TECHNISCHE UNIVERSITÄT WIEN

AUTOMATION & CONTROL INSTITUTE
INSTITUT FÜR AUTOMATISIERUNGS- & REGELUNGSTECHNIK

Robotic Grasping of Tiny Objects with a Suction Cup

DIPLOMARBEIT

Conducted in partial fulfillment of the requirements for the degree of a

Diplom-Ingenieur (Dipl.-Ing.)

supervised by

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. M. Vincze
Projektass. Dipl.-Ing. P. Ausserlechner

submitted at the

TU Wien
Faculty of Electrical Engineering and Information Technology
Automation and Control Institute

by
Lukas Rinnofner
Matr.Nr.: 01525111

Wien, am 8. März 2023

Vision for Robotics Group
A-1040 Wien, Gusshausstr. 27, Internet: http://www.acin.tuwien.ac.at

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass die vorliegende Arbeit nach den anerkannten Grundsätzen für wissenschaftliche Abhandlungen von mir selbstständig erstellt wurde. Alle verwendeten Hilfsmittel, insbesondere die zugrunde gelegte Literatur, sind in dieser Arbeit genannt und aufgelistet. Die aus den Quellen wörtlich entnommenen Stellen, sind als solche kenntlich gemacht.

Das Thema dieser Arbeit wurde von mir bisher weder im In- noch Ausland einer Beurteilerin/einem Beurteiler zur Begutachtung in irgendeiner Form als Prüfungsarbeit vorgelegt. Diese Arbeit stimmt mit der von den Begutachterinnen/Begutachtern beurteilten Arbeit überein.

Wien, am 8. März 2023

Lukas Rinnofner

Danksagung

Am Beginn meiner Diplomarbeit möchte ich meinem Betreuer Herrn Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze für die Anregung zu diesem interessanten Thema, die zahlreichen hilfreichen Ideen und die konstruktive Kritik zu meiner Arbeit meinen Dank aussprechen. Ebenfalls danken möchte ich meinem Zweitbetreuer Herrn Dipl.-Ing. Philipp Ausserlechner für sein Feedback und seine fachliche Unterstützung.

Abschließend gilt mein Dank meinen Eltern, die mich bei meinem gesamten Bildungsweg stets unterstützt haben. Ohne sie wäre ein Abschluss meines Studiums in dieser Form sicher nicht möglich gewesen.

Abstract

Evolved from simple manipulators, robots have progressively turned into all-round helpers for various tasks. The ability to grasp objects poses a fundamental skill for autonomous machines. Pick-and-place tasks in a crowded household remain still challenging for state-of-the-art robots. For this thesis, a program to perform pick-up maneuvers with a suction cup was created. The goal was to pick tiny objects reliably and place them at a target position. To achieve this, a custom dataset was created to set up an object detection system. With the dataset, an object detector, able to localize things of interest, was trained. Subsequently, feasible grasp poses can be determined and reached using a motion planner to avoid any collision with obstacles. After a successful grasp, the robot is able to recognize a storage box and drop an object in it. For transitions between these program parts, a state machine was designed. Various experiments were carried out to test the developed concept. These involved picking tasks with different items and additional obstacles. The results showed that the robot can pick up tiny objects reliably in real-world scenarios. The final program enables further research to easily investigate different approaches for detection and motion planning.

Kurzfassung

Ausgehend von einfachen Manipulatoren haben sich Roboter zunehmend zu vielseitig einsetzbaren Helfern für diverse Aufgaben entwickelt. Die Möglichkeit Objekte zu ergreifen stellt eine grundlegende Fähigkeit für autonome Maschinen dar. Pick-and-Place-Aufgaben in einem überfüllten Haushalt sind für moderne Roboter immer noch eine Herausforderung. Für diese Diplomarbeit wurde ein Programm zur Durchführung von Aufhebe-Manövern mit einem Saugnapf entwickelt. Ziel war es, kleine Objekte zuverlässig aufzuheben und sie an einem Zielort abzulegen. Um dies zu erreichen, wurde ein eigener Datensatz erstellt und ein Objekterkennungssystem entwickelt. Mit dem Datensatz wurde das Erkennungsprogramm trainiert. Damit können Objekte lokalisiert werden und anschließend passende Greifpositionen berechnet werden. Die Verwendung eines Bewegungsplaners ermöglicht es, die berechnete Position zu erreichen und eine Kollision mit Hindernissen zu vermeiden. Nach einem erfolgreichen Aufheben ist der Roboter in der Lage eine Lagerbox zu erkennen und ein Objekt darin abzulegen. Für die Übergänge zwischen diesen Programmteilen wurde eine State-Machine entworfen. Um das entwickelte Konzept zu testen, wurden verschiedene Experimente durchgeführt. Dabei handelte es sich um Aufhebe-Aufgaben mit unterschiedlichen Gegenständen und zusätzlichen Hindernissen. Die Ergebnisse zeigten, dass der Roboter in der Lage ist kleine Objekte in einem realen Szenario zuverlässig aufzunehmen. Für zukünftige Forschungen ermöglicht das finale Programm eine einfache Untersuchung von verschiedenen Ansätzen zur Objekterkennung und Bewegungsplanung.

Contents

CHAPTER 1

Introduction

In the past years, robots have been able to offer more and more support for humans. In particular, various of them found their way into many households. These so-called service robots can perform useful tasks for humans. Today, already available commercial products for households can clean floors and pools or mow the lawn. In the future, more sophisticated tasks will likely be done by robots too. This would involve tasks like cleaning a complete household and assisting elderly or disabled people with their daily needs. As the number of possible jobs for a robot increases, as complex the development of such machines gets. Designing an ideal kinematic construct for a manipulator, finding a suitable motion planning algorithm, or detecting and localizing various objects are just a few examples of current challenges. These interdisciplinary challenges intersect with the domains of computer science, electrical and mechanical engineering and others. Using the previously done work on hard- and software in the field of robotics, this thesis aims to contribute to this knowledge by tackling the problem of a pick-and-place task.

1.1 Motivation

The Human Support Robot (HSR) from Toyota, was developed to serve humans in their households. A simple task like bringing a glass of water is a large challenge for a robot. It must be able to move around in a crowded environment and find a water source. The tap must be known, and feasible grasps must be calculated. Robotic researchers tackle various other examples of such problems. To push this development, Toyota provides the HSR to different universities as a research object. One was given to the Vision For Robotics team (V4R) at the TU Wien, which is focused on computer vision related problems. Different object detection and pose estimation algorithms are tested with the HSR [1], [2]. These

methods are examined with grasping experiments using the robot's gripper. The gripper is not practicable for tiny objects and therefore, pick-up maneuvers with the suction cup shall be investigated. In doing so, a basic structure for grasping tasks with the ability to test other more refined detection and grasping algorithms should be created.

1.2 Challenge

Grasping an object resembles a simple task for humans but represents a complex problem for a robot. There is still an extensive amount of research left until robots are able to operate in a normal household environment without problems. In order to adapt robots to specific problems in the human surrounding they can be equipped with hardware that is superior to its human equivalent. An example in terms of grasping would be the use of a suction cup. A cup can lift objects just by creating a vacuum on plain surfaces. Hence the term "tiny objects" in the thesis title could also be interpreted as "thin" or "flat objects", which have proper surfaces to generate a vacuum on. Many household items have flat surfaces, and by exploiting those, appropriate grasping points can be calculated without the knowledge of all 6 degrees of freedom (DoF) of the object. As another approach to the formerly developed pose estimators, a simpler way of finding suitable grasp points is examined here.

Figure 1.1: The Human Support Robot with household items and the storage box

Three goals are defined. The first is a functional detection of tiny objects with the HSR visual system. Next is achieving a pick-up maneuver with the suction cup, and the last goal is an additional handling of the picked objects. This involves finding the proper storage box and performing a drop maneuver. Figure 1.1 shows the HSR with a selected

set of ordinary household objects on the table. The card, box, and coin to the right resemble tiny objects, which are the targets of the suction cup pick-up tasks, and the spam and mustard function as additional distractors and obstacles. Also, the storage box is visible in the background. In Figure 1.2, a sketch of the overall goal can be seen. After reaching a given waypoint, the robot should pick a tiny object, move to the known drop point and place the item inside the storage box.

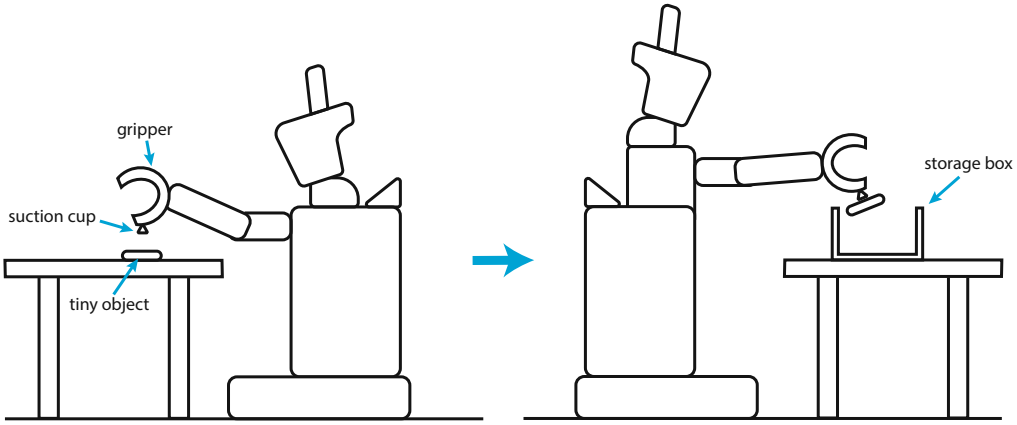


Figure 1.2: Pick-and-place goal

1.3 Approach and Contributions

To achieve the chosen goals, a Python [3] program for the robot was created. The developed grasping program is called Tiny Objects Program or short TOP. As shown in Figure 1.3, the goals of the three main parts resemble the core parts of the software. A successful pick-up move must run through the program in the given order. In the design process the same sequence was kept. First, an individual detection system was created. This system was trained with a specially created dataset including images of cards, boxes, and coins. Each of the three objects can be detected with the HSR's RGB-D sensor. A ROS service is used for the detection system, making it possible to change the trained model easily. In the next step, as shown in the yellow box in Figure 1.3, a feasible grasp pose had to be created and reached with the robot's manipulator. An appropriate suction cup position over the sought-after item is determined using its localization given by the three translational DoFs. Obstacles are detected by the depth sensor and a motion planner is used to find a collision free trajectory to the grasping pose. The method's precision is sufficient to pick cards and boxes. The green box outlines the last step of TOP. In this, the robot can handle a picked object in different ways. A drop maneuver was developed to store objects. Thereby, the HSR is able to recognize a storage box with a QR code. Again the motion planner is used to move the robot's arm in the box. Then the picked item is dropped by pushing the suction cup over the box's edge.

The main contribution of this thesis is a working program for the Toyota HSR to execute pick-and-place tasks of tiny objects with the suction cup. The complete program can be accessed over [4]. All essential goals of the three main parts could be achieved. Each section

Figure 1.3: The three main parts of the Tiny Objects Program (TOP)

was designed separately, making it possible, to execute the main functions independently. A state machine was created to manage the transitions between these functionalities and additional error-handling parts. To execute a complete pick-up task, a set of waypoints is given to the robot. Those are used to define locations with a good view on a table or shelf and mark the storage box or the handover position. The basic structure of the state machine is also adaptable to integrate more error treatments.

1.4 Results

The feasibility of various grasping tasks was verified by experiments. With the developed program, the HSR is able to perform pick-and-place maneuvers with cards and small boxes. For coins, another method with more precision is needed. The tests proved that a simple approach to object detection and grasp point estimation is enough to pick up objects with a flat surface. Different grasping tasks involving obstacles were carried out to examine the detection system and the motion planner. A rating system was created to review the success rate of the grasping tasks. The achieved score shows the capability to pick up tiny objects in a simple real-world setup reliably. Further experiments were done to investigate the weaknesses of the program. They showed additional promising possibilities in terms of object detection and motion planning, especially in avoiding collisions with other objects, to accomplish better results in more crowded real-world settings.

1.5 Thesis Outline

The following chapter depicts the state of the art in related terms of science. This includes modern pick-and-place robots and methods for object detection and motion planning. Chapter 3 outlines the basics behind the programming of a robot and describes some of the used software. In chapter 4, the development of TOP and the thoughts behind it are shown in detail. The finished program was tested with a variety of experiments, which are documented in chapter 5. In the last chapter, this thesis ends with an overall conclusion and suggestions for potential future work.

CHAPTER 2

Related Work

In this chapter, a look is taken at the related research areas of this thesis. The first section provides an outline of different robots and their use of suction cups. The build of the robot, which is later used for experiments, is explained and it is compared to similar robots equipped with suction cups. In the next subchapter, computer vision and the basics behind it are explained. The last section shows different solutions to the problem of robot motion planning.

2.1 Pick-and-Place Robots

In this section, three different robots for pick-and-place operations are listed and described. The first one was developed to compete in a challenge with other robots to compare different ways of tackling one task. The insight of the development team is outlined. The second one is an example of a robot sold by an industrial robot company. Some features, more specific for an industry robot, are listed. The last section describes the robot used later in this thesis for picking up thin objects.

2.1.1 Amazon Picking Challenge Winner

To compare the accomplishment of different robots, competitions, and other scoring systems are needed. One competition was the Amazon Picking Challenge, which was attended by 25 university teams from all over the world. The goal was to address the typical warehouse problem of picking a variety of things from a subdivided shelf. The list of objects included paper boxes, plastic packages, balls, a pencil cup, and more things

of other shapes. The robot should be able to locate a specific object correctly, grasp it, and put it in a bin beside the shelf. Difficulties of such a task are the alternating form of the objects, the different materials and surfaces, the long shelf, which leads to visually declined images in the back area, and the cluttered order with objects lying beside each other.

In [5], the winner team of the 2015 challenge is describing their robot. A few of their key elements during the designing process and their evaluation are summarized here. Taking a look at the hardware of the robot, part of its success is, besides the holonomic mobility, its end effector. The end effector consists of a suction cup on a crevice nozzle. The system is capable of producing an air flow with a 250 W vacuum cleaner. This strong suction force, which can lift up to 1.5 kg, is insensitive to small deviations of the ideal grasp point location and therefore reduces the precision requirements of the object detection. In Figure 2.1 the slim form of the end effector can be seen. This form is ideal for reaching objects in the back of a crowded shelf, without a large risk of colliding with other objects. The evaluation of the failure cases shows that the most likely unsuccessful picking attempt is due to an object recognition failure. A disadvantage of the suction cup system is the lack of the ability to pick up the pencil cup, because of its permeable structure.

Figure 2.1: The robot during the challenge [5]

On the software side, the robot is built on a modular system with ROS [6] in combination with other modules like OpenCV [7] for perception or [8] for navigation. Due to the high grade of uncertainties, complex motion planning is replaced by very simple planning and a feedback approach. The grasp pose is planned beforehand and during the execution, the robot uses pre-defined motions in combination with sensor feedback gained from the contact surfaces of the objects or the shelf. There are two possible grasp poses: One from the top and one from the side. The selected pose is picked with the use of a scoring system, which considers the success rate of a pose in combination with the orientation of the object and the available space for the end effector. During the grasp, the suction cup presses the items against the shelf. In doing so, the need to calculate an ideal grasp point is neglected by exploiting the environment. The team's solution is positioned far on the

feedback side. In comparison to that, the majority of other teams relied more on motion planning.

The winner of the 2018 challenge is described in [9]. This robot is also equipped with a suction cup as the primary picker and an additional gripper for manually specified items. Calculating the grasp pose is done with three different strategies. The first tries to find surface-normals if accurate depth information of an item is available. These normals are weighted, and the best grasp point is chosen. If there are just scattered depth points, such as coming from partially reflective objects, the second method picks a single grasp point at the center of the depth point cloud. The last strategy uses the RGB image of the camera to estimate an object's position. The center of the object's RGB segment is used as grasp point and the grasp is executed vertically.

2.1.2 Industry Robots

Properties like reliability and economic efficiency are not prioritized in robotic scientific research. The research focuses on developing new concepts and methods to gain knowledge. The robot of the Amazon Picking Challenge winner team succeeded in picking up ten of twelve objects [5]. This performance can be considered sufficient for a challenging task with many uncertainties and where feasibility is the goal. If the robot should support or replace a human worker instead, this success rate must be raised. Therefore more common industrial solutions aim to achieve higher reliability in handling simpler tasks with known boundaries. An example could be a pick-and-place task, where the form and the location of the parts are known, or a transport of an object from one point to another on a predefined route. In such cases, the robot can be designed and programmed specifically, and the software just needs to handle minor deviations from the predefined assumptions.

Figure 2.2: SWIFTI CRB 1100 from ABB [10]

One example of collaborative robots or cobots would be the SWIFTI CRB 110 from ABB [10], which can be seen in Figure 2.2. This stationary manipulator can be controlled by an interface or programmed for pick-and-place operations. In combination with an

external camera, the objects can be in a cluttered order. The robot can be equipped with suction cups or a gripper, depending on the use case. ABB developed safety features for a collaborative environment where humans and robots work and interact with each other. The presence of humans can be detected by a laser scanner, and the system slows down any movements when a person is in a closer range. If the person gets too close, the robot stops, and removing or adding parts to the working space is possible. In terms of working capacity, the maximum payload is 4 kg with a Tool Center Point (TCP) speed of 5 m/s. In operation, the achievable accuracy is up to 0.01 mm.

2.1.3 The Toyota HSR

This section gives an overview of the Human Support Robot (HSR) from Toyota. The HSR was used for developing the program to pick up flat objects in this thesis. In [11], the development of the HSR is documented, and the following information is taken from there. The motivation to design such a robot was the demand for a helping companion in households of older people similar to service dogs. To identify the main tasks, which HSR should be able to perform, potential service dog users were asked. After evaluating the responses, the focus was given to carrying tasks and the ability to remote control the robot. The finished form of HSR can pick up objects with a weight of up to 1.2 kg from the floor or up to a height of 725 mm by grasping from the top, side, and front directions. The arm can be extended to a height of 1375 mm. A maximum velocity limit of 0.8 km/h is set for safety reasons. The given target of not exceeding a maximum kinetic force of 10 J is reached, and in an unfavorable case the energy was 3,24 J. For flat objects, a suction pad, capable of producing a force up to 3 N, is added to the gripper.

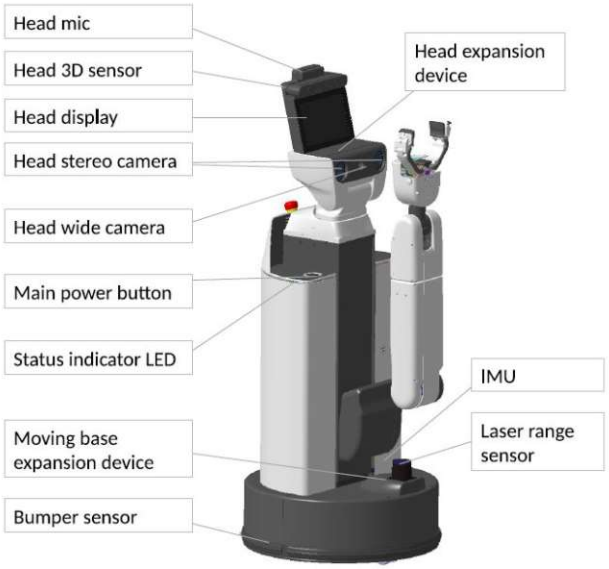


Figure 2.3: The HSR is equipped with various sensors [12]

The software architecture operates with ROS, and a real-time subsystem with a Linux kernel manages the motion control. The robot has a variety of sensors, which can be seen

in Figure 2.3. The access of the sensor data can be done over ROS topics or with Python. To detect objects, either the RGB images of the head stereo camera or the 3D sensor can be used. The 3D sensor is capable of providing an RGB image and a depth image. Intense calculations for object detection can be outsourced on external hardware, transferring data over wireless or wired LAN. The laser range sensor on the mobile base is used for collision detection. There are also additional sensors to those in Figure 2.3 like a hand camera, a force and torque sensor in the hand, and a pressure sensor for the suction cup. The use of the software and the motion planning are explained in more detail in later sections.

The HSR and the SWIFTI CRB 1100 are designed for different applications. The ABB robot achieves a high precision with faster speed compared to the HSR. This is possible due to the safety measures, where no human is near during fast movements. The HSR ensures safety by reducing the kinetic energy and also achieves a more harmless impression, compared to a fast-moving industry robot. Path planning of HSR movements is done by inverse kinematics. If the gripper touches a surface, the feedback can be used over the arm force sensor. Similar to the Amazon Picking Challenge winner [5], this can be used for tasks with known boundaries.

2.2 Computer Vision

The term computer vision covers a variety of domains in science, which all have the goal to extract information from images, similar to the human visual system. The processed data can be multiple images, videos or also depth information from 3D scanners. This interdisciplinary field deals with getting, processing and evaluating digital images. The focus of this section lies in the analysis of images in terms of object detection and pose estimation. With the use of object detection, a machine should be able to classify and localize a specific object shown in an image. Classification is the ability to identify the presence of an object which could be a person, household things, an animal or something else. Localization defines the position of an object, which is then marked with a bounding box.

2.2.1 Neural Networks used for Computer Vision

Popular state-of-the-art methods for computer vision tasks use convolutional neural networks (CNN). In recent years some of those methods had become so powerful that even human competitive results are achievable [13]. In this subsection, at first a look is taken at the basics of feedforward neural networks (FNN) followed by CNNs. The presented information of CNNs is based on [14]. Figure 2.4 shows a simple FNN, which consists of 3 layers. A multidimensional vector is mostly the input for the first neurons. Depending on the type, usually inside a neuron the inputs are multiplied with weights, summed up, and fed into an activation function. This function transmits the output to the next layer, a hidden layer in the example. A perceptron is a single neuron network, which has just zero and one as output value. The weights of this neuron get updated by calculating the loss function on the training set and using backpropagation. Supervised learning is

the training with labeled data, meaning for a given input a specific output is predefined. Unsupervised learning is without prior labeling of the data, but in terms of image pattern recognition usually the training is supervised. The structure of FNNs comes along with disadvantages in terms of object detection. For just a 28x28 pixel, black-and-white image one neuron of the first hidden layer would receive 784 inputs. This produces a complex structure with a large number of weights, which scales with the image size. Training those weights is computationally intense, and it is more difficult to prevent overfitting. CNNs have proven to be better in terms of object detection, as they are reducing the number of trainable weights.

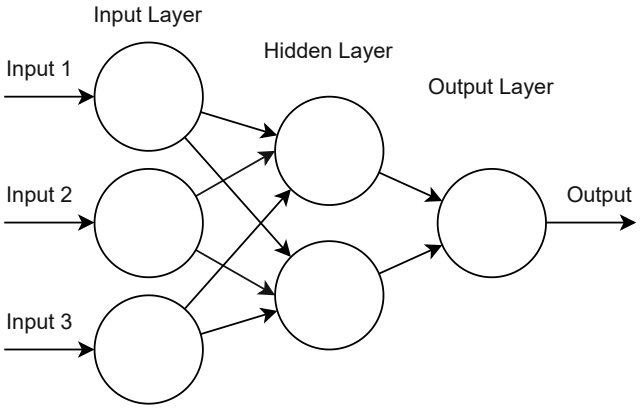


Figure 2.4: Simple feedforward neural network

Compared to the FNN a CNN consists typically of three different layer types: convolutional, pooling and fully-connected layers. A convolutional layer uses a trainable kernel and parts of the input matrix to calculate a new matrix. A kernel can be understood as a filter for specific patterns for example edges. In Figure 2.5 the process of a 2D convolution and pooling can be seen. In this example the input is a 6x6 matrix. By multiplying 3x3 parts with different kernels, which consist of weights trained for different patterns, the values of new 6x6 matrices are calculated. Different methods exist for handling the matrix edges, which can also lead to a reduction in the matrix size. The pooling layer always reduces the size of the matrix, where max pooling is the preferred method. This leads to a smaller amount of weights needed in the next layers and usually the number of kernels is increased. A CNN consists mostly out of different convolution and pooling layers. After those, fully-connected layers are used. They are created with a structure of normal feedforward layers. The number of neurons in the output layer can be the amount of different classes, which are detectable by the network. The advantage of CNNs is their reduced complexity compared to FNNs. In a convolution layer, the same kernel values are used for the whole input matrix. The number of training weights are reduced to just the kernel values, allowing CNNs to handle larger input images without an increase of network complexity. This makes them the preferred state-of-the-art method in terms of object detection.

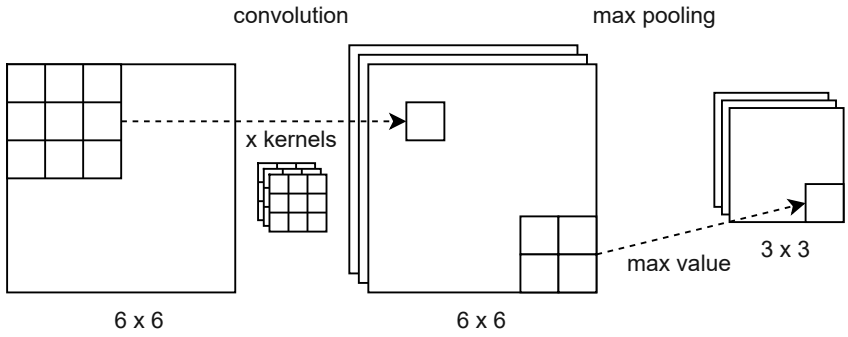


Figure 2.5: CNN convolution and pooling

Figure 2.6: Instance segmentation done with Mask R-CNN [16]

2.2.2 Mask R-CNN

With CNNs classification tasks can be done significantly better as with FNNs, but additional parts are needed to localize the found object on the image. The desired goal here is instance segmentation, meaning to detect all pixels from the same object and not only a bounding box. An example for instance segmentation can be seen in Figure 2.6, where known objects and persons are marked with different colors. The library Detectron2 [15] provides algorithms for detection and segmentation. With this platform the Mask R-CNN detector is later used.

Mask R-CNN [16] is built up on the Faster R-CNN framework [17]. The structure of Faster R-CNN is shown in Figure 2.7. An input image is first processed with convolutional layers into feature maps. The feature maps serve as input for the Region Proposal Network (RPN), which outputs candidate bounding boxes. This is done with anchor boxes in different scales and aspect ratios, a classifier and regressor. The probability, that an object is inside a proposal, is determined by the classifier. The regressor regresses the bounding box, estimated by an anchor box, into a refined estimated bounding box. The RPN gives back proposals for potential objects in the feature map.

Mask- and Faster R-CNN proceed differently after the RPN. Faster R-CNN uses Region of Interest (RoI) pooling to produce fixed size feature maps. These maps are then classified and the outputs are bounding boxes with their specific object labels. As seen in Figure

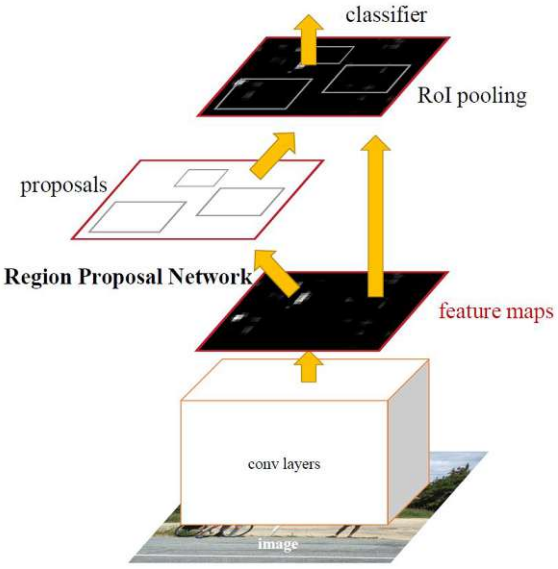


Figure 2.7: Faster R-CNN framework [17]

2.8 Mask R-CNN uses RoIAlign instead. This is an improved method for pooling. Next, the RoIAlign output is used with a CNN to determine the object class at first. Decoupled from the classification, another CNN is used to finally determine the segmentation. For each object learned by the network, the output of Mask R-CNN is then the object class, a bounding box and a binary mask indicating the pixels of the object.

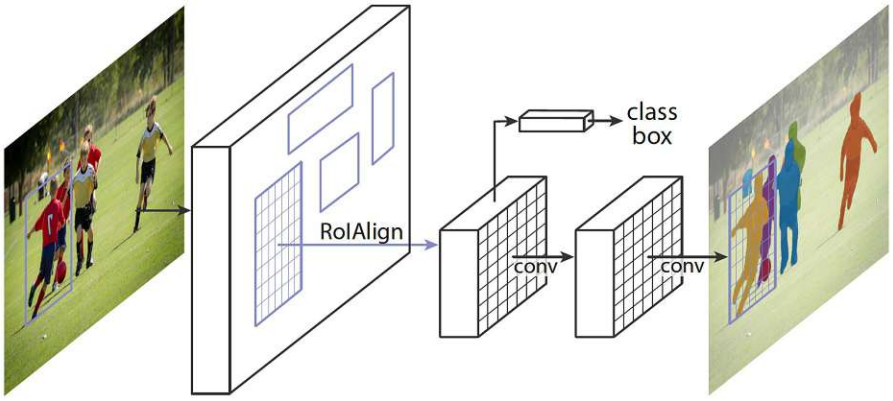


Figure 2.8: Mask R-CNN framework [16]

2.3 Motion planning

Motion planning is the process of finding a feasible sequence of joint trajectories so that a robot's end effector can reach its target pose. This includes solving the problems of path planning, trajectory planning, and joint control. Starting with a given goal as a grasp pose, which could come from a human user interface or the object detection system of

the robot, the end effector has to reach this position. In Figure 2.9, a simple planning flowchart can be seen. The first step is to define the final position in joint coordinate space. Therefore the path planning algorithm uses inverse kinematics to calculate the joint angles from a given gripper pose. In most cases, this results in an over-determined kinematic problem, because there are several possible joint positions. Here the nearest possible solution to the actual position can be picked to obtain the joint angles. If there are obstacles present, the path cannot be a straight movement to the goal. Then the path planning algorithm needs to find a path with collision avoidance. If a possible path is found, the trajectory planner calculates suitable velocities and accelerations for the joint controller. This is done by using splines as paths and derivating them.

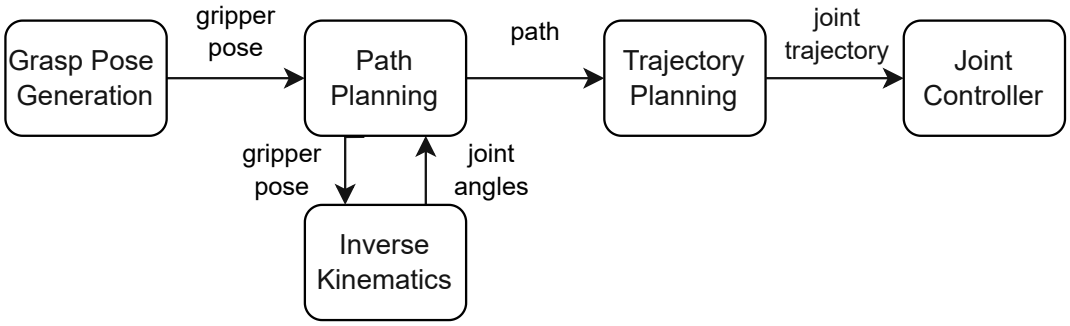
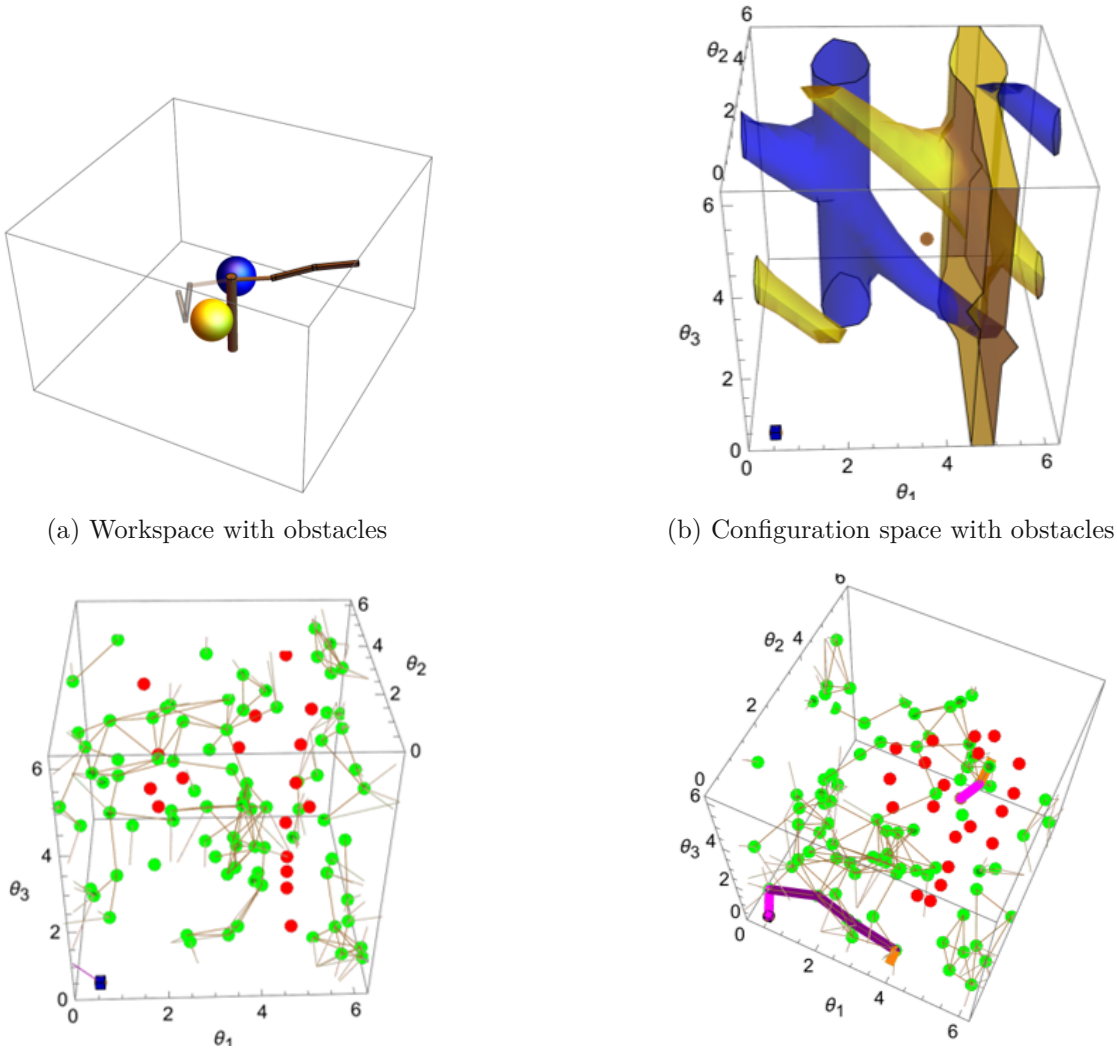


Figure 2.9: Motion planning flowchart

Most common mounted manipulators have 6 degrees of freedom. Able to move its base, the HSR for example has a total of 8 DoFs. Therefore, to solve the inverse kinematic problem, two joints are calculated and then used as parameters in the inverse kinematics analytic equation. The obtained solutions are then optimized to a final one with the use of a weighting function. The weight is previously defined and either prefers a larger base or arm movement [12]. If there are no obstacles to avoid, the robot's path just needs to respect the joint limitations and can be designed as a simple, desired angle value for each joint controller. The typical workspace of a service robot is mostly crowded with a variety of obstacles, which can change locations. Therefore collision avoidance has to be taken into consideration.

To explain how a path with collision avoidance can be found, an example is shown in Figure 2.10. The shown method is a probabilistic roadmap method (PRM), which is one possible way of pathfinding. This method was introduced in [19]. In (a) a robot arm with 3 DoFs and two spherical obstacles can be seen. The translucent arm represents the goal pose. The configuration space can be seen in (b), where the 3 DoFs represent the 3 axes of the coordinate system. The colored areas are all configurations where the robot would be in contact with the obstacles, and the two points in blue and yellow are the start and goal poses. Usually, these analytical areas of the obstacles in the configuration space are not calculated. Only collision checks for each configuration are done. In (c) a set of random configurations is created, and the red points indicate an invalid configuration. The valid ones are connected with their neighbors within a defined radius. A path is found if there is a possible connection between the valid configurations from the start to the goal point. After adding more random configurations, a collision-free path is found, as shown



(a) Workspace with obstacles

(b) Configuration space with obstacles

(c) Configuration space with random configurations

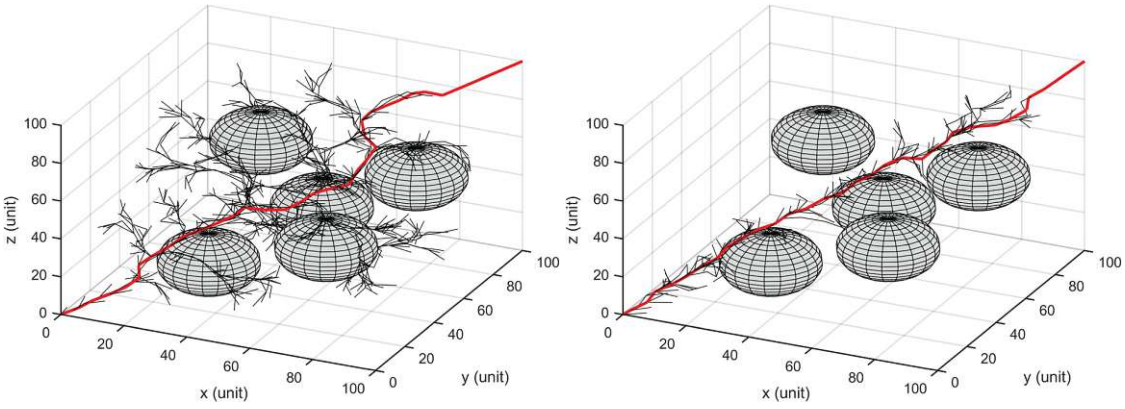
(d) Configuration space with a found path

Figure 2.10: Example of PRM planning (Generated with [18])

in (d). This might not be the shortest path, resulting in some unnecessary movements of the robot arm. If the number of configurations is increased, shorter paths can be found. Another benefit of the PRM method is that for fixed obstacles a once-created roadmap can be used for arbitrary start and goal points. Modifications of the shown PRM are one of the state-of-the-art methods in path planning.

Another concept behind many state-of-the-art methods was published in [20] called rapidly exploring random tree (RRT). In this method, trees of possible configurations expand from the starting point. The key advantage is the expansion towards unexplored regions. One point is created randomly and then added as a vertex to the closest available node if it is not in collision with an obstacle. A connection to a new point must also match with any non-holonomic constraints of the robot's mechanics. The random generation is customizable, making it possible to use some RRT methods for better performance in

specific cases. In Figure 2.11 further developed RRT methods are shown. Both methods are capable of finding a suitable path in the configuration space. RRT* evaluates the distance from a newly added vertex to the starting point and picks the shortest path. Therefore the trees can reroute themselves. In [21] is shown that RRT* is more likely to converge to an optimal solution and outperforms the RRT algorithm. As another example of an RRT variant, the random node generation in RRT*N is additionally biased in a way, that new nodes are generated toward the goal point. This results in the development of fewer tree branches, as seen in Figure 2.11 (b).



(a) RRT* path planning (b) RRT*N path planning,

Figure 2.11: Different variants of RRT methods [22]

In [12], the used path planner of the HSR is called the Constrained BiDirectional RRT 2 (CBiRRT2) [23]. This algorithm tries to connect two RRTs starting from the start and goal position. It also allows planning with constraints such as collisions, torque, or balance. MoveIt was also used in some cases instead of the original planner, which is accessible through Python commands. This was done because the HSR MoveIt package offers better support in terms of dynamic obstacle detection. MoveIt uses the Open Motion Planning Library (OMPL), which includes a set of different motion planners [24]. The default planner of OMPL is KPIECE [25], but the planner cannot operate with the HSR, so RRT* was chosen. More details on the HSR motion planning can be found in section 4.3.

CHAPTER 3

Basics for Robotic Programming

This chapter gives a brief overview of the software ROS as the operating system for many robots. The core concept behind ROS is explained, and some practical, additional software is shown. Then a brief look is taken at the basics of state machines and their use for sequencing operations. Last, an example for Python commands to control the HSR is given.

3.1 ROS and its Plug-ins

ROS functions as the core of the information and command manager behind the developed program from this thesis. The idea and thoughts of the ROS developers are described in [6]. Robot Operating System or short ROS is not an operating system in a traditional way. Instead, it manages the communication between programs on a host operating system. Before ROS, robotic researchers had to create frameworks to handle the complexity of a robot. In many cases, software was adjusted for a specific purpose, and other frameworks had to be designed for other cases. ROS was created to provide one single framework, which should be used in various cases without repeatedly redesigning essential software. The design goals of ROS are the following. The peer-to-peer communication ability is one key element. Computers, executing computational-intense tasks like object detection, can share information directly with others. Another advantage of ROS is the possibility of using different programming languages. According to [26], ROS supports Python, C++, and Lisp. It should be mentioned that the later used packages in chapter 4 have instructions for Python and C++, where C++ can have additional commands in cases like in MoveIt [27]. The core system is designed efficiently so that additional debugging or logging tools can be used but are not loaded in as default. Various other software projects

are available for ROS, and the focus was to make the entanglement between those as small as possible.

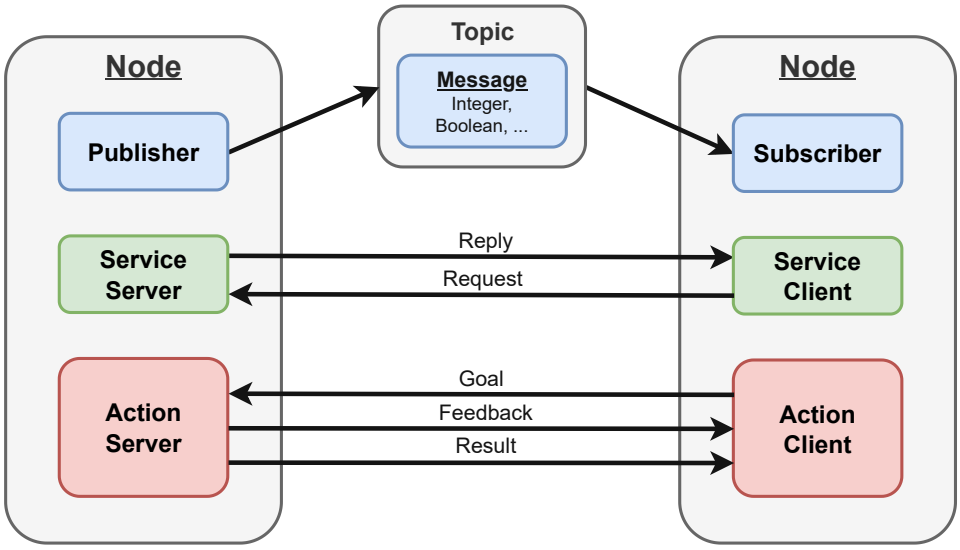


Figure 3.1: Communication concept in ROS

A few terms of the concepts in ROS are described here [28]. The most common terms are the following. Packages are the primary unit for organizing software in ROS. A package includes nodes, configuration files, dependencies, and other information. Distributing software can be done by releasing a package or a repository consisting of a collection of packages. Nodes are performing computational tasks. These operations can be a Python or C++ script calculating something, a program controlling a sensor, or something else. A message is used to communicate between nodes, and the transferred information is previously declared. Figure 3.1 shows the information-sharing concept between two nodes. The information exchanged inside a message can be standard types like integers, arrays, or custom types. A service is used for the one-way exchange of data. For example, controlling a LED can be done by a service server. The server takes requests from a client, handles them, and gives a response to the client. In a similar way, an action server is operating [29]. An action client sends a defined goal to the server, additionally the client can request feedback during the operation or abort it. An action server could control the base movements of a robot and get a location as the goal. The client gets the result once the goal is reached or aborted. Goal, feedback, and result can contain standard types similar to messages.

One of the most significant benefits of using ROS is the access to a variety of additional software over an interface. The robotic simulator Gazebo can be used with ROS to simulate a robot in a physical environment [30]. Before testing code on a real robot, simulating it is a cheaper and faster option, especially for debugging. It is possible to create a simulation world in Gazebo with the use of pre-built objects. Toyota has already built a model of the HSR for Gazebo and provides objects like household articles, furniture, and boxes with QR-Codes. With those, the object detection of the HSR can be tested without having access to the real robot. The robot can be seen in a simulated household

in Figure 3.2. A physics engine calculates interactions with objects. Therefore motion planning and grasping can be tested in Gazebo too.

Figure 3.2: A HSR world created from Toyota in Gazebo

Figure 3.3: The view from the robot in RViz

Another helpful visualization tool of ROS is RViz [31]. With the usage of camera data, RViz is able to create a 3D visualization environment. In Figure 3.3, the view of the HSR robot can be seen. The RGB image of the head sensor, combined with the depth information, generates a 3D model of the environment the robot is observing. Other images from the head stereo aspect or the hand camera are visible on the right side. Additionally to the camera, other sensor data of the lidar system is displayed as red voxels near the ground surface, indicating known collision points. Other collision areas,

detected by the depth sensor, can also be displayed. The green areas on the ground are labeling objects and walls from a predefined map. The visual information about collision points and obstacles can be used to understand specific paths during a robot's movement with collision avoidance. Coordinate systems of fixed points, objects, or robot parts can be displayed for geometry-related tasks. The end effector's coordinate system is visible in Figure 3.3. A variety of custom display settings can be chosen in RViz, making it a convenient tool for debugging.

3.2 State Machines

In the domain of computer science, state machines are computational models to perform specific tasks in a row. A state can have an input value, do calculations with it, or compare it to another saved value and transmit a result to different outputs, which can be other states or the end of a finite state machine. In robotics, constructing a state machine helps build a problem-solving algorithm. It is often easier to divide a task into smaller steps for clarity.

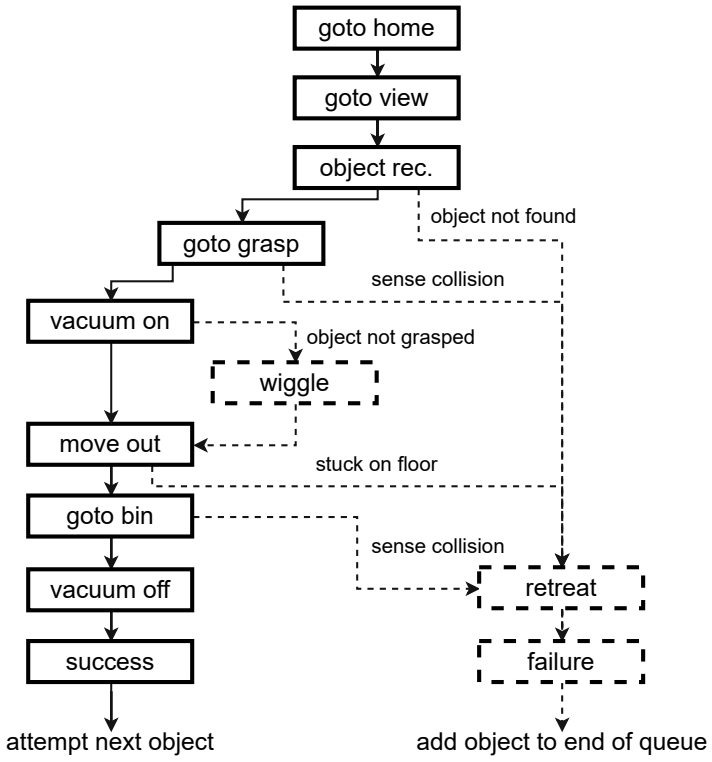


Figure 3.4: The state machine of the Amazon Picking Challenge robot [5]

A simplified version of the state machine used in [5] can be seen in Figure 3.4. Beginning from the starting state "goto home", the desired workflow is indicated by solid arrows. The dashed lines lead to problem-handling states. For example, if the proper object is found in the "object rec." state, the next step would move the robot to a grasp position. Otherwise,

the robot would retreat and try to pick a different object. The shown state machine is just a reduced version, and the complete one in [5] has 26 states and 50 transitions. For implementing a state machine in ROS, the package SMACH can be used [32]. SMACH uses Python as a programming language. States, saved values, and their in- and outputs can be defined in a simple way, making the whole state machine easy to adapt for future applications.

3.3 Python Controls

To program the HSR, either Python or ROS functions can be used. The developed software for this thesis can be accessed over [4]. In Listing 3.1, an example of HSR Python commands is shown. The code defines the *StartSuction* state of a SMACH state machine. This state is just used in the user input and has only the outcome 'succeeded'. The robot and the needed parts have to be initialized first to send them commands afterwards. The suction cup can be started by sending a Boolean variable. In similar ways other parts, like the robot's base or arm, can be controlled.

Listing 3.1: Code example

```
class StartSuction(smach.State):

    def __init__(self):
        smach.State.__init__(self, outcomes=['succeeded'])
        self.robot = Robot()
        self.suction = self.robot.try_get('suction')

    def execute(self, userdata):
        print(50*'#' + '\n')
        if self.suction is not None:
            rospy.loginfo('Starting suction')
            self.suction.command(True)
        else:
            rospy.loginfo('Could not start suction')
        print('\n' + 50*'#' + '\n')
        return 'succeeded'
```

CHAPTER 4

Tiny Objects Program

The following chapter describes the proposed method for pick-and-place tasks with tiny objects. The created method is named Tiny Objects Program, abbreviated with TOP. The first section outlines the corresponding state machine for the whole process. In each state, the transitions between and the handling of unwanted outcomes are described. With the concept of a state machine in mind, key elements of the finding and picking processes are explained next. In section 4.4, the different options for further operation after a successful pick are depicted. The last section briefly overviews the user interface for interacting with the robot.

4.1 Program States

This thesis aims to tackle the problem of picking up tiny objects. To completely cover a robot's pick-up process, all the different processes involved are separated first. Each individual task must be executed in a specific order, dealing with any known errors that occur. A state machine was chosen as the basic structure to build a simple and easy-to-adapt framework for the grasping program. The framework can be seen in Figure 4.1. The green arrows indicate a state's desired outcome, and the red ones lead to error-handling states colored in yellow or the program exit. If a state crashes during execution or connections are lost, the state machine will also transfer to the exit state, which aborts the program.

The state machine starts with the robot moving to the next known viewpoint. Those viewpoints are predefined x and y coordinates with an additional angle as the viewing direction of the robot. It is assumed that the HSR was able to map the room where the

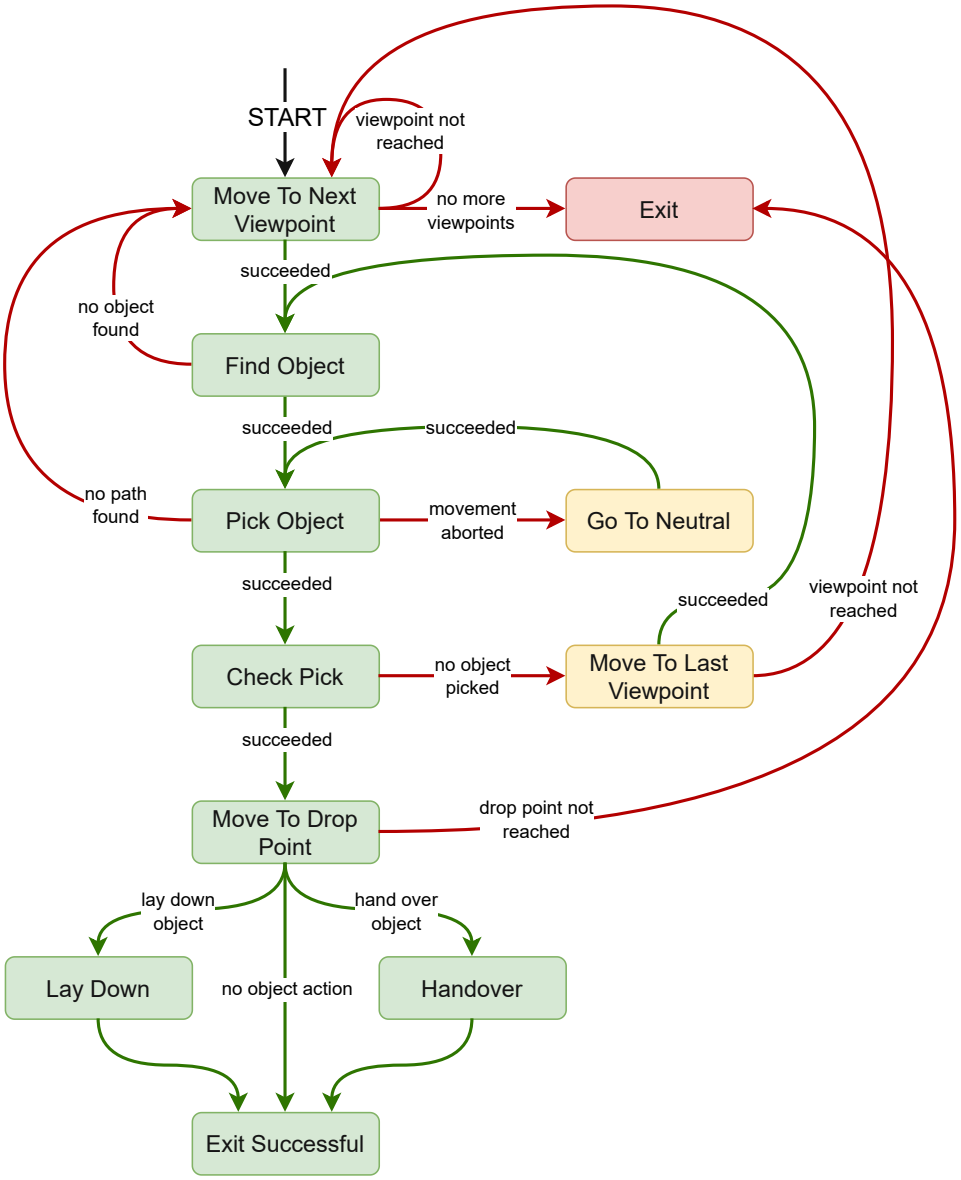


Figure 4.1: TOP state machine

desired object is located. In most cases, the object will lie on a table or other furniture, and therefore, the robot must be able to find appropriate points with a clear perspective of the object. The state machine can work with a mapping program, which provides viewpoints for the moving states to work in an unknown environment. If the room and the furniture are known, and changes to the layout are not possible, predefined viewpoints are sufficient. In case of a blocked viewpoint, the next one is chosen as a goal, or the program will exit unsuccessfully if no points are left. After the state has succeeded, the RGB-D camera of the HSR is used to find the desired object. The name of the object must be provided to the *Find_Object* state. Section 4.2 explains the robot's object detection in detail. If the object detection cannot find the expected item, the robot moves to the next viewpoint. This could be the case if the view is blocked at this point or the lighting is

disturbing the camera input. When the state outcome is successful, the robot tries to pick up the object. If the object is visible, but there is no feasible way to reach it, the HSR will move to the next viewpoint and try to grasp the item from a different position. Collision checks are done before and during movements. A possible collision stops the execution of movements. After a stop, the robot goes into the neutral position and tries the pickup again. In case of a successful pick, the suction cup pressure sensor is used to make sure an object is sucked up. If the pressure sensor cannot detect anything, the robot repositions itself and tries to locate the item again. When the suction cup absorbs something, the HSR moves to the predefined drop point and executes a handover or a lay-down operation optionally.

The presented state machine is a basic structure to pick up objects. The state machine can be added as a sub-state machine in a more advanced program. The necessary states can be adapted with additional problem handling by simply adding more transitions. As explained in the following sections, most states are also independently executable for easy debugging, testing, and integration into other programs.

4.2 Finding Objects

The goal of the object-finding state is to reliably detect and locate a specific object. As the core platform of this task, Detectron2 [15] from Meta is used. The picked framework for detection and segmentation is Mask R-CNN, which is implemented in Detectron2. An explanation of Mask R-CNN is given in section 2.2. Before the detection system can be trained, a dataset must be created. There are a variety of datasets available, which include several objects. One is the COCO (Common Objects in Context) dataset as described in [33]. The collection includes images with annotations, bounding boxes, and segmentations for over 80 different object categories. As a state-of-the-art dataset, COCO is widely used to develop object detection models. Tiny or flat objects, which should be covered in this thesis, are not available in the COCO dataset. Also, other datasets and pre-trained models do not focus on tiny objects. Therefore an individual dataset was created for the object detection part of this thesis.

In order to create the dataset, three object classes were defined: card, box, and coin. All three object classes can be seen in Figure 4.2. The card is a regular ID card with standard measurements (86 mm x 54 mm) from the TU Wien, containing a few text blocks, logos, and an image of a person on a white background. The second class "box" should represent different boxes used in a household environment. As a training object, the "gelatin box" from the YCB object set [34] with the dimensions 85 x 73 x 28 mm was used. The YCB object set is a collection of various household items to provide a standard object set for benchmark tests of different robots and manipulators. The last class "coin" is chosen as a test for the robot's visual system and manipulator precision. A detection system capable of detecting those three objects could not be found and therefore a custom object detection was needed. Over 120 pictures were taken of the items in different environments alone or with other objects around. The pictures were reduced in resolution and segmented with the software "labelme" [35]. These segmentations are visible as green lines and points in Figure 4.2.

Figure 4.2: A training image with the three object classes: card, box and coin

After all objects in the image set had been labeled, the training of the Detectron2 detector was done with the official tutorial [36]. As the basic model, the pre-trained backbone version "mask_rcnn_R_50_FPN_3x" was used. The model has already learned to detect and segment the different classes of the COCO dataset. This pre-training makes it possible to re-train the model for a different class with only a small amount of training images. The focus of this thesis is not to develop a ready-to-use prototype but to test the feasibility of a pick-and-place task. Therefore this simple object detection approach is sufficient in terms of tiny objects in a not too crowded area. A stable and reliable model needs more training data and further investigations. By using the tutorial's training settings, new model weights for the three chosen classes were obtained. Those weights were loaded with the Detectron2 service from the TU Wien vision for robotics research group [37]. The service was modified to be able to load custom model weights and define self-created metadata. With the service subscribed to the topic of the HSR's RGB-D camera, Detectron2 is able to do an object detection of the actual image on request. In Figure 4.3, the output of the model can be seen. This output includes the name, the segmented region, and the bounding box of detected objects. The percentage indicates the model's confidence in a guess. The threshold to neglect less confident detections is set at the start of the service.

The *Find_Object* state is created as a normal SMACH state, calling an action server named *Find_Object_Action_Server*. All relevant parts of the finding process are done inside the server. The benefit of separating it from the normal state is the ability to handle the different results and to execute a finding operation alone. To call the action server, a corresponding action message is used. The message includes a string with the sought-after object name as the goal and another string as the result to provide information on how the state succeeded. A given goal can either be succeeded, aborted, or preempted by the action server. If the known problem that the object could not be found occurs, the server is set to succeeded. The problem information is written into the result message. This message is then evaluated in the SMACH state, and depending on the result, the state will transfer to *Move_To_Next_Viewpoint* or *Pick_Object*. If there is an unknown error or a timeout, the action server will be set to aborted, and the state will transfer to the

Figure 4.3: The HSR detects a card and a box

exit. During the execution of the action server, the Detectron2 service is called. The robot lowers its head to get a slightly different view if the Detectron2 model is not able to find the specific object. Sometimes, light irradiation affects the image, and the state can find the target with a second try.

To identify the position of an object, the center of the item's bounding box is chosen as the object's location on the image. This localization is shown in Figure 4.4 with the image coordinates x and y. If more objects of the same type appear in the image, the closer one to the lower edge is picked. With the x and y values, the information of the same point in the depth image is extracted. The x, y, and z distances to the RBG-D camera are stored for each point in the depth image. With the actual camera position known, this distance can be used to localize the object in the room, as shown in Figure 4.5. Visible in the left Figure, the found card is located with a published coordinate frame using the "tf" package in ROS [38]. The orientation of the detected object coordinate system is the same as the "base_link" frame, which marks the position of the robot base.

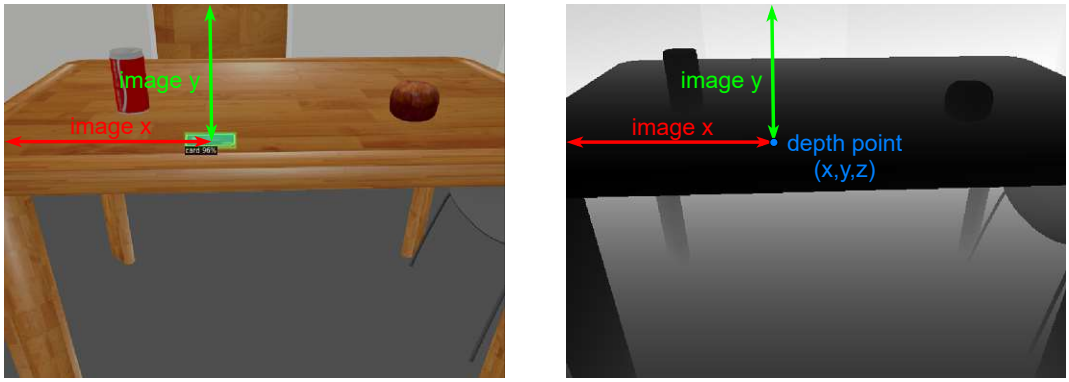


Figure 4.4: RGB and depth image of the HSR camera

Therefore, the three translational DoFs of a found object are determined. The same angles as the robot's base are used to define the rotatory DoFs. The angles around the x- and y-axis are always given by the surface the object is lying on. For a household environment, it is likely that this surface has the same orientation as the ground floor. The third angle around the z-axis, is the same as the z-angle in the base_link frame. This angle affects the grasp pose generation in the next state. In cases without obstacles between the base and the object, the robot can drive from any wheel position straight to its goal to pick it up. This is possible due to the used dual-wheel caster-drive mechanism described in [11]. The robot uses caster wheels in combination with two separately powered wheels mounted on a rotating structure. This mechanism is holonomic and capable of generating speed in all directions. Therefore the resulting wheel position from approaching the viewpoint does not need to be considered for the next state.

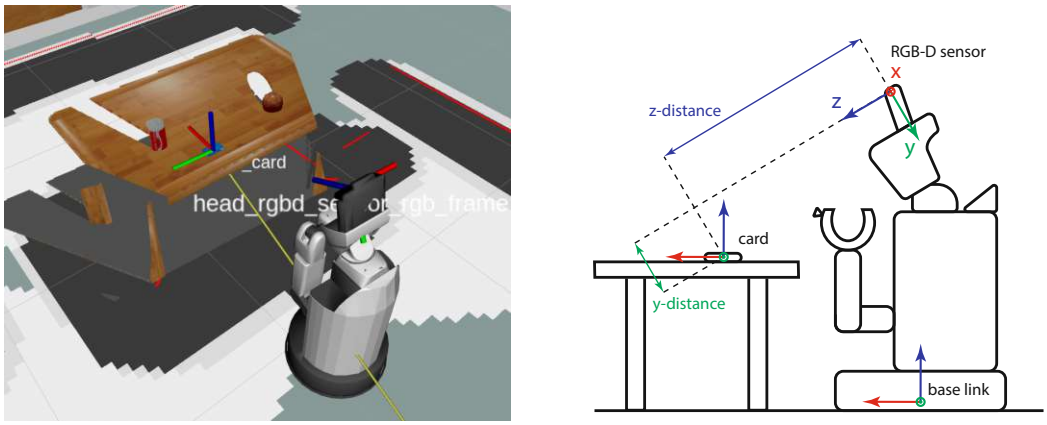


Figure 4.5: Localization of an object

The Figures 4.4 and 4.5 were taken in Gazebo. For fast and easy access to a testing environment for the grasping program, a Gazebo world similar to the experiment location in chapter 5 was created. The standard HSR objects were used, and additionally, a card object with a credit card texture was generated. As noticeable in the figures, the simulated environment looks artificial compared to Figure 4.3. Different lighting settings and additional textures for the floor and the walls could be used, but the Gazebo world is only sufficient for early tests of the *Find_Object_Action_Server*. The main focus was to provide a reliable object detection in a real-world environment. Therefore only non-artificial images were used as training data, and an actual examination of the program was done in a real-world setting.

4.3 Picking up Objects

After the HSR has been able to successfully find the sought-after object, the state machine transfers to the *Pick_Object* state. The goal of this state is to provide a stable picking process with the suction cup. The focus here lies on tiny objects, which can vary in length and width but only have minor deviations in their height. Therefore the same picking process can be used for the three objects: card, box, and coin. Similar to the previous

one, this state also calls an action server *Arm_Movement_Action_Server*. The action message also contains a string with the result information and, as the goal, either the command "pick_" plus the object's name or "lay_down". The action server is used by the *Pick* and also by the *Lay_Down* state. This is to use only one MoveIt commander for both path-planning operations.

The action server starts with the initialization of MoveIt. Here the motion planner can be chosen, and different options like planning time and planning attempts can be set. In section 2.3, it was mentioned that the standard planner from MoveIt is not available for the HSR at the current state of the used package. Therefore the HSR chooses RRT Connect by default. This planner is a bidirectional version of the basic RRT algorithm. As shown in [21], RRT is outperformed by RRT*, and hence RRT* is chosen as planner. There are additional planners available that work with the HSR. RRT* is sufficient for the pick-and-place tasks investigated in this thesis, and therefore no other planners were used. The planning time is set to 10 seconds to ensure an efficient path and reduce the amount of confusing movements, which can occur if the first feasible path is executed. The number of planning attempts, meaning the number of different tree structures built, is set to 5. After the setup of the planner settings, a box object resembling the ground floor is added to the planning scene. Collision objects get detected by the RGB-D camera, and they are loaded as voxel points into the planning scene. They are visible in Figure 4.6, where the HSR plans a pick move in a real-world environment. The depth estimates of the floor sometimes lead to wrong voxel predictions, which subsequently give false collision warnings for the robot's base. If those warnings occur, the robot cannot move in the evadable collision areas. By adding the floor as a scene object, MoveIt ignores voxels near the ground. The floor and another box below the table plane are visible in green. Depending on the view angle, the table stand is often not detected. If the planner does not consider these collision areas, unwanted contact between the robot base and the stands can happen. Therefore the dimensions of the plane table surface are projected downwards and added as a second collision object. This is done with the *table_plane_extractor_server* from the TU Wien team [39].

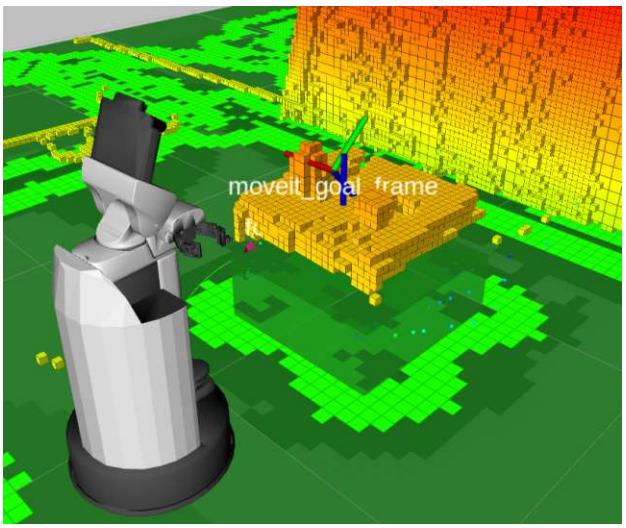


Figure 4.6: MoveIt planning scene in RViz

Pick-up and lay-down operations have the same initialization followed by a check for a published coordinate frame. For picking tasks, the server checks if the frame of the target object is available. If the frame is accessible, another frame indicating the goal position for the end effector is published. For the HSR, the arm's only end effector in MoveIt is the gripper center. The suction cup is mounted on the outside of one gripper. The lack of the ability to use the suction cup as the end effector makes it necessary to calculate the goal position by taking the distance between the suction cup and the gripper center into account. In Figure 4.7, the placement of the goal frame is illustrated. A z-offset is set between the suction cup and the table to ensure no gripper collision during the movement. During picking tests, it was noticeable that MoveIt does not consider the gripper for collision avoidance, but in some cases the z-offset was able to compensate for this. After setting the goal, the planner calculates the path with the given settings. If the planner is successful and a feasible path can be found, the robot starts to execute the trajectory. For cases where a grasp pose from the front is not achievable, another grasp with a 90-degree rotated goal frame is checked. If no path is found for the second pose either, the state machine transfers to *Move_To_Next_Viewpoint*. The robot might get a better position at the next viewpoint, and the entire find and pick process is repeated. During the execution of the picking movement, the HSR moves its arm and base through the room, whereas the depth camera constantly scans for new collision objects. For example, if a person goes through the planned trajectory, the robot will stop, and the *Pick_Object* state will transfer to *Go_To_Neutral*. After the neutral position is reached, the robot tries to plan a new path from its actual position, which differs from the viewpoint.

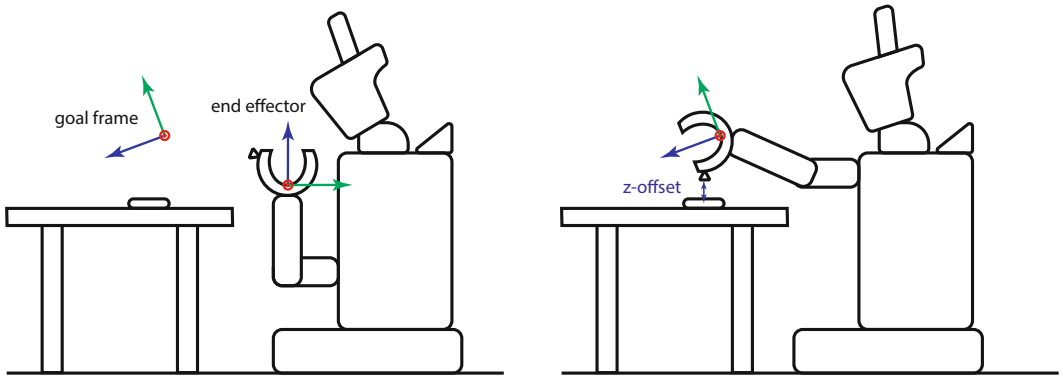


Figure 4.7: Setting the goal frame for the end effector

If the goal frame is reached without complications, as shown in the Figure 4.7 to the right, the HSR will activate the suction cup and move it straight downwards. This movement continues until the pressure sensor inside the suction cup detects a change coming from a stuck object or the table. Next to the successful execution of a picking trajectory, the robot raises its arm and moves slightly backward. In Figure 4.8, the HSR raised the suction cup after picking a card with the first grasp pose. The pressure sensor is evaluated again in the *Check_Pick* state to check the result of the pick after successfully performing all movements. If no object is stuck on the cup, the find and pick operations are repeated with the previous viewpoint. Rare failures with clearly visible and reachable objects can occur by the motor controllers' lack of precision and reliability.

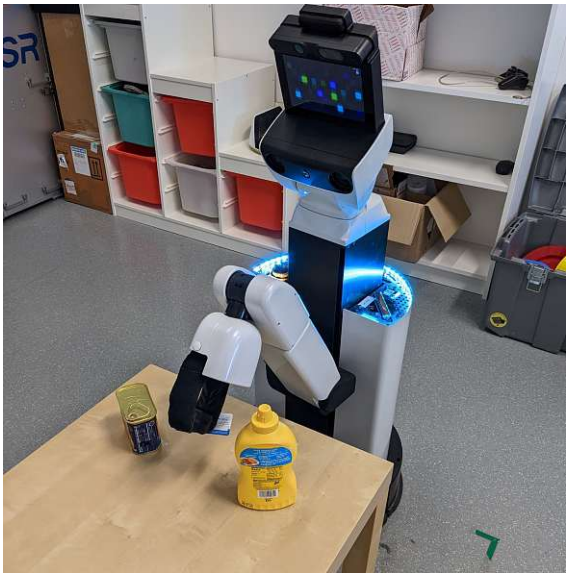


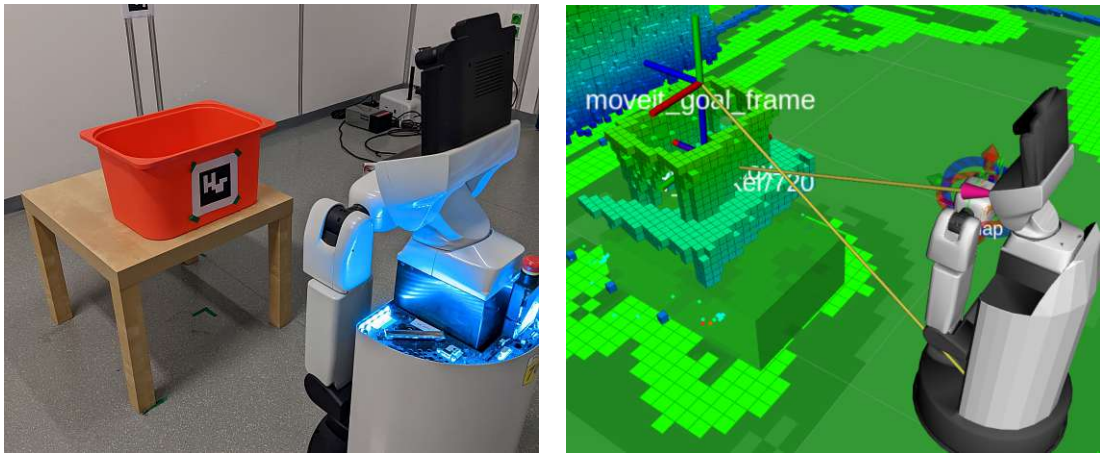
Figure 4.8: The HSR picks a card between obstacles

4.4 Lay-Down and Handover

After a successful pick, the robot will start to move to a predefined point and execute another optional task with the object. The TOP state machine can transfer to three different outcomes. One is to exit the program without any more movements with the object remaining on the suction cup. The other two are the *Lay_Down* and the *Handover* states. These states aim to deliver the picked object either to a human or place it inside a box to tidy up the room. A script from the TU Wien vision for robotics team was used for the handover. It runs another action server, which extends the robot arm so that a person can easily take the object. After the HSR notices a contact force applied on the gripper, it will go to the neutral position again.

In the *Lay_Down* state, the object should be placed in a box labeled with a QR code marker. This can be seen in Figure 4.9 (a). The HSR has a built-in marker detection, which uses the stereo camera. A corresponding coordinate frame is published and named with the marker number if a marker is visible for the vision system. The frame is then used to place another frame above the box as the end effector goal. Shown in Figure 4.9 (b), the goal frame is placed so that the gripper of the HSR will point toward the blue z-direction. Like in the section before, the *table_plane_extractor_server* adds a collision box on the ground and MoveIt uses the rest of the voxels from the table surface and the box for path planning with collision avoidance.

In the Figure 4.10 to the left, the robot has reached the goal frame given to the MoveIt planner. At this position, a conventional method would be just to stop the suction force and invert the pressure inside the suction cup to drop the object. Unfortunately, the mechanic of the HSR is not capable of doing so. If the pressure is turned off, the card is still stuck on the suction cup because of the remaining partial vacuum. Most objects tend to fall after about one minute, which is not an acceptable time for such a task. Therefore another goal for the hand end effector is given to the robot. The new frame is put toward



(a) Box labeled with marker (b) Goal frame above the box

Figure 4.9: Lay-down box in real and in RViz

the negative z-direction of the hand. During the execution, the card is pulled off by the box. The standard planner of the HSR is used for this movement because the goal is just simply moving the arm backward and does not need any consideration of collisions. After reaching the trajectory end, the HSR will return to neutral, and the state machine can exist successfully.

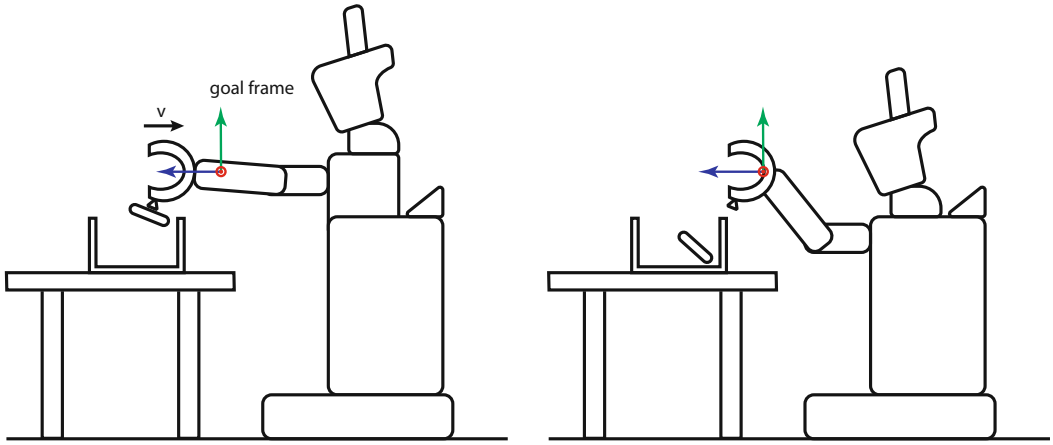


Figure 4.10: Dropping a card in the box

4.5 User Interface

A user interface can be accessed with the Unix shell. It can be seen in Figure 4.11. The left row is the interface controlled by character commands. Here the whole state machine or parts of it can be started. A map object is loaded, which stores the required waypoints for the program. Also, custom maps can be created for new environments. Thereby, new map points can be added by setting the actual position of the HSR as a waypoint.

Feedback for chosen options or state transfers is given on the left side. Information about both action servers responsible for object detection and arm movement is shown on the right. These windows resemble multiple shells running in one. Each sector is a node running the corresponding code and printing feedback. To split the shell window, the software "tmux" was used. In a second terminal window accessible with tmux commands, the *table_plane_extractor_server* and the *handover_server* are started.

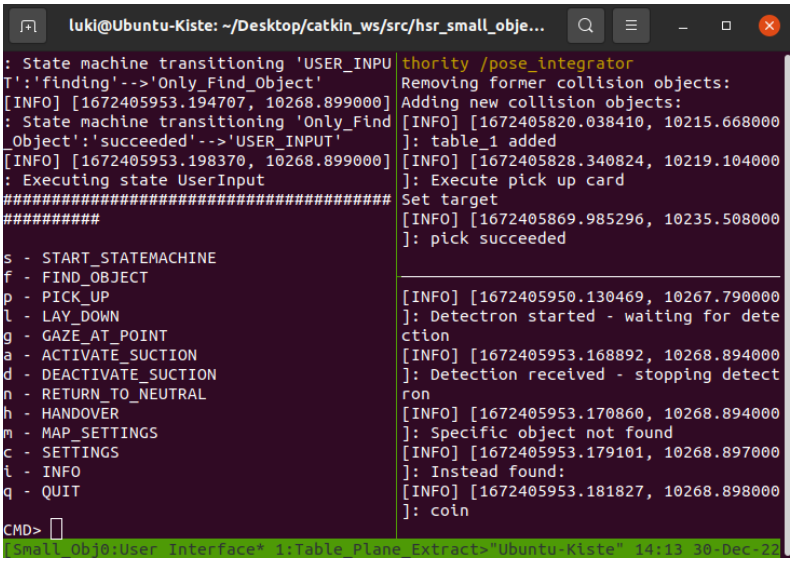


Figure 4.11: User interface with action servers

CHAPTER 5

Experimental Evaluation of TOP

In this chapter, the results and evaluation of the TOP functionalities are presented. The first section describes the setup of the experiments to test the program. Different layouts were designed to challenge the core features of the grasping program. In the second section, the results are presented. Failures, which occurred during the experiments, are discussed in the next section. Thereby, additional program flaws are also mentioned. In the last section, the overall performance of the developed grasping method is outlined.

5.1 Experiment Setup

The goal of the proposed experiments in this section is to give reproducible results to evaluate the core features of the grasping program. The main testing focus lies on the detecting and picking process, which are the essential parts of the given problem examined in this thesis. *Lay_Down*, *Handover*, and other states are developed as additional parts. Hence they are not involved in the following experiments. A scoring system was used to rate task executions and to provide comparable results. A suggestion to benchmark robotic grasping is given in [40]. Inspired by this concept, a scoring system to verify the grasping quality of a robot with a suction cup was created for this thesis. For the sake of simplicity, the suggested system uses a reduced complexity to evaluate a pick task of a tiny object. Five rating categories were defined to investigate the strengths and weaknesses of TOP. One point is given for the successful execution of a category, and zero points are given if the robot fails. If an error occurs, the maximum score of the next rating category is reduced by the number of previous errors. This ensures that each category is evaluated on its own and no prior mistake has an influence on the success rate of the following

categories. The relative success rate is calculated afterward to highlight the weaknesses. The chosen categories are:

- Found: The object was detected and located.
- Planned: The motion planner was able to create an executable path.
- Picked: The object was picked up by the suction cup.
- Stable: The object remains on the suction cup.
- Collision: The pick-up was successful and no other object was touched during the movement (Only at tests with obstacles).

Five different experiment setups were chosen to investigate the program's performance and to show its limits of usability. Four times the ID card was the sought-after object, and one time the gelatin box. In three experiments, the objects "spam" and "mustard" were placed as additional obstacles. Those items are like the gelatin box part of the YCB object set [34]. The chosen experiments are:

1. 25 tries to pick up the ID card from different positions without any obstacles
2. 5 tries to pick up the ID card with mustard and spam as obstacles aside
3. Same as 2. but spam is located in front of the card
4. 5 tries to pick up the gelatin box with mustard and spam aside
5. 5 tries to pick up the ID card on a table tilted by 10 degrees

The HSR starts each attempt in the same starting position to provide comparable results. Figure 5.1 gives a sketch of the setup. This position is used in all experiments. The robot will always start by looking at the gaze point to ensure an optimal view of the table. The point is defined by the distance of 1 m in x and 0.3 m in z from the base coordinate system. A 40 cm distance between the robot and the desk ensures enough arm movement space and also forces the HSR to approach it using the base link drive. The dimensions of the table are smaller than a regular desk or dinner table to enable a direct view of the top surface from the robot's neutral position.

Figure 5.2 shows the different setups for the five experiments. In Figure 5.2a and 5.2b, the 1. experiment can be seen. The pattern is used to position the card after each try. For each of the five card positions, the HSR had to pick up the card five times, resulting in 25 tries for the 1. experiment. Positions one and two are chosen to check if there is a difference between a vertical and horizontal position. The robot must pick the object near the table edge at positions three and four. This is to investigate if possible collisions with the desk stand occur. The fifth location is chosen to challenge the HSR's reachability of distant objects. After the card was placed, the pattern was removed, and only the card remained on the table. In the 2. experiment, the card, spam, and mustard were placed like in Figure 5.2c to investigate the collision avoidance. The added items should represent two typical household obstacles, where the sought-after object lies in between. The same

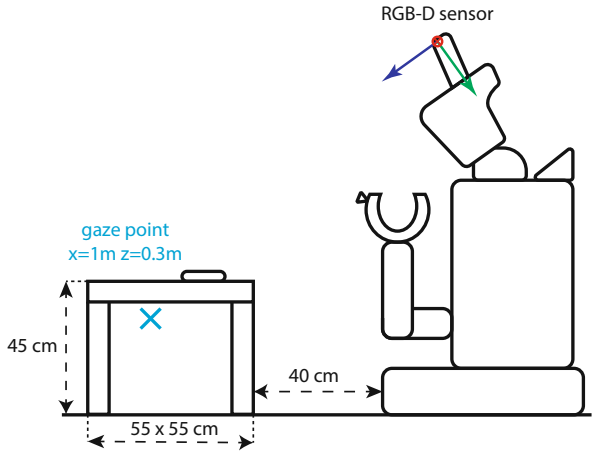
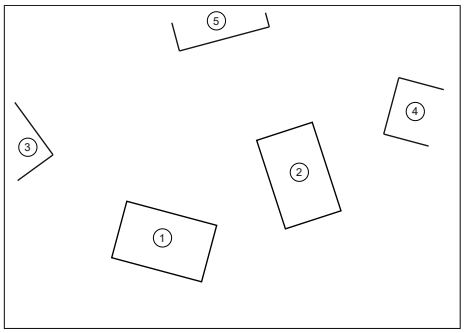


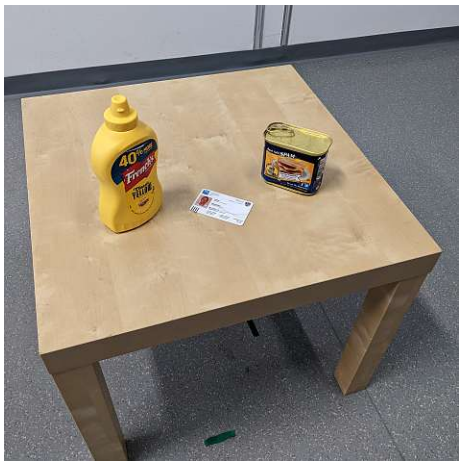
Figure 5.1: Setup for all experiments

three objects but in a different layout are used for the 3. experiment. The spam is placed in front of the desired object to challenge the object detection. The spam did not block the visibility of the card. In the 4. test, the card was replaced with the gelatin box, as it can be seen in Figure 5.2e. This experiment should outline if there are any different behaviors between a card and a box pick-up maneuver. In the last experiment, the table was tilted by 10°, as shown in Figure 5.2f. In this test, the table surface is not parallel to the ground floor, like in the case of a lectern. The grasp pose is still the same as shown in Figure 4.7, with the suction cup direction orthogonal to the ground floor. It should be investigated if the grasp still works even if the direction vector of the suction cup is not orthogonal to the card's surface.

(a) 1. experiment - pattern was removed before start

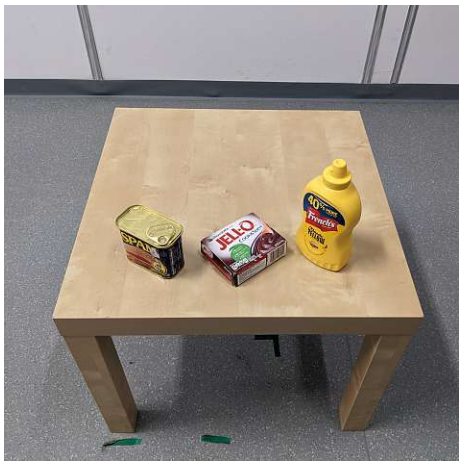


(b) Pattern for card positions in the 1. experiment



(c) 2. experiment - spam and mustard added

(d) 3. experiment - spam and mustard with different layout



(e) 4. experiment - box instead of card

(f) 5. experiment - table tilted by 10°

Figure 5.2: Different setups for each experiment

5.2 Experiment Results

Table 5.1 lists the results from the 1. experiment. Out of 25 attempts, the HSR managed to pick up the card successfully in 23 cases. This success rate shows the practicality of TOP for simple pick-up tasks without any other disturbances. Two failures occurred during the picking movement. Therefore the maximum score in the stable category was reduced two times. In the first picking error, the robot tried to move the arm center above the card. In this movement, the gripper was too close to the card and pushed it away from the original position, resulting in a miss of the card with the suction cup. During the second failure, the movement was aborted due to an error of the motor controller. No obvious mistake could be determined, and the ROS nodes of the program were working correctly. There is the possibility that the failure occurred in the execution of a standard package from the HSR. Besides these two misses, the robot was able to pick up the card from any position with a high success rate. No significant performance differences could be determined between the card's positions. At location five, the HSR used the second grasping pose twice, which resulted in the robot driving to another side of the table. The card was placed by alternating the top and back sides. The object detection was able to detect both sides reliably.

Position	Found	Planned	Picked	Stable	Failure
1	5/5	5/5	4/5	4/4	Card displaced
2	5/5	5/5	5/5	5/5	
3	5/5	5/5	5/5	5/5	
4	5/5	5/5	4/5	4/4	Controller failed
5	5/5	5/5	5/5	5/5	

Table 5.1: Results of the 1. experiment - card only

In the 2. experiment, the robot picked up the card twice without any point deduction. In the first flawed attempt, the spam was slightly touched by the gripper. This did not affect the picking maneuver compared to the fifth attempt, where a collision happened and the robot stopped its motion. If a stronger force is recognized due to a collision, the HSR's motor controller will stop. The third attempt failed due to a motor controller error. Like in the 1. experiment, there did not occur any collision, and no action server seemed to have a problem. Out of these five tries, no problem with the object detection algorithm is noticeable with this lineup of objects. The path planning has problems to avoid objects near the target position.

In Table 5.3, the scoring of the 3. experiment can be observed. This experiment had a high number of failures, resulting in only one attempt without any error. As it can be seen in Figure 5.2, the only difference between the 2. and the 3. experiments was the lineup of the objects. The spam was placed in front of the card. From the HSR's perspective, the card was clearly visible behind the spam, but the object detection failed to find it three times. This is likely due to a lack of appropriate training data for such cases. The training was done without images where items were placed in front of the training objects. Therefore, the detector cannot handle occlusions. If the robot was able to find the card,

Attempt	Found	Planned	Picked	Stable	Collision	Failure
1	1	1	1	1	0	Spam collision
2	1	1	1	1	1	
3	1	1	0	-	-	Controller failed
4	1	1	1	1	1	
5	1	1	0	-	-	Spam collision

Table 5.2: Results of the 2. experiment - card with mustard and spam aside

it could pick up the card and avoid a collision with the spam. During the first attempt, the gripper slightly collided with the table. The depth sensor detected the table, and the path planning algorithm could find a path without any collision. Nevertheless, a collision occurred. A possible explanation would be that the planning algorithm only considers the arm's hand center as the end effector, and the remaining gripper geometry with the suction cup is ignored for collision avoidance.

Attempt	Found	Planned	Picked	Stable	Collision	Failure
1	1	1	1	1	0	Table collision
2	0	-	-	-	-	Card not found
3	1	1	1	1	1	
4	0	-	-	-	-	Card not found
5	0	-	-	-	-	Card not found

Table 5.3: Results of the 3. experiment - card with mustard aside and spam in front of card

The sought-after object was replaced with the gelatin box in experiment 4. As the results in Table 5.4 show, only two attempts were without a mistake. Both failures in attempts two and five resulted from collisions with the box, while the robot tried to reach the MoveIt goal frame from Figure 4.7. In two, the collision was minimal, and therefore it had no impact on the pick-up task. In five, the gripper moved the box too far away, leading to a miss with the suction cup. In the third attempt, a card was detected instead of the box. The similar form of both objects could explain this. With only a slight height difference, both objects appear like a rectangle from the robot's perspective, and the focus of the object detection is more on the form than the texture of the items.

Attempt	Found	Planned	Picked	Stable	Collision	Failure
1	1	1	1	1	1	
2	1	1	1	1	0	Collision with box
3	0	-	-	-	-	Card found instead
4	1	1	1	1	1	
5	1	1	0	-	-	Box displaced

Table 5.4: Results of the 4. experiment - box with mustard and spam aside

The results of the 5. experiment are given in Table 5.5. The experiment was done without additional collision objects, so the collision rating was skipped. As shown in Figure 5.2f the table was tilted. Only one attempt was a successful pick-up. During the other four, the robot failed by displacing the card during the movement. This always happened while the HSR was trying to reach the MoveIt goal frame. The error is likely caused by the path planner ignoring the gripper dimensions. In the next section, the experiment was repeated with an increased z-offset between the MoveIt goal position and the card position to ensure the found trajectory is presumably without a collision between the table and gripper.

Attempt	Found	Planned	Picked	Stable	Failure
1	1	1	0	-	Card displaced
2	1	1	0	-	Card displaced
3	1	1	1	1	
4	1	1	0	-	Card displaced
5	1	1	0	-	Card displaced

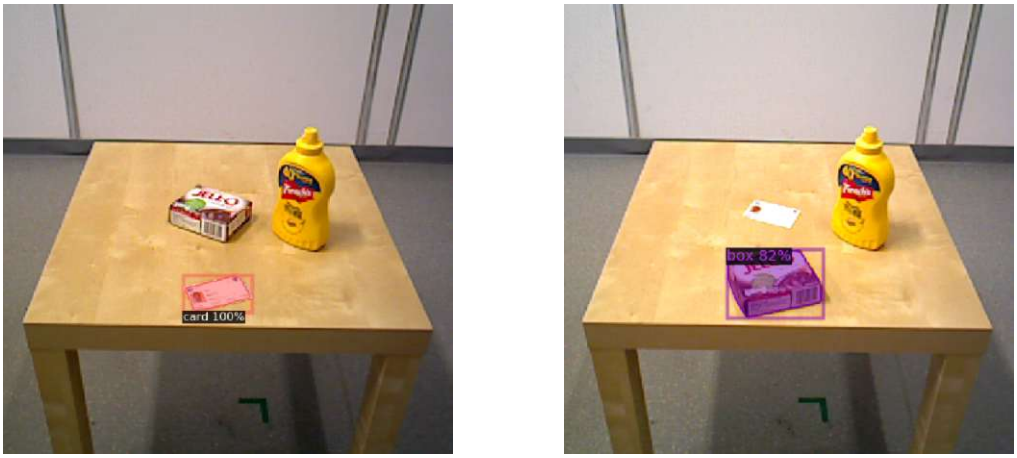
Table 5.5: Results of the 5. experiment - card and table tilted by 10°

5.3 Discussion of Failures

Often occurring mistakes already show up in the 1. experiment. The abrupt stop during the execution of a trajectory, where no apparent mistakes could be determined, needs to be investigated more thoroughly. The second most common failure, the collisions between the gripper and other objects, could be prevented by improving the robot's geometric model used in path planning. The same enhancements can be the key solution for the collisions in the 2. experiment. Besides these errors, no other complications were observed in the other categories in the first two experiment setups.

The failure of the object detection system in experiment 3 is shown in Figure 5.3. The pictures were taken to reproduce the failure after the experiment, and the spam was exchanged with the gelatin box. The object detection algorithm only takes objects into account which are placed in front and neglects those in the area behind. From the robot's perspective in Figure 5.3, both items are clearly detectable for the HSR, but either the box or the card gets detected if they are placed in front. More training data, which includes more cases of various layouts and occlusions, is needed to fix the mistake. The failures in the 4. experiment also have similar causes to those before. The object detection focuses too much on the geometry of the sought-after objects. A better training dataset also needs to take this problem into account. Besides the gripper geometry improvement, a better depth sensor resolution could also lead to less collision.

Due to the bad performance during the 5. experiment, the attempts were repeated with the test of an enhancement. In the repeat, the changed offset improved the success rate, as shown in Table 5.6. The fourth attempt was another controller failure. Those happen sometimes for no apparent reason. Furthermore, the gripper touched the card slightly on



(a) Card gets detected in front (b) Box gets detected in front

Figure 5.3: Object detection failure occurring in the 3. experiment

the last try, and the item was displaced by a few millimeters. Hence the predefined goal for the suction cup was on the card's edge. The card was lifted briefly and dropped caused by the lack of suction power resulting in a zero for the stable category. Nevertheless, the changed offset led to a quick and easy enhancement for such a picking task.

Attempt	Found	Planned	Picked	Stable	Failure
1	1	1	1	1	
2	1	1	1	1	
3	1	1	1	1	
4	1	1	0	-	Controller failed
5	1	1	1	0	Card dropped

Table 5.6: Repeat of the 5. experiment with increased z-offset

Another issue of the object detection, which was not part of the experiments, is shown in Figure 5.4a. A card on top of the gelatin box cannot be detected in the current state of the algorithm. A working prototype of a care robot must be able to manage cases like a crowded household. Therefore additional training images need to be taken where objects appear stacked and in occluded and messy scenarios. Training images of stacked objects would also reduce the strong geometry-focused detection. A coin was chosen as a third target for the pick-up task. It is possible to detect a coin with the HSR's RGB-D sensor, as presented in Figure 5.4b. The success rate of this detection is unreliable due to the low resolution of the image and the strong influence of light disturbances and reflections. Although the robot is able to localize the coin sometimes, the pick-up never succeeds. The localization method and the robot's mechanical system is not precise enough to hit the coin with the suction cup reliably. Besides that, the suction power is not strong enough to hold coins with small pits like standard euro coins. Therefore no other experiments with coins were done. Another pick-up method, which could achieve more precision, should be able to pick up coins with appropriate surfaces. A possible way to achieve more precision

would be using the arm camera. In the first step, the camera is moved above the coin, and in the second step, the hand camera's image is used to reposition the suction cup precisely. Further investigation must be done if such a method could achieve reliable results.

(a) Box gets detected, card on top not (b) Coin can be detected

Figure 5.4: Additional object detection examples

5.4 Performance Evaluation

In Figure 5.5, the results of all experiments are pictured. 45 tests were done in total, including 15 with the additional collision objects rating score. If a failure occurred in one category, the score of the following ones was reduced. All categories were summarized, and the relative score is shown at the corresponding scale in Figure 5.5. For experiment five, the score of the repeated version shown in Table 5.6 was picked. The suggested enhancement of the changed z-offset was added during the failure discussion and this change would not affect the results of the other four previously done experiments.

As visible in the total score, the motion planning algorithm was always successful. If the object could be located, the RRT* algorithm was able to find an executable trajectory in the given time of 10 seconds. All the objects were positioned in reach of the HSR. It should be mentioned that the path planning category just evaluates the capability of finding a path and does not consider the quality of the path. The second best category is the stability of a grasped object. The only failure occurred due to a poor grasp with the suction cup positioned on the edge of the card. If the robot was able to position the suction cup entirely on the object's surface, no problems with the stability were noticed. Various arm movements are executable without the risk of losing the object. Other objects like a euro coin or larger boxes cannot be lifted with the HSR. Nevertheless, objects with appropriate weight and surfaces, like the gelatin box or ID cards, do not lead to any complications in terms of stability. A disadvantage of the mechanics of the suction cup was described in section 4.4. On the one side, the vacuum provides good stability, but on the other side, objects get stuck on the suction cup, making it hard to place them at a specific location.

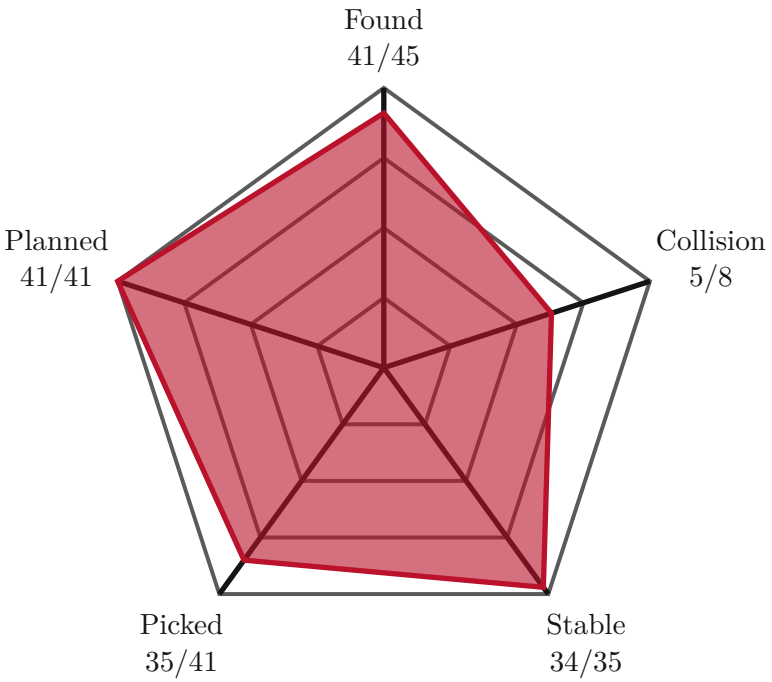


Figure 5.5: Performance of each category (experiment 1,2,3,4 and repeat of 5)

Problems concerning object detection mainly happened during the 3. experiment. As mentioned in the previous section, the focus lies on objects in front, while those in the background are neglected. This apparent weakness must be dealt with more appropriate training data, which should tackle the problems with occlusions, items in the background and stacked items. In terms of more complex tasks, object detection is likely to be the hardest part in the development of a working prototype that is able to operate in a chaotic household. Enhanced detection models can be loaded with the provided Detectron2 service. The TOP state machine could therefore work as a basic framework for more advanced algorithms.

The failures that occurred in the last two categories: "picked" and "collision" can be summarized by two cases. The first is the motor controller failure. During the execution of the motion trajectory, the controller failed three times. A causality between this error and something else could not be determined. After the controller failures occur, the HSR can still execute further commands. Hence, TOP offers a simple solution. As shown in Figure 4.1, the robot would go to its neutral position and repeat the pick-up process. The second error source was collisions between the gripper and various objects, which led to displacements of the sought-after object or movement abortions. In some cases, like in the tilted table experiment, a simple solution is provided by increasing the z-offset between the MoveIt goal and the object. Another suggested approach would be to change the geometry of the HSR's gripper and arm parts used in MoveIt for collision avoidance. Enlarging parts of the stored gripper model would force the planner to ensure a larger safety distance between the robot and objects. Therefore, this change could also lead to an improvement for the motion planning related problems.

CHAPTER 6

Conclusion

In this thesis, a program to pick up tiny objects with a suction cup was developed. TOP was designed for the Toyota HSR, but the concept can be transferred to any arbitrary mobile manipulator. The three main parts of the program are detecting tiny objects, executing the pick-up movement by using a previously determined grasp point, and dropping picked objects in a storage box. The robot localizes the sought-after object using a 2D image detection and a depth sensor. A grasp point for the suction cup is determined by this localization without the need to identify the other three rotatory DoFs of the object. This solution exploits household object geometries, which often feature flat surfaces, to simplify the given grasping task. A motion planner is used to reach the grasp point while avoiding any collisions. To identify the storage box, a QR code label was used. For a complete pick-up and store task, a state machine was created to manage the transition between the parts and handle possible errors.

Each of the program's three main parts were successfully tested, first in a simulated environment and then in a real-world scenario. Pick-and-place tasks could be executed with an ID card and a box. Five different experiment setups with 45 carried-out attempts in total were done to investigate the object detection and motion planning parts of TOP. The experiments have proven that the robot can reliably detect and pick a card on a table without the presence of any other obstacles. Possible enhancements for better collision avoidance and object detection were observed regarding scenarios with additional obstacles. Besides a card and box, the third target object was a coin. The developed localization method, combined with the robot's mechanics, led to a lack of precision, making it impossible to lift the coin.

Future works can therefore investigate other methods to pick up coins. A suggestion to improve the precision would be using the robot's hand camera to refine an object's position

after a first rough estimation with the depth camera. The Detectron2 service, called by the object detection state, can easily be adapted to test another detection method. With a changed models, the basic system can still be used to grasp objects. An alternative object detection could be an enhanced version capable of detecting items in a typical crowded household area. Another promising direction would be the development of a surface detection system. The TOP grasp pose estimation uses the center point of tiny objects whose height is distinctly smaller than the length and width. Flat surfaces of larger objects could be detected to find an appropriate grasp point for them.

In conclusion, feasibility could be achieved with the TOP grasping method. This offers a variety of possible extensions for the basic structure presented in this thesis. There is still a large amount of research left until a prototype of a complex service robot is ready to operate in the real world. The work done by autonomous robots is increasing rapidly and the future will show what achievements are possible for them.

Bibliography

[1] D. Bauer, T. Patten, and M. Vincze, *VeREFINE: Integrating Object Pose Verification with Physics-guided Iterative Refinement*, arXiv:1909.05730 [cs], May 2020. [Online]. Available: `http://arxiv.org/abs/1909.05730` (visited on 02/22/2023).

[2] S. Thalhammer, M. Leitner, T. Patten, and M. Vincze, „PyraPose: Feature Pyramids for Fast and Accurate Object Pose Estimation under Domain Shift," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 13 909–13 915.

[3] G. Van Rossum and F. L. Drake Jr, *Python reference manual*. Centrum voor Wiskunde en Informatica Amsterdam, 1995.

[4] *Luke-rinney/hsr_small_objects*, en. [Online]. Available: `https://github.com/luke-rinney/hsr_small_objects` (visited on 03/03/2023).

[5] N. Correll, K. Bekris, D. Berenson, *et al.*, „Lessons from the Amazon Picking Challenge," Jan. 2016.

[6] M. Quigley, K. Conley, B. Gerkey, *et al.*, „ROS: An open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, vol. 3, Jan. 2009.

[7] G. Bradski, „The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.

[8] E. Marder-Eppstein, E. Berger, T. Foote, B. Gerkey, and K. Konolige, „The Office Marathon: Robust navigation in an indoor office environment," in *2010 IEEE International Conference on Robotics and Automation*, 2010, pp. 300–307.

[9] D. Morrison, A. Tow, M. McTaggart, *et al.*, „Cartman: The Low-Cost Cartesian Manipulator that Won the Amazon Robotics Challenge," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 7757–7764.

[10] *CRB 1100*, en. [Online]. Available: `https://new.abb.com/products/robotics/collaborative-robots/crb-1100` (visited on 09/26/2022).

[11] T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara, and K. Murase, „Development of Human Support Robot as the research platform of a domestic mobile manipulator," *ROBOMECH Journal*, vol. 6, Apr. 2019.

[12] T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara, and K. Murase, „Development of the Research Platform of a Domestic Mobile Manipulator Utilized for International Competition and Field Test," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 7675–7682.

[13] D. Ciresan, U. Meier, and J. Schmidhuber, „Multi-column deep neural networks for image classification," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, Providence, RI: IEEE, Jun. 2012, pp. 3642–3649, ISBN: 978-1-4673-1228-8 978-1-4673-1226-4 978-1-4673-1227-1. [Online]. Available: `http://ieeexplore.ieee.org/document/6248110/` (visited on 11/02/2022).

[14] K. O'Shea and R. Nash, *An Introduction to Convolutional Neural Networks*, 2015. [Online]. Available: `https://arxiv.org/abs/1511.08458`.

[15] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, *Detectron2*, 2019. [Online]. Available: `https://github.com/facebookresearch/detectron2`.

[16] K. He, G. Gkioxari, P. Dollár, and R. Girshick, *Mask R-CNN*, arXiv:1703.06870 [cs], Jan. 2018. [Online]. Available: `http://arxiv.org/abs/1703.06870` (visited on 11/28/2022).

[17] S. Ren, K. He, R. Girshick, and J. Sun, *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*, arXiv:1506.01497 [cs], Jan. 2016. [Online]. Available: `http://arxiv.org/abs/1506.01497` (visited on 11/29/2022).

[18] Y. L. Aaron T. Becker, *Probabilistic Roadmap Method in 3D*, en, Feb. 2020. [Online]. Available: `http://demonstrations.wolfram.com/ProbabilisticRoadmapMethodIn3D/` (visited on 11/19/2022).

[19] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, „Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[20] S. M. LaValle, „Rapidly-exploring random trees : A new tool for path planning," *The annual research report*, 1998.

[21] S. Karaman and E. Frazzoli, *Incremental Sampling-based Algorithms for Optimal Motion Planning*, arXiv:1005.0416 [cs], May 2010. [Online]. Available: `http://arxiv.org/abs/1005.0416` (visited on 12/24/2022).

[22] H. Mohammed, L. Romdhane, and M. A. Jaradat, „RRT*N: An efficient approach to path planning in 3D for Static and Dynamic Environments," en, *Advanced Robotics*, vol. 35, no. 3-4, pp. 168–180, Feb. 2021, ISSN: 0169-1864, 1568-5535. [Online]. Available: `https://www.tandfonline.com/doi/full/10.1080/01691864.2020.1850349` (visited on 11/20/2022).

[23] D. Berenson, S. Srinivasa, and J. Kuffner, „Task Space Regions: A framework for pose-constrained manipulation planning," en, *The International Journal of Robotics Research*, vol. 30, no. 12, pp. 1435–1460, Oct. 2011, ISSN: 0278-3649, 1741-3176. [Online]. Available: `http://journals.sagepub.com/doi/10.1177/0278364910396389` (visited on 11/19/2022).

[24] *Available Planners*. [Online]. Available: `https://ompl.kavrakilab.org/planners.html` (visited on 11/19/2022).

[25] I. A. Sucan and L. E. Kavraki, „A Sampling-Based Tree Planner for Systems With Complex Dynamics," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 116–131, 2012.

[26] *ROS/Introduction - ROS Wiki*. [Online]. Available: `http://wiki.ros.org/ROS/Introduction` (visited on 10/01/2022).

[27] *MoveIt-Tutorials — moveit_tutorials Noetic documentation*. [Online]. Available: `https://ros-planning.github.io/moveit_tutorials/` (visited on 10/01/2022).

[28] *ROS/Concepts - ROS Wiki*. [Online]. Available: `http://wiki.ros.org/ROS/Concepts` (visited on 10/01/2022).

[29] *Actionlib - ROS Wiki*. [Online]. Available: `http://wiki.ros.org/actionlib` (visited on 10/03/2022).

[30] E. Ackerman, *Latest Version of Gazebo Simulator Makes It Easier Than Ever to Not Build a Robot*, en, Section: Robotics, Feb. 2016. [Online]. Available: `https://spectrum.ieee.org/latest-version-of-gazebo-simulator` (visited on 10/03/2022).

[31] *Rviz - ROS Wiki*. [Online]. Available: `http://wiki.ros.org/rviz` (visited on 10/28/2022).

[32] *Smach - ROS Wiki*. [Online]. Available: `http://wiki.ros.org/smach` (visited on 10/29/2022).

[33] T.-Y. Lin, M. Maire, S. Belongie, *et al.*, *Microsoft COCO: Common Objects in Context*, arXiv:1405.0312 [cs], Feb. 2015. [Online]. Available: `http://arxiv.org/abs/1405.0312` (visited on 12/16/2022).

[34] B. Calli, A. Walsman, A. Singh, S. Srinivasa, P. Abbeel, and A. M. Dollar, „Benchmarking in Manipulation Research: Using the Yale-CMU-Berkeley Object and Model Set," *IEEE Robotics & Automation Magazine*, vol. 22, no. 3, pp. 36–52, 2015.

[35] K. Wada, *Labelme: Image Polygonal Annotation with Python*, Dec. 2022. [Online]. Available: `https://github.com/wkentaro/labelme` (visited on 12/17/2022).

[36] *Google Colaboratory*, de. [Online]. Available: `https://colab.research.google.com/drive/16jcaJoc6bCFAQ96jDe2HwtXj7BMD_-m5` (visited on 12/17/2022).

[37] *V4r-tuwien/detectron2_ros*. [Online]. Available: `https://github.com/v4r-tuwien/detectron2_ros` (visited on 12/17/2022).

[38] *Tf - ROS Wiki*. [Online]. Available: `http://wiki.ros.org/tf` (visited on 12/17/2022).

[39] *V4r-tuwien/table_plane_extractor: Service for horizontal table plane extraction*. [Online]. Available: `https://github.com/v4r-tuwien/table_plane_extractor` (visited on 12/27/2022).

[40] F. Bottarel, G. Vezzani, U. Pattacini, and L. Natale, „GRASPA 1.0: GRASPA is a Robot Arm graSping Performance benchmArk,“ *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 836–843, Apr. 2020, arXiv:2002.05017 [cs], ISSN: 2377-3766, 2377-3774. [Online]. Available: `http://arxiv.org/abs/2002.05017` (visited on 12/30/2022).

Acronyms

Constrained BiDirectional RRT 2	CBiRRT2
Convolutional Neural Networks	CNN
Degrees of Freedom	DoF
Feedforward Neural Network	FNN
Human Support Robot	HSR
Open Motion Planning Library	OMPL
Probabilistic Roadmap Method	PRM
Rapidly Exploring Random Tree	RRT
Region of Interest	RoI
Region Proposal Network	RPN
Robot Operating System	ROS
Tool Center Point	TCP

List of Figures

List of Tables

Listings