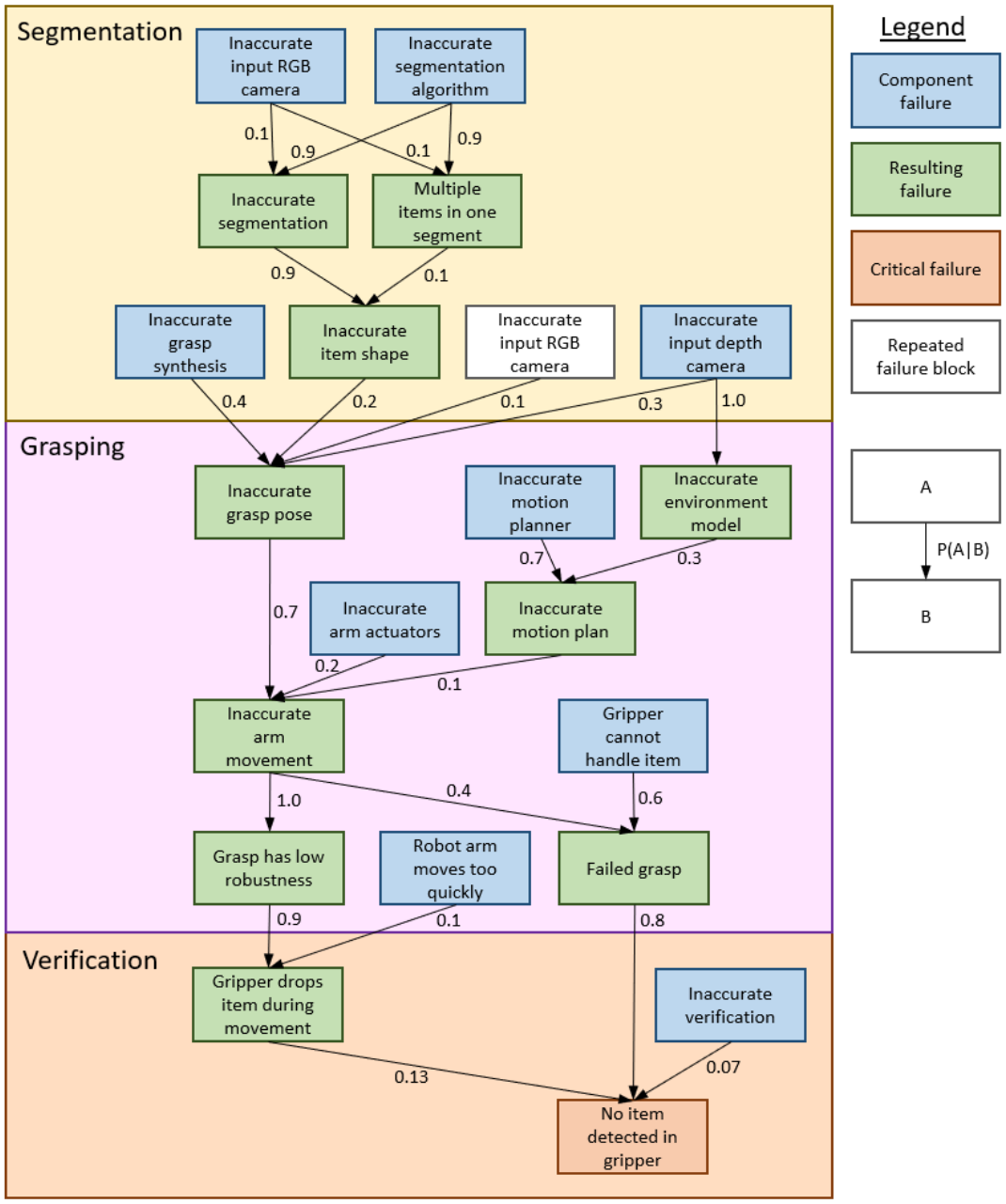


Figure 6.20: Causes for grasping failures that can occur in SIR. The "repeated failure blocks" are blocks that are added twice in the diagram for clarity. Values of probabilities are included only as an example.

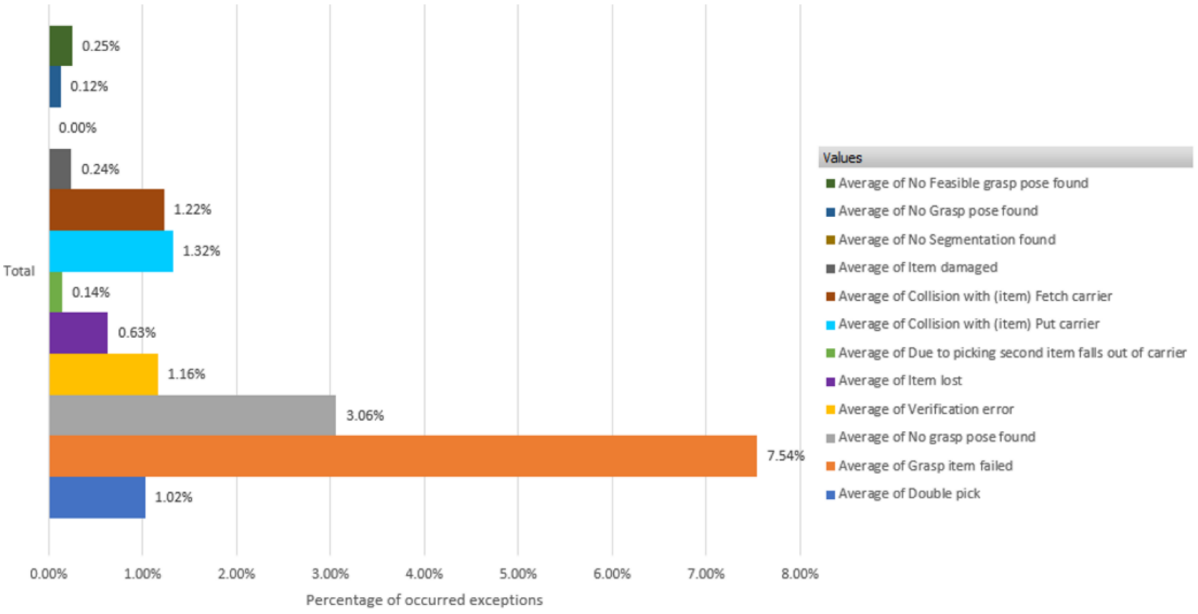


Figure 6.21: Failures observed for SIR [36]. Listed percentages are percentages of the total number of pick-and-place attempts performed.

Iteration 2: Action-based Probabilistic Reasoning

The second iteration attempts to make the probabilities item-dependent by including knowledge of the item and capability constraints. The item properties are modelled as probability distributions as explained previously. Furthermore, a capability can have a constraint per item property. This is modelled as shown in Figure 6.27, which uses confidences. For this example, the vacuum gripper has a 20% chance of succeeding if the item's mass is 5kg. The confidence score of the gripper for the mass is then $\Sigma c_i p_i$, where c_i is the confidence of the gripper for mass i and p_i is the value of the mass distribution for mass i. This confidence score is then added to Figure 6.20. For example, the probability that the gripper could not handle the item was previously computed to be 0.48. This probability is now combined with the confidence score of the capability. Assuming no prior knowledge of the mass, the confidence score of the gripper from Figure 6.27 is 0.65. Then the score of the gripper becomes $0.48/0.65 = 0.74$. The capability with the highest score is pointed out as the cause behind the failure.

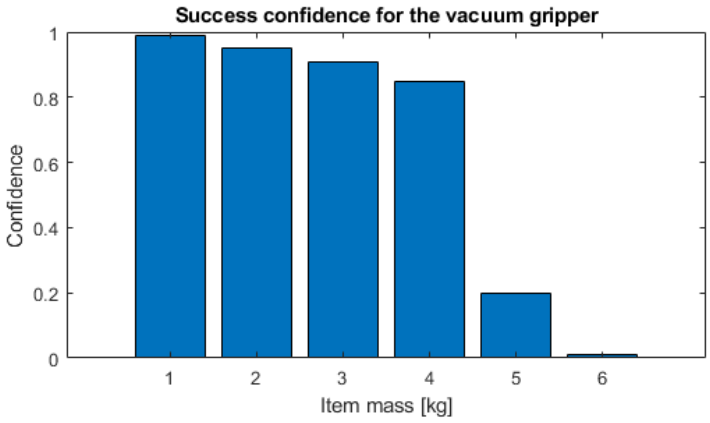
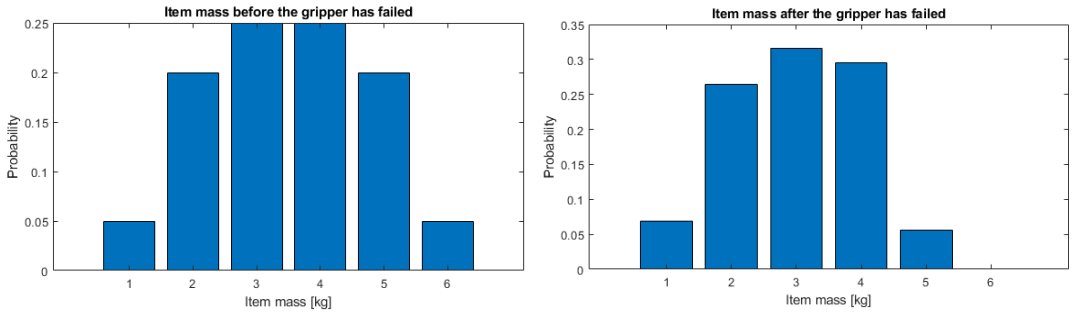


Figure 6.22: The constraints on item mass for the vacuum gripper, modelled as confidence values.

Furthermore, the item properties are updated according to the most likely cause. Let us assume that the most likely cause is the gripper. Using Bayes' rule, the most likely mass distribution can be derived. Figure 6.23 shows an example of how a mass distribution changes when Bayes' rule is applied using the gripper from Figure 6.27.



(a) The prior knowledge of the mass distribution.

(b) The mass distribution after the gripper has failed.

Figure 6.23: Item mass distribution before and after the gripper has failed.

The main advantage of this approach is that RSIR includes knowledge of the item when reasoning about the cause behind a failure. Capabilities that are less likely to succeed given the knowledge are more likely to be the cause. Additionally, the item properties are updated when RSIR decides what the cause behind a failure is.

This approach has two disadvantages. The first is that the system must still be modelled as shown in Figure 6.20, which must be updated every time a capability is added to or removed from the system.

The second disadvantage is that the cause behind a failure can be derived incorrectly. To illustrate this, consider a system that has the grasp synthesis approach in Figure 6.24 and no other synthesis approaches. Regardless of the item properties, this synthesis approach has a confidence of 0.5. This can be the case for example if the algorithm behind the approach was designed poorly. Consider now an item that is transparent, and the verification algorithm concludes that there is no item in the gripper. The cause behind this is that the verification camera cannot detect transparent items. However, RSIR does not know this. Following Figure 6.20, the score for the cause to be the synthesis is $0.48/0.5 = 0.96$, whereas the score for the inaccurate verification is $0.07/P$, with P being the confidence score of the verification camera. P would have to be smaller than 0.07 for RSIR to point out the verification camera as the cause behind the failure. This is very unlikely, since RSIR does not know that the item is transparent, and the verification is generally reliable. Thus RSIR points out the synthesis approach as the cause, and updates the item properties according to the limitations of the this approach. During the next attempt, the synthesis approach still has a confidence score of 0.5, and the cycle repeats. Thus RSIR never finds the true cause of the failure.

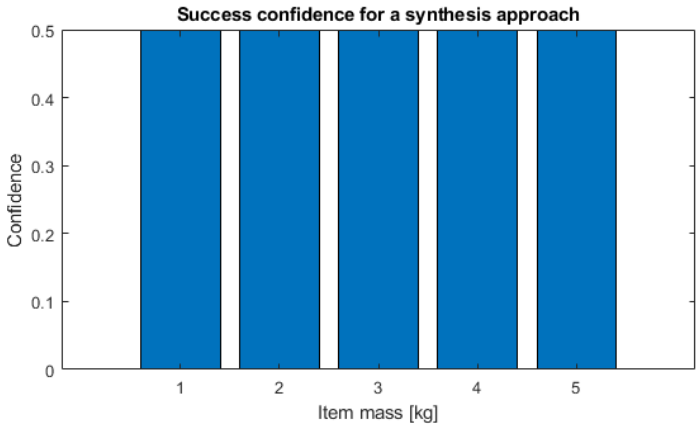


Figure 6.24: A capability that has a probability of success of 0.5, regardless of the item's mass.

Iteration 3: Plan-based Probabilistic Reasoning

The final iteration of reasoning about failures abandons the model from Figure 6.20. Instead, all actions that the robot has executed are entered into a probabilistic model. A Bayesian network is selected as this model, because it is relatively simple and because it is the logical next step from the Bayes' rule used in the previous iteration. The exact model is explained in more detail in the next section.

The advantage of this approach is that RSIR reasons about all actions it has performed. For the example of the transparent item, this means that RSIR mainly updates the item properties based on the limitations of the synthesis approach, but also slightly based on the limitations of the verification camera. After several attempts, RSIR will know that the item is likely transparent.

The abandoning of the model in Figure 6.20 is both an advantage and a disadvantage. A strong advantage is that the Bayesian network does not need to be updated manually if the robot's capabilities change. The disadvantage that the Bayesian network provides less insight into the failure modes. It provides a likelihood of failure per capability, but not to the level of detail that Figure 6.20 provides. However, the advantage outweighs the disadvantage because updating the model manually goes against the requirement that RSIR must be (easily) expandable to related tasks (R3). Hence the final version of RSIR makes use of a Bayesian network.

This concludes the iterations of RSIR. The next section explains the final version of RSIR's ontology and reasoning.

6.4. Resulting Ontology and Reasoner

This section provides the final version of RSIR's ontology (subsection 6.4.1) and reasoner (subsection 6.4.2). Readers that are unfamiliar with terminology in the field of ontologies are referred to Appendix C.1 for a brief introduction to the terminology.

6.4.1. Ontology

This subsection presents RSIR's ontology. Some parts of section 6.3.1 are repeated here for the sake of completeness. The ontology is visualized in Figure 6.25, with some details being visualized in Figure 6.26. The concepts are discussed below, but their formal definitions included in Appendix C.2. Note that all concepts in the ontology have the domain `http://www.semanticweb.org/vi#`, except when stated otherwise. The domain can be shortened to `vi:`, but is omitted in this section for clarity. Furthermore, the object properties present in Figure 6.25 are explained in more detail in Table 6.2.

RSIR's ontology starts with a **Robot**. A Robot is equipped with one or multiple **Devices**, where each Device provides one or multiple **Capabilities**. Each Capability is defined by its inputs and outputs (**Input/Output**), as well as its **Constraints**. The Robot must perform Capabilities one after another to complete a **Task**. A Task is defined by a required Output. An ordered list of Capabilities the Robot will execute is called an **Action Plan**. The concept of **Target Item** contains knowledge of the item.

As shown in Figure 6.26, Devices are split up into **Communicating**, **Knowing**, **Processing**, **Sensing**, and **Actuating** Devices. Capabilities are split up into the same categories. These categories are taken from CORA, with the exception of Knowing. This category was added to keep track of knowledge such as gripper properties, something that was not present in CORA. These classifications are purely for clarity and have no influence on any of the reasoning done by RSIR. Similarly, the Actuating, Knowing, and Processing Capabilities, as well as the Input/Output have been given subclasses for clarity.

Table 6.2: Object properties in RSIR's ontology.

Object property	Domain	Range	Explanation
abstractPart	ActionPlan	Robot ∪ Capability	Links an action plan to the list of actions in it, and the robot for which it is defined.
constraint	Capability	Constraint	Adds a constraint to a capability.
propertyConstraint	Capability	ItemProperty	Provides the limitations of a capability based on item properties.
input	Capability	Information	Defines the information/state a capability needs to be performed.
output	Capability	Information	Defines the information/state that completing a capability results in.
provides	Device	Capability	Defines what capabilities can be done using a device.
robotPart	Robot	Device	Could be split up into actuating, communicating, processing, knowing, and sensing parts.
taskOutput	Task	Information	Defines at what state or knowledge a task is complete.

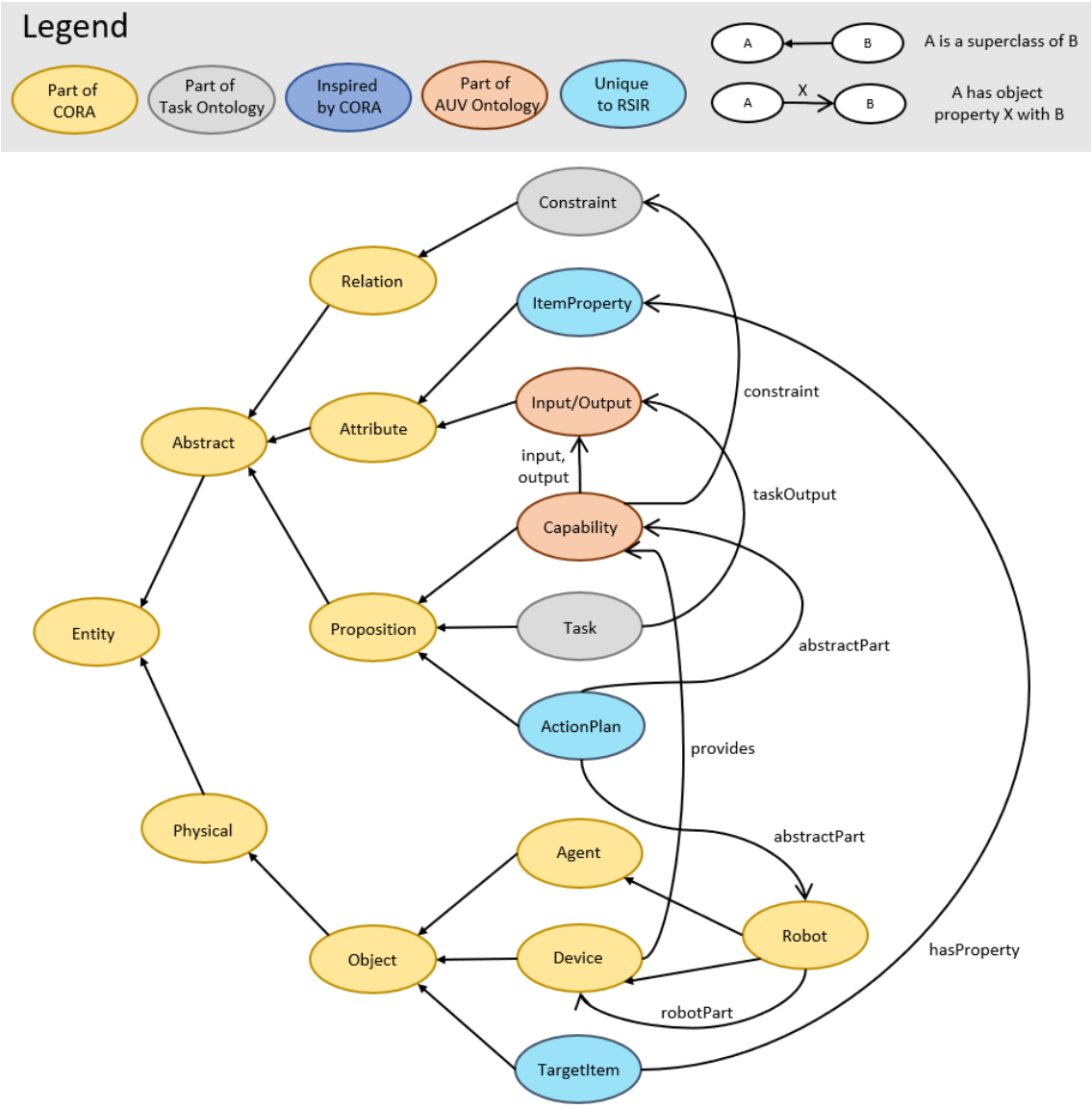


Figure 6.25: The final version of RSIR's ontology.

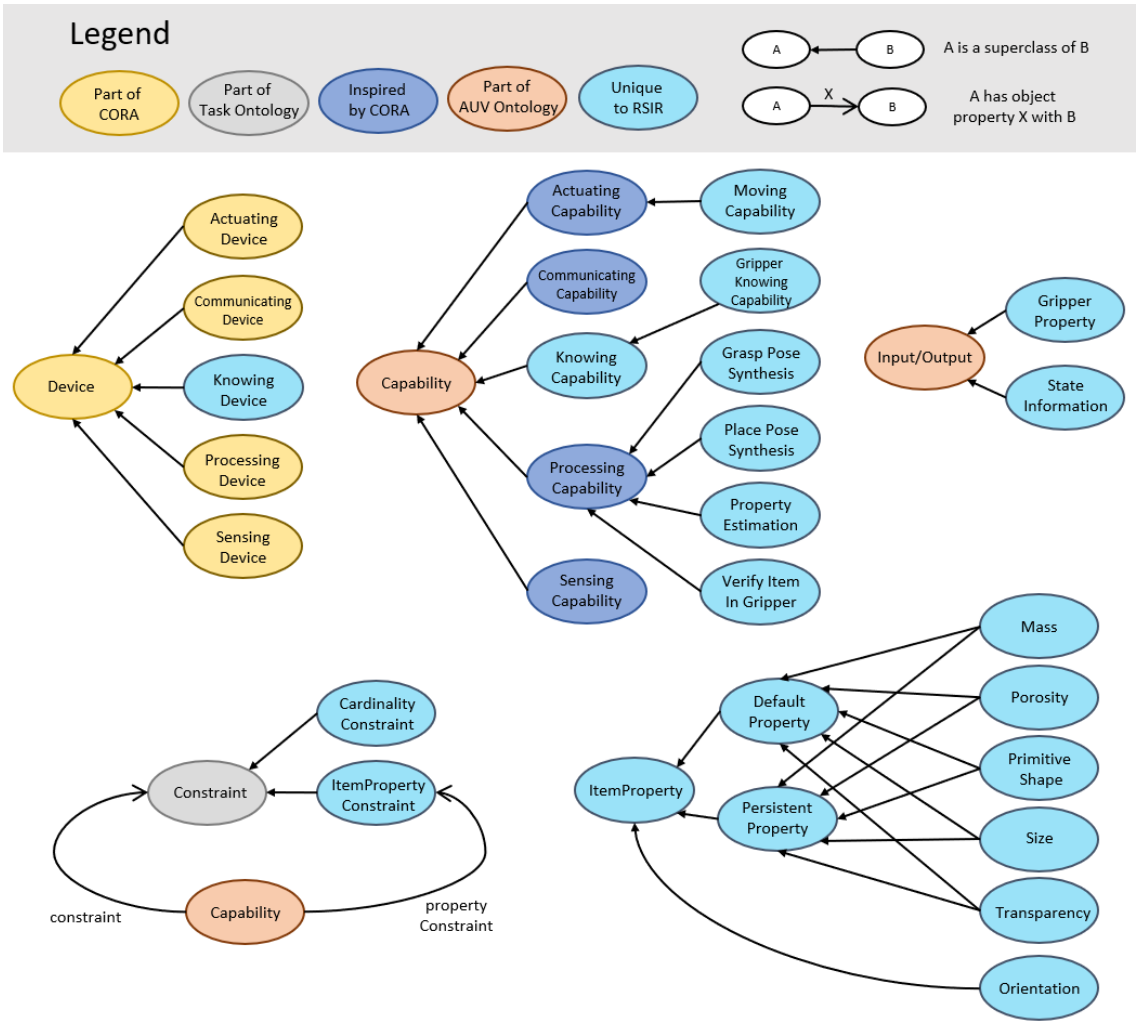


Figure 6.26: Details of the final version of RSIR's ontology.

Constraints are divided into two subclasses: Cardinality Constraints and Item Property Constraints. **Cardinality Constraints** define constraints regarding the number of capabilities of a certain type that can be used in an action plan. For example, there is a constraint that the system can only use one gripper during a pick-and-place cycle. Secondly, **Item Property Constraints** define the limitations of capabilities. For example, the limitations on item mass for a vacuum gripper are shown in Figure 6.27, where the vacuum gripper has a 20% chance of succeeding if the item's mass is 5kg. Limitations on Capabilities are assumed to be known and remain fixed unless they are changed manually.

Item Properties are split up into Default Properties and Persistent Properties. Any properties that inherit from **Default Property** are considered when reasoning about the item properties. Any properties that inherit from **Persistent Property** do not change between pick-and-place cycles. For example, Mass is a Persistent Property, whereas Orientation is not. Figure 6.26 shows six properties, but properties can be added, removed, and linked to Default Property as desired.

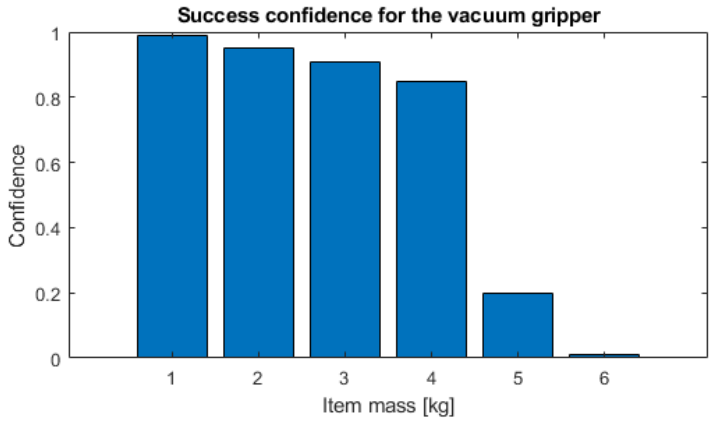


Figure 6.27: The limitations of a vacuum gripper on the item's mass, modelled as confidence probabilities.

In addition to the concepts and object properties in Figure 6.25, RSIR contains several data properties. They are visualized in Figure 6.28 and summarized in Table 6.3. Each Capability should have an **executionTime** and **defaultConfidence**. A Capability can optionally have an **assertedConfidence**, which is the confidence score computed from the item properties and the constraints of the Capability.

An ItemProperty must be defined by at least its **definedOnValue** and its **defaultPdfValue**. The definedOnValue defines the discretization of the property, for example, {1.0, 2.0, 3.0, 4.0} for mass (in kg), or {solid, semi-porous, porous} for porosity. The set of values can have any number of entries and supports both numbers and strings. The defaultPdfValue is the default discrete probability distribution of a property, which contains knowledge of the general distributions of properties in the system. Obviously, the distribution must have the same discretization as the definedOnValue. The **pdfValue** is the (non-default) distribution as computed by the Bayesian network. Finally, the **confidenceValue** contains the confidences for which an example is presented in Figure 6.27.

Table 6.3: Data properties in RSIR's ontology.

Data property	Domain	Range	Explanation
confidence	-	-	The category of confidences.
actionPlanConfidence	ActionPlan	xsd:decimal	Derived from constraints on the capabilities in an action plan and the target item properties.
assertedConfidence	Capability	xsd:decimal	Derived from constraints and target item properties.
defaultConfidence	Capability	xsd:decimal	Default confidence, when the specific limitations of the capability are unknown but there is still a general confidence that applies
executionTime	Capability	xsd:decimal	An estimate of how long executing the capability takes.
value	-	-	The category of values.
cardinalityValue	CardinalityConstraint	xsd:nonNegativeInteger	Defines a cardinality, in this case how many similar capabilities the robot can use at once.
confidenceValue	ItemProperty	xsd:string	Defines the confidence values of a capability being successful for an item with said property. String represents an array, e.g. "0.1 0.5 0.9".
definedOnValue	ItemProperty	xsd:string	Provides the definition for the property. String represents an array, e.g. "small medium large".
pdfValue	ItemProperty	xsd:string	Defines the discrete probability density function of an item property, using the definedOnValue as definition for the property.
defaultPdfValue	ItemProperty	xsd:string	The default version of the pdfValue. Item with unknown properties are give these values.

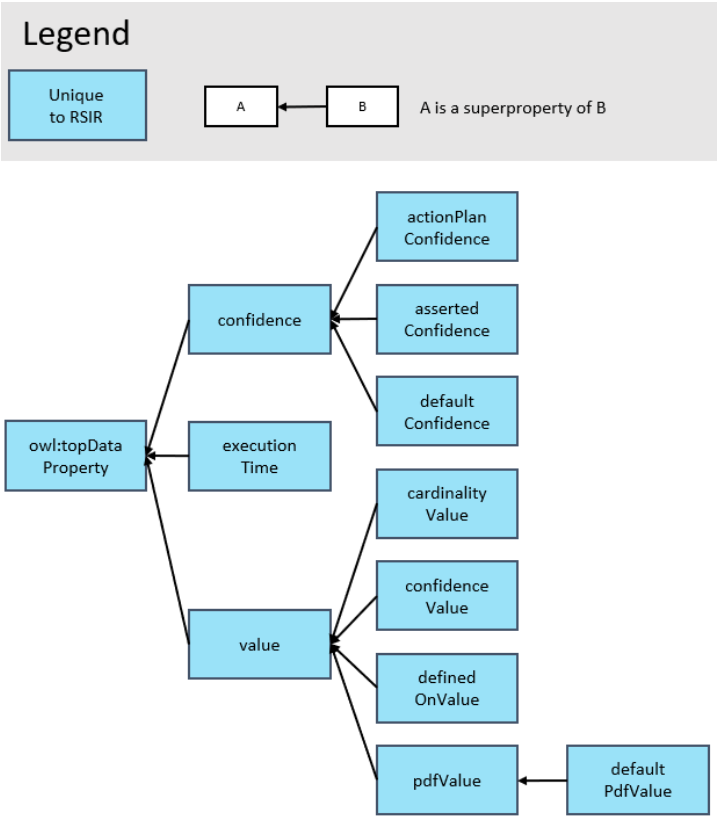


Figure 6.28: Hierarchy of RSIR's data properties.

6.4.2. Reasoner

The reasoning of RSIR contains three aspects: creating action plans, computing confidences and execution times, and deriving item properties. These three aspects are discussed in this subsection.

Action Plans

The goal of this reasoning is to convert a Task to an Action Plan. To do so, RSIR combines the capabilities a Robot has access to based on the output defined by the Task and the inputs and outputs of each Capability. RSIR finds all possible (non-redundant) combinations of Capabilities that complete the Task. Additionally, the cost is computed for each Action Plan according to a cost function. This is a function of the plan's confidence and execution time. An Action Plan's confidence is the product of the confidences of all the Capabilities that are in the plan. Similarly, an Action Plan's execution time is the sum of the execution times of all the Capabilities that are in the plan. The Action Plan with the lowest cost is selected for execution.

Capability Confidence and Execution Time

When the Action Plan's cost is being computed, a Capability's confidence is taken as its assertedConfidence, if this is defined. The assertedConfidence is computed from the knowledge of the item properties and the Capability's constraints using Equation 6.1. Here, $c_j(i)$ is the ith confidenceValue for property j, $p_j(i)$ is the item's i^{th} pdfValue for property j, n_j is the number of values defined for property j, and m is the number of properties for which the capability has constraints defined. Note that if no constraint is defined for a property, the capability is assumed to have a confidence of 1.0 with respect to that property, that is: $c_j(i) = 1 \forall i$.

$$\text{assertedConfidence} = \prod_{j=1}^{m} (\sum_{i=1}^{n_j} c_j(i) p_j(i))) \tag{6.1}$$

If no assertedConfidence is known (for example, if a Capability has no constraints), the defaultConfidence is used instead. If no defaultConfidence is known, RSIR assumes a confidence of 1.0 and outputs a warning. Similarly, if no executionTime is defined, RSIR assumes a large value and outputs a warning.

Deriving Item Properties

The reasoning that derives item properties uses the Bayesian network presented in Figure 6.29. The network includes all actions the Robot has executed until a failure was detected, excluding any actions that have no constraints. These are actions 1 through N. Excluding actions without constraints simplifies the network and thus increases the inference speed. The network also includes the properties that are constraints of one (or more) of the actions. These are properties 1 through M. Outcomes are defined as binary, with 1 denoting a successful result and 0 denoting a failure. The first step is to determine the outcome of an action due to one property. For example, the outcome of sensing the RGB image given the item's transparency distribution. An action has an outcome of 1 if and only if the outcome of that action due to a property is 1 for all properties. For example, if sensing the RGB image succeeds for the item's mass but not for the item's transparency, then sensing the RGB image fails. Similarly, the outcome of the complete action plan is 1 if and only if the outcome of all the actions is 1.

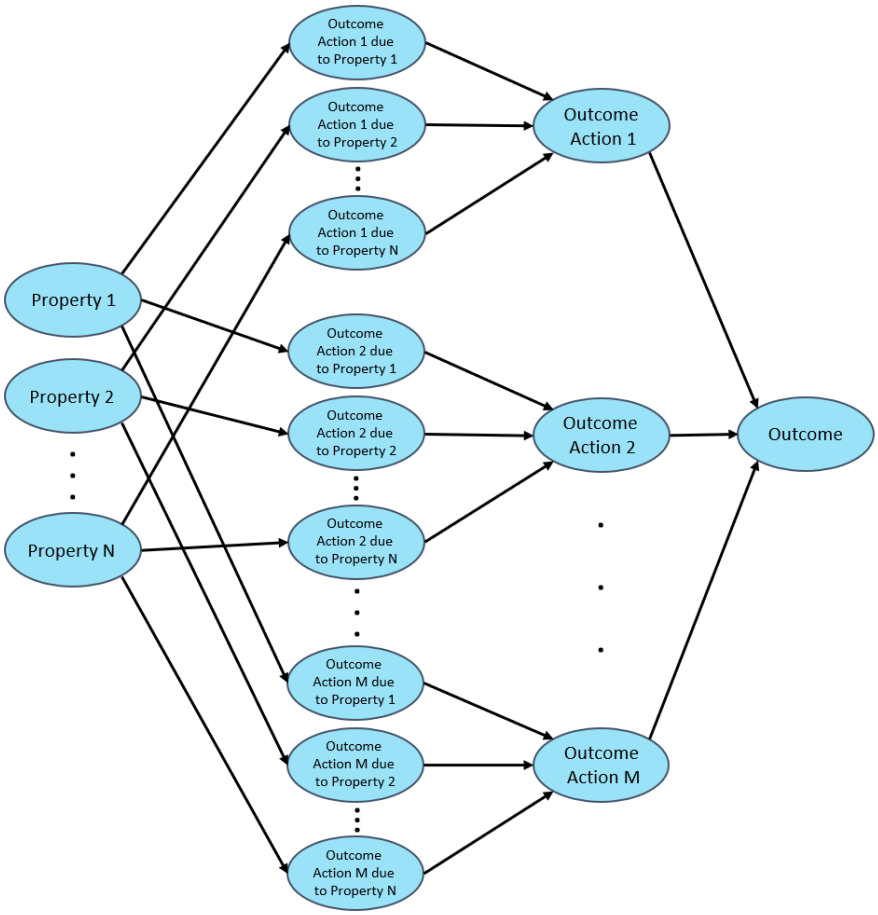


Figure 6.29: The Bayesian network used to infer item properties.

The added value of the Bayesian network comes from the fact that it can infer outcomes and item properties based on an observation. Given the currently known distributions of each property (pdfValue), the confidence for each action due to a property (confidenceValue), and the outcome of an action plan (1 or 0, i.e. success or failure), the network infers the most likely item property distributions and the likelihood of each action having failed. This way, a failure is used to gain knowledge of the item. When RSIR next creates an action plan, the capability confidences are computed according to the inferred item properties. Hence capabilities that likely caused a failure are unlikely to be selected again. Similarly, a successful execution of an action plan is used to gain knowledge of the item. This knowledge could be saved and loaded the next time the robot encounters that item.

This concludes the development of RSIR's ontology and reasoner. Appendix C.3 presents an example for a simple robot that uses RSIR for a grasping task. The next chapter discusses the implementation of RSIR.

7

Implementation

This chapter addresses the problem of implementing RSIR (RQ3). Section 7.1 explains the general software architecture. Afterwards, sections 7.2, 7.3 and 7.4 explain the implementations of the state machine, ontology, and reasoner, respectively. Using these implementations, the following chapters can evaluate RSIR's performance.

7.1. Software Architecture

This section explains RSIR's software architecture. RSIR is implemented on top of the Robotics Operating System (ROS), because it is *the* standard for robotics software [88] and because the components of SIR are already implemented in ROS. Thus implementing in ROS simplifies the implementation process (R1, R16). Figure 7.1 illustrates RSIR's software architecture for a grasping task. The main node is the **state_machine**. The state machine contains the information about the robot's state and sensory information (such as depth images). Additionally, the state machine queries the reasoner for action plans and converts the reasoner's response into low-level commands. For example, the action defined by the reasoner is `synthesizeBaumgartlGrasp`, but the state machine understands that this means calling the `/synthesis/baumgartl` service and saving the result into the `graspPose` variable.

The state machine communicates with the reasoner through the **rosprolog_service** node. This node listens for service calls and converts said calls into Prolog-understandable queries. This node contains the ontology and part of the reasoner. Additionally, the rosprolog_service node can call the Bayesian network to estimate the item properties after a failure has occurred, or after an action plan has been executed successfully. Note that although the Bayesian network is considered part of the reasoner, it is implemented as a separate node: **bayesian_network**. This is discussed in more detail in section 7.4.

Finally, there are the nodes that implement capabilities defined in the knowledge base. In the case of Figure 7.1 they are: gray_camera, stereo_camera, moveit_client, and baumgartl_synthesis. For example, the baumgartl_synthesis node implements the grasping approach developed in this thesis. Nodes that implement capabilities can be added, removed, and changed as desired by the engineer. However, it is important that there exist implementations for all capabilities that a robot can use (as defined in the ontology).

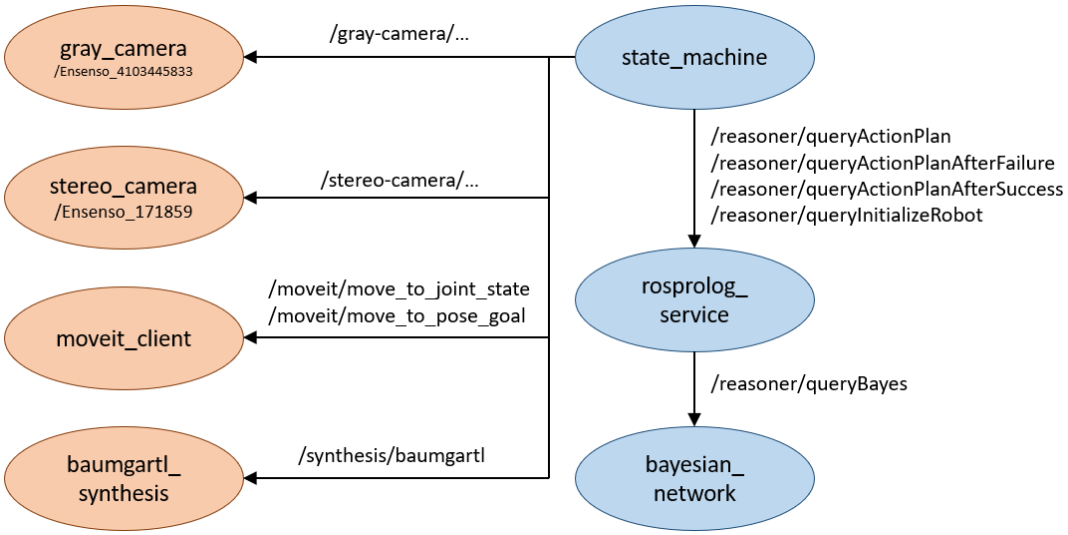


Figure 7.1: The software architecture used in RSIR. Displayed are ROS nodes and services. The nodes on the right are always present in RSIR, whereas the nodes on the left are specific to an application and can be changed as desired.

7.2. State Machine

This section explains the state machine. The state machine is the core of RSIR. It connects the output of the reasoner to the ROS implementations of capabilities, and detects any failures that occur. It is implemented using Smach because Smach provides a simple implementation for state machines in Python, provides a data structure (`userdata`) to store the inputs and outputs of capabilities, and is already in use by Vanderlande. The states in RSIR's state machine are shown in Figure 7.2. Their functions are as follows:

- **Init** initializes the knowledge base and moves the robot to its initial position. The former handles aspects such as resetting asserted item properties to their default values, reactivating components that were concluded to be in failure, and computing the confidences of components given the target item.
- **GetPlan** queries the reasoner for an action plan. The state machine saves this action plan as a list of strings. For example, getting a segmentation of the input carrier can be done using the action plan: `["senseInputCarrierRGB", "segmentInputCarrier"]`.
- **Task** executes a task, in the order as tasks appear in the action plan. Tasks are implemented by methods `execute_X`, where `X` is the name of the capability as it appears in the knowledge base. For example, the method `execute_senseInputRGB` implements the capability that is defined as `senseInputRGB` in the knowledge base. If the task is executed successfully, the system moves on to the next task in the action plan.
- **Failure** is reached when a task execution fails. This state calls the reasoner to derive the item properties from the failure, and queries the reasoner for a new action plan.
- **Finish** is reached when RSIR has successfully executed the last action in its action plan. It calls the reasoner to derive the item properties based on the executed action plan. Furthermore, it asks the user whether the task should be repeated for a new item.
- **Terminate** cleans up any processes before RSIR exits, such as closing the cameras.

The outputs of the tasks are saved in the `userdata` struct using the names defined in the knowledge base. For example, the grasp pose computed by a synthesis algorithm is saved in `userdata.graspPose`. At compile time, Smach requires a list of properties that `userdata` can have. Hence this list is hard-coded, and must be updated every time a new type of input/output is added to the knowledge base. To facilitate this, one can run Rosprolog and use the query `all_smach_keys(Keys)`. The output of this query can directly be pasted into the script for the state machine.

Unfortunately, Smach's `userdata` only supports basic data types, such as integers, arrays, and strings. This means that inputs and outputs that are complex or structured cannot be accessed via this struct. As a

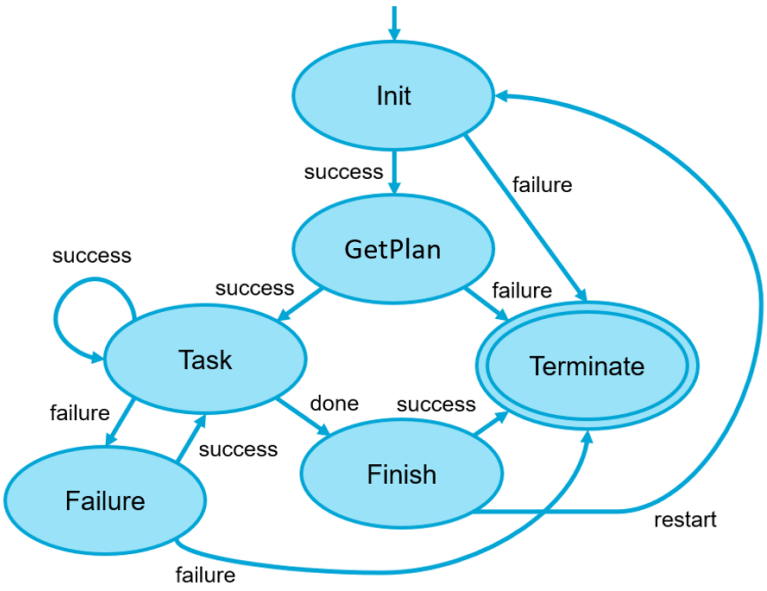


Figure 7.2: The state machine used in RSIR.

result, it not possible to include RGB images, segmentation masks, and point clouds in the `userdata`, and they were defined as global variables instead.

7.3. Ontology

This section addresses the problem of implementing RSIR's ontology, that is, it handles the implementation phase in Methontology. Decisions made in this phase consider the ontology language, reasoner, and tools used.

Recall that the common trade-off in selecting an ontology language is between expressiveness and decidability. The most prominent ontology language is OWL 2, which is divided into several sub-languages as shown in Figure 7.3. The sub-languages limit expressiveness but improve decidability. For example, ontologies using features outside of OWL 2 DL are undecidable. OWL 2 is split up further into profiles, examples of which are OWL 2 Lite, OWL 2 QL (Query Language), and OWL 2 RL (Rule Language) [87].

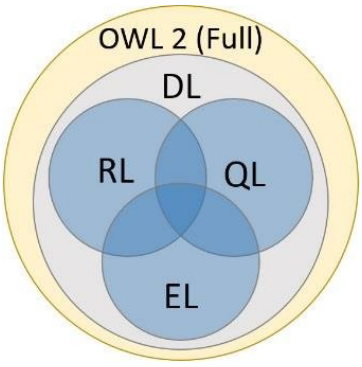


Figure 7.3: Venn diagram showing the relation between the OWL 2 profiles [32].

There are some languages that can be used to expand OWL 2, such as SWRL and SHACL [65][101]. The use of these languages allows to define rules. For example, if it is known that person A is the parent of person B and person A has a brother that is person C, then SWRL rules can conclude that person B has an uncle that is person C [101]. These types of rules cannot be defined in OWL 2. However, the use of such rules also makes the ontology less decidable [101].

Another important fact regarding the ontology languages is that KnowRob is implemented in OWL 2 DL

[87]. Even though RSIR's ontology is not integrated with the KnowRob ontology, KnowRob still provides useful functionalities. The main functionality is that KnowRob provides an implementation for using ontologies with ROS. Furthermore, current work on KnowRob considers the robot learning from actions demonstrated by humans in virtual reality, or learning from instruction videos on the internet [19]. Although currently immature and irrelevant, this research might become relevant in the future if RSIR is applied in tasks it is not familiar with (R3).

Another aspect of KnowRob is that it uses the standardized language SWI Prolog (henceforth referred to as Prolog) to load, save, and query for knowledge saved in its ontologies [20][102][130]. According to Pangercic et al., Prolog "combines fast inference and computation with declarative, logics-based semantics" [102]. The fast inference and logics-based semantics are promising for RSIR (R2, R14). It should be noted that Prolog goes beyond simple OWL reasoning [87] and the reasoning designed in Prolog is saved in a Prolog file, rather than in the ontology file.

When considering these advantages, it is promising to implement RSIR in KnowRob. Additionally, the use of Prolog eliminates the need for languages such as SWRL and SHACL, because Prolog proved capable of implementing most of the reasoning done by RSIR over the course of this thesis. However, implementing in KnowRob comes with three disadvantages. The first is that RSIR's ontology is not integrated with the KnowRob ontology. This makes it difficult to use some of KnowRob's functionalities with RSIR. However, this is not a problem at this point in time because RSIR does not make use of most of KnowRob's functionalities, and approaches exist for mapping concepts between ontologies [42]. This is, however, the second disadvantage: the fact that most of KnowRob's functionalities are unused by RSIR. Although they are unused, they could be taking up computational resources, thus decreasing performance (R14). This is especially relevant because KnowRob is shown to be slow in demonstrations [3]. The final disadvantage is that KnowRob's documentation is outdated, and thus it might be time-consuming to get the system to work correctly.

Despite these disadvantages, RSIR is implemented in KnowRob because of the advantages it brings. The evaluation of RSIR must determine whether its performance is sufficient despite KnowRob's unused functionality possibly using up computational resources.

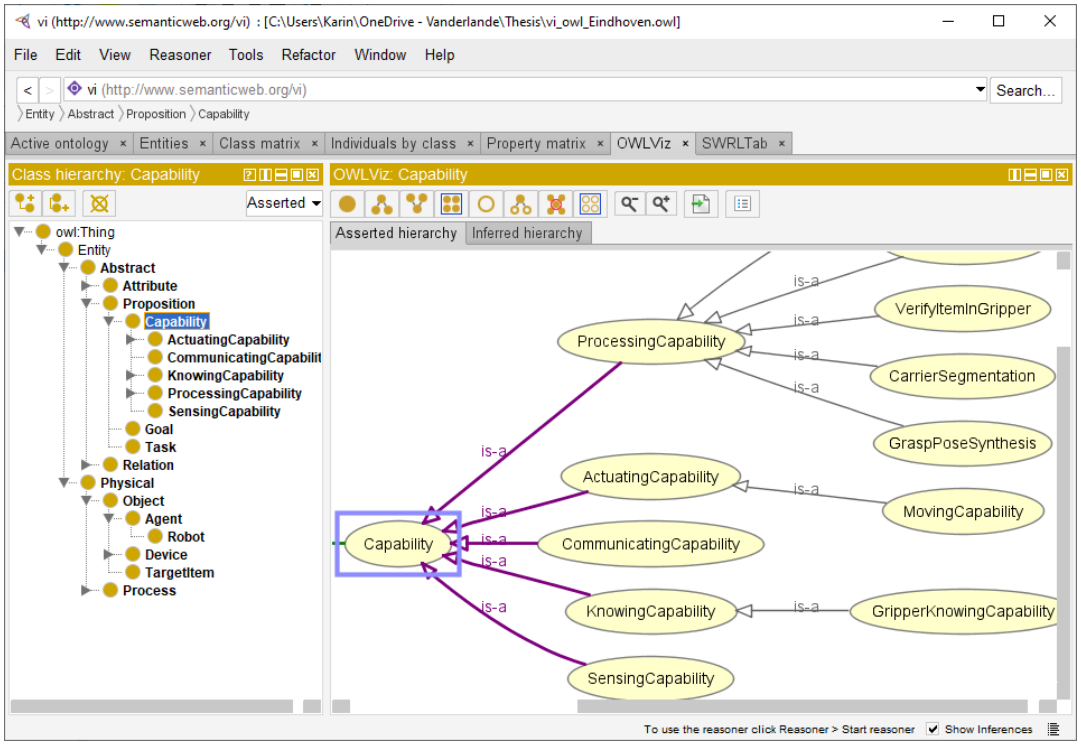


Figure 7.4: A part of RSIR's ontology visualized in Protégé.

Finally, Protégé is used to aid in implementing the ontology. Protégé is a semantic editor that helps in constructing, visualizing, and debugging ontologies [49][94][137]. Figure 7.4 shows RSIR's ontology being edited

in Protégé. This is more user-friendly than constructing ontologies directly in XML (or similar) files [98]. Furthermore, Protégé comes with the built-in reasoner HermiT. This reasoner detects any inconsistencies in the ontology and thus helps in debugging. Note that this reasoner is not the "reasoner" RSIR makes use of to, for example, create action plans. HermiT is only used at design time. The resulting ontology is saved as an XML file such that it can be loaded by Prolog at runtime. More about the reasoning is discussed in the next section.

7.4. Reasoner

The previous section considered the implementation for the ontology. However, an ontology is only a set of facts, and thus a reasoner is required to draw conclusions from those facts. Following the decisions in the previous section, this section explains how the reasoner is implemented in Prolog and Python.

Prolog is a logic programming language. The logic programming paradigm is vastly different from the more commonly-used paradigms that contain languages such as Python, C++, and Matlab [100]. Because of this, Prolog is well-suited for deriving conclusions from the facts saved in an ontology. However, it also means that Prolog is not as well-suited to mathematical computations as languages such as Python are. Hence implementing the Bayesian network in Prolog is not ideal.

Two approaches can solve this problem. The first approach is to expand Prolog with the capabilities required to easily create Bayesian networks. ProbLog claims to do just this [74]. However, a quick test in their online editor shows that ProbLog does not accept variables as probabilities. This is critical, because the item probability distributions and capability confidences must be passed from the ontology and thus cannot be hard-coded in ProbLog. The second approach is to implement the Bayesian network in another programming language, such as Python. The disadvantage of this is that the reasoner then consists of two files: the Prolog file and the Python file. This requires communication between the two languages, and makes it less clear to the user which file is responsible for which part of the reasoning.

Because ProbLog is not an option, the Bayesian network is implemented in a language other than Prolog. Python is selected for this language, because of its user-friendly syntax, integration with ROS, and integration with pomegranate. Pomegranate is a well-documented, easy-to-use package that implements a large variety of probabilistic models, including Bayesian networks [117]. RSIR's Bayesian network is thus implemented in pomegranate. It requires the relevant item properties (as specified in the ontology), the executed action plan, and the result of the action plan (success or failure). It outputs the likelihood of each action having failed, as well as the most likely distribution per item property. Regarding the inputs, the network supports any number of item properties and any number of actions in the action plan. Additionally, it supports any definition of item properties, as long as the entries are unique. For example, mass can be defined as `[1,2,3]`, `[0.1,0.2,0.3,0.4,0.5,0.6]`, or `["light","medium","heavy"]`.

7.5. Grasp Synthesis

The grasp synthesis approach is implemented in Matlab because of Matlab's built-in edge and line detection algorithms, its ease of use with point clouds and images, and its familiarity. Matlab can communicate over ROS using the ROSmatlab toolbox, and custom ROS messages and services can be defined by using an add-on [132]. Appendix D.1 presents the steps of the algorithm in full detail.

8

Grasping Approach Evaluation

This chapter addresses the problem of evaluating the grasp synthesis approach that was selected and improved upon in chapter 5. The insight this evaluation gives is also useful for implementing the approach into RSIR. Recall that this evaluation is not meant to be a full test of the gripper and synthesis approach, as such an evaluation is outside the scope of this thesis (R34). Instead, this section provides a general idea of the limitations of the grasp synthesis approach and a starting point for future work. Specifically, section 8.1 investigates the influence of the approach's parameters, section 8.2 evaluates the general performance of the synthesis approach (R13, R24, R31), and section 8.3 evaluates the approach when it uses segmentations (R25).

All evaluations (with the exception of the evaluation of the parameters) are done based on the depth data from 20 input carriers obtained from the set-up in Eindhoven (R26, R32). The synthesis approach is evaluated based on the variety of items it can synthesize reasonable grasps for (R23), the quality of the grasp the algorithm concludes to be the best grasp (R24), and the time it takes to synthesize a grasp (R14). Whether a grasp is reasonable or not is determined by a human.



Figure 8.1: The items used in evaluating the grasp synthesis approach.

Each carrier differs in the items it contains. To get a good overview of what items the synthesis approach can handle (R32), the following item properties are varied: shape, size, texture, deformability, level of clutter, reflectivity, transparency, and colour. The former five are the main properties synthesis approaches are evaluated on in literature [135]. Usually this also includes mass, but mass is meaningless in this evaluation because only the synthesis algorithm is evaluated, whereas mass is a limitation for the gripper hardware. Reflectivity and transparency are limitations of the depth camera and can thus hinder the synthesis approach. Colour is currently not relevant but might be in future work (R3), when edges are detected based on the RGB image.

Figure 8.1 shows the set of items considered in the evaluation. Note that this set of items is not purely selected based on what RSIR's vacuum gripper cannot handle. It can be useful in practice if the fingered gripper can handle items that the vacuum gripper can also handle, as this would require less switching between grippers and would thus save time (R6, R14). Hence some items that the vacuum gripper can handle are included in this evaluation.

Several aspects of the approach were kept constant to limit the scope of the evaluation. The paired lines were not filtered based on gripper length, minimum gripper opening width, or maximum opening width. This is to make the evaluation less dependent on the specific gripper it is used with (R3). For the same reason, the maximum grasping height was not considered. Additionally, the metric used in all tests is enclosed depth.

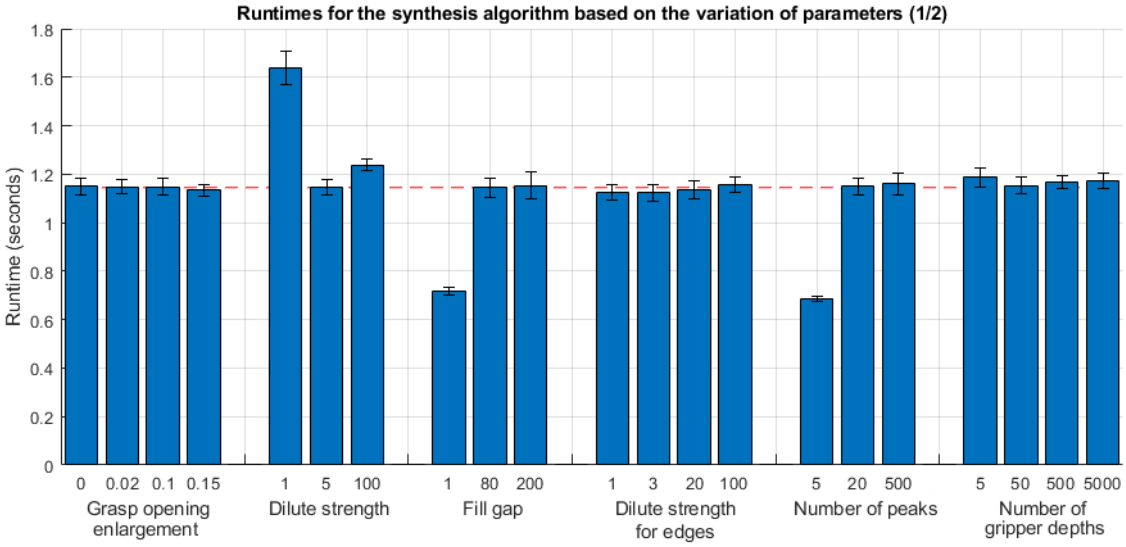
8.1. Tuning Parameters

This section investigates the effect of the tuning parameters of the synthesis algorithm. Insight in these effects allow for better tuning of the algorithm, which in turn could result in a larger variety of items being grasped (R5, R23) or a decrease in runtime (R14). There are nine parameters that can be tuned. They are summarized in Table 8.1. The full evaluation, including examples, is included in Appendix D.2. The runtimes resulting from the evaluation in the appendix are presented in Figure 8.2.

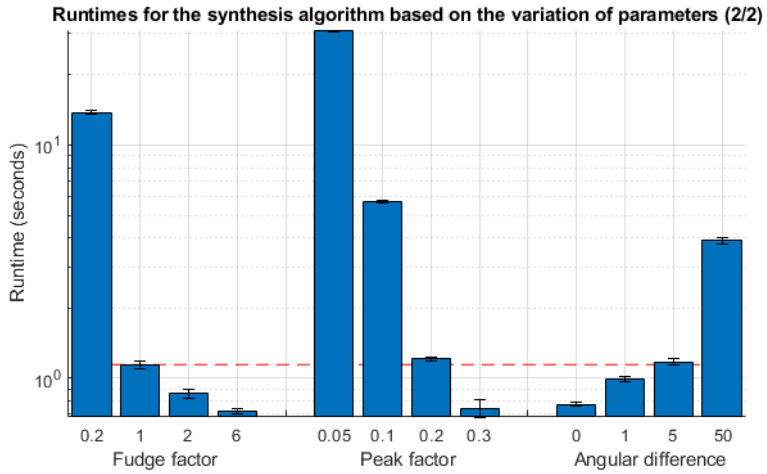
In summary, the parameters that most influence the resulting grasps and runtime are the fudge factor, peak factor, and angular difference. These parameters influence the edge detections, line detections, and line pairs, respectively. The tuning of these parameters comes down to a trade-off between runtime and the number of grasps found. The number of peaks could be tuned alongside the peak factor to limit the runtime.

Table 8.1: Tuning parameters for the grasp synthesis approach.

Parameter name	**Matlab name**	**Interpretation**	**Unit**	**Effect**
Grasp opening enlargement	enlargeGraspOpening	Extra amount by which the gripper is opened before grasping	metres	Prevents collisions between the gripper and the item
Dilute strength	dilStrength	Strength with which to dilute the segmentation mask	pixels	Compensates for missing parts of the segmentation
Fudge factor	fudgeFactor	Threshold for edge detections	-	Creates a trade-off between the number of detected edges and the runtime
Fill gap	fillGap	Maximum distance between two detected lines to merge them into one	pixels	Merges multiple detected lines into one
Dilute strength for edges	dilStrengthEdge	Strength with which to dilute the edge detections	pixels	Strengthens detected edges
Number of peaks	nPeaks	Maximum number of Hough peaks to identify	-	Limits the number of detected lines
Peak factor	facPeak	Minimum value to be considered a Hough peak, defined as a fraction	-	Creates a trade-off between the number of detected lines and the runtime
Angular difference	dTheta	Similarity measure for finding line pairs	degrees	Allows to grasp non-parallel edges, but increases runtime
Number of gripper depths	nGripperDepths	Number of points to check for determining a line's depth	-	Determines accuracy when determining a line's height



(a) Parameters that have little influence on the runtime.



(b) Parameters that greatly influence the runtime. Note the logarithmic vertical axis.

Figure 8.2: Runtimes for the synthesis approach when evaluating the effect of the parameters. When one parameter is changed, the others are kept constant. The dashed red line indicates the average runtime over all runs of the algorithm.

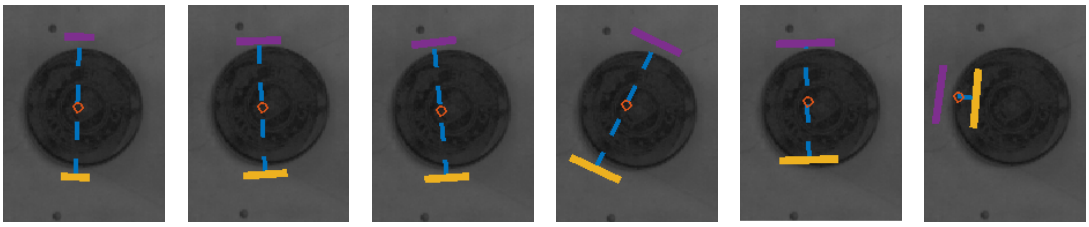
8.2. General Performance

This section evaluates the general performance of the synthesis approach. The type of evaluation is exploratory rather than descriptive. A descriptive evaluation of the synthesis approach is not useful at this stage, because too little is known about the competences of the synthesis approach. For example, it might be that one set of parameters is exceptionally good at grasping circular items, whereas another set of parameters is exceptionally good at grasping in clutter. If this is the case, it is wise to implement three versions of the synthesis algorithm in RSIR: one for circular items, one for items in clutter, and one for all other cases. Because of RSIR's reasoning capabilities, it could select the best version of the synthesis approach based on what it knows about the item. However, at this point in time, it is unknown which sets of parameters result in which competences. Hence this evaluation explores what items the approach can handle, and which sets of parameters work for which items.

8.2.1. Method

Grasps are synthesized for the 20 input carriers, and the three highest-ranking, qualitatively different grasps are saved (R23, R24). The approach is run several times with different parameters until a reasonably successful grasp is found, while keeping the runtime short (R14). The runtime is the average time measured using Matlab's `tic toc` function during five runs of the algorithm. Qualitatively different grasps are defined as il-

lustrated in Figure 8.3, which shows the top six synthesized grasps for a jar of peanut butter. Four of these grasps are similar to each other and thus there are only three qualitatively different grasps among the six grasps.



(a) Grasp 1. (b) Grasp 2. (c) Grasp 3. (d) Grasp 4. (e) Grasp 5. (f) Grasp 6.

Figure 8.3: The top six highest-scoring grasps for a jar of peanut butter.

To get some insight into how the item properties influence the grasp synthesis algorithm, the correlation coefficient between the item properties and the parameters is determined, as well as the correlation coefficient between the item properties and the grasp score and runtime. This insight allows better tuning of the parameters to grasp a larger variety of items (R5, R23) or achieve a shorter runtime (R14). Only parameters that vary between carriers and only the highest-ranking grasp in each carrier is used for this analysis. The item properties used are number of curves, size, clutter, reflectivity, and transparency. Other properties such as mass and shape are not included because they are irrelevant (in the case of mass) or because the item test set is too small (in the case of shape). Pearson's correlation coefficient is used with a significance level of 0.1.

Finally, the artificial segmentation shown in Figure 8.4 is passed to the algorithm, henceforth referred to as the "box segmentation". This prevents the algorithm from synthesizing grasps on the carrier, yet tests its performance when not provided with a segmentation of the items in the carrier (R31). Without the box segmentation, the algorithm considers the edges of the carrier promising locations for grasps, which is undesirable.

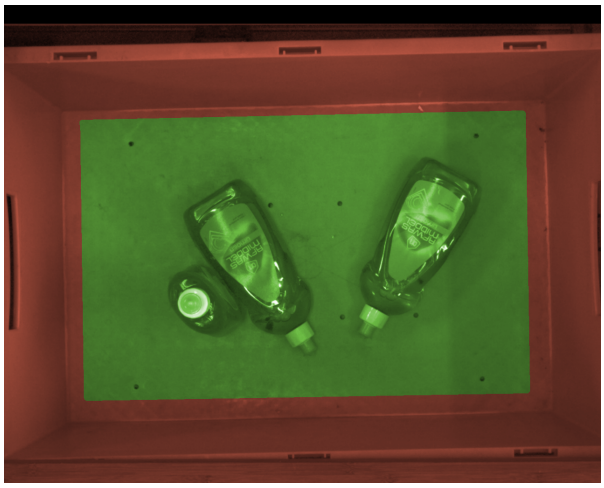


Figure 8.4: The segmentation as used in the evaluation of the synthesis approach's general performance. The green area is part of the segmentation mask, whereas the red area is not.

8.2.2. Results

Figure 8.5 shows some of the highest-scoring grasps synthesized by the algorithm. The average grasp score of the highest-scoring grasps is 0.74 with a standard deviation of 0.17. Furthermore, Figure 8.6 summarizes the runtime, which is 1.67 seconds on average with a standard deviation of 0.79. Finally, Table 8.2 presents the correlation coefficients. More results are included in Appendix E.1, including the depth data, detected edges and lines, and all resulting grasps.

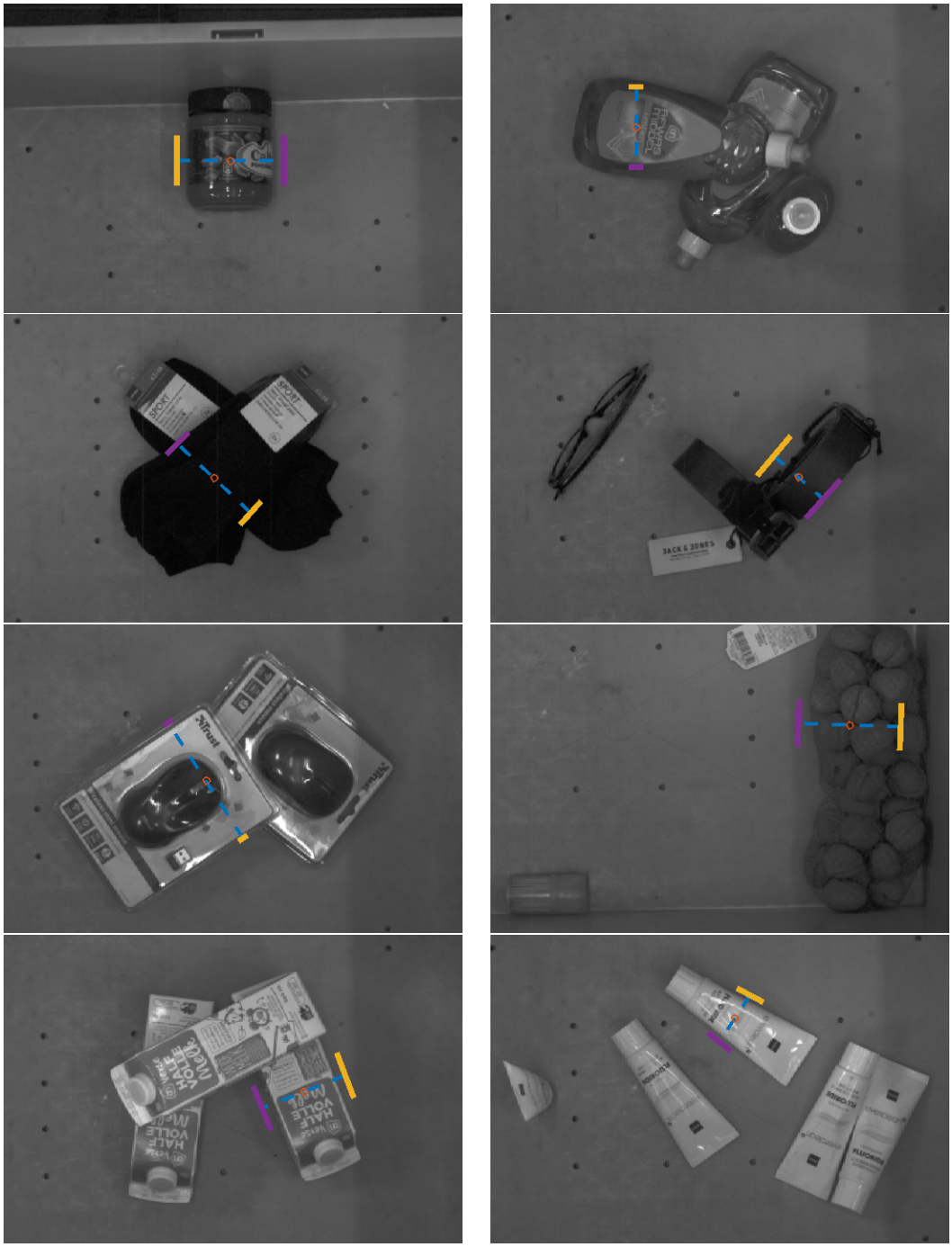


Figure 8.5: Some of the highest-scoring grasps synthesized by the algorithm when using the box segmentation.

Table 8.2: Correlation between item properties, parameters of the grasp synthesis algorithm, and results. The values are listed as ρ(p), where ρ is the correlation coefficient and p is the p-value. Statistically significant results are highlighted in green.

	Fudge factor	Peak factor	Angular difference	Max grasp score	Runtime
Curves	-0.25 (0.29)	-0.65 (0.00)	-0.37 (0.11)	-0.04 (0.88)	0.31 (0.19)
Size	0.34 (0.14)	0.35 (0.13)	-0.18 (0.45)	-0.35 (0.13)	-0.05 (0.84)
Clutter	0.14 (0.57)	0.39 (0.09)	0.19 (0.45)	0.20 (0.41)	-0.16 (0.52)
Reflectivity	0.19 (0.42)	-0.26 (0.26)	-0.28 (0.23)	0.07 (0.77)	0.29 (0.22)
Transparency	0.26 (0.27)	-0.31 (0.19)	-0.19 (0.42)	0.00 (0.99)	0.25 (0.28)

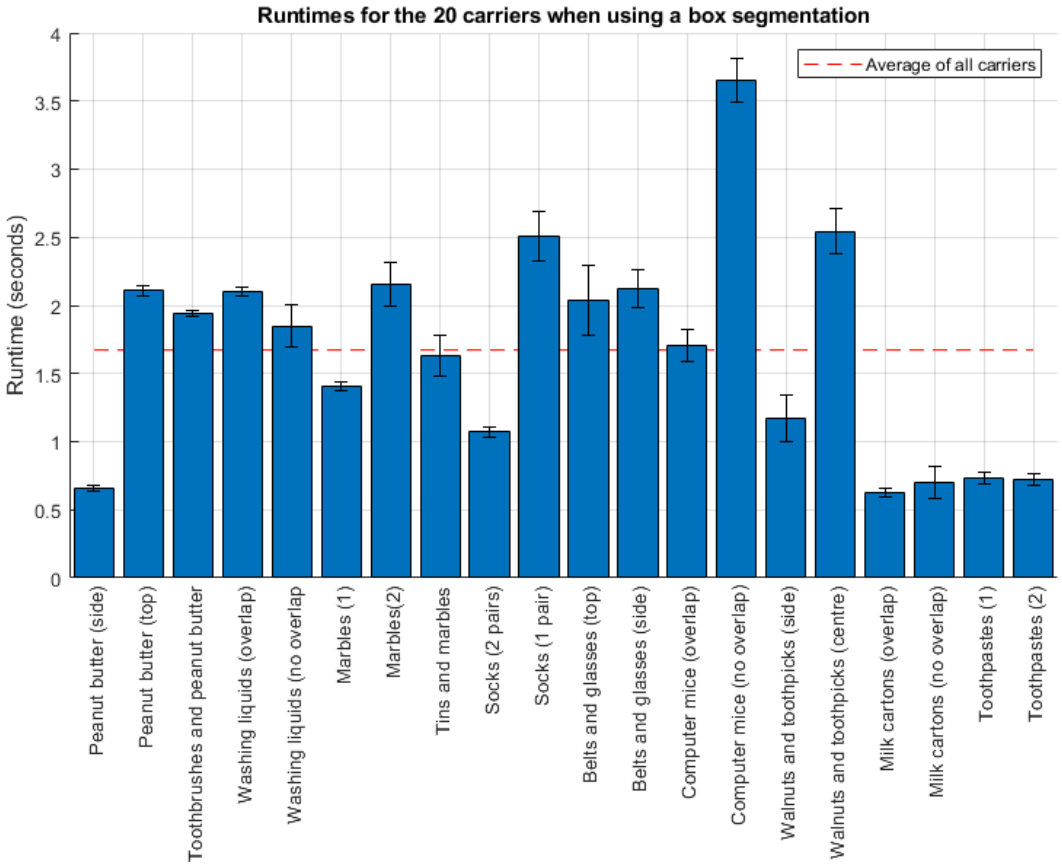
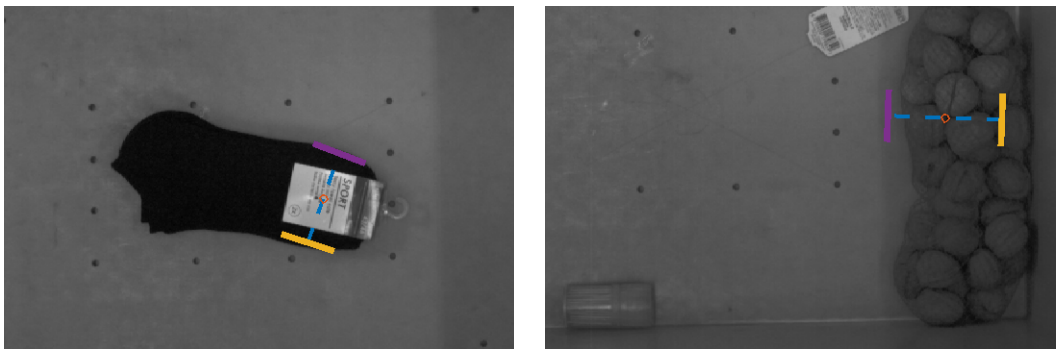


Figure 8.6: Runtimes for the 20 carriers when using the box segmentation.

8.2.3. Discussion

The results show that the approach synthesizes reasonable grasps for a variety of items, some of which are typically not handleable by a vacuum gripper, such as the belt and the net of walnuts. Additionally, the grasp score generally is a good measure of how good a grasp is. Reasonable grasps have a score of at least 0.6, whereas grasps that are unlikely to succeed have a score lower than 0.4. The range of approximately 0.4 to 0.6 implies uncertainty. Figure 8.7 shows two grasps with a score within that range. The grasp on the socks might succeed if the fingers force the fabric into a crease between the fingers. Similarly, the success of grasping the walnuts depends on whether one or multiple walnuts end(s) up between the fingers. However, it is difficult to predict whether these situations will occur. Hence it is useful that the score implies uncertainty.



(a) A pair of socks (score = 0.49).

(b) A net of walnuts (score = 0.38).

Figure 8.7: Some of the results from Appendix E.1 that have a grasp score approximately between 0.4 and 0.6.

The second important metric for the synthesis algorithm is its runtime, which is 1.67 seconds on average. An advantage is that the runtime is quite consistent when using the proper parameters. However, it is signif-

icantly longer than the 0.03 seconds reported by Baumgartl and Henrich [17]. Four possible causes could be behind this. One cause is the metric: the calculation of enclosed depth is relatively expensive and not considered by Baumgartl and Henrich. A second cause is that Baumgartl and Henrich filter lines and line pairs based on the limitations of the gripper. The evaluation in this thesis did not use such filtering, thus resulting in the algorithm considering more grasps and therefore having a longer runtime. The third cause could be the used hardware. The evaluation in this thesis used a common PC, but Baumgartl and Henrich might have used more advanced hardware. The final cause could be differences in implementation. The implementation by Baumgartl and Henrich is different from the implementation used in RSIR, and this can have a big impact on runtime [140]. One example of this is the programming language: an algorithm optimized in Matlab is generally slower than the same algorithm optimized in C++ [140].

Note also that the runtime of 1.67 seconds is relatively long with respect to the 9 seconds SIR takes to finish a pick-and-place cycle [127]. However, once again, hardware plays an important role in this. SIR is equipped with a powerful computer that is capable of quickly doing segmentations and motion planning. Hence the synthesis algorithm is expected to be faster when implemented on that hardware.

Finally, there is significant correlation between the item's curves and level of clutter, and the peak factor parameter. The results show that items with more curves require a lower peak factor. In practice, this means that the filtering on line detections is less strict. It makes sense that this is required for detecting items with curves, because the lines on such items are less strong than on items with many straight edges. Furthermore, items in more cluttered carriers require a higher peak factor. This is likely because the algorithm tends to detect lines between items, as shown in Figure 8.8. A high peak factor filters out most of those unwanted lines, which otherwise cause a long runtime. These results give reason to split the algorithm into three: one set of parameters for clutter, one set of parameters for items with many curves, and one set of parameters for general cases. Further research is required to see if using three such variants results in a better performance than when using only one variant.

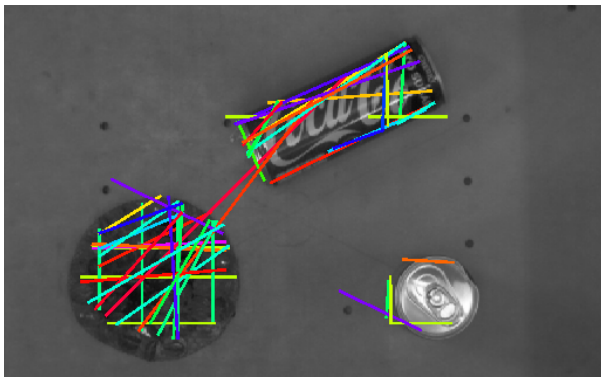
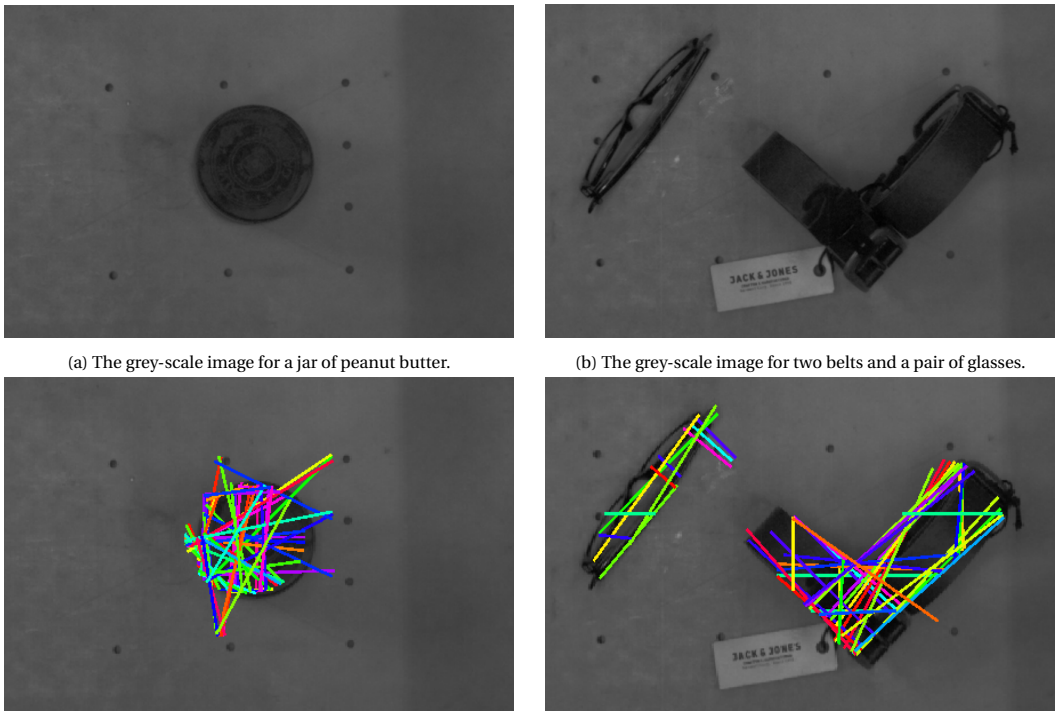


Figure 8.8: One of the results from Appendix E.1 that shows line detections between items in clutter.

Limitations of the Synthesis Approach

There are three main limitations regarding the competences of the synthesis algorithm. The first one is regarding the item. As could be expected, the algorithm performs worse for items that have few or no straight edges. Figure 8.9 shows some examples of this. Some of the detected lines are actual edges, but some lines seem unrelated to the edges. Reasonable grasps are synthesized for some of the items in Figure 8.9, but the question rises how much of this success can be attributed to the design of the algorithm, and how much of it must be attributed to random line detections.



(a) The grey-scale image for a jar of peanut butter.

(b) The grey-scale image for two belts and a pair of glasses.

(c) The detected lines for the jar of peanut butter.

(d) The detected lines for two belts and a pair of glasses.

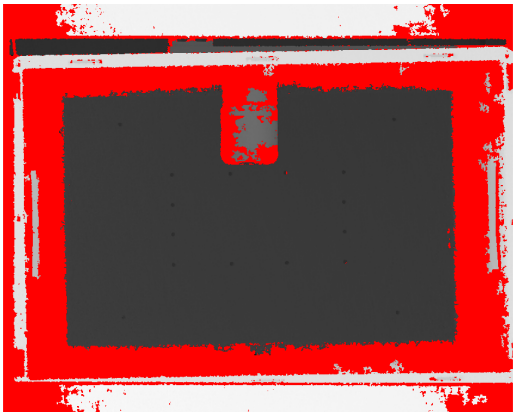
Figure 8.9: Some of the results from Appendix E.1 that show poor line detections due to the items having few straight edges.

The second limitation is regarding the reliability of the depth image. Figure 8.10 illustrates this. The depth camera cannot detect the liquid in the bottles or the reflective areas of the peanut butter jar, leaving the synthesis algorithm to fill in this missing data. In the case of the washing liquid, too much data is missing and the filtered image estimates the liquid lower in the carrier than it is in reality. Hence the algorithm synthesizes a grasp that would cause a collision. A similar situation occurs for the jar of peanut butter. In this case the resulting grasp is reasonable, but has a low score because the height of the jar is not estimated properly.

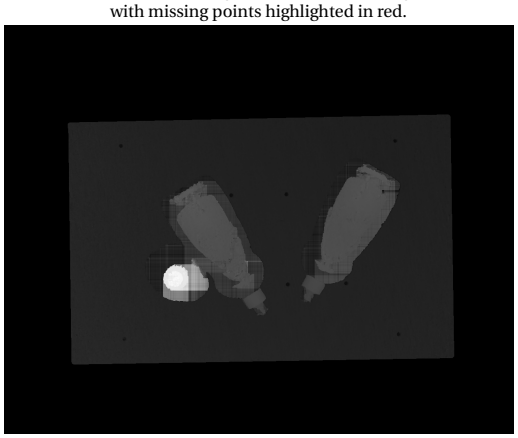
The final limitation is the similarity between grasps. From the results, it becomes clear that many of the high-scoring grasps are similar to each other. These grasps could be grouped together into one grasp. This would decrease the runtime, since the time-consuming calculation of enclosed depth would be executed fewer times. Additionally, in practice it might occur that the highest-scoring grasp proves unsuccessful. In that case, the robot might try the second-best grasp. However, if all the top-scoring grasps are similar, this is not useful.



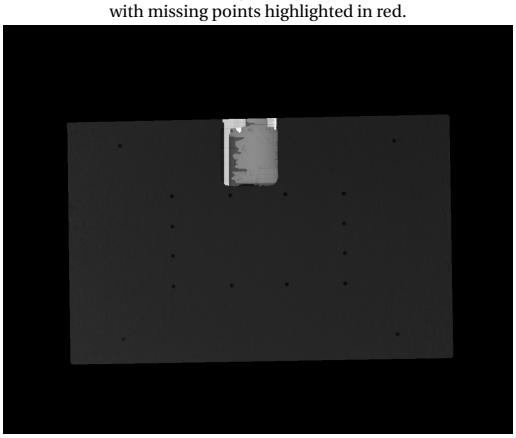
(a) The raw depth image for bottles of washing liquid, with missing points highlighted in red.



(b) The raw depth image for a jar of peanut butter, with missing points highlighted in red.



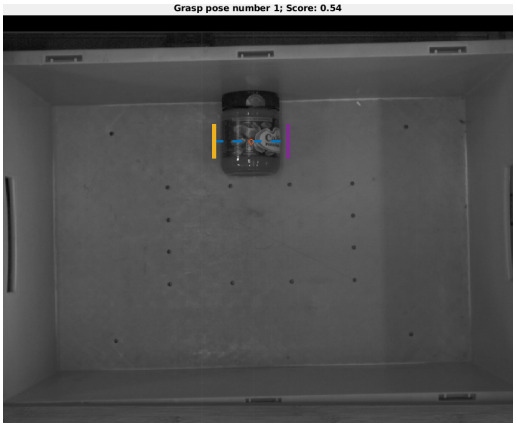
(c) The filtered depth image for the bottles of washing liquid.



(d) The filtered depth image for the jar of peanut butter.



(e) A synthesis result that unlikely leads to a successful grasp.



(f) A reasonable grasp, but with a lower score than it should have.

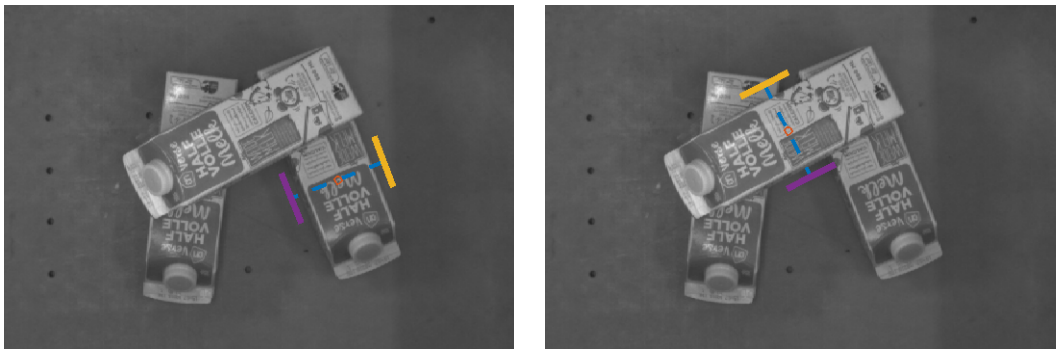
Figure 8.10: Some of the results from Appendix E.1 that show poor(ly ranked) grasps due to missing depth information.

Limitations of the Evaluation Method

The main limitation in this evaluation is that it only tests the synthesis approach and not the full grasping approach. When considering the full grasping approach, other factors may cause grasps that look good now to fail. Examples of such factors are inaccuracies in the motion control and modelling of the gripper.

A second limitation is the scope of this evaluation. Compared to the set of items RSIR can encounter in practice, the set of items considered in this evaluation is tiny. Additionally, the evaluation tuned several parameters per carrier. On the one hand, the parameters should not be tuned to obtain in-depth knowledge of the competences of one specific set of parameters. On the other hand, more parameters could be tuned to investigate their influence and derive new sets of parameters. For example, the evaluation metric could be investigated more. Figure 8.11 illustrates an example where the current metric is insufficient. The robot

would be better off grasping the top carton, which could be when the metric includes the item height. A final limitation in scope is that the algorithm is only testing using Matlab's implementations for filtering depth images, detecting edges, and detecting lines. There are more such algorithms available, and an improvement in those could result in better grasps.



(a) The highest-scoring grasp (score = 0.91).

(b) The second highest-scoring grasp (score = 0.85).

Figure 8.11: An arguably wrong ranking of potential grasps.

A third limitation is that this evaluation required a box segmentation to prevent the approach from synthesizing grasps on the carrier's edges. In practice, the carrier is not guaranteed to be at the same location every time, thus this box segmentation might not suffice. The segmentation could be made smaller to avoid the carrier's edges regardless of the exact position of the carrier. This might suffice when there are items near the centre of the carrier. However, it might be a problem if all the items are near the edge of the carrier, as these will not be (fully) visible to the synthesis algorithm.

The final limitation in this evaluation is that the RGB camera was not fully calibrated to be aligned with the depth camera. This makes no difference in the grasp synthesis, as the synthesis is based only on the depth data. However, it does create a slight difference when visualizing the grasps on the grey-scale image. This is especially apparent in some figures in Appendix E.1 that visualize the detected lines.

8.2.4. Conclusion

The goal of the grasp synthesis approach is to enable RSIR to grasp more items than if it only used the vacuum gripper. This section evaluated the grasp synthesis approach based on the variety of items it can handle (R5, R23), what it considers to be the best grasp (R24), and its runtime (R14), all when it is not provided with a segmentation of the items in the carrier (R31).

The approach synthesizes reasonable grasps for a variety of items (R5), some of which are typically not handleable by a vacuum gripper (R23). This includes items in isolation, items in sparse clutter, and items in dense clutter (R25, R28), as well as multiple types of items being in one carrier (R30). The grasp score generally is a good measure of how good a grasp is (R24). During testing, the algorithm had a reasonable runtime (R14) on an ordinary computer (R15), but it could be faster when implemented on better hardware. The set of items the approach can handle is limited because it requires the items to have several (near) straight edges, and it requires (most of) the item to be detectable by the depth camera. If the latter is not the case, the quality of grasps is estimated poorly. Furthermore, the algorithm cannot take the complete carrier as input because it then synthesizes grasps on the edge of the carrier. To counter this, a rectangular segmentation is provided. If the use of such a segmentation is feasible in practice, the segmentation mask can be hard-coded and no further segmentations are required at runtime (R31).

Initial results show that it might be beneficial to create three variants of the grasp synthesis algorithm: one for items with many curves, one for items in clutter, and one variant for general cases. Further research is required to conclude whether using these three variants indeed results in better performance than when using only one (general) variant. The limitations in this evaluation are the lack of testing on a physical robot, the relatively small item set used, the limited variation of tuning parameters, and the need for the rectangular segmentation. The final limitation is a misalignment between the RGB and depth cameras, but this only influences the visualization of the results.

8.3. Performance for Segmentations

This section considers the synthesis algorithm's performance when making use of the segmentation that SIR can provide (R25, R28). Making use of such a segmentation could improve the synthesis performance compared to when it uses the box segmentation. The outcome of SIR's segmentation algorithm consists of n masks, where n is the number of detected items. There are two ways this set of masks can be passed to the synthesis algorithm. The first way is to provide the mask of only one item. In this case, the algorithm synthesizes a grasp for that specific item. This type of segmentation is referred to as a partial segmentation. The other way is to merge the masks into one mask and pass this mask to the algorithm. In that case, the algorithm can select the item most suited for synthesis. This type of segmentation is referred to as a complete segmentation. The downside of a complete segmentation is that synthesis might take longer (R14), because more items are taken into account. This section evaluates the synthesis approach's performance for both ways of passing the segmentation mask.

8.3.1. Method

The first challenge in this evaluation is obtaining the segmentations, as the set-up used for evaluation does not include the segmentation algorithm. Instead, the 20 carriers used in this evaluation are recreated on the main SIR set-up in Veghel, which is capable of segmenting. The resulting masks are manually mapped to the images obtained on the set-up for evaluation.

The synthesis parameters used are the same as those used in the evaluation with the box segmentation in section 8.2, with the exception of the peak factor and the dilute strength. The peak factor requires tuning because the original value results in a much larger number of detected lines, most of which were irrelevant. This increased number is likely due to the algorithm considering a smaller area of the carrier when it uses a segmentation. Hence the peak factor is tuned on a case-by-case basis, but is kept constant for all the items in one carrier. The dilute strength is set at 20 for all carriers, because this value decently makes up for some parts of the items that are missing in the segmentation.

The evaluation metrics are the same as those used in the evaluation with the box segmentation in section 8.2. However, for the partial segmentations, the synthesis is run and the highest-scoring grasp is saved per segment. For the complete segmentations, the three highest-scoring, qualitatively different grasps are saved for only 5 out of the 20 carriers to limit the scope. The grasps are then compared to those synthesized using the box segmentation. For all cases, the runtime is recorded using Matlab's `tic toc` function and averaged over 5 runs of the algorithm.

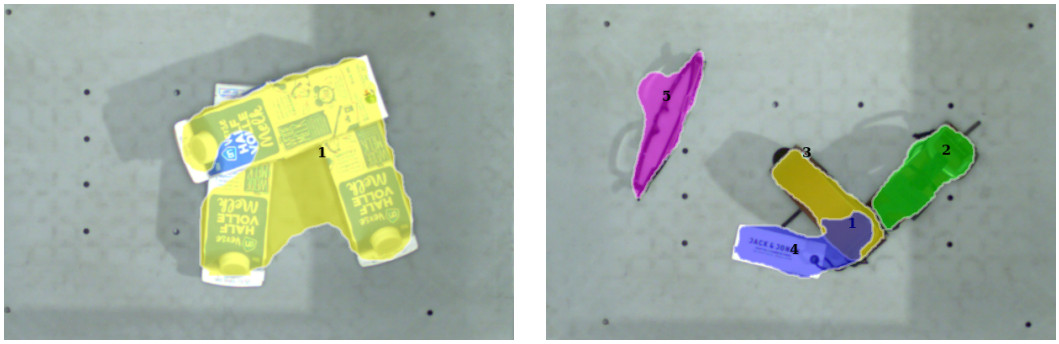
8.3.2. Results

Some examples of segmentations as output by SIR's algorithm are presented in Figure 8.12. The segmentations for all carriers, including the manually mapped ones, are included in Appendix section E.2.

Examples of grasps synthesized using the partial segmentations are shown in Figure 8.17, with Figure 8.16 summarizing the runtimes. The average runtime over all carriers is 1.01 seconds with a standard deviation of 0.56. The average grasp score of the highest-scoring grasps is 0.58 with a standard deviation of 0.31.

Examples of grasps synthesized using the complete segmentations are shown in Figure 8.15, with Figure 8.14 summarizing the runtimes. The average runtime over the 5 carriers is 1.35 seconds with a standard deviation of 0.90 seconds. For the same 5 carriers and the box segmentation, this is 1.29 and 0.80 seconds, respectively. The average grasp score of the highest-scoring grasps is 0.83 with a standard deviation of 0.11. For the same 5 carriers and the box segmentation, this is 0.84 and 0.072, respectively.

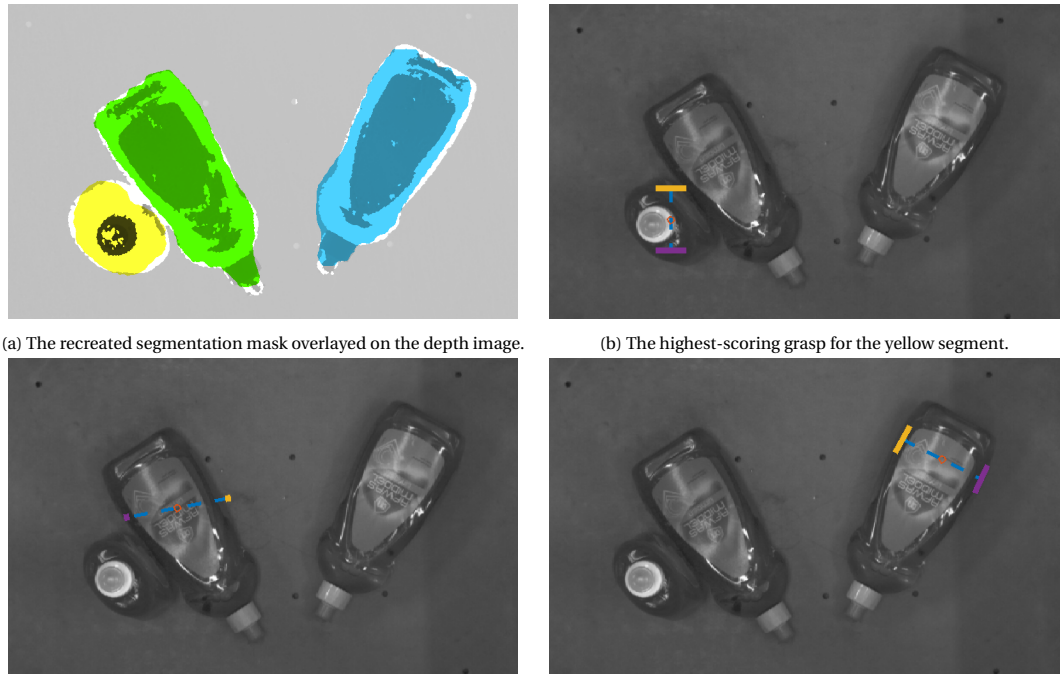
Again, all results are included in appendix E.2, including the depth data, detected edges and lines, resulting grasps, and runtimes.



(a) Three cartons of milk.

(b) Two belts and a pair of glasses.

Figure 8.12: Examples of masks computed by SIR's segmentation algorithm.



(a) The recreated segmentation mask overlayed on the depth image.

(b) The highest-scoring grasp for the yellow segment.

(c) The highest-scoring grasp for the green segment.

(d) The highest-scoring grasp for the blue segment.

Figure 8.13: An example of grasps synthesized using partial segmentations.

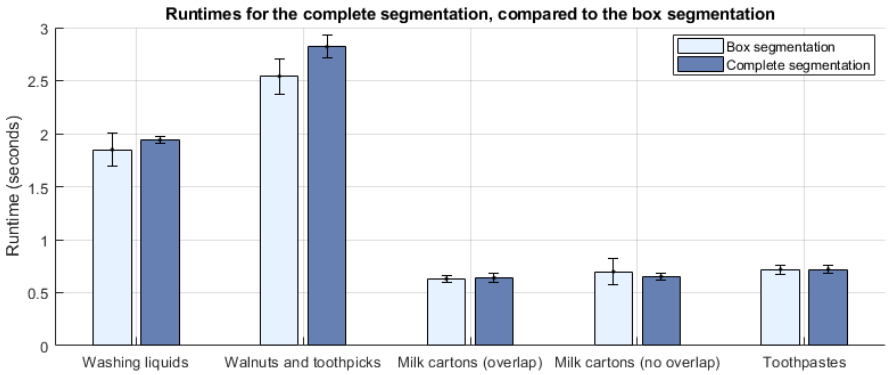


Figure 8.14: Runtimes for 5 carriers when using the complete segmentations.

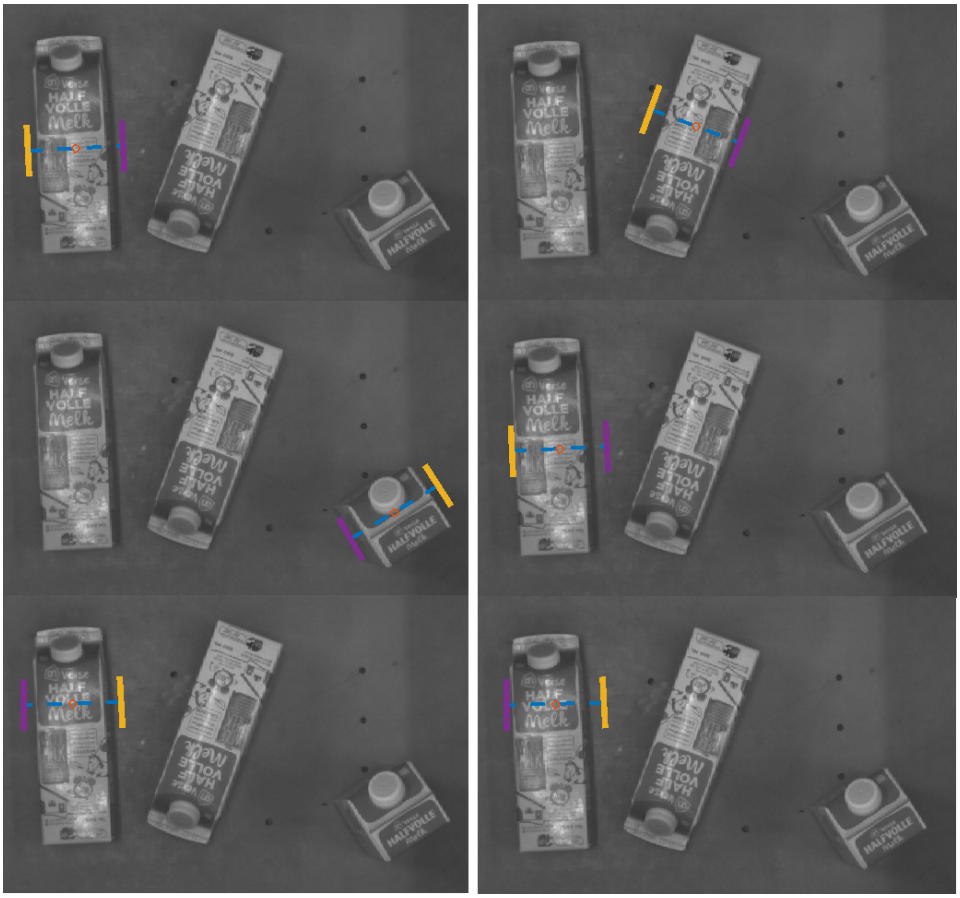


Figure 8.15: An example for the top three grasps using the complete segmentation (left) compared to the top three grasps using the box segmentation (right).

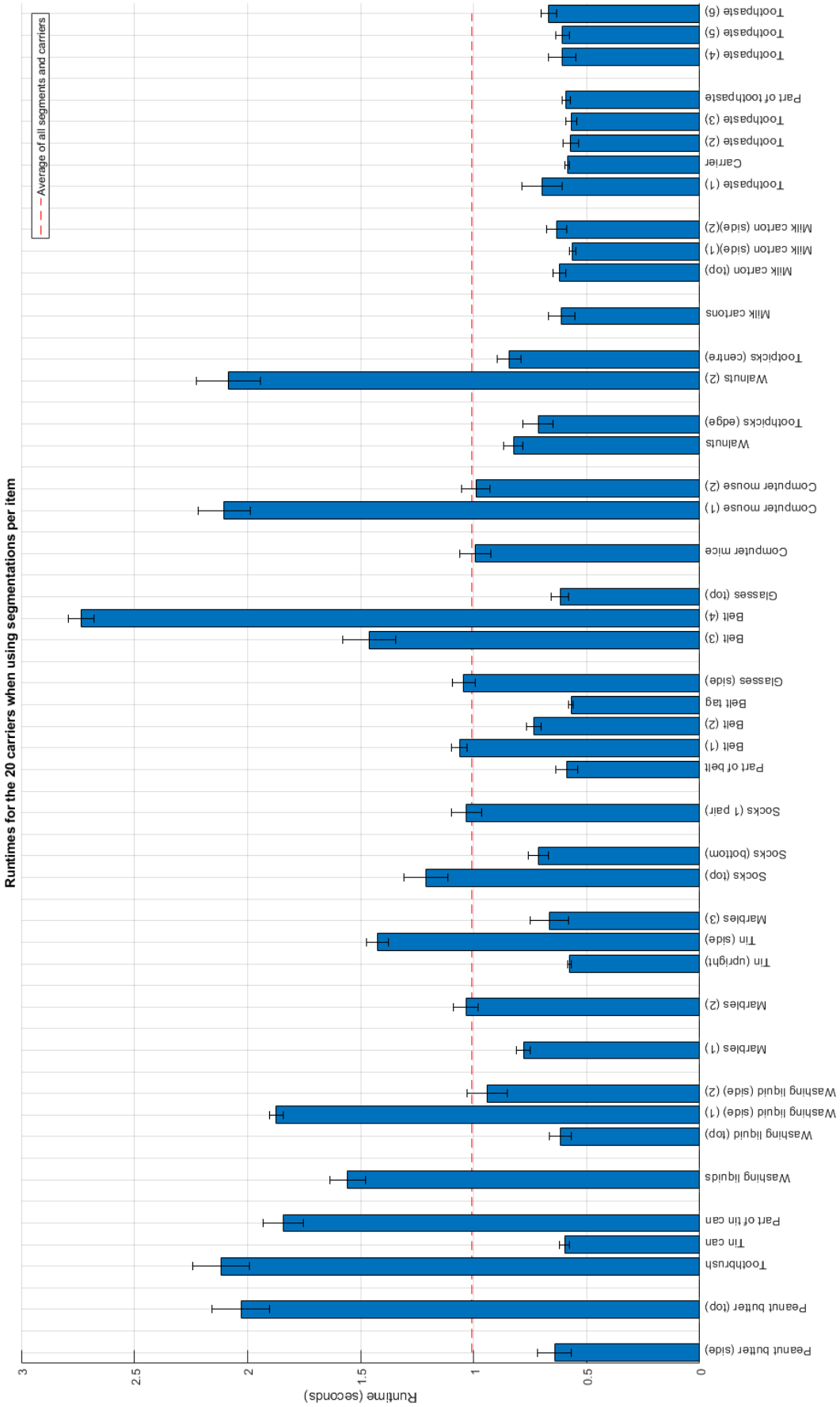


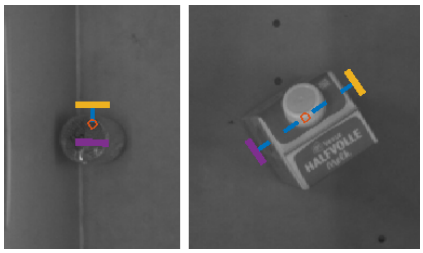
Figure 8.16: Runtimes for the 20 carriers when using the partial segmentations.

8.3.3. Discussion

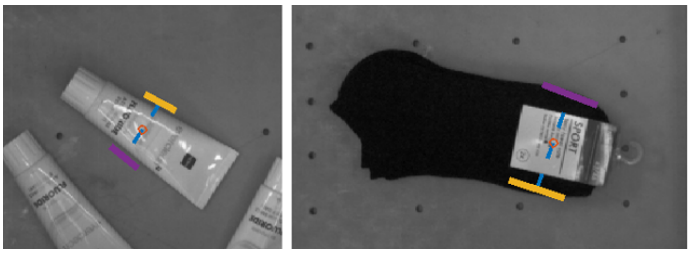
The results are now discussed for the partial segmentations and the complete segmentations.

Partial Segmentations
For the partial segmentations, the resulting grasps are very similar to those found with the box segmentation (R5, R23), and they have similar grasp scores (R24). However, in two cases, the partial segmentation allowed to find grasps not found with the box segmentation. The cases are shown in Figure 8.17a. The grasp on the toothpicks was not found previously because they are located so near the edge of the carrier that they are outside the box segmentation. This is not the case for the milk. This new grasp was likely not found using the box segmentation because there were two more milk cartons in the segmentation. The other milk cartons are on their side and thus have stronger edges, likely taking the attention away from the milk carton in Figure 8.17a.

In two other cases, good grasps found with the box segmentation were not found with the partial segmentation. These cases are shown in Figure 8.17b. The grasp on the socks is likely not found because the segmentation is imperfect and cuts off the depth information on the top right of the sock. It is unclear why the grasp on the toothpaste is not found. This might be due to parameter tuning.



(a) Two grasps found when using the partial segmentation that were not found when using the box segmentation: toothpicks(left) and milk carton (right).

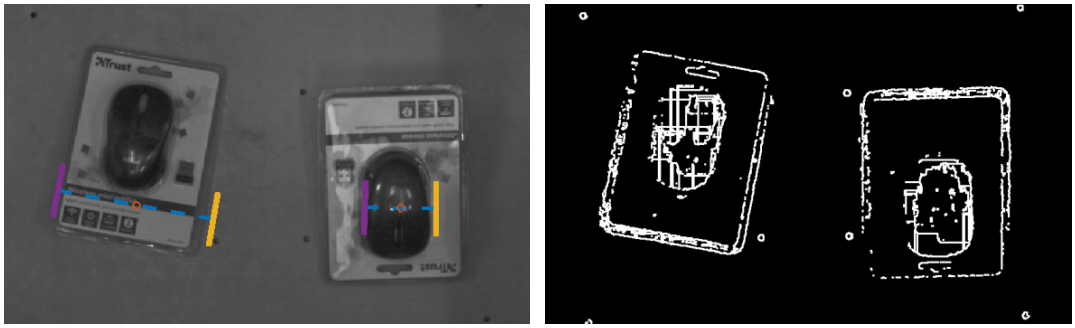


(b) Two grasps found when using the box segmentation that were not found when using the partial segmentation: toothpaste (left) and socks (right).

Figure 8.17: Some of the resulting grasps from appendix E.2 that differ when the partial segmentation is used compared to when the box segmentation is used.

The use of segmentations has several more consequences for the grasp synthesis algorithm. First, sometimes the algorithm fails because the segment is a false detection. Second, some items are not grasped because they are not detected by the segmentation network, even if they are more graspable than the other items in the carrier. Finally, the synthesis algorithm cannot find (equally good) grasps for all segments. This is reflected in the grasp score's lower average and larger standard deviation compared to that for the box segmentation. Hence randomly selecting one segment to synthesize a grasp on is not advisable.

The runtime for partial segmentations is similar to the runtime with the box segmentation (R14), and shows the same trend per carrier (for example, that the carrier with the walnuts and the toothpicks has a long runtime). However, there is a large variation in runtime between the items in a carrier. The variation appears between different items, between the same item in different poses, and between the same item in similar poses. For example, for the carrier in Figure 8.18, the algorithm takes 2.1 seconds for the grasp on the left mouse, but only 1.0 seconds for the grasp on the right mouse. This is because more lines are detected for the mouse on the left. Looking at Figure 8.18, the left mouse has stronger straight lines (where depth information is missing), and this is likely why more lines are detected on that mouse.



(a) The two grasps on the computer mice, visualized in one image.

(b) The corresponding edge detections.

Figure 8.18: An example of a carrier from appendix E.2 that results in a different runtime depending on which item mask is passed to the grasp synthesis algorithm.

Because there is no significant improvement in grasp quality or runtime, and segmenting the carrier is a time-consuming process, it is generally not advisable to run the synthesis algorithm with the partial segmentations. The exception to this is when an item is very near the edge of the carrier or when no box segmentation is available. Furthermore, if multiple item types are in one carrier and only of those types is of interest, it might be useful to use a partial segmentation to force the synthesis algorithm to grasp that one type of item.

Limitations in this evaluation are the same as those for the evaluation using the box segmentation. An additional limitations is that the segments were manually mapped between the images from Veghel and those from Eindhoven. Hence the segmentations used in this evaluation are not a perfect representation of reality. However, the algorithm mostly makes up for this by diluting the segmentation masks.

Complete Segmentations

The consequences for using a complete segmentation are similar to the consequences for using a partial segmentation. There is no significant difference in resulting grasps (R5, R23), grasp score (R24), or runtime (R14) between the complete segmentation and the box segmentation. Even so, the top-ranked grasps are different for some carriers, as visible inFigure 8.15. The cause behind this is likely the same cause that changes the grasps for the partial segmentation compared to the box segmentation. However, it should be noted that the grasp scores are similar for all grasps in Figure 8.15. Therefore it might make only a slight difference in practice that the partial segmentation ranks the grasps differently compared to the box segmentation.

It is not advisable to use the complete segmentation in practice, because there is no significant improvement in grasp quality or runtime, and segmenting the carrier is a time-consuming process. It might be the case that the complete segmentation can allow the algorithm to synthesize grasps for items very near the edge of the carrier, similar to the partial segmentation. However, such items were not present in this evaluation, so no conclusions can be made regarding that capability.

8.3.4. Conclusion

This section evaluated the grasp synthesis approach based on the variety of items it can handle (R5, R23), what it considers to be the best grasp (R24), and its runtime (R14), all when it is provided with a segmentation of the items in the carrier (R31). Note that this evaluation is the same as in section 8.2 except for the provided segmentation. Two types of segmentations were considered: partial segmentations (which contains one item) and complete segmentations (which contains all the items in the carrier).

Partial segmentations mostly result in similar grasps and runtimes compared to the box segmentation in section 8.2 (R5, R14, R23, R24). However, the partial segmentations can find grasps that the box segmentation cannot find for items that are very near the edge of the carrier, or items that are graspable but have less detectable edges than other items in the carrier. The synthesis approach cannot find equally good grasps for all segments, and fails if the segment is a false detection. Given this fact and the fact that segmenting is a time-consuming process (R14), it is not advisable to use partial segmentations. The exception to this is when an item is very near the edge of the carrier, or when no box segmentation is available.

Similarly, there is no difference in grasp scores or runtimes when using the complete segmentation compared to the box segmentation. The one difference is that the grasps are ranked differently (R24), but this likely makes little difference in practice because all those grasps have similar grasp scores. Again, it is not advisable to use complete segmentations in practice, except when no box segmentation is available.

In summary, this chapter evaluated the grasp synthesis approach based on the grasps it synthesizes (R24) and the time it takes to do so (R14). The tuning of the approach's parameters is a trade-off between runtime and the number of grasps found. When properly tuned, the approach synthesized reasonable grasps for a variety of items (R5), including items in clutter (R25, R28, R30), some of which are typically not handleable by a vacuum gripper (R23). It had a reasonable runtime on an ordinary computer (R14, R15). The approach was not suited for grasping items without straight edges or items that were not detectable by a depth camera. Furthermore, it required a segmentation to prevent it from grasping the carrier. For this, a simple rectangular segmentation (R31) was preferred over a segmentation of the items in the carrier (R25), because the latter is more computationally expensive and provided no improvement in performance for most cases. This evaluation was mostly limited by the lack of testing on a physical robot and the relatively small item set used (R32, R34). The next chapter uses this grasp synthesis approach to demonstrate RSIR on a physical robot.

9

Ontology and Reasoner Evaluation

This chapter addresses the problem of evaluating RSIR's performance, thus answering the research question: *What is RSIR's performance on selecting the correct actions to be executed after the model and software architecture have been implemented?* First, section 9.1 addresses Methontology's evaluation phase: it confirms that the ontology is correct by answering the competency questions. Section 9.2 then evaluates RSIR's performance for a virtual robot tasked with grasping an item with unknown properties. Finally, section 9.3 demonstrates RSIR being used for a grasping task on a robotic set-up. For this final section, the grasp synthesis approach evaluated in chapter 8 is used.

9.1. Competency Questions

This section goes through the competency questions defined in subsection 4.1.3 to evaluate the design of RSIR. Section 9.1.1 evaluates whether RSIR can answer all competency questions, and 9.1.2 discusses in more detail the limitations in the answers RSIR can give.

9.1.1. Answers to Competency Questions

This subsection evaluates whether RSIR can answer the competency questions. If RSIR is designed according to its requirements, it can answer all the questions. This subsection provides the logic behind the answers. Concrete Prolog queries that RSIR uses are included in appendix F.1.

Task Questions

Recall that the competency questions regarding tasks are:

Q1. What capabilities does the system have access to?

Q2. How are the capabilities constrained?

Q3. What is the predicted performance of each capability?

Q4. What actions must the system do to complete a task?

Q5. How confident is the system that it can complete a task?

Q6. What is the predicted performance for completing a task?

Q7. If there are multiple approaches to completing a task, which approach is optimal?

RSIR answers Q1 by looking at which Devices a Robot is connected to and what Capabilities these Devices provide. A Capability is constrained by the Inputs it requires, any Item Property Constraints it might have, and any Cardinality Constraints it might have. Item Property Constraints are defined as a confidence of the Capability succeeding if the item has that property. An example is illustrated in Figure 9.1. Cardinality Constraints limit how many Capabilities within one group a robot can use during one action plan. For example, a constraint is that a robot with one arm can only use one gripper during a grasping task. This answers Q2. The answer to Q3 can be found through the cost function, which can currently be a function of the capability's

execution time (t_i) and confidence value (c_i). For example, the cost function $C_i = \sqrt{t_i}/c_i$, where C_i is the cost of capability i. To answer Q4, RSIR looks at the Capabilities a Robot has access to and chains these such that each Capability's Input is provided by another Capability's output. In doing this, RSIR starts at the definition of the Task and continues until all input constraints are satisfied. The resulting Action Plan is an ordered list of actions that the Robot must do to complete the task. Q5 and Q6 can be answered by the same cost function that answered Q3. Following the same example, the cost of the action plan would be $C = \sqrt{\sum t_i}/\prod c_i$ with i iterating over all the actions in the action plan. Finally, RSIR finds the optimal approach by selecting the action plan with the lowest cost, thus answering Q7.

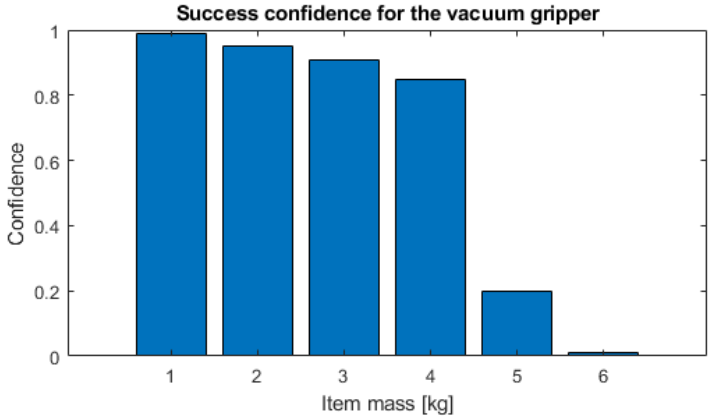


Figure 9.1: The limitations of a vacuum gripper on the item's mass, modelled as confidence probabilities. For example, the vacuum gripper has a 20% chance of succeeding if the item's mass is 5kg.

Failure Questions

Recall that the competency questions regarding failures are:

Q8. What kinds of failures can occur?

Q9. Which of these failures are likely to occur?

Q10. When a failure has occurred...

(a). ...what type of failure is it?

(b). ...what is the cause behind the failure?

(c). ...what can be concluded about...

(i). the properties of the item?

(ii). the capabilities of the robot?

(iii). the actions the system should take to prevent this failure from re-occurring?

RSIR contains the answer to Q8 through the constraints defined for capabilities. An Item Property Constraint defines for which values of a property a capability can fail, thus any failure with one or more Item Property Constraints can cause a failure. When provided with this constraint and the probability density function of the item, RSIR can compute the confidence value per capability. Capabilities with lower confidence values are more likely to cause a failure, thus answering Q9. The state machine detects a failure when executing a specific action, answering Q10(a). Based on this detection, the Bayesian network infers which capability and item property most likely caused the failure, as well as the most likely probability distributions of the item's properties (Q10(b,c)). No conclusions are drawn for the capabilities because they are assumed fixed (Q10(a)(i)). However, the costs per action plan are re-computed based on the inferred item properties: plans that are unlikely to succeed given the new knowledge get a higher cost. This way, RSIR answers Q10(a)(iii).

Item Questions

Finally, the competency questions regarding items are:

Q11. Which item properties are relevant for the task at hand?

Q12. Which item properties are independent of the scenario?

Q13. Which item properties are dependent on the scenario?

Q14. Which item properties are known, and with what accuracy?

Q15. Which item properties can be estimated?

The answer to Q11 depends on the selected Action Plan. For example, the relevant item properties are different when a vacuum gripper is used to complete a task compared to when a fingered gripper is used. When an action plan is selected, RSIR can find the relevant item properties by finding the Item Property Constraints on the Capabilities in the Action Plan. Item properties that are independent of the scenario are a subclass of Persistent Property (Q12). On the other hand, properties that are dependent on the scenario are a subclass of Item Property but not of Persistent Property (Q13). Item properties that are known are saved as discrete probability distributions for the Target Item, thus answering Q14. Finally, RSIR can answer Q15 by defining a Task that has as output the desired Item Property. For example, the triple `vi:task1 vi:taskOutput vi:'Mass'` defines a task for which the Robot estimates the target item's mass.

RSIR has no implementation to answer the competency question: *What tasks should the system do to complete a goal?* However, this is acceptable because the design of such an algorithm was considered to be outside the scope of this thesis.

9.1.2. Limitations

Although RSIR can answer all the required competency questions, there are three limitations in its design. The first is clear: the lack of an algorithm for answering the question *What tasks should the system do to complete a goal?* In its current state, RSIR can reason how to complete a task, but it cannot reasoner whether, for example, it is better to first estimate an item property and then attempt a grasp, or to immediately attempt a grasp. This limitation means that the algorithm RSIR uses for exploration is based on randomly selecting one action plan, which is not optimal.

The second limitation is that RSIR was developed using the sense-plan-act paradigm. This allows RSIR to create action plans and use each action's output as the next action's input, and it works well for most of the implementations of related tasks. However, it makes it so that RSIR is unsuited for feedback-heavy capabilities such as visual servoing. There is a workaround to implementing feedback-heavy capabilities, but this is not ideal. Additionally, although many related tasks were considered, it is infeasible to investigate every possible related task for RSIR and every implementation of said task. Instead, RSIR was designed to handle most of the related tasks considered.

A final limitation in RSIR's ontology is that it does not contain knowledge of dependencies between item properties. In fact, the Bayesian network is built on the assumption that properties are independent. This assumption is violated in some cases. Consider the example of a large, heavy item with an eccentric centre of mass. The vacuum gripper can handle this item if it is heavy but small, or if it is large but light. However, it cannot handle the item if it is both heavy and large because of the mass eccentricity. This knowledge cannot be contained in RSIR using only Mass and Size, because the properties are assumed to be independent. A workaround is to include a Mass Eccentricity property. However, it is unclear if this workaround is applicable to all cases were properties are dependent.

In summary, RSIR can answer the required competency questions, but it only uses a simple algorithm for exploration, is unsuited for feedback-heavy capabilities, and cannot capture knowledge of dependencies between item properties.

9.2. Ontology and Reasoning

This section addresses the problem of evaluating RSIR's performance regarding whether it selects the optimal action plan (R6), the item properties it estimates, and the time it takes for its reasoning (R14). The first results are presented in subsection 9.2.1. Afterwards, subsections 9.2.2, 9.2.3, and 9.2.4 investigate the influence of prior knowledge, the influence of exploration, and the system's robustness, respectively.

All evaluations use the following set-up. A robot is tasked to grasp an item and verify that the grasp was successful. To do so, the robot must select one gripper, one synthesis approach, and one method to verify the grasp. There are four verification methods: move the item to the verification camera and detect the item's footprint; measure the opening of the gripper; measure the payload on the robot arm; and measure the pressure in the vacuum gripper. Table 9.1 summarizes the limitations of the capabilities the robot has.

The capabilities are presented in more detail in appendix F.2. Additionally, the robot has knowledge of five item properties: mass, porosity, shape, size, and transparency. At the start of a test, the robot is presented with one of the eight items presented in Figure 9.2. These items are selected because together they have a large variety in item properties. The robot has no knowledge about the specific item it encounters. RSIR then computes the action plans and selects one of them to be executed.

Table 9.1: The general limitations of the capabilities available to RSIR during evaluation.

	Capability	Best at handling	Worst at handling	Requires the use of
Gripper	Three-finger gripper	Spherical shapes	Non-spherical shapes, heavy masses	-
	Two-finger gripper (large)	Shapes with straight edges, large masses	Spherical shapes	-
	Two-finger gripper (small)	Shapes with straight edges	Spherical shapes	-
	Vacuum gripper (large)	Flat surfaces, large masses	Porous items, sizes smaller than the suction cup	-
	Vacuum gripper (small)	Flat surfaces, small sizes	Porous items, large masses	-
Synthesis approach	Circular	Spherical shapes	Non-spherical shapes	Three-finger gripper
	Flat surface	Flat surfaces	Non-flat surfaces	Vacuum gripper
	Parallel line	Shapes with straight edges	Shapes without straight edges	Two-finger gripper
	Principal axis	Shapes with strong principal axis	Shapes without strong principal axis	Two-finger gripper
Verification	Use item footprint	-	Transparent items	-
	Use gripper opening	-	Small sizes	Two- or three-finger gripper
	Use payload	Large masses	Small masses	-
	Use pressure	-	-	Vacuum gripper
Other	Move to verification pose quickly	Small masses	Large masses	-
	Move to verification pose slowly	Small masses, Large masses	-	-
	Sense input depth	-	Transparent items, complex shapes	-



	Dog food	Glasses	Milk	Petri Dish	Sponge	Stressball	Teddy Bear	Toothpicks
Mass	2.5kg	0.5kg	1.5kg	0.5kg	0.5kg	0.5kg	1.0kg	0.5kg
Porosity	Solid	Solid	Solid	Solid	Porous	Solid	Semi-porous	Solid
Shape	Complex	Complex	Box	Disk	Box	Sphere	Complex	Cylinder
Size	25cm	10cm	15cm	10cm	25cm	5cm	20cm	5cm
Transparency	Opaque	Semi-transparent	Opaque	Transparent	Opaque	Opaque	Opaque	Semi-transparent

Figure 9.2: The items considered during the evaluation of RSIR.

Action plans are executed virtually. This is done by first computing the likelihood of a capability succeeding based on its constraints and the item's properties (which are unknown to the robot). If an action is executed, a random number is generated and compared against the likelihood of success. This determines whether an action succeeds or fails. Any action with one or more constraints can fail, but failures are only detected during the action that verifies that there is an item in the gripper. RSIR goes through 100 grasping cycles, continually updating the beliefs it has about the item and recomputing action plan confidences. This test is done 5 times for each item to lessen the influence of randomness. Unless specified otherwise, RSIR only creates action plans during the first grasping cycle and updates said plans (instead of recreating them) for every cycle afterwards.

RSIR's performance regarding is determined as follows. Once RSIR has completed its 100 cycles, the stabilization cycle is determined from the resulting data. The stabilization cycle is defined as the cycle that is start of 10 consecutive cycles with the same action plan. The number of cycles executed before the stabilization cycle is reached is counted, which is a measure for how quickly the system stabilizes. Additionally, it is inspected whether the action plan selected after stabilization is the action plan that would be optimal if the item properties were known (R6). Additionally, the confidence of said optimal action plan is included, as well as the runtime for the exploitation evaluation (R14). RSIR's performance regarding its estimation of item properties is done by plotting the item properties for two points in time: directly after the stabilization cycle and 20 cycles after the stabilization cycle. The results are compared with the item's actual properties.

The evaluations differ in the following aspects. First is the prior knowledge RSIR has: in some cases it knows nothing, whereas in other cases it has knowledge about the distribution of item properties that it can encounter. Secondly, the exploitation to exploration ratio differs between evaluations.

9.2.1. Exploitation

During the exploitation evaluation, RSIR has no prior knowledge of the item, and thus assumes the properties to be evenly distributed. RSIR does no explorations, thus during every cycle the action plan with the lowest cost is selected and executed. The runtime is recorded as the time required for a full cycle: creating/updating the action plans, selecting and executing the best plan, and updating the item properties based on the outcome of the execution. This is measured for 100 grasping cycles that have a 50% chance of succeeding, for the case where RSIR reuses the action plans and for the case where it does not. Since initial tests suggested that reasoning is quicker for successes than for failures, the runtime was also measured when the chance of success was 100% and 0%.

Results
The results for the action plans are presented in Table 9.2. RSIR takes 16 grasping cycles on average to stabilize, with a standard deviation of 20 cycles. Additionally, Table 9.3 presents the runtimes, with Figure 9.3 showing the runtime per grasping cycle for one test of RSIR when it does not reuse action plans. Figures of the estimated item properties are included in appendix F.3.

Table 9.2: Results for the action plan when RSIR does not explore, averaged over 100 grasping cycles.

Item	Number of cycles before stability is reached	Fraction of tests that result in the optimal action plan	Confidence of the optimal action plan
Dog food	24 ± 18	0.0	0.77
Glasses	39 ± 36	0.40	0.12
Milk	2.0 ± 0.0	1.0	0.87
Petri dish	30 ± 31	0.40	0.028
Sponge	7.0 ± 2.0	1.0	0.26
Stressball	1.0 ± 0.0	1.0	0.70
Teddy bear	27 ± 7.4	0.40	0.047
Toothpicks	4.4 ± 2.8	0.60	0.30

Table 9.3: Runtimes for RSIR per full grasping cycle when not doing explorations, average over 100 cycles.

Reuse of action plans	Probability of successful execution	Runtime (seconds)
Yes	1	0.44 ± 0.05
Yes	0.5	0.78 ± 0.39
Yes	0	0.90 ± 0.13
No	0.5	8.1 ± 7.6

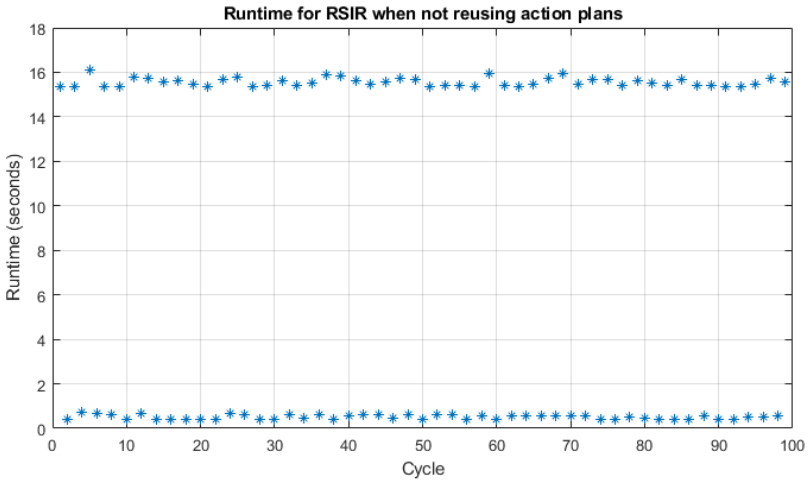


Figure 9.3: The runtime of RSIR when action plans are recreated for every grasping cycle.

Discussion

During most tests, RSIR stabilizes at the optimal action plan (R5, R6) because of the information it derives from failures (R4). For some items RSIR does not find the optimal action plan. For example, the vacuum grippers are never used in any of the cycles because the item is assumed to have a probability of only 0.33 to be solid. Hence the vacuum grippers get a high cost and action plans with vacuum grippers are never selected. For example, the optimal action plan for the dog food makes use of a vacuum gripper, and is thus never selected. The solution to this lies in gathering more knowledge about the item's properties. In the case of the vacuum gripper, this property is porosity. The figures in Appendix F.3 shows that the distribution for the item's porosity is never updated. This is because the only capabilities with knowledge about porosity - the vacuum grippers - are never selected. These capabilities must be selected to gain more knowledge, despite the fact that they are not the capabilities that lead to the highest chance of success according to RSIR's beliefs. This calls for exploration.

A second result regarding stabilization is that the number of cycles required to reach stability differs greatly. Items that stabilize quicker seem to be those for which the optimal action plan has a high confidence and for which the optimal plan is found.

The time it takes for RSIR to reason is relatively short (R14) on an ordinary computer (R15) when reusing action plans. The results show that the reasoning is significantly faster for a successful execution than for a failed execution. This is likely because a failed execution contains more uncertainty (which capabilities failed, which capabilities succeeded, which item properties caused the failure?) than a successful execution, when it is known that all capabilities succeeded. RSIR seems to be faster when reusing action plans than when not reusing them, although this cannot be confirmed due to the large variation of the data for not reusing plans. This large variation is due to the odd pattern shown in Figure 9.3. The raw data show that the cycles with short runtime always select the large two-finger gripper, parallel line synthesis, and verification through the gripper opening. This consistency suggests that there is a bug in the architecture, which is likely a timeout that occurs because the reasoner is taking too long, and the reasoner ends up selecting the first action plan in memory instead. If this is the case, it is clear that the reuse of action plans greatly improves RSIR's runtime.

The correctness of the estimated item properties varies. For some items, such as the milk, the properties are estimated correctly. For other items, such as the dog food, the properties are estimated incorrectly. This incorrect estimation is partially due to a lack of knowledge in the capabilities and partially due to blaming the cause of a failure on the wrong property. An example of the former can be seen in the item's mass. Masses between 0.5 and 2.0 kg have identical probabilities for all items because the only capabilities that contain knowledge about mass are the 3-fingered gripper, the small 2-fingered gripper, and the small vacuum gripper. Out of these, only the former is selected by RSIR. This gripper can handle all masses up to 2.0 kg equally well, which explains how the mass probability distribution got its shape.

The wrong selection of a cause behind a failure can be seen in the results for the petri dish. The optimal action plan has only an 2.7% chance of succeeding, so even when RSIR selects this plan, failures occur.

The cause for these failures is mainly the transparency of the item. However, as more failures occur, RSIR concludes the item is more transparent, larger in size, and different in shape. These last two conclusions are incorrect. However, RSIR does not realize this because the optimal action plan is limited by transparency, size, and shape. Other capabilities that can differentiate between these three properties must be executed to correct these wrong conclusions. Again, this calls for exploration.

Conclusion
This section evaluated RSIR's performance for selecting the optimal grasping approach when presented with an item with unknown properties. During most tests, RSIR found the optimal action plan (R6) for a variety of items (R5). This is because RSIR reasoned about the causes of failures and in the future selected actions that were less likely to fail (R4). The reasoning was finished within a relatively short time (R14) on an ordinary computer (R15) when reusing existing action plans. However, RSIR occasionally did not find the optimal action plan and incorrectly estimated the item's properties. This is due to exploitation of incorrect knowledge about the item and wrong conclusions about failures. The most promising solution to this seemed to be the introduction of exploration.

9.2.2. Prior Knowledge

The previous evaluation concluded that RSIR did not find the optimal action plan partially because it exploited incorrect knowledge about the item's properties. Some of this incorrect knowledge came from the assumption that item properties are evenly distributed (for example, that only 33% of the items is non-porous). This evaluation instead provides RSIR with more accurate prior knowledge (shown in Figure 9.4) and investigates the effects on the selected action plan and item properties. Again, RSIR does no explorations.

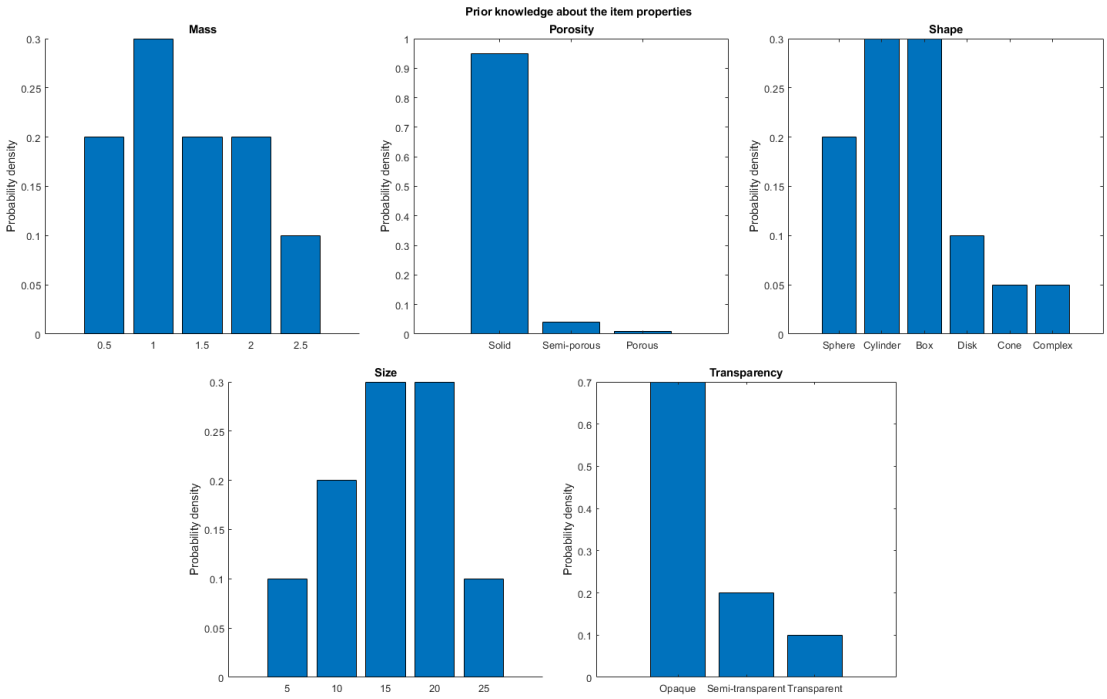


Figure 9.4: The prior knowledge about item properties that is provided to RSIR.

Results
Table 9.4 presents the resulting action plans when RSIR applies the prior knowledge in Figure 9.4 compared to when it assumes evenly distributed item properties. Additionally, appendix F.4 presents figures of the estimated item properties.

Table 9.4: Results for the action plan when RSIR uses prior knowledge, compared to when it does not.

Item	Prior knowledge	Number of cycles before stability is reached	Fraction of tests that result in the optimal action plan	Confidence of the optimal action plan
Dog food	Yes	2.2 ± 1.8	0.80	0.77
	No	24 ± 18	0.0	
Milk	Yes	1.0 ± 0.0	0.0	0.87
	No	2.0 ± 0.0	1.0	
Petri dish	Yes	29 ± 20	0.60	0.028
	No	30 ± 31	0.40	
Sponge	Yes	13.6 ± 22	1.0	0.26
	No	7.0 ± 2.0	1.0	

Discussion
The effects of the prior knowledge on RSIR finding the optimal action plan (R6) varies depending on the item (R5). Based on the prior knowledge, the action plan with the lowest cost contains the vacuum gripper. Because this is the optimal action plan for the dog food, RSIR stabilizes more quickly for the dog food. However, it also leads RSIR to believe the vacuum gripper is optimal for the milk, even though it is not. Because RSIR does not explore, it never finds the optimal action plan for the milk. The results for the petri dish and sponge do not differ greatly from those for the evaluation without prior knowledge. This is likely because the confidence of the optimal plan is relatively low, and thus there is no action plan that is clearly a lot better than the others. This makes it harder for RSIR to reason, and thus the added benefit from prior knowledge is small.

The prior knowledge both improves and worsens the accuracy of the estimated properties. Improvements can be seen in the transparency for all items except the petri dish and the porosity for the dog food and milk. This is because these items' properties are in line with what RSIR believes to be most common. Properties that are uncommon, on the other hand, are estimated with worse accuracy. This is the case for the dog food's shape and mass, the petri dish's mass, shape, size, and transparency, and the sponge's porosity and size. Note that the values of all of these properties have a low probability in Figure 9.4. Because of this low probability, RSIR attributes the cause of failures to other properties, and thus ends up with an incorrect belief of the properties.

Conclusion
This subsection evaluated RSIR's performance for selecting the optimal grasping approach when presented with an item with unknown properties and prior knowledge about the distribution of said properties. RSIR more often selected action plans that performed well based on the prior knowledge. This resulted in RSIR finding the optimal grasping approach for items for which those plans were optimal, but not for items for which those plans were not optimal (R5, R6). Similarly, RSIR estimated the item properties well for items whose properties were in line with the prior knowledge, but poorly for items whose properties were not in line with the prior knowledge. It can be concluded that providing RSIR with prior knowledge is not sufficient to make it always find the optimal action plan. Instead, the most promising solution to this seemed to be the introduction of exploration.

9.2.3. Exploration

Previous evaluations showed that RSIR might use explorations to correct wrong conclusions about failures and item properties. Hence this evaluation considers the effect of randomly selecting an action plan every n grasping cycles, for n=5 and n=2. During the remaining cycles, the action plan with the lowest cost is selected. Again, RSIR has no prior knowledge of the item, and thus assumes the properties to be evenly distributed. This evaluation is done for three items: the dog food, milk, and sponge.

Results
Table 9.5 presents the resulting action plans when RSIR explores compared to when it does not explore. On average, RSIR takes 11±14 cycles to stabilize without exploration, 9.1±7.7 cycles when exploring 1 in 5 cycles, and 8.6±7.0 cycles when exploring 1 in 2 cycles. Appendix F.5 presents figures of the estimated item properties.

Table 9.5: Results for the action plan when RSIR explores, compared to when it does not explore.

Item	Number of explorations	Number of cycles before stability is reached	Fraction of tests that result in the optimal action plan	Confidence of the optimal action plan
Dog food	None	24 ± 18	0.0	0.77
	1 in 5	10 ± 8	0.0	
	1 in 2	13 ± 6	1.0	
Milk	None	2.0 ± 0.0	1.0	0.87
	1 in 5	3.6 ± 2.2	1.0	
	1 in 2	3.4 ± 0.9	0.8	
Sponge	None	7.0 ± 2.0	1.0	0.26
	1 in 5	14 ± 8	1.0	
	1 in 2	9.0 ± 8.0	1.0	

Discussion
The results show that exploration allows RSIR to find the optimal action plan (R6) for all items (R5). Additionally, the estimated item properties are more correct than when exploration is not used for the dog food and the milk. This is however not the case for the sponge, for which the estimations are still mostly incorrect. This is likely due to the fact that the confidence of the optimal action plan is low, and thus RSIR still draws incorrect conclusions about which item properties caused a failure. There is no significant change in the number of cycles before stability is reached.

A limitation in this evaluation is that explorations are done randomly, which is not at all an optimal approach to obtaining knowledge about the item. A better approach would take into consideration which properties have the most uncertainty and which capabilities have knowledge of those properties, for example. This could lower the number of cycles required for stabilization. However, designing such an approach is considered outside the scope of this thesis because it is closely related to deciding between estimating item properties and executing the task (the competency question that is outside the scope of this thesis). Note also that the explorations in this evaluation are only useful for the dog food when it is done every 1 in 2 cycles. This is another indication that the algorithm for exploring must be designed properly to achieve the best results with RSIR.

Conclusion
This subsection evaluated RSIR's performance for selecting the optimal grasping approach when presented with an item with unknown properties. In this evaluation, RSIR did random explorations to correct wrong conclusions about failures and item properties (R4). The explorations allowed RSIR to find the optimal action plan (R6) for all items (R5). Additionally, the estimated properties were more correct for two out of the three items. The use of explorations had no effect on the number of cycles required for stabilization. The most promising approach to further improving RSIR is to design a more intelligent algorithm for exploration.

9.2.4. Robustness

The evaluation using exploration showed that RSIR finds the optimal action plan in most cases. However, this evaluation (as did all previous evaluations) assumed that the confidence values of the capabilities are known. However, in practice these values are inaccurate, for example due to limited knowledge of the capabilities or limited available data for machine learning. For example, it might be that the confidence value for a capability and a semi-porous item is defined as 0.2, but in reality this capability succeeds only 10% of the time. For RSIR to be applicable to grasping (R1) and other related tasks (R3), it must handle such imperfect knowledge of its capabilities. This subsection evaluates RSIR's robustness to such imperfect knowledge.

The imperfect knowledge is introduced as follows. For each value for a constraint on a capability, a random number (noise) is added. This random number is generated from a normal distribution with a mean of 0 and a deviation of 0.1. This deviation is large enough to introduce inaccuracies but small enough to not remove the general knowledge of RSIR. The resulting confidence values are constrained to be between 0 and 1. As an example, Figure 9.5 shows the resulting confidence for the three-fingered gripper and an item's shape. The inaccuracies are added to the confidences before the evaluations are done, and are not changed during the evaluations. The remaining parameters are the same as those for exploration in subsection 9.2.3. The results are compared against those of exploration without noise to determine how robust RSIR is.

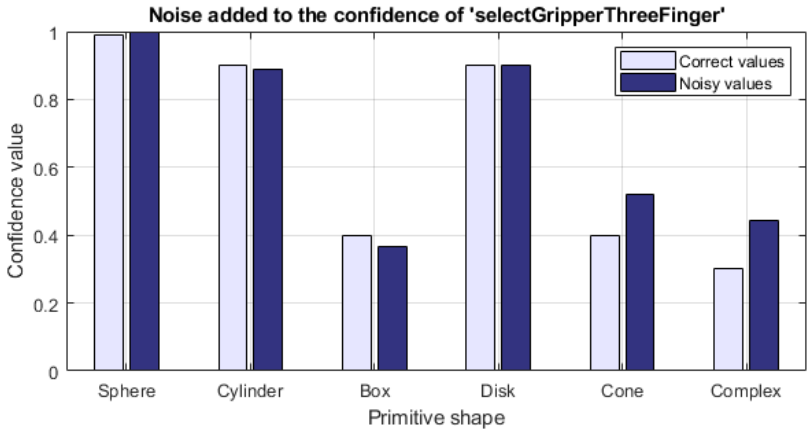


Figure 9.5: An example of the confidence values after noise is added.

Results

Table 9.6 compares the resulting action plans when RSIR has perfect knowledge about its capabilities and when the noise is added. Additionally, appendix F.6 presents figures of the estimated item properties.

Table 9.6: Results for the action plan when RSIR explores, compared to when it does not explore.

Item	Exploration	Noise	Number of cycles before stability is reached	Fraction of tests that result in the optimal action plan	Confidence of the optimal action plan
Dog food	1 in 5	Yes	4.6 ± 2.5	0.0	0.77
		No	10 ± 8	0.0	
	1 in 2	Yes	9.8 ± 6.7	0.4	
		No	13 ± 6	1.0	
Milk	1 in 5	Yes	1.0 ± 0.0	1.0	0.87
		No	3.6 ± 2.2	1.0	
	1 in 2	Yes	1.0 ± 0.0	1.0	
		No	3.4 ± 0.9	0.8	
Sponge	1 in 5	Yes	11 ± 6	0.8	0.26
		No	14 ± 8	1.0	
	1 in 2	Yes	14 ± 24	1.0	
		No	9.0 ± 8.0	1.0	

Discussion

The fraction of tests for RSIR finds the optimal action plan is slightly lower compared to when RSIR has perfect knowledge about its capabilities (R6). This makes sense because RSIR is less able to reason because of the noisy knowledge it has. Regarding stabilization, the milk is somewhat of an odd case because in all 5 tests, RSIR selects the optimal action plan during the first cycle and selects no other plans after that. Hence the stabilization for the milk has a standard deviation of zero. The results show no significant increase in number of cycles before stability is reached for the other items. This shows that RSIR has a decent level of robustness and can be applied in practical applications (R1, R3).

The figures of the item properties show that the estimation of the properties is slightly worse compared to when there is no noise on the confidence values. The biggest worsening seems to be for the item's size, where RSIR incorrectly concludes that the item is smaller than 20 cm. A closer look at the noise explains this: the large two-fingered gripper is often selected, and Figure 9.6 shows that the confidence for large sizes is greatly underestimated. Hence RSIR quickly discards large item sizes if an action plan with the large two-fingered gripper succeeds. Finally, there seem to be no other general trends in the results for the item properties.

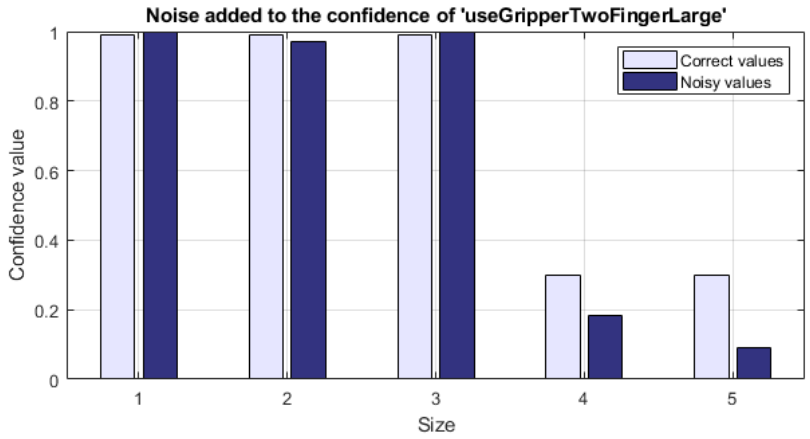


Figure 9.6: The confidence for the large two-fingered gripper for an item's size.

The main limitation in this evaluation is the limited number of tests done. These results are no evidence that RSIR is robust for all possible item properties and tasks. However, this evaluation does provide some first positive results that point towards robustness.

Conclusion
This subsection evaluated RSIR's performance for selecting the optimal grasping approach when provided with imperfect knowledge about its capabilities and when presented with an item with unknown properties. This performance is relevant because perfect knowledge is not available in practical applications (R1, R3). The imperfect knowledge caused RSIR to not find the optimal action plan in a small fraction of tests on a variety of items (R5, R6). Additionally, the estimation of item properties was slightly worse. This is because RSIR was less able to reason about failures because of the noisy knowledge it had (R4). There was no significant change in number of grasping cycles RSIR required to stabilize.

9.3. Demonstration

This section performs the final evaluation of RSIR: its ability to implement a pick-and-place approach (R1). Because of the limited components available in the lab in Eindhoven (R19), it is not possible to evaluate the full pick-and-place approach. Instead, the robot has access to the following capabilities:

1. Loading the verification pose, which is the pose the robot must move to after grasping an item
2. Taking an RGB image of the input carrier.
3. Taking a depth image of the input carrier.
4. Using a parallel-jaw gripper (a Schunk gripper).
5. Loading a segmentation that does not include the edges of the carrier.
6. Synthesizing a grasp using the algorithm developed in this thesis.
7. Moving a virtual version of the robot to the grasp pose.
8. Activating the gripper.
9. Moving a virtual version of the robot to the verification pose.
10. Asking the human operator whether the grasp was successful.

When executed in this order, the robot completes the task. Because of the limited components available, RSIR can only use this action plan to complete the task. Hence this demonstration does not evaluate RSIR's reasoning capabilities. Instead, this demonstration proves that RSIR can be integrated with existing components to complete a grasping task (R1, R19). A video of the demonstration is available at `https:`

//youtu.be/X2FCn5cXoZs. The demonstration shows that RSIR correctly creates the action plan, communicates with the required components over ROS, detects failures, infers item properties and causes behind failures, and retries the grasping approach after failure.

In summary, this chapter evaluated RSIR by answering the competency questions, evaluating RSIR's performance for grasping an item with unknown properties, and demonstrating its implementation on a robotic set-up. RSIR was able to answer all competency questions. When grasping an unknown item, RSIR found the optimal grasping approach when doing sufficient explorations. Reasoning was better for items with properties that are common compared to those that are uncommon. Initial results indicated decent robustness against incomplete knowledge about capabilities. RSIR finished its reasoning within a reasonable amount of time. Finally, RSIR worked correctly when completing a grasping task using components of a physical robot.

10

Conclusion

This chapter concludes the development of RSIR: a pick-and-place robot with reasoning capabilities that make it more robust against failures, and a new grasping approach that allow it to handle more items. Together, the reasoning and grasping approach result in a higher pick-and-place capacity. In developing RSIR, this thesis answered the research question: *How can a pick-and-place task of a robotic arm with access to multiple grippers be modelled using knowledge representation and reasoning such that the system is robust against failures, such that it can handle a larger variety of unknown items, and such that it can be expanded to other related tasks?* Additionally, a main requirement for RSIR is that it must provide information about its reasoning process. The research question is divided into the following subquestions:

RQ1. What knowledge and reasoning capabilities should RSIR contain?

RQ2. What method best provides this knowledge and these reasoning capabilities?

RQ3. What software architecture best integrates this knowledge and reasoning with the robot?

RQ4. What is RSIR's performance on selecting the correct actions to be executed after the model and software architecture have been implemented?

Knowledge and Reasoning
To achieve the improved capacity, RSIR must contain knowledge (RQ1) about tasks, failures, and item properties. This knowledge is best captured in an ontology, because ontologies are expandable, can provide robustness, and provide insight into the robot's thought process. The best development method for the ontology is Methontology. The best approach to achieving robustness (RQ2) is though traded control and applying known capabilities in new ways (as opposed to developing qualitatively new behaviour).

Following Methontology, the ontology that best captures the knowledge (RQ2) is based on the Core Ontology for Robotics and Automation [106] and the Task Ontology [96]. These ontologies were extended to include all knowledge required by RSIR. The resulting ontology revolves around a robot being equipped with multiple devices, where each device provides one or multiple capabilities. A capability is considered a block with inputs and outputs, as well as constraints. RSIR combines capabilities into an ordered list called an action plan. When the actions in an action plan are executed consecutively, they complete the defined task. RSIR selects the best action plan according to a cost function based on the estimated probability of success and estimated execution time. Furthermore, item properties are modelled as probability density distributions because this allows RSIR to handle uncertainty. Capability constraints are modelled as confidence scores per item property. If an action plan is complete or fails, a Bayesian network infers the most likely item properties based on the prior knowledge of the item, the capabilities executed, and the constraints of each capability. It also provides information about which capability most likely caused the failure. Based on the information from the network, RSIR recomputes the costs of the action plans and selects a new plan to execute.

The software architecture (RQ3) that was selected to implement RSIR is based on a state machine that communicates with the ontology, reasoner, and other modules over ROS. The ontology was implemented in KnowRob because of its connection to ROS. The reasoner was implemented in Prolog (which is part of

KnowRob) and Python (for the Bayesian network).

Finally, RSIR's performance was measured (RQ4) by how much of the required knowledge it contained, by how often it selected the optimal action plan, by how well it estimated an unknown item's properties, and by how much time it required for its reasoning. RSIR contained all of the required knowledge with the exception of the knowledge required to device when to exploit knowledge and when to explore for new knowledge. When doing sufficient random explorations, RSIR nearly always found the optimal action plan, even when the knowledge it had about its capabilities' constraints was noisy. Furthermore, the estimation of item properties was generally better with explorations than without explorations. The estimations got worse when noise was introduced to RSIR's knowledge of the constraints on its capabilities. When provided with prior knowledge about the distribution of item properties in the system, the belief RSIR had about item properties improved for items that are common and declined for items that are uncommon. Finally, the time it took for RSIR to execute and reason about a task was 8.1 seconds when creating action plans, or 0.78 seconds when updating action plans. The number of pick-and-place attempts it required to find the optimal action plan was 8.6 on average when exploring often. This is reasonable with respect to the time other components of RSIR take.

The main limitation in RSIR's design is that is developed using the sense-plan-act paradigm. This works well for most of the implementations of related tasks, but it makes it so that RSIR is unsuited for feedback-heavy capabilities such as visual servoing. Additionally, RSIR cannot contain knowledge of dependencies between item properties. These two limitations are inherent to RSIR's design.

Furthermore, there are several limitations that could be improved given more time. The main limitation in this category is the simple algorithm RSIR uses for exploration. A better exploration algorithm could likely better infer item properties and achieve the optimal action plan in less pick-and-place cycles. A second limitation is that the components were implemented without a focus on code optimization, which increases runtime. For example, the function in RSIR that creates action plans uses the `rdf_assert` and `rdf_retract` functions, which are computationally expensive compared to saving knowledge in dynamic memory. Another important limitation is that the state machine currently does not distinguish between failures due to unavailable components and failures due to other causes.

Grasping Approach

The grasping approach most suited for RSIR (RQ2) uses a parallel-jaw gripper, because this gripper is best at grasping items that RSIR's vacuum gripper cannot handle. The approach synthesizes grasps based on detecting parallel lines in depth images. This approach is most suited for RSIR because of its simplicity, low computational cost, and high grasp success rate. Additionally, using depth information improves the evaluation metric, makes the approach less sensitive to visual textures, and allows the approach to detect the height of an item's supporting surface.

The resulting grasp synthesis algorithm was evaluated based on the variety of items it can grasp and its runtime. It was tested on a set of items encountered by RSIR. It can synthesize reasonable grasps for a variety of items, some of which are typically not graspable by a vacuum gripper, and it has a reasonable runtime of 1.57 seconds. The grasp metric is generally a good measure of how good a grasp is. Furthermore, the approach can work with a simple rectangular segmentation, as opposed to an item segmentation that is more computationally expensive to obtain.

The main limitations of the grasping approach are that it requires the items to have several (near) straight edges, and it requires most of the item to be detectable by the depth camera. It is also relatively slow compared to its equivalent in literature. This might be due to the changes in the algorithm (using depth data) or the implementation. A final limitation is the limited set of items the approach was tested on in this thesis.

11

Future Work

This chapter presents future work that can be done regarding RSIR. The first section presents future work regarding scientific research topics that have not been answered in this thesis. The second section, on the other hand, presents future work in engineering that must be done to make RSIR available for application in Vanderlande's systems.

Further Research

This section presents possible research topics for future work. The following topics could be interesting:

- The first potential topic for future research is how to effectively integrate feedback-heavy control capabilities into RSIR. This is an interesting topic because the feedback-heavy capabilities deviate from the sense-plan-act paradigm that is the basis for RSIR, and thus likely require a drastically different approach. Furthermore, a version of RSIR that can effectively implement such capabilities would be more general than the current version of RSIR. This generalizability is not only one of the goals of RSIR, but is also a goal of ontologies in general [87][130][131].
- The second potential topic for further research is intelligent exploration. In this thesis, RSIR explored randomly once every n cycles, but this approach wastes precious time if the selected action plan does not provide new information about the item, or if the best action plan is (likely) already used. Future work could focus on how to explore efficiently, thus answering the competency question "What tasks should the system do to complete a goal?" Ideally, this knowledge would be integrated into the ontology.
- A final topic could be to acquire information from the internet and add this to the ontology. Examples are the online search based on EAN-codes mentioned in this thesis, and the research being conducted by the researchers behind KnowRob that aim to learn from instruction videos on the internet or actions demonstrated in virtual reality [19]. Given the vast amount of information available on the internet, it is likely that RSIR can find useful information through such online searches. The research would focus on how to effectively search for this information and how to integrate it into the ontology.

Finally, future research could focus on an ontology for emergent behaviour and/or shared control. However, these topics are less related to RSIR because emergent behaviour and shared control were not considered in the design of RSIR.

Implementation Improvements

These improvements focus on further improving and testing RSIR to include it in Vanderlande's systems. This category of future work is divided into three levels of priority based on how important the work is in getting RSIR ready for warehousing applications.

High Priority

High priority work is work that must be done before RSIR can be fully tested. Exploration. Two more important points are to find a better algorithm for filling in the missing data in the depth image, and testing

the approach on a physical set-up. The former is important because the poor filtering is a limiting factor in synthesizing grasps. The latter is crucial because it could prove that the synthesized grasps actually lead to the robot grasping the item. Finally, it is important to have a closer look at the parameters, because they have a big influence on the runtime. Selecting a poor value for the peak factor, for example, results in an impractically long runtime. The answer to this problem most likely lies in the number of peaks parameter because it sets an upper bound for the number of detected lines and thus the runtime. This must be researched in more detail.

Medium Priority
Medium priority work considers testing RSIR on a larger set of items than considered in this thesis, and on the actual SIR set-up. To do so, the constraints on SIR's capabilities must be determined. This includes selecting the relevant item properties, discretizing said properties, and finding the confidence values per item property per capability. This can be done based on expert knowledge (likely inaccurate) or statistical analysis and/or machine learning (based on performance tests, likely more accurate). Note that this includes the grasp synthesis approach based on parallel lines. The conclusion of this thesis already provided some suggestions for creating four variants of the synthesis approach, based on the provided segmentation and the item. Furthermore, the state machine should be edited so that it distinguishes between component failures and unavailable components. This provides RSIR with the information required to draw correct conclusions.

Once the above work is finished, RSIR can be set up on the SIR system and it could run (unsupervised) for a while. RSIR could load and save its knowledge so that it can recall knowledge about items it has encountered before. This test should give good insight into how well RSIR estimates the items' properties and how well it selects action plans.

Low Priority
Finally, low priority work focusses on minor improvements that could be beneficial for RSIR in the long run. The are two main points for improvement regarding RSIR in general. The first point is to re-use actions done in a previous action plan. For example, if RSIR segments the input carrier to determine an item's size, it can re-use the segmentation for grasp synthesis. However, another example is that RSIR cannot re-use a segmentation after a grasp has failed, because the robot arm might have moved the items. The questions of when and how RSIR can re-use information could be the subject of future work. The second point of improvement for RSIR is code optimization, especially regarding the creation of action plans and the grasp synthesis algorithm.

Finally, there are four minor points of improvement for the grasp synthesis algorithm. The first point is to investigate more evaluation metrics for the grasp synthesis approach. For example, it could into account the item's centroid, or the item's height with respect to the other items in the carrier. Similarly, more algorithms could be investigated for detecting edges and lines. The third point is to expand the approach to more grasps besides top grasps, for example by determining the angle of the item's top surface. The final point is to implement the synthesis approach in a language other than Matlab, because then the point cloud can be passed via ROS, the approach can be started via a ROS launch file, and it likely improves runtime.

Appendices

A

GS1 Product Categories

This appendix presents the GS1 definitions for the five categories for which an additional gripper can achieve the most improvements for RSIR. The definitions are as shown as in Figure A.1.

Name	Example	GDSN code	Description
Not packed		NE	The item is supplied without packaging or 'unpackaged'.
Net		NT	A package with an open structure of threads or strips, woven in a regular pattern with openings between the threads. The packaging is intended for holding or carrying products.
Card		CM	A flat packaging on which the product is mounted or is attached to for presentation purposes.
Box		BX	A non-specific term referring to a non-flexible, three-dimensional packaging with closed sides completely enclosing the contents and may be made of any material. Although some boxes are suitable for reuse or reclosing, they may also be single use, depending on the product hierarchy.
Blister pack		BPG	A type of packaging where the item is packaged between a preformed 'blister' (usually made of transparent plastic) and a cardboard 'carrier'. These may be attached to each other by means of staples, heat welding, gluing or in some other way. In some cases, the blister packaging is completely wrapped around the product like an oyster shell. Blister packs are usually made from polyvinyl chloride by thermoforming, but almost any thermoplastic may be formed into a blister.
Cylinder		CY	A non-flexible, cylindrical packaging with straight sides and circular ends of equal size.

Figure A.1: Definitions for the GS1 categories for which an additional gripper can achieve the most improvements for RSIR. Adapted from [97].

B

Implementations of Related Tasks

This appendix presents the results of the literature study done to find implementations for related task RSIR should be expandable to. Each implementation is broken down into capabilities. This appendix follows the same structure as the list of related tasks in section 6.1.2.

Item Variety

- Switch between grippers. Vanderlande implements this by having a rack in which grippers are hanging located next to FM-GtP. The grippers are modular and can be swapped through a pneumatic system. To swap grippers, the robot moves to a free spot in the rack, deactivates the pneumatic link, moves to an occupied spot, and activates the link. Capabilities: `Move robot arm`, `Activate pneumatic link`, `Deactivate pneumatic link`, `Find free spot in gripper rack`, `Find gripper in gripper rack`

- Do preshaping. A preshape is defined as a gripper pose [135]. Hence making the gripper use a preshape is as simple as setting its pose. The main difficulty in using preshapes is the algorithms that determines when to use which preshape. To provide the knowledge of the preshape to RSIR, the gripper can be included once per preshape. For example, consider the gripper in Figure B.1. Capabilities: `Select three-fingered gripper (spherical)`, `Select three-fingered gripper (cylindrical)`, `Select three-fingered gripper (precision tip)`, `Select three-fingered gripper (hook)`



Figure B.1: Preshapes for a three-fingered hand [90]. From left to right: spherical, cylindrical, precision-tip, and hook grasps.

- Shift items. Items can be shifted by closing the gripper, then pushing the gripper onto the item and moving the gripper parallel to the bin. The approach by Berscheid et al. determines when and how to shift based on machine learning [22], but a similar approach to shifting could be implemented analytically. Capabilities: `Close gripper`, `Take RGB image`, `Take depth image`, `Determine item position`, `Move robot arm`, `Apply vertical force`, `Move gripper horizontally`

- Limit the grasp force. The main challenge in this action is determining what the maximum grasp force is. Once this is determined, setting the limit is simple. Capabilities: `Determine maximum grasp force`, `Set maximum grasp force`

Three other aspects regarding item variety are types of clutter, item flows, and supporting surfaces. Because the implementation differs per task, these aspects are discussed for grasping and placing separately.

Grasping

- Synthesize grasps for each type of gripper RSIR has. The literature study done prior to this thesis provides an overview of synthesis approaches for grippers with two or more fingers [135]. The approaches can be analytical (based on item and/or gripper models) or data-driven (based on prior experience). Some analytical approaches simplify the item model to make synthesis easier. Note that the differences in grasp synthesis approaches mostly come from the algorithms used. Hence most approaches can be considered a "synthesize grasp" capability, for which the required inputs differ (for example, depth data or an item's primitive shape). Capabilities: `Take depth image`, `Determine item's principal axis`, `Determine item's primitive shape`, `Synthesize grasp via simulation`, `Estimate item shape`, `Load item model`, `Map grasp from known model to sensory data`, `Synthesize grasp`

- Recognize a specific item in the input carrier. The most straightforward way to recognize a specific item is through vision, as machine vision approaches have proven useful in detecting objects and cameras are already present in SIR [24]. Detection algorithms can be based on RGB images, depth images, or both. Capabilities: `Take RGB image`, `Take depth image`, `Detect item in RGB image`, `Detect item in depth image`, `Detect item in RGB and depth image`

- Synthesize a grasp for one specific item in the carrier. Implementations for this are:

  - An extension of the approach currently used by Vanderlande [125]: instead of passing the full segmentation mask to the synthesis algorithm, it could pass only the segment that contains the target item. Capabilities: `Segment image`, `Detect target item`, `Isolate target item from segmentation`
  - Another implementation is to determine the target item's location, and to include this in the synthesis approach's metric [135]. An example of such a metric is a metric that minimizes the distance between the gripper centre and the target item's centroid. Capabilities: `Detect target item`, `Determine target item position`

- Search for a non-visible item. An approach based on heuristics moves non-target items within a compartment, or between compartments, to make the underlying items visible. The system remembers where previous target items were detected and searches there. If none were detected previously, the system chooses items based on how likely they are obscuring other items (that is, their size and their height in the carrier). Capabilities: `Segment image`, `Synthesize grasp`, `Move robot arm to grasp pose`, `Move robot arm`, `Determine item size`, `Determine item height`, `Recall positions of previous target items`

- Grasp an item in specific location. Again, this can be done by including the location in the synthesis approach's metric [135]. An example is a metric that selects the item with the largest average height. This makes the synthesis approach grasp the topmost item. This functionality is included in the synthesis approach and is thus not a separate capability. Capabilities: none

Types of Clutter

- Grasp an item in isolation. Many grasping approaches in literature consider items in isolation, as this is the simplest type of item placement [135]. Capabilities: `Synthesize grasp`

- Grasp an item in sparse clutter. Items in sparse clutter could be picked up with synthesis approaches for items in isolation, as long as a segmentation is provided in which only one item is present [125]. Otherwise, approaches that work on dense clutter can be used [135]. Capabilities: `Segment image`, `Synthesize grasp`

- Grasp an item in dense clutter. Some synthesis approaches are designed to handle dense clutter. An example of a consequence is that these approaches cannot assume the target item is on a supporting surface at a known height. These approaches can also be used for items in sparse clutter or isolation [135][125]. Capabilities: `Synthesize grasp`

- Grasp a densely packed item. Densely packed items may pose problems. For example, the items in the corner of a carrier cannot be picked up with a fingered gripper, as this gripper requires two sides of the item to be approachable. In this case, the item could be shifted such that it can be picked up. Shifting

items was considered earlier in this appendix. Capabilities: `Detect ungraspable item, Determine shift direction`

- Grasp an item in a narrow space. Implementations for this are:
  - The general approach is synthesizing grasps and checking for collisions afterwards, and leaving navigating around obstacles to the motion planner citeSchooten[121]. Capabilities: `Synthesize grasp, Verify grasp validity, Create motion plan, Move robot arm`
  - Another approach is to explicitly include the environment in the grasp synthesis. For example, the environment can be included through depth information or a 3D model of the environment [135][120][90]. Capabilities: `Take depth image, Load environment model, Synthesize grasp`
  - Yet another approach to grasping in narrow spaces is through reinforcement learning, thus having the system learn by itself to detect narrow spaces and navigate around them. These approaches generally take an RGB image of the scene as input and output the desired motion command [75][61]. Capabilities: `Take RGB image, Compute motion command, Move robot arm`

Item Flows

The implementation for grasping depends on how predictable the item flow is:

- For predictable flows, a general approach is to predict the position of the item on the conveyor using the conveyor velocity, and to plan a trajectory based on that prediction. This approach requires precise calibration. Capabilities: `Know/Determine conveyor velocity, Predict item position, Synthesize grasp`
- For less predictable flows, a common approach is visual servoing. This requires less calibration, but does require continuous sensing and computing of the motion command. Visual servoing allows the robot to handle unexpected disturbances and uncertainty [62][92][75]. The camera can be either static or attached to the robot, and the motion command can be computed analytically or through machine learned algorithms. An example for analytical computation tracks the item using its known 3D shape [62]. Machine learning algorithms can be based on neural networks and/or reinforcement learning [92][75]. Machine learning approaches typically benefit from cameras attached to the robot, because the relative visual input makes it easier to learn [92]. Capabilities: `Take RGB image, Detect item, Detect robot arm in image, Synthesize grasp, Compute motion command`

Supporting Surfaces

There are several implementations for grasping from different supporting surfaces:

- Assume supporting surface. This is the easiest solution but also the least general one. A handful of grasp synthesis approaches make this assumption [135]. One example is the approach by Baumgartl and Henrich considered in this thesis [17]. Capabilities: `Load surface height, Synthesize grasp`
- Estimate supporting surface. This is a more common approach to handling surfaces and usually makes use of a depth camera [135]. The grasp synthesis approach developed in this thesis falls under this category. For approaches that assume a flat surface, the process could be split into two capabilities: estimating the height and synthesizing the grasp pose. However, it is more common to directly pass the depth data to the synthesis approach. Capabilities: `Sense depth, Estimate surface height, Synthesize grasp`
- Detect supporting surface. If the supporting surface is not estimated, it can be detected via force feedback: the gripper is slowly lowered until a force is detected, which indicates that the gripper has reached the item or supporting surface. This is currently being researched by Vanderlande. Capabilities: `Move robot arm, Sense force, Detect supporting surface via force`
- Grasp from a vertical stack. The approach that won the Amazon Robotics Challenge in 2016 makes no clear distinction between the horizontal and vertical stacks (shown in Figure B.2) when it comes to grasping [30]. Instead, they define four motion primitives: suction grasp from the top, suction grasp from the front, suction grasp from the side, and pinch grasp (using two fingers). Their metric, based on the item and the surrounding clutter, takes care of selecting the proper primitive. Capabilities: `Take depth image, Synthesize grasp, Select motion primitive`



Figure B.2: The horizontal stack (bin, left) and the vertical stack (shelf, right) handled by the system by Corbato et al. [30].

- Take an item from a human (or robot). From a robot's point of view, this is not significantly different from other grasping challenges (for example, grasping hanging goods or grasping items that move unexpectedly). The main challenge comes from the risk in human-robot interaction. One way to decrease this risk is to use haptic feedback for recognizing when the robot has grasped the human instead of the item [53]. A support vector machine can recognize this based on the sensed force and deformation. Capabilities: `Sense force, Sense deformation, Detect grasp on human`

- Catch an item. An implementation for this is to predict the item's motion and catch it where its trajectory intersects with a predefined horizontal plane [**?**]. The main challenge in this is the required high-frequency feedback loop (visual servoing), and designing the algorithm that detects the item [**?**][**?**]. Detecting the item is easier when using cameras with a viewing direction perpendicular to the item's trajectory [**?**][**?**]. Capabilities: `Sense RGB image, Detect item, Compute expected motion, Compute grasp location, Move robot arm`

Placing

The implementations of tasks regarding placing are:

- Place the item stably. Two implementations for this are:

  - The approach by Baumgartl et al. places sensor-modelled items stably on non-planar surfaces based on the item's geometry [18]. The approach consists of four steps: selecting a location on the placement area suitable for the item, orienting the item, computing the placement pose, and computing the quality metric. Orienting can be done according to the item's principal axis or semantic need (e.g. an upright orientation). Figure B.3 illustrates the approach. In step 1, the placement area is selected and the item is oriented. It is lowered until it touches the supporting area, and is then rotated until there are three points of contact between item and supporting area. The metric is evaluated (using the item's centre of mass and coefficient of friction) to confirm that the item is in static equilibrium. Capabilities: `Find suitable placement area, Orientate item for placing, Compute placement pose`

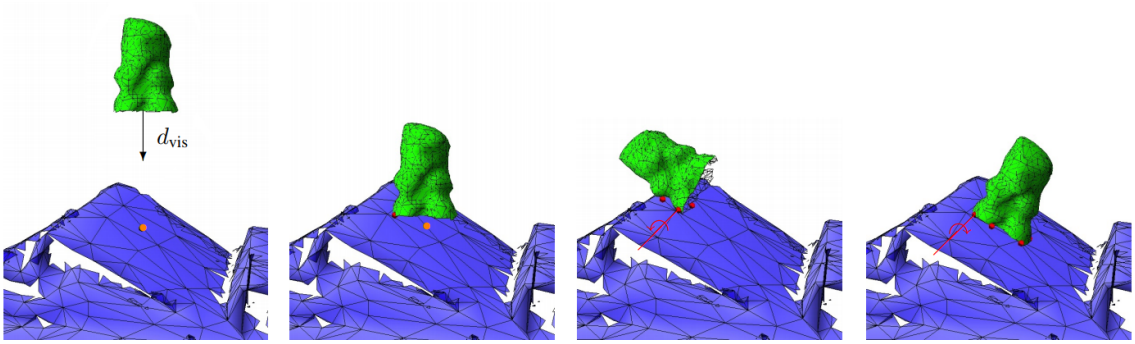


Figure B.3: Illustration of some of the steps in the placing approach by Baumgartl et al., adapted from [18]. From left to right: the placement area and orientation of the item, first contact between item and supporting area, a result for a left-handed rotation, and a result for a right-handed rotation.

  - Another option is to run a physics simulation to determine which item poses are stable [18]. The disadvantages are that the environment must be modelled, and that simulations are relatively computationally expensive, especially for deformable items [90][113]. Capabilities: `Create item model, Create environment model, Run placement simulation`

- Place an item such that it can be picked up again. An implementation of this places items such that they are densely packed but have enough distance between them to be picked up again [17]. The carrier in which items are placed is divided into rectangles based on the items' bounding boxes. Figure B.4 illustrates these rectangles after two items have already been placed. Capabilities: `Detect empty areas, Segment carrier into rectangles, Detect item footprint, Select rectangle segment for placement`

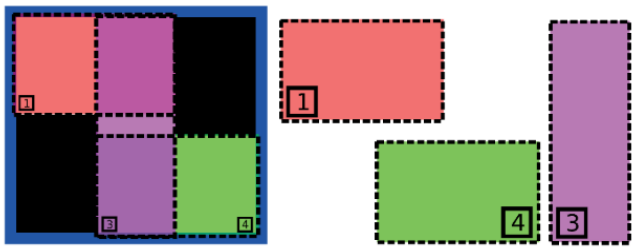


Figure B.4: The rectangular segments created by the algorithm by Baumgartl et al. [17]. Two items have already been placed in the carrier (black). The remaining areas are sorted based on the item's bounding box.

- Estimate the item's pose. Implementations for this are:

  - One implementation is to match the sensory (depth) data to a known item model [19]. Since the pose of the model is known, the pose of the detected item can be determined. However, this does require the item model to be known. Capabilities: `Sense depth, Load item model, Compare sensory data to item model`
  - An item's pose can also be determined through its principal axis, as done by Baumgartl et al. [17]. The principal axis can be computed given (noise, incomplete) geometrical data of the item. Capabilities: `Take depth image, Determine principal axis`
  - Another common approach in describing an item's pose is by using its primitive shape. Examples of primitive shapes are spheres, cylinders, and cones. A primitive shape can be determined from the item's (noisy, incomplete) geometrical data and the pose of the shape can be saved. Capabilities: `Take depth image, Determine item's primitive shape`
  - Semantic knowledge. A final way to estimate an item's pose is through semantic knowledge, for example by recognizing certain parts of the item (such as the handle of a mug [11]), or recognizing the orientation of an item (such as "upright" [18]). Capabilities: `Take depth image, Take RGB image, Detect semantic item part`

Types of Clutter
When it comes to placing, the clutter present around the supporting surface can be considered obstacles [17] or part of the supporting surface [18]. Because of this, placement algorithms do not explicitly consider the type of clutter.

Item Flows
The implementation for item flows for placing is very similar to that for grasping:

- For predictable flows, the position of the (items on) the conveyor can be estimated, and the robot's trajectory can be computed to match said position [**?**]. Capabilities: `Know/Determine conveyor velocity, Predict item position, Synthesize place pose`
- For less predictable flows, the robot can make use of visual servoing [62] [92][75]. Capabilities: `Take RGB image, Track belt, Track item on belt, Track robot arm, Synthesize place pose, Compute motion command`

Supporting Surfaces
Implementations for placing on different supporting surfaces are:

- Drop the item, which is the easiest solution and which is commonly used [135]. However, the item might get damaged and the approach likely does not fill the carrier efficiently. Capabilities: `Load drop location, Move robot arm, Deactivate gripper`
- Assume surface height. As with grasping, this is easy but not general. Capabilities: `Sense output depth, Estimate surface height, Synthesize place pose`
- Sense supporting surface. Again, this approach is similar to the approach for grasping. Capabilities: `Move robot arm, Sense force, Detect supporting surface via force`
- Hand an item to a human. The study by Huber et al. shows that the performance of a robot handing an item to a human is surprisingly efficient, despite the robot using a (relatively simple) minimum-jerk velocity profile with a set end position [57]. The hand-off is approximately 1 second slower compared to when a human is handing the item to a human, mainly because the robot waits after the human has taken the item from the robot's gripper. This study shows that a simple algorithm already results in fast human-robot hand-off. However, it might be beneficial to have the robot partially deactivate its gripper when presenting the item to the human, so that the human does not need a lot of force to pull the item out of the gripper. Capabilities: `Compute motion plan, Move robot arm, Wait for human, Partially deactivate gripper`
- Throw the item. This is done at Vanderlande by moving the robot arm at high velocity and deactivating the gripper at the correct time. Capabilities: `Move robot arm, Compute deactivation moment, Deactivate gripper`

Robot Movement
The implementations of tasks regarding robot movement are:

- Avoid static obstacles. Known, static obstacles can be included in motion planning software such as MoveIt! [28]. This makes the motion planner automatically avoid collisions with said obstacles. Measurements from depth cameras could add runtime information. Additionally, a shape could be added to the simulated gripper to take into account that not only the gripper but also the item in the gripper should not collide. Capabilities: `Update environment model, Add item to model, Create motion plan, Move robot arm`
- Avoid dynamic obstacles. Avoiding (unpredictable) dynamic obstacles requires continuous feedback, as opposed to the sense-plan-act approach commonly used by motion planners such as MoveIt! [28]. Visual servoing is a common way of implementing this. Although the exact algorithm may differ (such as which camera data to use), the common factor in visual servoing approaches is the continuous feedback [92][75][61][62]. Capabilities: `Take RGB image, Take depth image, Move robot arm, Compute motion command`

- Scan barcodes on items. This can be implemented by moving the item to the camera and then rotating the item to show its different surfaces. If the barcode is occluded by the gripper, the robot places the item back on the table in a different position and tries again. This approach can use off-the-shelf barcode scanning software [64]. Capabilities: `Rotate item in-hand, Take RGB image, Detect barcode, Move gripper, Grasp item, Place item`
- Move with different velocities. After the reasoner has determined with what velocity the robot arm should move, RSIR can enter this value in the motion planner and/or executer [28]. Capabilities: `Set robot arm velocity`

Human interaction

The implementations of tasks regarding human interaction are:

- Allow the operator to determine whether an action was successful. The operator can answer a yes/no question on whether the action was successful. This could be entered via the command line or integrated in FM-GtP's user interface. Capabilities: `Ask operator for confirmation`
- Allow the operator to edit the knowledge base. The operator could edit the knowledge base with the tool that the ontology is created in, if this tool is user-friendly enough. Another option is to have the operator update the knowledge base through the FM-GtP user interface. This editing task is not part of the ontology as that would lead to the ontology editing itself, possibly creating errors. Capabilities: `none`
- Inform the operator of failures that occur. RSIR could print information to the command line, or be integrated with FM-GtP's user interface. If RSIR is implemented in middleware such as ROS, it could also use ROS's logging system as a basis for communicating failures to operators. This is not a separate capability, because RSIR must always communicate about the actions it undertakes. Capabilities: `none`
- Estimate item properties. This task has a large variety of implementations and has thus been given its own section in this appendix.

Reasoning

The implementations of tasks regarding reasoning are:

- Reason whether RSIR can handle an item before the input carrier arrives. Based on the known properties of an item, RSIR should determine whether an item can be handled or not. This is a query to the knowledge base and therefore not included in the ontology. Capabilities: `none`
- Implement learning to predict whether an item can be handled. The resulting algorithms can be considered black boxes that require an input and result in some outputs, usually including a confidence score. For example, a machine-learned grasp synthesis approach can take a depth image and gripper parameters as input and output a grasp pose and confidence score [135]. In this case, the black box is the capability `Synthesize grasp`. Hence learning is considered at implementation the implementation level, and does not introduce any new capabilities. Capabilities: `none`

Failure Detection

The implementations of tasks regarding failure detection are:

- Detect whether there is an item in the gripper. There is a large variety of implementations for this:
  - Vanderlande detects items by determining the item's footprint. This is done by taking an RGB image of the gripper from below. If the footprint of the item is non-zero, it can be concluded that there is an item in the gripper [128]. Capabilities: `Take RGB image, Determine item footprint`
  - Another approach is to use a vacuum sensor in the vacuum gripper. If the pressure is sufficiently low compared to the atmospheric pressure, it can be concluded that there is an item in the gripper [30][128]. Capabilities: `Select vacuum gripper, Sense pressure in gripper, Verify item through pressure`

  - For a parallel-jaw gripper, it might be that the command to close has been executed but the position reading of gripper is (still) larger than 1cm. In this case, it can be concluded that the gripper is not fully closed and therefore there is an item in the gripper. However, during tests it was found that this approach often did not work for thin items [75]. Capabilities: `Select gripper with position sensors`, `Activate gripper`, `Sense gripper position`, `Verify item through gripper position`
  - It is also possible to take an image of the carrier, attempt a grasp, and take another image of the carrier. The images before and after grasping can be compared to determine whether there was an item in the gripper [75]. This approach used an RGB camera for taking the image, but this could also be done with a depth camera. Additionally, it can also be used for images of the output carrier before and after placing the item. Capabilities: `Take RGB image`, `Take depth image`, `Deactivate gripper`, `Verify item by comparing images`
  - A similar approach to comparing images before and after grasping is to segment the image before and after grasping, and count the number of items in each of them. Depending on the difference, it can be concluded whether there is an item in the gripper. Capabilities: `Take RGB image`, `Take depth image`, `Segment RGB image`, `Segment depth image`, `Segment RGB and depth image`, `Verify item by comparing segmentations`
  - If there is a scale underneath the carrier, the mass of the carrier before and after grasping can be compared. If the difference is sufficient, it can be concluded that the item is in the gripper [93]. Capabilities: `Sense carrier mass`, `Verify item by comparing carrier masses`
  - The load on the robot arm can be measured and compared to the load when there is no item in the gripper. If the load is sufficiently larger, it can be concluded that there is an item in the gripper. Capabilities: `Sense arm load`, `Verify item by comparing arm loads`

- Determine whether a specific item is in the gripper. Implementations for this are:

  - Most of the approaches for "Detecting whether there is an item in the gripper" can be used, and their values can be compared to the value corresponding to the target item. For example, the footprint of the item could be compared to a list of footprints in a database. Capabilities: `Load correct item property`, `Verify specific item by comparing properties`
  - If an RGB image of the item in the gripper is available, a Siamese-like neural network can be used to compare an image of the grasped item with product images in a database [145]. Capabilities: `Take RGB image`, `Load product image`, `Verify specific item by comparing item images`

- Detect how many items are in the gripper. Again, the images before and after grasping or dropping can be (segmented and compared) to determine how many items were in the gripper [75]. Capabilities: `Take RGB image`, `Take depth image`, `Segment RGB image`, `Segment depth image`, `Segment RGB and depth image`, `Verify items in gripper by comparing segmentations`

- Detect the stability of a grasp. One approach for this makes use of detecting slip. The more the item slips, the more unstable the grasp. Slip can be measured using tactile sensors. Additionally, this approach increases the grasping force until slip is no longer detected [103]. `Select gripper with tactile sensors`, `Detect slip from tactile measurements`, `Compute grasping force`, `Set grasping force`

Item Property Estimation

The implementations for estimating item properties are presented in Table B.1. The properties considered are based on those discussed section 6.1.3. Besides the capabilities, the required hardware is listed per implementation. This gives a clear overview of what hardware must be added to RSIR to implement the estimation of item properties.

Table B.1: Approaches in literature for determining item properties in tasks that make use of grippers.
The "required hardware" column does not include some hardware components that are already required for SIR, such as the robot arm and gripper.

Property	Approach	Required primitive actions	Required hardware	Source
Centre of mass	Grasp the item three times at different locations with two fingers. For each grasp, measure the difference in item angle before and after lifting the item. Determine centre of mass and friction coefficient through force balancing. Requires item to have a known, uniformly-distributed mass. Approach uses a 2D item.	Synthesize grasp on specific area of item, Move robot arm, Activate gripper, Take RGB image, Compute angular displacement from RGB images	Hardware to sense angular displacement	[146]
Friction coefficient (between item and gripper)	Grasp the item three times at different locations with two fingers. For each grasp, measure the difference in item angle before and after lifting the item. Determine centre of mass and friction coefficient through force balancing. Requires item to have a known, uniformly-distributed mass. Approach uses a 2D item.	Synthesize grasp on specific area of item, Move robot arm, Activate gripper, Take RGB image, Compute angular displacement from RGB images	Hardware to sense angular displacement	[146]
	Grasp the item and push it downwards onto the supporting surface. Detect when slip occurs between gripper and item using gripper position measurements. Determine friction coefficient from pushing force and grasping force.	Synthesize grasp on specific area of item, Move robot arm, Activate gripper, Apply (known) vertical force, Sense grasping force, Sense gripper position, Detect slip from gripper position measurements	Grasping force sensor, gripper position sensor	[123]
	Determine the friction coefficient from the acoustic signals that occur when a tactile sensor slides over the item, or when the gripper taps on the item	Slide over item, Tap item, Record sound	Microphone	[84]
Friction coefficient (between item and supporting surface)	Push the side of the item until the item starts to slip, determine friction coefficient from applied force. Detect slip through position measurements of the gripper.	Sense grasping force, Sense gripper position, Push item, Detect slip from position measurements	Grasping force sensor, gripper position sensor	[123]
Mass	Sense the carrier mass using a scale under the carrier. Determine the item mass by comparing the carrier masses before and after grasping the item.	Synthesize grasp, Move robot arm, Activate gripper, Sense carrier mass	Scale under carrier	-
	Push the side of the item above its centre of mass until it topples over. Mass is determined from the moment applied by the gripper right before the item topples over. If possible, place a second finger on the other side of the item to prevent slip. Requires touch sensor for deformable items for detecting toppling.	Determine item shape (to determine where to push the item), Push item, Sense grasping force, Sense grasping position, Detect toppling in position/force measurements	Grasping force sensor, gripper position sensor	[123]
	Estimate mass based on the difference in torque measurements in the robot arm before and after grasping	Synthesize grasp, Move robot arm, Activate gripper, Sense robot arm torque	Torque sensors in robot arm	-
Material	Determine material from the contact angle between gripper and item sensed by a tactile sensor, or from the tactile sensor electrode data.	Touch item, Sense tactile, Determine item material from tactile measurements	BioTac tactile sensor	[84]
	Detect material through vibrations that occur during interactions between the item and the sensor, e.g. the sensor sliding over the item. Sense vibrations with strain gauges, force sensors, tactile sensors, and/or accelerometers	Slide over item, Sense strain, Sense force	Strain gauges, force sensors, tactile sensor, and/or accelerometers.	[84]
	Detect material from RGB image.	Take RGB image, Detect material in RGB image	RGB camera	[84]
Shape	Determine shape from visual measurements, for example RGB and depth images. However, limited use in occlusion or poor illumination conditions.	Take RGB image, Take depth image, Determine item shape from RGB/depth image	RGB camera, depth camera	[125] [84]
	Obtain several tactile measurements and transform these into a point cloud. Then fit a geometric model to the points, for example a superquadric. Get the item shape from the model's contour. Other approaches are based on machine learning rather than point clouds. Some of these approaches require the item to not move.	Touch item, Sense tactile, Determine item shape from tactile measurements	Tactile sensor	[84]
	Estimate local shape from tactile measurements. The main challenge in this approach is extracting the correct features from the data.	Synthesize grasp, Move robot arm, Activate gripper, Touch item, Sense tactile, Determine local item shape from tactile measurements	Tactile sensor	[84]
Stiffness	Grasp the item and increase the force until the item is dented, while measuring the applied force. Detect dent through position measurements of the gripper.	Synthesize grasp, Move robot arm, Activate gripper, Set grasping force, Sense grasping force, Sense gripper position, Detect denting from position measurements	Variable-force gripper, grasping force sensor, gripper position sensor	[123] [84]
	Determine stiffness from the contact angle between gripper and item sensed by a tactile sensor, or from the tactile sensor electrode data.	Touch item, Sense tactile, Determine stiffness from tactile measurements	BioTac tactile sensor	[84]
	Knock on the item and determine the stiffness (and elasticity and hardness) from accelerometer measurements	Sense acceleration, Knock on item, Determine stiffness from acceleration measurements	Accelerometers	[84]
Surface roughness	Detect surface roughness through vibrations that occur during interactions between the item and the sensor, e.g. the sensor sliding over the item. The sensor(s) can be strain gauge, force sensor, tactile sensor, and/or accelerometers	Slide over item, Sense strain, Sense force, Sense tactile, Sense acceleration, Detect surface roughness from measurements	Strain gauges, force sensors, tactile sensor, and/or accelerometers	[84]

C

Details on the Resulting Ontology

This appendix presents more details on RSIR's ontology. Section C.1 presents an introduction to terminology used in the field of ontologies. Secondly, C.2 provides the definitions of the concepts in RSIR's ontology. Finally, C.3 presents an example for a simple robot that uses RSIR to reason about a grasping task.

C.1. Introduction to Ontology Terminology

This appendix discusses some basic terminology in the field of ontologies. This terminology must be understood to fully understand RSIR's ontology. Some of the terms discusses in this appendix are specific for the implementation of RSIR. This implementation is discussed in chapter 7. However, these terms are necessary for understanding RSIR's ontology and are therefore included in this appendix.

RSIR's ontology is implemented in the Web Ontology Language (OWL). In OWL, resources are defined by their Universal Resource Identifiers (URIs). For example, RSIR defines a robot as `http://www.semanticweb.org/vi#Robot`. This URI can be split into the namespace (`http://www.semanticweb.org/vi#`) and the resource (`Robot`). By defining a namespace `vi:` as `http://www.semanticweb.org/vi#`, we can simplify the URI to `vi:Robot` to improve readability [112].

OWL defines relations between resources through triples. A triple consists of a Subject, Predicate, and Object as shown in Figure C.1. A predicate can have its range and domain defined. For example, the Predicate `vi:robotPart` has a domain of `vi:Robot` and a range of `vi:Device`. Thus `vi:robotPart` can only create relations between robots and devices. Predicate names always start with a lower case letter. They are occasionally referred to as Properties. An ontology in OWL consists mostly of a collection of triples. The ontology can be saved in one of several formats, one of which is XML [112].

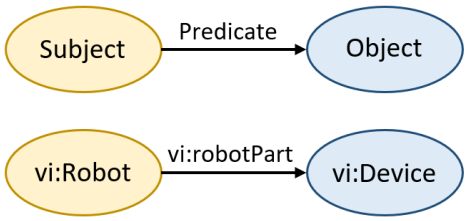


Figure C.1: The definition of a triple (top) and an example of a triple (bottom).

Furthermore, an important distinction in OWL is between Classes and Individuals. This distinction is very similar to that between Classes and Instances in object-oriented programming: Individuals can have relations with other Individuals and data types, whereas Classes cannot. For example, the class `vi:Robot` can have the predicate `vi:robotPart` with `vi:Device`, but it is impossible to define an execution time of 5 seconds for `vi:Robot`. Instead, `vi:robotA`, which is an individual of `vi:Robot`, can have an execution time of 5 seconds. Class names always start with an upper case letter, whereas Individual names always start with a lower case letter. OWL also contains the subclass relation through the predicate `rdfs:subClassOf`. Besides Classes

and Individuals, OWL contains the definition of Datatypes. Examples of Datatypes are integers, strings, and booleans [94][112].

The difference between Classes and Individuals has consequences for Predicates: Predicates fall apart into Object Properties and Data Properties. For Object Properties, the Subject and the Object are classes or individuals. An example is `vi:Robot vi:robotPart vi:Device`, where `vi:robotPart` is the object property. For Data Properties, on the other hand, the Subject is an individual and the Object is a datatype. An example is `vi:robotA vi:executionTime 5.0`, where `vi:executionTime` is the data property. Characteristics can be defined for object properties, such as symmetry and inverses. For example, an object property `ex:hasSpouse` would be symmetric, because if `ex:personA ex:hasSpouse ex:personB`, then also `ex:personB ex:hasSpouse ex:personA`. An example of inverse is that `vi:provides` has the inverse `vi:isProvidedBy` [94][112].

A final piece of basic information is that many OWL ontology editors come with a reasoner. This reasoner can infer knowledge from what is defined in the ontology. For example, if the ontology contains the knowledge that `ex:personA ex:hasSpouse ex:personB` and that `ex:hasSpouse` is symmetrical, the the reasoner infers that `ex:personB ex:hasSpouse ex:personA`. Reasoners are also useful for debugging, because they detect inconsistencies in the ontology. For example, if `vi:camera1` and `vi:camera2` are both `vi:Devices` and the triple `vi:camera1 vi:robotPart vi:camera2` is defined, then the reasoner gives an error. This is because `vi:camera1` is not a `vi:Robot` and `vi:robotPart` is only defined for subjects that are `vi:Robots` [94][112]. Note that the reasoner in the editor is not the same as the reasoner that is referred to as "RSIR's reasoner" in this thesis.

C.2. Definitions of Concepts in the Resulting Ontology

This appendix provides the definitions for the concepts in RSIR's ontology. Concepts solely used for organisation purposes (such as Moving Capability and Gripper Knowing Capability) are not included because such concepts can be freely added, removed, or renamed if the application requires so. The concepts in RSIR are:

- **Abstract** [59]: Properties or qualities as distinguished from any particular embodiment of the properties/qualities in a physical medium. Instances of Abstract can be said to exist in the same sense as mathematical objects such as sets and relations, but they cannot exist at a particular place and time without some physical encoding or embodiment.
- **Action Plan**: A list of capabilities that, when followed by the robot in the appropriate order and executed successfully, results in the outputs as defined by each capability in the list.
- **Actuating Capability**: A capability that allows the robot to move and act in the surrounding environment (based on the CORA definition for robotActuatingPart [59]).
- **Actuating Device**: A device that allows the robot to move and act in the surrounding environment (based on the CORA definition for robotActuatingPart [59]).
- **Agent** [59]: Something or someone that can act on its own and produce changes in the world.
- **Attribute** [59]: Qualities that we cannot or choose not to reify into subclasses of object.
- **Capability** [96]: A generic term referring to an action the robot has been programmed to perform, a skill the robot provides, or a behaviour the robot has available.
- **Cardinality Constraint**: A constraint concerning the number of similar capabilities a robot can use.
- **Communicating Capability**: A capability that serves as an instrument in a robot-robot or human-robot communication process by allowing the robot to send (or receive) information to (or from) a robot or human (based on the CORA definition for robotCommunicatingPart [59]).
- **Communicating Device**: A device that serves as an instrument in a robot-robot or human-robot communication process by allowing the robot to send (or receive) information to (or from) a robot or human (based on the CORA definition for robotCommunicatingPart [59]).
- **Constraint:** A limitation of a capability.

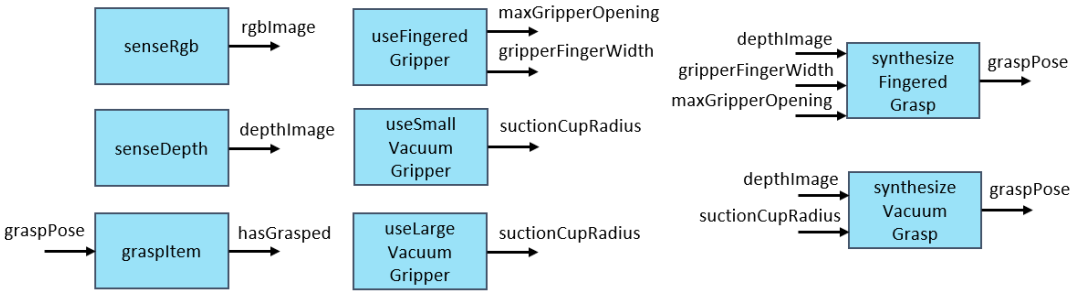
- **Default Property**: An item property that is given a default probability density function, and that is considered when reasoning about the item's properties.
- **Device** [59]: A device is an artefact whose purpose is to serve as an instrument in a specific subclass of a process.
- **Entity** [59]: The universal class of individuals. This is the root node of the ontology.
- **Goal** [96]: The externally defined desired end (or continuing) state of the system. Note that the goal is the action that the operator or other external entity wants the robot to do. If the task is decomposed to subtasks, the goal is the desired end (or continuing) state of each subtasks, as defined by the task.
- **Input/Output**: Any possible abstract or physical outcome of a capability, such as a robot state (e.g. an activated gripper) or gained knowledge (e.g. the bounding box of an item).
- **Item Property**: An item property is an abstract representation of a physical aspect of an item. Examples are Mass, Primitive Shape, and EAN Code.
- **Item Property Constraint**: A constraint regarding the item properties a capability can handle.
- **Knowing Capability**: A capability that allows the robot to load (or save) knowledge from (or to) a knowledge base.
- **Knowing Device**: A device that allows the robot to load (or save) knowledge from (or to) a knowledge base.
- **Object** [59]: Corresponds roughly to the class of ordinary objects. Examples include normal physical objects, geographical regions, locations of processes, and the complement of objects in the physical class.
- **Persistent Property**: An item property that persists over time, i.e. it does not change or is assumed not to change depending on the scenario under which the robot encounters the item. For example, Mass and Porosity.
- **Physical**: An entity that has a location in space-time. Note that locations are themselves understood to have a location in space-time. [59]
- **Process** [59]: The class of things that happen and have temporal parts or stages. Examples include extended events like a football match or a race, actions like pursuing and reading, and biological processes. The formal definition is: anything that occurs in time but is not an object.
- **Processing Capability**: A capability that allows the robot to process information (based on the CORA definition for robotProcessingPart [59]).
- **Processing Device**: A device that allows the robot to process information (based on the CORA definition for robotProcessingPart [59]).
- **Proposition [59]**: Propositions are abstract entities that express a complete thought or a set of such thoughts. As an example, the formula "(instance Yojo Cat)" expresses the proposition that the entity named Yojo is an element of the class Cat.
- **Relation** [59]: The class of relations. There are three kinds of relation: predicate, function, and list. Predicates and functions both denote sets of ordered n-tuples. The difference between these two classes is that predicates cover formula-forming operators, while functions cover term-forming operators. A list, on the other hand, is a particular ordered n-tuple.
- **Robot** [59]: An agentive device in a broad sense, purposed to act in the physical world in order to accomplish one or more tasks. In some cases, the actions of a robot might be subordinated to actions of other agents, such as software agents (bots) or humans. A robot is composed of suitable mechanical and electronic parts. Robots might form social groups, where they interact to achieve a common goal. A robot (or a group of robots) can form robotic systems together with special environments geared to facilitate their work.

- **Sensing Capability**: A capability that can play the role of a sensing part of a robot when it is connected to the robot (based on the CORA definition for robotSensingPart [59]).
- **Sensing Device**: A device that can play the role of a sensing part of a robot when it is connected to the robot (based on the CORA definition for robotSensingPart [59]).
- **Target Item**: The item that is the focus of the robot's attention.
- **Task** [96]: A restatement of the goal from the robot's perspective. If the goal is the expression of what the operator wants done, the task is how the robot interprets it (i.e., task planning). Tasks are accomplished via behaviours and actions.

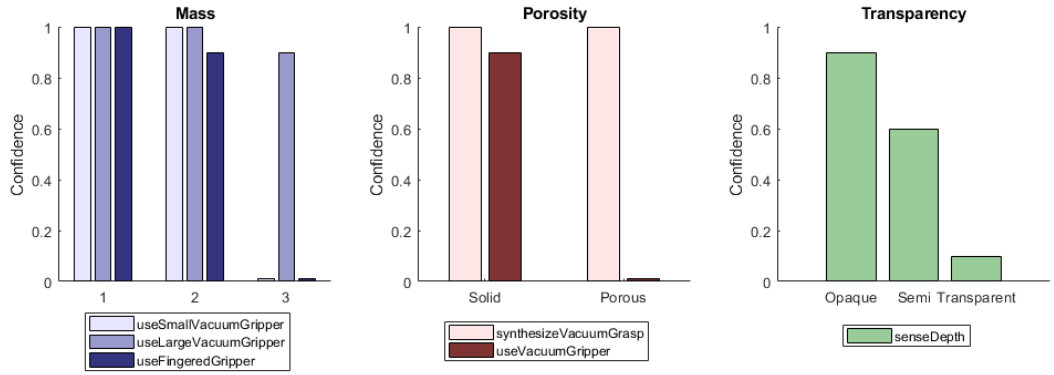
C.3. Example Ontology for Robot Kevin

This appendix provides an example of RSIR's ontology and reasoner for a simple robot. The example considers robot Kevin. Kevin was engineered to grasp items. To do this, Kevin has six Devices: a depth camera, an RGB camera, a synthesis module, a vacuum gripping mechanism, a fingered gripper, and a motion module. The vacuum gripping mechanism can use either a large or a small suction cup. The synthesis module can synthesize grasps for a vacuum gripper and for a fingered gripper. Through these devices, Kevin has access to the capabilities visualized in Figure C.2a. Additionally, Kevin contains knowledge of three Item Properties: Mass, Transparency, and Porosity. Kevin knows that some of its Capabilities have Constraints regarding these properties as shown in Figure C.2b. Additionally, Kevin knows that he is twice as likely to encounter a non-porous item than a porous item.

Because the engineers would like Kevin to grasp an item, the task is defined with a `vi:taskOutput` of `vi:hasGrasped`. The engineers define the cost function as $C = \sqrt{t}/c$, where C is the cost, t is the execution time, and c is the confidence score. They define it like this because they think both execution time and confidence are important.



(a) The inputs and outputs of the capabilities.



(b) The constraints on the capabilities.

Figure C.2: The capabilities Kevin has access to.

Kevin's Ontology

The ontology for Kevin is visualized in three figures. First, Figure C.3 shows the capabilities Kevin has, and

which inputs they require and outputs they provide. Second, Figure C.4 presents the constraints on the capabilities, and how they are connected to the devices. Finally, Figure C.5 shows the item properties, their default probability distributions, and the confidences per capability.

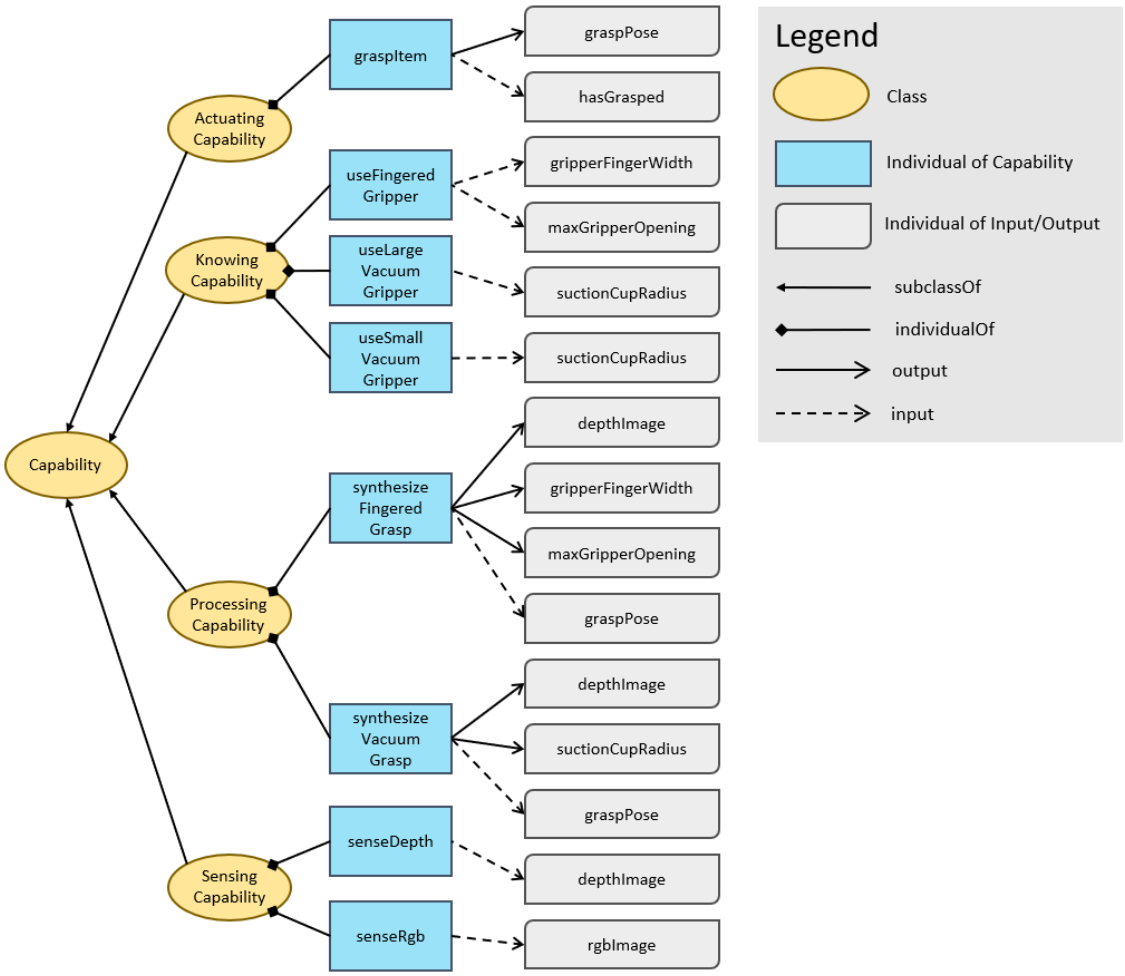


Figure C.3: The Capabilities and the Input/Outputs they provide.

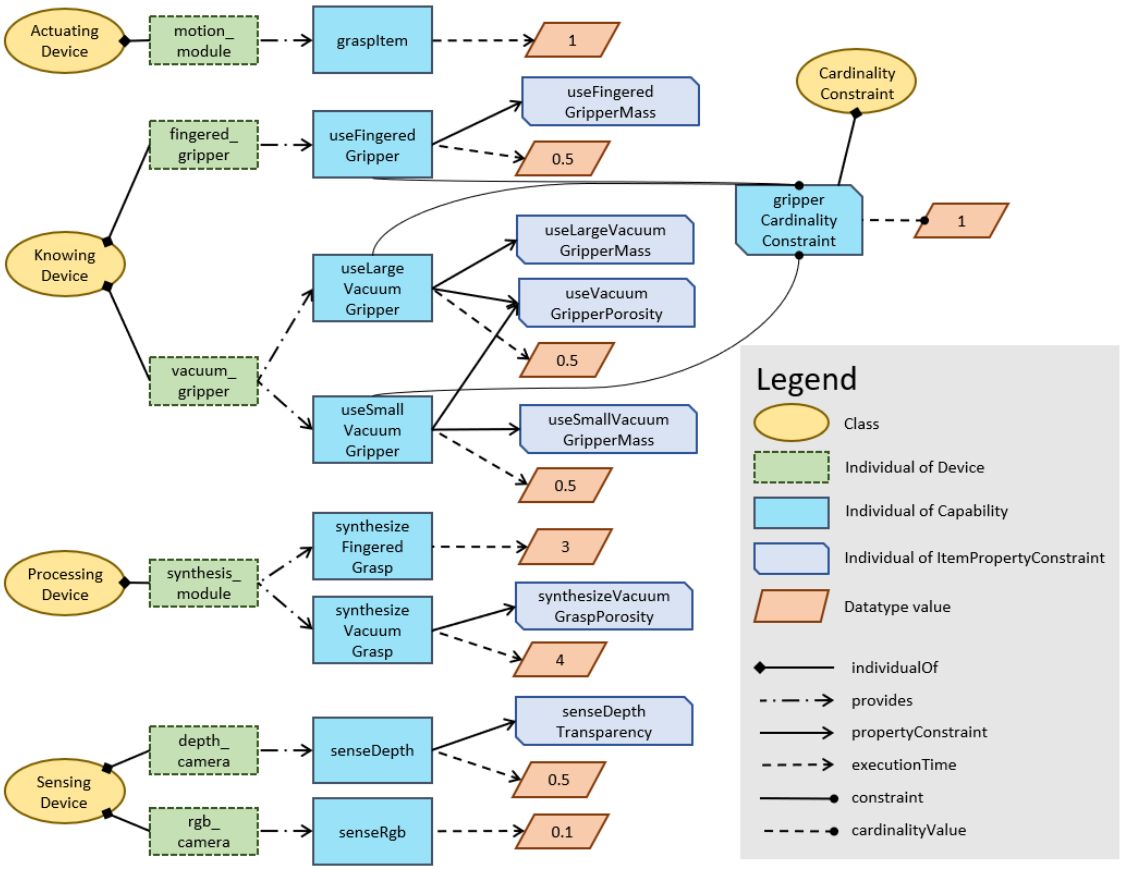


Figure C.4: The Devices, Capabilities, and their constraints and execution times.

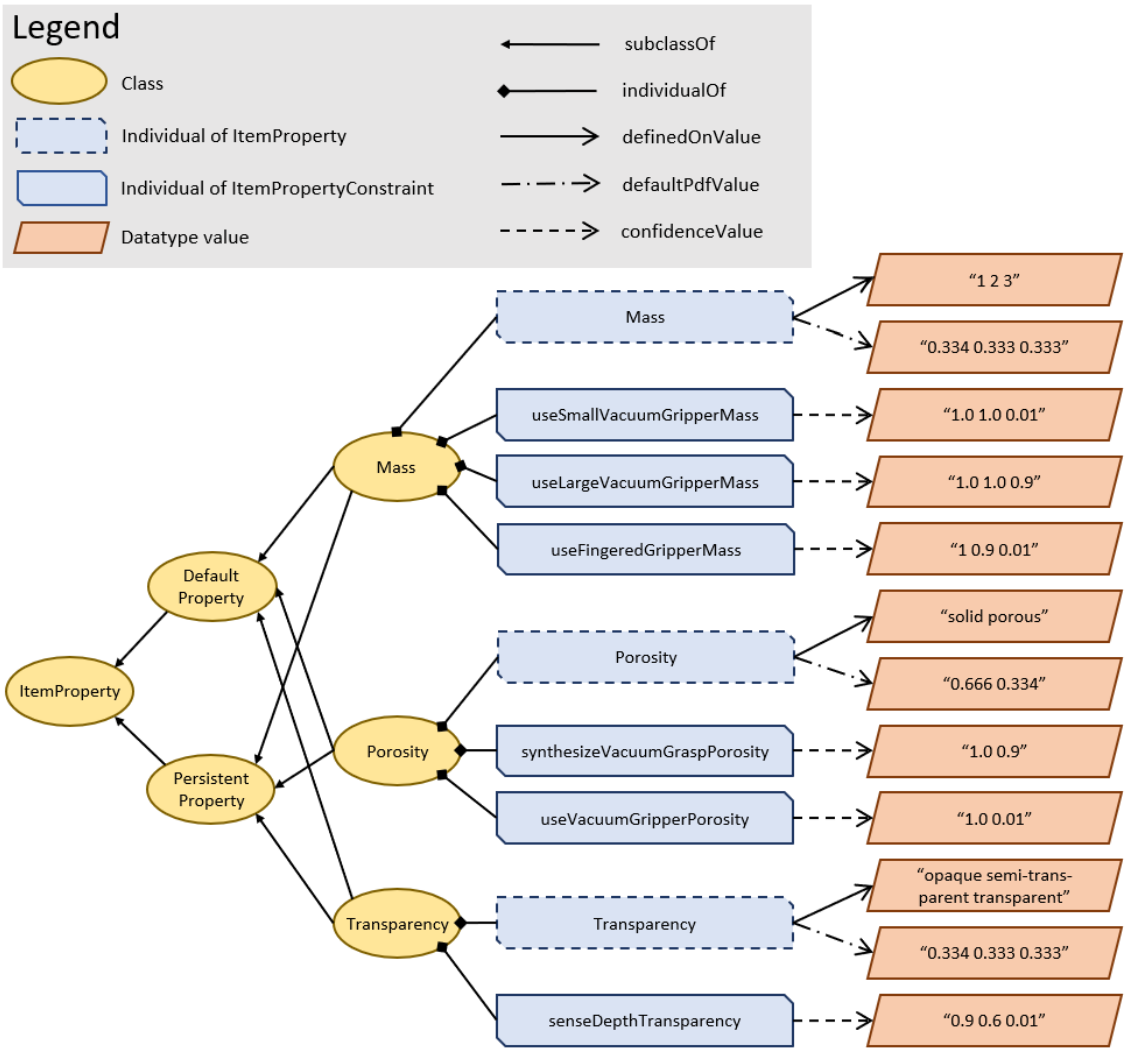


Figure C.5: The Item Properties and the confidences per Capability.

Task Execution

Kevin is powered on and finds the task that he must complete in his ontology. He starts reasoning about what capabilities he can use to complete this task. Kevin comes up with the three action plans shown in Figure C.6. Now Kevin must decide which of these plans is the best one. Kevin was not given any information about the item he must grasp, so he assigns the default properties to the item. This includes the knowledge that non-porous items are more common than porous items. Now Kevin can calculate the cost for each Action Plan using the cost function and each plan's confidence score and execution time.

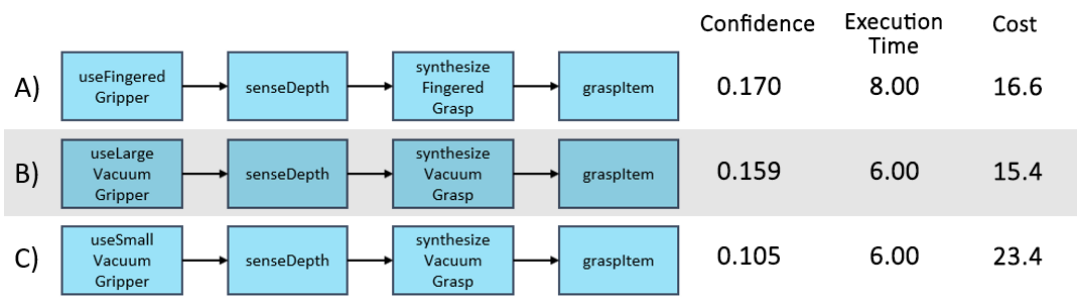


Figure C.6: The action plans Kevin creates to complete the task, with the confidences, execution times, and costs as computed from the default knowledge of the item.

Kevin concludes that Action Plan B is the best, because Action Plan B has the lowest cost. He is now ready to execute the plan. He starts by readying his large vacuum gripper. Then he takes a depth image, and synthesizes a grasp pose. Finally, he uses the motion module to move to the grasp pose and activate the gripper. But the motion module gives an error that says that the grasp has failed!

Now Kevin must reason about what went wrong. He enters into his Bayesian network what he knows about the item property distributions and the constraints on his capabilities. The network then looks like Figure C.7, and the probabilities are defined as shown in Tables C.1 through C.3. Note that the network does not include the `graspItem` capability. This is because Kevin believes that this capability can never fail, because it has no constraints.



Figure C.7: The Bayesian network for Robot Kevin after grasping with the vacuum gripper has failed.

Table C.1: The probability distributions Kevin enters into the Bayesian network.

Probability	**Value**
P(Transparency = Opaque)	0.334
P(Transparency = Semi-transparent)	0.333
P(Transparency = Transparent)	0.333
P(Mass = 1)	0.334
P(Mass = 2)	0.333
P(Mass = 3)	0.333
P(Porosity = Solid)	0.666
P(Porosity = Porous)	0.334
P(Outcome = 1)	0
P(Outcome = 0)	1

Table C.2: The conditional probabilities Kevin enters into the Bayesian network, part 1.
X ∈ (senseDepth, useLargeVacuumGripper,synthesizeVacuumGrasp)

Outcome X due to Transparency	Outcome X due to Mass	Outcome X due to Porosity	P(Outcome X=1)	P(Outcome X=0)
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

Outcome senseDepth	Outcome use-LargeVacuumGripper	Outcome synthesize-VacuumGrasp	P(Outcome=1)	P(Outcome=0)
0	0	0	0	1
0	0	1	0	1
0	1	0	0	1
0	1	1	0	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	0

Table C.3: The conditional probabilities Kevin enters into the Bayesian network, part 2.
Note that all numbers that appear in this table are obtained from the knowledge base.

		senseDepth		useLarge-VacuumGripper		synthesize-VacuumGrasp	
		P(1)	P(0)	P(1)	P(0)	P(1)	P(0)
Trans-parency	Opaque	0.9	0.1	1	0	1	0
	Semi-transparent	0.6	0.4	1	0	1	0
	Transparent	0.01	0.99	1	0	1	0
Mass	1	1	0	1	0	1	0
	2	1	0	1	0	1	0
	3	1	0	0.9	0.1	1	0
Poros-ity	Solid	1	0	1	0	1	0
	Porous	1	0	0.01	0.99	0.9	0.1

After running the inference, Kevin checks the results. The likelihoods of each capability having succeeded is

shown in Table C.4. Kevin can see that the most likely cause of the failed grasp is the `senseDepth` capability. He can also see that if the large vacuum gripper failed, it is more likely that it failed because of the item's porosity than because of the item's mass. Kevin does not use this information, but it does help the engineers understand what is going on in Kevin's mind.

Table C.4: The likelihood of each capability having succeeded, per item property and for all properties combined. For example, the likelihood that the vacuum gripper succeeded despite the item's mass is 0.95.

	Transparency	Mass	Porosity	All properties
senseDepth	0.29	1.00	1.00	0.29
useLargeVacuumGripper	1.00	0.95	0.51	0.48
synthesizeVacuumGrasp	1.00	1.00	0.92	0.93

The information that Kevin does use from the results of the inference is the updated item properties, visualized in Figure C.8. For example, Kevin sees that the item is more likely to be porous than he had thought before. Kevin takes another look at the three Action Plans he created. He recomputes the confidence scores based on his new knowledge of the item. They have the costs as shown in Figure C.9. And behold, now Kevin sees that Action Plan A has the lowest cost. Thus the next time Kevin attempts to grasp the item, he will use the fingered gripper instead.

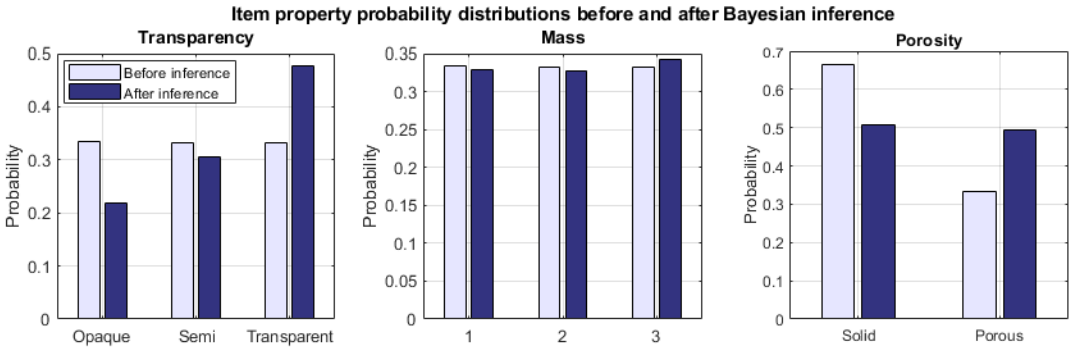


Figure C.8: The beliefs Kevin has about the item properties before and after Bayesian inference.

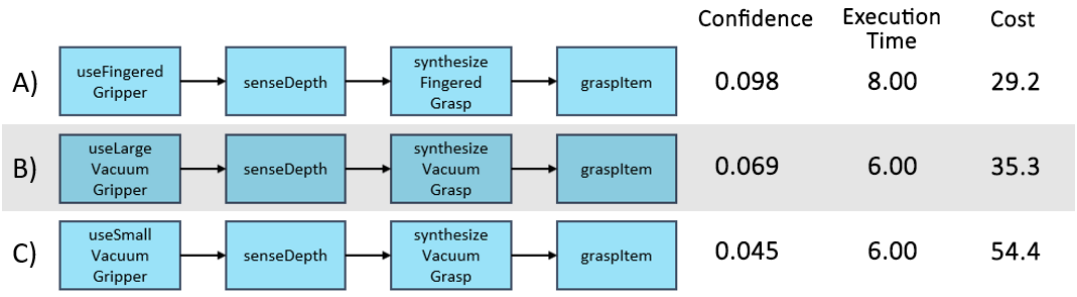


Figure C.9: The action plans Kevin creates to complete the task, with the confidences, execution times, and costs as computed from the inferred knowledge of the item.

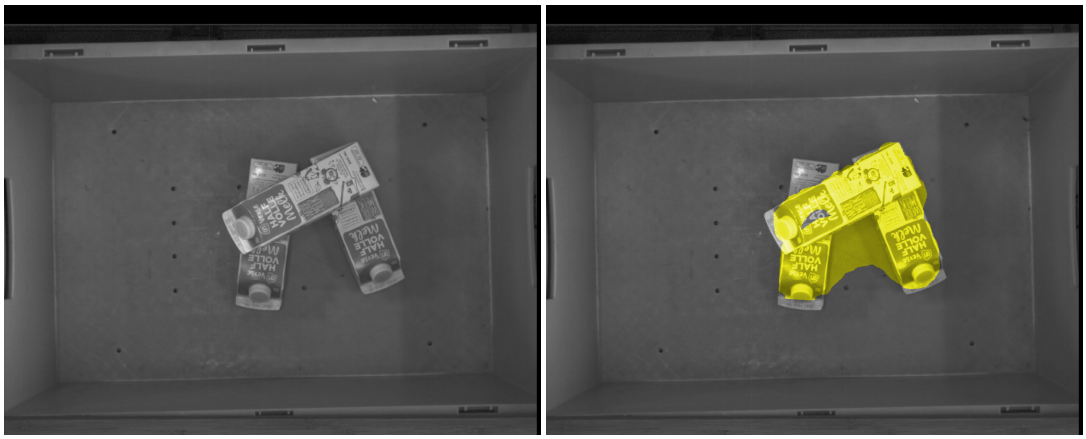
D

Details on the Grasp Synthesis Algorithm

D.1. Implementation of the Algorithm

This section explains RSIR's grasp synthesis algorithm as implemented in Matlab. Because Baumgartl and Henrich do not provide much detail on the implementation of their algorithm, their implementation might differ from the implementation in RSIR, although the algorithms follow the same basic idea. Because Matlab's Hough transform does not output the lines and pairs of lines, some additional coding is required. The steps in RSIR's grasp synthesis algorithm is as follows:

1. The algorithm receives a ROS service call containing the RGB image, segmentation, and gripper parameters. The latter consists of the minimum opening width, the maximum opening width, the maximum grasping height, and the finger width (all defined in metres). Figure D.2 illustrates these parameters and Figure D.1 shows an example of an RGB image and segmentation.



(a) The RGB image. (b) The segmentation visualized onto the RGB image.

Figure D.1: RGB and segmentation.

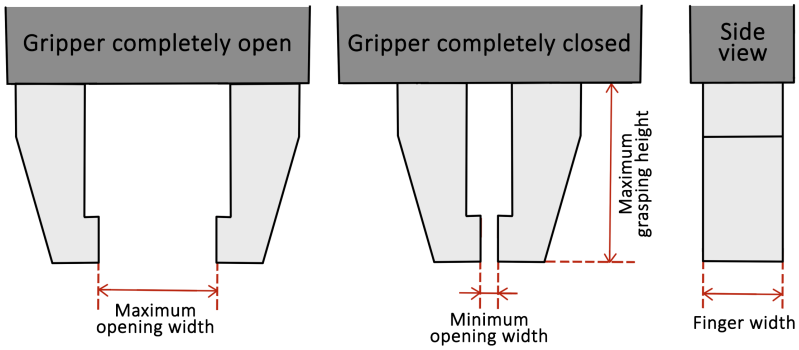
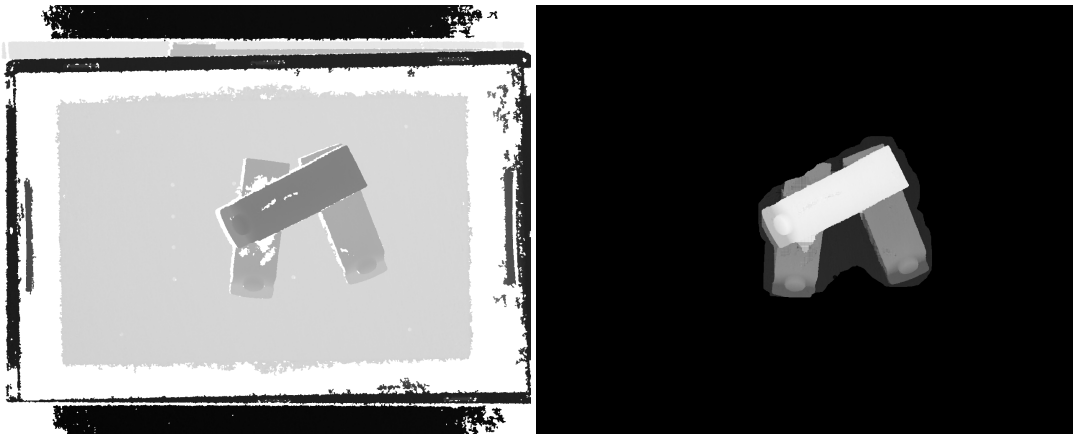


Figure D.2: The gripper parameters used by RSIR's grasp synthesis algorithm.

2. The corresponding point cloud is loaded from a file. This point cloud is not passed via the ROS service call because Matlab could not correctly read the point cloud from that call. The point cloud is projected onto the horizontal plane. The resulting depth image is shown in Figure D.3a.

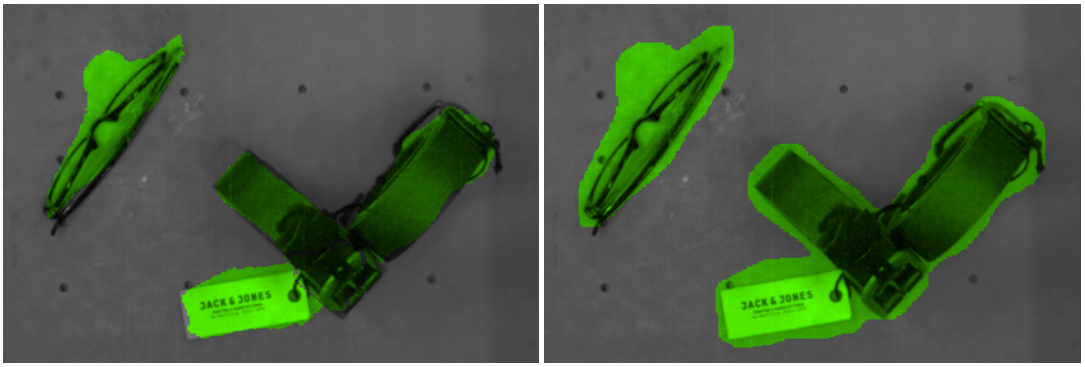


(a) The raw depth image. Pure white areas indicate NaN values.

(b) The depth image after filtering.

Figure D.3: The depth image before and after filtering.

3. The segmentation is diluted as shown in Figure D.4. This prevents important information from being lost, as the segmentation might miss some areas of the item or be just on the edge of it, which would mean the edges of the item are not within the segmentation.



(a) The undiluted segmentation mask, indicated in green.

(b) The segmentation mask diluted by 20 pixels, indicated in green.

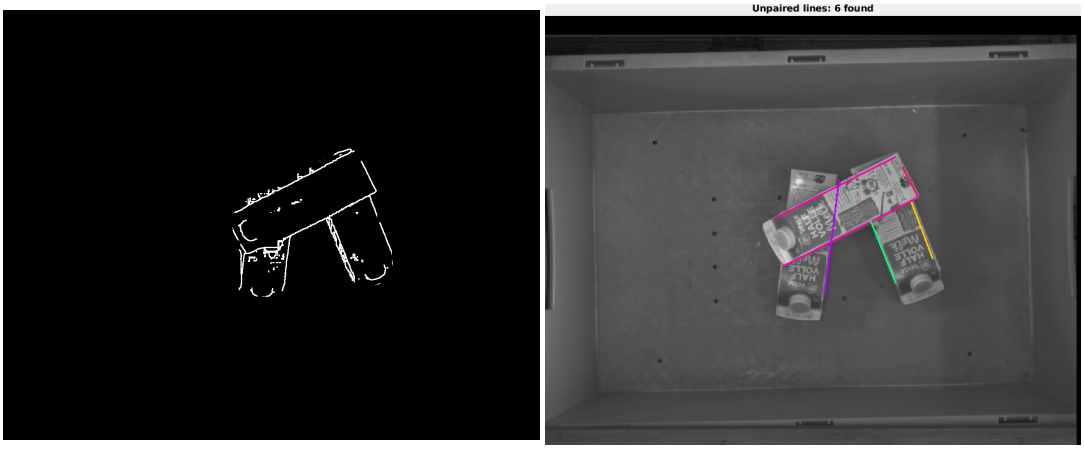
Figure D.4: En example of a segmentation mask before and after it is diluted.

4. The depth image is prepared for edge detection. First, the image is filtered using Matlab's `fillmissing`

function to remove any NaNs. The image is then inverted and rescaled to values between 0 and 1, with 1 being closest to the camera. Areas of the image that are not within the segmentation mask are replaced with the smallest value present in the image. Finally, the image is rescaled again to have values between 0 and 1. The rescaling creates more contrast that helps in detecting edges. Figure D.3b shows the resulting depth image. Alternatively, the RGB image can be used instead of the depth image. In this case, the RGB image is converted to a grey-scale image and it is not rescaled.

5. The gripper's minimum opening width, maximum opening width, and length are converted from metres to pixels. The pixels/metre resolution is determined from the depth image and the RGB image.

6. The edited depth image is now used for detecting edges, using Matlab's `edge` function with the Sobel method. The resulting image is called the binary gradient mask. It is diluted so that the edges are more apparent. An example for detected edges is shown in Figure D.5a.



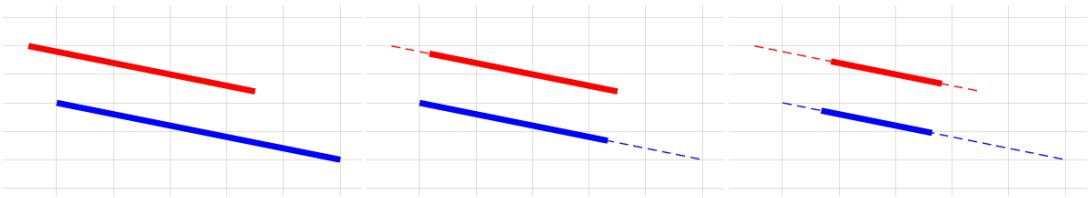
(a) The detected edges.

(b) The detected lines.

Figure D.5: The results of the edge and line detection algorithms.

7. The Hough transform is used on the binary gradient mask using the Matlab function `hough`, resulting in a list of lines. The lines are visualized in Figure D.6. Pairs of lines are created by combining lines with similar angles. The required level of similarity for lines to be paired is defined by a parameter.

8. For each pair of lines, the lines are shortened such that they overlap when projected onto each other and such that the length of the line is at most the finger width. Figure D.6 illustrates this. This is a necessary step because lines that have little or no overlap are unlikely to result in good grasps. Additionally, the distance between the lines is calculated.



(a) The original pair of lines.

(b) The parts of the lines that overlap.

(c) The lines after applying the finger width.

Figure D.6: Illustration of the overlap between pairs of lines. The dotted lines show the size of the original lines.

9. At this point, infeasible line pairs are filtered out. Pairs are infeasible when the distance between the lines is larger than the maximum opening width or smaller than the minimum opening width, or if the length of the lines is smaller than the finger width. Finally, pairs that appear more than once are removed until they only appear once. This filtering is useful for improving runtime, especially since the next step is relatively time-consuming.

10. The line pairs are ranked based on the defined metric. The current version of the algorithm only considers enclosed depth. This is a measure for how much of the volume between the gripper fingers is occupied by the item. It is a number between 0 and 1, and could thus act as a confidence score. The enclosed depth is computed as:

$$\frac{\sum_x \sum_y m(x,y) \cdot max(h_g - d(x,y), 0)}{\sum_x \sum_y m(x,y) \cdot max(h_g - h_i, 0)}$$

where m is the grasp mask, d is the depth image after NaNs are removed, h_g is the grasping height, h_i is the item height, and x and y are the image coordinates. Note that a small value of $d(x,y)$ means that (x,y) is close to the camera. The grasping mask is defined as $m(x,y) = 1$ if (x,y) is between the lines in a line pair and $m(x,y) = 0$ if (x,y) is not between the lines. The item height is defined as $h_i = min(m(x,y)d(x,y))$. Finally, the grasping height is the height at which the gripper fingers are. It is computed through $h_g = min(min(d(x_n, y_n)), h_i - h_{g,max})$. This equation evaluates the depth at n points on the lines to determine the height of the lines. These points are selected slightly further apart than the detected lines are, to prevent the fingers from colliding with the item. Additionally, the gripper is lifted such that the "hand" of the gripper does not collide with the item. This is where the maximum grasping height $h_{g,max}$ comes in. Figure D.7 shows some examples of the enclosed depth.

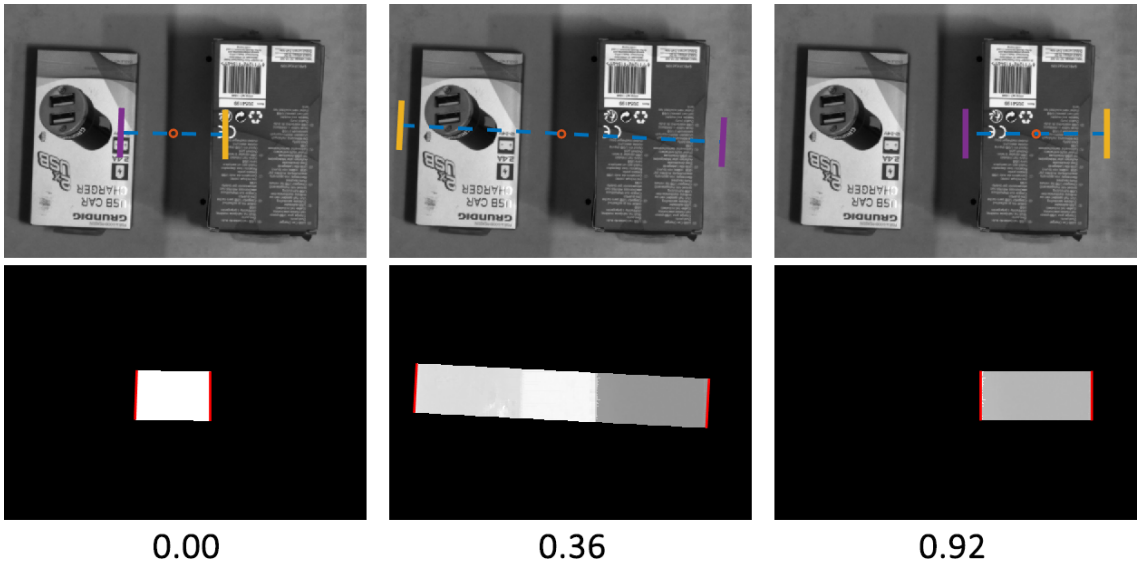


Figure D.7: Three examples of values of the enclosed depth metric. From top to bottom: the enlarged grasp, the grasp mask with the depth image in it, and the value of the metric. The lines in the top row are the lines on which the points (x_n, y_n) are taken.

11. Finally, the line pair with the highest value of the metric is selected as the grasp. Again, the lines are taken slightly further apart than the detected lines are. How much further apart they are taken is defined by a parameter. The pose's image coordinates are then converted to coordinates in the robot frame through known transformation matrices. The ROS service call is answered with a string containing the grasp pose (as a quaternion) and the gripper opening. Figure D.8 illustrates the resulting grasp pose.

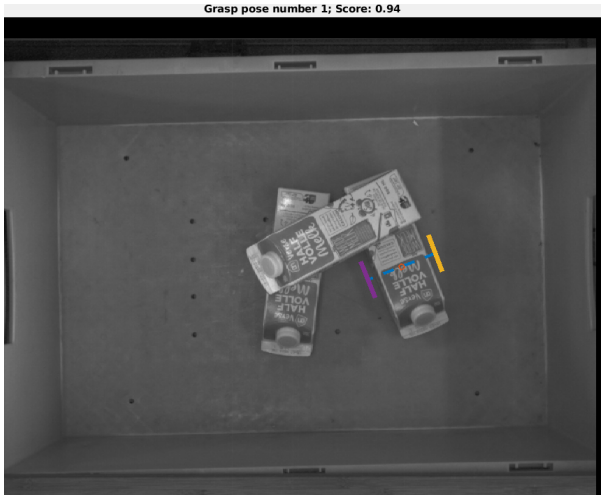
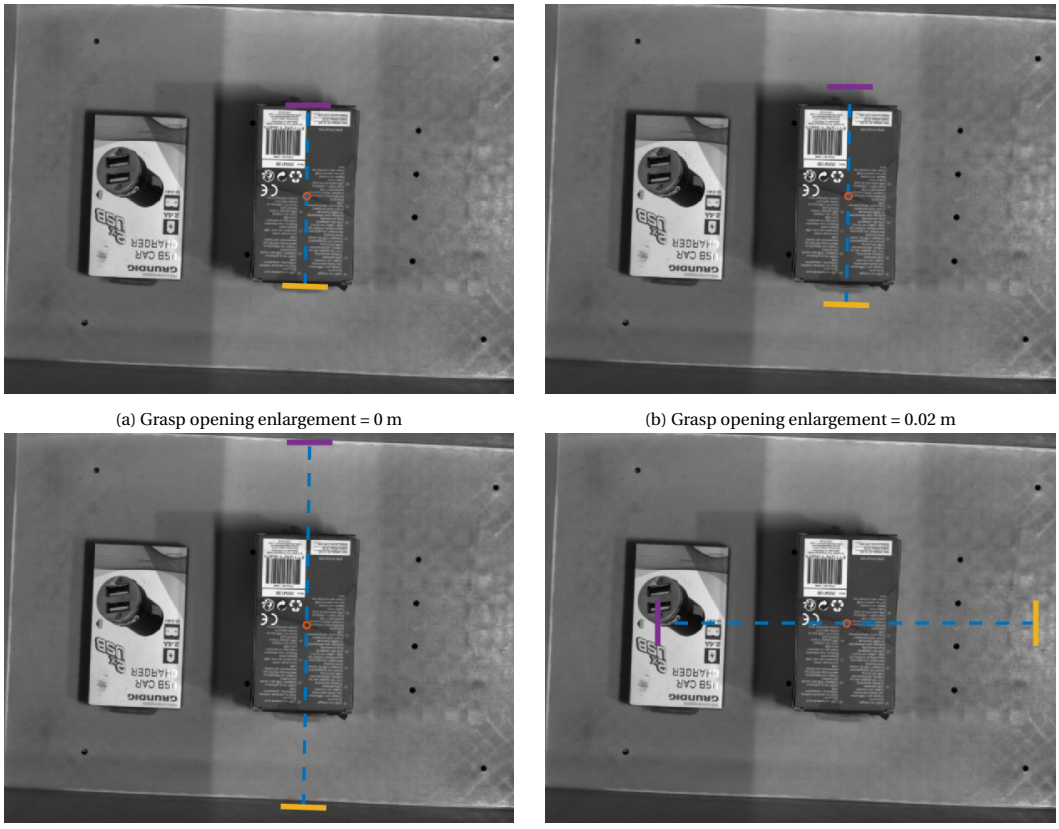


Figure D.8: The resulting grasp pose.

D.2. Grasp Synthesis Algorithm Parameters

This appendix investigates the influence of the synthesis algorithm's parameter on the resulting grasps and the runtime. The parameters are varied one for the carrier and default parameters shown in Figure D.9. Every time one parameter is investigated, the other parameters are kept at their default values. The runtimes resulting from this carrier are visualized in Figure 8.2.

The **grasp opening enlargement** is the extra distance by which the gripper is opened with respect to the line pair. This is illustrated in Figure D.10. A larger grasp enlargement could provide robustness to uncertainties in depth information and motion control. However, a large enlargement can also result in a different grasp than was intended. This is visible in Figure D.10d: because the enlargement causes the previously best grasp to collide with the carrier, the approach considers another grasp that is arguably less good. Grasp opening enlargement has no influence on the runtime.



(a) Grasp opening enlargement = 0 m

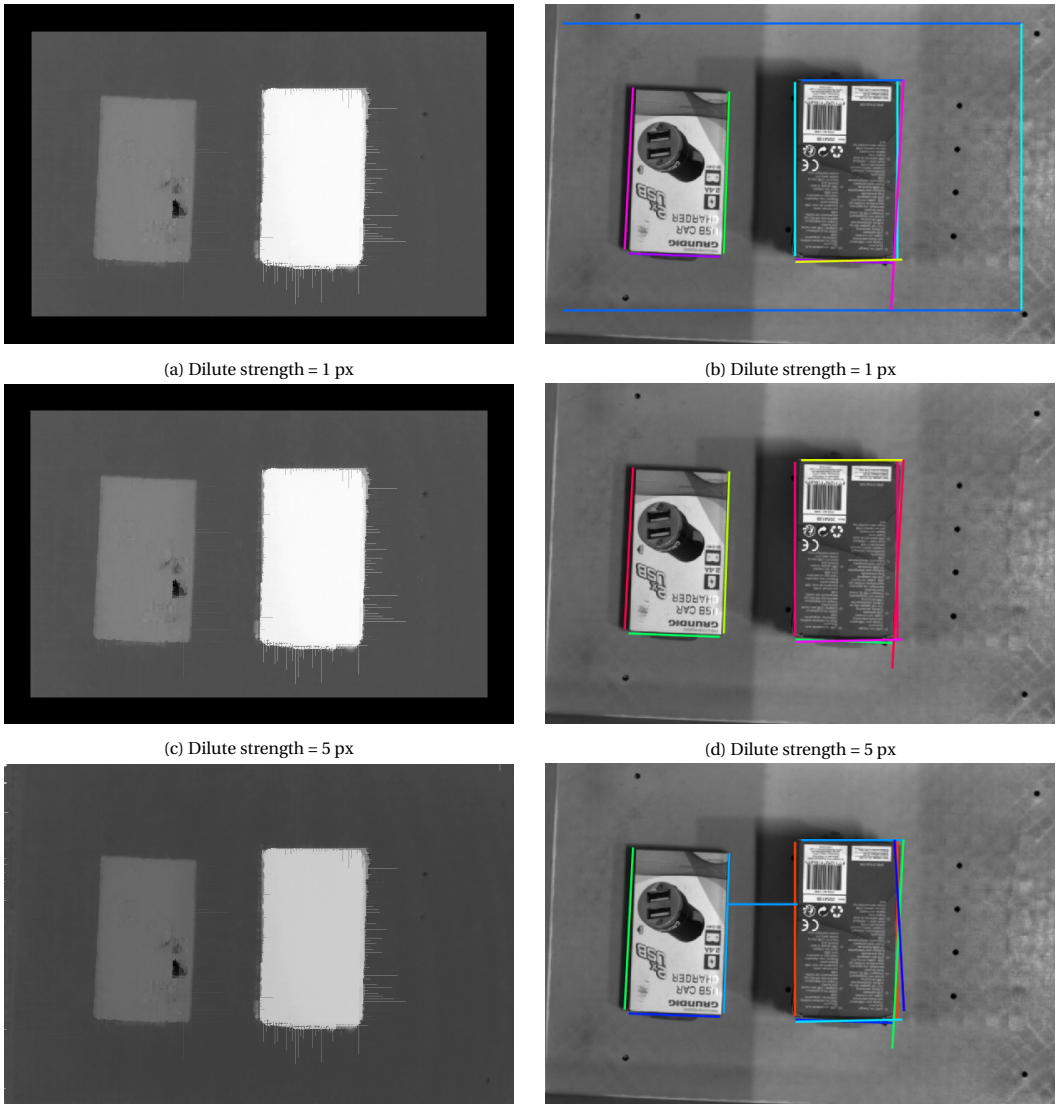
(b) Grasp opening enlargement = 0.02 m

(c) Grasp opening enlargement = 0.1 m

(d) Grasp opening enlargement = 0.15 m

Figure D.10: Highest-scoring grasps for different values of the grasp opening enlargement.

The **dilute strength** enlarges the segmentation mask as shown in Figure D.11. The enlarged mask includes some parts of the image that are not part of the target item. If a segmentation of an item is provided, this way, the algorithm sees both the item and the background and can thus detect the edges of the item. Additionally, the dilution prevents the algorithm from detecting lines on the edge of the segmentation mask (as shown in Figure D.11, there is a lower bound for which this works). The dilution is also useful in cases where the segmentation misses parts of the item. Interestingly, Figure D.11 shows that the line detections are slightly different for different values of the dilute strength. This is likely due to the segmented image being larger, and thus allowing for more line detections. The dilute strength influences the runtime, but this is likely due to the (irrelevant) additional lines being detected.



(a) Dilute strength = 1 px

(b) Dilute strength = 1 px

(c) Dilute strength = 5 px

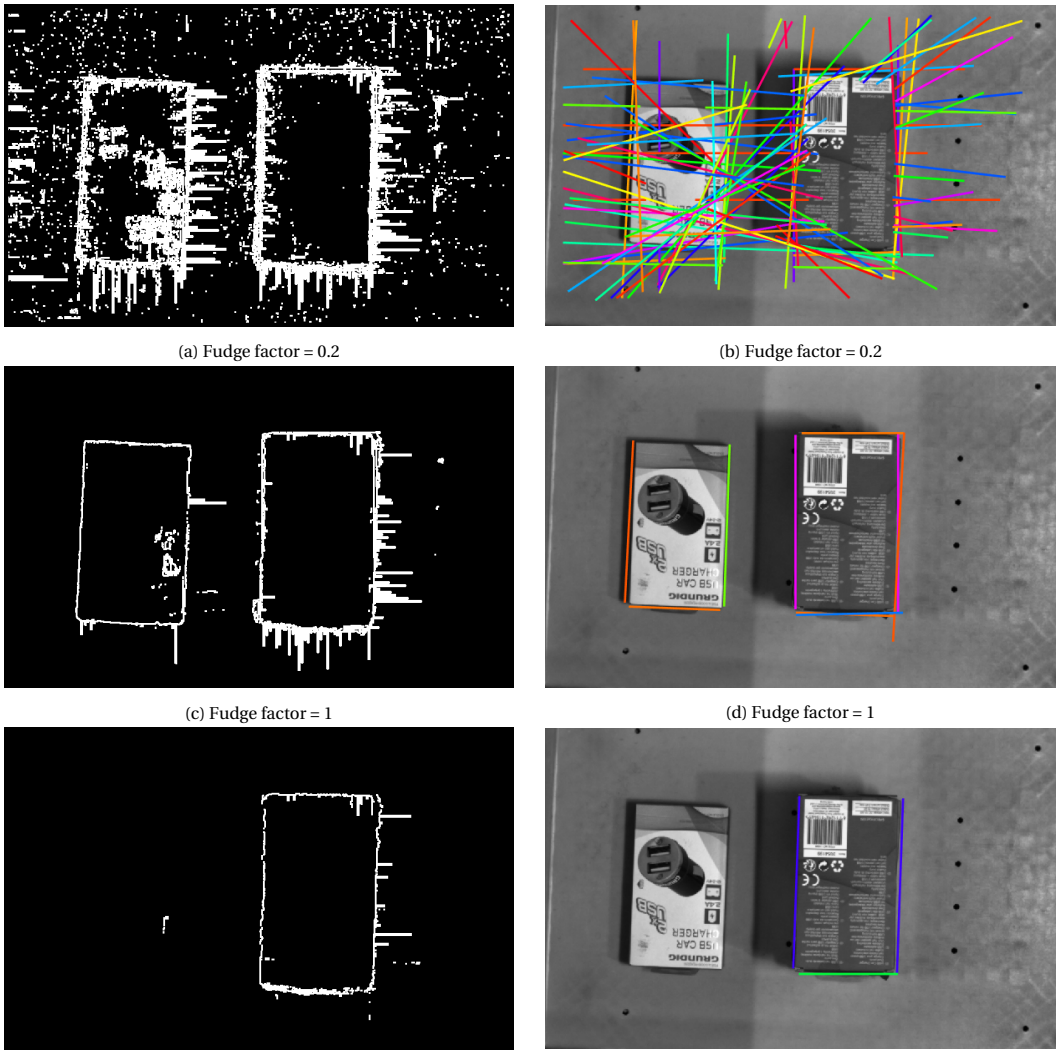
(d) Dilute strength = 5 px

(e) Dilute strength = 100 px

(f) Dilute strength = 100 px

Figure D.11: The filtered and segmented depth image (left) and detected lines (right) for different values of the dilute strength.

The **fudge factor** defines a threshold for detecting edges. Figure D.12 illustrates this. A lower threshold results in more edges and thus more line detections. The fudge factor should be selected properly: too high a threshold filters out important edges and thus important grasps, but too low a threshold results in irrelevant edges that slow down the approach significantly.



(a) Fudge factor = 0.2

(b) Fudge factor = 0.2

(c) Fudge factor = 1

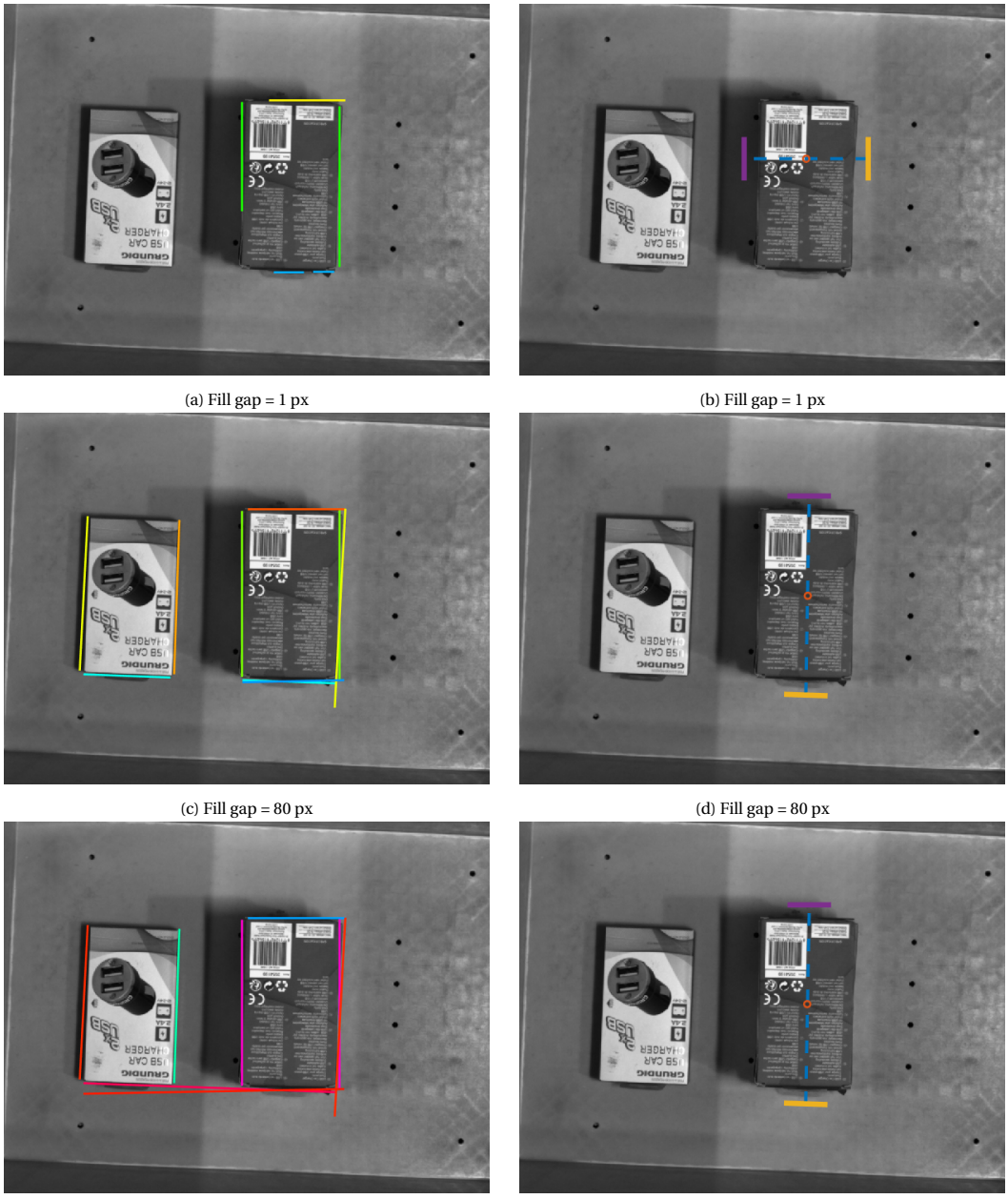
(d) Fudge factor = 1

(e) Fudge factor = 6

(f) Fudge factor = 6

Figure D.12: Detected edges (left) and detected lines (right) for different values of the fudge factor.

The results for different values of the **fill gap** are shown in Figure D.13. The fill gap is the maximum pixel distance between two line segments for the lines to be merged into one. As expected, the figure shows that larger values of the fill gap result in longer lines. The fill gap should be tuned properly to detect the full edges of items, without merging edges of multiple items into one (as visible in Figure D.13e). The fill gap influences the runtime insofar that small values of the fill ap result in fewer line detections and thus a shorter runtime.



(a) Fill gap = 1 px

(b) Fill gap = 1 px

(c) Fill gap = 80 px

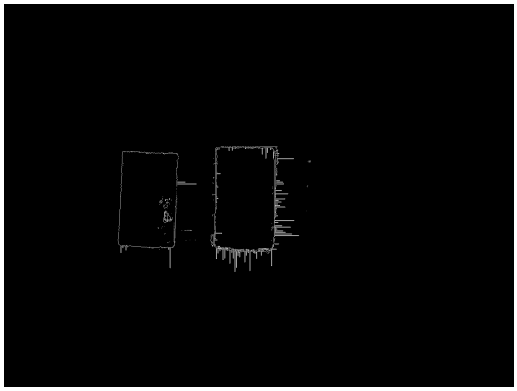
(d) Fill gap = 80 px

(e) Fill gap = 200 px

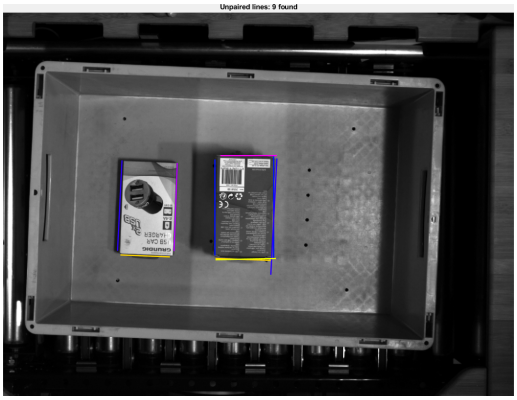
(f) Fill gap = 200 px

Figure D.13: Detected lines (left) and highest-scoring grasps (right) for different values of the fill gap.

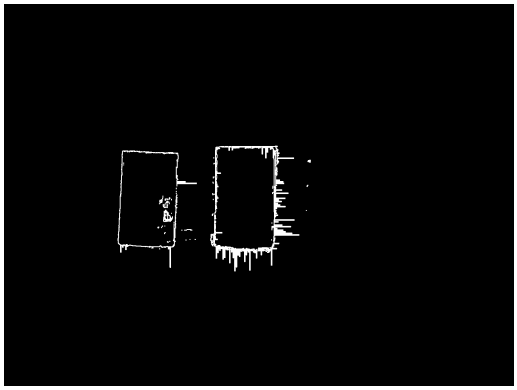
The **dilute strength for edges** enlarges the edge detections. Figure D.14 illustrates this. There are no differences in detected edges for small values of the strength, but larger values cause the approach to detect the edge of the segmentation, resulting in invalid edge detections. The dilute strength must be larger than the dilute strength for edges to prevent this. The dilute strength for edges does not significantly influence the runtime.



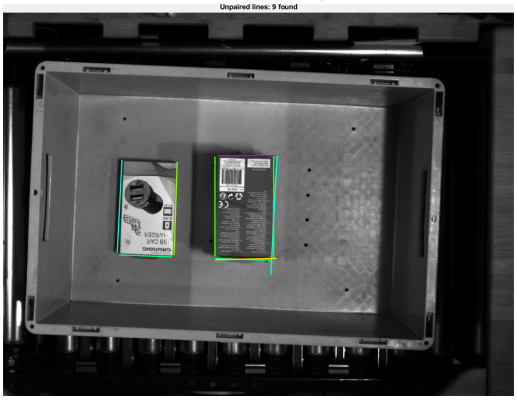
(a) Dilute strength for edges = 1 px



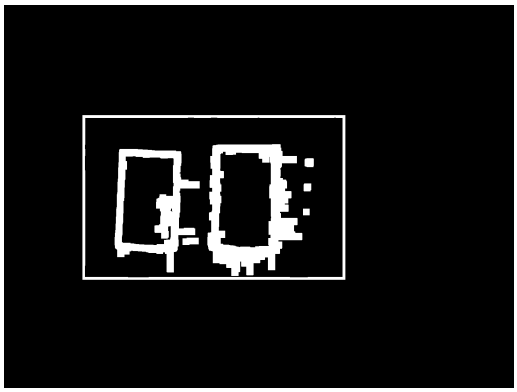
(b) Dilute strength for edges = 1 px



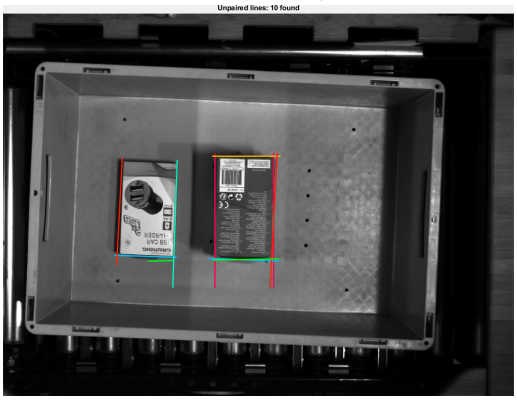
(c) Dilute strength for edges = 3 px



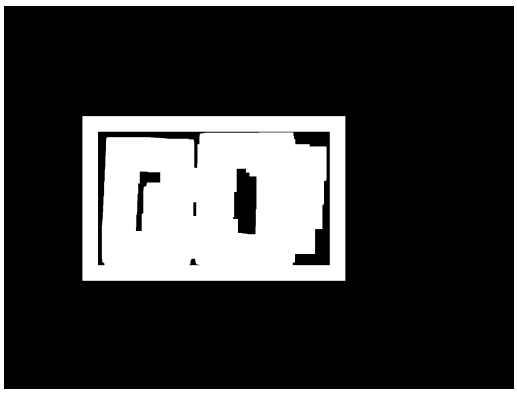
(d) Dilute strength for edges = 3 px



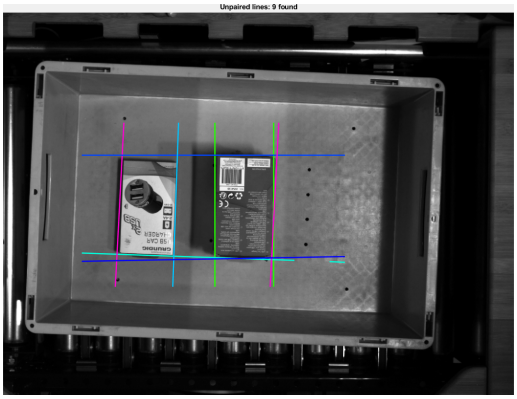
(e) Dilute strength for edges = 20 px



(f) Dilute strength for edges = 20 px



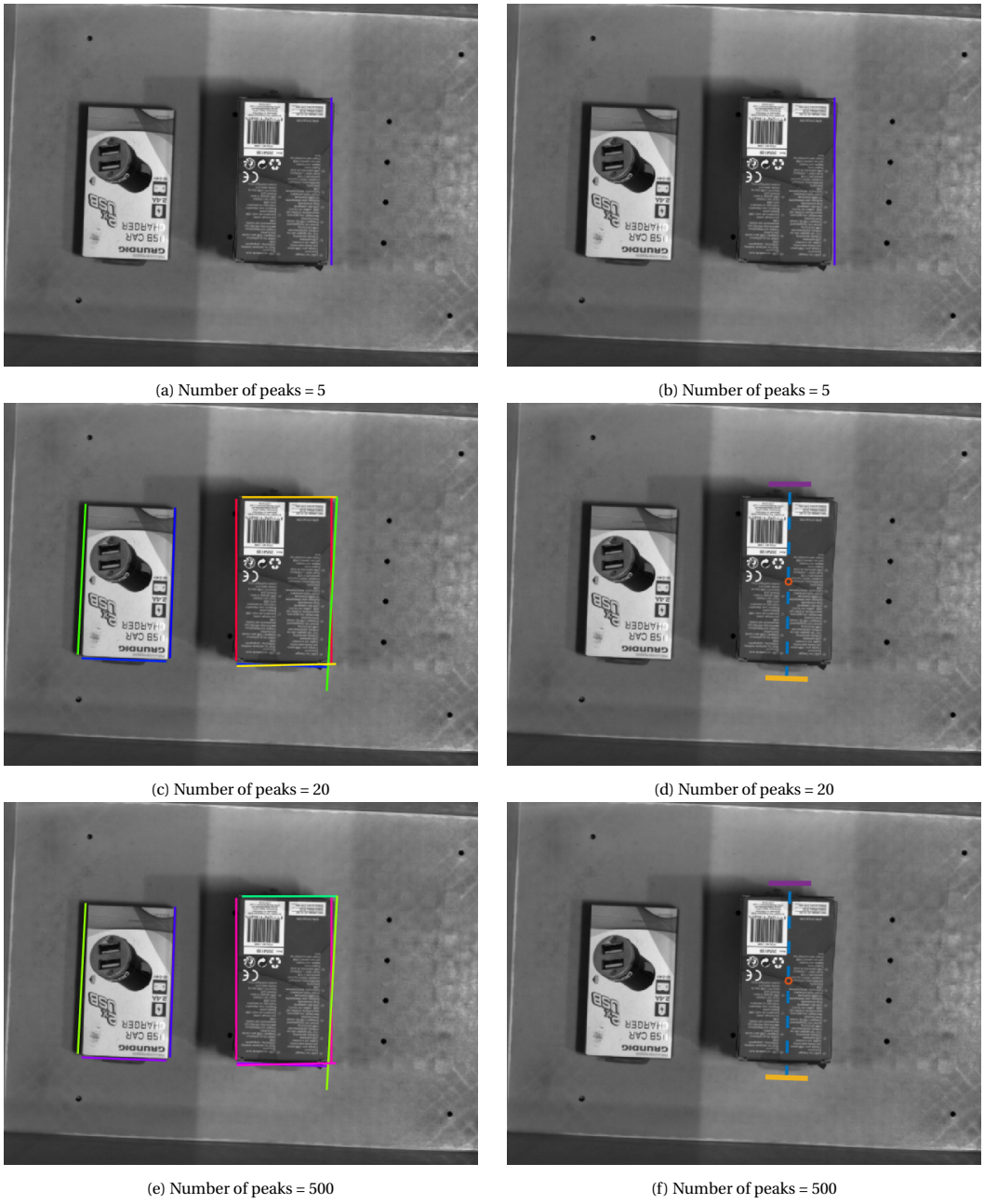
(g) Dilute strength for edges = 100 px



(h) Dilute strength for edges = 100 px

Figure D.14: Detected edges (left) and detected lines (right) for different values of the dilute strength for edges.

The **number of peaks** puts an upper bound on the number of detected lines, as illustrated in Figure D.15. As expected, if lines are discarded, the runtime is decreased. If no lines are discarded, the number of peaks does not influence the runtime. Note that the value of the number of peaks does not equal the number of detected lines.



(a) Number of peaks = 5

(b) Number of peaks = 5

(c) Number of peaks = 20

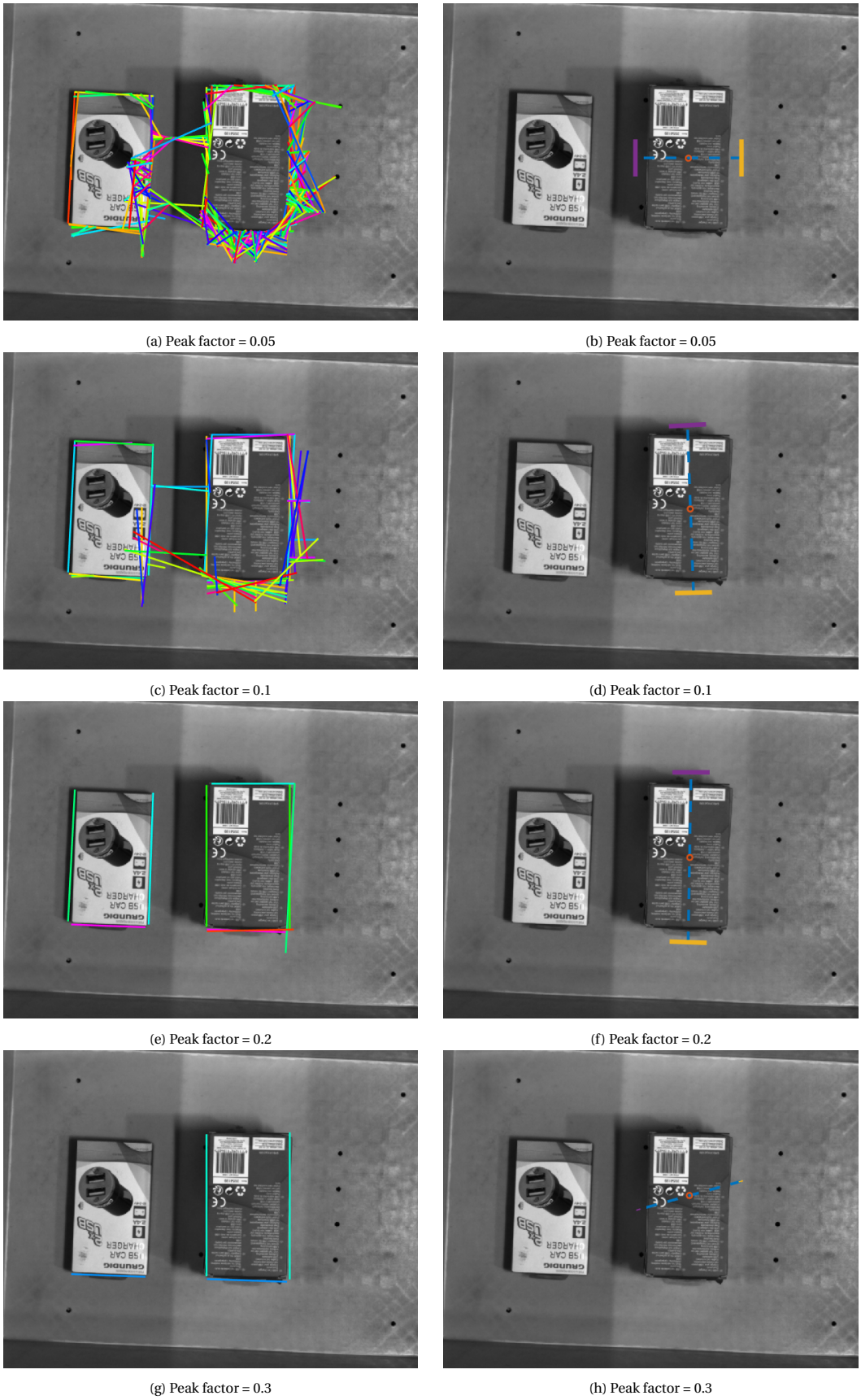
(d) Number of peaks = 20

(e) Number of peaks = 500

(f) Number of peaks = 500

Figure D.15: Detected edges (left) and highest-scoring grasps (right) for different values of the number of peaks.

The **peak factor** influences the number of detected lines, as it defines the minimum value for the algorithm to detect a Hough peak. It is one of the parameters that influence the runtime the most. Lower values result in more line detections, and a longer runtime. On the other hand, higher values decrease runtime, but might overlook important edges. This is illustrated in Figure D.16. The peak factor could be tuned along with the number of peaks, as the latter can prevent the algorithm from detecting an unreasonable number of lines.



(a) Peak factor = 0.05

(b) Peak factor = 0.05

(c) Peak factor = 0.1

(d) Peak factor = 0.1

(e) Peak factor = 0.2

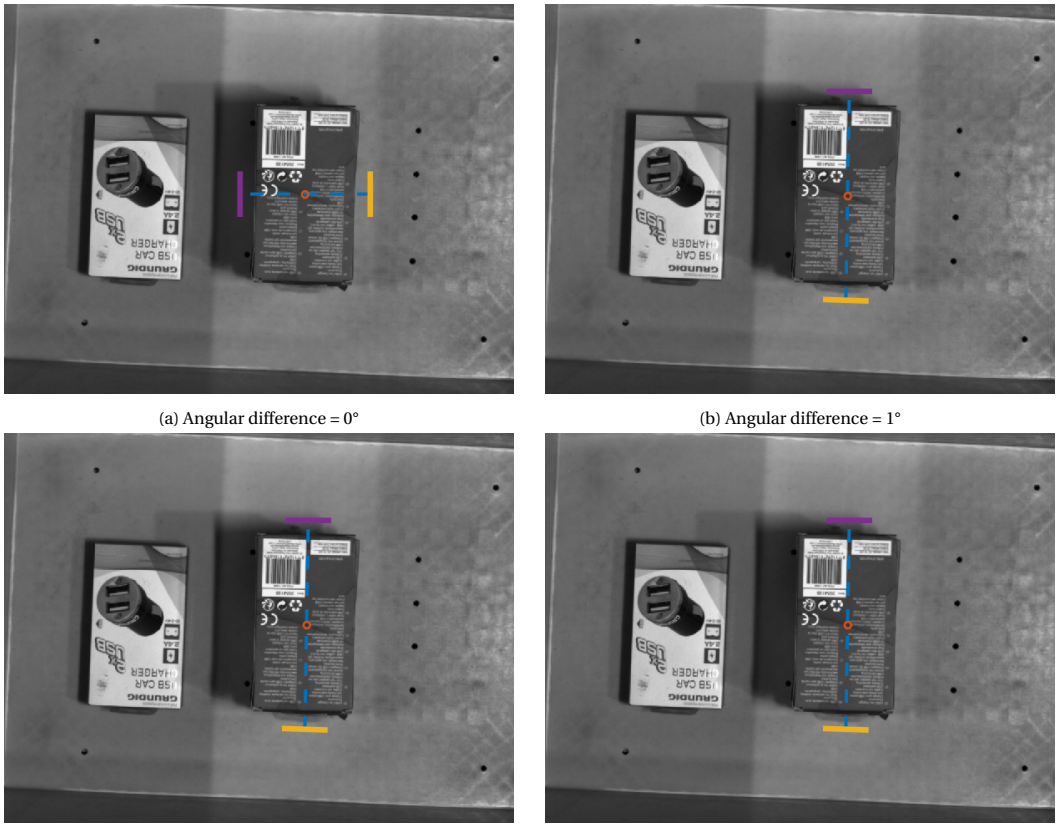
(f) Peak factor = 0.2

(g) Peak factor = 0.3

(h) Peak factor = 0.3

Figure D.16: Detected edges (left) and highest-scoring grasps (right) for different values of the peak factor.

The **angular difference** determines how similar lines must be to be considered a pair. It is defined in angles. For example, an angular difference of 2 implies that lines with at most 2 degrees difference between them are considered a pair. This is especially relevant for items with non-parallel edges, such as tubes of toothpaste. Figure D.18 shows that the resulting grasp does not change for angular differences above a certain value. However, the larger the angular difference, the longer the runtime.



(a) Angular difference = 0°

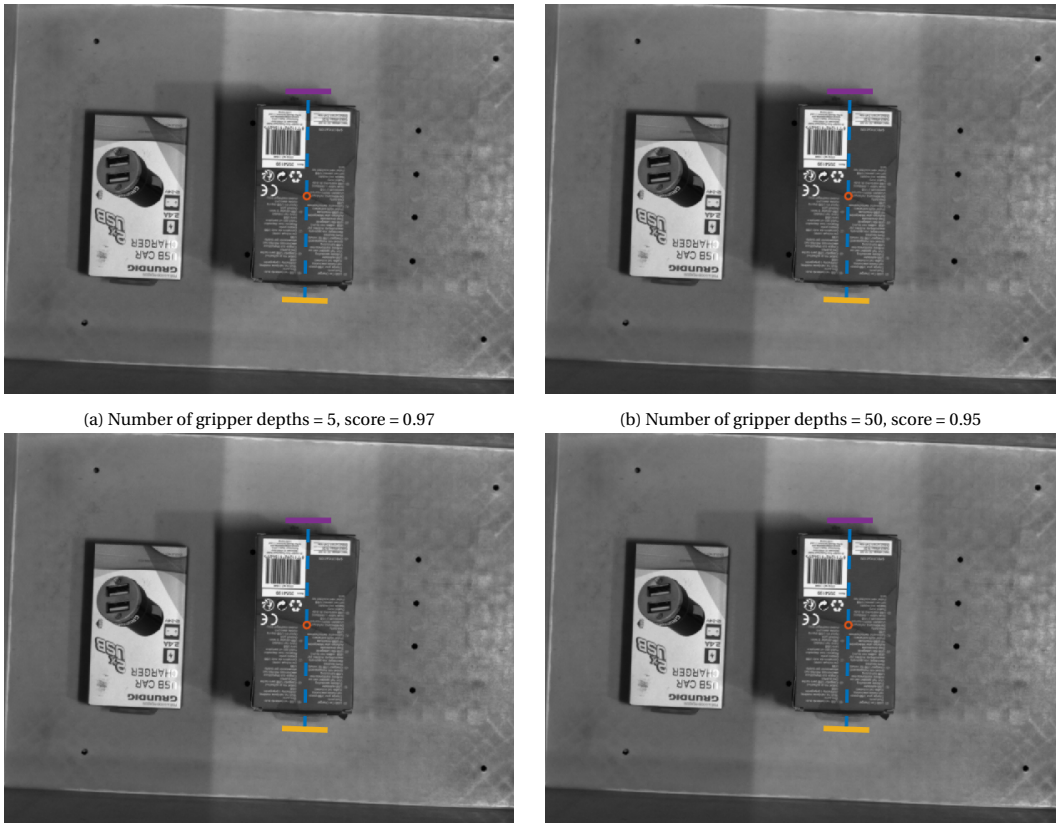
(b) Angular difference = 1°

(c) Angular difference = 5°

(d) Angular difference = 50°

Figure D.17: Highest-scoring grasps for different values of the angular difference.

The **number of gripper depths** is the number of points the algorithm evaluates to determine the height of a line. Selecting this value too low could result in an incorrect estimation of the line's height. The value does not influence the runtime.



(a) Number of gripper depths = 5, score = 0.97

(b) Number of gripper depths = 50, score = 0.95

(c) Number of gripper depths = 500, score = 0.95

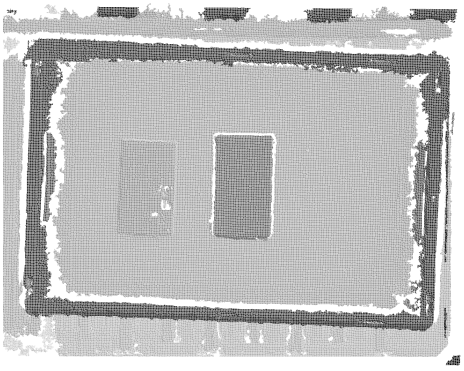
(d) Number of gripper depths = 5000, score = 0.95

Figure D.18: Highest-scoring grasps for different values of the number of gripper depths.

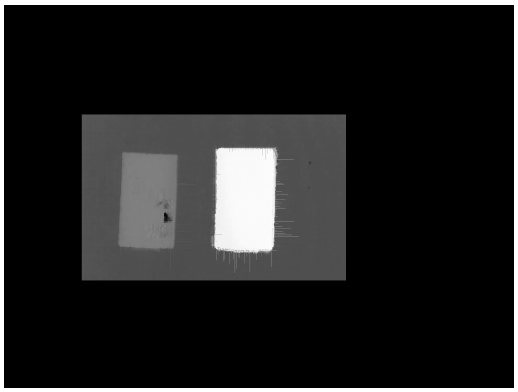
In summary, the parameters that most influence the resulting grasp and runtime are the fudge factor, peak factor, and angular difference. These parameters influence the edge detections, line detections, and line pairs, respectively. The number of peaks could be tuned alongside the peak factor to limit the runtime.



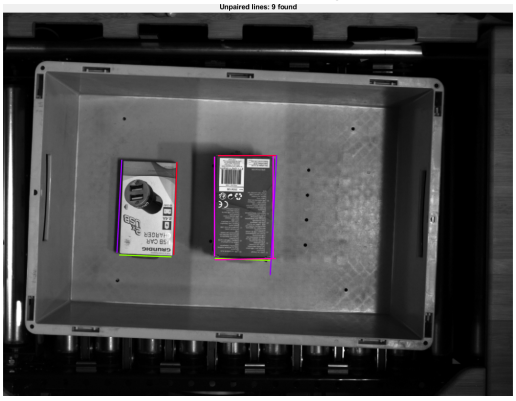
(a) The grey-scale image.



(b) The raw depth image.



(c) The filtered and segmented depth image.



(d) The detected lines for the default values of the parameters.



(e) The resulting grasp for the default values of the parameters.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.2000	2	100

(f) The default values of the parameters.

Figure D.9: The carrier considered while inspecting the parameters,
and the synthesis algorithm's outcome for the default values of the parameters.

E

Grasp Synthesis Results

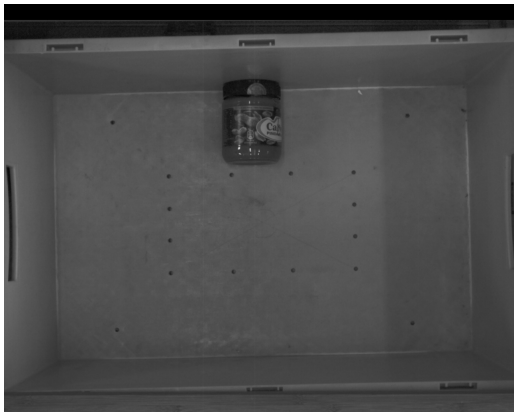
This appendix shows the results for the evaluation of the grasp synthesis approach. Sections E.1, E.3, E.2 and E.4 shows the results for the box segmentation, mapped segmentations, partial segmentations and complete segmentations, respectively.

E.1. Grasps for Box Segmentations

This appendix presents the grasp synthesis results for the box segmentation. For each test, the following figures are included: the greyscale image of the carrier, the raw depth image of the carrier, the depth image after filtering and segmentation, the detected edges, the detected lines, and three synthesized grasps. The three grasps are the grasps with the highest scores that are qualitatively different. Additionally, the values of synthesis algorithm's parameters that are used to achieve the results are included for each test. The results are presented in Figures B.3 through B.22. Furthermore, Table E.1 summarizes the contents of each carrier and the properties used for calculating correlations.

Table E.1: The contents of the carrier, the item grasped with the grasp that has the highest grasp score, and properties of said item. To compute the correlation coefficient, item properties have a value of 0, 1, or 2, where 0 denotes no presence of said property and 2 denotes full presence of that property. For example, a Transparency of 0 means the item is opaque.

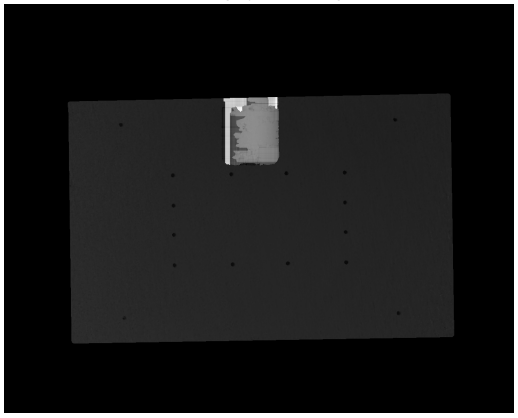
Carrier number	Contents	Item with highest grasp score	Curves	Size	Clutter	Reflectivity	Transparency
1	Peanut butter (side view)	Peanut butter (side view)	0	1	0	1	0
2	Peanut butter (top view)	Peanut butter (top view)	2	1	0	0	0
3	Two toothbrushes and peanut butter (top view)	Peanut butter (top view)	2	1	2	0	0
4	Three bottles of washing up liquid (overlap)	Washing up liquid (side view)	1	2	2	1	1
5	Three bottles of washing up liquid (no overlap)	Washing up liquid (side view)	1	2	1	1	1
6	Marbles (label on top)	Marbles	2	1	0	1	1
7	Marbles (label on side)	Marbles	2	1	0	1	1
8	Marbles and two tin cans	Tin can (top view)	2	0	0	2	0
9	Two pairs of socks	Pair of socks	1	2	2	0	0
10	One pair of socks	Pair of socks	1	2	0	0	0
11	Two belts (top view) and a pair of glasses	Belt (top view)	0	1	2	0	0
12	Two belts (side view) and a pair of glasses	Belt (side view)	2	1	1	0	0
13	Two computer mice (overlap)	Computer mouse	0	2	2	1	0
14	Two computer mice (no overlap)	Computer mouse	0	2	1	1	0
15	Walnuts and two packs of toothpicks (edge)	Walnuts	2	2	1	0	0
16	Walnuts and two packs of toothpicks (centre)	Toothpicks (top view)	2	0	1	1	1
17	Three cartons of milk (overlap)	Carton of milk (side view)	0	2	2	0	0
18	Three cartons of milk (no overlap)	Carton of milk (side view)	0	2	1	0	0
19	Five tubes of toothpaste (1 standing)	Tube of toothpaste (side view)	0	1	1	0	0
20	Five tubes of toothpaste (2 standing)	Tube of toothpaste (side view)	0	1	2	0	0



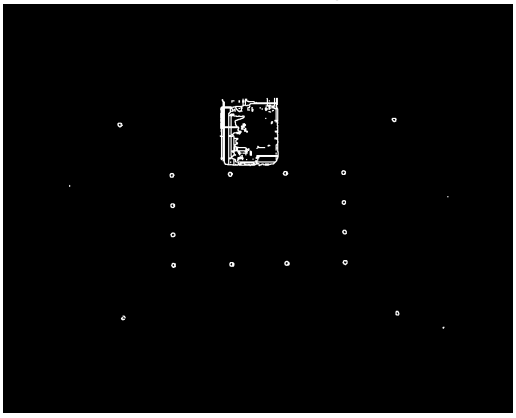
(a) The grey-scale image



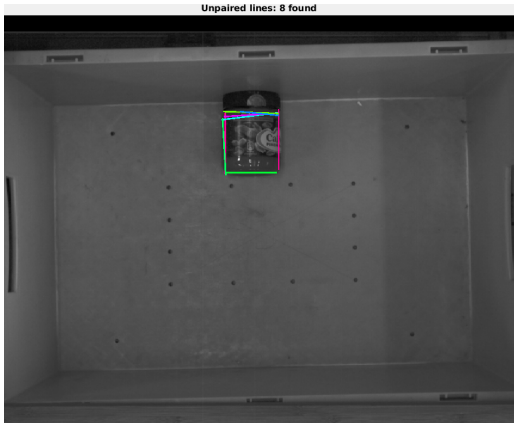
(b) The raw depth image.



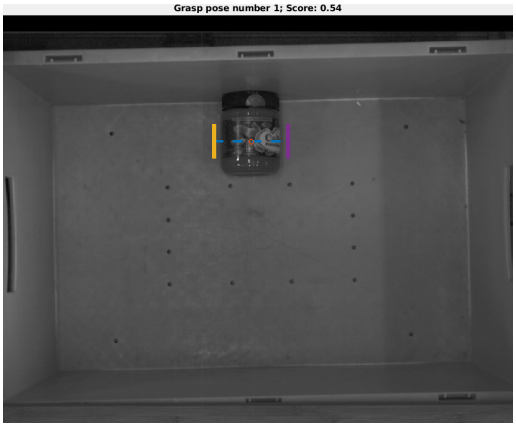
(c) The depth image after filtering and segmentation.



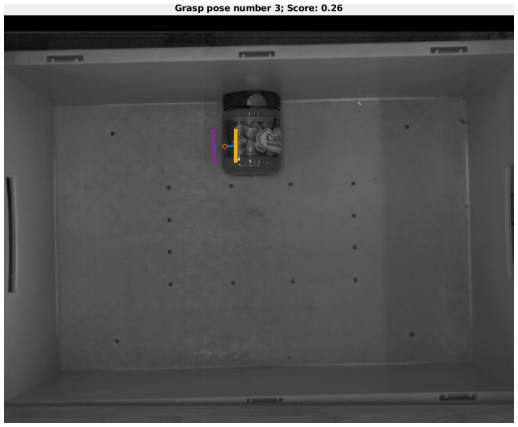
(d) The edges detected in the filtered and segmented depth image.



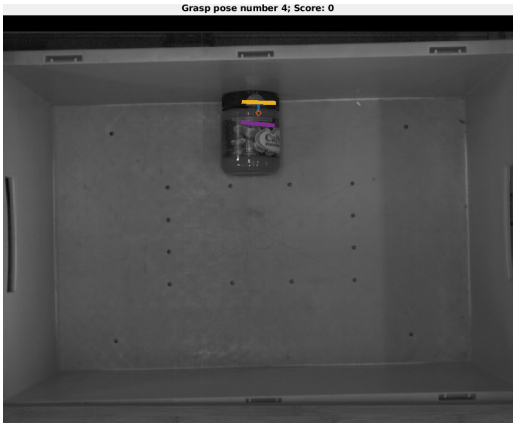
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.

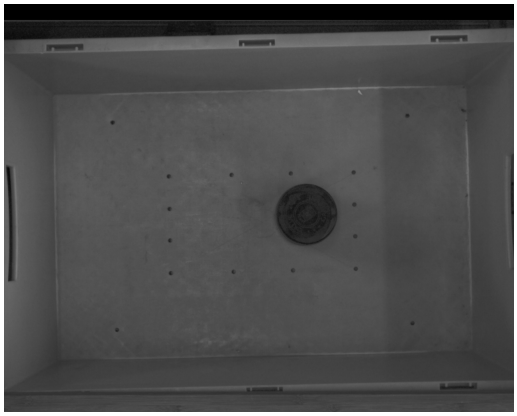


(h) One of the synthesized grasps.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1000	2	100

(i) The parameters used for this result.

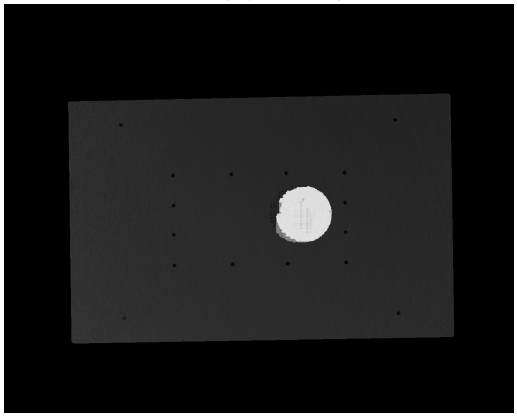
Figure E.1: Grasp synthesis for a peanut butter jar on its side.



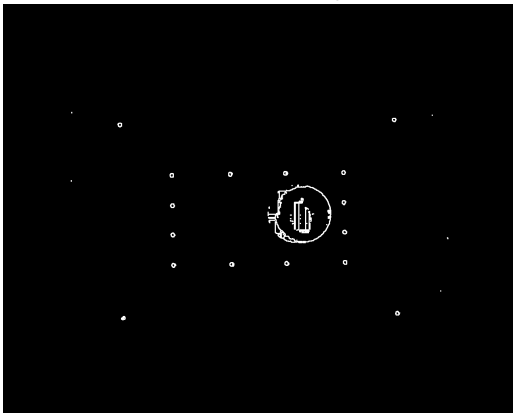
(a) The grey-scale image



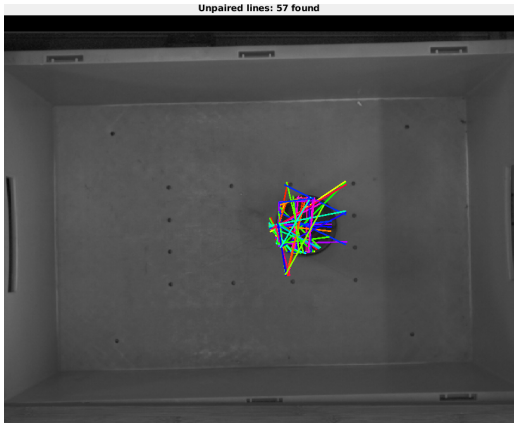
(b) The raw depth image.



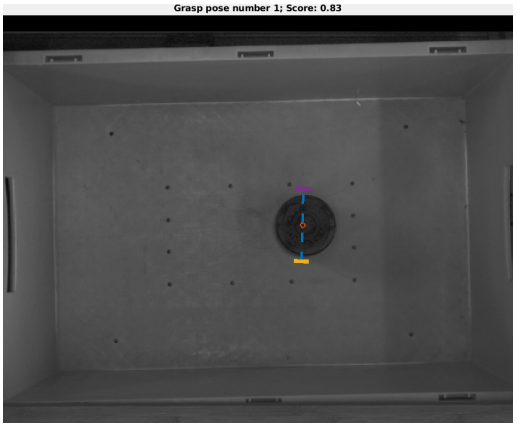
(c) The depth image after filtering and segmentation.



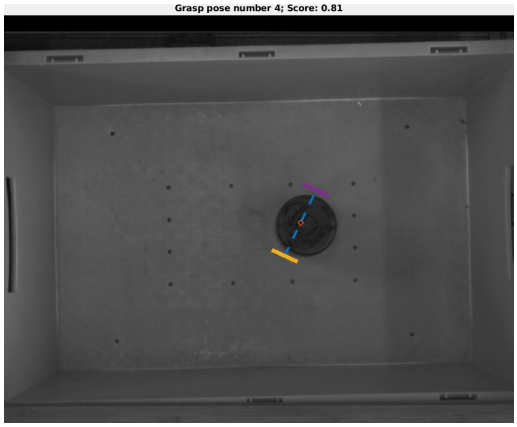
(d) The edges detected in the filtered and segmented depth image.



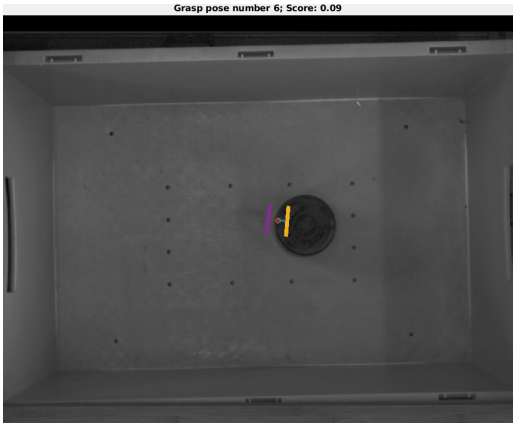
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.

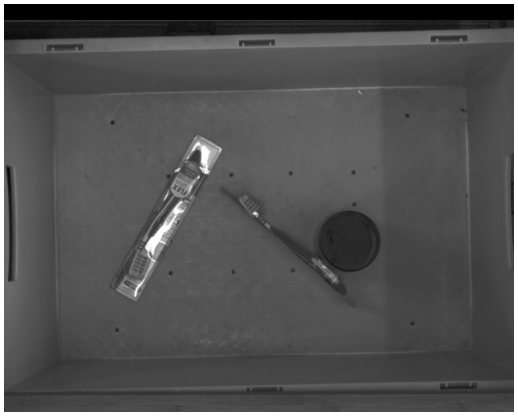


(h) One of the synthesized grasps.

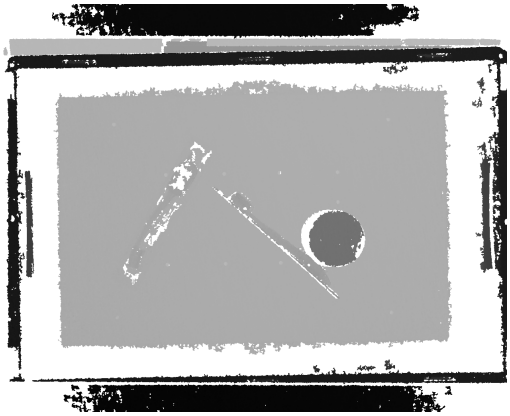
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.0500	2	100

(i) The parameters used for this result.

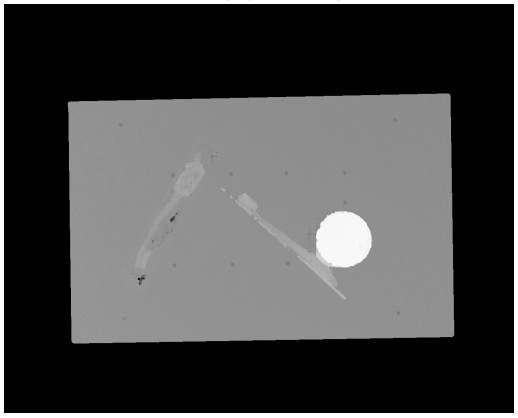
Figure E.2: Grasp synthesis for a peanut butter jar.



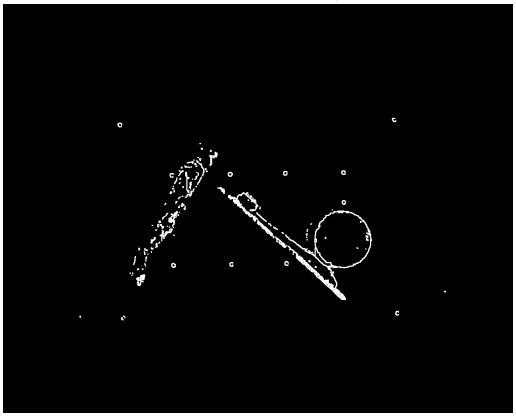
(a) The grey-scale image



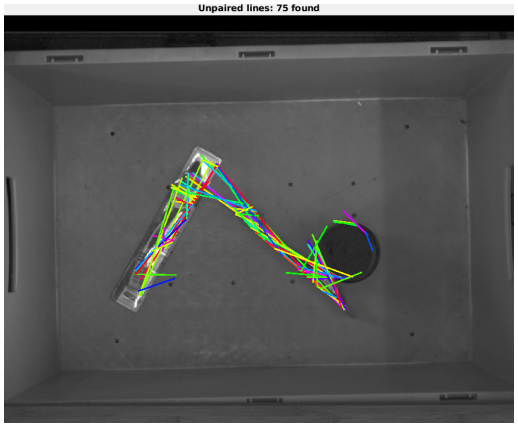
(b) The raw depth image.



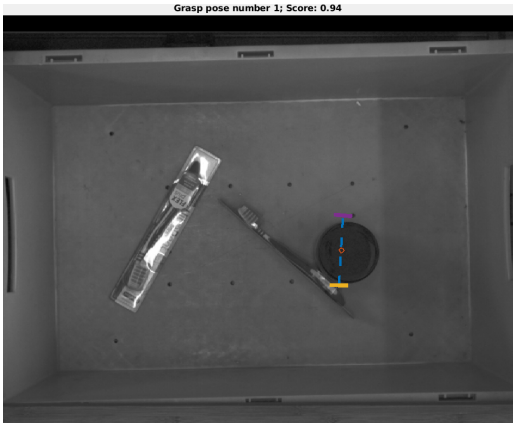
(c) The depth image after filtering and segmentation.



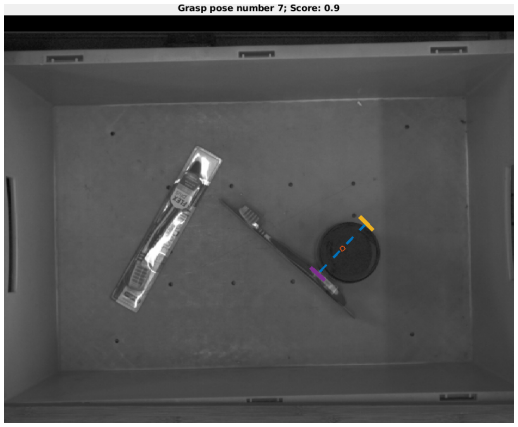
(d) The edges detected in the filtered and segmented depth image.



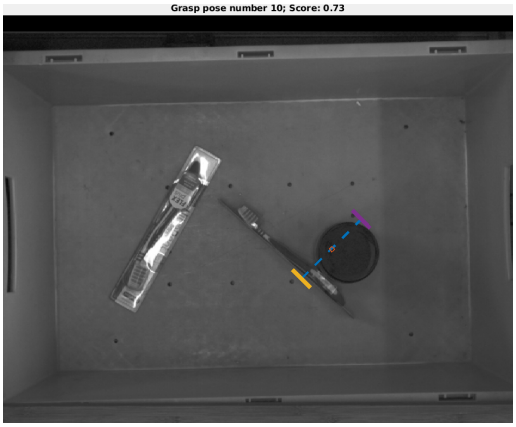
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.



(h) One of the synthesized grasps.

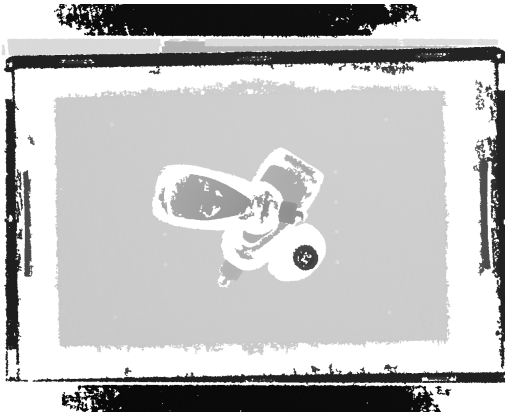
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	0.5000	80	3	200	0.0500	2	100

(i) The parameters used for this result.

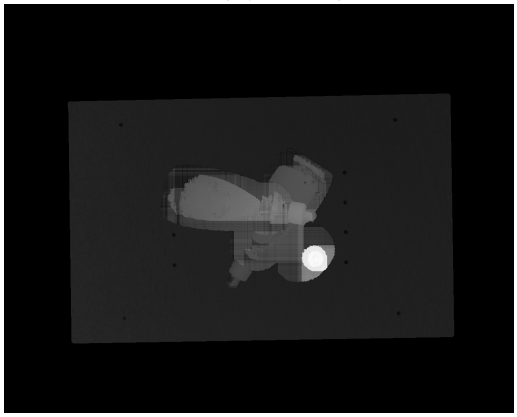
Figure E.3: Grasp synthesis for a peanut butter jar and two toothbrushes.



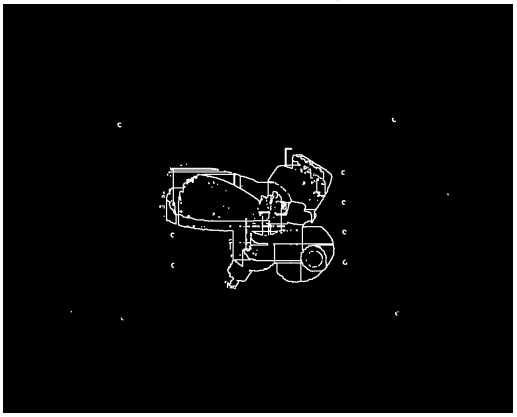
(a) The grey-scale image



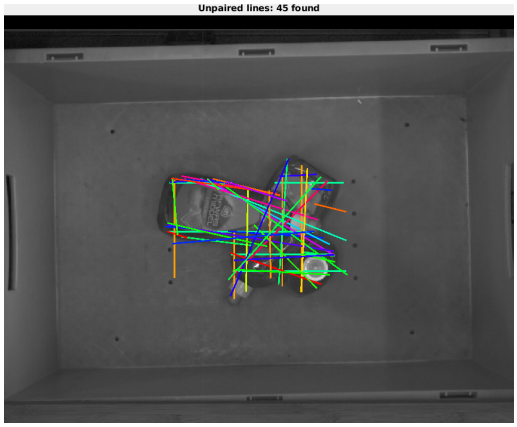
(b) The raw depth image.



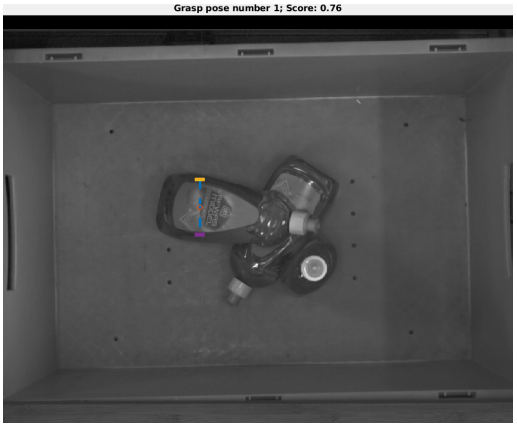
(c) The depth image after filtering and segmentation.



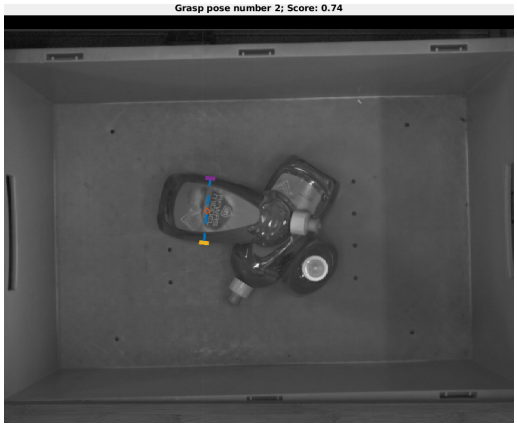
(d) The edges detected in the filtered and segmented depth image.



(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.



(h) One of the synthesized grasps.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1.5000	80	3	200	0.1000	2	100

(i) The parameters used for this result.

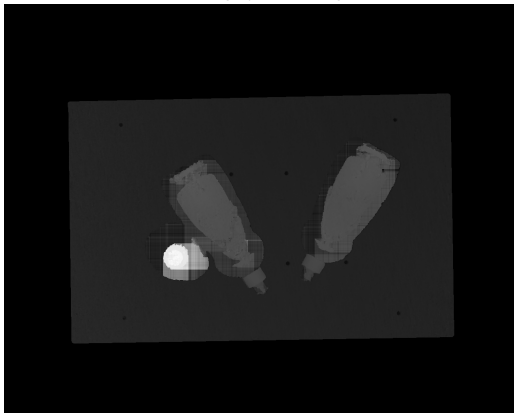
Figure E.4: Grasp synthesis for three bottles of washing up liquid.



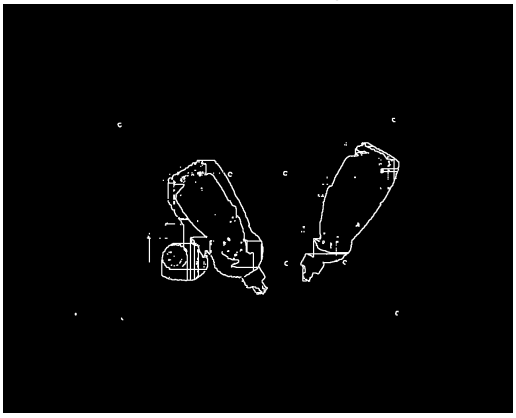
(a) The grey-scale image



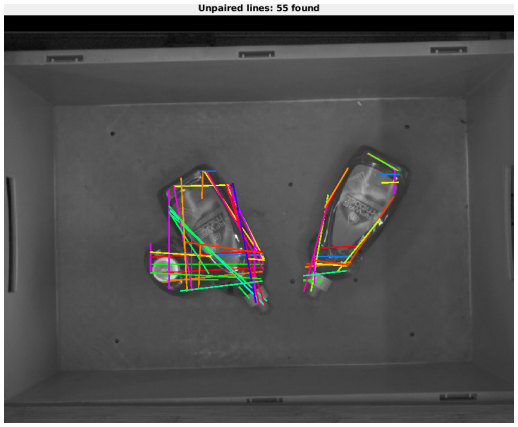
(b) The raw depth image.



(c) The depth image after filtering and segmentation.



(d) The edges detected in the filtered and segmented depth image.



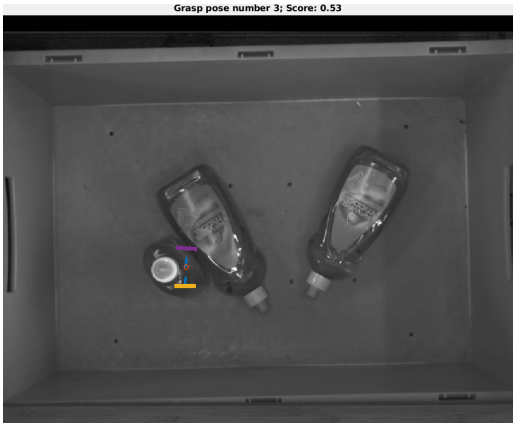
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.

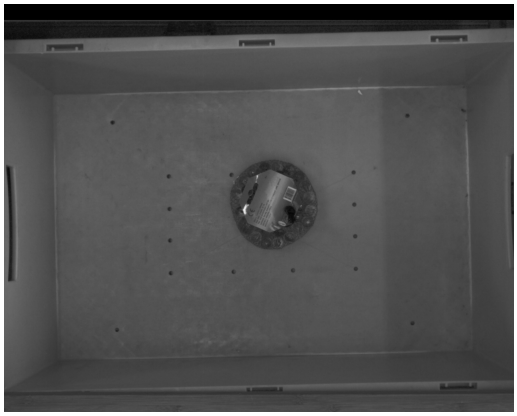


(h) One of the synthesized grasps.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1.5000	80	3	200	0.0800	2	100

(i) The parameters used for this result.

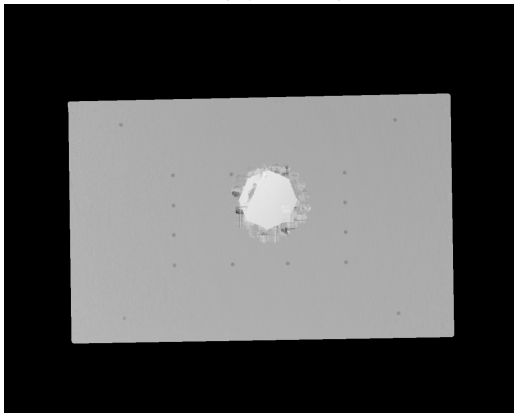
Figure E.5: Grasp synthesis for three bottles of washing up liquid.



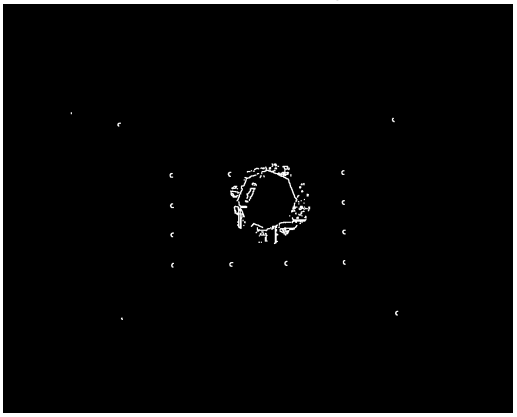
(a) The grey-scale image



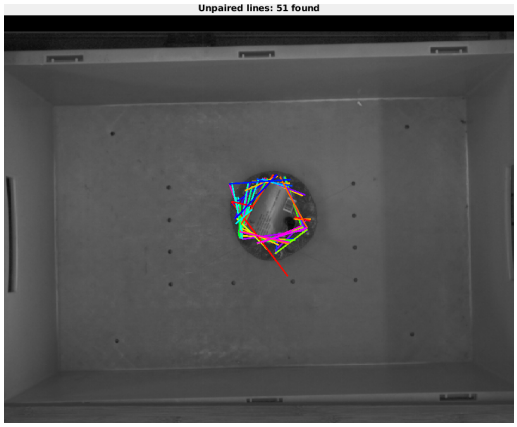
(b) The raw depth image.



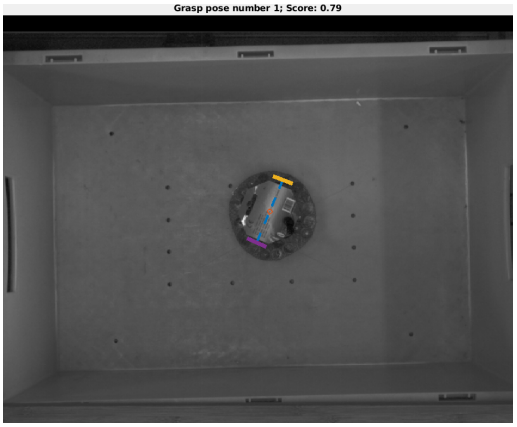
(c) The depth image after filtering and segmentation.



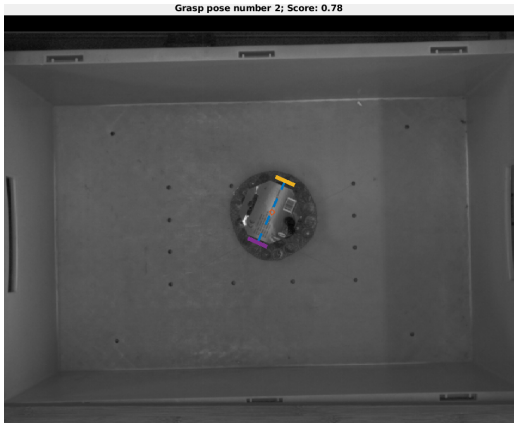
(d) The edges detected in the filtered and segmented depth image.



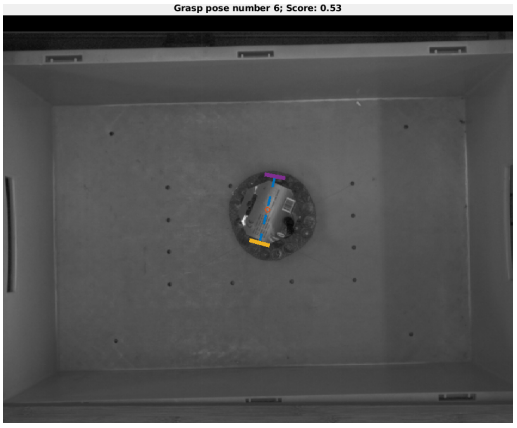
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.

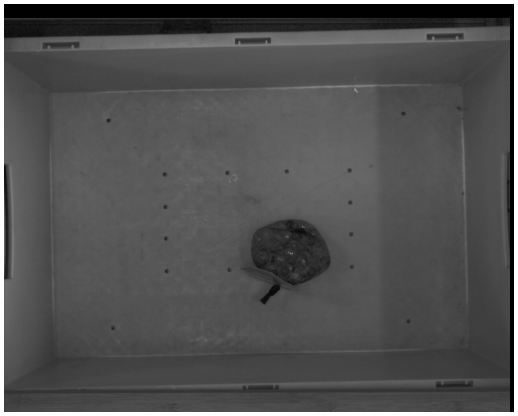


(h) One of the synthesized grasps.

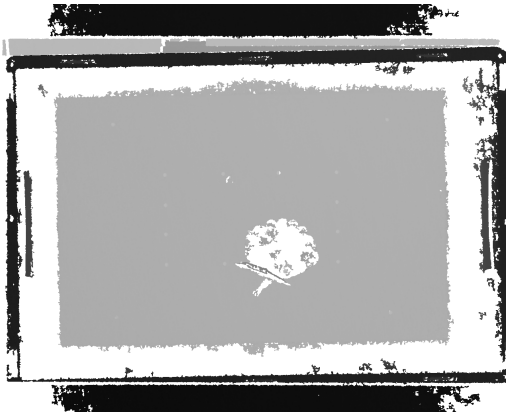
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.0500	2	100

(i) The parameters used for this result.

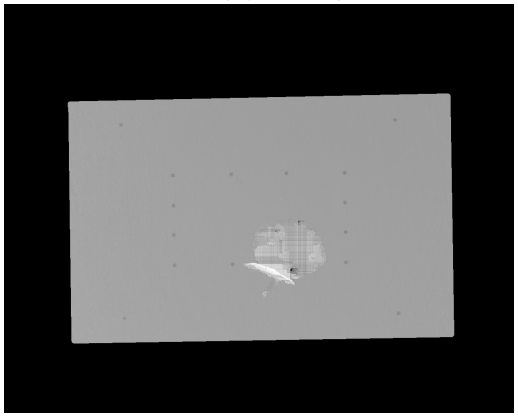
Figure E.6: Grasp synthesis for a net of marbles.



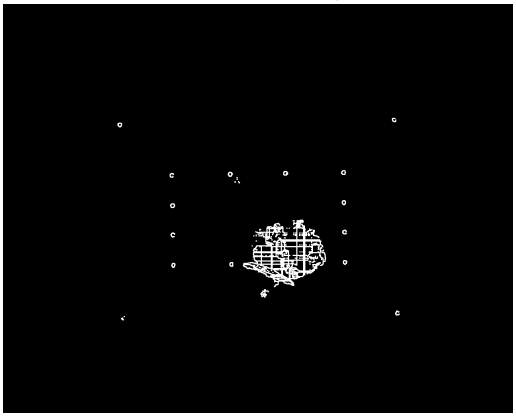
(a) The grey-scale image



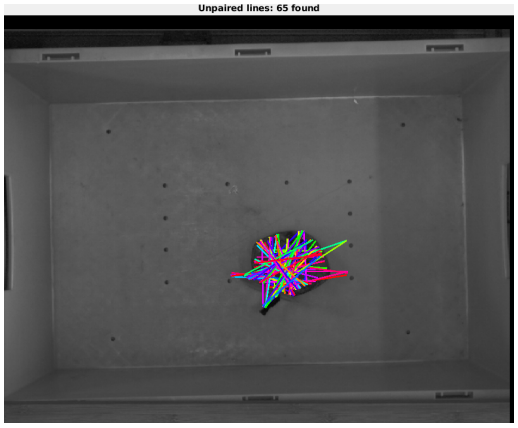
(b) The raw depth image.



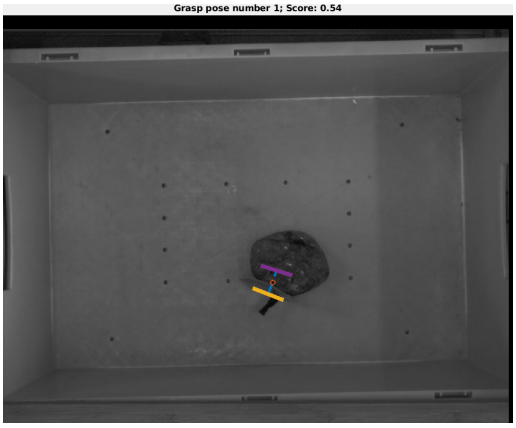
(c) The depth image after filtering and segmentation.



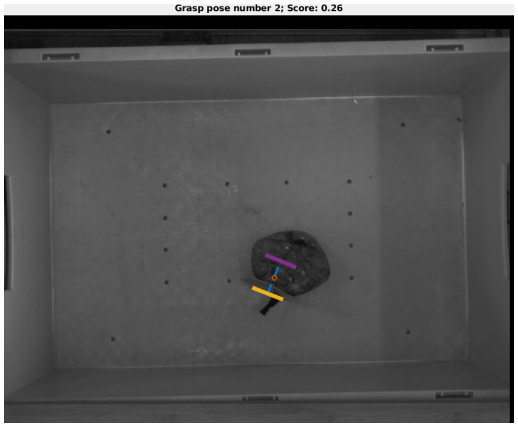
(d) The edges detected in the filtered and segmented depth image.



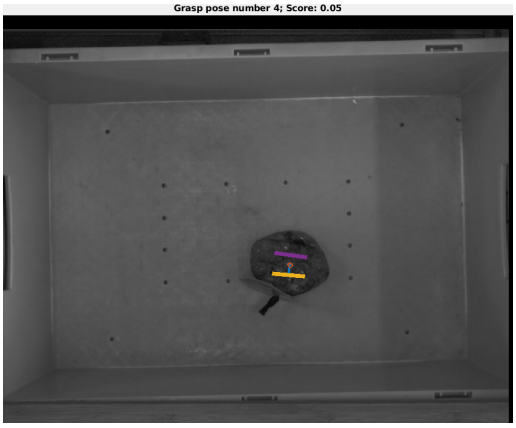
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.



(h) One of the synthesized grasps.

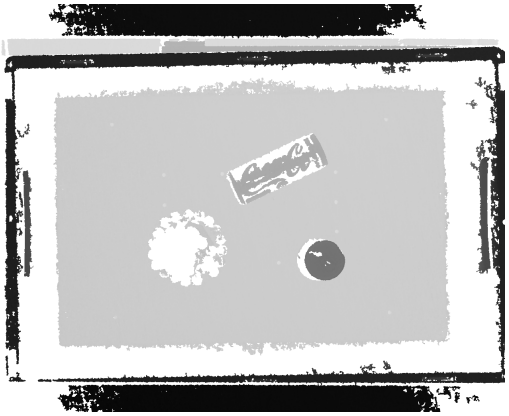
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	0.5000	80	3	200	0.0800	2	100

(i) The parameters used for this result.

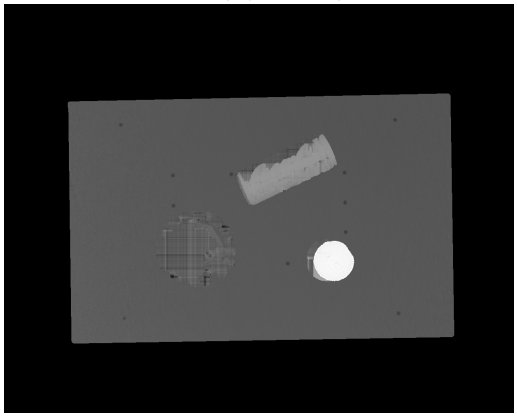
Figure E.7: Grasp synthesis for a net of marbles.



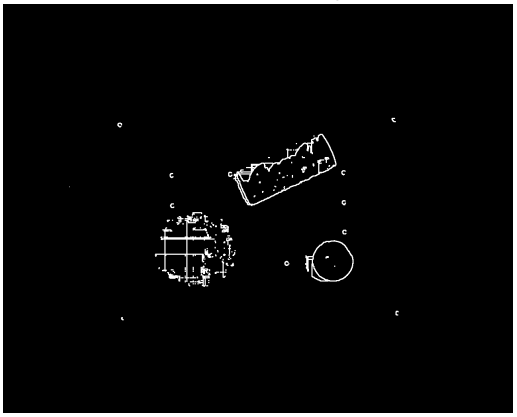
(a) The grey-scale image



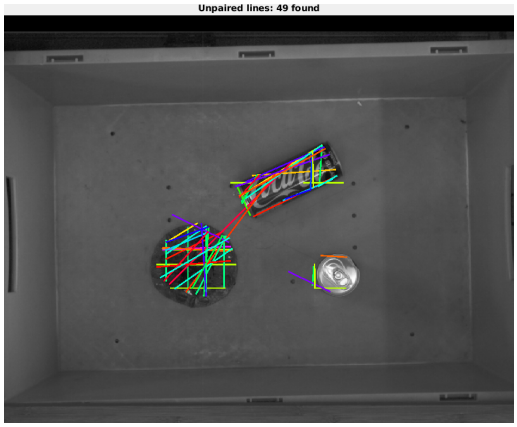
(b) The raw depth image.



(c) The depth image after filtering and segmentation.



(d) The edges detected in the filtered and segmented depth image.



(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.

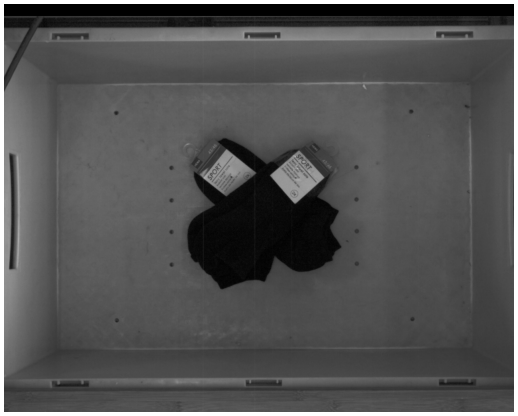


(h) One of the synthesized grasps.

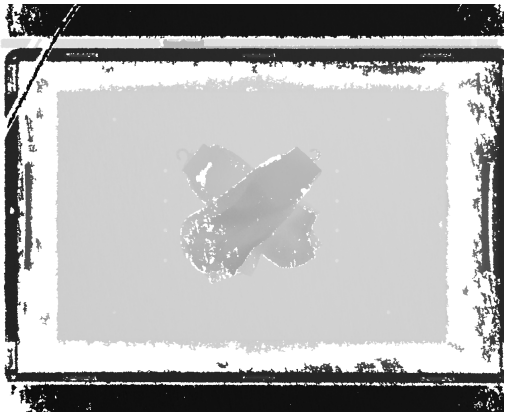
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.0800	2	100

(i) The parameters used for this result.

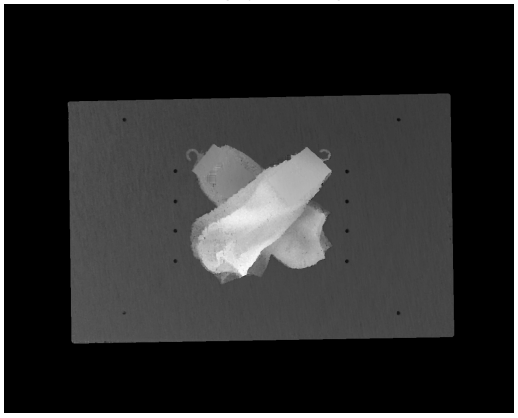
Figure E.8: Grasp synthesis for a net of marbles and two tin cans.



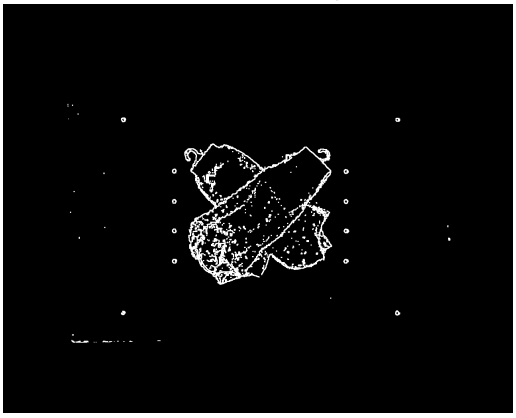
(a) The grey-scale image



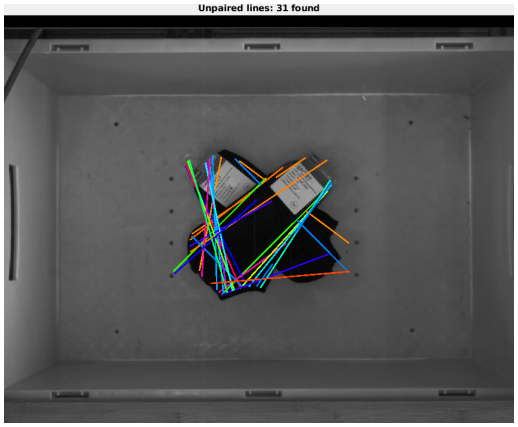
(b) The raw depth image.



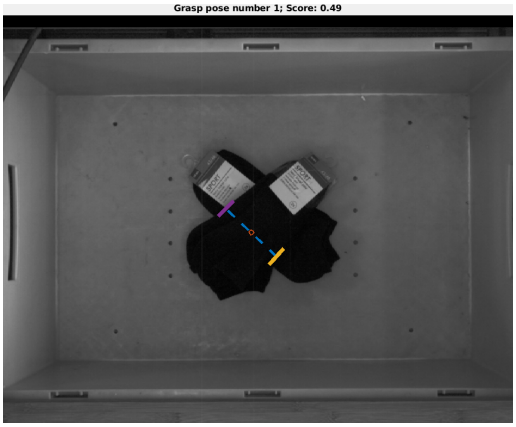
(c) The depth image after filtering and segmentation.



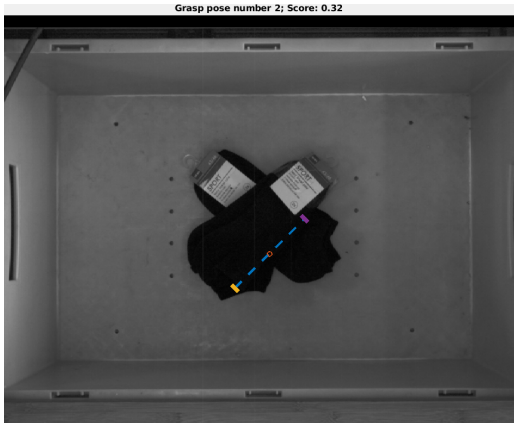
(d) The edges detected in the filtered and segmented depth image.



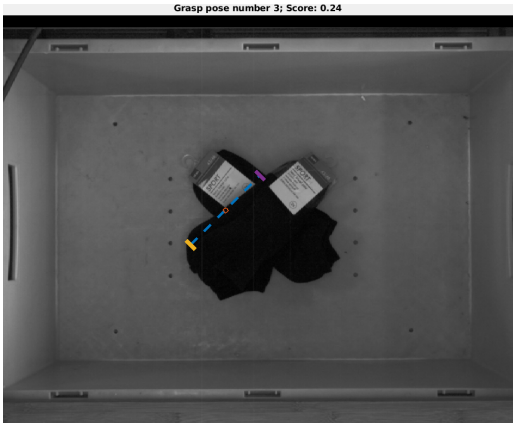
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.



(h) One of the synthesized grasps.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1000	2	100

(i) The parameters used for this result.

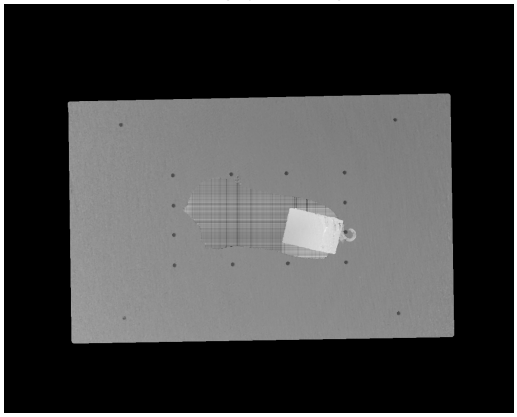
Figure E.9: Grasp synthesis for two sets of socks.



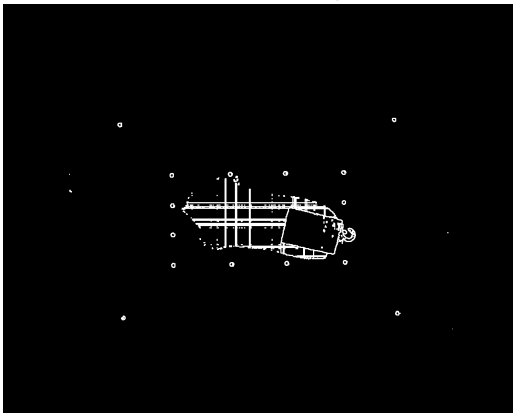
(a) The grey-scale image



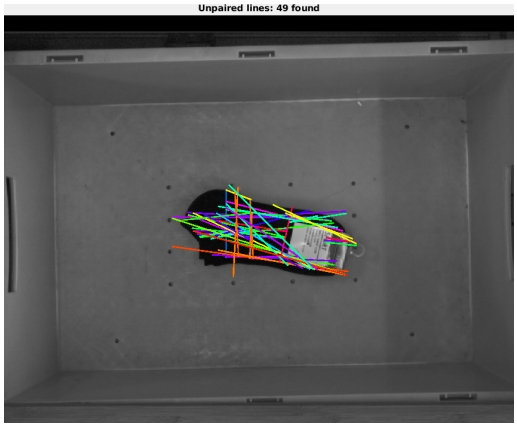
(b) The raw depth image.



(c) The depth image after filtering and segmentation.



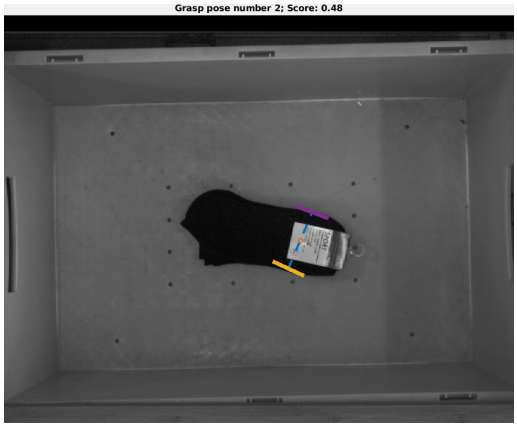
(d) The edges detected in the filtered and segmented depth image.



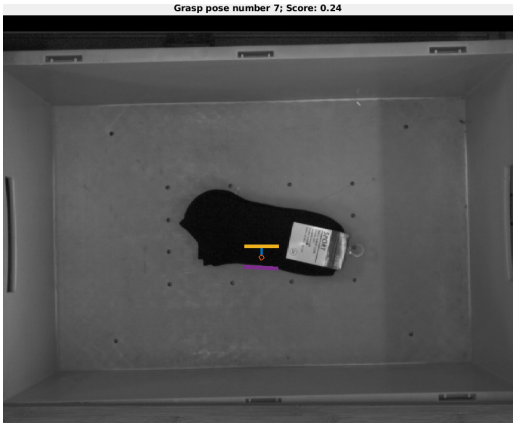
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.

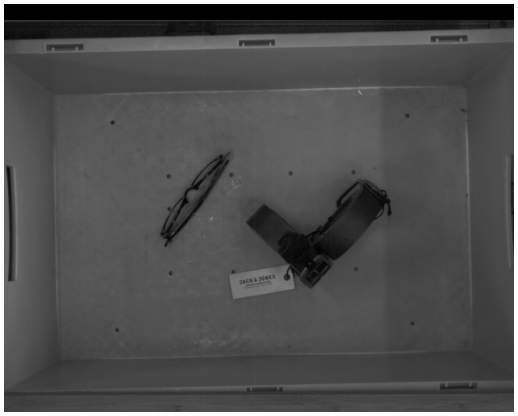


(h) One of the synthesized grasps.

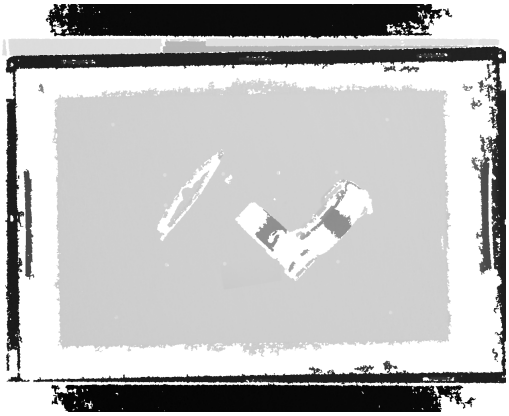
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1000	2	100

(i) The parameters used for this result.

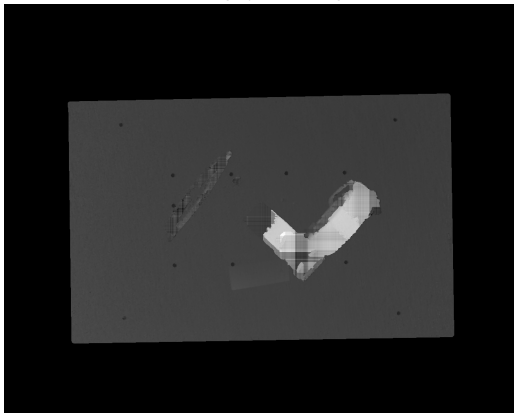
Figure E.10: Grasp synthesis for a set of socks.



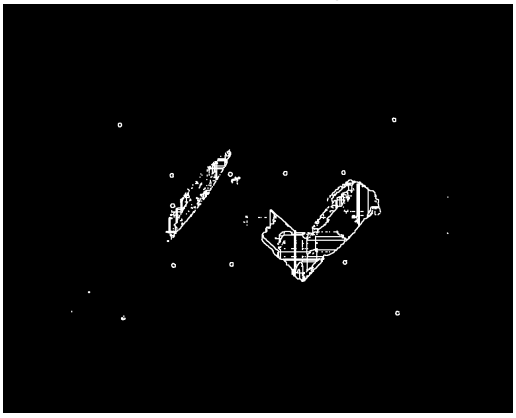
(a) The grey-scale image



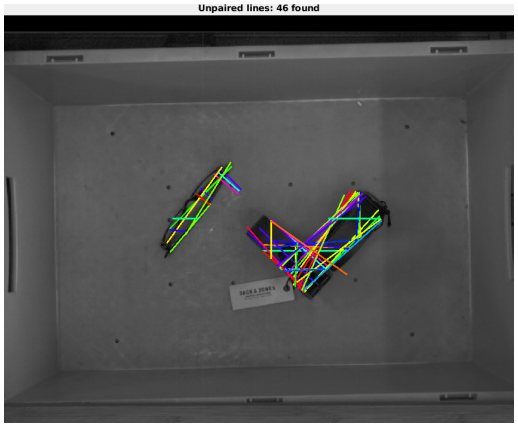
(b) The raw depth image.



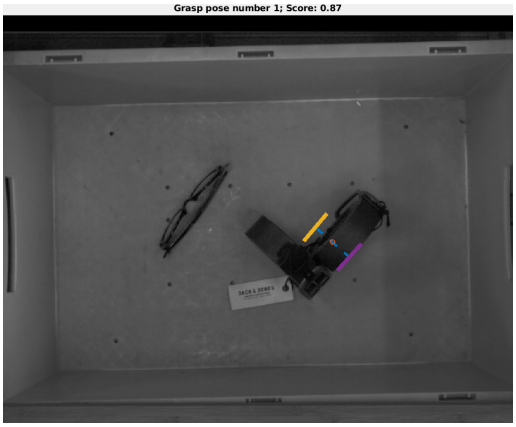
(c) The depth image after filtering and segmentation.



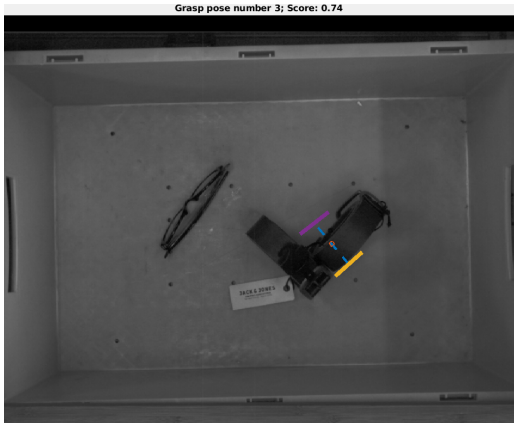
(d) The edges detected in the filtered and segmented depth image.



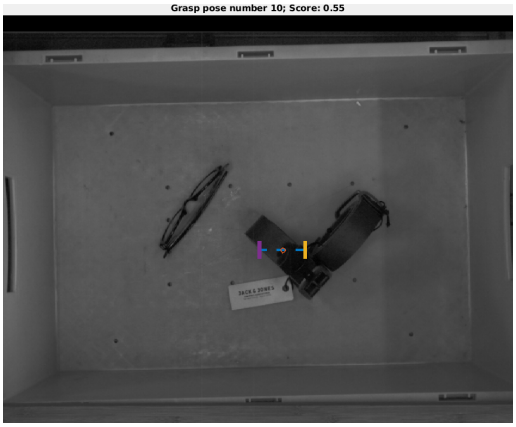
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.



(h) One of the synthesized grasps.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1000	2	100

(i) The parameters used for this result.

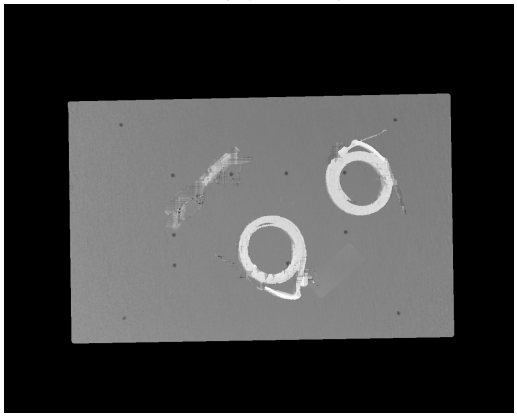
Figure E.11: Grasp synthesis for two belts and a pair of glasses.



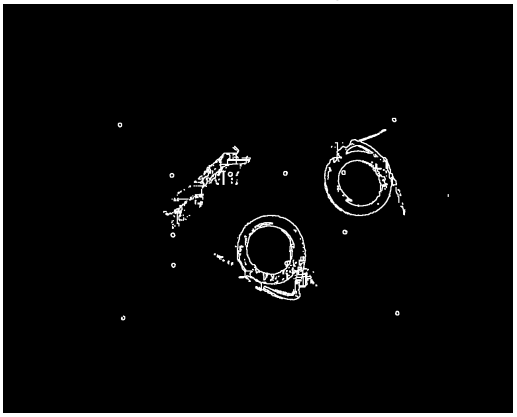
(a) The grey-scale image



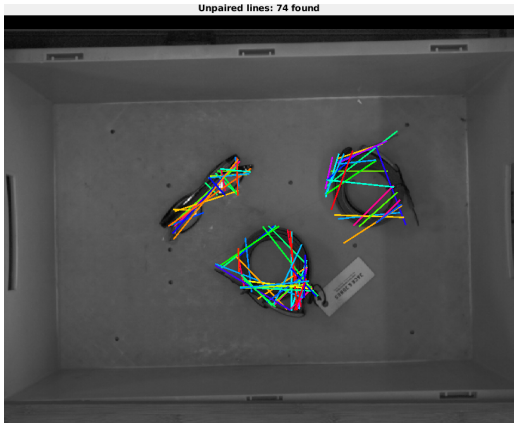
(b) The raw depth image.



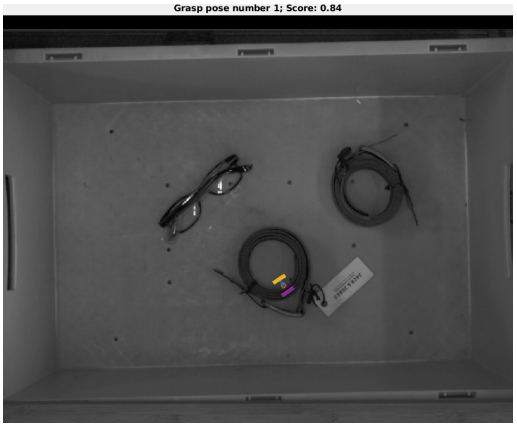
(c) The depth image after filtering and segmentation.



(d) The edges detected in the filtered and segmented depth image.



(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.

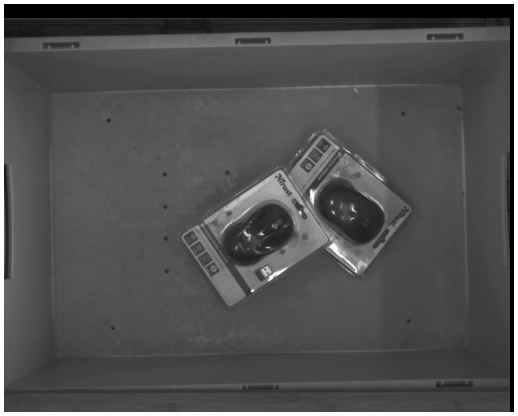


(h) One of the synthesized grasps.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.0900	2	100

(i) The parameters used for this result.

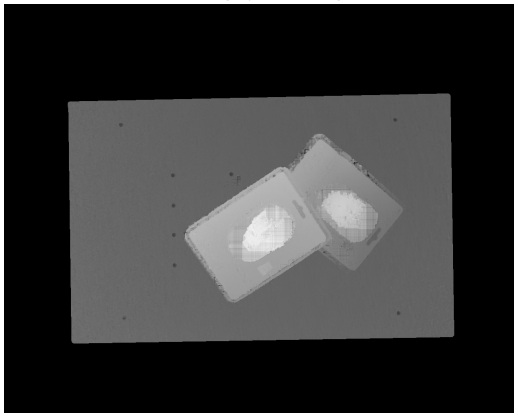
Figure E.12: Grasp synthesis for two belts and a pair of glasses.



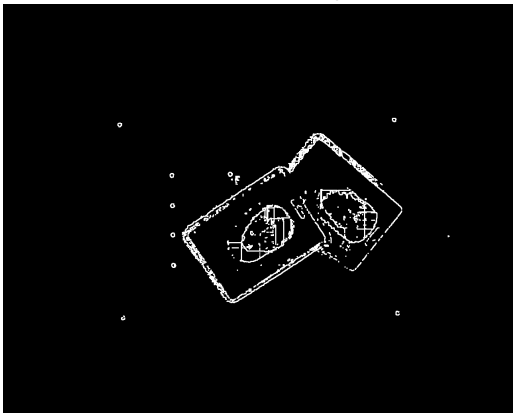
(a) The grey-scale image



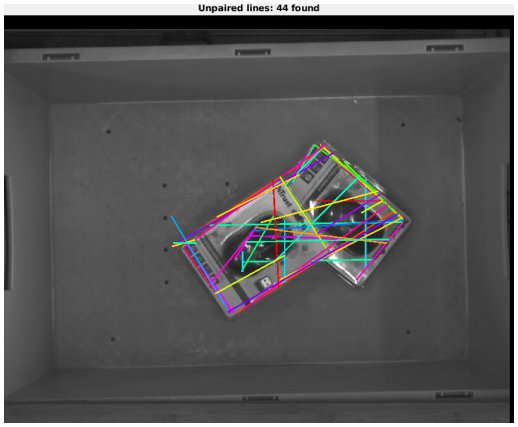
(b) The raw depth image.



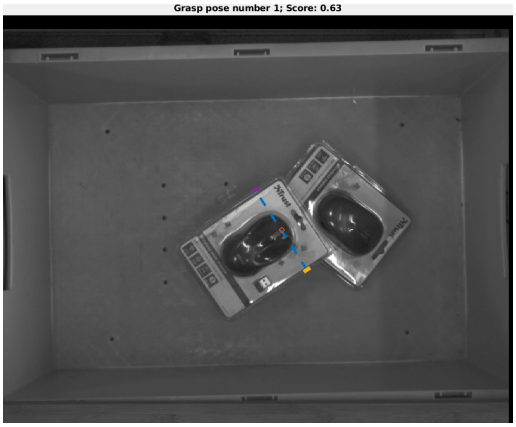
(c) The depth image after filtering and segmentation.



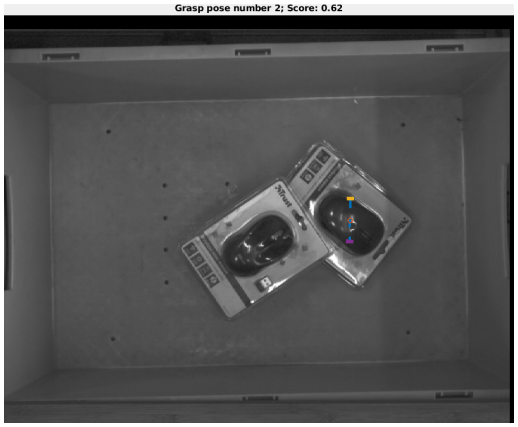
(d) The edges detected in the filtered and segmented depth image.



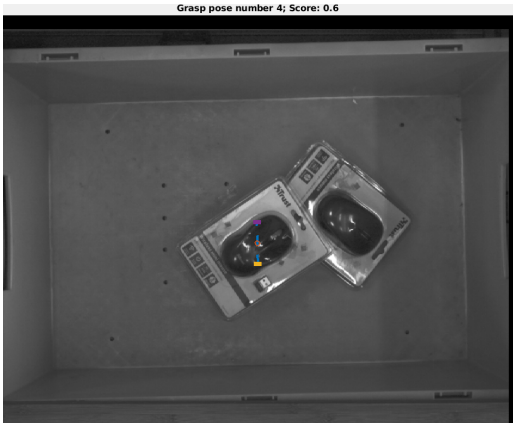
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.



(h) One of the synthesized grasps.

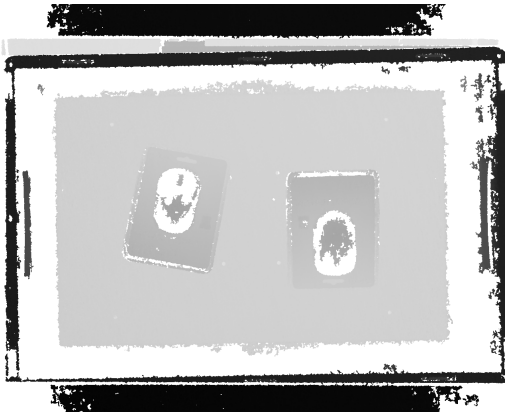
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1000	2	100

(i) The parameters used for this result.

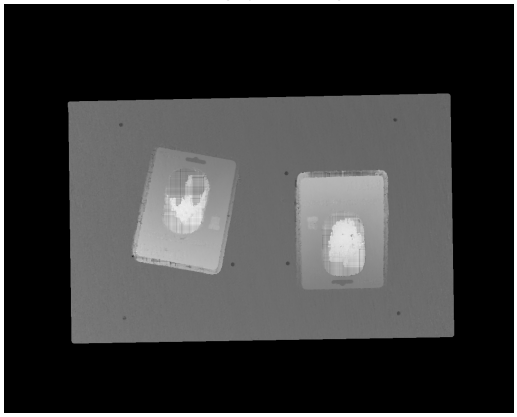
Figure E.13: Grasp synthesis for two computer mice that overlap.



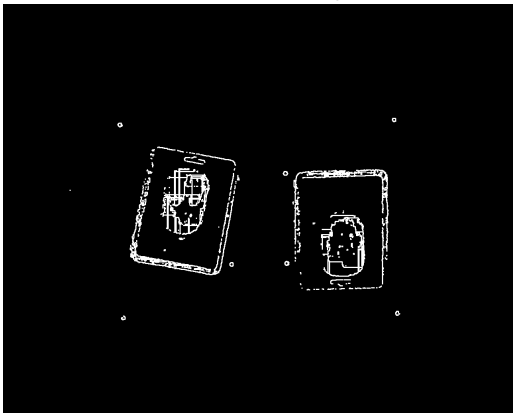
(a) The grey-scale image



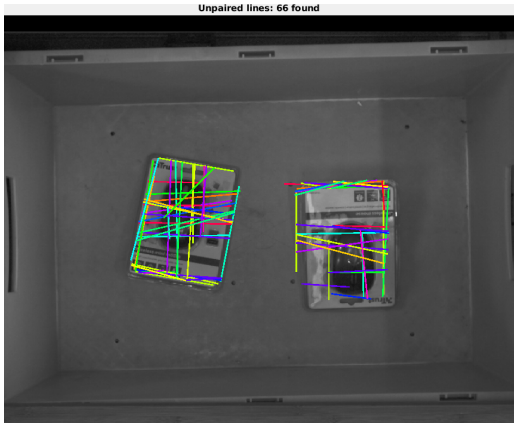
(b) The raw depth image.



(c) The depth image after filtering and segmentation.



(d) The edges detected in the filtered and segmented depth image.



(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.

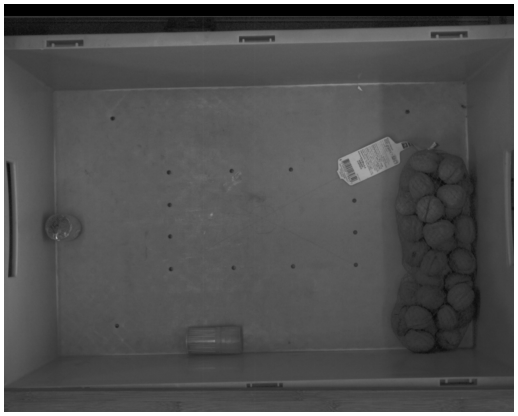


(h) One of the synthesized grasps.

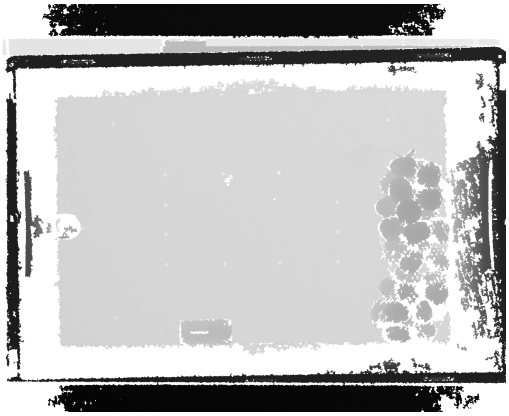
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1000	2	100

(i) The parameters used for this result.

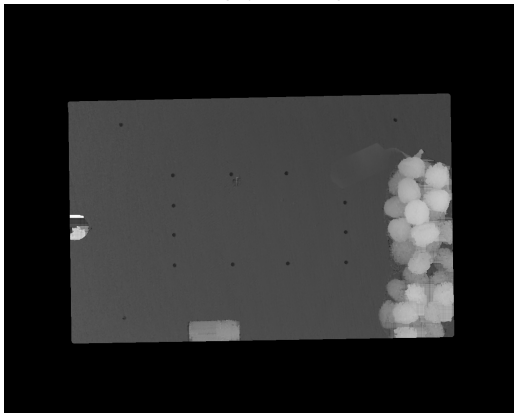
Figure E.14: Grasp synthesis for two computer mice.



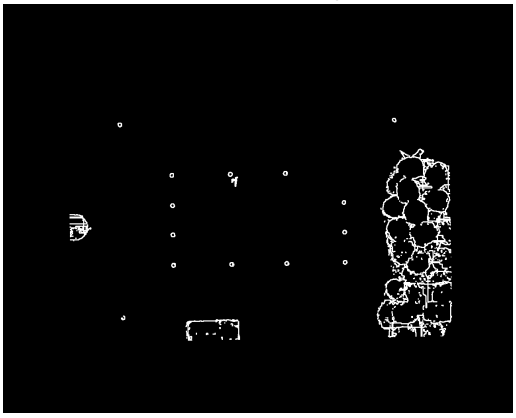
(a) The grey-scale image



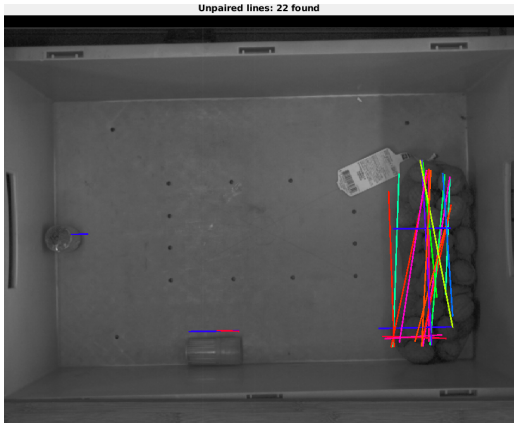
(b) The raw depth image.



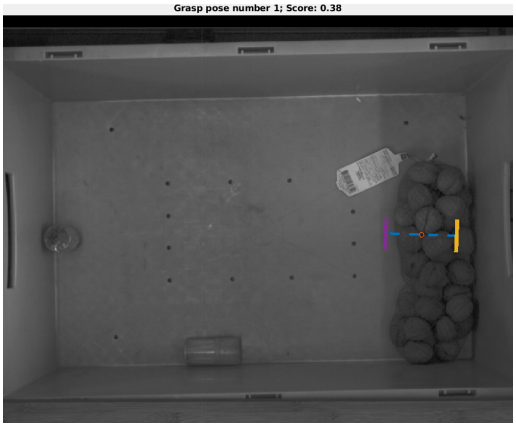
(c) The depth image after filtering and segmentation.



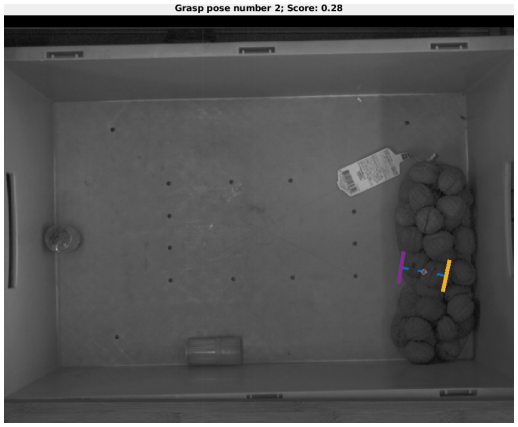
(d) The edges detected in the filtered and segmented depth image.



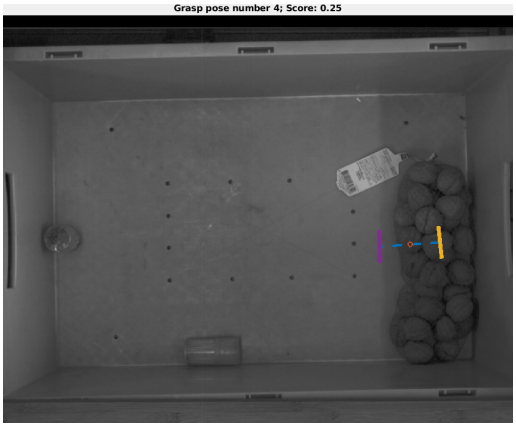
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.



(h) One of the synthesized grasps.

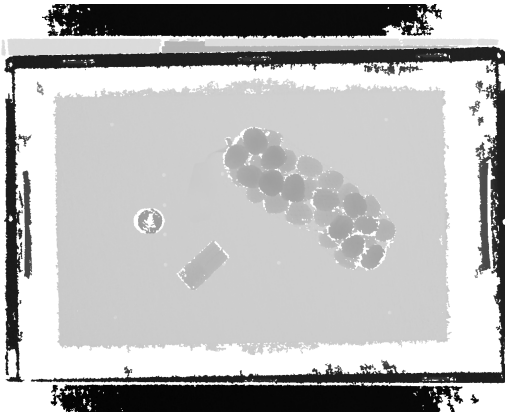
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1200	2	100

(i) The parameters used for this result.

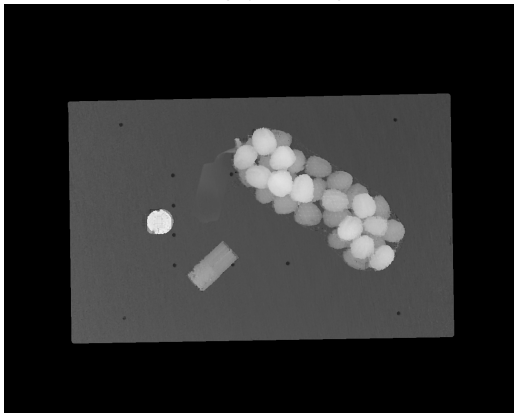
Figure E.15: Grasp synthesis for a net of walnuts and two packs of toothpicks, all at the edge of the carrier.



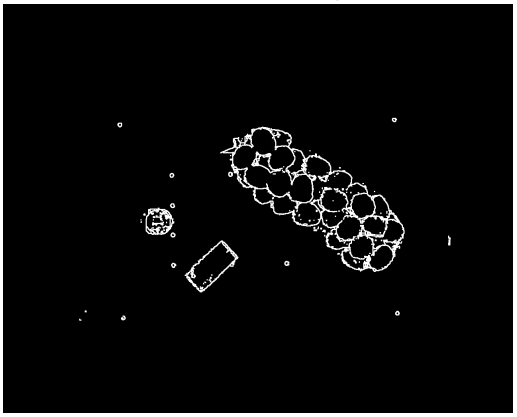
(a) The grey-scale image



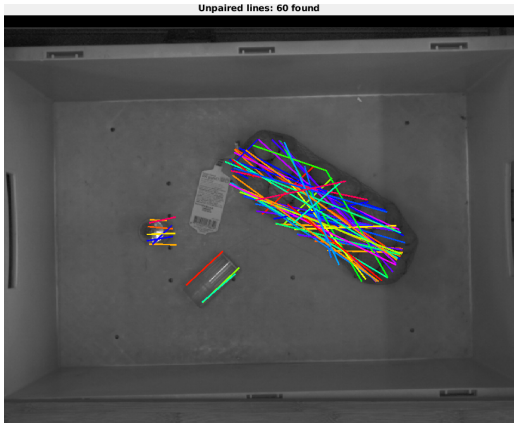
(b) The raw depth image.



(c) The depth image after filtering and segmentation.



(d) The edges detected in the filtered and segmented depth image.



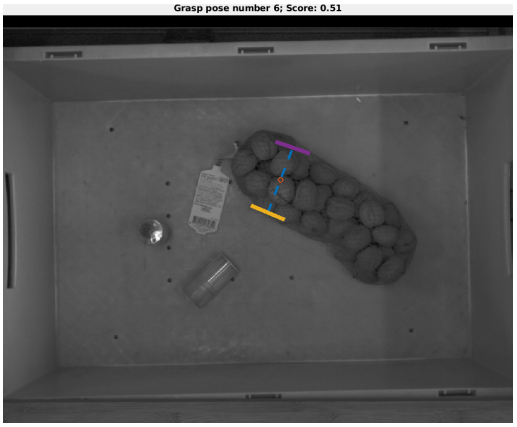
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.

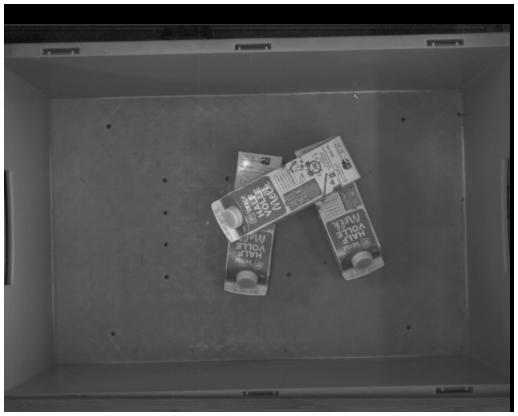


(h) One of the synthesized grasps.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1000	2	100

(i) The parameters used for this result.

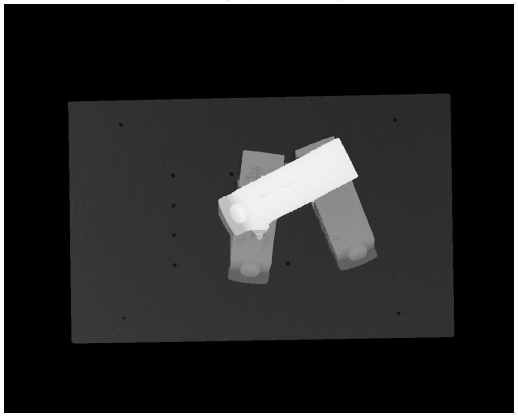
Figure E.16: Grasp synthesis for a net of walnuts and two packs of toothpicks.



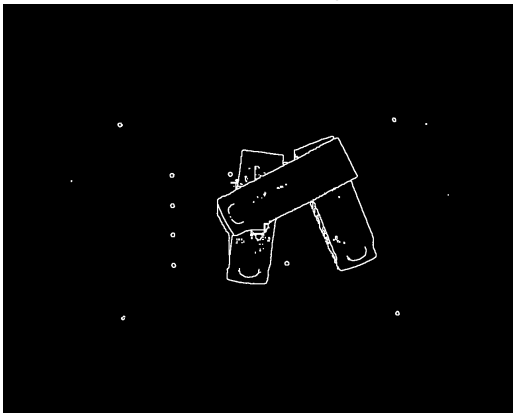
(a) The grey-scale image



(b) The raw depth image.



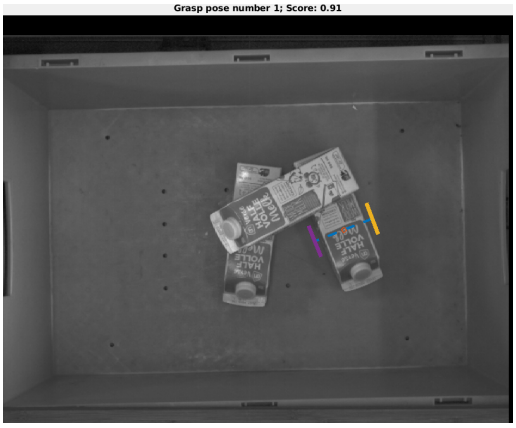
(c) The depth image after filtering and segmentation.



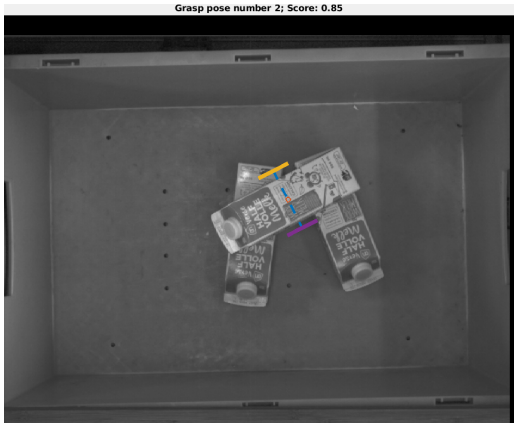
(d) The edges detected in the filtered and segmented depth image.



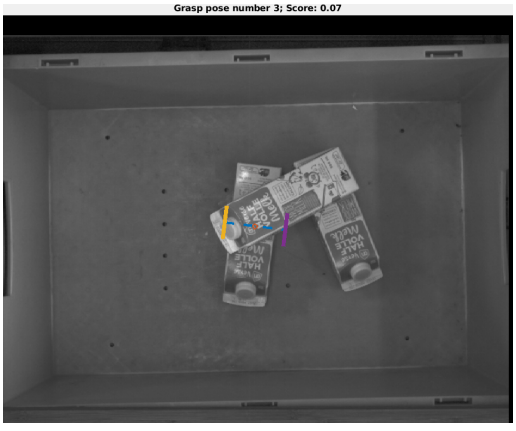
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.



(h) One of the synthesized grasps.

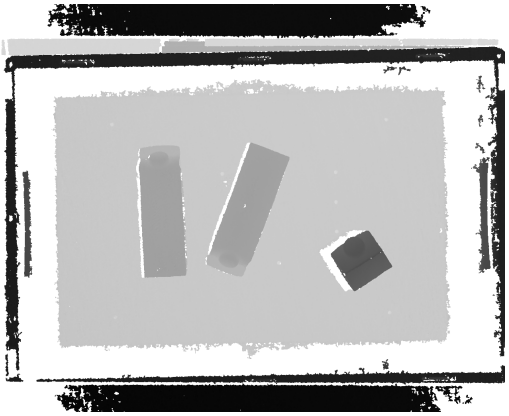
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1200	2	100

(i) The parameters used for this result.

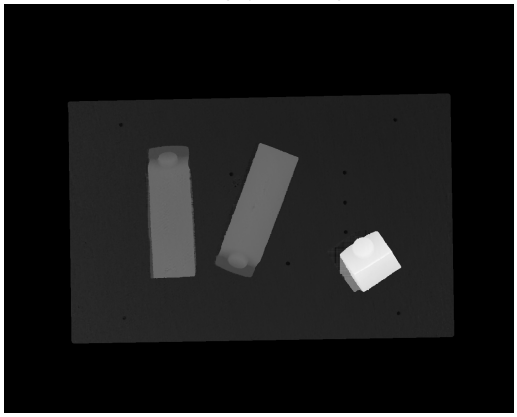
Figure E.17: Grasp synthesis for three cartons of milk that overlap.



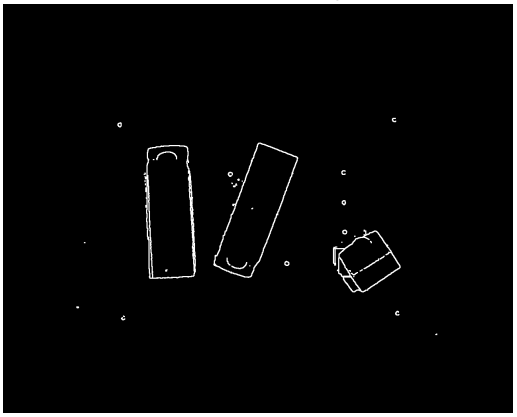
(a) The grey-scale image



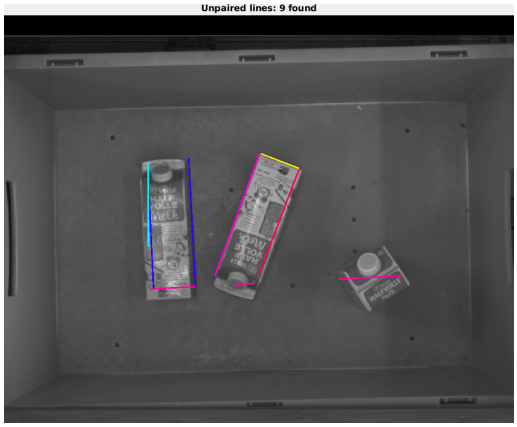
(b) The raw depth image.



(c) The depth image after filtering and segmentation.



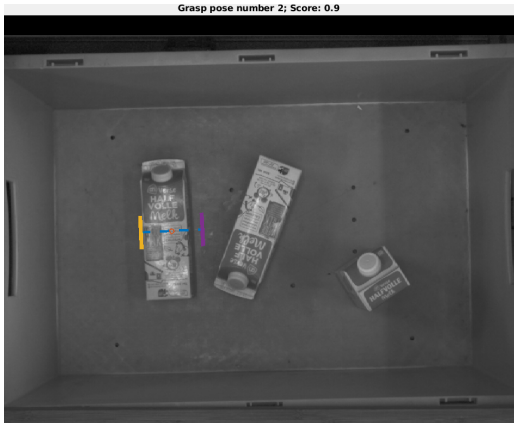
(d) The edges detected in the filtered and segmented depth image.



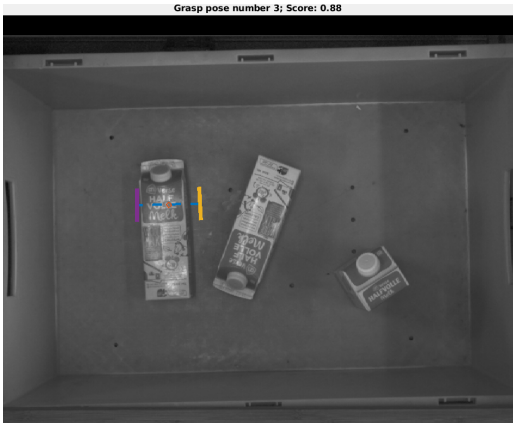
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.

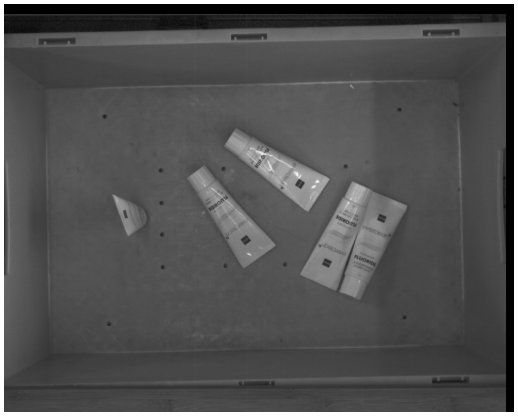


(h) One of the synthesized grasps.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1200	2	100

(i) The parameters used for this result.

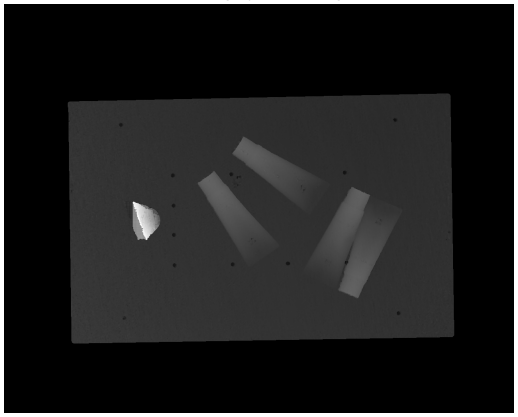
Figure E.18: Grasp synthesis for three cartons of milk.



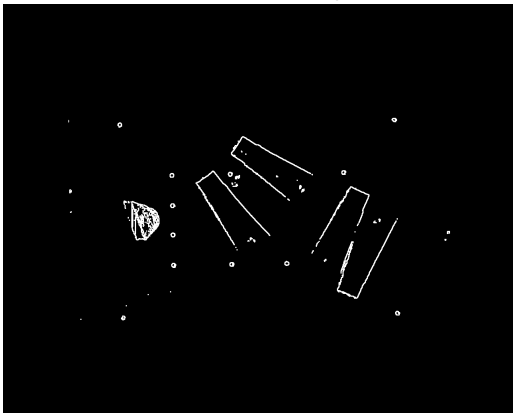
(a) The grey-scale image



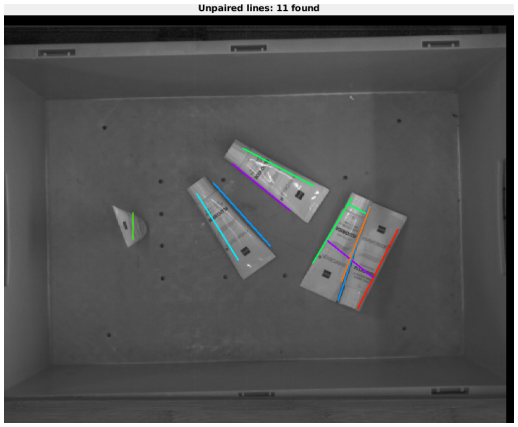
(b) The raw depth image.



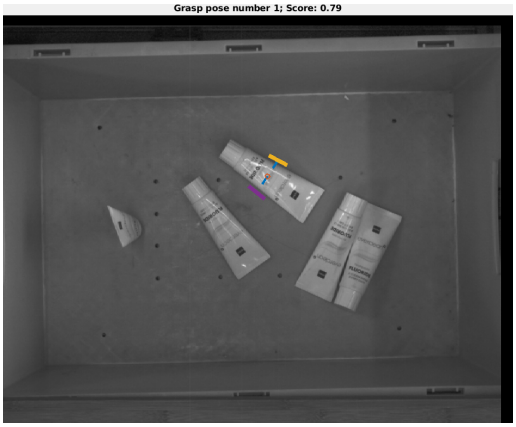
(c) The depth image after filtering and segmentation.



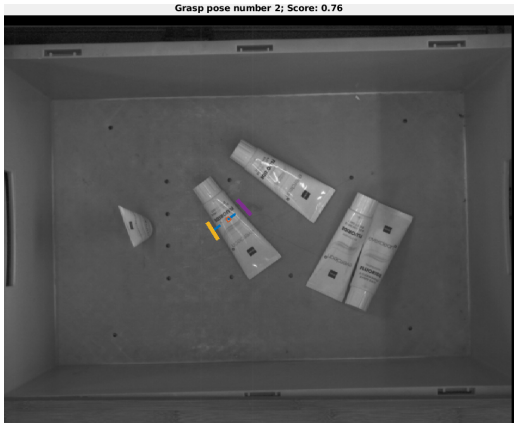
(d) The edges detected in the filtered and segmented depth image.



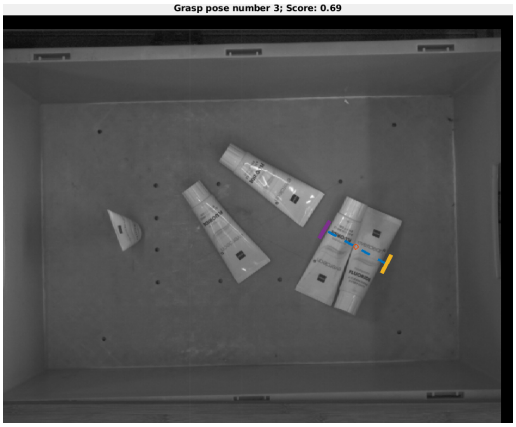
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.

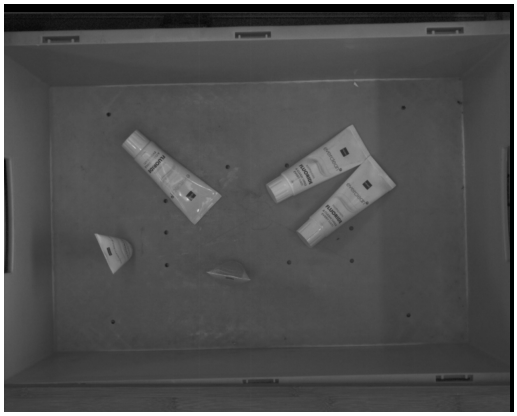


(h) One of the synthesized grasps.

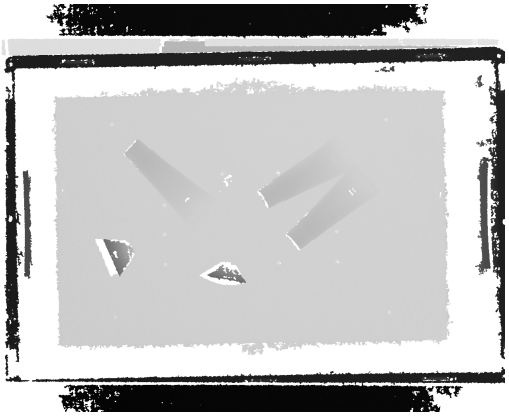
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1200	6	100

(i) The parameters used for this result.

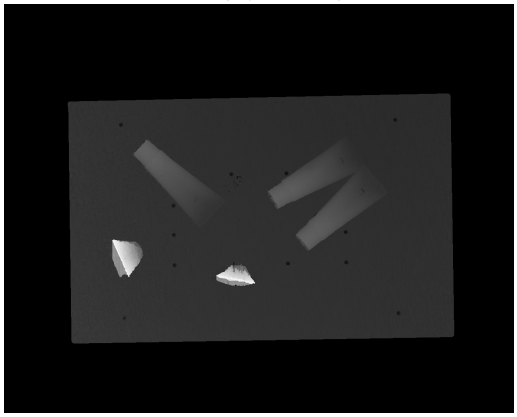
Figure E.19: Grasp synthesis for five tubes of toothpaste.



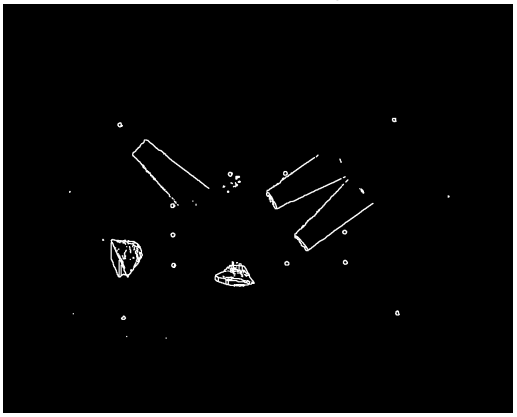
(a) The grey-scale image



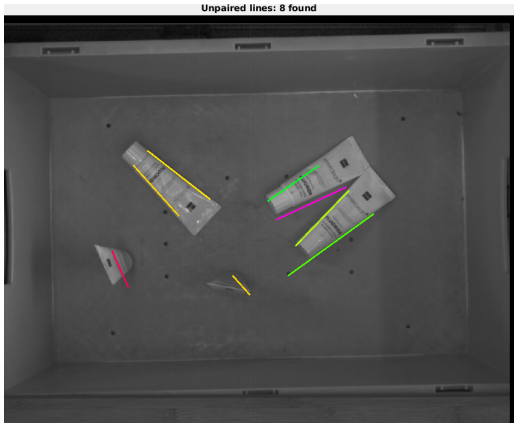
(b) The raw depth image.



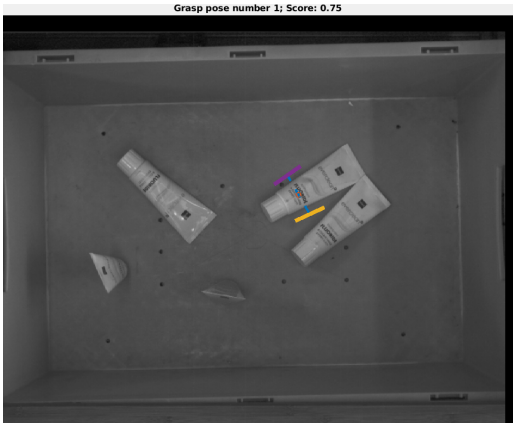
(c) The depth image after filtering and segmentation.



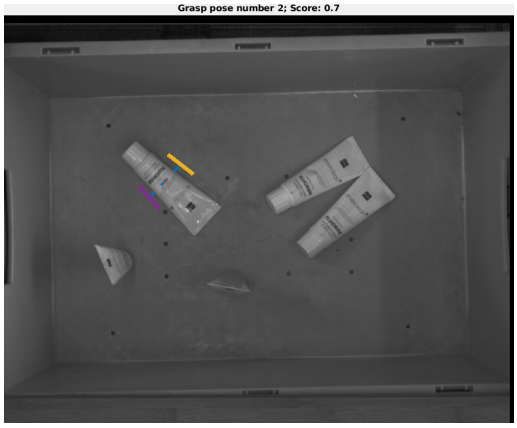
(d) The edges detected in the filtered and segmented depth image.



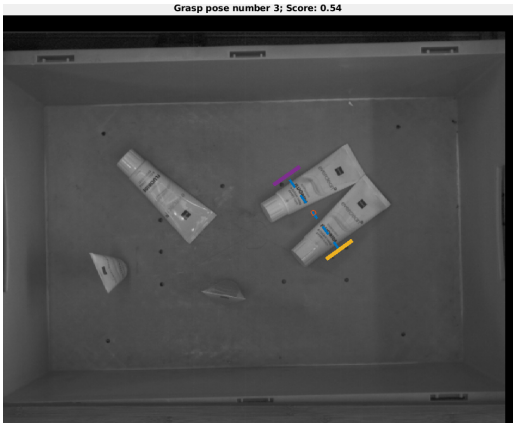
(e) The lines detected from the edges.



(f) One of the synthesized grasps.



(g) One of the synthesized grasps.



(h) One of the synthesized grasps.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1200	6	100

(i) The parameters used for this result.

Figure E.20: Grasp synthesis for five tubes of toothpaste.

E.2. Segmentation Masks

This section presents the segmentations found on the SIR set-up in Veghel, as well as the results for manually mapping the segmentations to the images obtained on the set-up in Eindhoven. The segmentation masks are shown in Figures E.21 through E.24.

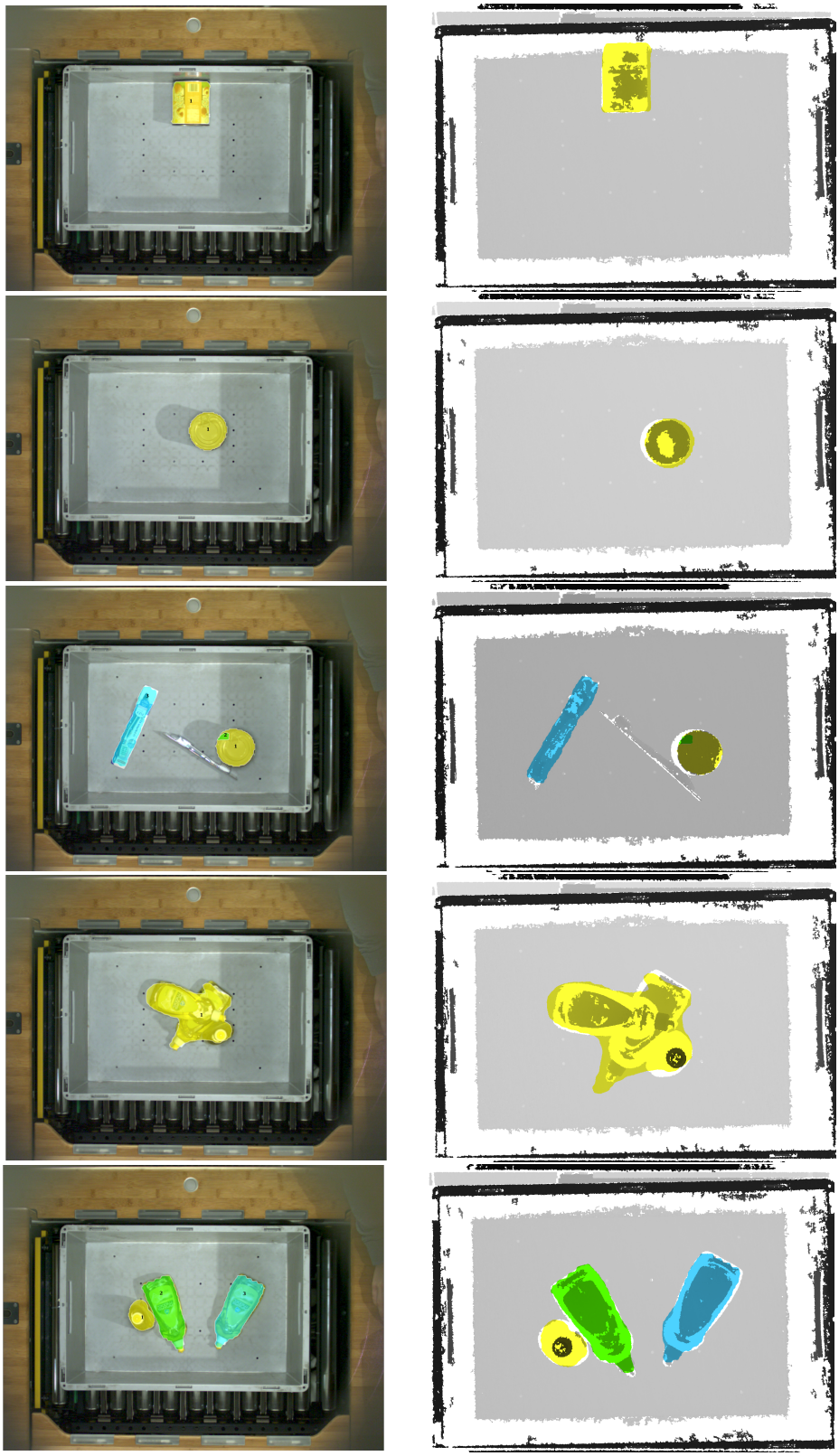


Figure E.21: Results for manually mapping the segmentation masks (1/4).

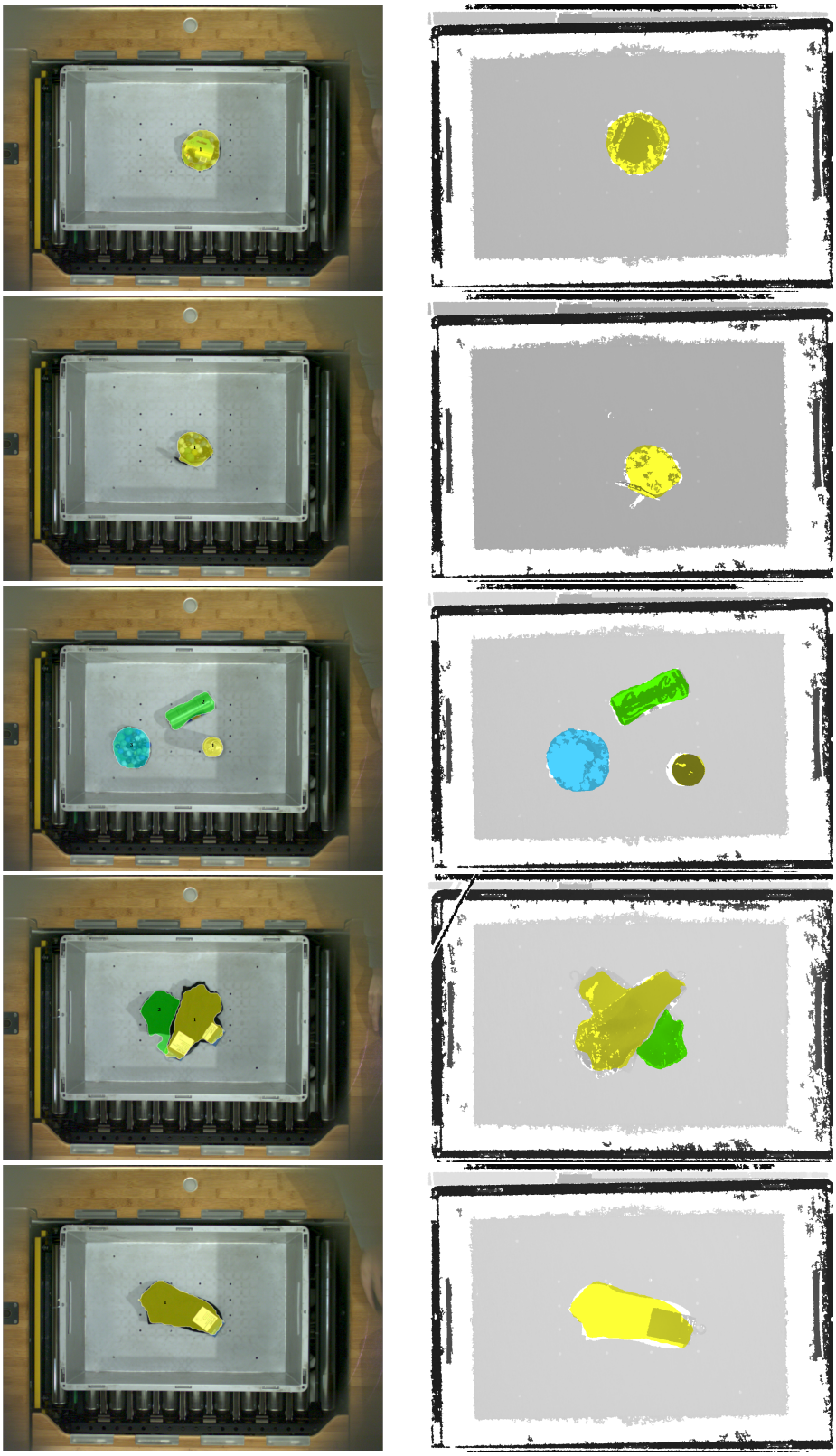


Figure E.22: Results for manually mapping the segmentation masks (2/4).

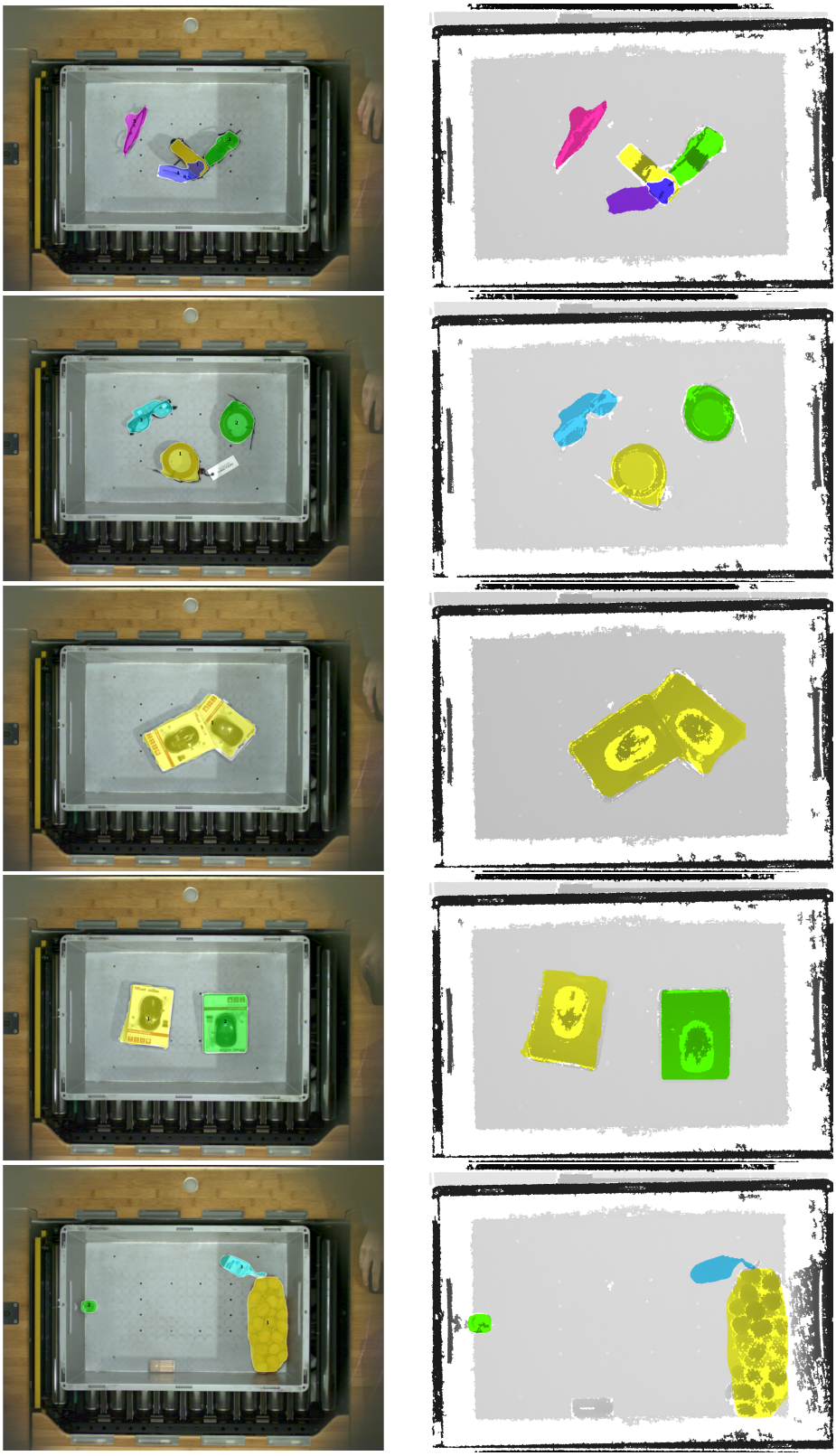


Figure E.23: Results for manually mapping the segmentation masks (3/4).

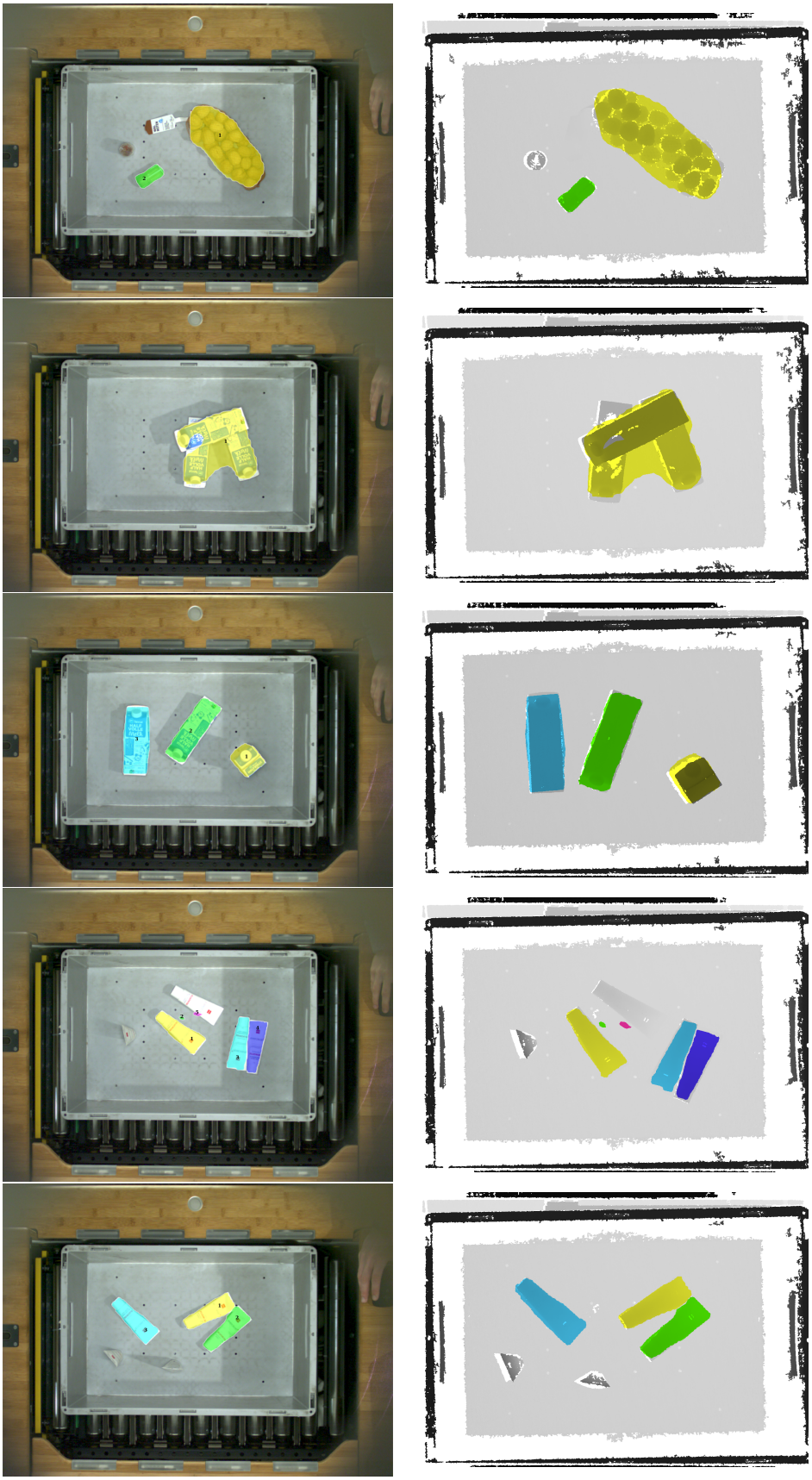
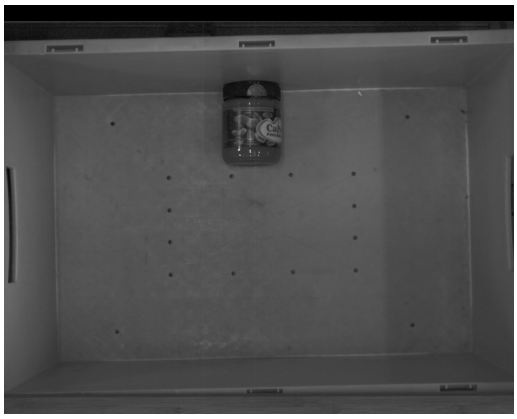


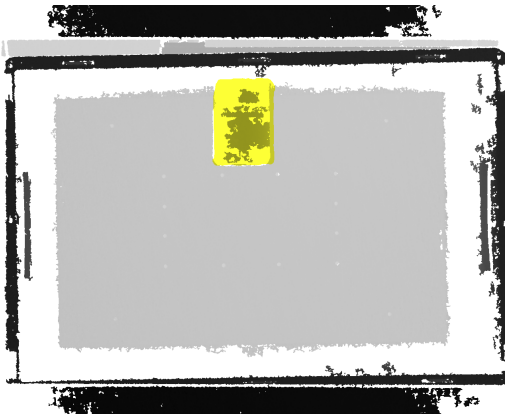
Figure E.24: Results for manually mapping the segmentation masks (4/4).

E.3. Grasps for Partial Segmentations

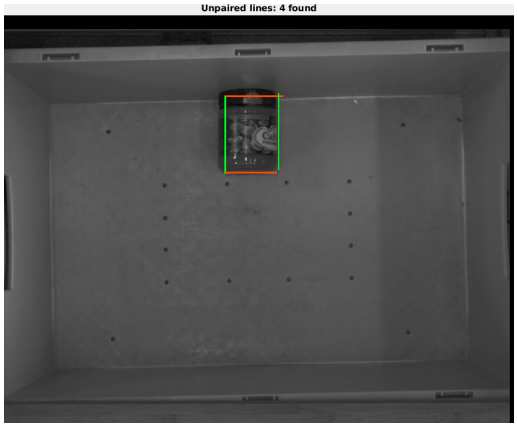
This appendix presents the full results of the synthesized grasps when including segmentation masks. For each test, the following figures are included: the grey-scale image, the segmentation mask visualized on the raw depth image, the detected lines per segment, and the highest-scoring grasp per segment. Additionally, the values of the parameters are included. These can vary between tests, but not within a test. That is, if one test has three segmentations, the same parameters are used for all segments.



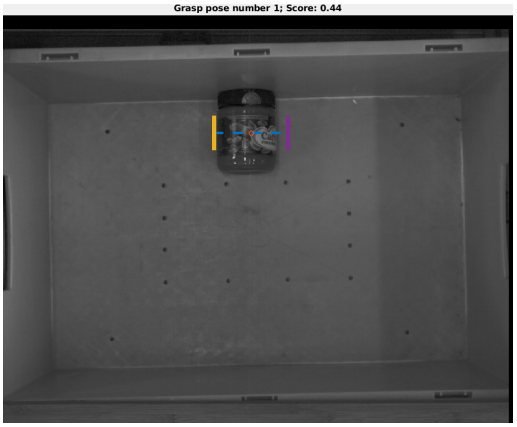
(a) The grey-scale image



(b) The segmentation mask visualised on the raw depth image.



(c) The detected lines for segment 1.

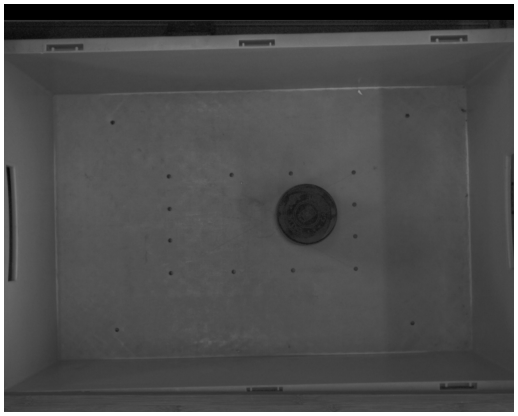


(d) The synthesized grasp for segment 1.

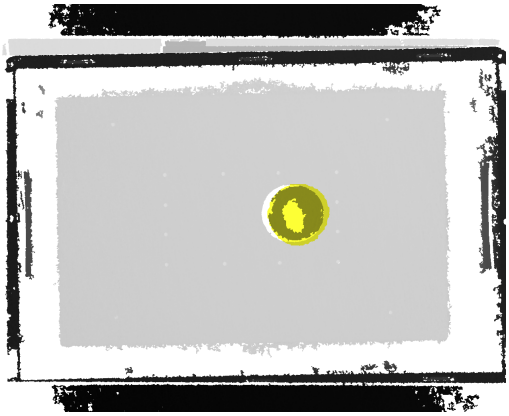
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1000	2	100

(e) The parameters used for this result.

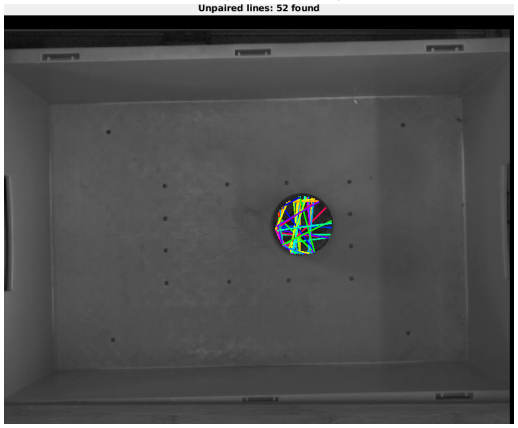
Figure E.25: Grasp synthesis for a peanut butter jar on its side, using segmentations.



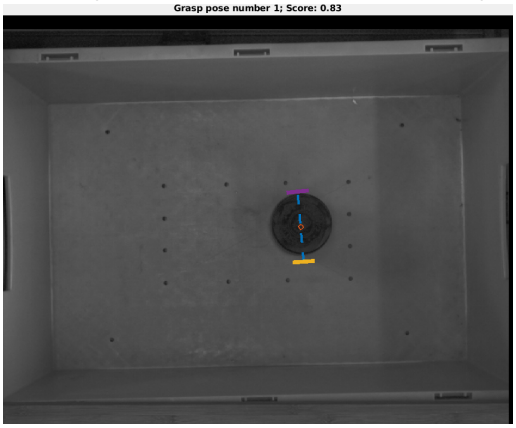
(a) The grey-scale image



(b) The segmentation mask visualised on the raw depth image.



(c) The detected lines for segment 1.

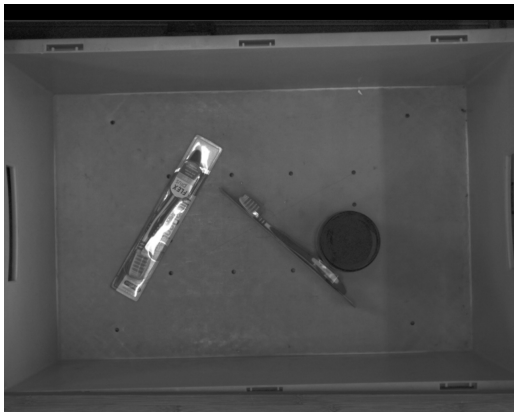


(d) The synthesized grasp for segment 1.

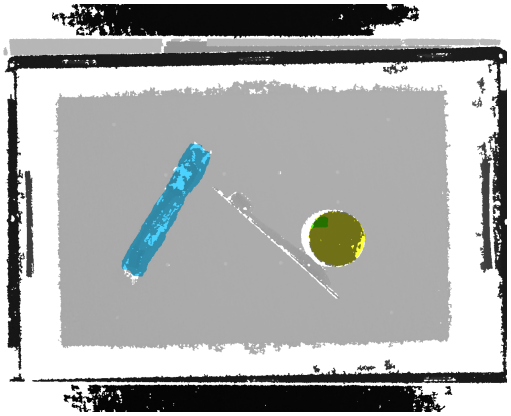
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.0500	2	100

(e) The parameters used for this result.

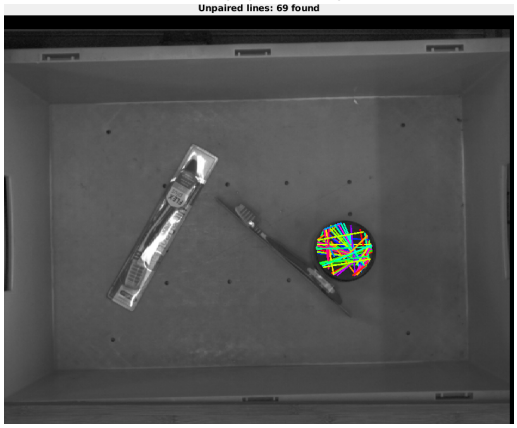
Figure E.26: Grasp synthesis for a peanut butter jar, using segmentations.



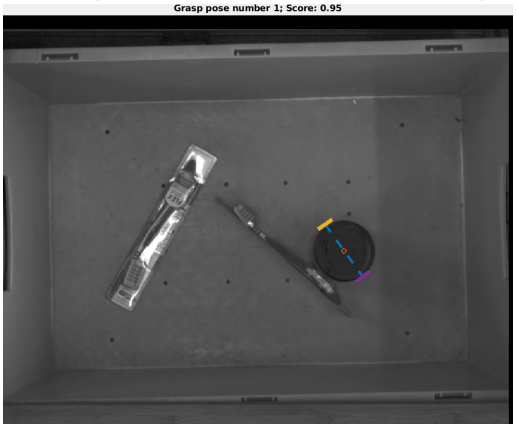
(a) The grey-scale image



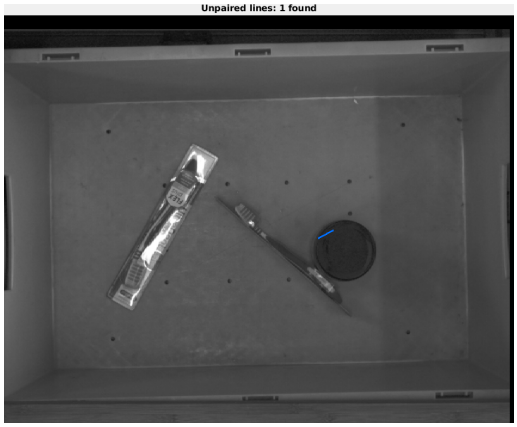
(b) The segmentation mask visualised on the raw depth image.



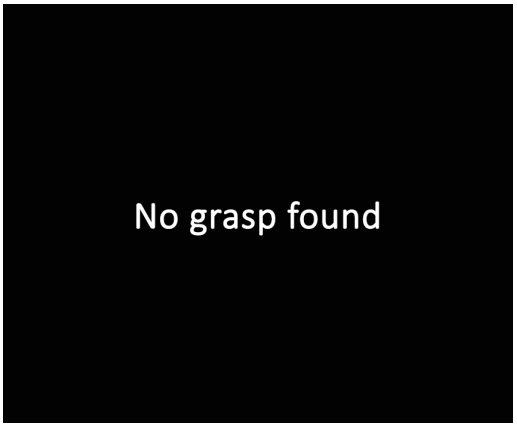
(c) The detected lines for segment 1.



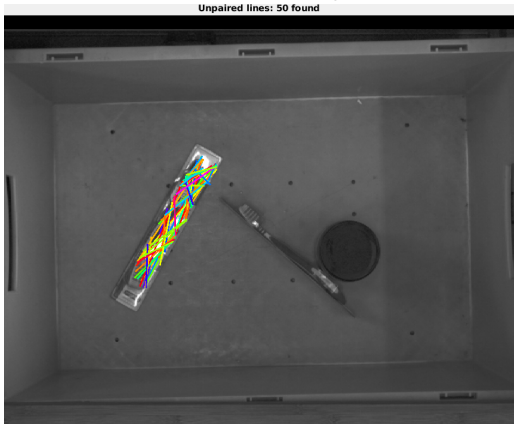
(d) The synthesized grasp for segment 1.



(e) The detected lines for segment 2.



(f) The synthesized grasp for segment 2.



(g) The detected lines for segment 3.



(h) The synthesized grasp for segment 3.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	0.5000	80	3	200	0.0500	2	100

(i) The parameters used for this result.

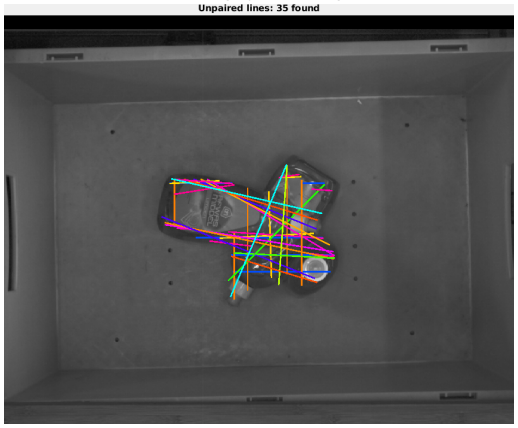
Figure E.27: Grasp synthesis for a peanut butter jar and two toothbrushes, using segmentations.



(a) The grey-scale image



(b) The segmentation mask visualised on the raw depth image.



(c) The detected lines for segment 1.



(d) The synthesized grasp for segment 1.

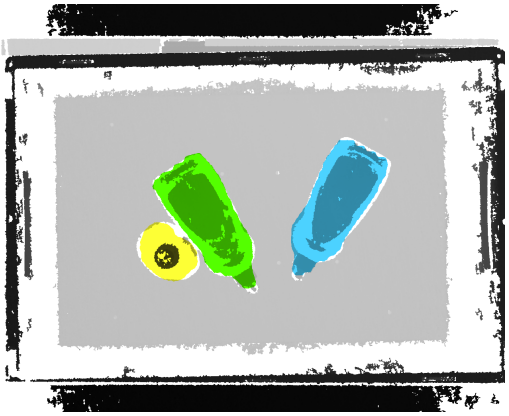
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1.5000	80	3	200	0.1000	2	100

(e) The parameters used for this result.

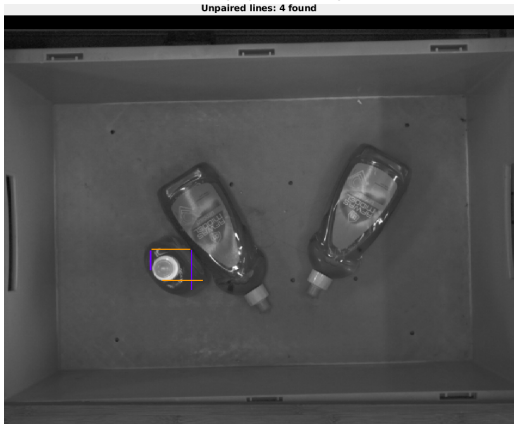
Figure E.28: Grasp synthesis for three bottles of washing up liquid, using segmentations.



(a) The grey-scale image



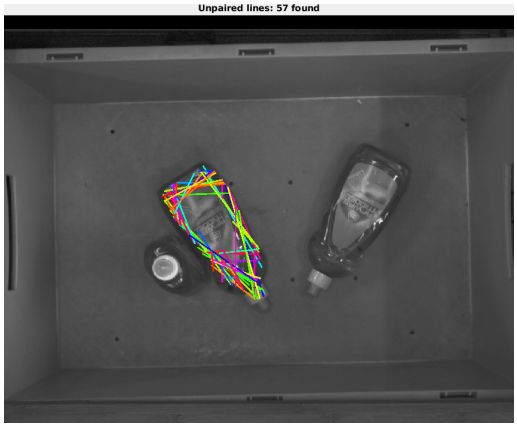
(b) The segmentation mask visualised on the raw depth image.



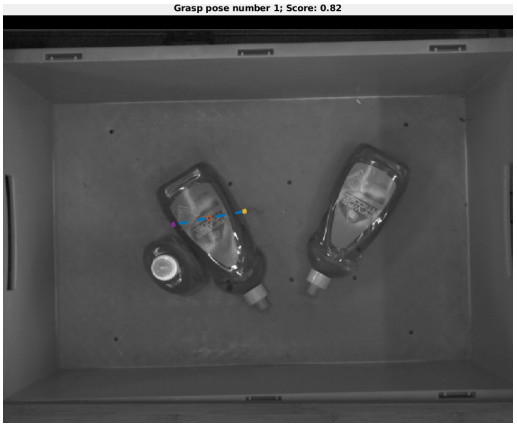
(c) The detected lines for segment 1.



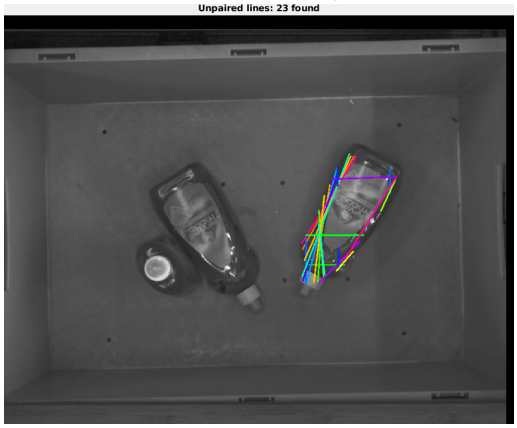
(d) The synthesized grasp for segment 1.



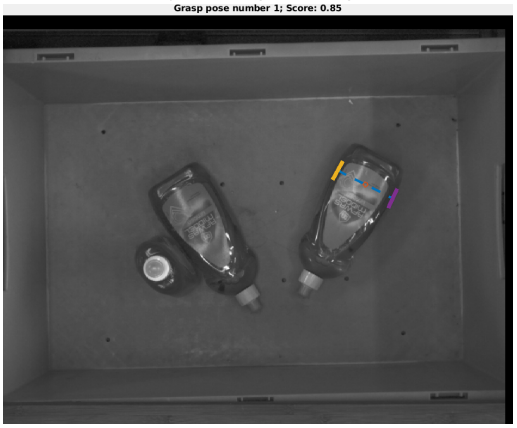
(e) The detected lines for segment 2.



(f) The synthesized grasp for segment 2.



(g) The detected lines for segment 3.

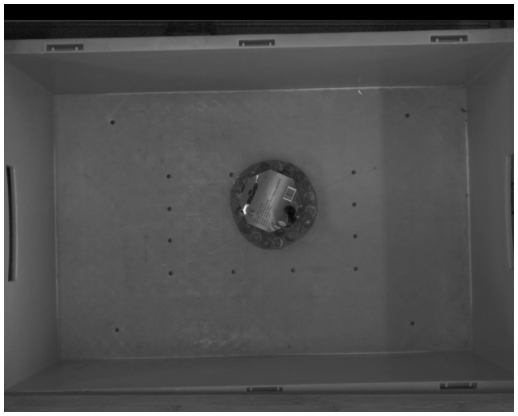


(h) The synthesized grasp for segment 3.

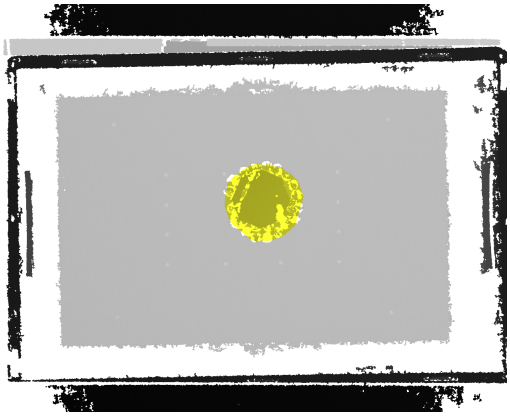
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1.5000	80	3	200	0.0800	2	100

(i) The parameters used for this result.

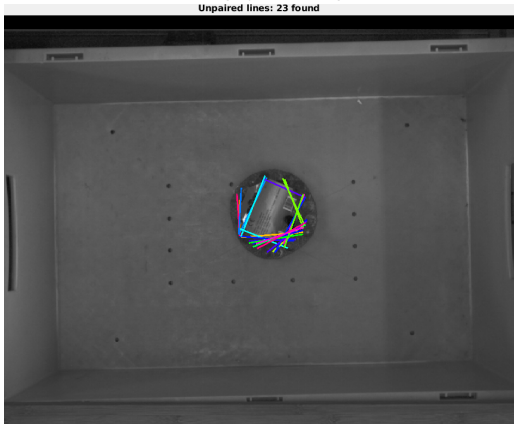
Figure E.29: Grasp synthesis for three bottles of washing up liquid, using segmentations.



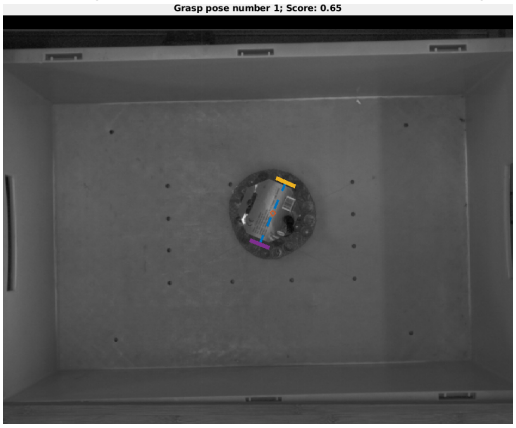
(a) The grey-scale image



(b) The segmentation mask visualised on the raw depth image.



(c) The detected lines for segment 1.

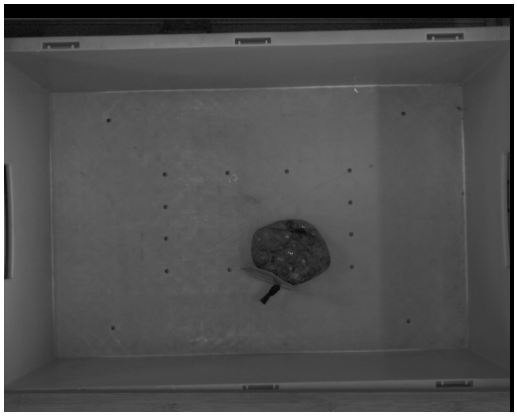


(d) The synthesized grasp for segment 1.

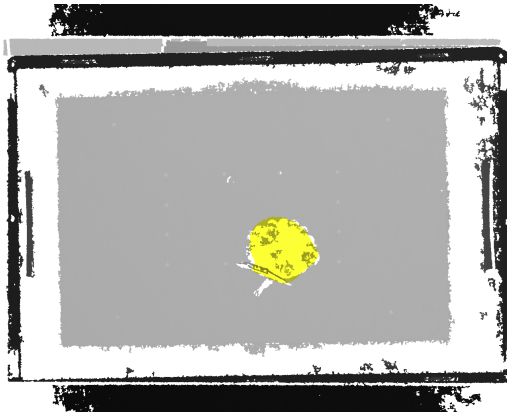
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.0500	2	100

(e) The parameters used for this result.

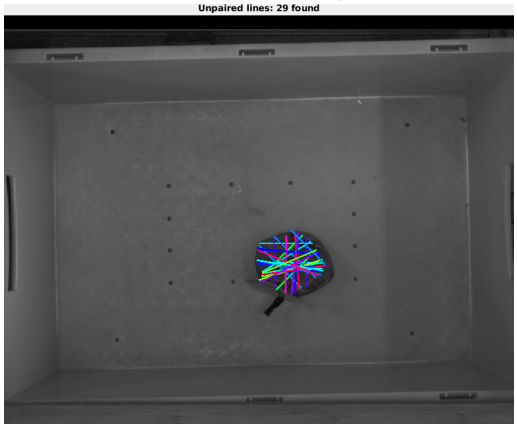
Figure E.30: Grasp synthesis for a net of marbles, using segmentations.



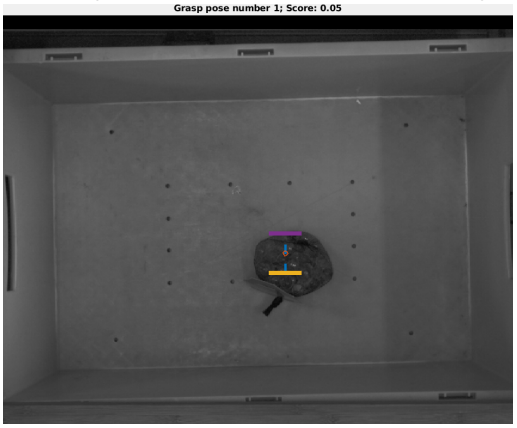
(a) The grey-scale image



(b) The segmentation mask visualised on the raw depth image.



(c) The detected lines for segment 1.



(d) The synthesized grasp for segment 1.

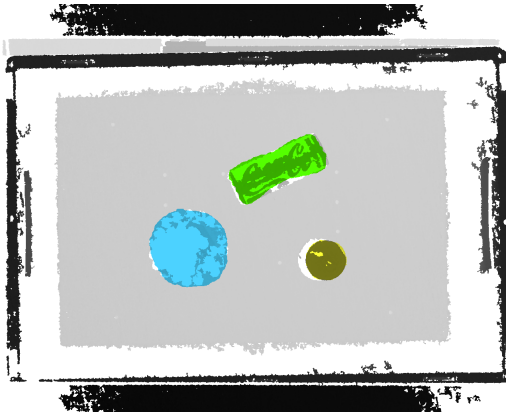
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	0.5000	80	3	200	0.0800	2	100

(e) The parameters used for this result.

Figure E.31: Grasp synthesis for a net of marbles, using segmentations.



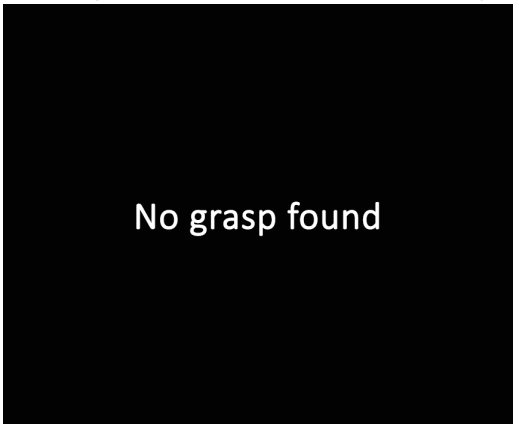
(a) The grey-scale image



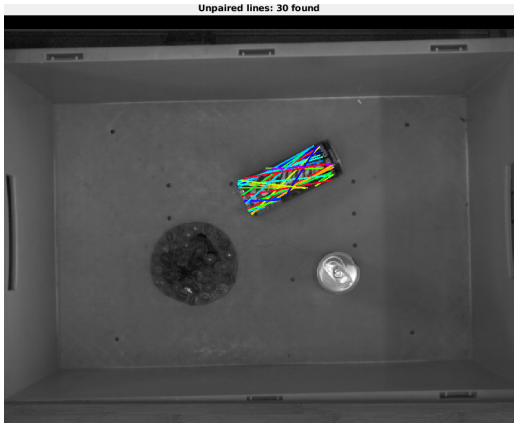
(b) The segmentation mask visualised on the raw depth image.



(c) The detected lines for segment 1.



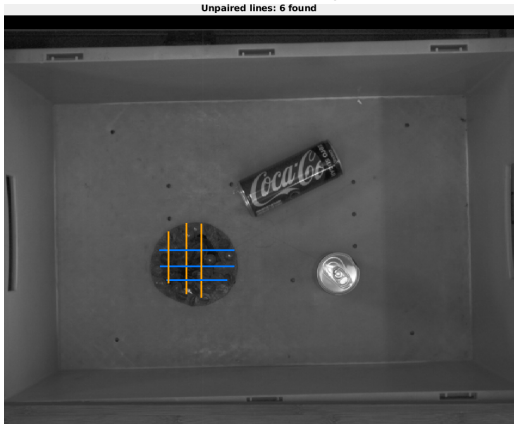
(d) The synthesized grasp for segment 1.



(e) The detected lines for segment 2.



(f) The synthesized grasp for segment 2.



(g) The detected lines for segment 3.

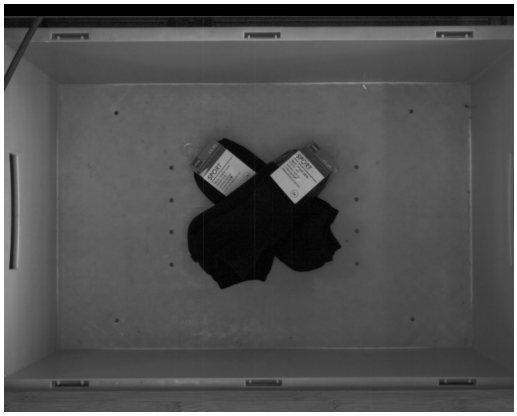


(h) The synthesized grasp for segment 3.

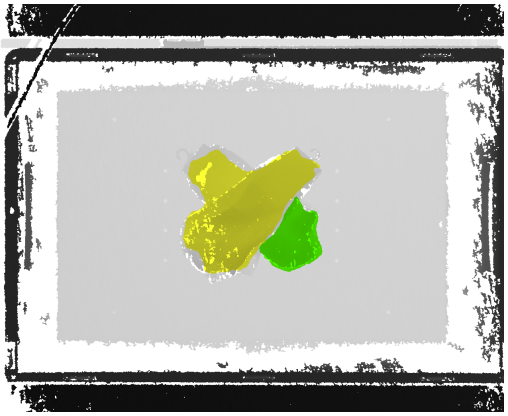
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.0800	2	100

(i) The parameters used for this result.

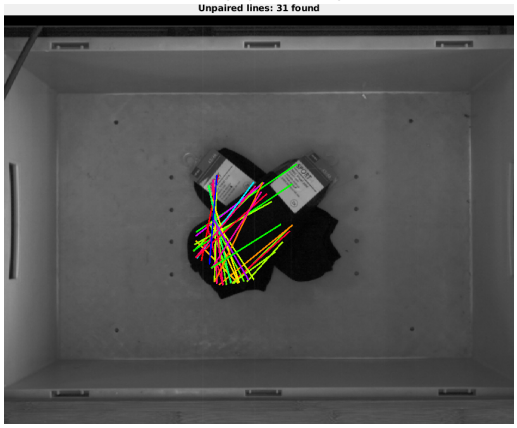
Figure E.32: Grasp synthesis for a net of marbles and two tin cans, using segmentations.



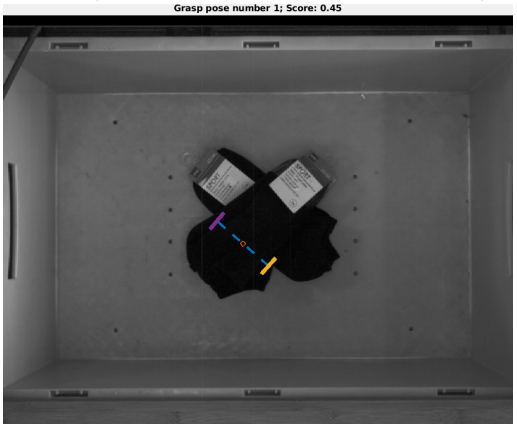
(a) The grey-scale image



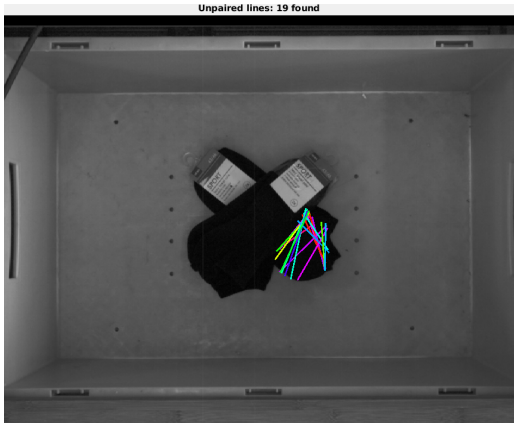
(b) The segmentation mask visualised on the raw depth image.



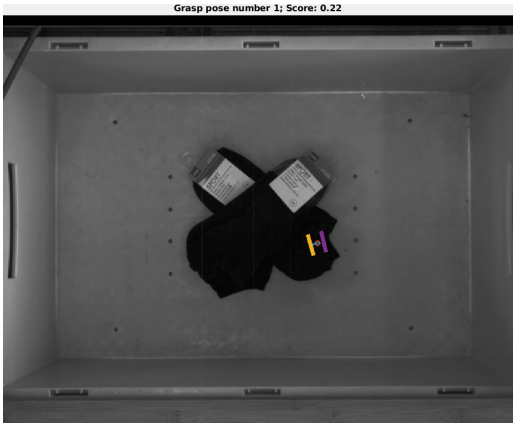
(c) The detected lines for segment 1.



(d) The synthesized grasp for segment 1.



(e) The detected lines for segment 2.



(f) The synthesized grasp for segment 2.

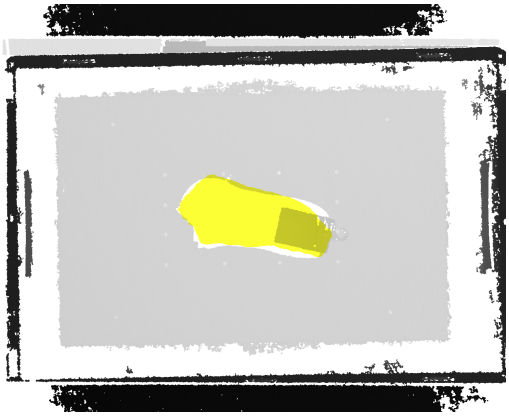
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1000	2	100

(g) The parameters used for this result.

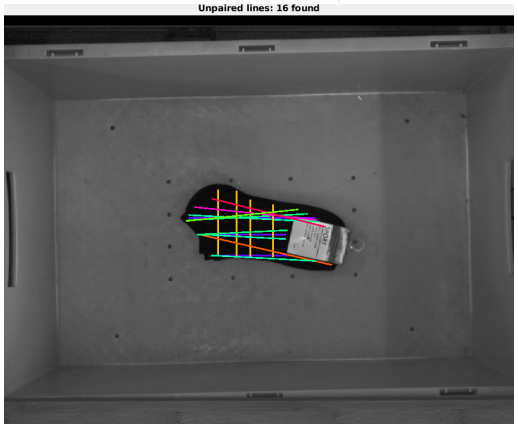
Figure E.33: Grasp synthesis for two sets of socks, using segmentations.



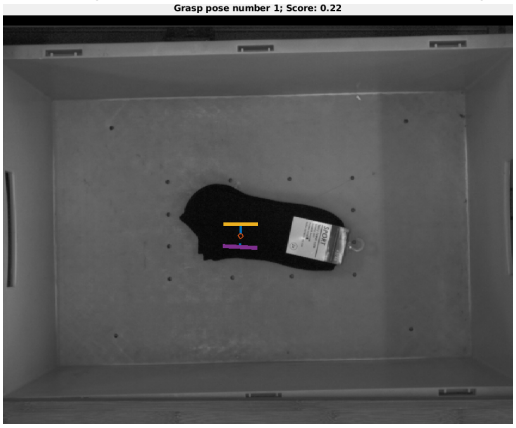
(a) The grey-scale image



(b) The segmentation mask visualised on the raw depth image.



(c) The detected lines for segment 1.

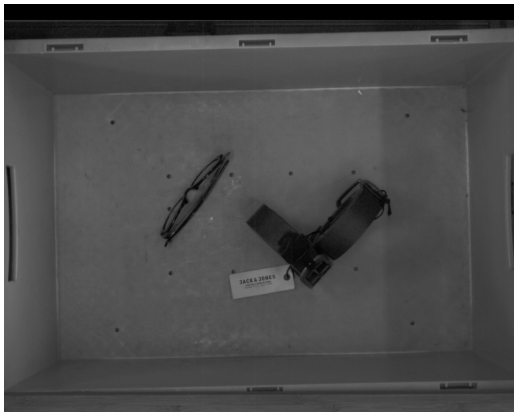


(d) The synthesized grasp for segment 1.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1000	2	100

(e) The parameters used for this result.

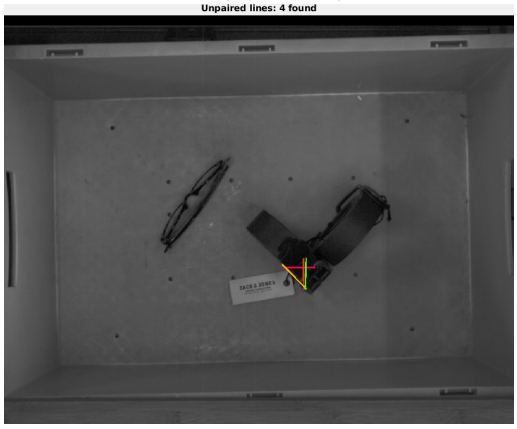
Figure E.34: Grasp synthesis for a set of socks, using segmentations.



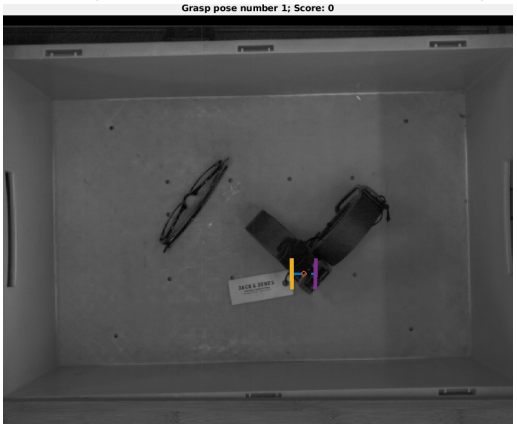
(a) The grey-scale image



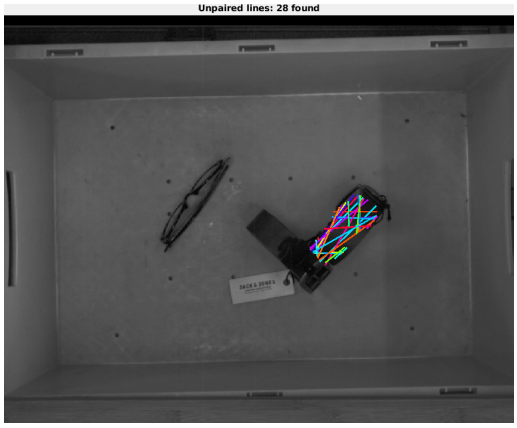
(b) The segmentation mask visualised on the raw depth image.



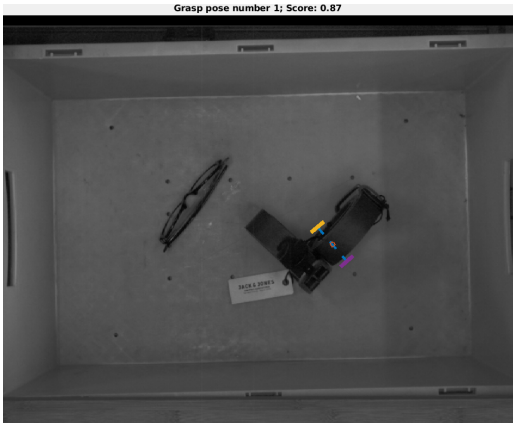
(c) The detected lines for segment 1.



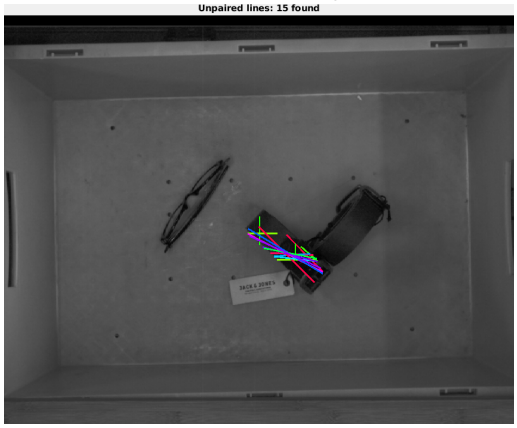
(d) The synthesized grasp for segment 1.



(e) The detected lines for segment 2.



(f) The synthesized grasp for segment 2.



(g) The detected lines for segment 3.

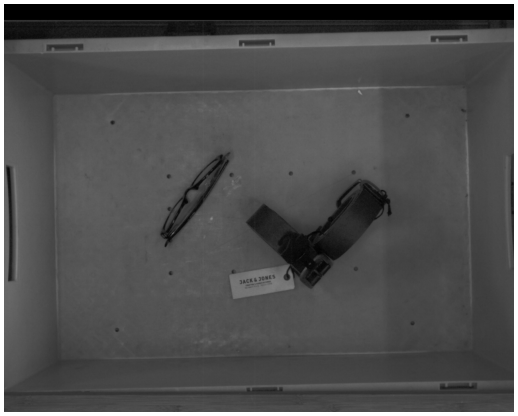


(h) The synthesized grasp for segment 3.

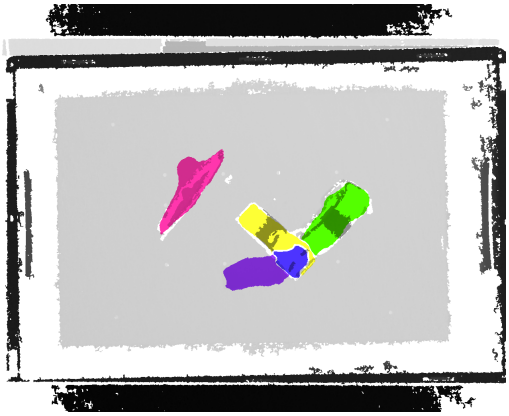
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1000	2	100

(i) The parameters used for this result.

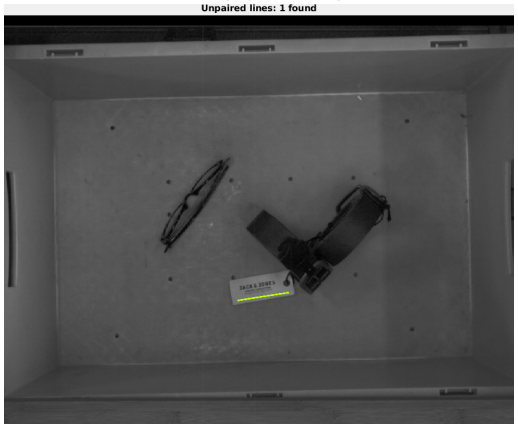
Figure E.35: Grasp synthesis for two belts and a pair of glasses, using segmentations.



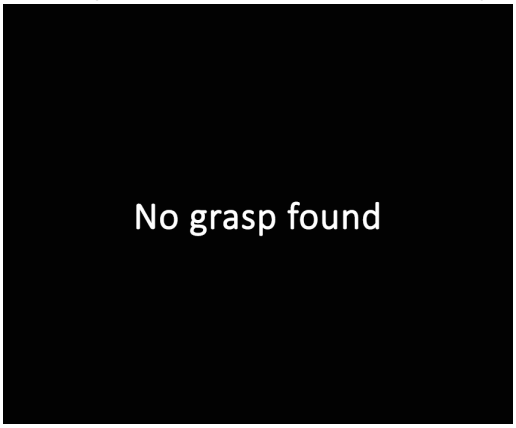
(a) The grey-scale image



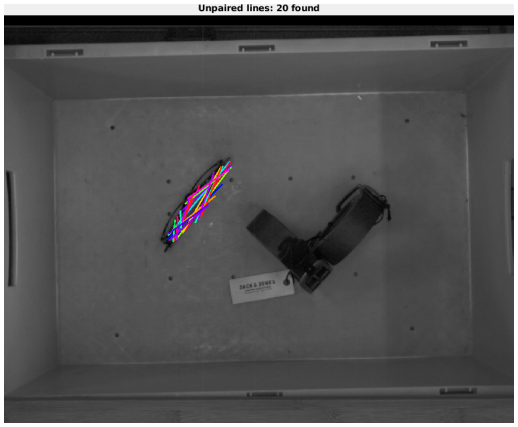
(b) The segmentation mask visualised on the raw depth image.



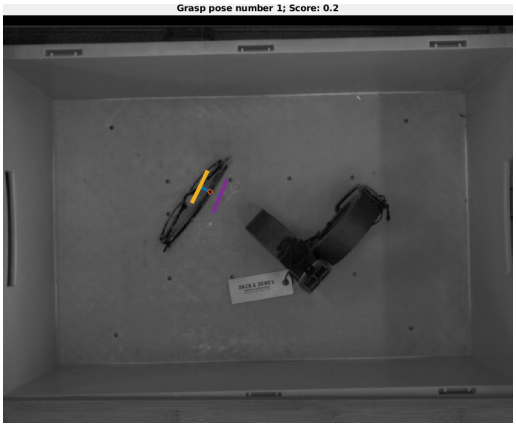
(c) The detected lines for segment 4.



(d) The synthesized grasp for segment 4.



(e) The detected lines for segment 5.



(f) The synthesized grasp for segment 5.

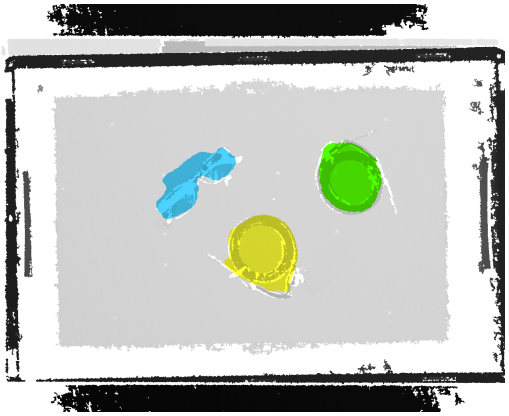
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1000	2	100

(g) The parameters used for this result.

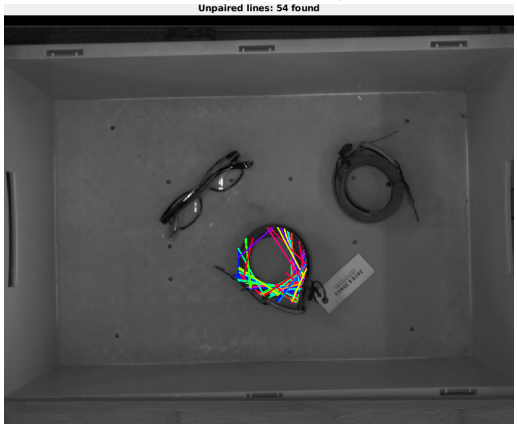
Figure E.36: Grasp synthesis for two belts and a pair of glasses, using segmentations (continued).



(a) The grey-scale image



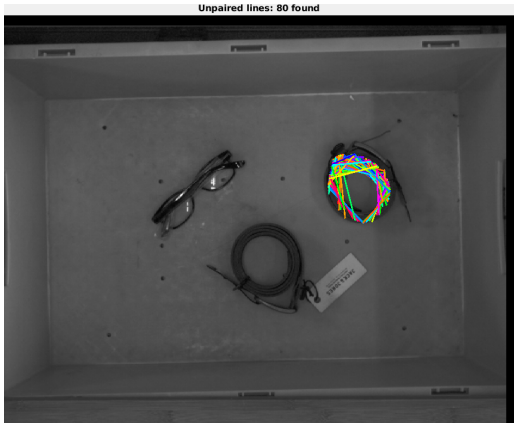
(b) The segmentation mask visualised on the raw depth image.



(c) The detected lines for segment 1.



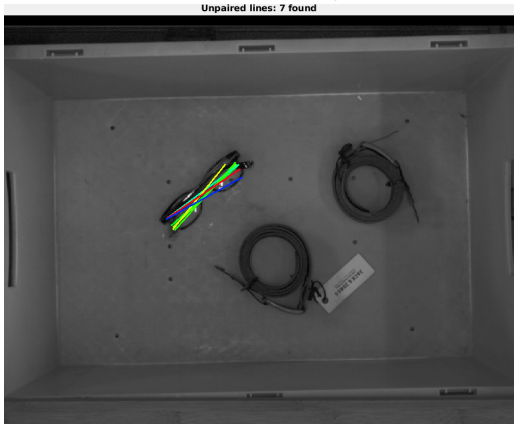
(d) The synthesized grasp for segment 1.



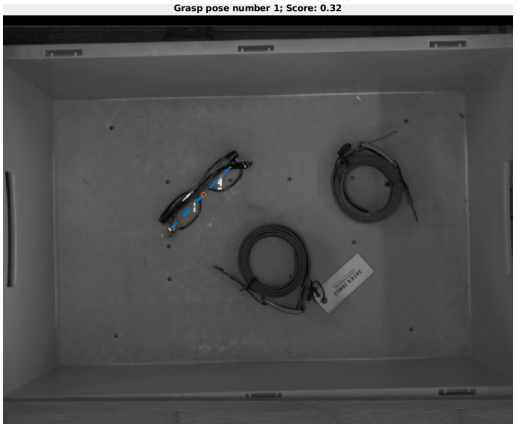
(e) The detected lines for segment 2.



(f) The synthesized grasp for segment 2.



(g) The detected lines for segment 3.

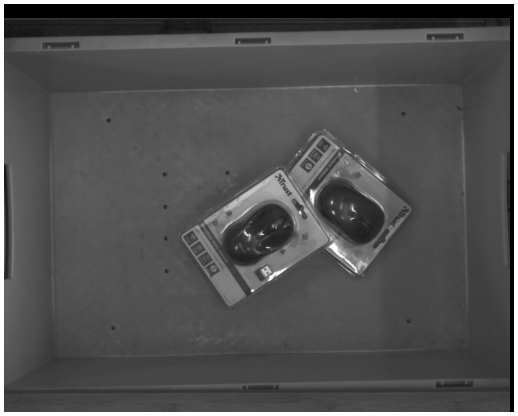


(h) The synthesized grasp for segment 3.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.0900	2	100

(i) The parameters used for this result.

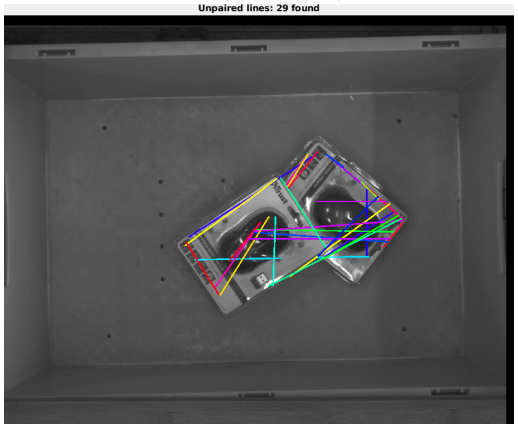
Figure E.37: Grasp synthesis for two belts and a pair of glasses, using segmentations.



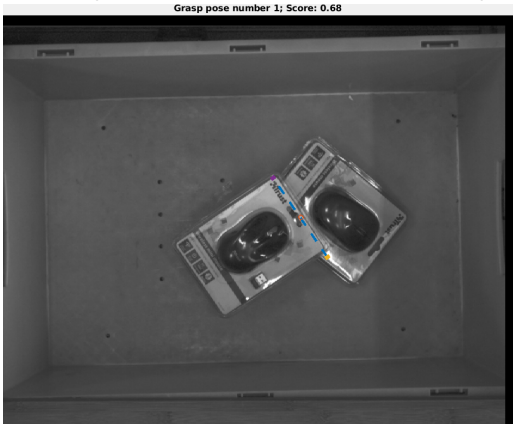
(a) The grey-scale image



(b) The segmentation mask visualised on the raw depth image.



(c) The detected lines for segment 1.



(d) The synthesized grasp for segment 1.

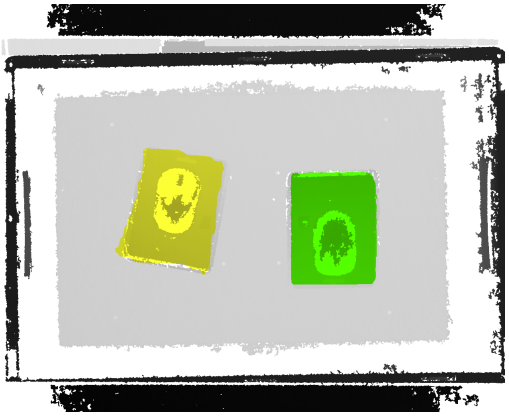
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1000	2	100

(e) The parameters used for this result.

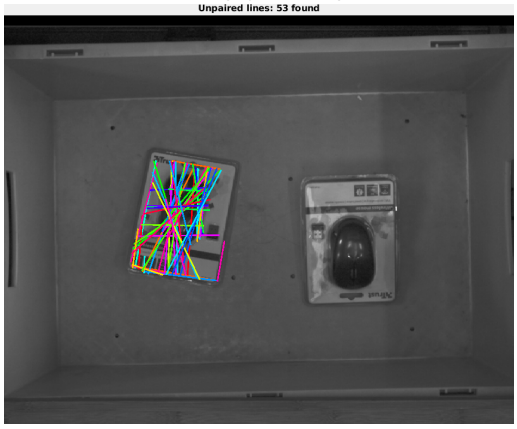
Figure E.38: Grasp synthesis for two computer mice that overlap, using segmentations.



(a) The grey-scale image



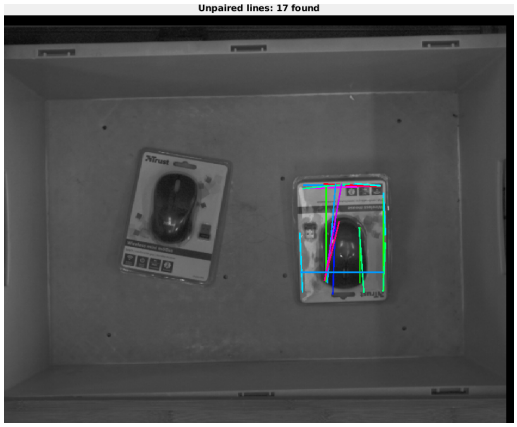
(b) The segmentation mask visualised on the raw depth image.



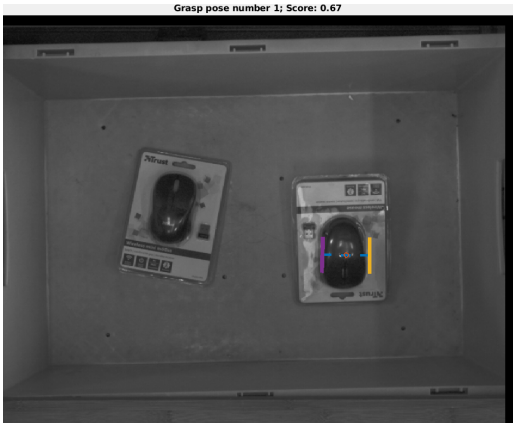
(c) The detected lines for segment 1.



(d) The synthesized grasp for segment 1.



(e) The detected lines for segment 2.

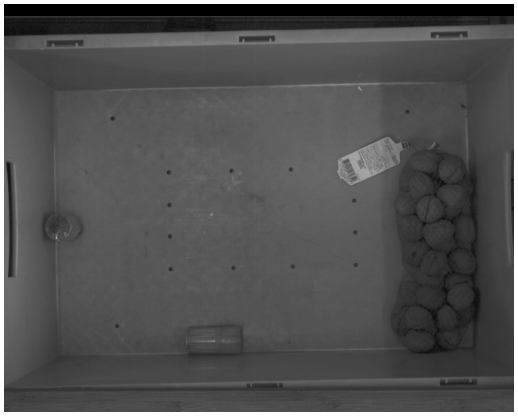


(f) The synthesized grasp for segment 2.

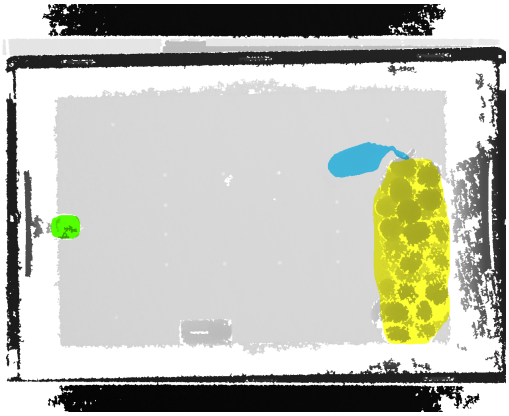
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1000	2	100

(g) The parameters used for this result.

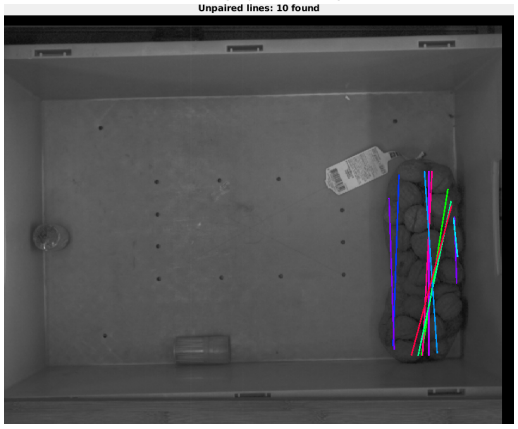
Figure E.39: Grasp synthesis for two computer mice, using segmentations.



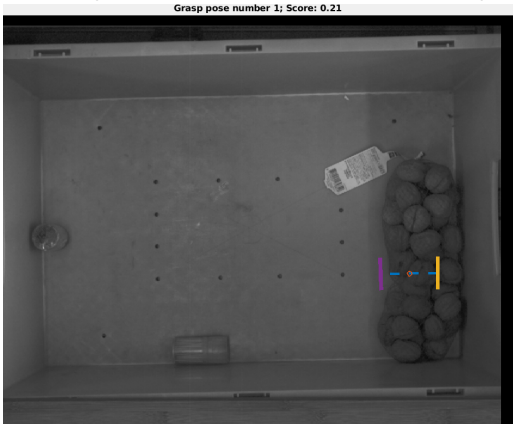
(a) The grey-scale image



(b) The segmentation mask visualised on the raw depth image.



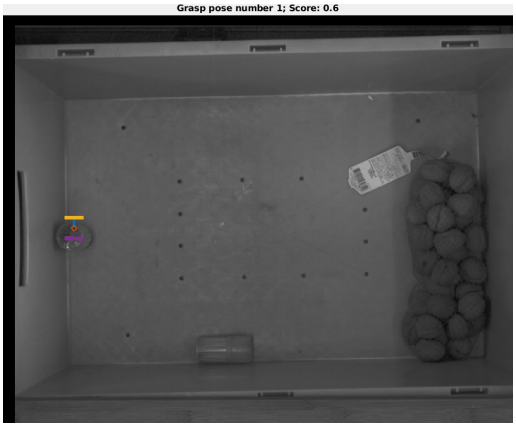
(c) The detected lines for segment 1.



(d) The synthesized grasp for segment 1.



(e) The detected lines for segment 2.



(f) The synthesized grasp for segment 2.

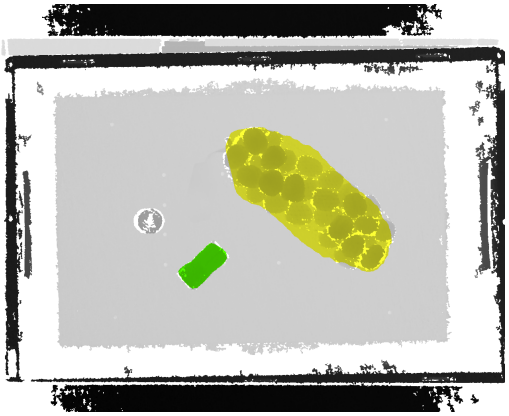
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1200	2	100

(g) The parameters used for this result.

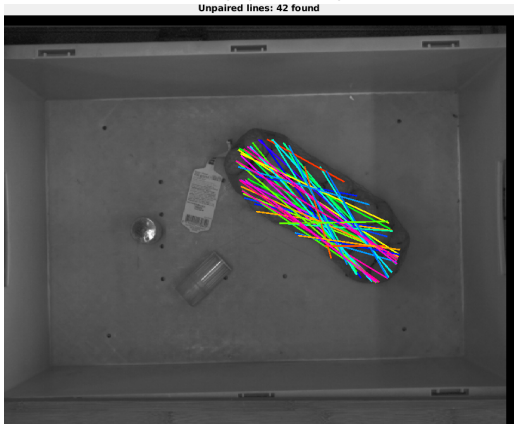
Figure E.40: Grasp synthesis for a net of walnuts and two packs of toothpicks, all at the edge of the carrier, using segmentations.



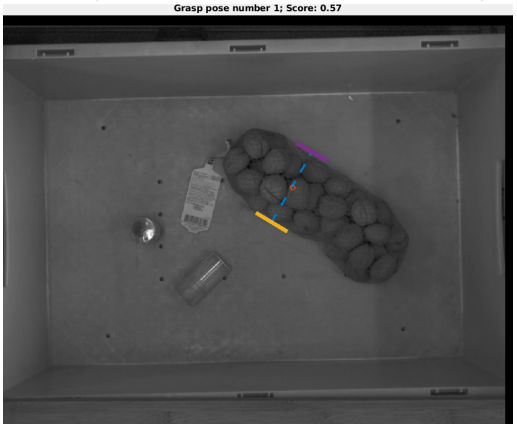
(a) The grey-scale image



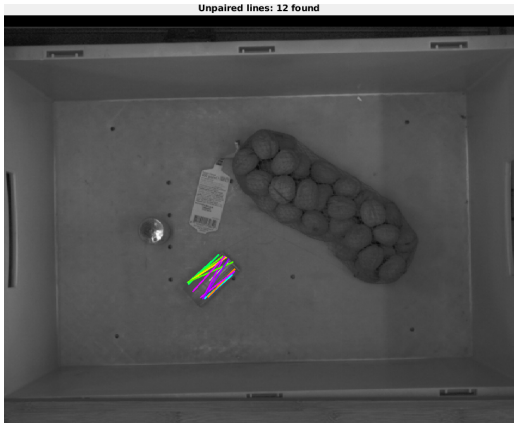
(b) The segmentation mask visualised on the raw depth image.



(c) The detected lines for segment 1.



(d) The synthesized grasp for segment 1.



(e) The detected lines for segment 2.

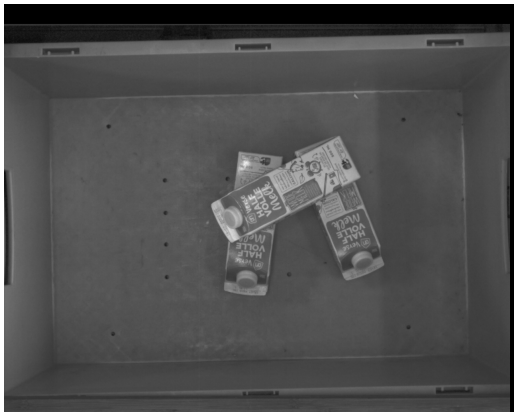


(f) The synthesized grasp for segment 2.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1000	2	100

(g) The parameters used for this result.

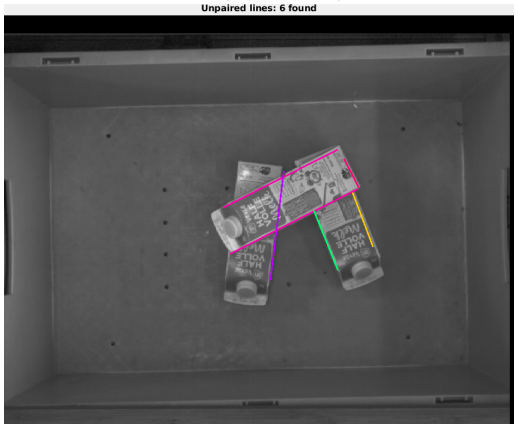
Figure E.41: Grasp synthesis for a net of walnuts and two packs of toothpicks, using segmentations.



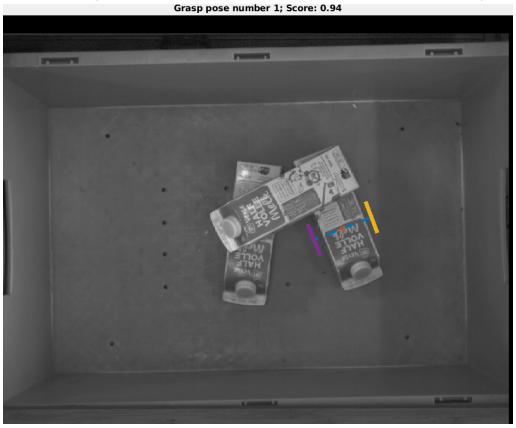
(a) The grey-scale image



(b) The segmentation mask visualised on the raw depth image.



(c) The detected lines for segment 1.



(d) The synthesized grasp for segment 1.

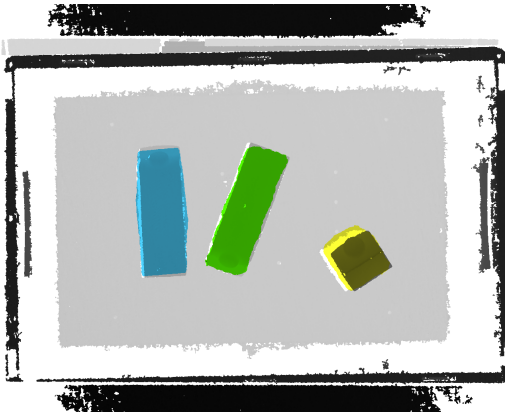
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1200	2	100

(e) The parameters used for this result.

Figure E.42: Grasp synthesis for three cartons of milk that overlap, using segmentations.



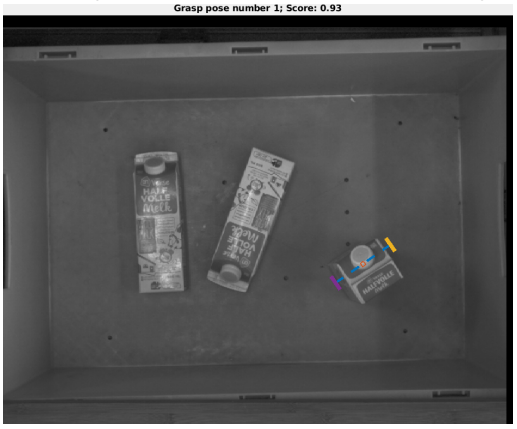
(a) The grey-scale image



(b) The segmentation mask visualised on the raw depth image.



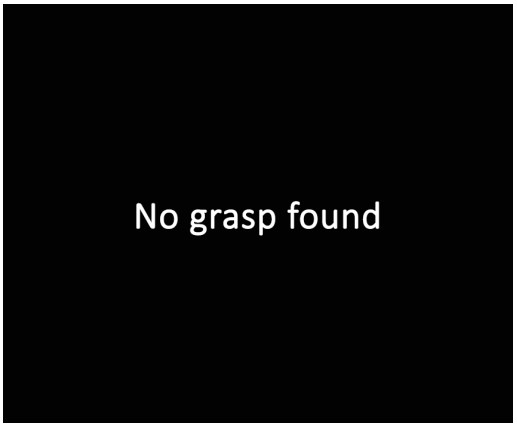
(c) The detected lines for segment 1.



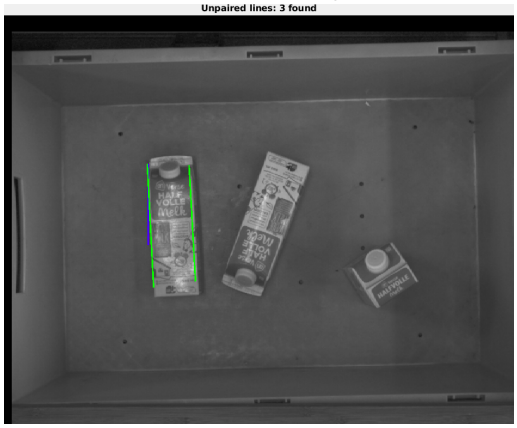
(d) The synthesized grasp for segment 1.



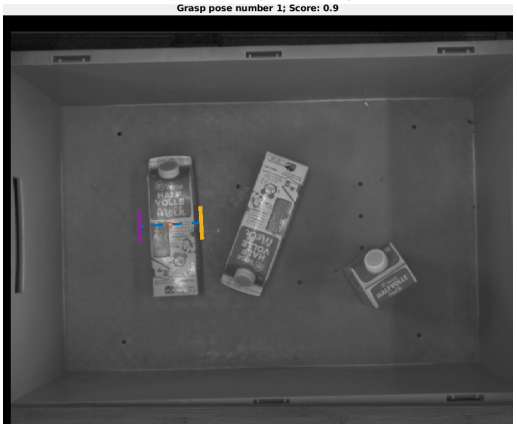
(e) The detected lines for segment 2.



(f) The synthesized grasp for segment 2.



(g) The detected lines for segment 3.

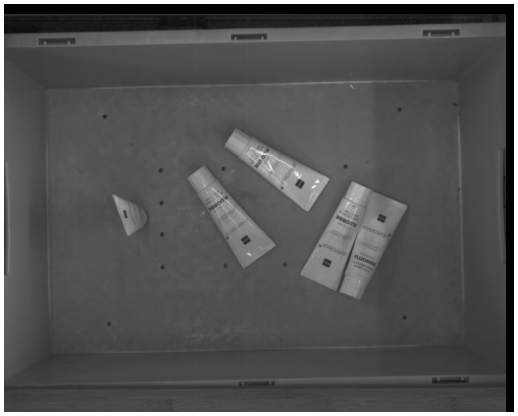


(h) The synthesized grasp for segment 3.

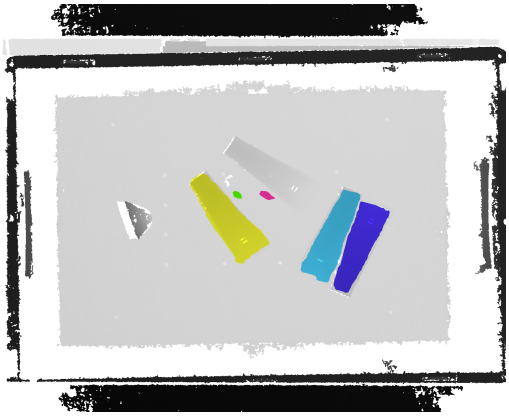
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1200	2	100

(i) The parameters used for this result.

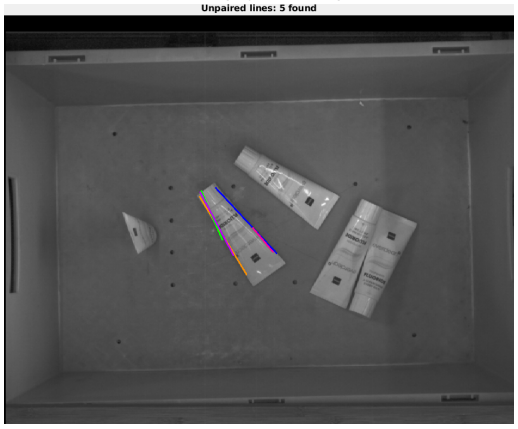
Figure E.43: Grasp synthesis for three cartons of milk, using segmentations.



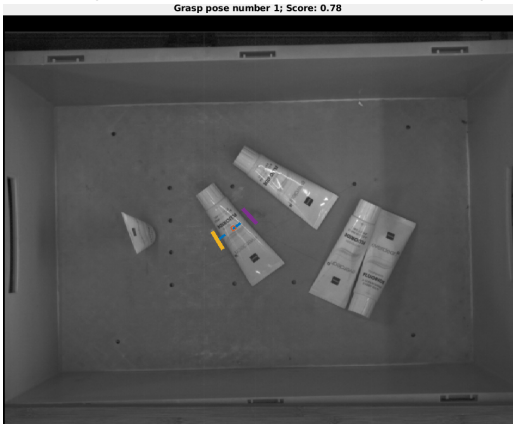
(a) The grey-scale image



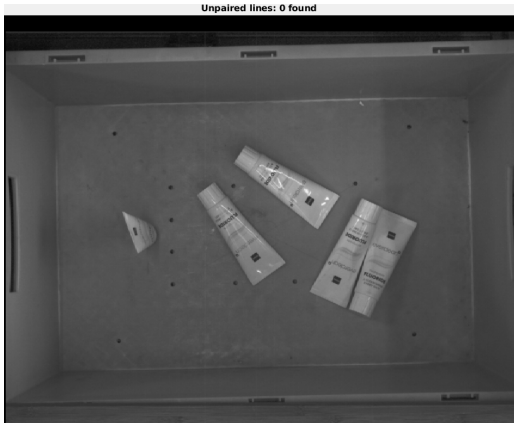
(b) The segmentation mask visualised on the raw depth image.



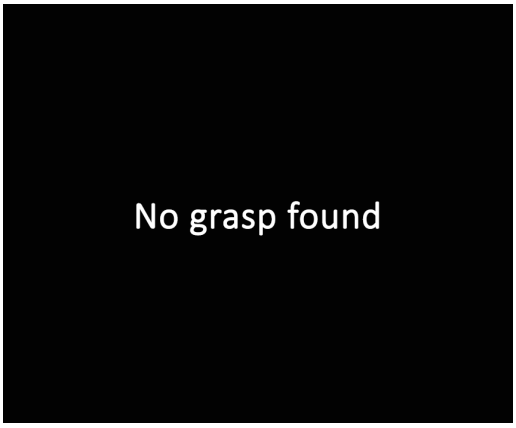
(c) The detected lines for segment 1.



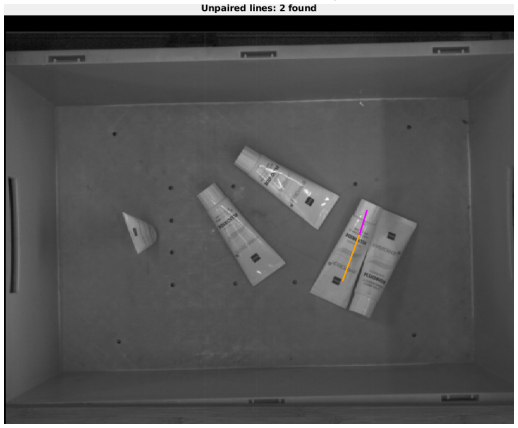
(d) The synthesized grasp for segment 1.



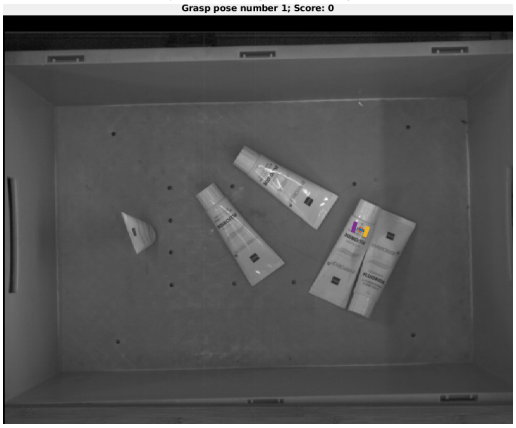
(e) The detected lines for segment 2.



(f) The synthesized grasp for segment 2.



(g) The detected lines for segment 3.

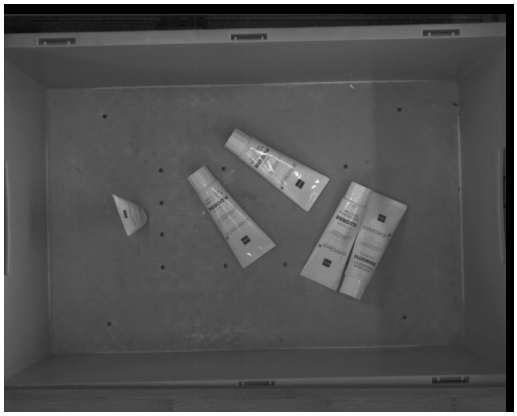


(h) The synthesized grasp for segment 3.

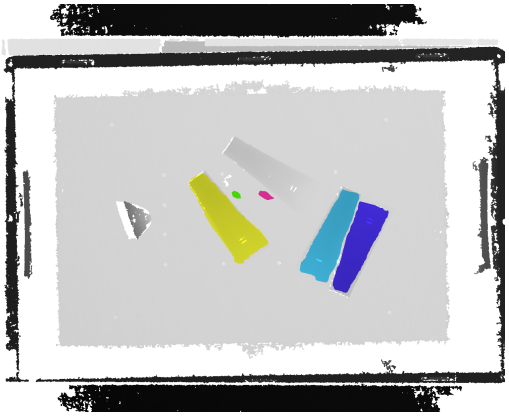
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1200	6	100

(i) The parameters used for this result.

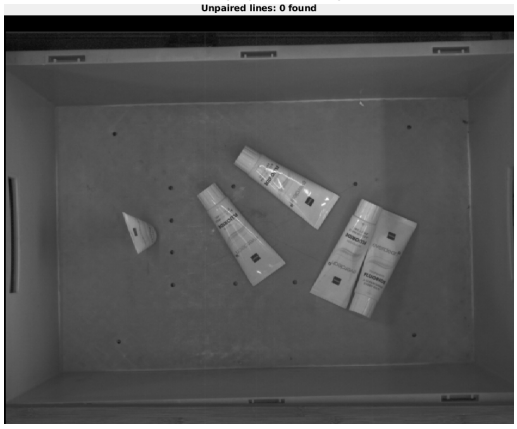
Figure E.44: Grasp synthesis for five tubes of toothpaste, using segmentations.



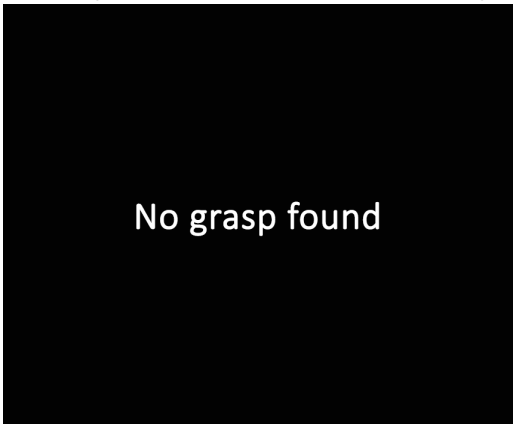
(a) The grey-scale image



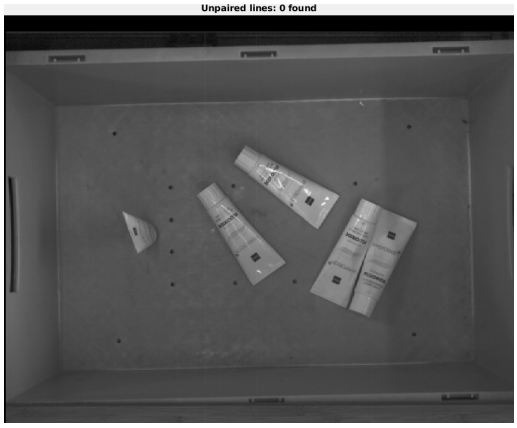
(b) The segmentation mask visualised on the raw depth image.



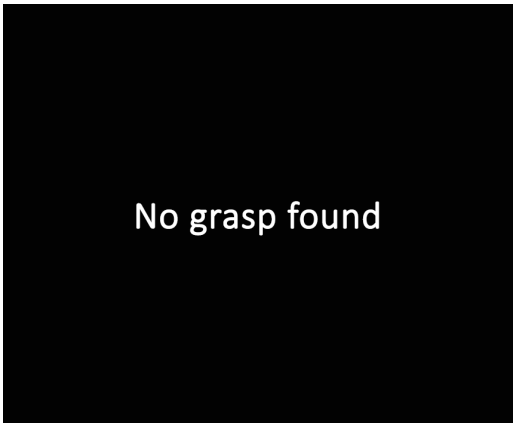
(c) The detected lines for segment 4.



(d) The synthesized grasp for segment 4.



(e) The detected lines for segment 5.

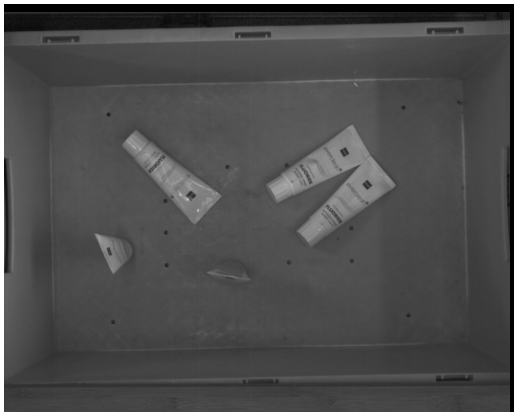


(f) The synthesized grasp for segment 5.

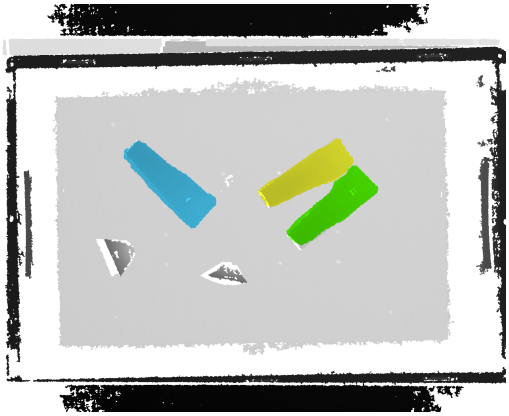
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1200	6	100

(g) The parameters used for this result.

Figure E.45: Grasp synthesis for five tubes of toothpaste, using segmentations (continued).



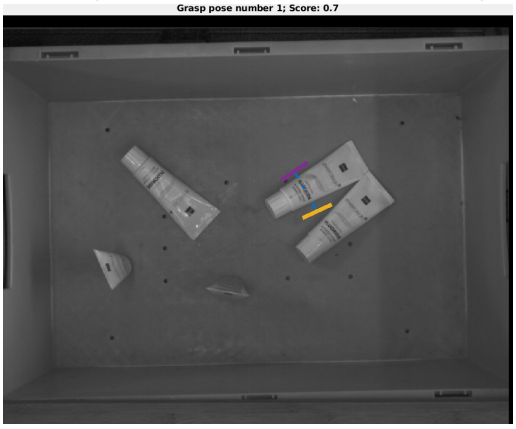
(a) The grey-scale image



(b) The segmentation mask visualised on the raw depth image.



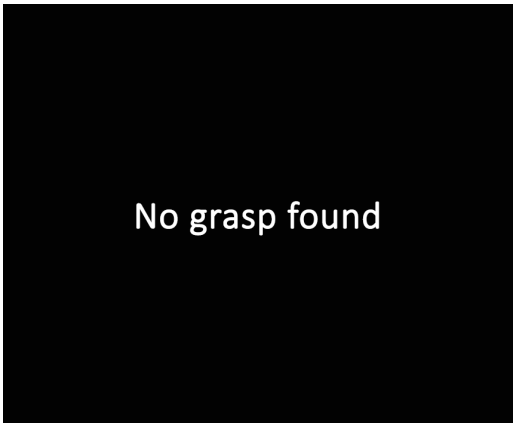
(c) The detected lines for segment 1.



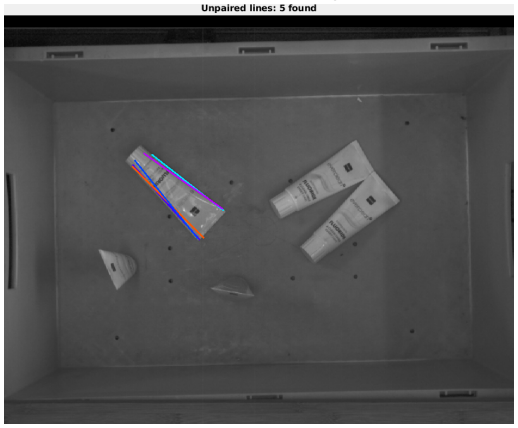
(d) The synthesized grasp for segment 1.



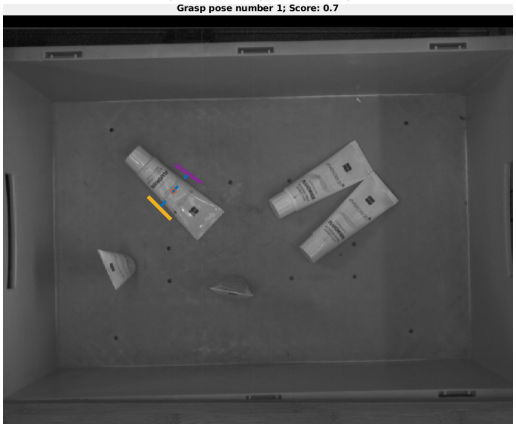
(e) The detected lines for segment 2.



(f) The synthesized grasp for segment 2.



(g) The detected lines for segment 3.



(h) The synthesized grasp for segment 3.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	5	1	80	3	200	0.1200	6	100

(i) The parameters used for this result.

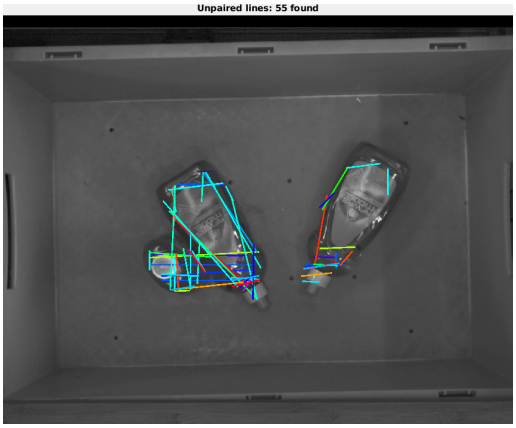
Figure E.46: Grasp synthesis for five tubes of toothpaste, using segmentations.

E.4. Grasps for Complete Segmentations

This appendix presents the full results of the synthesized grasps when including segmentation masks. For each test, the following figures are included: the grey-scale image, the segmentation mask visualized on the raw depth image, the detected lines per segment, and the highest-scoring grasp per segment. Additionally, the values of the parameters are included. These can vary between tests, but not within a test. That is, if one test has three segmentations, the same parameters are used for all segments.



(a) The grey-scale image.



(b) The filtered depth image using the segmentation mask.



(c) One of the synthesized grasps using the mask.



(d) One of the synthesized grasps without using the mask..



(e) One of the synthesized grasps using the mask.



(f) One of the synthesized grasps without using the mask.



(g) One of the synthesized grasps using the mask.



(h) One of the synthesized grasps without using the mask.

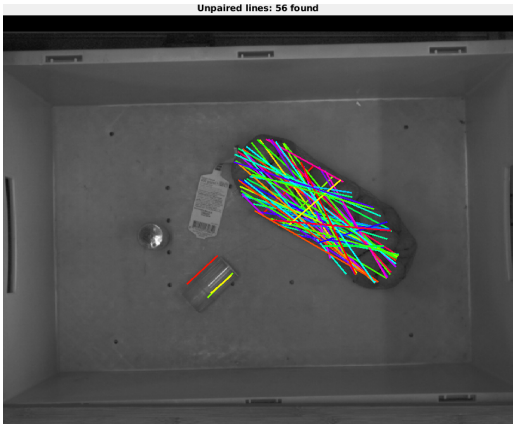
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	20	1.5000	80	3	200	0.4000	2	100

(i) The parameters used when using the segmentation mask.

Figure E.47: Grasp synthesis for three bottles of washing up liquid, comparing between no segmentation and a segmentation mask.



(a) The grey-scale image.



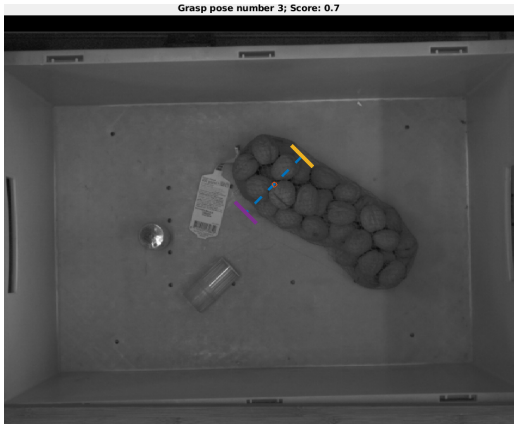
(b) The filtered depth image using the segmentation mask.



(c) One of the synthesized grasps using the mask.



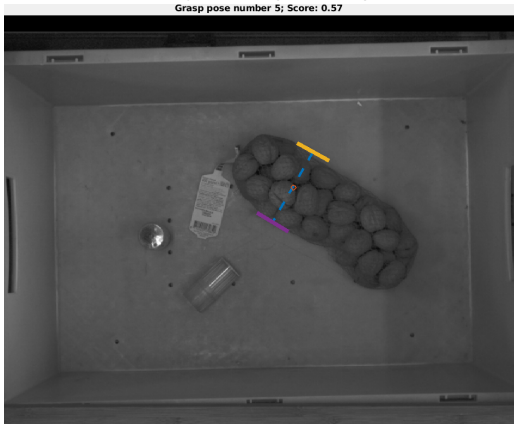
(d) One of the synthesized grasps without using the mask..



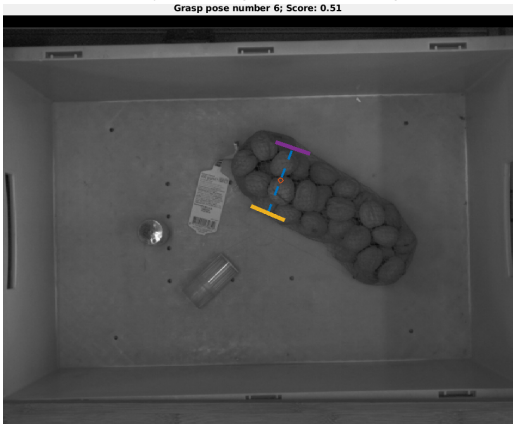
(e) One of the synthesized grasps using the mask.



(f) One of the synthesized grasps without using the mask.



(g) One of the synthesized grasps using the mask.

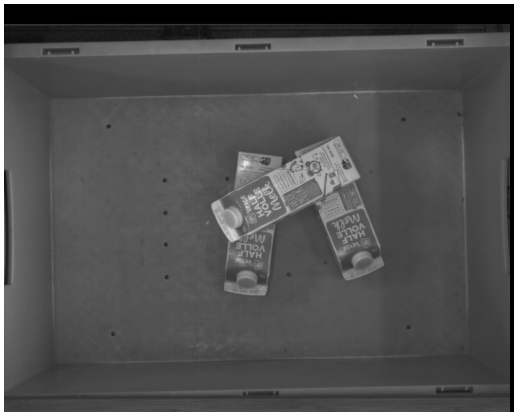


(h) One of the synthesized grasps without using the mask.

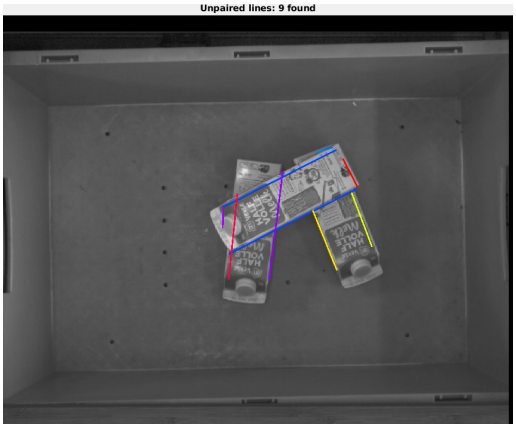
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	20	1	80	3	200	0.6000	2	100

(i) The parameters used when using the segmentation mask.

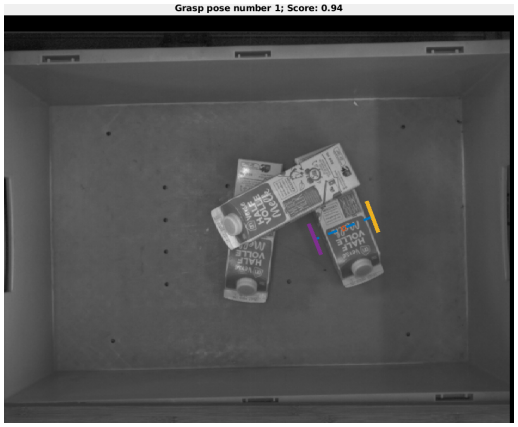
Figure E.48: Grasp synthesis for a net of walnuts and two packs of toothpicks.



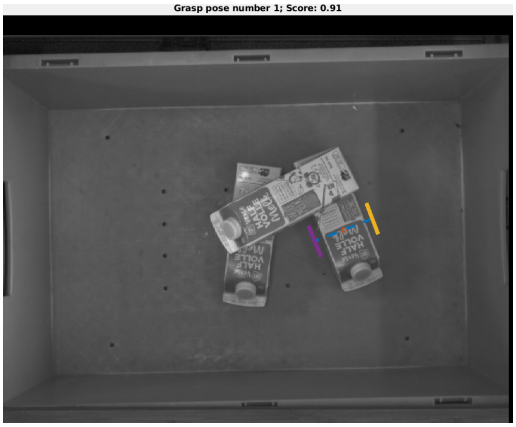
(a) The grey-scale image.



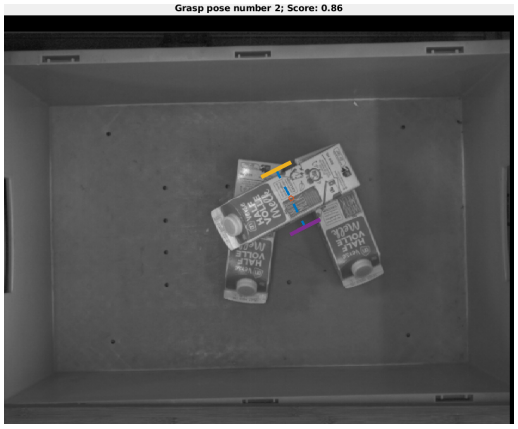
(b) The filtered depth image using the segmentation mask.



(c) One of the synthesized grasps using the mask.



(d) One of the synthesized grasps without using the mask..



(e) One of the synthesized grasps using the mask.



(f) One of the synthesized grasps without using the mask.



(g) One of the synthesized grasps using the mask.



(h) One of the synthesized grasps without using the mask.

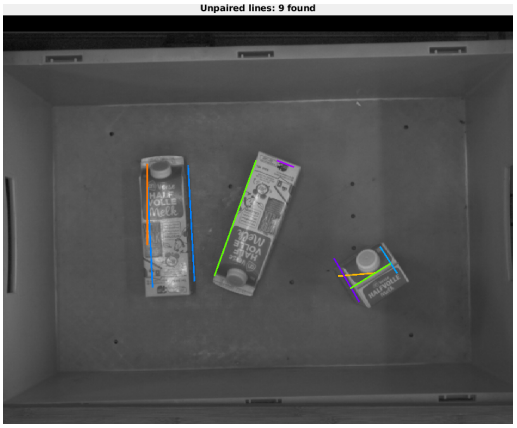
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	20	1	80	3	200	0.4900	2	100

(i) The parameters used when using the segmentation mask.

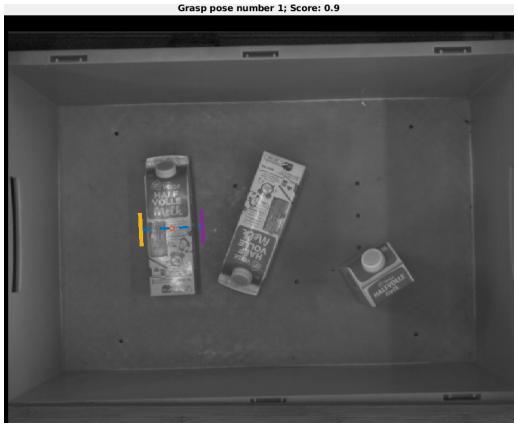
Figure E.49: Grasp synthesis for three cartons of milk that overlap.



(a) The grey-scale image.



(b) The filtered depth image using the segmentation mask.



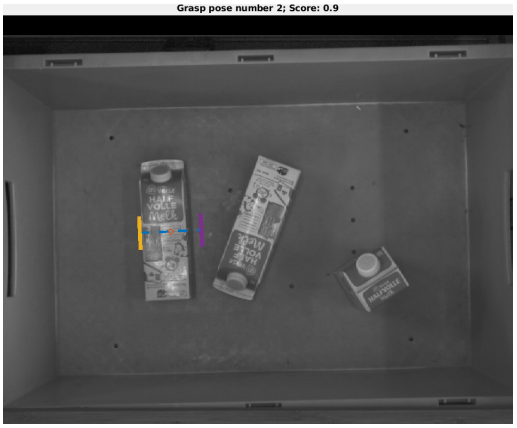
(c) One of the synthesized grasps using the mask.



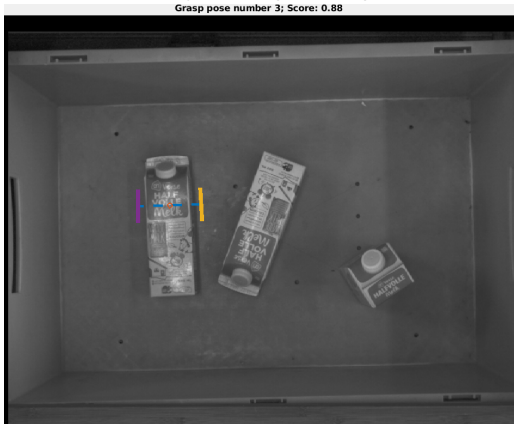
(d) One of the synthesized grasps without using the mask..



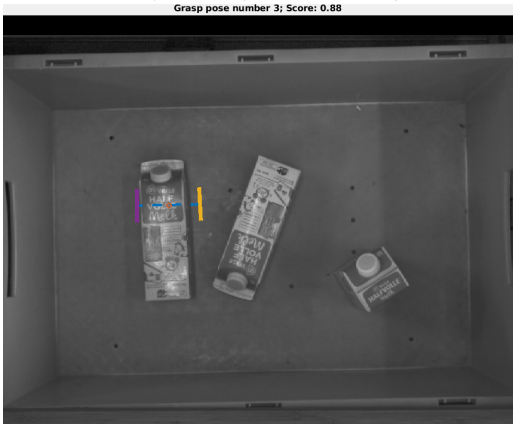
(e) One of the synthesized grasps using the mask.



(f) One of the synthesized grasps without using the mask.



(g) One of the synthesized grasps using the mask.

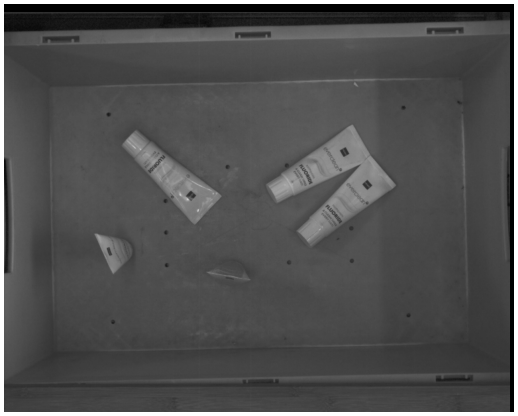


(h) One of the synthesized grasps without using the mask.

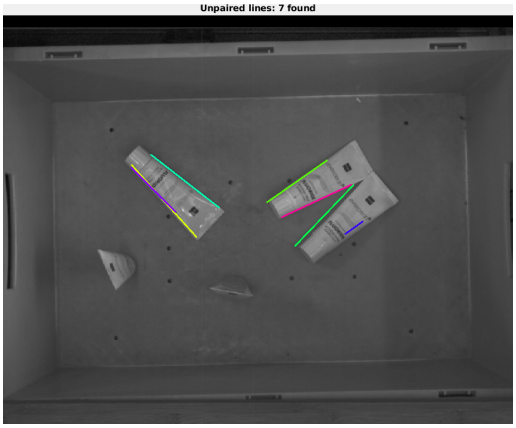
	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	20	1	80	3	200	0.3600	2	100

(i) The parameters used when using the segmentation mask.

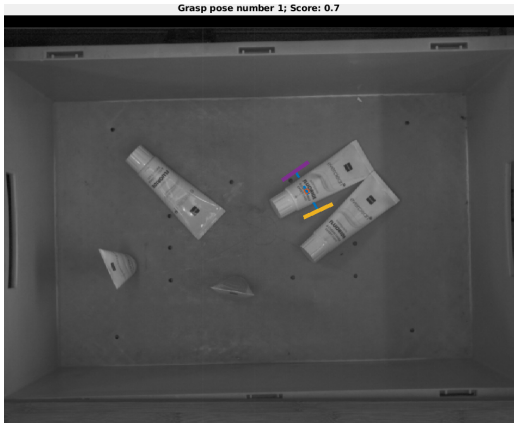
Figure E.50: Grasp synthesis for three cartons of milk.



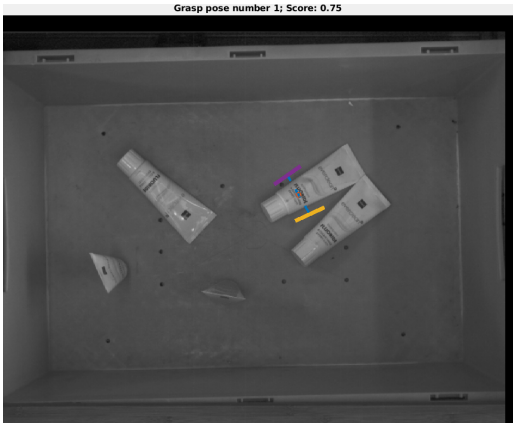
(a) The grey-scale image.



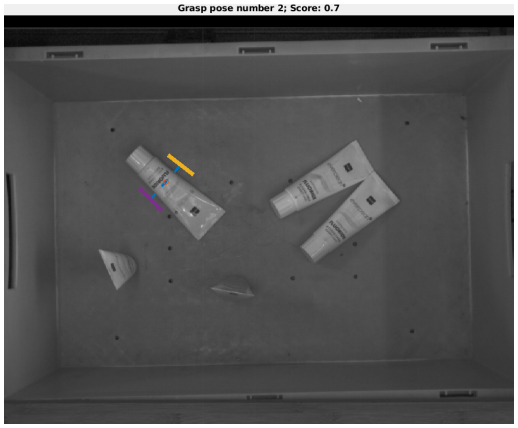
(b) The filtered depth image using the segmentation mask.



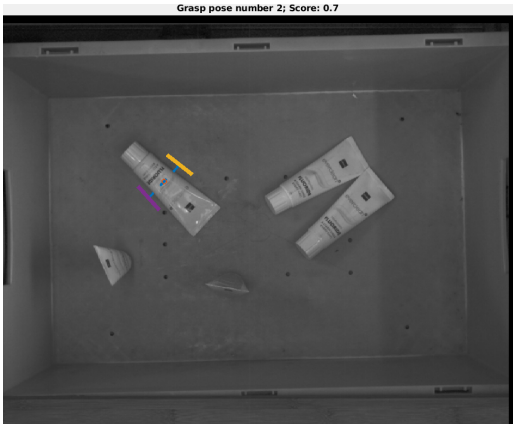
(c) One of the synthesized grasps using the mask.



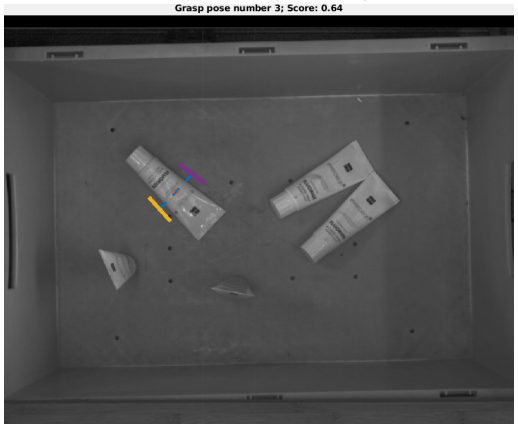
(d) One of the synthesized grasps without using the mask..



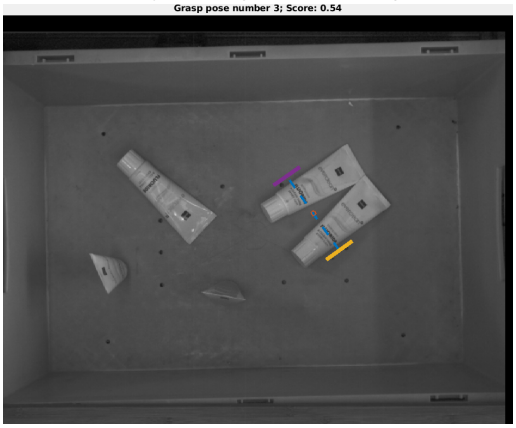
(e) One of the synthesized grasps using the mask.



(f) One of the synthesized grasps without using the mask.



(g) One of the synthesized grasps using the mask.



(h) One of the synthesized grasps without using the mask.

	gripperLength	enlargeGraspOpening	dilStrength	fudgeFactor	fillGap	dilStrengthEdge	nPeaks	facPeak	dTheta	nGripperDepths
1	0.0500	0.0150	20	1	80	3	200	0.4000	6	100

(i) The parameters used when using the segmentation mask.

Figure E.51: Grasp synthesis for five tubes of toothpaste.

F

Ontology and Reasoner Results

This appendix presents the results of the evaluation of RSIR. Section F.1 presents the Prlog queries that RSIR uses to answer competency questions. Afterwards, section F.2 presents the ontology used in the evaluation of RSIR. Finally, sections F.3, F.4, F.5 and F.6 present the results for the evaluation using exploitation, prior knowledge, exploration, and noisy knowledge of capabilities, respectively.

F.1. Queries for Answering Competency Questions

This section provides the concrete queries that RSIR uses to answer the competency questions. For each question, a screen capture is included that shows the query being entered in ROSprolog, and the output RSIR gives.

Task Questions
The task questions are answered as follows:

Q1. What capabilities does the system have access to?

```
?- rdf(vi:robotE,vi:robotPart,Device).
Device = vi:device_camera_input_RGB ;
Device = vi:device_camera_input_depth ;
Device = vi:device_gripper_opening_sensor ;
Device = vi:device_vacuum_sensor ;
Device = vi:fizyr_segmentation ;
Device = vi:grippers_evaluation_robot ;
Device = vi:smartrobotics_motion ;
Device = vi:synths_evaluation_robot ;
Device = vi:verification_gripper_opening ;
Device = vi:verification_vacuum_pressure.

?- rdf(vi:grippers_evaluation_robot,vi:provides,Capability).
Capability = vi:selectGripperThreeFinger ;
Capability = vi:selectGripperTwoFingerLarge ;
Capability = vi:selectGripperTwoFingerSmall ;
Capability = vi:selectGripperVacuumLarge ;
Capability = vi:selectGripperVacuumSmall.
```

Figure F.1: The queries that answer Q1. The first query finds Devices, whereas the second query finds the Capabilities the Device provides.

Q2. How are the capabilities constrained?

```
?- rdf(vi:'selectGripperVacuumLarge',vi:'propertyConstraint',Constraint).
Constraint = vi:selectGripperVacuumLargeSize ;
Constraint = vi:selectGripperVacuumPorosity ;
Constraint = vi:selectGripperVacuumShape.

?- rdf(vi:'selectGripperVacuumPorosity',P,O).
P = rdf:type,
O = owl:'NamedIndividual' ;
P = rdf:type,
O = vi:'ItemPropertyConstraint' ;
P = rdf:type,
O = vi:'Porosity' ;
P = vi:confidenceValue,
O = literal('0.99 0.2 0.01').
```

Figure F.2: The queries that answer Q2. The first query finds constraints, whereas the second query finds the details for a constraint.

Q3. What is the predicted performance of each capability?

```
?- vi_prolog:capability_cost(robotE,senseVerificationRGB,Cost).
Warning: assuming default confidence for capability http://www.semanticweb.org/vi#senseVerificationRGB
Cost = 1.25.
```

Figure F.3: The query that answers Q3. A warning is printed because the system uses the capability's default confidence, rather than the confidence computed based on the item's properties and the capability's constraints.

Q4. What actions must the system do to complete a task?

```
?- createactionplans(robotE,taskVerifyItem).
   (...)
actionPlan_a0_d26_e23:
- senseInputCarrierDepth
- selectGripperTwoFingerSmall
- synthesizeParallelLineGrasp
- moveToGraspPose
- activateGripperToGrasp
- senseGripperOpeningAfterGrasping
- verifyItemThroughGripperOpening
with confidence: 0.5959008000000001
with execution time: 2003.25
with cost: 75.10928145266668

actionPlan_a0_d22_e1:
- senseInputCarrierDepth
- selectGripperTwoFingerLarge
- synthesizeParallelLineGrasp
- moveToGraspPose
- activateGripperToGrasp
- senseGripperOpeningAfterGrasping
- verifyItemThroughGripperOpening
with confidence: 0.5959008000000001
with execution time: 2003.25
with cost: 75.10928145266668
true.
```

Figure F.4: The query that creates the action plans. Only two resulting action plans are included in the figure for brevity.

Q5. How confident is the system that it can complete a task?

Q6. What is the predicted performance for completing a task?

Q7. If there are multiple approaches to completing a task, which approach is optimal?

```
?- bestactionplan(robot1,taskVerifyItem,BestActionPlan).
   (...)

actionPlan_a3_d11_a1:
- selectGripperVacuumSmall
- senseInputCarrierDepth
- senseInputCarrierRGB
- segmentInputRGBImage
- synthesizeFizyrGrasp
- moveToGraspPose
- activateGripperToGrasp
- senseSuctionPressureAfterGrasping
- verifyItemThroughPressure
with confidence: 1.0
BestActionPlan = [vi:senseInputCarrierDepth, vi:selectGripperSchunk,
vi:synthesizeBaumgartlGraspUnsegmented, vi:moveToGraspPose, vi:activ
ateGripperToGrasp, vi:senseGripperOpeningAfterGrasping, vi:verifyIte
mThroughGripperOpening] .
```

Figure F.5: The query that answers Q5, Q6, and Q7. All action plans are created (Q5) and for each the confidence score (Q6), execution time, and cost are computed. The action plan with the lowest cost is selected as the optimal plan (Q7). Out of the action plans created by this query, only one is included in the figure for brevity.

Failure Questions

The failure questions are answered as follows:

Q8. What kinds of failures can occur?

```
?- rdf(vi:'selectGripperVacuumLarge',vi:'propertyConstraint',Constraint).
Constraint = vi:selectGripperVacuumLargeSize ;
Constraint = vi:selectGripperVacuumPorosity ;
Constraint = vi:selectGripperVacuumShape.

?- rdf(vi:'selectGripperVacuumPorosity',P,O).
P = rdf:type,
O = owl:'NamedIndividual' ;
P = rdf:type,
O = vi:'ItemPropertyConstraint' ;
P = rdf:type,
O = vi:'Porosity' ;
P = vi:confidenceValue,
O = literal('0.99 0.2 0.01').
```

Figure F.6: The query that answers Q8. Any capability that is constrained for one or more item properties can fail.

Q9. Which of these failures are likely to occur?

```
?- vi_prolog:capability_confidence(robot1,moveToGraspPose,Confidence).
Warning: assuming default confidence for capability http://www.semanticweb.org/vi#moveToGraspPose
Confidence = 0.8.
```

Figure F.7: The query that answers Q9. Capabilities with low confidence scores are more likely to fail. The queries in Figure F.6 provide more details on what item properties would likely cause the capability to fail.

Q10. When a failure has occurred...

(a). ...what type of failure is it?

(b). ...what is the cause behind the failure?

(c). ...what can be concluded about...

(i). the properties of the item?

(ii). the capabilities of the robot?

(iii). the actions the system should take to prevent this failure from re-occurring?

This question is answered by the Bayesian network, and thus there is no Prolog query to answer it. However, question Q10(c)(iii) is answered by the query in Figure F.5 after the Bayesian network has updated the item properties.

Item Questions

Finally, the competency questions regarding items are:

Q11. Which item properties are relevant for the task at hand?

```
?- rdf(vi:actionPlan_a3_d11_a1,vi:abstractPart,_Capability),rdf(_Capability,vi:propertyConstraint,_Constraint),
rdf(_Constraint,rdf:type,Property),owl_individual_of(Property,vi:'ItemProperty').
Property = vi:'PrimitiveShape' ;
Property = vi:'PrimitiveShape' ;
Property = vi:'Transparency' ;
Property = vi:'PrimitiveShape' ;
Property = vi:'Transparency' ;
Property = vi:'Porosity' ;
Property = vi:'PrimitiveShape' ;
Property = vi:'Mass'.
```

Figure F.8: The query that answers Q11. In this example, the properties relevant for Action Plan `vi:actionPlan_a3_d11_a1` are derived.

Q12. Which item properties are independent of the scenario?

```
?- owl_individual_of(Property,vi:'PersistentProperty'),rdf(Property,rdf:type,owl:'Class').
Property = vi:'Mass' ;
Property = vi:'PrimitiveShape' ;
Property = vi:'Size' ;
Property = vi:'Texture' ;
Property = vi:'Transparency' ;
false.
```

Figure F.9: The query that answers Q12.

Q13. Which item properties are dependent on the scenario?

```
?- owl_individual_of(Property,vi:'ItemProperty'),
not(owl_individual_of(Property,vi:'PersistentProperty')),rdf(Property,rdf:type,owl:'Class').
Property = vi:'BoundingBox' ;
Property = vi:'Footprint' ;
Property = vi:'Segmentation' ;
false.
```

Figure F.10: The query that answers Q13.

Q14. Which item properties are known, and with what accuracy?

```
?- vi_prolog:item_properties(robotE,Property,Probability,Definition).
Property = 'Mass',
Probability = '0.2 0.3 0.2 0.2 0.1',
Definition = '0.5 1.0 1.5 2.0 2.5' ;
Property = 'Porosity',
Probability = '0.95 0.04 0.01',
Definition = 'solid semi porous' ;
Property = 'PrimitiveShape',
Probability = '0.2 0.3 0.3 0.1 0.05 0.05',
Definition = 'sphere cylinder box disk cone complex' ;
Property = 'Size',
Probability = '0.1 0.2 0.3 0.3 0.1',
Definition = '5 10 15 20 25' ;
Property = 'Transparency',
Probability = '0.7 0.2 0.1',
Definition = 'opaque semi transparent' ;
false.
```

Figure F.11: The query that answers Q14.

Q15. Which item properties can be estimated?

```
?- createactionplanforoutput(robot1,'Mass').
   (...)
actionPlan_c10_d2_a0:
- selectGripperVacuumLarge
- senseInputCarrierDepth
- senseInputCarrierRGB
- segmentInputRGBImage
- synthesizeFizyrGrasp
- moveToGraspPose
- activateGripperToGrasp
- senseSuctionPressureAfterGrasping
- verifyItemThroughPressure
- sensePayloadAfterGrasping
- knowIntrinsicPayload
- estimateMassFromPayload
with confidence: 0.29514489903360003
actionPlan_c10_d11_a1:
- selectGripperVacuumSmall
- senseInputCarrierDepth
- senseInputCarrierRGB
- segmentInputRGBImage
- synthesizeFizyrGrasp
- moveToGraspPose
- activateGripperToGrasp
- senseSuctionPressureAfterGrasping
- verifyItemThroughPressure
- sensePayloadAfterGrasping
- knowIntrinsicPayload
- estimateMassFromPayload
with confidence: 0.29514489903360003
true.
```

Figure F.12: The query that answers Q15 for the example of estimating an item's mass. Only the last two action plans are included in this figure for brevity.

```
?- createactionplanforoutput(robot1,'Segmentation').

Considering task: taskSegmentation...

Retracting previously computed action plans...

Computing action plans...
Warning: assuming default confidence for capability http://www.semanticweb.org/vi#segmentInputRGBImage
falseWarning: assuming default confidence for capability http://www.semanticweb.org/vi#senseInputCarrierRGB

Merging action plans...

Removing invalid action plans...

Recovering chronological order of action plans...

Calculating confidences and execution times for action plans...
Warning: assuming default confidence for capability http://www.semanticweb.org/vi#segmentInputRGBImage
Warning: assuming default confidence for capability http://www.semanticweb.org/vi#senseInputCarrierRGB

actionPlan:
- senseInputCarrierRGB
- segmentInputRGBImage
with confidence: 0.6859999999999999
true.
```

Figure F.13: The query that answers Q15 for the example of estimating an item's segmentation. The output shows that there is one action plan that estimates this property.

```
?- createactionplanforoutput(robot1,'PrimitiveShape').

Considering task: taskPrimitiveShape...

Retracting previously computed action plans...

Computing action plans...

Merging action plans...

Removing invalid action plans...

Recovering chronological order of action plans...

Calculating confidences and execution times for action plans...
true.
```

Figure F.14: The query that answers Q15 for the example of estimating an item's primitive shape. The output shows that there are no action plans that estimate this property.

This concludes the section on concrete queries that answer the competency questions.

F.2. Ontology Used for Evaluation

This section presents the capabilities available to RSIR during its evaluation of grasping an unknown item in section 9.2. The capabilities are listed in Table F.1. Furthermore, Figures F.15, F.16, and F.17 present the limitations on the capabilities regarding item properties. Finally, all gripper capabilities have a Cardinality Constraint of 1, meaning RSIR can only use one gripper when grasping the item.

Table F.1: The capabilities available to RSIR when evaluating its reasoning capabilities for grasping an unknown item.

	Capability	**Input**	**Output**
Knowing Capabilities	knowIntrinsicPayload	-	intrinsicPayload
	knowVerificationPose	-	verificationPose
	selectGripperThreeFinger	-	gripperRadius, minOpeningWidth, useOpeningSensingGripper
	selectGripperTwoFingerLarge	-	gripperLength, maxGraspingHeight, maxOpeningWidth, minOpeningWidth, useOpeningSensingGripper
	selectGripperTwoFingerSmall	-	gripperLength, maxGraspingHeight, maxOpeningWidth, minOpeningWidth, useOpeningSensingGripper
	selectGripperVacuumLarge	-	activationPressure, deactivationPressure, suctionCupArea
	selectGripperVacuumSmall	-	activationPressure, deactivationPressure, suctionCupArea
Sensing Capabilities	senseInputCarrierRGB	-	inputCarrierRGB
	senseInputCarrierDepth	-	inputCarrierDepth
	senseVerificationRGB	isAtVerificationPose	verificationRGB
	senseGripperOpeningAfterGrasping	hasGrasped, useOpeningSensingGripper	measuredGripperOpening
	senseSuctionPressureAfterGrasping	hasGrasped	measuredPressureAfterGrasping
	sensePayloadAfterGrasping	hasGrasped	measuredPayload
Processing Capabilities	segmentInputRGBImage	inputCarrierRGB	segmentation, itemCountInInputCarrier
	synthesizeCircularGrasp	gripperRadius, inputCarrierDepth	graspPose
	synthesizeFlatSurfaceGrasp	segmentation, inputCarrierDepth, suctionCupArea	graspPose
	synthesizeParallelLineGrasp	gripperLength, inputCarrierDepth, maxGraspingHeight, maxOpeningWidth, minOpeningWidth	graspPose
	synthesizePrincipalAxisGrasp	segmentation, inputCarrierDepth, maxOpeningWidth	graspPose
	verifyItemThroughGripperOpening	measuredGripperOpening, minOpeningWidth	itemIsInGripper
	verifyItemThroughPressure	activationPressure, deactviationPressure, measuredPressureAfterGrasping	itemIsInGripper
	verifyItemThroughFootprint	verificationRGB	itemIsInGripper, footprint
	verifyItemThroughPayload	intrinsicPayload, measuredPayload	itemIsInGripper
Actuating Capabilities	moveToGraspPose	graspPose	isAtGraspPose
	activateGripperToGrasp	isAtGraspPose	hasGrasped
	moveToVerificationPoseSlowly	hasGrasped	isAtVerificationPose

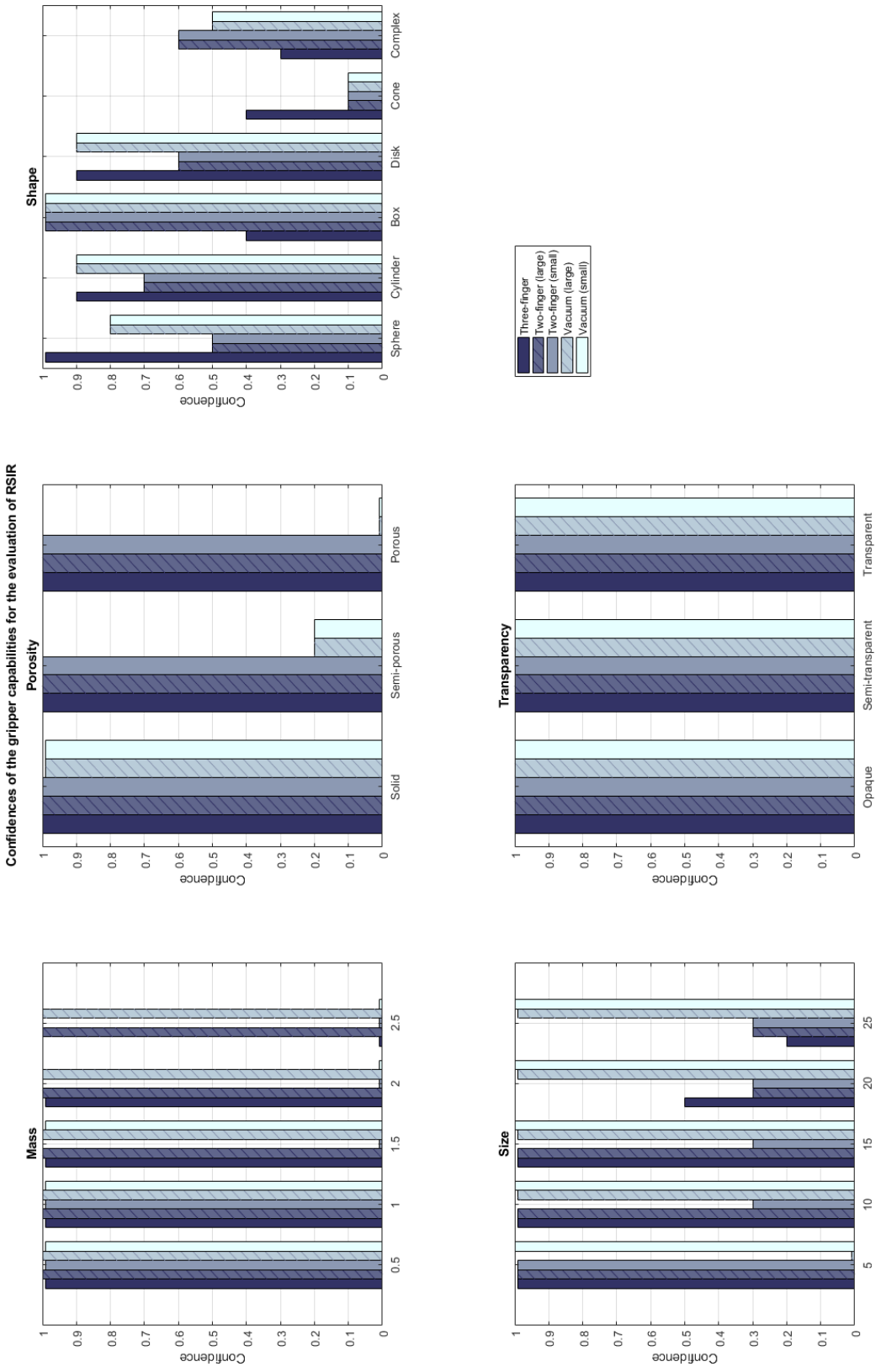


Figure F.15: The constraints for the grippers when evaluating RSIR.

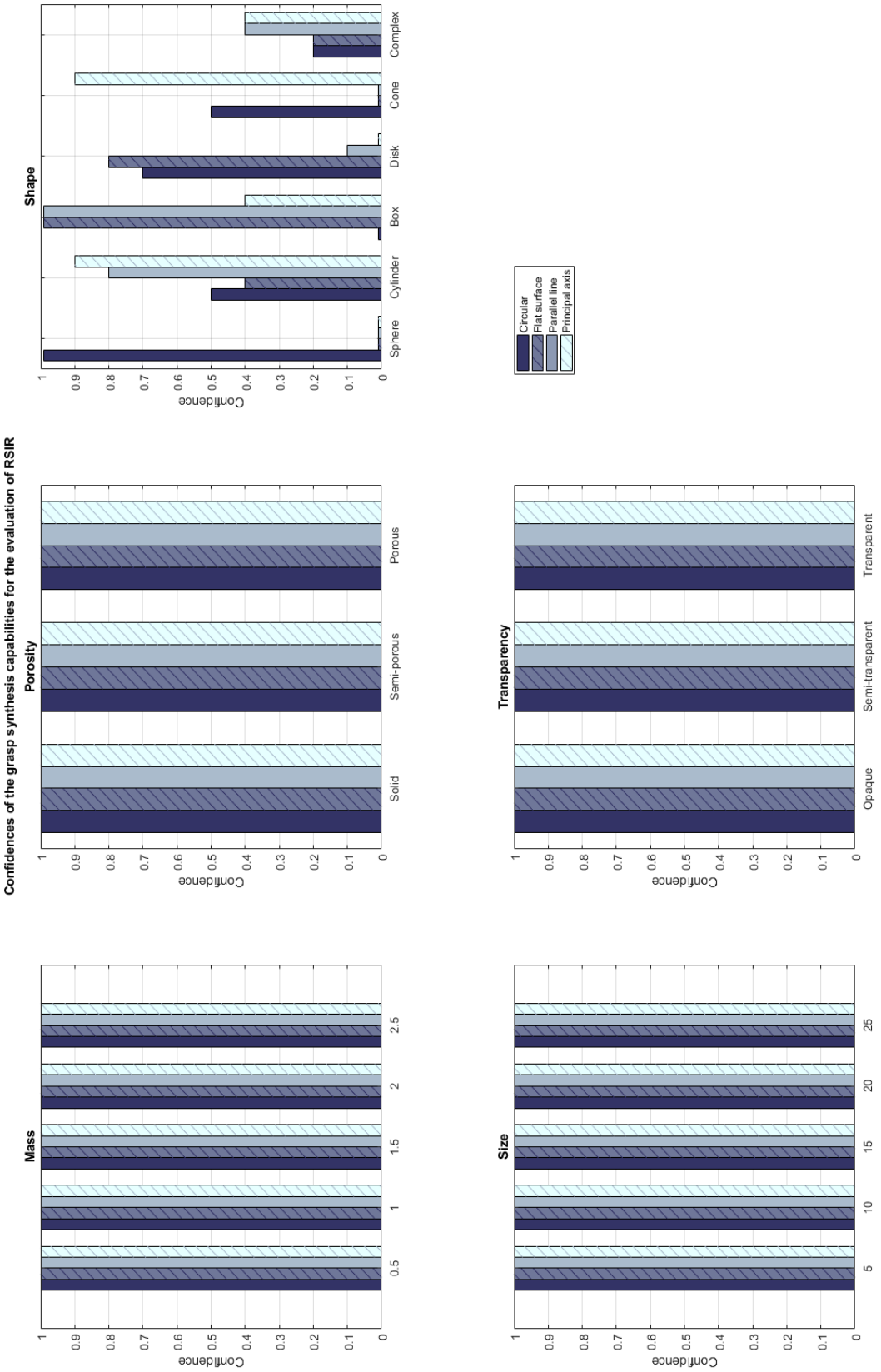


Figure F.16: The constraints for the grasp synthesis approaches when evaluating RSIR.

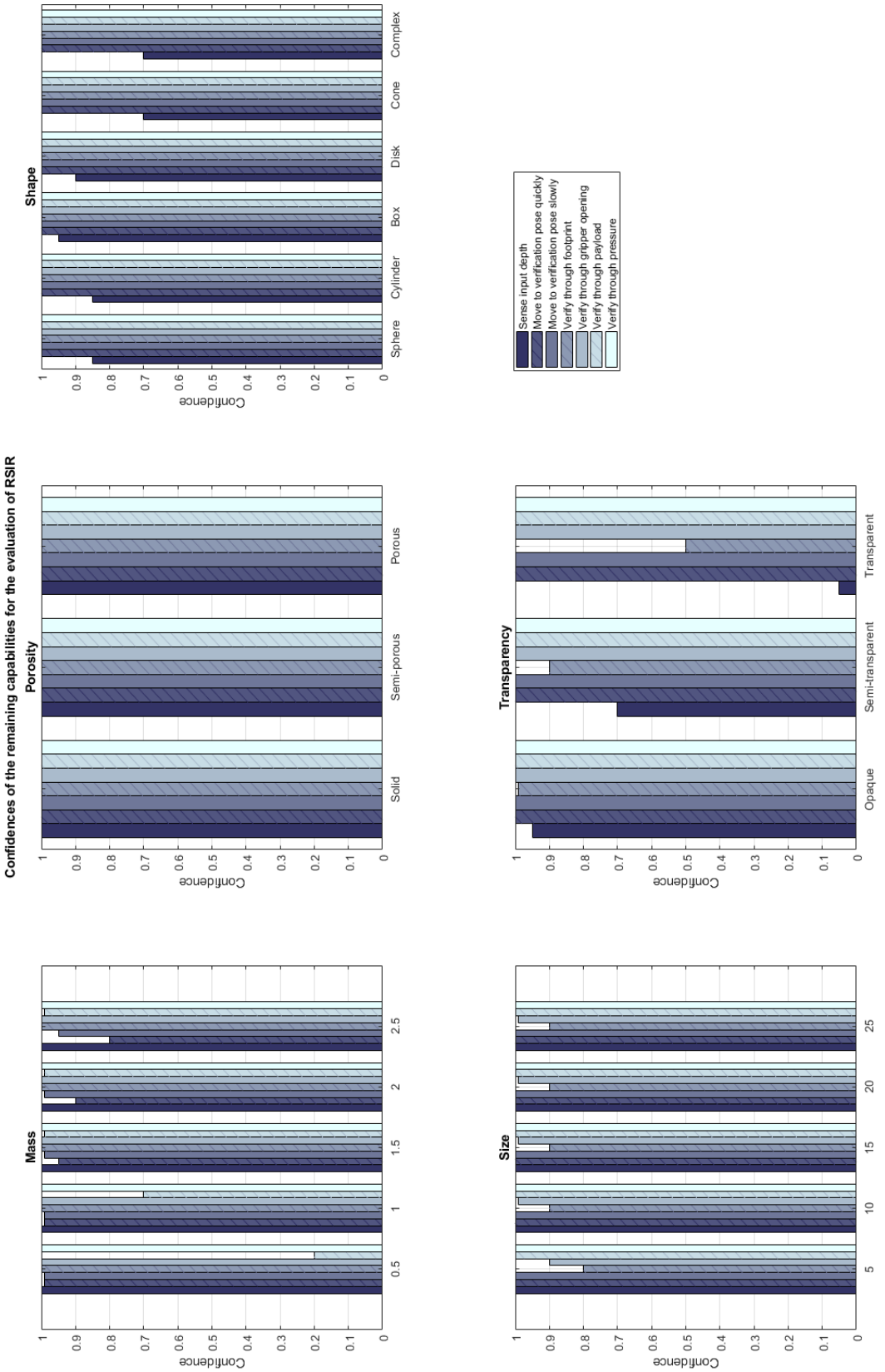


Figure F.17: The constraints for the remaining capabilities when evaluating RSIR.

F.3. Estimated Item Properties using Exploitation

This section presents the estimated item properties for the evaluation of RSIR where it exploits knowledge. The properties are presented in Figures F.18 through F.25.

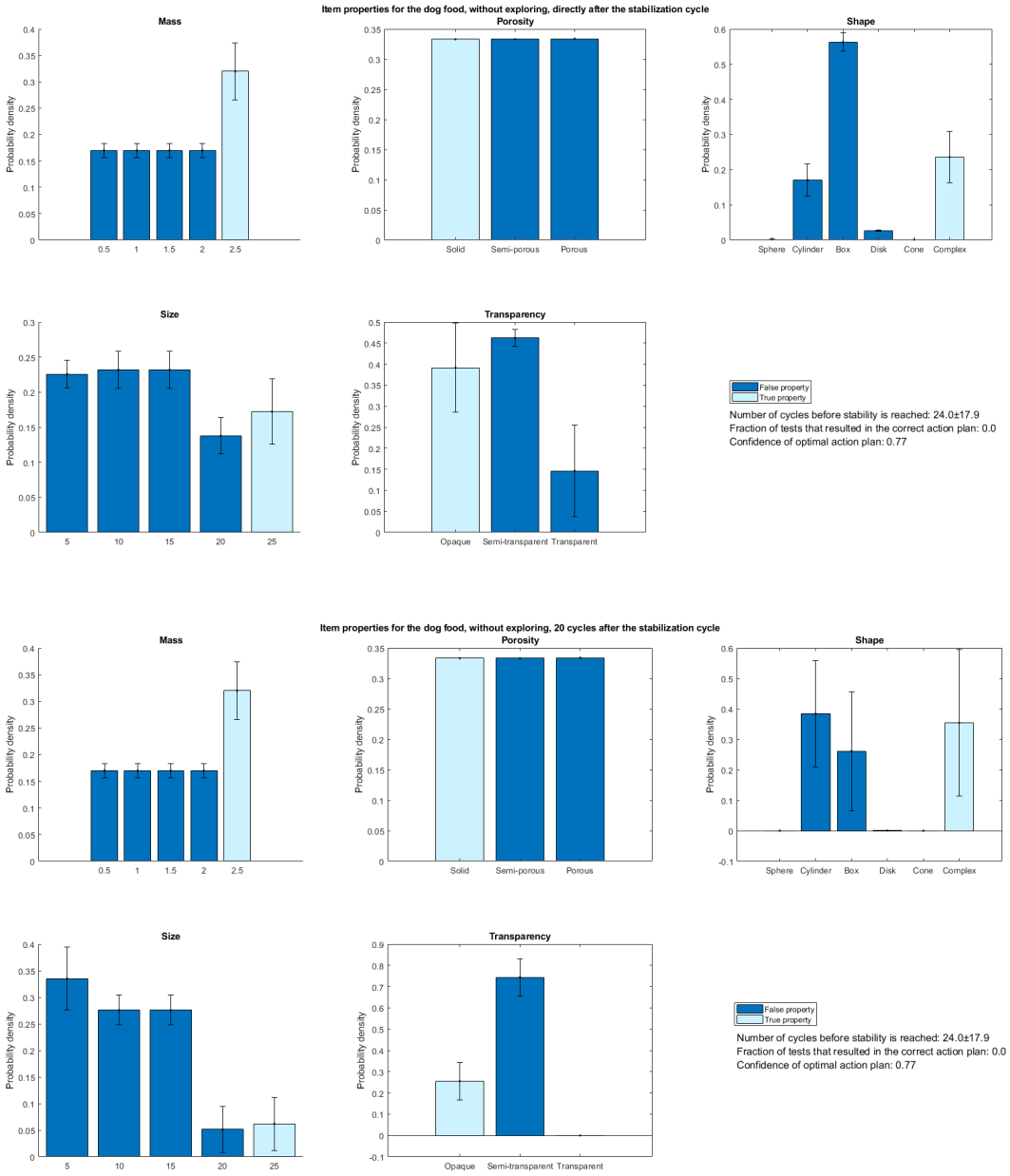


Figure F.18: The estimated item properties for the dog food, when RSIR exploits knowledge.

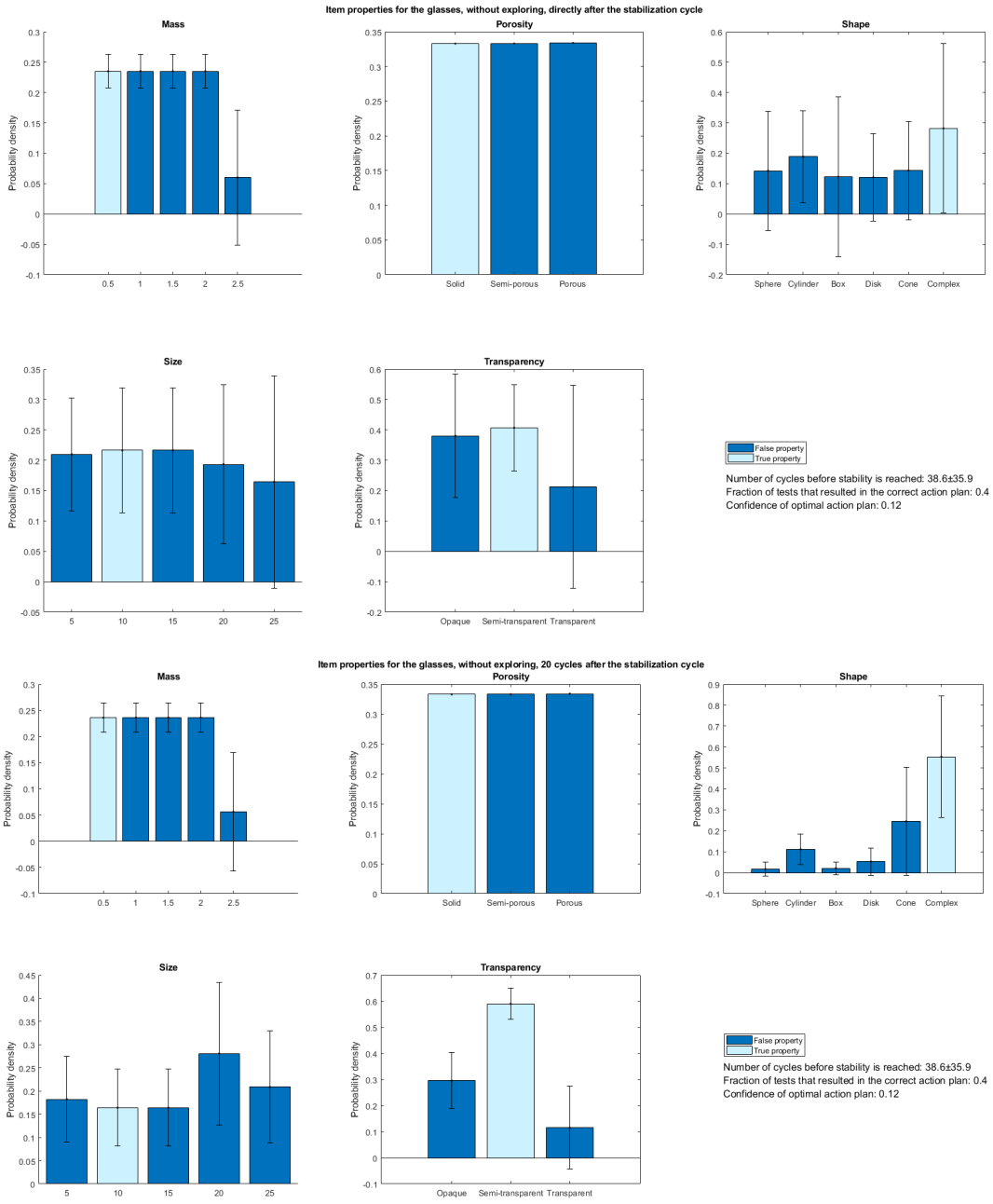


Figure F.19: The estimated item properties for the glasses, when RSIR exploits knowledge.

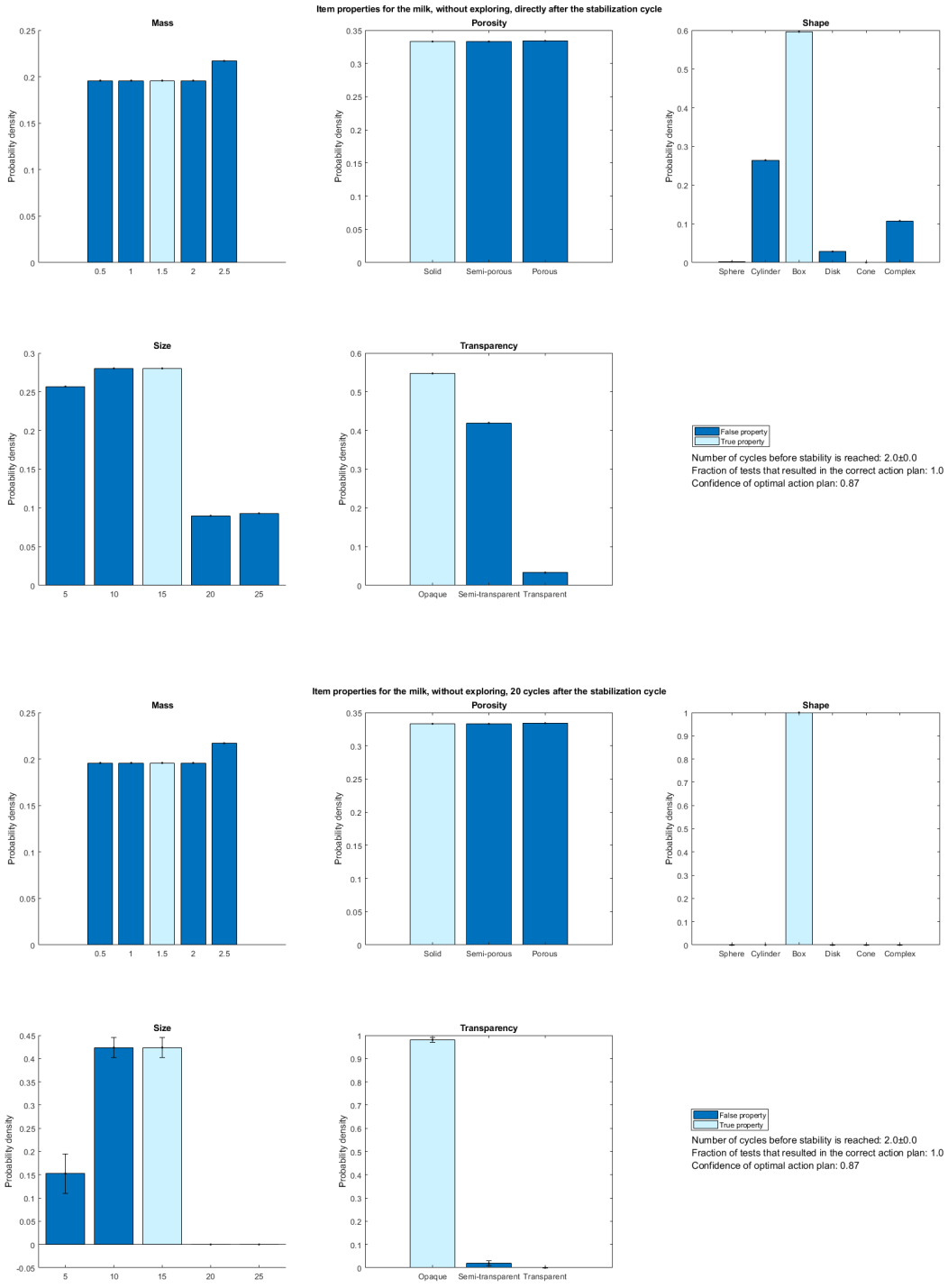


Figure F.20: The estimated item properties for the milk, when RSIR exploits knowledge.

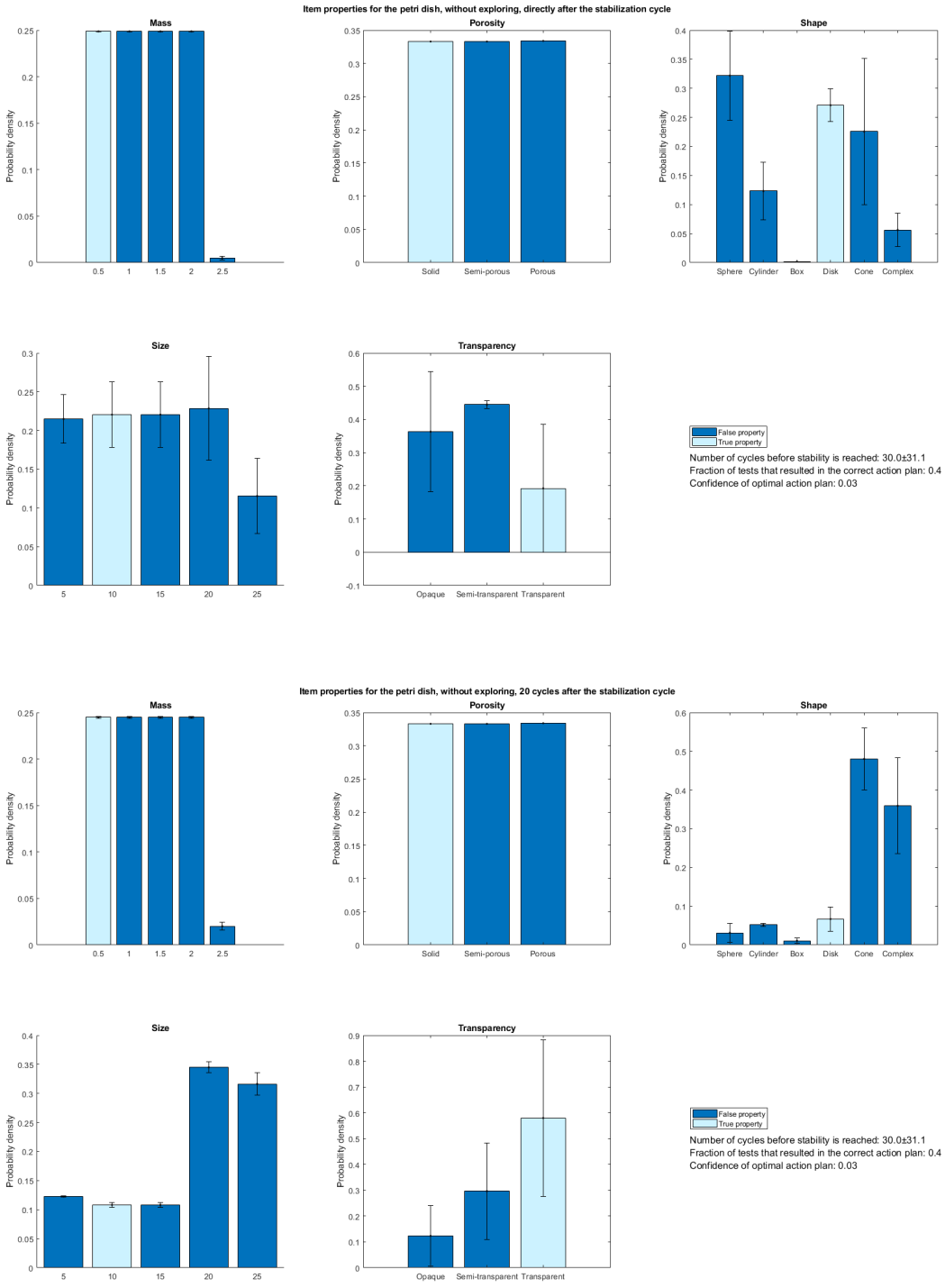


Figure F.21: The estimated item properties for the petri dish, when RSIR exploits knowledge.

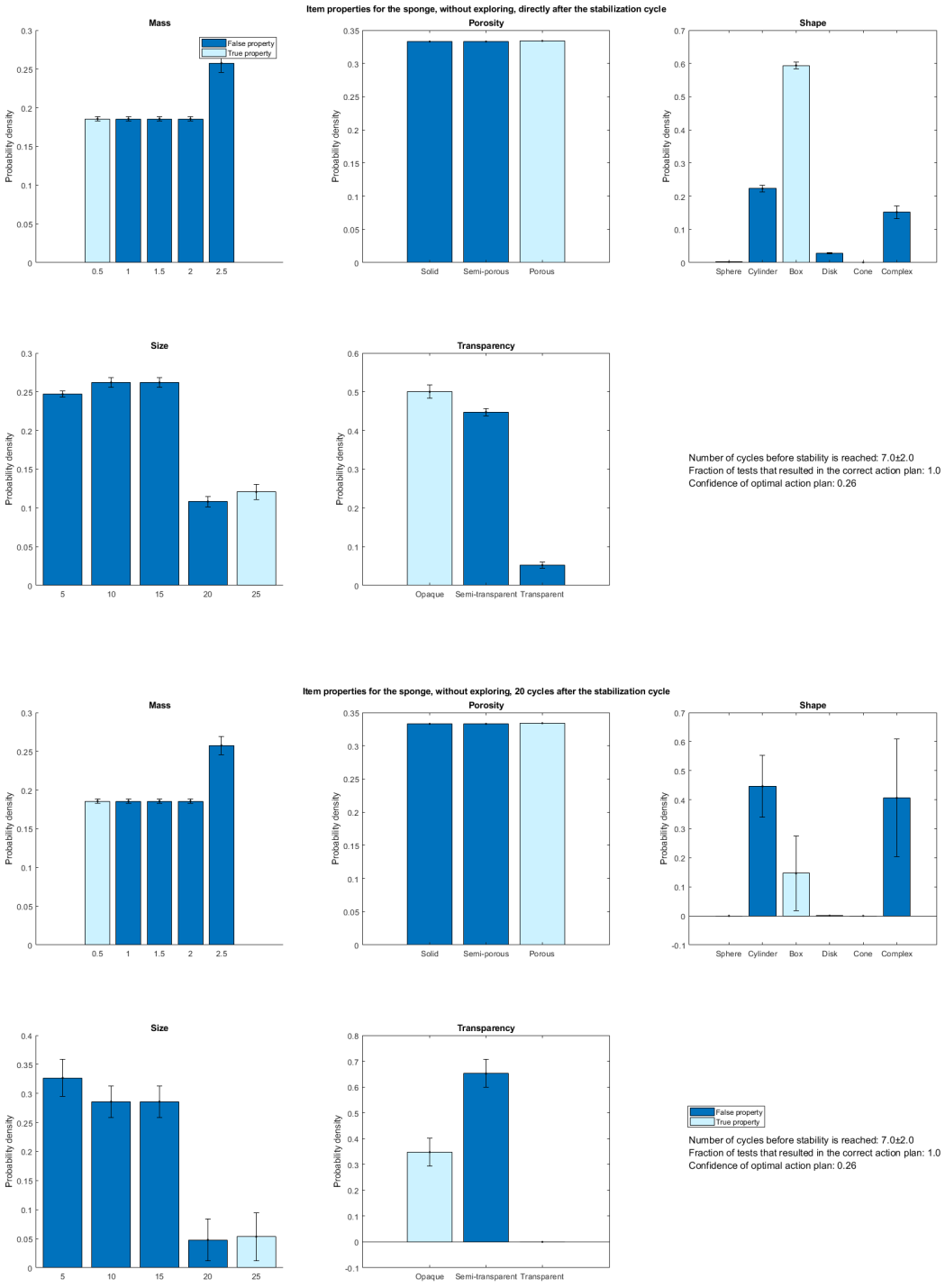


Figure F.22: The estimated item properties for the sponge, when RSIR exploits knowledge.

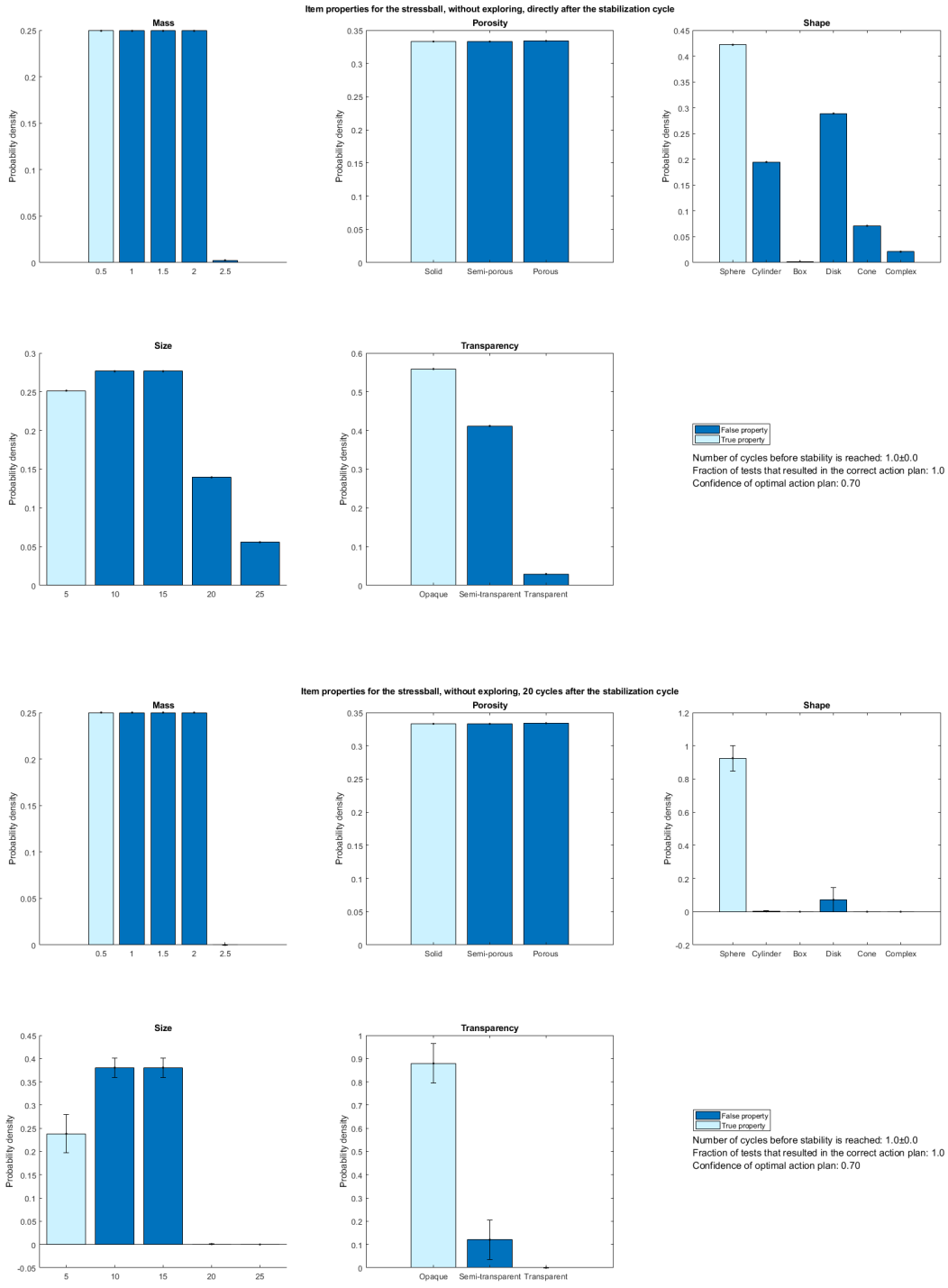


Figure F.23: The estimated item properties for the stressball, when RSIR exploits knowledge.

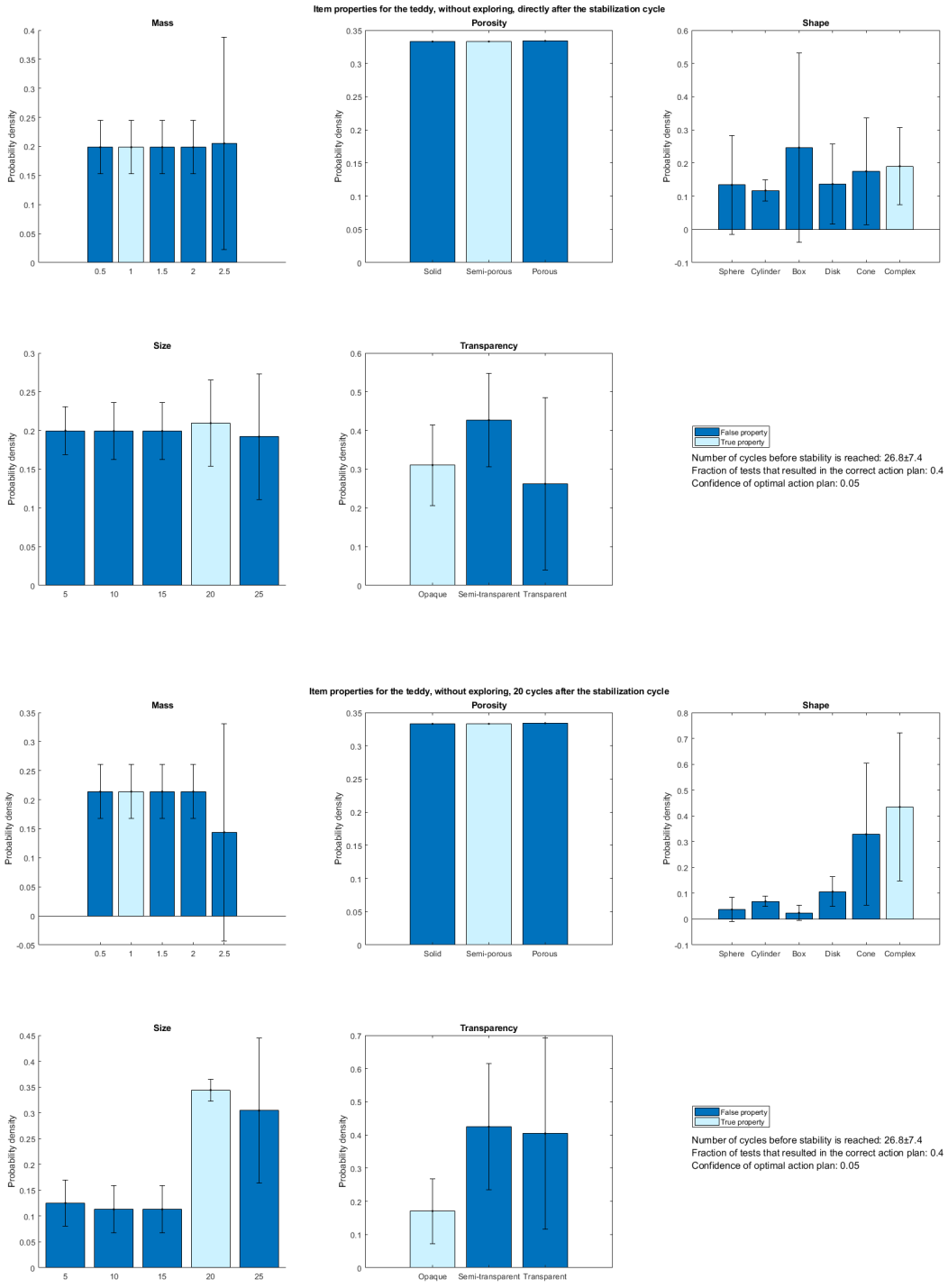


Figure F.24: The estimated item properties for the teddy bear, when RSIR exploits knowledge.

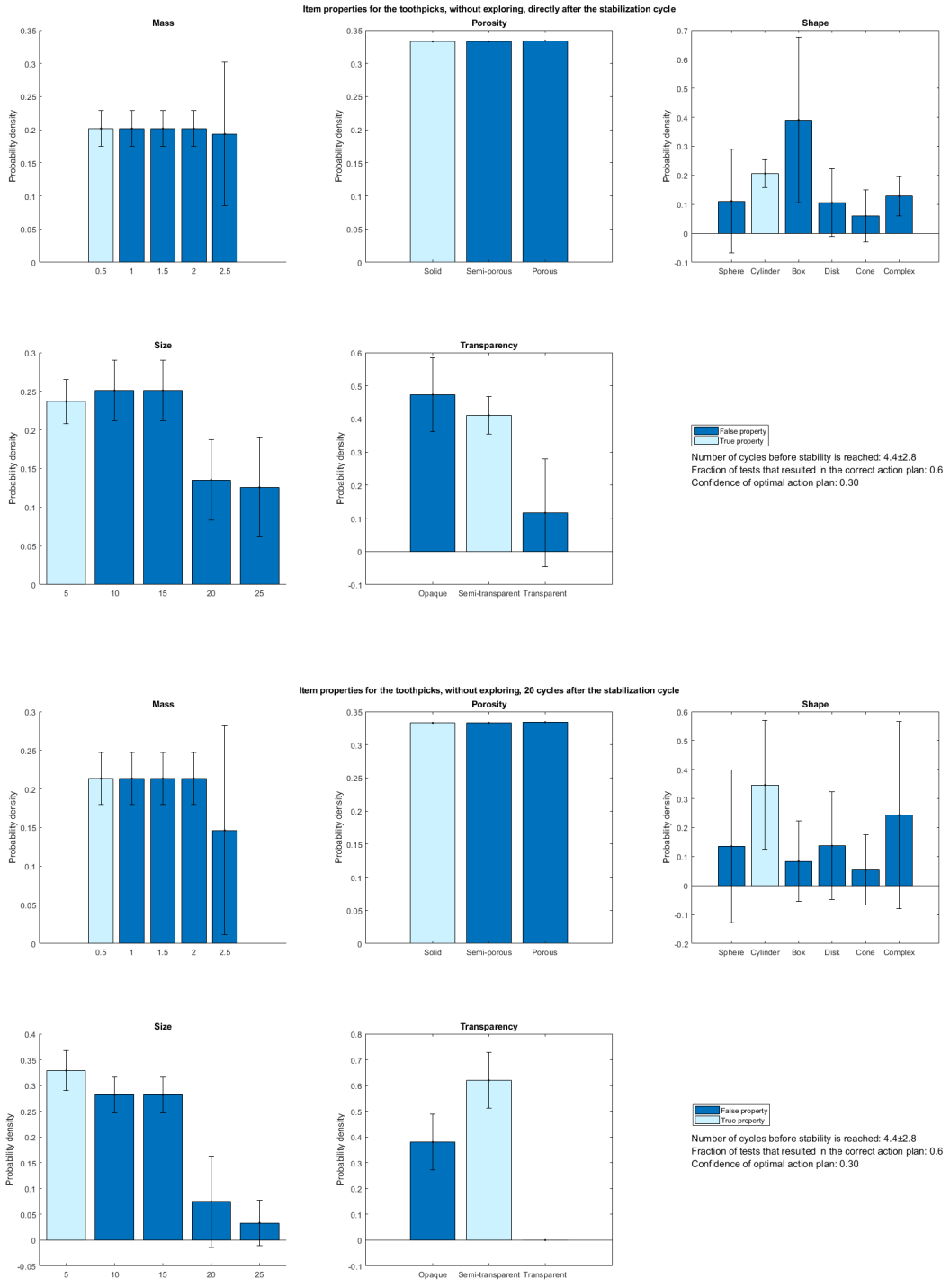


Figure F.25: The estimated item properties for the toothpicks, when RSIR exploits knowledge.

F.4. Estimated Item Properties using Prior Knowledge

This section presents the estimated item properties for the evaluation of RSIR where it exploits knowledge and has access to prior knowledge. The properties are presented in Figures F.26 through F.29.

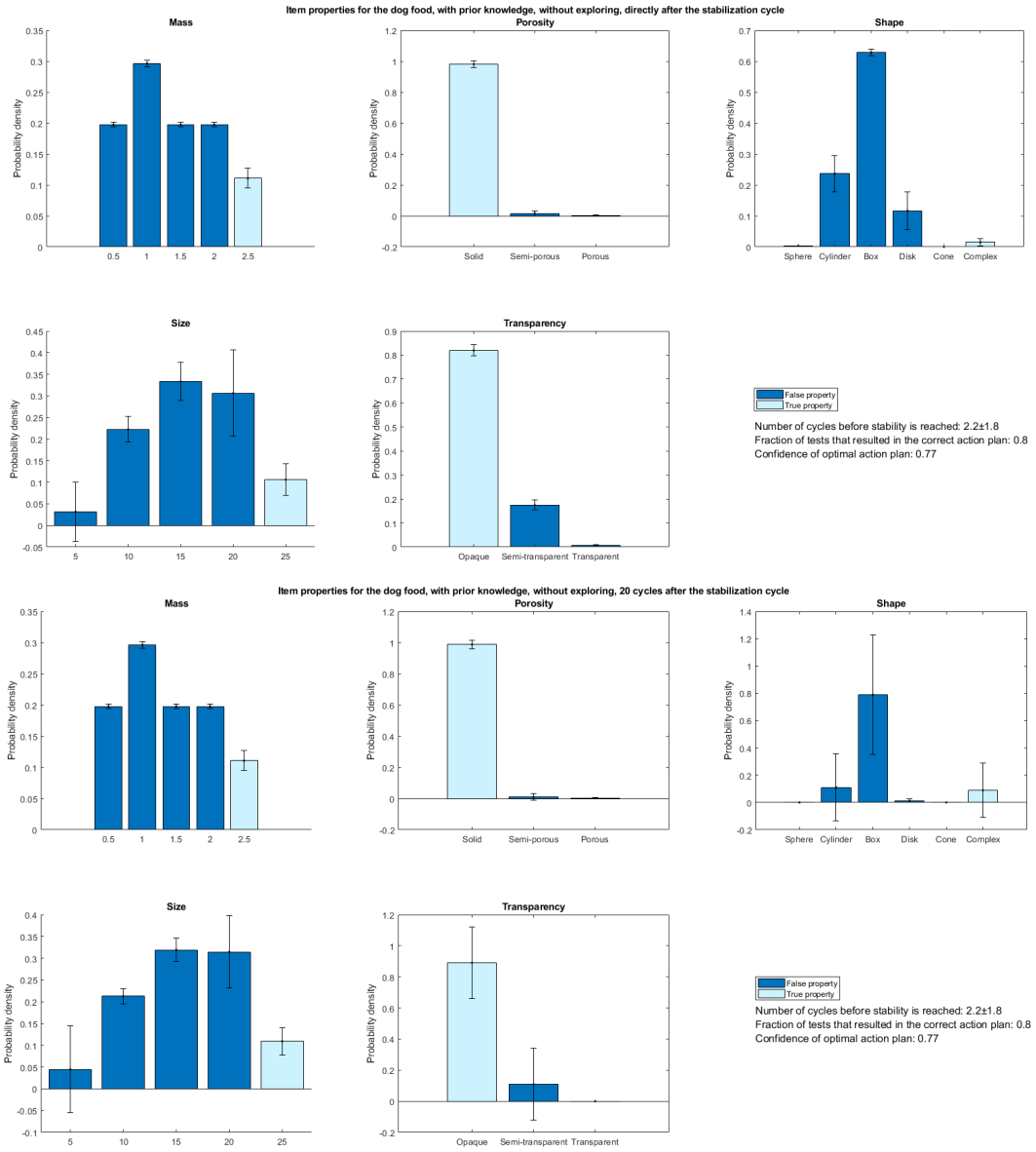


Figure F.26: The estimated item properties for the dog food, when RSIR exploits knowledge and has access to prior knowledge.

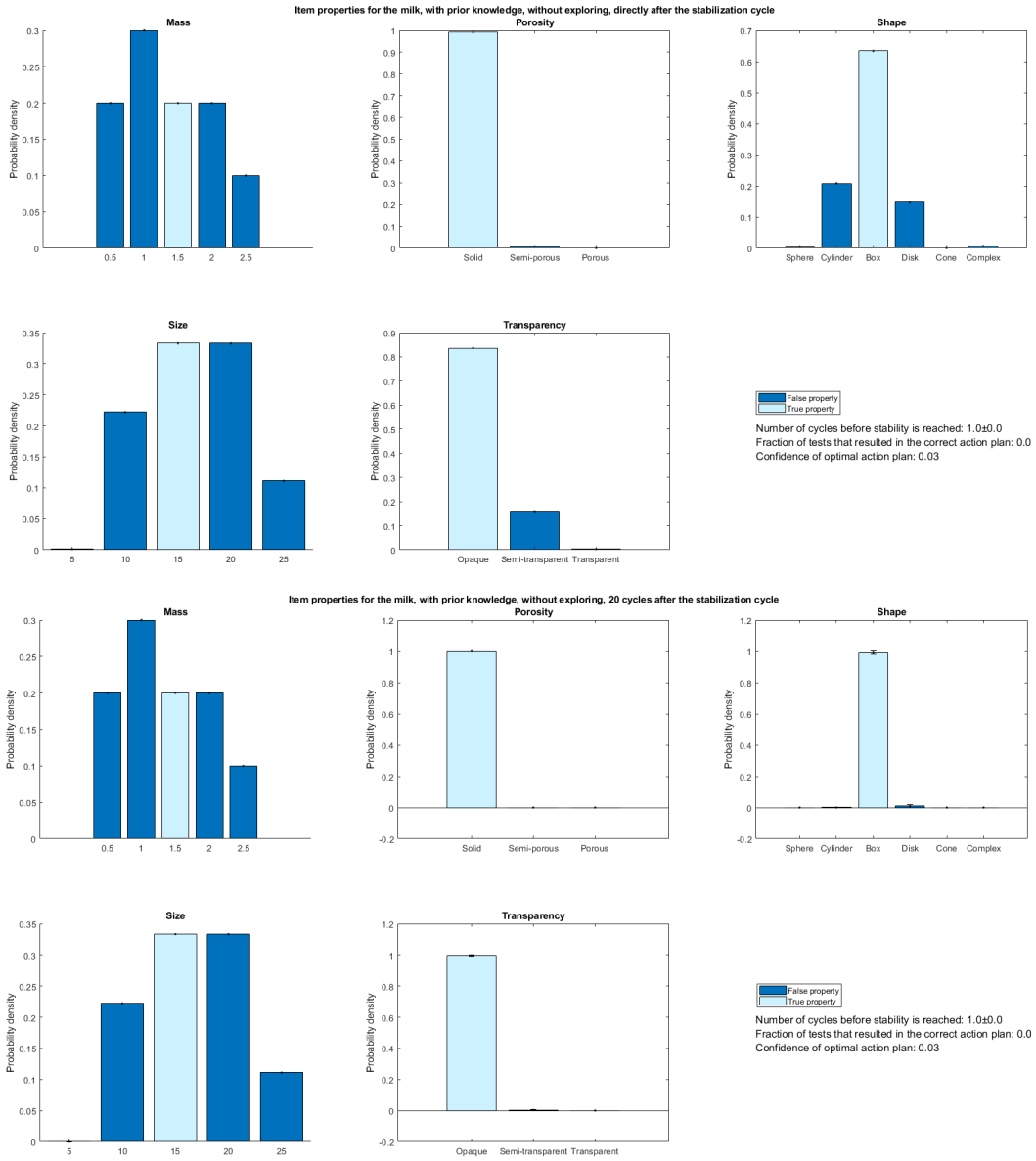


Figure F.27: The estimated item properties for the milk, when RSIR exploits knowledge and has access to prior knowledge.

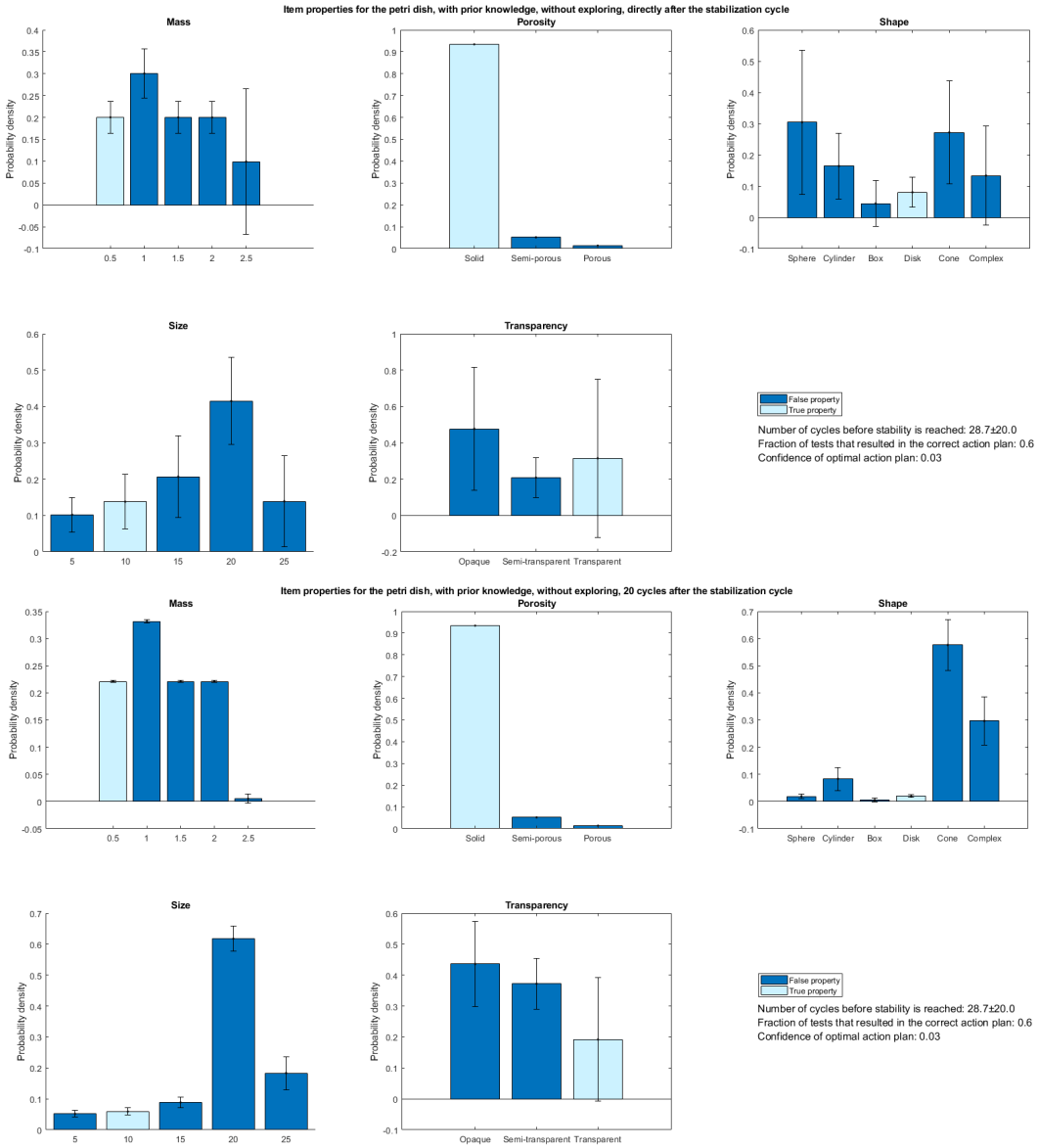


Figure F.28: The estimated item properties for the petri dish, when RSIR exploits knowledge and has access to prior knowledge.

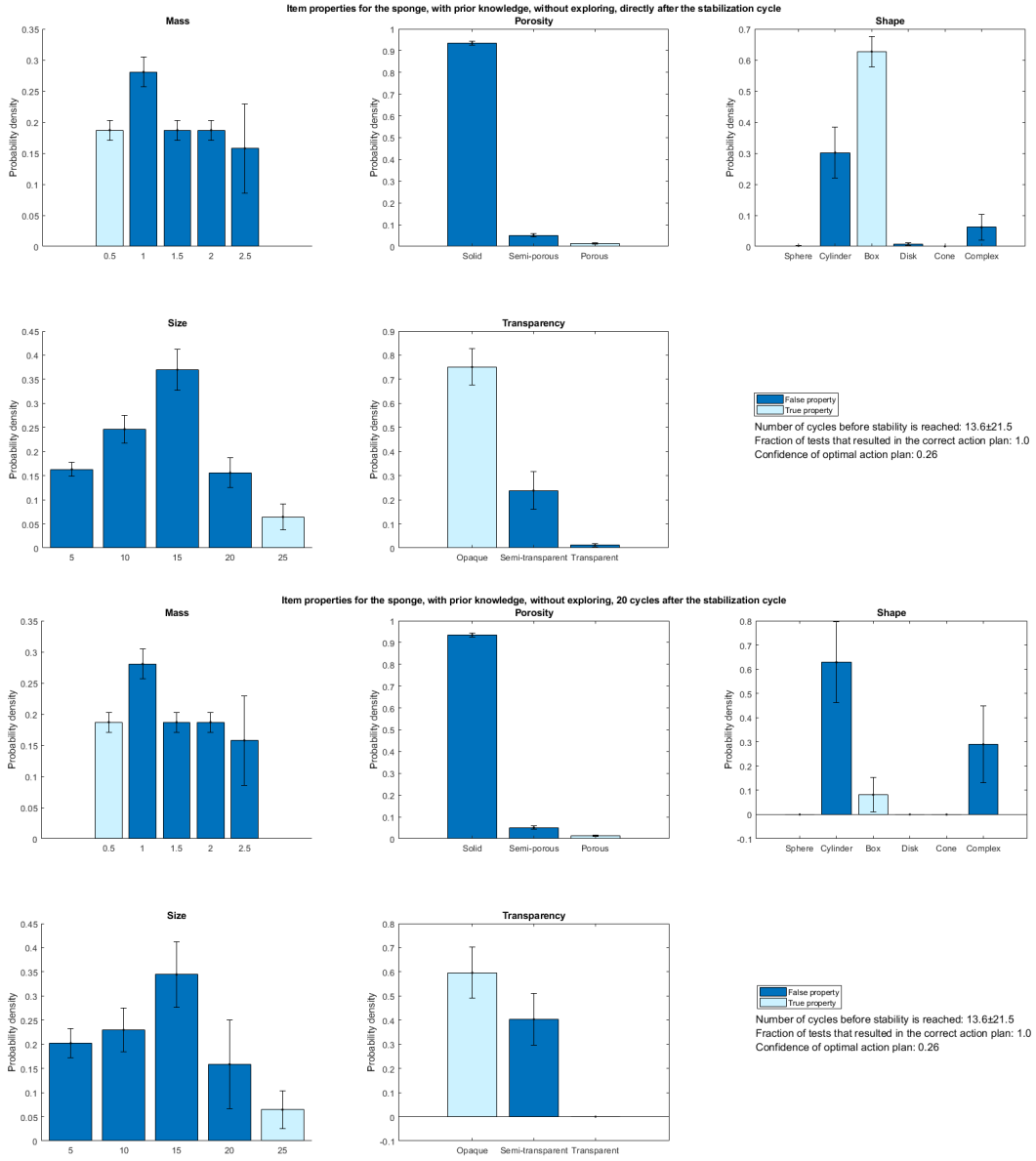


Figure F.29: The estimated item properties for the sponge, when RSIR exploits knowledge and has access to prior knowledge.

F.5. Estimated Item Properties using Exploration

This section presents the estimated item properties for the evaluation of RSIR when it uses explorations.

F.5.1. Exploring 1 in 5 cycles

This subsection presents the estimated item properties for the evaluation of RSIR when it explores once every 5 grasping cycles. The properties are presented in Figures F.30 through F.32.

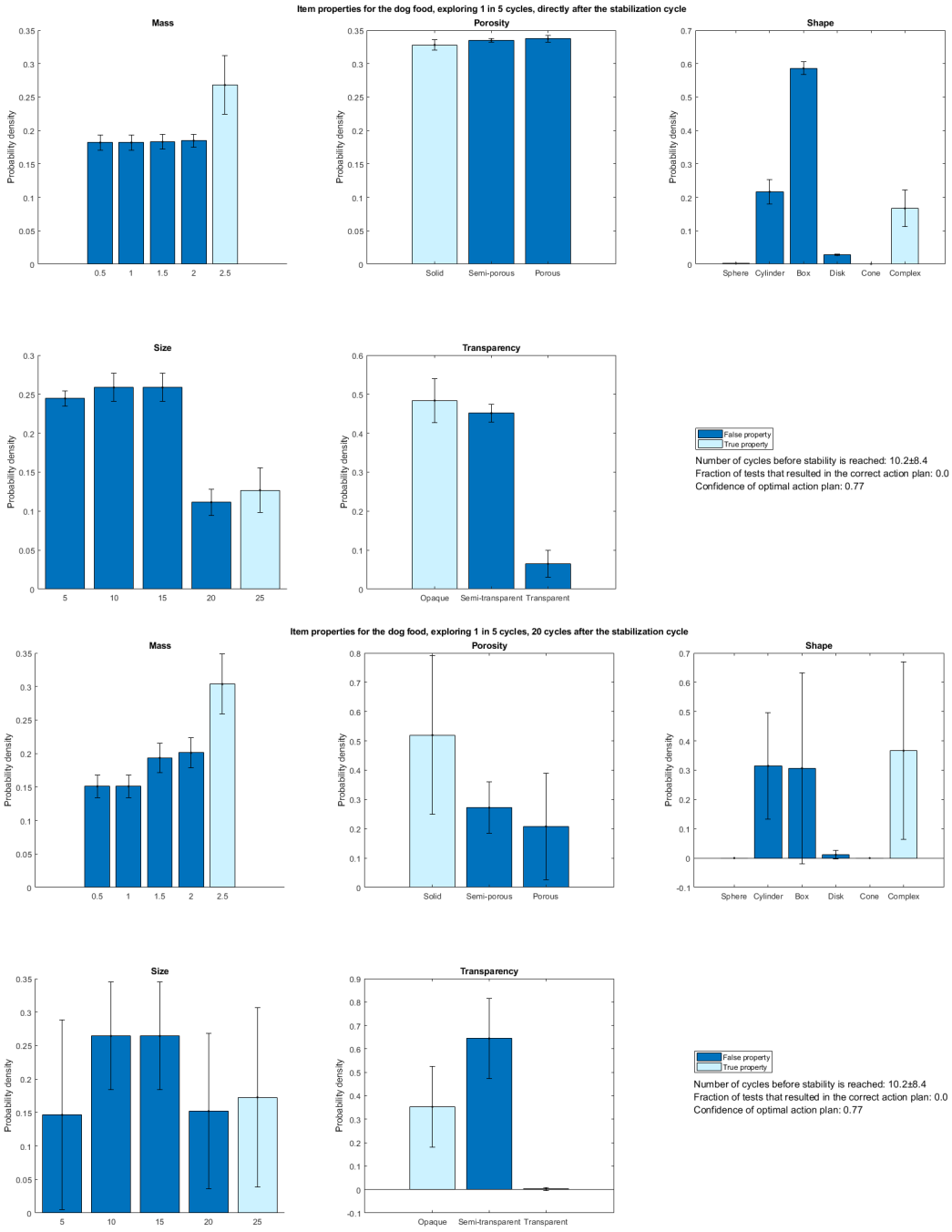


Figure F.30: The estimated item properties for the dog food, when RSIR explores once every 5 grasping cycles.

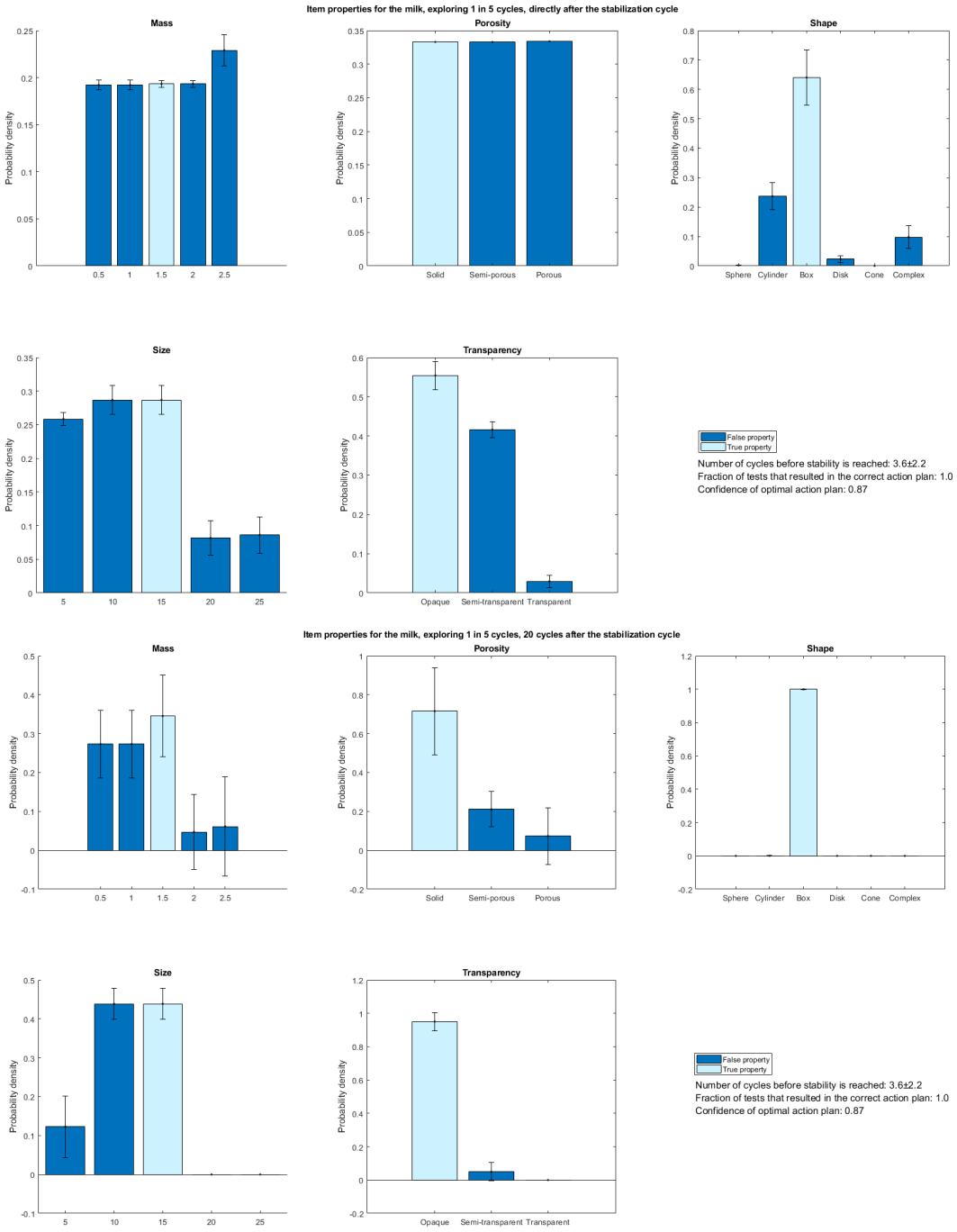


Figure F.31: The estimated item properties for the milk, when RSIR explores once every 5 grasping cycles.

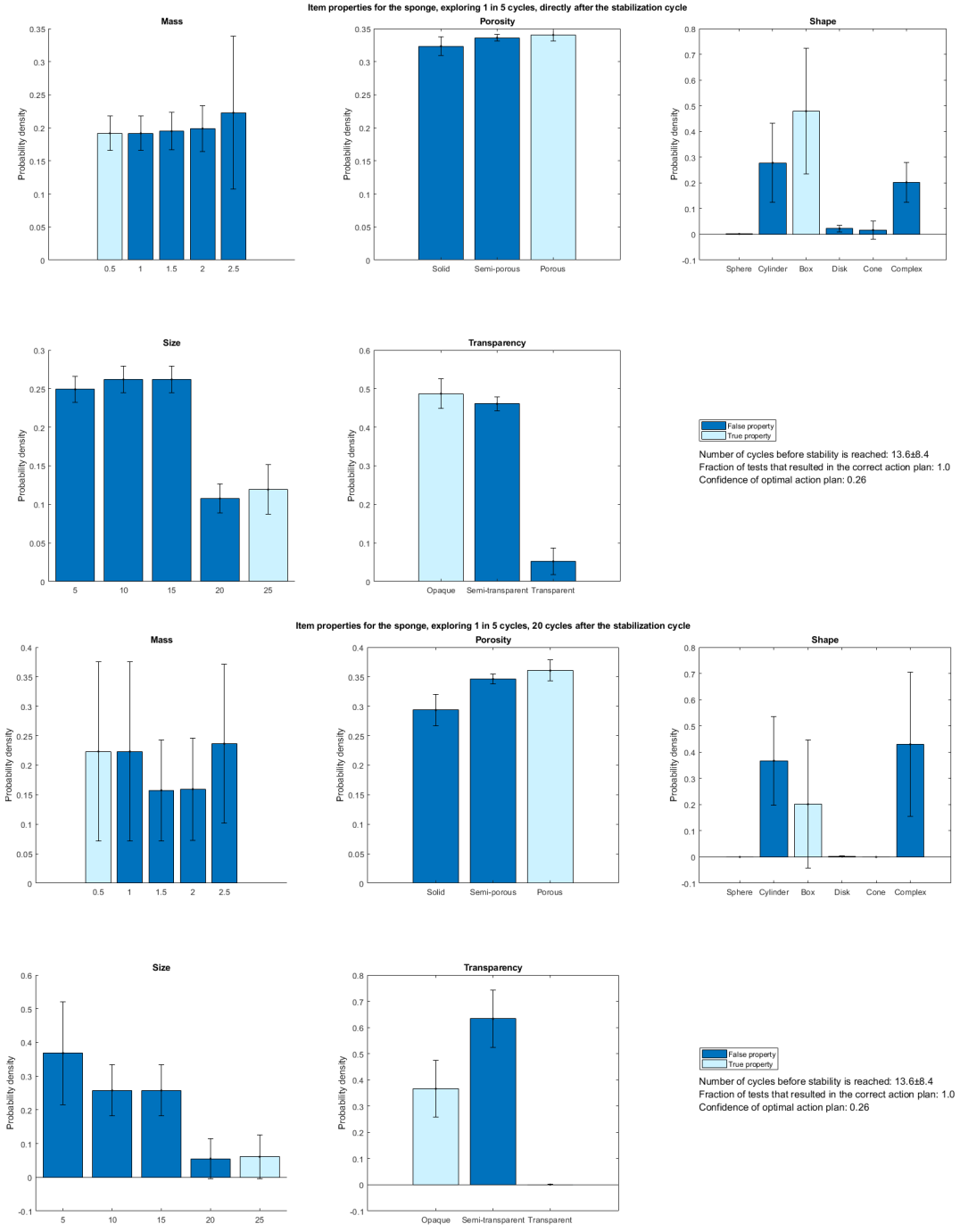


Figure F.32: The estimated item properties for the sponge, when RSIR explores once every 5 grasping cycles.

F.5.2. Exploring 1 in 2 cycles

This subsection presents the estimated item properties for the evaluation of RSIR when it explores once every 2 grasping cycles. The properties are presented in Figures F.33 through F.35.

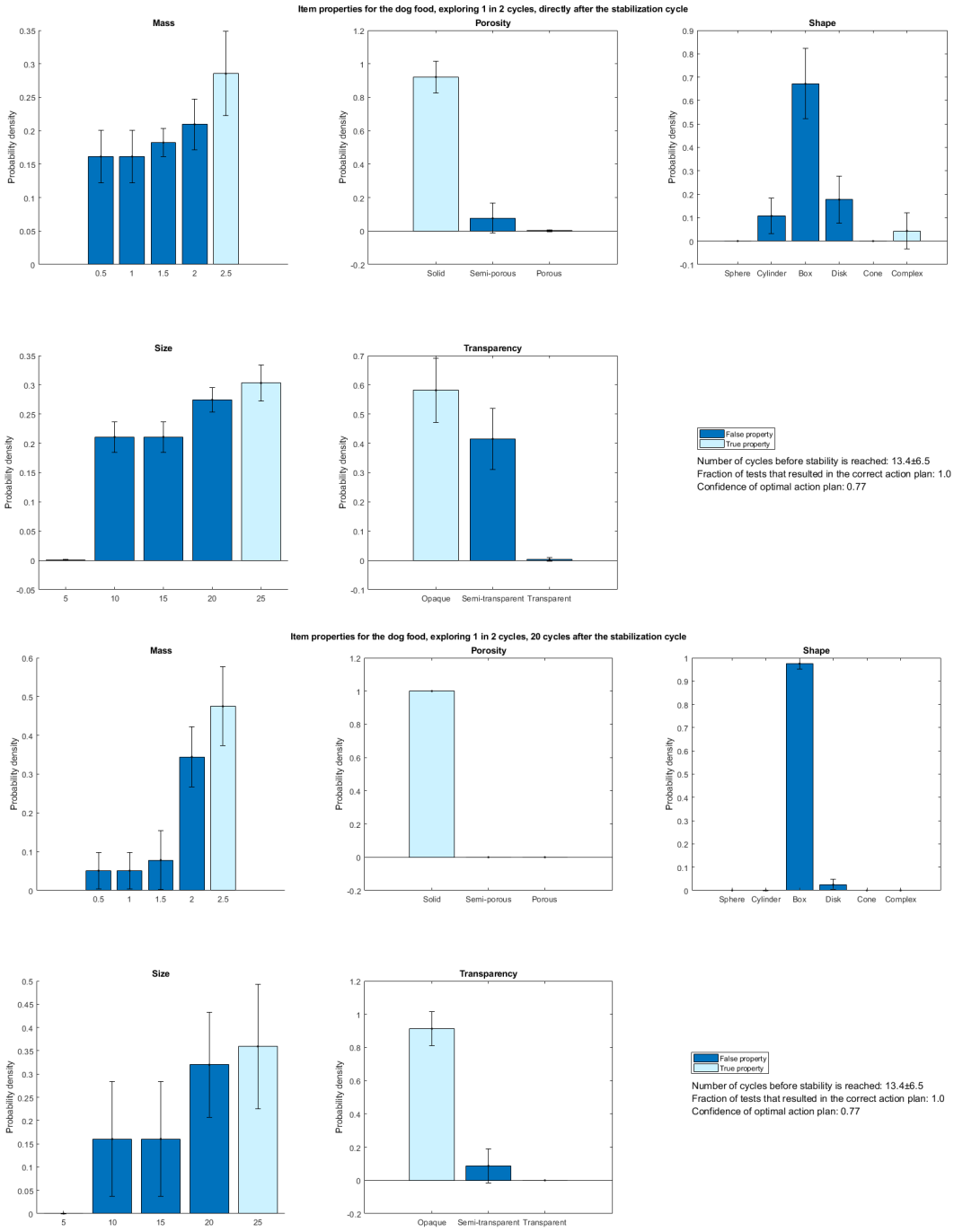


Figure F.33: The estimated item properties for the dog food, when RSIR explores once every 2 grasping cycles.

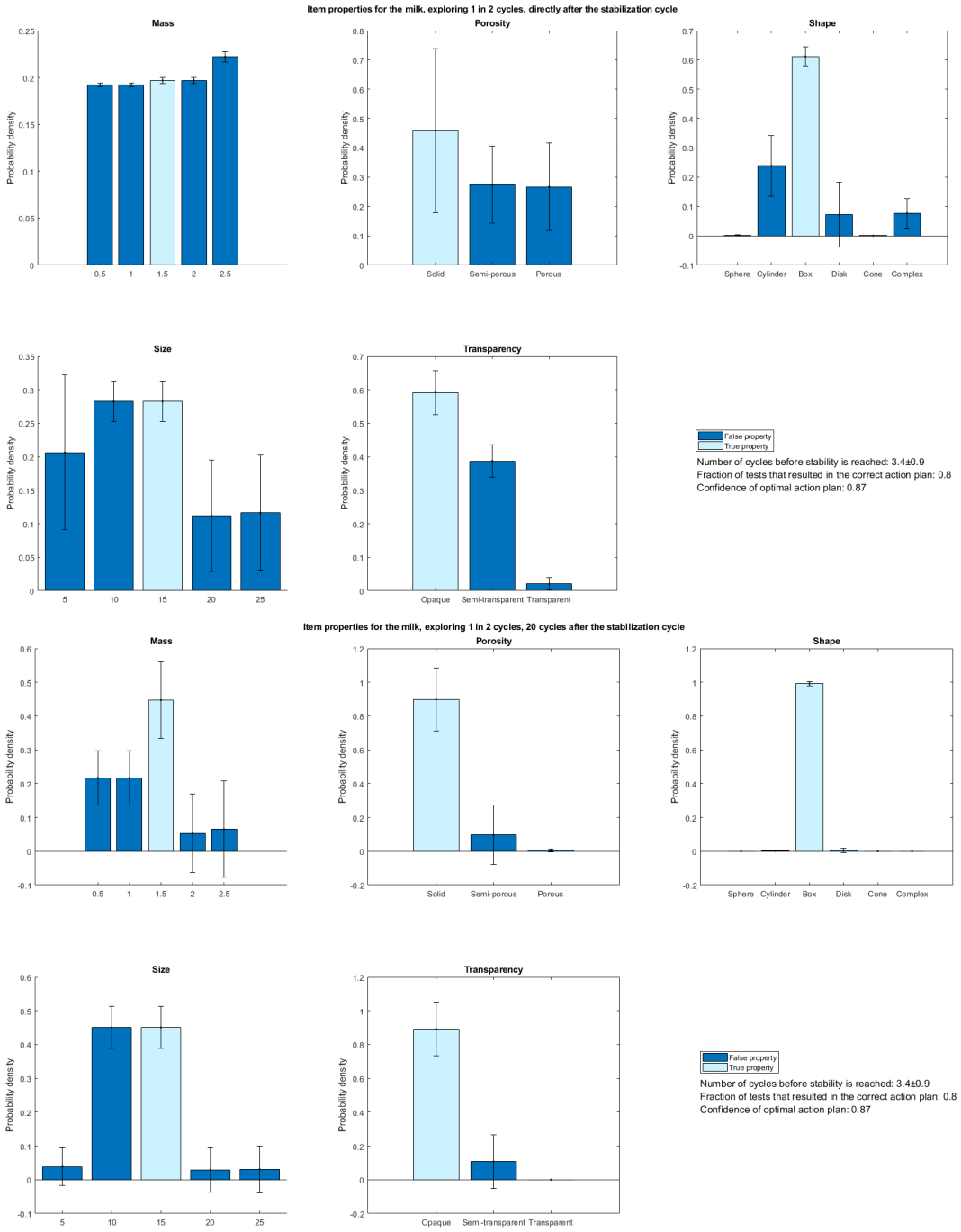


Figure F.34: The estimated item properties for the milk, when RSIR explores once every 2 grasping cycles.

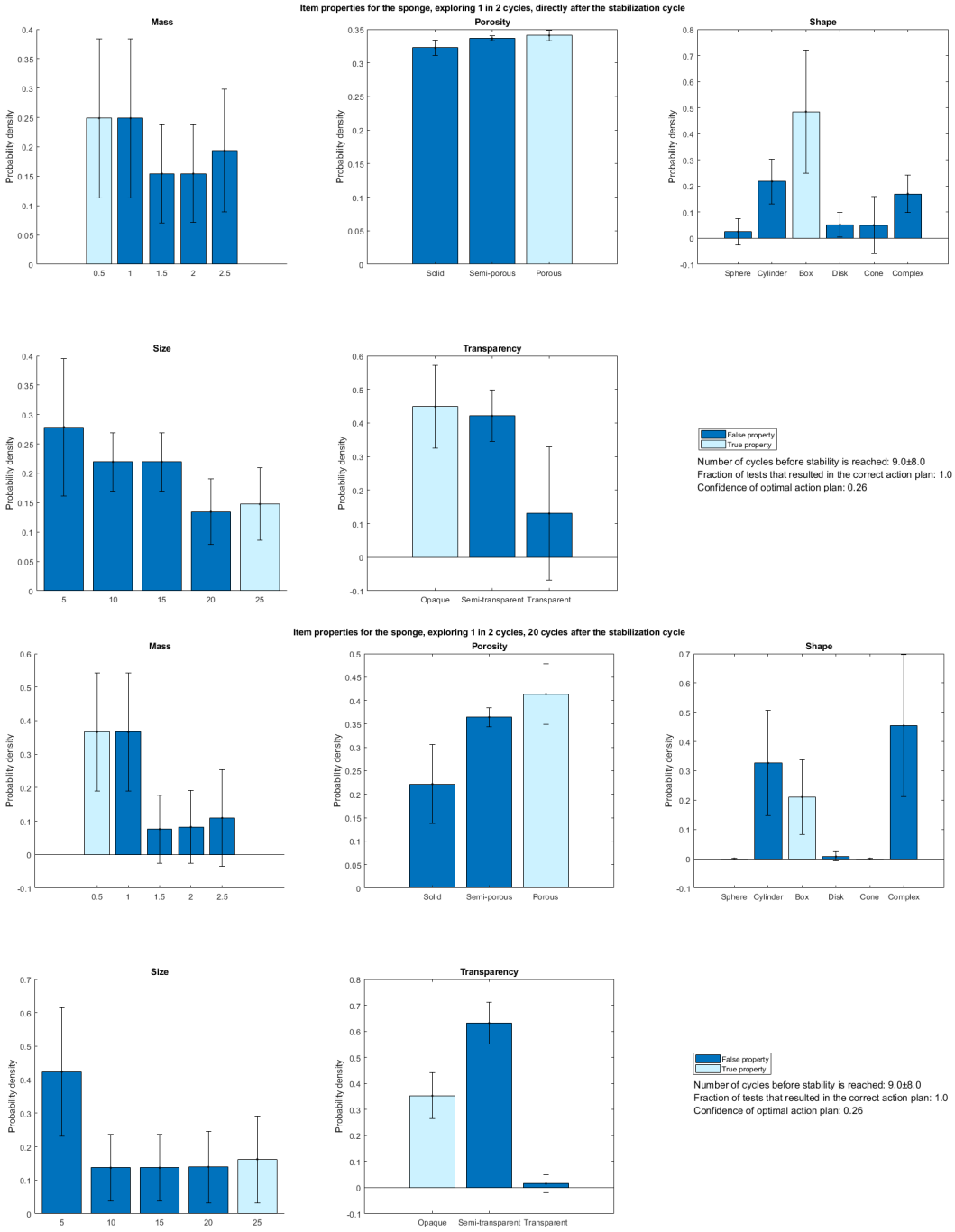


Figure F.35: The estimated item properties for the sponge, when RSIR explores once every 2 grasping cycles.

F.6. Estimated Item Properties for Robustness

This section presents the estimated item properties for the evaluation of RSIR when it has noisy knowledge about its capabilities and when it uses explorations.

F.6.1. Exploring 1 in 5 cycles

This subsection presents the estimated item properties for the evaluation of RSIR hen it has noisy knowledge about its capabilities and when it explores every 5 grasping cycles. The properties are presented in Figures F.36 through F.38.

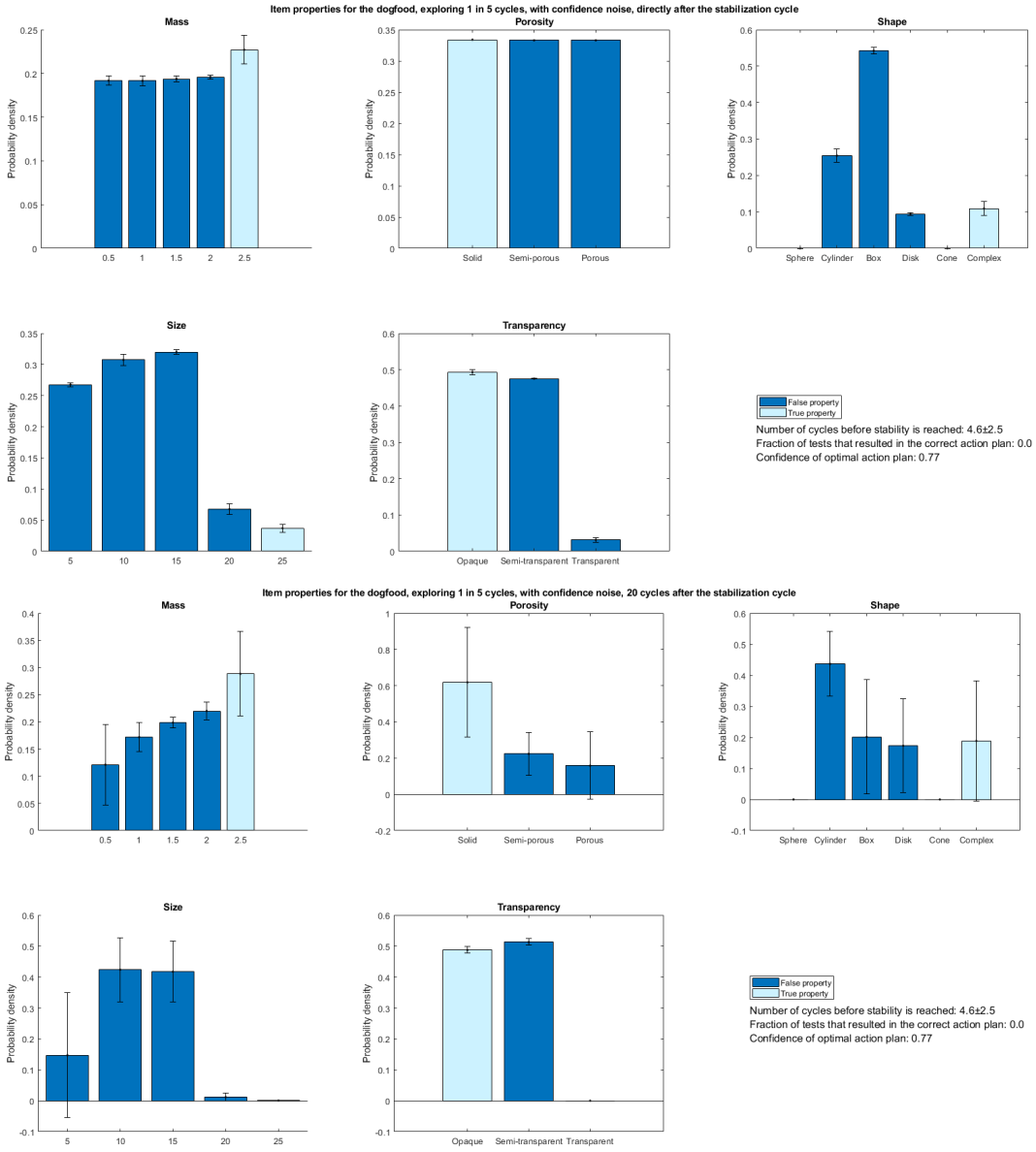


Figure F.36: The estimated item properties for the dog food, when RSIR has noisy knowledge about its capabilities and explores once every 5 grasping cycles.

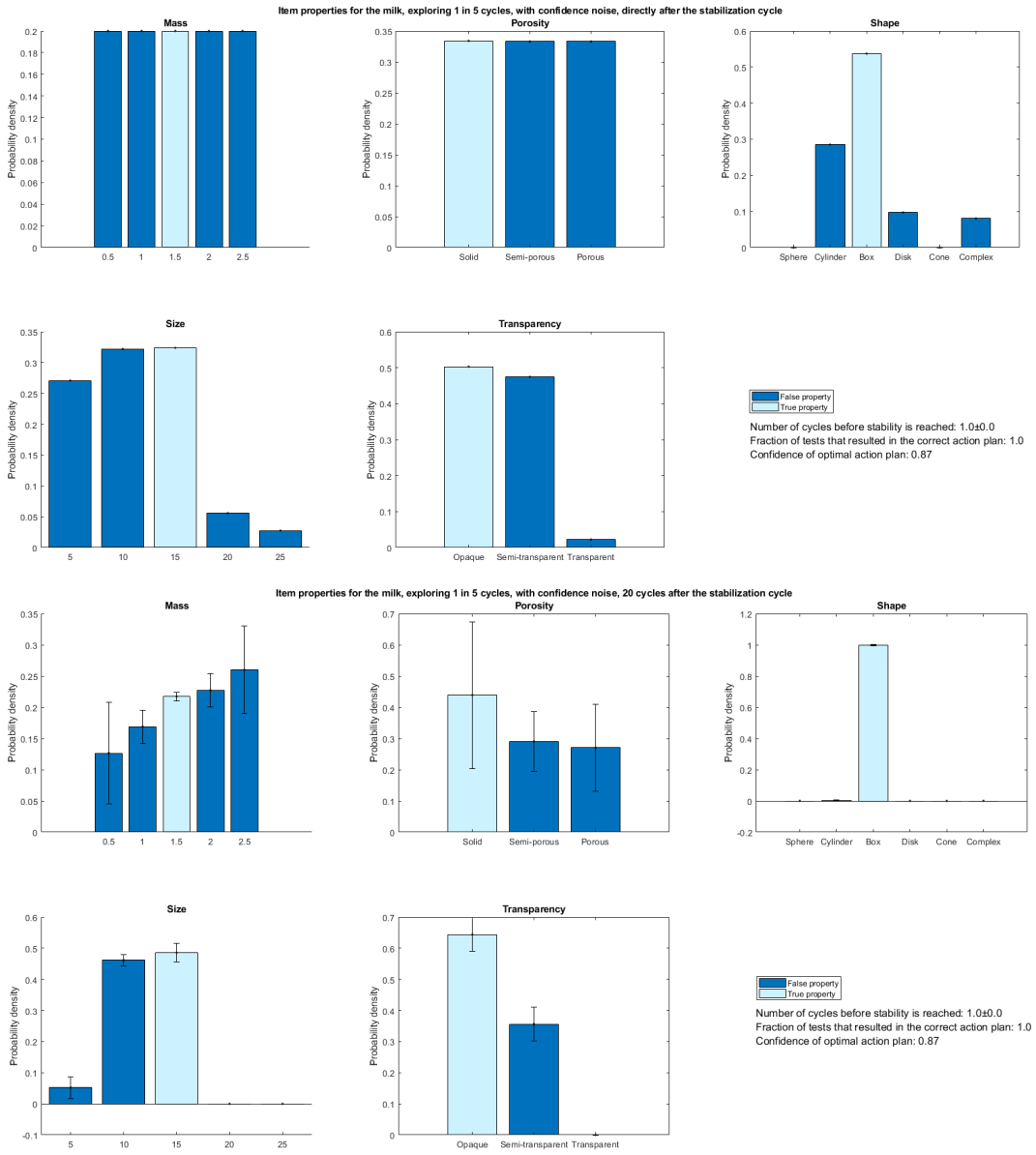


Figure F.37: The estimated item properties for the milk, when RSIR has noisy knowledge about its capabilities and explores once every 5 grasping cycles.

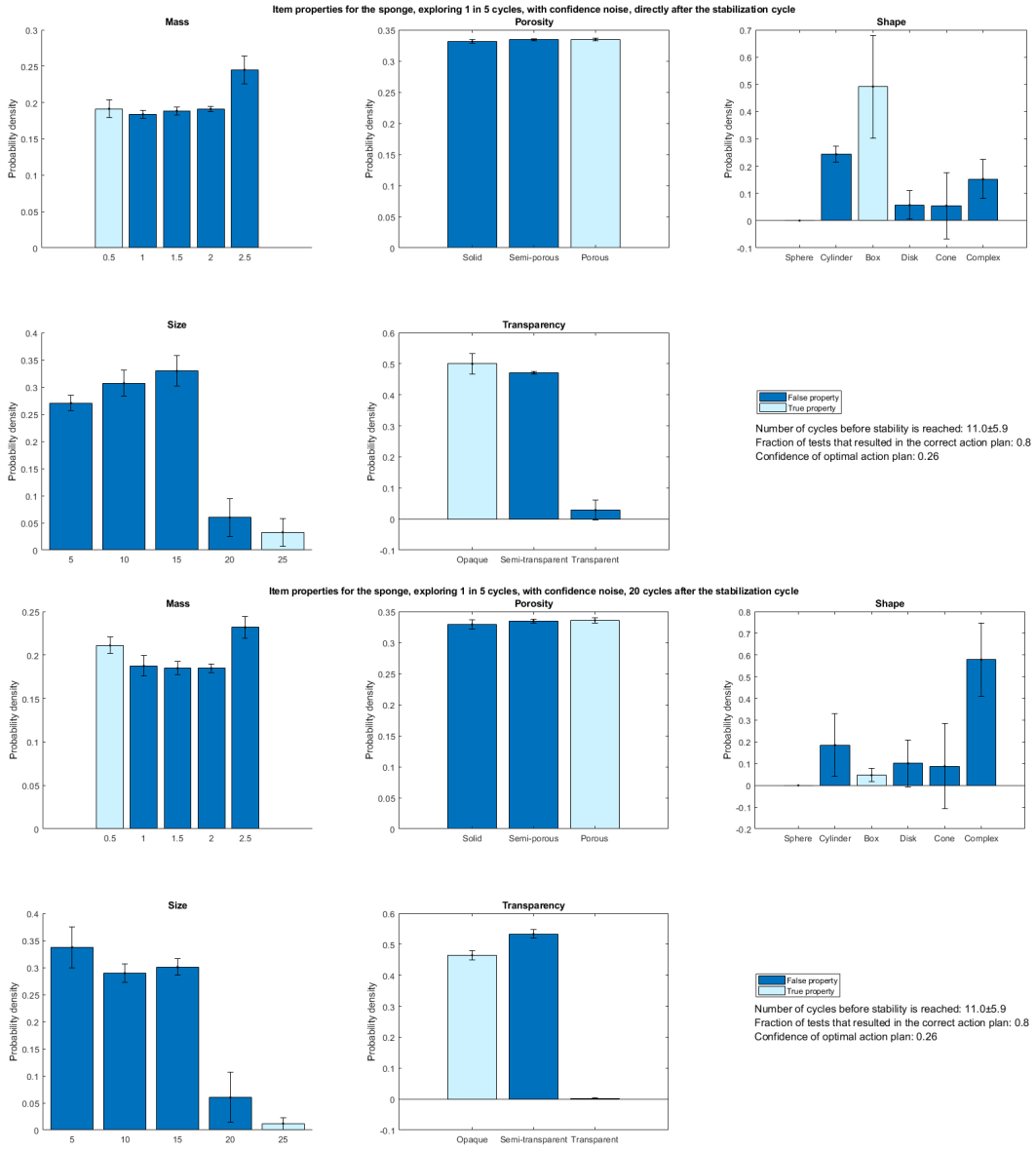


Figure F.38: The estimated item properties for the sponge, when RSIR has noisy knowledge about its capabilities and explores once every 5 grasping cycles.

F.6.2. Exploring 1 in 2 cycles

This subsection presents the estimated item properties for the evaluation of RSIR hen it has noisy knowledge about its capabilities and when it explores every 2 grasping cycles. The properties are presented in Figures F.39 through F.41.

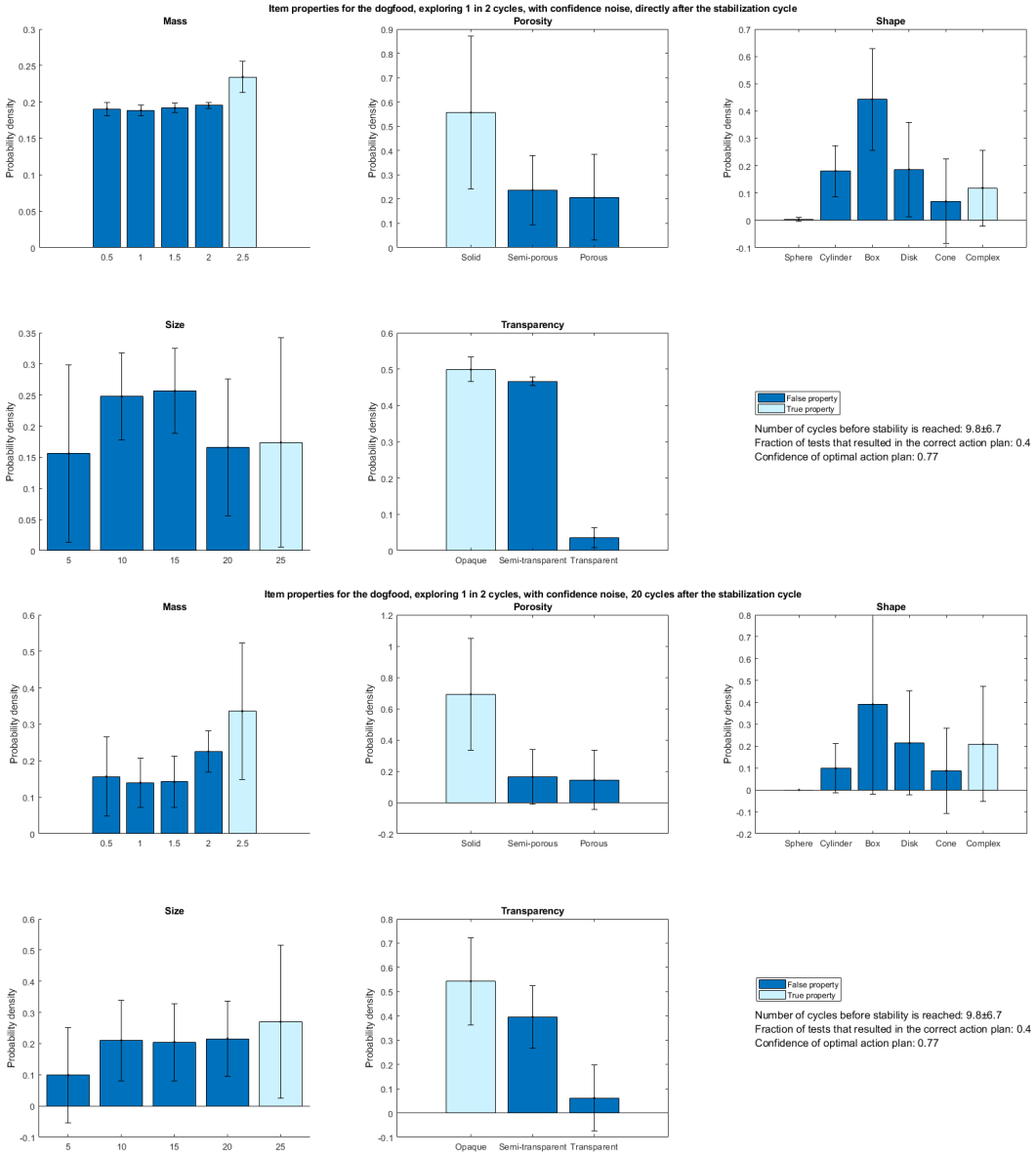


Figure F.39: The estimated item properties for the dog food, when RSIR has noisy knowledge about its capabilities and explores once every 2 grasping cycles.

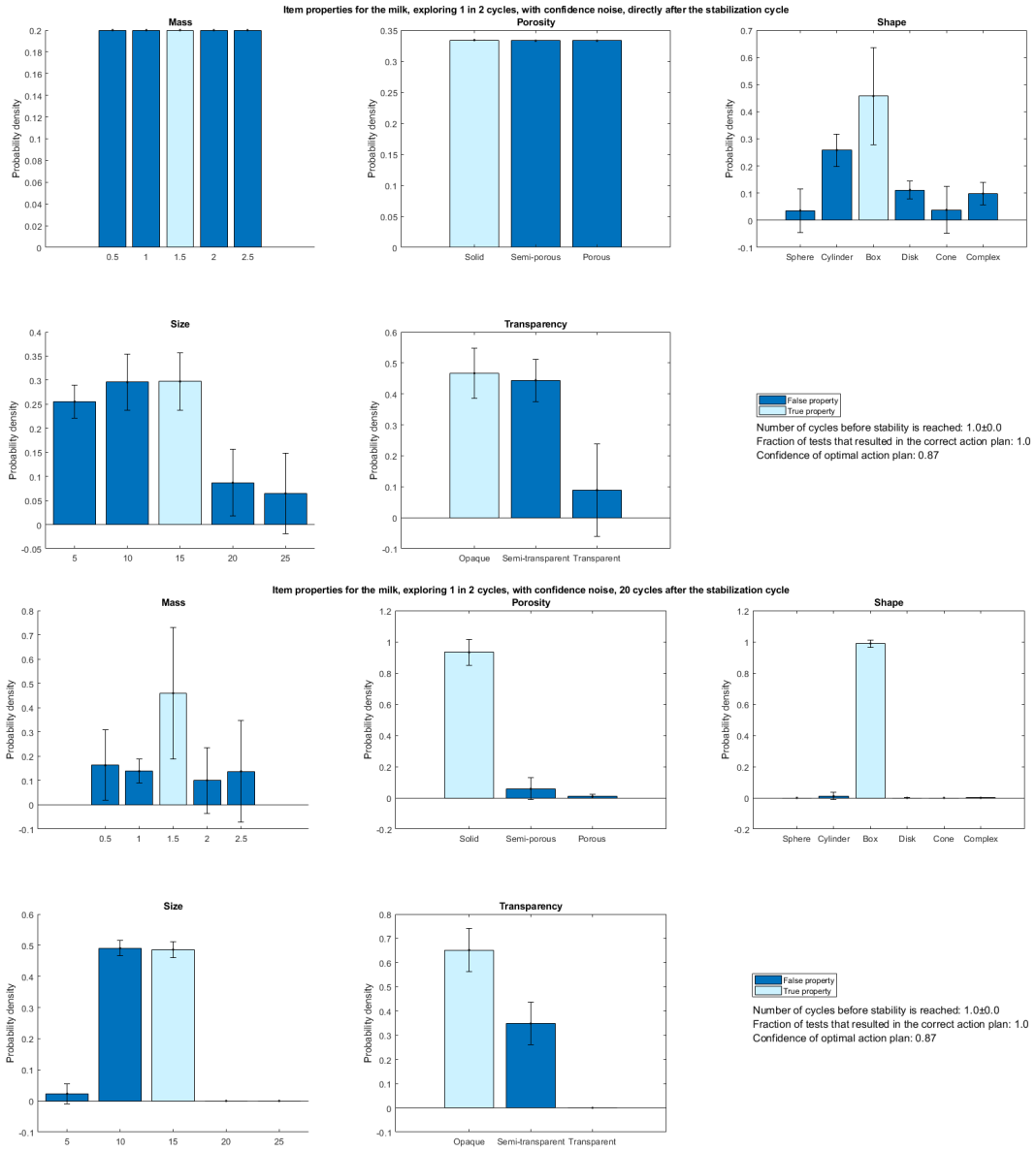


Figure F.40: The estimated item properties for the milk, when RSIR has noisy knowledge about its capabilities and explores once every 2 grasping cycles.

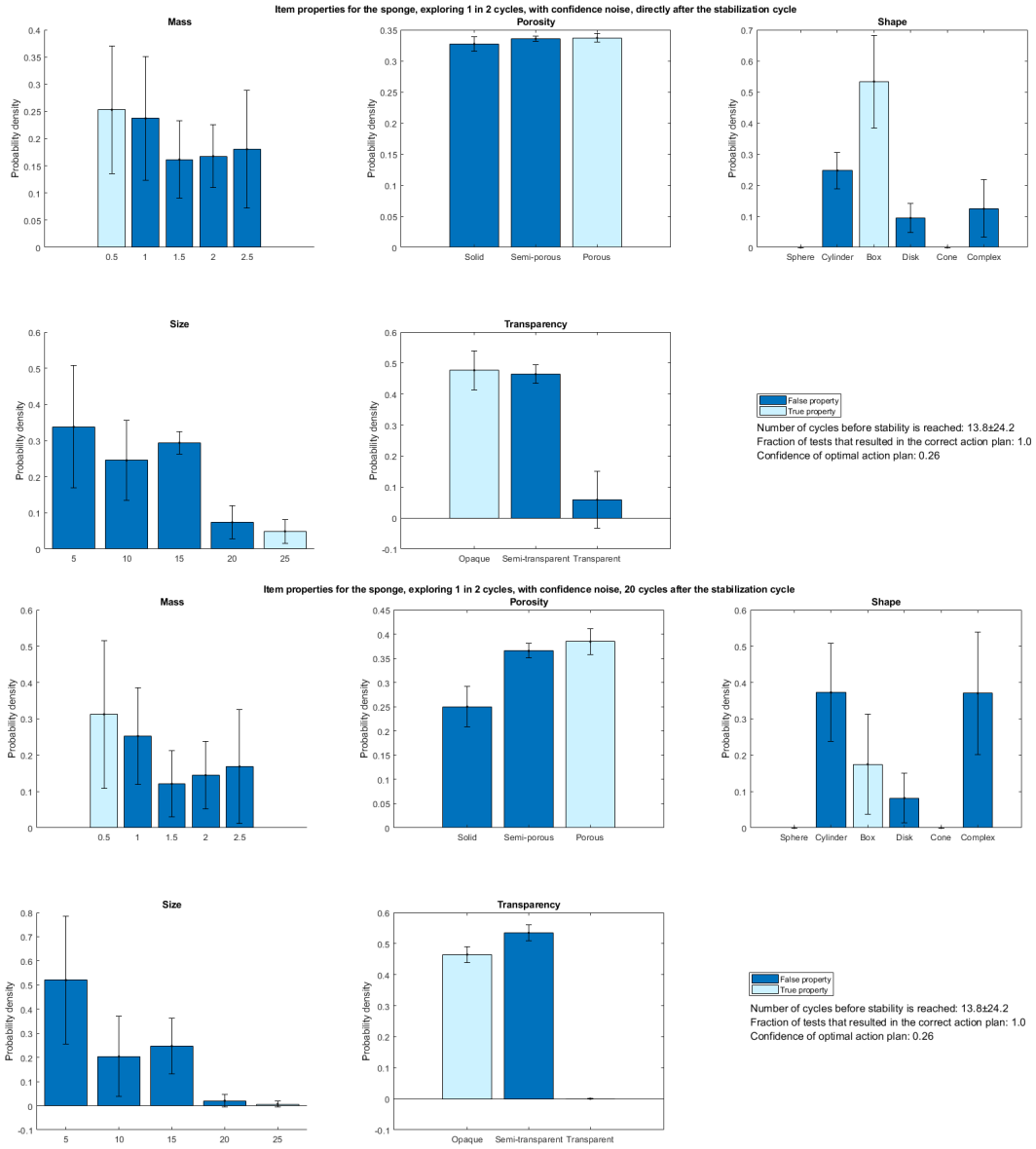


Figure F.41: The estimated item properties for the sponge, when RSIR has noisy knowledge about its capabilities and explores once every 2 grasping cycles.

Bibliography

[1] Barrett Hand. Retrieved 24 February 2020 from https://robots.ros.org/barrett-hand/.

[2] Prominent Users of GoodRelations, 2013. Retrieved 29 January 2020 from http://wiki.goodrelations-vocabulary.org/References.

[3] Application of KnowRob: Tidying up a kitchen, 2014. Retrieved 25 February 2020 from http://knowrob.org/blog/application_of_knowrob_tidying_up_a_kitchen.

[4] RoboEarth | A World Wide Web for Robots, 2014. Retrieved 24 January 2020 from http://roboearth.ethz.ch/.

[5] Open Product Data, 2020. Retrieved 26 February 2020 from http://product-open-data.com/home.

[6] Barcode Lookup | UPC, EAN & ISBN, 2020. Retrieved 26 February 2020 from https://www.barcodelookup.com/.

[7] EAN-Search.org, 2020. Retrieved 26 February 2020 from https://www.ean-search.org/.

[8] Open EAN Database - Datenbank und Produktbewertung, 2020. Retrieved 26 February 2020 from https://opengtindb.org/.

[9] Cobots Sweden AB. Robotiq 2-Finger 85 mm Gripper Kit for Universal Robots. Retrieved 24 February 2020 from https://cobots.se/produkt/2-finger-gripper-kit-for-universal-robots/.

[10] D.A. Abbink, M. Mulder, and E.R. Boer. Haptic shared control: Smoothly shifting control authority? *Cognition, Technology and Work*, 14(1):19–28, 2012. doi: 10.1007/s10111-011-0192-5.

[11] R. Ala, D.H. Kim, S.Y. Shin, C. Kim, and S.K. Park. A 3D-grasp Synthesis Algorithm to Grasp Unknown Objects based on Graspable Boundary and Convex Segments. *Information Sciences*, 295:91–106, 2015. doi: 10.1016/j.ins.2014.09.062.

[12] Allied Automation, Inc. Robotiq's 3-Finger Adaptive Robot Gripper: Versatile and Flexible, 2020. Retrieved 24 February 2020 from https://www.allied-automation.com/robotiqs-3-finger-adaptive-robot-gripper-versatile-and-flexible/.

[13] J.B. Alonso, R. Sanz, and M. Rodr. Ontology Engineering for the Autonomous Systems Domain. In *Communications in Computer and Information Science*, number 348, pages 263–277, 2013. doi: 10.1007/978-3-642-37186-8.

[14] DBpedia Association. DBpedia, 2019. Retrieved 21 February 2020 from https://wiki.dbpedia.org/.

[15] DPpedia Association. Ontology Classes. Retrieved 21 February 2020 from http://mappings.dbpedia.org/server/ontology/classes/.

[16] C. Barck-holst, M. Ralph, F. Holmar, and D. Kragic. Learning Grasping Affordance Using Probabilistic and Ontological Approaches. *International Conference on Advanced Robotics, 2009.*, pages 1–6, 2009.

[17] J. Baumgartl and D. Henrich. Fast Vision-based Grasp and Delivery Planning for Unknown Objects. In *7th German Conference on Robotics (ROBOTIK 2012)*, pages 1–5, 2012.

[18] J. Baumgartl, T. Werner, P. Kaminsky, and D. Henrich. A fast, GPU-based geometrical placement planner for unknown sensor-modelled objects and placement areas. In *IEEE International Conference on Robotics and Automation*, pages 1552–1559, 2014. doi: 10.1109/ICRA.2014.6907058.

[19] M. Beetz. Digital Twin Knowledge Bases - Knowledge Representation and Reasoning for Robotic Agents [PowerPoint presentation], July 2019. Retrieved 25 February 2020 from https://collegerama.tudelft.nl/Mediasite/Showcase/public/Presentation/3ea8e243a091499e8ea618edb68412a71d.

[20] M. Beetz, L. Mösenlechner, and M. Tenorth. CRAM - A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1012–1017, 2010. doi: 10.1109/IROS.2010.5650146.

[21] M. Beetz, D. Bessler, A. Haidu, M. Pomarlan, A.K. Bozcuoglu, and G. Bartels. Know Rob 2.0 - A 2nd Generation Knowledge Processing Framework for Cognition-Enabled Robotic Agents. In *IEEE International Conference on Robotics and Automation*, pages 512–519, 2018. doi: 10.1109/ICRA.2018.8460964.

[22] L. Berscheid, P. Meißner, and T. Kröger. Robot Learning of Shifting Objects for Grasping in Cluttered Environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 612–618, 2019. doi: 10.1109/IROS40897.2019.8968042.

[23] A. Bhat. A Soft and Bio-inspired Prosthesis with Tactile Feedback. Master's Thesis, Carnegie Mellon University, 2017.

[24] R. Bijl and J.J. Kostelijk. System/Subsystem Specification SSS_FM_GTP (Goods to Picker), 2019. Retrieved from Vanderlande's intranet.

[25] S. Brahmbhatt, A. Handa, J. Hays, and D. Fox. ContactGrasp: Functional Multi-finger Grasp Synthesis from Contact. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019. doi: 10.1109/IROS40897.2019.8967960.

[26] P. Bricker. Ontological commitment, 2016. Retrieved 26 February 2020 from https://plato.stanford.edu/archives/win2016/entries/ontological-commitment/.

[27] Festo BV. Parallel long-stroke gripper HGPL. Retrieved 24 February 2020 from https://www.festo.com/cms/nl_nl/67735.htm.

[28] S. Chitta, D. Hershberger, A. Pooley, D. Coleman, M. Gorner, F. Suarez, and M. Lautman. MoveIt Tutorials - moveit_tutorials Melodic documentation, 2020. Retrieved 24 February 2020 from https://ros-planning.github.io/moveit_tutorials/.

[29] C. Choi, W. Schwarting, J. Delpreto, and D. Rus. Learning Object Grasping for Soft Robot Hands. *IEEE Robotics and Automation Letters*, 3(3):2370–2377, 2018. doi: 10.1109/LRA.2018.2810544.

[30] C.H. Corbato, M. Bharatheesha, J. Van Egmond, J. Ju, and M. Wisse. Integrating Different Levels of Automation: Lessons from Winning the Amazon Robotics Challenge 2016. *IEEE Transactions on Industrial Informatics*, 14(11):4916–4926, 2018. doi: 10.1109/TII.2018.2800744.

[31] Doiq Corporation. SCHUNK 0340008. Retrieved 24 February 2020 from https://www.doigcorp.com/SCHUNK-0340008.html.

[32] G. Costa Jutglar. Integration of building product data with BIM modelling: a semantic-based product catalogue and rule checking system. 2017. Retrieved 24 February 2020 from https://www.tesisenred.net/handle/10803/450865.

[33] Interantional Barcodes Database. Barcodes Database, 2020. Retrieved 26 February 2020 from https://barcodesdatabase.org/.

[34] R. Detry, C.H. Ek, M. Madry, and D. Kragic. Learning a Dictionary of Prototypical Grasp-Predicting Parts from Grasping Experience. In *IEEE International Conference on Robotics and Automation*, pages 601–608, 2013. doi: 10.1109/ICRA.2013.6630635.

[35] Vanderlande Research & Development. Cause-effect diagram for risks to Key Performance Indicators (KPIs) of FM-GtP, . Retrieved from Vanderlande's intranet.

[36] Vanderlande Research & Development. GS1 Item Handleability Test Results_v13, . Retrieved from Vanderlande's intranet.

[37] M. Diab, M. Pomarlan, D. Beßler, A. Akbari, J. Rosell, J. Bateman, and M. Beetz. An Ontology for Failure Interpretation in Automated Planning and Execution. *Advances in Intelligent Systems and Computing*, 1092(1):381–390, 2019. doi: 10.1007/978-3-030-35990-4.

[38] EAN Database. Ean Database | EanDatabase | Legal Barcode | Database Barcode, 2020. Retrieved 26 February 2020 from `https://eandatabase.com/search.php`.

[39] C. Eppner and O. Brock. Grasping Unknown Objects by Exploiting Shape Adaptability and Environmental Constraints. In *IEEE International Conference on Intelligent Robots and Systems*, pages 4000–4006, 2013. doi: 10.1109/IROS.2013.6696928.

[40] A. Fayaz, A. Niedzwiecki, and G. Kazhoyan. Zero prerequisites demo tutorial: Simple fetch and place, 2020. Retrieved 21 February 2020 from `http://www.cram-system.org/tutorials/demo/fetch_and_place`.

[41] T. Feix, J. Romero, H.B. Schmiedmayer, A.M. Dollar, and D. Kragic. The GRASP Taxonomy of Human Grasp Types. *IEEE Transactions on Human-Machine Systems*, 46(1):66–77, 2016. doi: 10.1109/THMS.2015.2470657.

[42] M. Fernández, A. Gómez-Pérez, and N. Juristo. METHONTOLOGY: From Ontological Art Towards Ontological Engineering. In *AAAI Technical Report SS-97-06*, pages 33–40, AAAI Technical Report SS-97-06, 1997.

[43] F. Flemisch, D. Abbink, M. Itoh, M. P. Pacaux-Lemoine, and G. Weßel. Shared control is the sharp end of cooperation: Towards a common framework of joint action, shared control and human machine cooperation. volume 49, pages 72–77, 2016. doi: 10.1016/j.ifacol.2016.10.464.

[44] Max Planck Institute for Informatics. YAGO: A High-Quality Knowledge Base, 2018. Retrieved 21 February 2020 from `https://www.mpi-inf.mpg.de/departments/databases-and-information-systems/research/yago-naga/yago/`.

[45] Freepik. Robotic Arm, free icon, 2020. `flaticon.com/free-icon/robotic-arm-_17969`.

[46] S. Gifis. Mobile Shoppers' Top 5 Demands of E-Commerce Brands, 2018. Retrieved 24 February 2020 from `https://www.mytotalretail.com/article/mobile-shoppers-top-5-demands-of-e-commerce-brands/`.

[47] GS1. Search by GTIN | GEPIR, 2020. Retrieved 26 February 2020 from `https://gepir.gs1.org/index.php/search-by-gtin`.

[48] R. Gupta and M.J. Kochenderfer. Common Sense Data Acquisition for Indoor Mobile Robots. In *Proceedings of the National Conference on Artificial Intelligence*, volume 94041, pages 605–610, 2004.

[49] T. Haidegger, M. Barreto, P.J.S. Goncalves, M.K. Habib, S.V. Ragavan, H. Li, A. Vaccarella, R. Perrone, and E. Prestes. Robot Ontologies for Sensor- and Image-Guided Surgery. In *2013 IEEE International Symposium on Robotic and Sensors Environments, Proceedings*, pages 19–24, 2013. doi: 10.1109/ROSE.2013.6698412.

[50] K. Hang, J.A. Stork, and D. Kragic. Hierarchical Fingertip Space for Multi-Fingered Precision Grasping. In *IEEE International Conference on Intelligent Robots and Systems*, pages 1641–1648, 2014. doi: 10.1109/IROS.2014.6942775.

[51] K. Hang, J.A. Haustein, M. Li, A. Billard, C. Smith, and D. Kragic. On the Evolution of Fingertip Grasping Manifolds. In *IEEE International Conference on Robotics and Automation*, pages 2022–2029, 2016. doi: 10.1109/ICRA.2016.7487349.

[52] S. Hasegawa, K. Wada, Y. Niitani, K. Okada, and M. Inaba. A Three-Fingered Hand with a Suction Gripping System for Picking Various Objects in Cluttered Narrow Space. In *IEEE International Conference on Intelligent Robots and Systems*, pages 1164–1171, 2017. doi: 10.1109/IROS.2017.8202288.

[53] C. Hellmann, A. Bajrami, and W. Kraus. Enhancing a robot gripper with haptic perception for risk mitigation in physical human robot interaction. In *IEEE World Haptics Conference*, number 7, pages 253–258, 2019. doi: 10.1109/WHC.2019.8816109.

[54] M. Hepp. GoodRelations Language Reference, 2011. Retrieved 21 February 2020 from http://www.heppnetz.de/ontologies/goodrelations/v1.

[55] C. Hernández, J. Bermejo-Alonso, and R. Sanz. A Self-Adaptation Framework Based on Functional Knowledge for Augmented Autonomy in Robots. *Integrated Computer-Aided Engineering*, 25(2):157–172, 2018. doi: 10.3233/ICA-180565.

[56] U. Hillenbrand and M.A. Roa. Transferring Functional Grasps Through Contact Warping and Local Replanning. In *IEEE International Conference on Intelligent Robots and Systems*, pages 2963–2970, 2012. doi: 10.1109/IROS.2012.6385989.

[57] M. Huber, M. Rickert, A. Knoll, T. Brandt, and S. Glasauer. Human-robot interaction in handing-over tasks. *Proceedings of the 17th IEEE International Symposium on Robot and Human Interactive Communication, RO-MAN*, pages 107–112, 2008. doi: 10.1109/ROMAN.2008.4600651.

[58] J. Hughes, U. Culha, F. Giardina, F. Guenther, A. Rosendo, and F. Iida. Soft Manipulators and Grippers: A Review. *Frontiers Robotics AI*, 3(69):1–12, 2016. doi: 10.3389/frobt.2016.00069.

[59] Standards Association IEEE. *IEEE Standard Ontologies for Robotics and Automation IEEE Robotics and Automation Society*. 2015.

[60] RobotShop inc. GearWurx 2 Finger Gripper, 2020. Retrieved 24 February 2020 from https://www.robotshop.com/en/gearwurx-2-finger-gripper.html.

[61] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke, and S. Levine. QT-Opt: Scalable Deep Reinforcement Learning for Vision-Based Robotic Manipulation. In *Proceedings of The 2nd Conference on Robot Learning*, 2018.

[62] D. Kappler, F. Meier, J. Issac, J. Mainprice, C.G. Cifuentes, M. Wuthrich, V. Berenz, S. Schaal, N. Ratliff, and J. Bohg. Real-time Perception Meets Reactive Motion Generation. *IEEE Robotics and Automation Letters*, 3(3):1864–1871, 2018. doi: 10.1109/LRA.2018.2795645.

[63] SCHUNK GmbH & Co. KG. Parallel gripper (Stroke per finger > 50 mm), 2020. Retrieved 21 February 2020 from https://schunk.com/nl_en/gripping-systems/category/gripping-systems/schunk-grippers/parallel-gripper/.

[64] E. Klingbeil, D. Rao, B. Carpenter, V. Ganapathi, A.Y. Ng, and O. Khatib. Grasping with Application to an Autonomous Checkout Robot. In *IEEE International Conference on Robotics and Automation*, pages 2837–2844, 2011. doi: 10.1109/ICRA.2011.5980287.

[65] H. Knublauch. SHACL and OWL Compared, 2017. Retrieved 21 February 2020 from https://spinrdf.org/shacl-and-owl.html.

[66] S. Koos, A. Cully, and J.B. Mouret. Fast Damage Recovery in Robotics with the T-resilience Algorithm. *International Journal of Robotics Research*, 32(14):1700–1723, 2013. doi: 10.1177/0278364913499192.

[67] M. Kopicki, R. Detry, F. Schmidt, C. Borst, R. Stolkin, and J.L. Wyatt. Learning Dexterous Grasps that Generalise to Novel Objects by Combining Hand and Contact Models. In *Proceedings - IEEE International Conference on Robotics and Automation*, pages 5358–5365, 2014. doi: 10.1109/ICRA.2014.6907647.

[68] M. Kopicki, M. Adjigble, R. Stolkin, A. Leonardis, J.L. Wyatt, and R. Detry. One Shot Learning and Generation of Dexterous Grasps for Novel Objects. *International Journal of Robotics Research*, 35(8):959–976, 2015. doi: 10.1177/0278364915594244.

[69] M. Kopicki, M. Adjigble, R. Stolkin, A. Leonardis, J.L. Wyatt, and R. Detry. One Shot Learning and Generation of Dexterous Grasps for Novel Objects, 2015. Retrieved 24 February 2020 from https://www.youtube.com/watch?v=hjM4wCEzI4U.

[70] K. Korane. Going soft on grippers, 2018. Retrieved 24 February 2020 from https://www.pneumatictips.com/going-soft-on-grippers/.

[71] D. Kortenkamp, R.P. Bonasso, D. Ryan, and D. Schreckenghost. Traded Control with Autonomous Robots as Mixed Initiative Interaction. In *AAAI Technical Report SS-97-04*, pages 89–94, 1997.

[72] A. Kravstov. Iceclog: Manual for Open Icecat JSON Product Requests, 2018. Retrieved 26 February 2020 from `https://iceclog.com/manual-for-icecat-json-product-requests/`.

[73] Q. Lei, G. Chen, and M. Wisse. Fast Grasping of Unknown Objects Using Principal Component Analysis. *AIP Advances*, 7(9), 2017. doi: 10.1063/1.4991996.

[74] DTAI Research Group KU Leuven. Introduction. - ProbLog: Probabilistic Programming, 2017. Retrieved 24 February 2020 from `https://dtai.cs.kuleuven.be/problog/index.html`.

[75] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen. Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection. *International Journal of Robotics Research*, 37(4-5):421–436, 2018. doi: 10.1177/0278364917710318.

[76] R. Lewis and P. Weezepoel. Market research grippers [PowerPoint presentation]. Retrieved from Vanderlande's intranet.

[77] Lexico. Ontology | Meaning of Ontology by Lexico, 2020. Retrieved 24 February 2020 from `https://www.lexico.com/definition/ontology`.

[78] Build-A-Bear Workshop UK Limited. Online Exclusive Strike a Pose Teddy, 2020. Retrieved 21 February 2020 from `https://www.buildabear.co.uk/online-exclusive-strike-a-pose-teddy/427991.html?cgid=collections-occasions-christmas`.

[79] V. Lippiello. Grasp the Possibilities: Anthropomorphic Grasp Synthesis Based on the Object Dynamic Properties. *IEEE Robotics and Automation Magazine*, 22(4):69–79, 2015. doi: 10.1109/MRA.2015.2394711.

[80] V. Lippiello, F. Ruggiero, B. Siciliano, and L. Villani. Preshaped Visual Grasp of Unknown Objects with a Multi-Fingered Hand. In *IEEE International Conference on Intelligent Robots and Systems*, pages 5894–5899, 2010.

[81] C. Liu, B. Fang, F. Sun, X. Li, and W. Huang. Learning to Grasp Familiar Objects Based on Experience and Objects' Shape Affordance. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019. doi: 10.1109/tsmc.2019.2901955.

[82] Q. Lu and T. Hermans. Modeling Grasp Type Improves Learning-Based Grasp Planning. *IEEE Robotics and Automation Letters*, 4(2):784–791, 2019. doi: 10.1109/LRA.2019.2893410.

[83] Q. Lu, K. Chenna, B. Sundaralingam, and T. Hermans. Planning Multi-Fingered Grasps as Probabilistic Inference in a Learned Deep Network. In *International Symposium on Robotics Research*, 2017.

[84] S. Luo, J. Bimbo, R. Dahiya, and H. Liu. Robotic tactile perception of object properties: A review. In *Mechatronics*, volume 48, pages 54–67, 2017. doi: 10.1016/j.mechatronics.2017.11.002.

[85] R.R. Ma and A.M. Dollar. On Dexterity and Dexterous Manipulation. In *IEEE 15th International Conference on Advanced Robotics: New Boundaries for Robotics*, pages 1–7, 2011. doi: 10.1109/ICAR.2011.6088576.

[86] J. Mahler, J. Liang, S. Niyaz, M. Laskey, R. Doan, X. Liu, J. Aparicio, and K. Goldberg. Dex-Net 2.0: Deep Learning to Plan Robust Grasps with Synthetic Point Clouds and Analytic Grasp Metrics. In *Robotics: Science and Systems XIII*, 2017. doi: 10.15607/RSS.2017.XIII.058.

[87] P. Maier. *Modelling and Reasoning about Robot Knowledge with OWL Ontologies*. PhD thesis, Technische Universität München, 2018.

[88] I. Malavolta, G.A. Lewis, B. Schmerl, P. Lago, and D. Garlan. How do you Architect your Robots? State of the Practice and Guidelines for ROS-based System. *Proceedings of the 42nd International Conference on Software Engineering: Software Engineering in Practice*, (1), 2020.

[89] A. Maldonado, U. Klank, and M. Beetz. Robotic Grasping of Unmodeled Objects Using Time-of-Flight Range Data and Finger Torque Information. In *IEEE International Conference on Intelligent Robots and Systems*, pages 2586–2591, 2010. doi: 10.1109/IROS.2010.5649185.

[90] A.T. Miller, S. Knoop, H.I. Christensen, and P.K. Allen. Automatic Grasp Planning Using Shape Primitives. In *IEEE International Conference on Robotics and Automation*, pages 1824–1829, 2003. doi: 10.1109/robot.2003.1241860.

[91] A. Morales, E. Chinelalto, P.J. Sanz, A.P. Del Pobil, and A.H. Fagg. Learning to Predict Grasp Reliability for a Multifinger Robot Hand by Using Visual Features. In *Proceedings of the Eighth IASTED International Conference on Artificial Intelligence and Soft Computing*, pages 249–254, 2004.

[92] D. Morrison, J. Leitner, and P. Corke. Closing the Loop for Robotic Grasping: A Real-time, Generative Grasp Synthesis Approach. *Robotics: Science and Systems*, 2018. doi: 10.15607/rss.2018.xiv.021.

[93] D. Morrison, A. W. Tow, M. McTaggart, R. Smith, N. Kelly-Boxall, S. Wade-Mccue, J. Erskine, R. Grinover, A. Gurman, T. Hunn, D. Lee, A. Milan, T. Pham, G. Rallos, A. Razjigaev, T. Rowntree, K. Vijay, Z. Zhuang, C. Lehnert, I. Reid, P. Corke, and J. Leitner. Cartman: The Low-Cost Cartesian Manipulator that Won the Amazon Robotics Challenge. In *IEEE International Conference on Robotics and Automation*, pages 7757–7764, 2018. doi: 10.1109/ICRA.2018.8463191.

[94] M.A. Musen. The Protégé project: A look back and a look forward. *AI Matters. Association of Computing Machinery Specific Interest Group in Artificial Intelligence*, 1(4), 2015. doi: 10.1145/2557001.25757003.

[95] Y.C. Nakamura, D.M. Troniak, A. Rodriguez, M.T. Mason, and N.S. Pollard. The Complexities of Grasping in the Wild. In *IEEE-RAS International Conference on Humanoid Robotics*, pages 233–240, 2017. doi: 10.1109/HUMANOIDS.2017.8246880.

[96] H. Nakawala, J. Lu, M. Barreto, and J. Bermejo-alonso. Towards a Robot Task Ontology Standard. In *Proceedings of the ASME 2017 International Manufacturing Science and Engineering Conference*, pages 1–10, 2017.

[97] GS1 Netherlands. GS1 Data Source: Guideline for choosing the right packaging type, 2018. Retrieved 24 February 2020 from `https://www.gs1.nl/sites/default/files/so_gs1das_guideline_for_choosing_the_right_packaging_type.pdf`.

[98] A. Nilsson, R. Muradore, K. Nilsson, and P. Fiorini. Ontology for Robotics: A Roadmap. In *2009 International Conference on Advanced Robotics*, pages 1–6, 2009.

[99] J. Norder. FM_GTP_URS_SIR, 2019. Retrieved from Vanderlande's intranet.

[100] K. Nørmark. Overview of the four main programming paradigms, 2013. Retrieved 24 February 2020 from `http://people.cs.aau.dk/~normark/prog3-03/html/notes/paradigms_themes-paradigm-overview-section.html#paradigms_imperative-paradigm-overview_title_1`.

[101] M. O'Connor. The Semantic Web Rule Language, 2009. Retrieved 5 March 2020 from `https://protege.stanford.edu/conference/2009/slides/SWRL2009ProtegeConference.pdf`.

[102] D. Pangercic, M. Tenorth, D. Jain, and M. Beetz. Combining Perception and Knowledge Processing for Everyday Manipulation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1065–1071, 2010. doi: 10.1109/IROS.2010.5651006.

[103] J. Pijnacker. Control Strategies for Grasping and Holding Deformable Items Using a Gripper on a Robotic Arm. Master's Thesis, Delft University of Technology, 2018.

[104] L. Pinto and A. Gupta. Supersizing Self-supervision - Learning to Grasp from 50K Tries and 700 Robot Hours. In *IEEE International Conference on Robotics and Automation*, pages 3406–3413, 2016.

[105] Cambridge University Press. ADAPTABILITY | meaning in the Cambridge English Dictionary, 2020. Retrieved 24 February 2020 from `https://dictionary.cambridge.org/dictionary/english/adaptability`.

[106] E. Prestes, S.R. Fiorini, and J. Carbonera. Core Ontology for Robotics and Automation, 2014.

[107] Francisco Ramos, Andrés S. Vázquez, Raúl Fernández, and Alberto Olivares-Alarcos. Ontology based design, control and programming of modular robots. *Integrated Computer-Aided Engineering*, 25(2): 173–192, 2018. doi: 10.3233/ICA-180569.

[108] M.A. Roa and R. Suárez. Computation of Independent Contact Regions for Grasping 3-D Objects. *IEEE Transactions on Robotics*, 25(4):839–850, 2009. doi: 10.1109/TRO.2009.2020351.

[109] M.A. Roa and R. Suárez. Grasp Quality Measures: Review and Performance. *Autonomous Robots*, 38(1): 65–88, 2014. doi: 10.1007/s10514-014-9402-3.

[110] RoboSklep. Shadow Dexterous robotic hand, 2020. Retrieved 24 February 2020 from `http://robosklep.com/en/robotic-hands/127-shadow-dexterous-robotic-hand.html`.

[111] Wonik Robotics. Alegro Hand. Retrieved 24 February 2020 from `http://www.simlab.co.kr/Allegro-Hand.htm`.

[112] N. Sadawi. Semantic Web Tutorial, 2014. Retrieved 24 February 2020 from `https://www.youtube.com/playlist?list=PLea0WJq13cnDDe8V7eVLReIaOnFztOEAq`.

[113] J. Sanchez, J.A. Corrales, B. Bouzgarrou, and Y. Mezouar. Robotic Manipulation and Sensing of Deformable Objects in Domestic and Industrial Applications: A Survey. *International Journal of Robotics Research*, 37(7):688–716, 2018. doi: 10.1177/0278364918779698.

[114] R. Sanz, J. Bermejo, J. Morago, and C. Hernández. Ontologies as Backbone of Cognitive Systems Engineering. In *AISB Symposium on Cognition And Ontologies*, 2017.

[115] A. Saxena, L.L.S. Wong, and Y.N. Andrew. Learning Grasp Strategies With Partial Shape Information. In *Proceedings of the National Conference on Artificial Intelligence*, volume 3, pages 1491–1494, 2008.

[116] P. Schmidt, N. Vahrenkamp, M. Wachter, and T. Asfour. Grasping of Unknown Objects Using Deep Convolutional Neural Networks Based on Depth Images. In *IEEE International Conference on Robotics and Automation*, pages 6831–6838, 2018. doi: 10.1109/ICRA.2018.8463204.

[117] J. Schreiber. Home - pomegranate 0.6.0 documentation, 2016. Retrieved 26 February 2020 from `https://pomegranate.readthedocs.io/end/stable`.

[118] F. Song, Z. Zhao, W. Ge, W. Shang, and S. Cong. Learning Optimal Grasping Posture of Multi-Fingered Dexterous Hands for Unknown Objects. In *IEEE International Conference on Robotics and Biomimetics*, pages 2310–2315, 2018. doi: 10.1109/ROBIO.2018.8665277.

[119] HIRO Robotics S.R.L. Ubiros Gentle Duo. Retrieved 24 February 2020 from `https://www.coboticsworld.com/portfolio-items/ubiros-gentle-duo/`.

[120] T. Stoyanov, R. Krug, R. Muthusamy, and V. Kyrki. Grasp Envelopes: Extracting Constraints on Gripper Postures from Online Reconstructed 3D Models. In *IEEE International Conference on Intelligent Robots and Systems*, pages 885–892, 2016. doi: 10.1109/IROS.2016.7759155.

[121] J. Stückler and R. Steffens. Real-time 3D Perception and Efficient Grasp Planning for Everyday Manipulation Tasks. *Proc. of the 5th European Conference on Mobile Robots*, pages 1–6, 2011.

[122] F.M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In *16th International Conference on the World Wide Web*, pages 697–706, 2007.

[123] T. Sugaiwa, G. Gujii, I. Hiroyasu, and S. Sugano. A Methodology for Setting Grasping Force for Picking up an Object with Unknown Weight, Friction, and Stiffness. In *IEEE International Conference on Humanoid Robots*, pages 288–293, 2010.

[124] T. Suzuki and T. Oka. Grasping of Unknown Objects on a Planar Surface Using a Single Depth Image. In *IEEE International Conference on Advanced Intelligent Mechatronics*, pages 572–577, 2016. doi: 10.1109/AIM.2016.7576829.

[125] H. Taams. UDEA - Comsave introduction to SIR [PowerPoint presentation]. Retrieved from Vanderlande's intranet.

[126] H. Taams. Instructions on Gathering Dataset for SIR, 2019. Retrieved from Vanderlande's intranet.

[127] H. Taams, personal communication. 10 December 2019.

[128] H. Taams, personal communication. 11 September 2019.

[129] A. ten Pas and R. Platt. Using Geometry to Detect Grasps in 3D Point Clouds. In *International Symposium on Robotics Research*, 2015.

[130] M. Tenorth and M. Beetz. KnowRob - Knowledge Processing for Autonomous Personal Robots. *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4261–4266, 2009. doi: 10.1109/IROS.2009.5354602.

[131] M. Tenorth, A.C. Perzylo, R. Lafrenz, and M. Beetz. Representation and exchange of knowledge about actions, objects, and environments in the ROBOEARTH framework. *IEEE Transactions on Automation Science and Engineering*, 10(3):643–651, 2013. doi: 10.1109/TASE.2013.2244883.

[132] The MathWorks, Inc. ROS Custom Message Support, 2020. Retrieved 28 February 2020 from `https://nl.mathworks.com/help/ros/ug/ros-custom-message-support.html`.

[133] Ubiros. Buy Now, 2020. Retrieved 24 February 2020 from `http://ubiros.com/buy-now/`.

[134] E. van der Hall. Parcel and Airports Gripper Market Research [PowerPoint presentation], 2018. Retrieved from Vanderlande's intranet.

[135] K. van Schooten. *A Review on Multi-Fingered Grasp Synthesis for Picking Up Objects*. PhD thesis. Master's Literature Report, Delft University of Technology, 2019.

[136] F. Vanderhaegen, S. Debernard, and L. Automatique. Resilience of a Human-Robot System using Adjustable Autonomy and Human-Robot Collaborative Control. *International Journal of Adaptive and Innovative Systems*, 1(1):13–29, 2009.

[137] K.M. Varadarajan and M. Vincze. Ontological Knowledge Management Framework for Grasping and Manipulation. *IROS Workshop on Knowledge Representation for Autonomous Robots*, 2011.

[138] G. Vezzani, U. Pattacini, G. Pasquale, and L. Natale. Improving Superquadric Modeling and Grasping with Prior on Object Shapes. In *IEEE International Conference on Robotics and Automation*, pages 6875–6882, 2018. doi: 10.1109/ICRA.2018.8463161.

[139] S. Vredeveldt. Semantic knowledge to assess the capabilities of AUVs for planning. Master's Thesis, Delft University of Technology, 2019.

[140] H. Wang. Performance Tradeoff - When is MATLAB better/slower than C/C++, 2019. Retrieved 24 February from `https://stackoverflow.com/questions/20513071/performance-tradeoff-when-is-matlab-better-slower-than-c-c`.

[141] R. Whitwam. Soft robotics gripper uses vacuum pressure and a beanbag to move objects, 2014. Retrieved 24 February 2020 from `https://www.extremetech.com/extreme/174723-soft-robotic-gripper-uses-vacuum-pressure-and-a-beanbag-to-move-objects`.

[142] N. Winkler. What Is the Future of Ecommerce? 10 Insights on the Evolution of an Industry, 2020. Retrieved 24 February 2020 from `https://www.shopify.com/enterprise/the-future-of-ecommerce`.

[143] F. Wörgötter, E.E. Aksoy, N. Krüger, J. Piater, A. Ude, and M. Tamosiunaite. A Simple Ontology of Manipulation Actions Based on Hand-Object Relations. *IEEE Transactions on Autonomous Mental Development*, 5(2):117–134, 2013. doi: 10.1109/TAMD.2012.2232291.

[144] M. Yan, A. Li, M. Kalakrishnan, and P. Pastor. Learning Probabilistic Multi-Modal Actor Models for Vision-Based Robotic Grasping. In *The 2019 International Conference on Robotics and Automation*, 2019.

[145] A. Zeng, S. Song, K.T. Yu, E. Donlon, F.R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo, N. Fazeli, F. Alet, N.C. Dafle, R. Holladay, I. Morena, P. Qu Nair, D. Green, I. Taylor, W. Liu, T. Funkhouser, and A. Rodriguez. Robotic Pick-and-Place of Novel Objects in Clutter with Multi-Affordance Grasping and Cross-Domain Image Matching. In *Proceedings - IEEE International Conference on Robotics and Automation*, 2018. doi: 10.1109/ICRA.2018.8461044.

[146] Z. Zhao, X. Li, C. Lu, and Y. Wang. Center of Mass and Friction Coefficient Exploration of Unknown Object for a Robotic Grasping Manipulation. In *Proceedings of 2018 IEEE International Conference on Mechatronics and Automation*, pages 2352–2357, 2018. doi: 10.1109/ICMA.2018.8484499.