Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Flexible Robotic Grasping

Lev Kisselyov

Supervisor: Ing. Ondřej Novák
May 2021

BACHELOR'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kisselyov Lev** Personal ID number: **483610**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

II. Bachelor's thesis details

Bachelor's thesis title in English:

Flexible robotic grasping

Bachelor's thesis title in Czech:

Flexibilní robotické uchopování

Guidelines:

1) Select at least two objects to show-case grasp planning.
2) Select grippers capable of manipulating the objects.
3) Design and implement an algorithm to determine a grasp pose for a gripper mounted on an industrial robot. The grasp planner
should take into account kinematics of the gripper, environmental obstacles and user-specified object constrains.
4) Design and implement a program for verification of the planning algorithm by simulation for the selected objects.

Bibliography / sources:

[1] Siciliano Bruno, Khatib Oussama - Springer Handbook of Robotics - Berlin Heidelberg 2008
[2] Diankov Rosen - Automated Construction of RoboticManipulation Programs - Pittsburgh, Pennsylvania 2010
[3] Klampt Python API's documentation - Intelligent Motion Lab - 2018

Name and workplace of bachelor's thesis supervisor:

Ing. Ondřej Novák, Testbed, CIIRC

Name and workplace of second bachelor's thesis supervisor or consultant:

Ing. Pavel Burget, Ph.D., Testbed, CIIRC

Date of bachelor's thesis assignment: **15.01.2021** Deadline for bachelor thesis submission: **21.05.2021**

Assignment valid until:
by the end of summer semester 2021/2022

Ing. Ondřej Novák Supervisor's signature	prof. Ing. Michael Šebek, DrSc. Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

. | Date of assignment receipt

Student's signature

Acknowledgements

I want to thank my supervisor Ondřej Novák for his complete support during the entire project flow. I am grateful for his guidance, his advice in writing, implementation, and organizational aspects.

Thanks to Pavel Burget, because Testbed 4.0 for Industry gave me the opportunity and an exciting topic to write my thesis.

Big thanks to all my CTU friends, who always motivated me, encouraged me to solve the most complicated tasks during the study.

Thanks to Tomáš Musil for advising the right things at the right moment.

Special gratitude goes to my friend and colleague Ivan Čermák for all the times he helped me out of dead-ends.

Last but not least, I am very grateful to my family and my girlfriend for the great support during the study and the thesis.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 20, 2021

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 20. května 2021

Abstract

On the wave of an ubiquitous automation tendency, grasp planning automation could reduce the time spent by the operator on planning the movement and manipulation of objects using the robotic arm. Within the project, grasp planning is simulated by applying the `Klamp't` robotic software. The thesis introduces two different approaches to automate grasp planning: empirical and analytical. The empirical algorithm is based on creating a dataset of predefined grasps, while the analytical one focuses on the mesh structure of an object to find polygons matching set conditions for a possible grasp. The presented algorithms support two gripper types, which are used to grasp several uniquely shaped objects. To adapt a 3D object mesh to the application requirements, some techniques such as mesh decimation and subdivision are implemented using Python APIs. Time computational experiment demonstrates the behavior each algorithm exhibits in tasks with varying complexity. Later, examples of possible applications are proposed and visualized. One of the distinctive features of the thesis is a method of defining fragile and forbidden areas of the object by colorizing its mesh polygons.

Keywords: Klamp't, grasp, robotic manipulator, motion planning, flexible grasp, dataset, mesh, decimation, subdivision, faces, manipulation, inverse kinematics, visualization

Supervisor: Ing. Ondřej Novák
Testbed for Industry 4.0,
CIIRC CTU in Prague

Abstrakt

Na vlně všudypřítomné tendence automatizace by mělo být automatizované také plánování uchopovacích úkolů. To umožňuje výrazně zkrátit čas strávený operátorem na plánování pohybu a manipulace s objekty pomocí robotického ramene. V rámci projektu je provedena simulace plánování úchopů pomocí robotického softwaru `Klamp't`. Práce představuje dva různé přístupy k flexibilnímu uchopování: empirický a analytický. Empirický algoritmus je založen na vytvoření datové sady úchopů, které jsou předem definované. Analytický algoritmus se zaměřuje na geometrickou strukturu objektu a hledá polygony vyhovující nastaveným podmínkám pro možné uchopení. Prezentované algoritmy podporují dva typy chapadel, které se používají k uchopení několika jedinečně tvarovaných objektů. Aby síť polygonů 3D objektu byla přizpůsobena požadavkům aplikace, pomocí Python API jsou implementované některé techniky, jako je decimace sítě a její dělení. Byl proveden experiment na měření časové výpočetní složitosti, který demonstruje chování algoritmů v úlohách s různou prostorovou a polohovou složitostí. Později jsou navrženy a vizualizovány příklady možných aplikací. Zvláštní pozornost je věnována vývoji metody pro definování křehkých a zakázaných oblastí objektu pomocí obarvení jeho síťových polygonů.

Klíčová slova: Klamp't, uchopování, robotický manipulátor, plánování pohybu, flexibilní uchopování, dataset, síť polygonů, decimace, manipulace, vizualizace, inverzní kinematika

Překlad názvu: Flexibilní robotické uchopování

Contents

Figures

Tables

Chapter 1

Introduction

This chapter immerses the reader in the problems of robotic manipulation in simulation, defines the goals of this thesis, and the methods that are actively used to solve related tasks.

1.1 Motivation and goals

Nowadays, robotic automation enjoys rapidly growing popularity as a fundamental part of Industry 4.0. Such colossal popularity provokes an explosion in the number of tasks required to automate a particular industry. The typical way to plan a robotic application is to use simulation first. It helps to analyze all possible outcomes accurately, thereby preventing unwanted damage to the robot and the workspace. There are many software packages available to allow the operator to conduct a task within the simulation. Unfortunately, this is an energy-intensive task that takes a high amount of time to be completed. This time can be significantly reduced by automating the process of planning motion, grasping, and manipulating with the object. Many scientific articles associated with automatic object grasping use neural networks, geometric analysis, or empirical approaches for contact generation of robotic grasp. This project focuses on the practical application of theoretical foundations based on the powerful robotic software, `Klamp't`, which enables versatile robotic operations. `Klamp't` allows to numerically solve complex problems necessary for working with manipulator arms, such as solving an inverse kinematic problem and planning a trajectory of movement. Moreover, the whole process is simulated with the use of robust contact detection. The thesis aims to create and analyze a software tool that can automatically perform a feasible grasp by applying geometric filters or creating a dataset of predefined grasps. As the input to the program, objects of various shapes formed by individual mesh structures are presented. Another important aspect that should be taken into account is the preservation of a fragile or unreliable area of an object. The algorithm must prevent interaction with such areas, which expands the scope of its application to situations where the priority is the safety of the object. The workflow of the algorithm is applied to differently shaped grippers suitable for specific tasks and multipurpose.

1.2 Related work

To analyze the state of the art relevant for the implementation, we should research the existing works on grasp planning and its realization. The problem of interacting with the object is a prevalent task that was popular over several decades on the wave of industrial automation and 3D animation. However, it is not new: the attempts to analyze human hand movements to reproduce it using a mechanic manipulator occurred more than a hundred years ago [8]. A geometry-driven approach to solving grasping problems is a widespread way to design, qualify and conduct a prosperous task. Most of the existing research work can be divided into two main categories.

1.2.1 Empirical approach

Empirical approach to grasp planning is conducted by creating a dataset of stable predefined grasps. The foundation of the work by Aydin and Nakajima [2] is that predefined hand postures and trajectories can be computed in advance and stored as a dataset even for such a complicated task as human animation. These already computed grasps can be used to adapt and generate a new, more specific grasp. However, similarly as in [4], or [5], the objects are represented by a compound of primitive shapes. The aim of this method is to reduce the complexity of computation and use already implemented methods for this specific object type. Our approach is not limited to generic object shapes due to the specifics of the algorithm. Li and Pollard applied the concept of storing a database [3]. There, a grasp is produced based on adapted precaptured human movements by matching a hand and an object's shape. Later, the recorded motion data were used by Zhao et al. [6] to simulate physically realistic human grasping. The same idea of saving a dataset of the IK solver results computed in an "offline" program for a particular object is used in this thesis's first naive algorithm. However, as shown later, creating a dataset for each object has its negative sides, such as a lack of flexibility.

1.2.2 Analytical approach

The central concept of the approach is a metric system, which qualifies grasp's potential success based on the object features. Bierbaum, Dillmann, Rambow, and Asfour [7] applied several geometric feature filters to calculate the grasp affordance score. Based on the score, irrelevant grasps are filtered. The one with the highest score is chosen as the first candidate for the simulation. Another geometric analytical approach was presented by Przybylsky, Asfour and Dilmann a year later [23]. There, the authors analyzed 3D objects' symmetry and designed their representation by medial axes. In the second part of this thesis, symmetric features are used as well to find the most suitable feasible grasp.

1.2.3 Simulation tools analysis

The software foundation of the project that is covered in Chapter 2.2 is based on the works of Kris Hauser [10] [21] and his team. The package named `Klamp't` is a powerful tool that offered the resources needed for the thesis. However, several alternatives must be mentioned. Among other robotic simulation software packages, more widespread ones can be named, e.g., are `Graspit!` [17], `OpenGRASP` [18]. While there are some fundamental works on grasping based on these platforms [19], `Klamp't` has a lot to be explored yet. Existing works [22] [20] show the advantages of grasp simulation in `Klamp't`. Those papers single out this software among others for its unique approach to creating robust contacts with objects and terrains. Contacts generated in the simulation tend to have various artifacts. Those can be jitter, divergence, or phantom impulses described in [21]. Hauser also offers methods to avoid these defects.

Chapter 2

Tools, Environment, Setup

This chapter introduces the project's setup and describes the foundation of the work: the software, operating instruments, the robot, and other resources.

2.1 Device Properties

All the experiments within the thesis were conducted on the same device. Its hardware and software configuration defines the computation speed, therefore affecting the results directly.

Component	Characteristics (version)
CPU	Intel Core I5-6200U @ 2.3GHz
RAM	8 GB DDR4
SSD	512 GB
OS	Ubuntu 18.04 LTS
Python	3.6.0

Table 2.1: Properties of computer which was used for performance evaluation

2.2 Software

As was mentioned in Section 1.1, main framework used in this thesis is `Klamp't` (Kris' Locomotion and Manipulation Planning Toolbox). It is a software package for modeling, simulating, planning, and optimization of complex robots [9]. The package is implemented in C++ language and has a `SWIG` binding to Python providing a Python API. The algorithm presented in this work is implemented in Python language, and it is limited to the capabilities of `Klamp't` Python API.

2.2.1 Klampt modules

Klampt provides numerous modules for different computational and visual tasks. The core package is `klampt.robotsim`, which contains definitions of essential elements such as models of worlds, robots, simulation, visualization,

inverse kinematics, and object geometry. All mathematical operations, *e.g.*, vector multiplication, cross product as well as rotation operations in SO3[1] and SE3[2] are implemented in `klampt.math` package providing all the necessary tools for working with 3D geometry. The package `klampt.model` is responsible for the construction of the trajectory of the robot's movements. One of the main things this set of functions can offer is a collision control module, which is the cornerstone of this work.

2.2.2 Klamp't tools

`Klamp't` offers a wide variation of tools to design diverse robotic manipulations. Within the `Klamp't` visualization, the world environment can be built based on an `XML` file. This file contains a robot's definition and initialization, all the objects' and structures' scales and positions, and other helpful simulation parameters.
One of the most potent aspect of `Klamp't` software is that it allows us to solve inverse kinematics based on predefined constraints. Those may be, e.g., three points forming a plane or a specified rotation and translation of any robot's link.
Another aspect worth mentioning is a massive part of the software pack - motion planning. It is a problem of finding a transition between two states of the model. This transition can be realized as a feasible kinematic path or dynamic trajectory under certain robot's workspace limitations [11]. There are several unique algorithms used to prepare a trajectory (a set of milestones). Detailed description can be found in Section 3.3.
`Klamp't` also provides instruments to model applications considering all physical characteristics of robots, their control properties, and virtual sensor data, which are not used in the thesis.

Simulation is a distinctive feature of `Klamp't`. The main difference comparing with the alternative software packages in the method of generating contacts: each geometry object has a thin virtual boundary layer around it, and contacts are detected between those layers [10]. The method makes the simulation numerically stable and allows to work with low-quality, noisy and non-watertight [3] meshes.

2.3 Robot

The choice of the robot KUKA LBR iiwa 14 R820 is supported by the robot characteristics. This robotic manipulator is redundant as it possesses 7 degrees of freedom, which is more than it is needed to achieve any possible position within the robot's workspace [12]. Redundancy means that inverse kinematics has an infinite number of solutions. Hence, greater flexibility in

[1] 3D rotation group
[2] Special Euclidean group
[3] Watertight mesh is a mesh, which consists of one closed surface

finding a grasp is achieved.

KUKA LBR iiwa 14 R820 is a lightweight robot with a maximum load of 14 kg. It can be classified as a so-called collaborative robot, meaning it can cooperate with a human, maintaining a high safety level. Although the manipulator characteristics are distinctive, the presented algorithm can be generalized to almost every other robot model with at least 6 degrees of freedom.

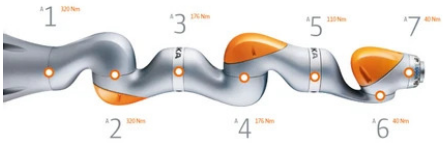


Figure 2.1: KUKA LBR iiwa robot links

(a) : Picture of the robot from the dataset.

(b) : Robot rendered in the simulation configured by the ROB file.

Figure 2.2: KUKA LBR iiwa 14 R820

Files with `.rob` extension represent the model of the robot in `Klamp't`. `ROB` file for KUKA iiwa robot stores the information about the manipulator, such as link structure, joint coordinates, geometry, and physical parameters. The file also defines joint limits and pairs of links, which self-collisions are ignored.

In the picture 2.2b the robot is illustrated in the way it is uploaded to the world within the thesis. The initial pose can be specified in the corresponding `ROB` file.

2.3.1 Grippers

The choice of a gripper is crucial as it defines the application and approach to solving the grasping problem. There is no universal gripper that can be used in every scenario.

PR2 robot's gripper

The first model is a *two-finger adaptive gripper*. This type of gripper is not as common as the standard parallel one. Nonetheless, the fact that they are able to adapt themselves to the different shapes of the object (rectangular, cylinder) makes adaptive grippers far more flexible than their parallel alternatives. It means that the range of applications is broadening: we can perform several equally successful grasps on almost every object's shape without the need of adjusting the gripper manually. Gripper consists of five links, each of them has a driver.

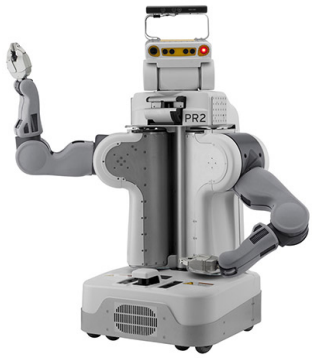


Figure 2.3: Robot PR2 uses two-fingered adaptive gripper

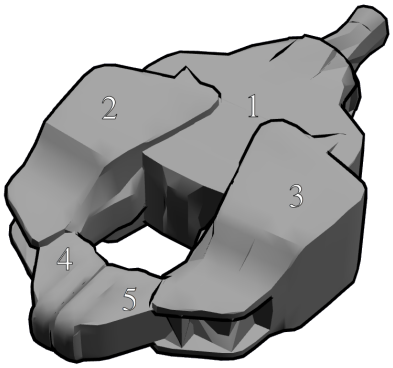
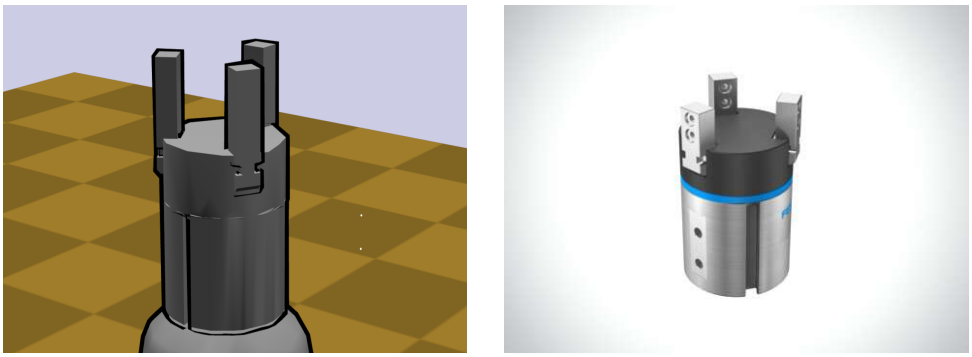


Figure 2.4: Adaptive two-finger gripper rendered in simulation

FESTO parallel gripper

The second gripper introduced in this work is three-finger parallel. It is a popular type of gripper due to its easy construction. The gripper model is called three-point DHDS created and owned by *Festo AG & Co. KG* [25]. Each finger is driven by a prismatic joint and has one DOF.

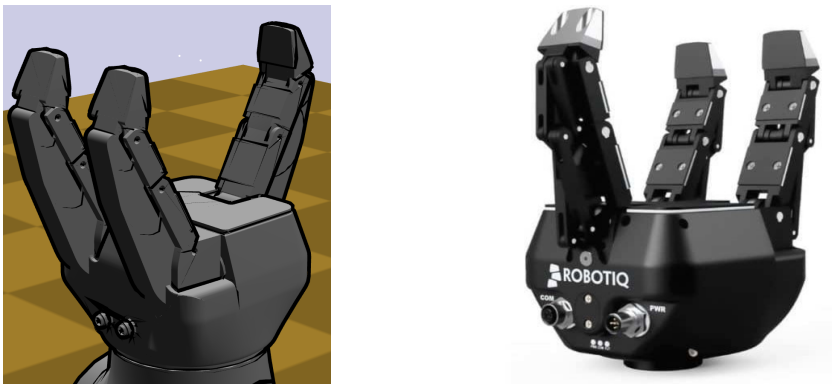


(a) : Gripper rendered in simulation **(b) :** Gripper picture from the datasheet

Figure 2.5: Parallel three-point DHDS gripper

RobotiQ adaptive tool

The final gripper is also three-fingered. Unlike the previous one, it is more complicated. It is patented by the company named Robotiq [26]. The gripper has great flexibility as it consists of 13 links and each one of the fingers is autonomous. This gripper is supported by the simulation and algorithm GUI. Due to its high adaptivity, it can be used to conduct a two-finger and or three-finger grasp. However, the experiments are mostly run on the other presented grippers as they uniquely define each grasp style. The adaptive three-finger gripper can be used for future experiments.



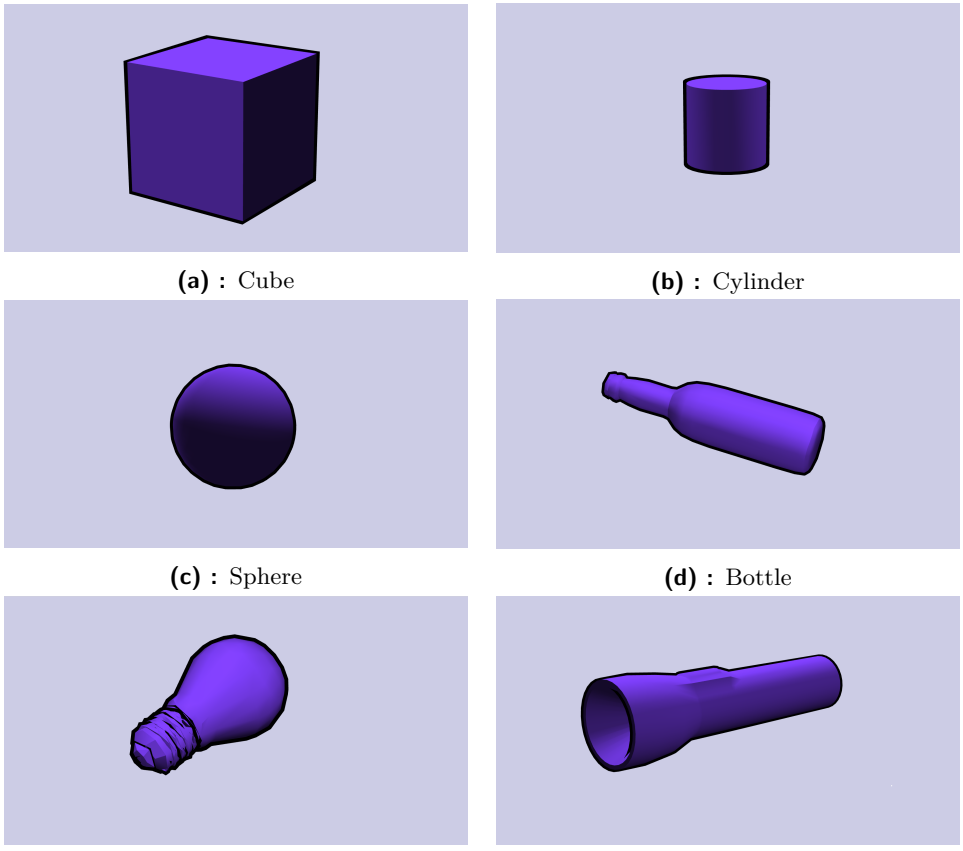
(a) : The gripper rendered in simulation **(b) :** Gripper picture from the datasheet

Figure 2.6: RobotiQ adaptive three-finger robot gripper

2.4 Grasped objects

The algorithm's input will be a 3D model of the object, and the whole project is based on its geometric features. The types of objects presented in this project are various in terms of model complexity. The range of objects used starts with a simple cube displayed on the fig. 2.7a with quite simple mesh structure and ends with objects containing more than 50 thousand mesh elements.

Work with special gripper types also creates limitations on the shape of objects. For example, the parallel three-finger hand is primarily used to grasp objects of cylindric shape or containing some cylindric parts. Therefore, some primitive-shaped objects were selected for experiments pictured in fig. 2.7b and 2.7c.



(a) : Cube **(b) :** Cylinder

(c) : Sphere **(d) :** Bottle

(e) : Light Bulb **(f) :** Flashlight

Figure 2.7: The objects used in the project

In order to introduce the algorithms' capabilities, the most demonstrative type of object has to be chosen. For this reason, one of the suitable candidates is a light bulb (fig. 2.7e). The bulb has a big fragile glass area that is easy to break. Undoubtedly, this area should be avoided to plan the safe grasp in simulation.

Similarly, the bottle (fig. 2.7d) can be easily destructured by a real robot in case of grasping it by the bottleneck. Moreover, the bottle cap may lead to an unstable grasp, which means that the force closure state is not likely to be achieved. Hence, the bottleneck and the cap area should be restricted for the robot.

The last presented object is a flashlight illustrated in the picture 2.7f. In this case, the button area is forbidden for grasp planning.

Chapter 3

Theoretical concepts

3.1 Polygon mesh

In computer graphics and modeling, all objects are represented by a polygon mesh. The mesh defines the shape of an object and the quality of the object's representation in a visualization. Since the thesis is inextricably linked with the work on the object's polygon mesh, it is necessary to describe the most relevant aspects used in it. One of the algorithms works with the triangle mesh concept in which three vertices form every face of the object. Consequently, three points in space define a unique plane. This fact will be used in Section 4.3.1 to compute normal vectors for each face and find a successful grasp.

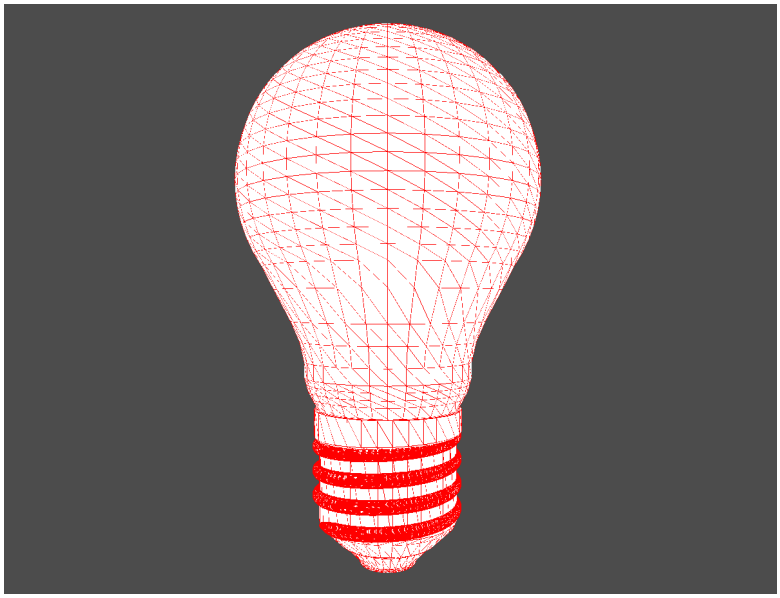


Figure 3.1: An example of a polygon mesh used in the thesis

Working with the mesh is greatly affected by its characteristics. A high number of polygons forming the object can multiply the computational time of the algorithm presented in Section 4.3. Accordingly, the mesh is likely to

be preprocessed[1] most of the time. There are several APIs for Python that provide tools to create, modify, and completely change the mesh structure. Their use is discussed in Section 4.3.3.

The structure of a mesh is wholly defined in a corresponding file with `.obj` extension. The file contains information about the location of vertices, normal vectors, the vertex indices forming a face, and specified material. `Klamp't` library works with its own unique `.obj` file type, which contains the object's simulation physical parameters. It is linked with the `.off` file complementing the main one with information about the faces and vertices. Nevertheless, to establish a stable and sustainable algorithm, uploading of the object should be adapted to the standard `.obj` file format. Trimesh Python library [13] is used to achieve that. It allows to perform various triangle mesh operations, e.g., loading different file types, repairing the mesh structure's mistakes or holes. With the use of trimesh, there can be uploaded common `.obj` and `.mtl` files. Files with `.mtl` extension store information about the object's textures, colors, and materials and are used as companions to the main `.obj` files. Successful loading of both files is necessary to operate on raw vertex and face data (including face colors).

3.2 Inverse Kinematics

One of the fundamental problems in robotics is solving IK (Inverse Kinematics) equations. It means that there have to be calculated robot's joint parameters, which correspond to the specified position of the end effector. Most of the time, for complex robots, this process is being computed numerically.

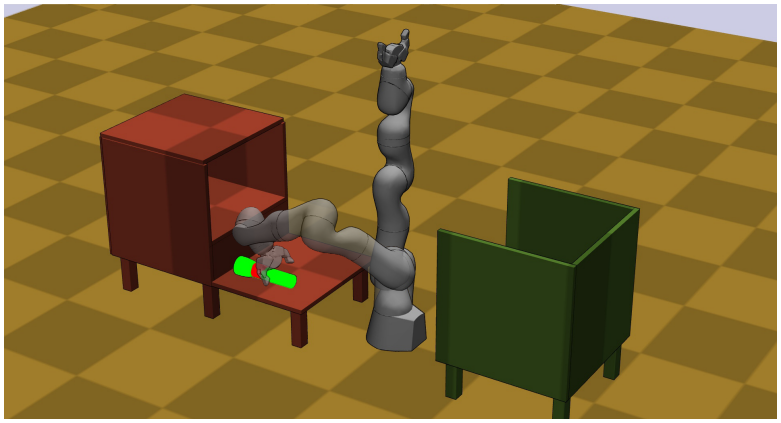


Figure 3.2: With the use of the IK numerical solver, the feasible arm configuration is found to achieve the specified position of end effector in world coordinates.

Inverse kinematics module in `Klamp't` provides IK Solver. It is a numerical solver that considers arbitrary constraints to be added. The solver uses the Newton-Raphson root solving method with line search. This numerical

[1] For example decimated, sub-divided or approximated by its convex hull

method is stable and never diverges [11]. IK solver considers the limitations of the environment, the robot's joint limits, and tries to solve the inverse kinematics task for the conditions specified by a user. The way the solver is modified, the solver distinguishes algorithms from each other.

3.2.1 Different grasp definitions

For each part of the simulation, grasp is defined uniquely. The design of the grasp definition is inextricably connected with the way the IK solver is implemented in `Klamp't`. IK constraints can be, *e.g.*, three (or two) local points placed on the robot and three (or two) points in the world coordinate system; fixed orientation and translation of any of the robot's links.
Three points define a plane in space, which has six degrees of freedom. That, in turn, means that every rotation and translation of these three points in the world coordinate system is distinguished. This approach to design IK constraints of the solver is practical when the grasped object's fixed rotation is needed. The described points are illustrated at picture 3.3.

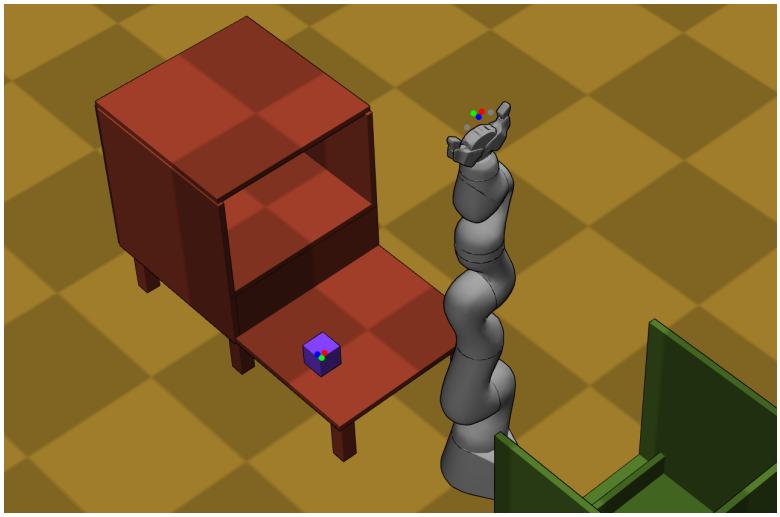


Figure 3.3: IK constraint formed by three points: IK solver connects three points on the robot with three points in the world coordinate system

Two points define an axis (picture 3.4), along which a suitable rotation can be found. That leaves the IK solver one degree of freedom. This approach is beneficial when dealing with cylindric or any rounded objects, which can be grasped from several directions equally.
Constraints on the robot's links are the best way to conduct an accurate grasp in more complicated conditions. For example, when a task involves transferring objects from a box. It would be suitable for this type of application to grasp objects vertically without causing damage to the environment or robot.

It is essential to know that the constraint created by modifying the IK objective applies to a specific robot's link. Most of the time, it is the last link

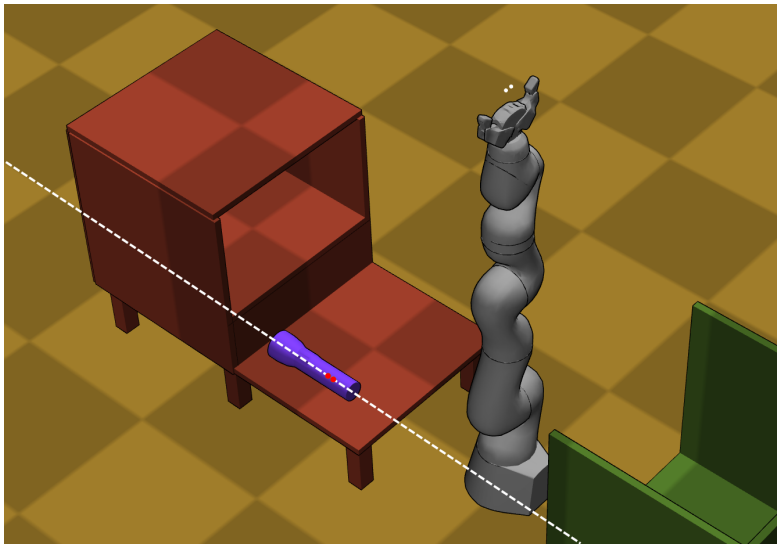


Figure 3.4: IK constraint formed by two points illustrated by an white axis

of the robot connected to the end effector. However, for particular application types, when it is necessary to operate on any robot's link, it can be specified within the IK solver definition.

3.3 Motion-planning algorithms

To plan the trajectory connecting two robot states, a set of milestones must be calculated to interpolate through. A milestone is a state of a robot at a certain point of movement. Each robot configuration is defined as an array of the robot's joint parameters.
Klampt offers a wide selection of motion planning algorithms. The reason to investigate different planners is that their choice significantly affects the computational time. In some cases, the time required to complete the task can be reduced at the cost of a more complicated trajectory containing redundant milestones. Based on this principle, all motion planners can be divided into two categories. Feasible algorithms are the ones that aim to choose the very first possible path without optimization. As a result, the path can be quite unnecessarily complicated. As soon as the optimizing algorithm finds the path, all the next iterations are spent improving the path's quality. Such algorithms are asymptotically optimal but have various convergence rates. Given the time, the solution always has a less or equal number of milestones than a path generated by a feasible algorithm.
Some examples of planners used in the project:

- `SBL`, which stands for Single-query, Bi-directional, Lazy in collision checking algorithm [14]. Its simulation behavior is suitable for fast applications, in case the simplest trajectory is not a priority.
- `RRT*`, `PRM*` - optimizing planners that asymptotically converge to the

optimum, unlike their predecessors `RRT`, `PRM` respectively [16].

- `Lazy-PRM*` and `Lazy-RRG*` are lazy optimizing planners. The key difference from `PRM*` is that collisions with the edges of constraints are not constantly checked. It happens only when a better path is found [15], which significantly decreases the computational time.

3.3.1 Building a path

The most common case of robotic manipulation with objects and environment consists of three stages. Firstly, a `transit` (or pregrasp) trajectory must be built to interpolate from the initial position of the robot to the fixed position defined by the output of the IK solver. The part of the transit stage is also to close the gripper, therefore conducting the grasp. When the interaction with the object is detected, the joints of the robot's gripper stop their movement. The contact is generated, which means that the robot and the attached object are ready for the subsequent motion stage - `transfer` (or ungrasp) trajectory. Similarly, the motion planning algorithm relies on the IK solver outcome. It starts planning a trajectory only in case the final configuration is found. Finally, when the transit and transfer paths are built, the last stage is returning the arm to the initial (or other desired) position - `retract` trajectory. If the motion planning algorithm successfully finds all feasible trajectories, the complete path is concatenated, and the motion takes place.

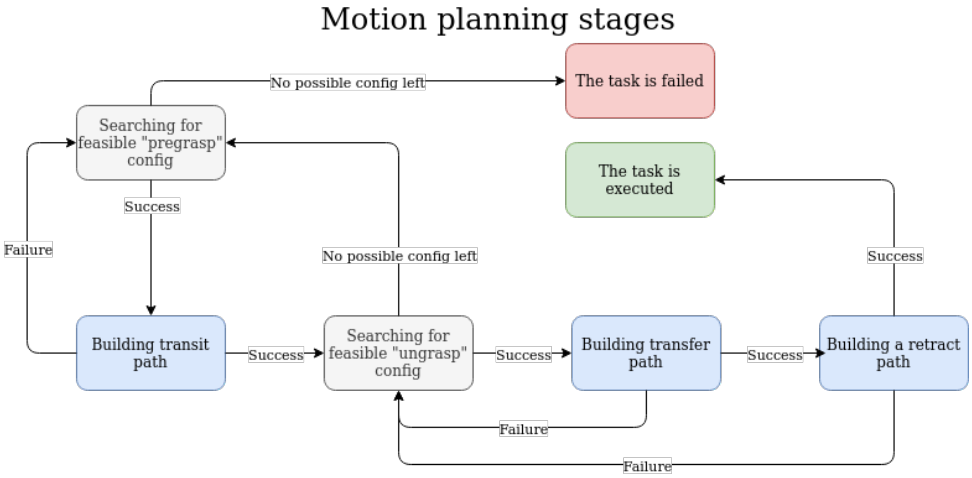


Figure 3.5: Diagram illustrates a task execution as a combination of solving IK problems and planning a feasible trajectory

3.4 Collision-free movements

To preserve the object's initial shape and keep the structures around them intact, all types of collisions should be avoided unless they are designed[2] to be. Therefore, the motion planning algorithms mentioned in Section 3.3 search for a non-collision trajectory based on captured robot-object, robot-terrain, robot-robot (including self-collision), object-terrain, and object-object interactions.
In a running program, collisions are checked at each simulation step, and the collision takes space in case two (or more) objects' geometries collide with a set padding around them. Padding is used to keep a safe distance from each other, leaving a potential implementation on a real robot.
The collision detection can be used for several other purposes, *e.g.*, indicating the contact of the gripper fingers with the object or checking the interaction between the objects involved in a structure construction.

[2]For example collision between the robot's gripper and the grasped object

Chapter 4

Implementation

This chapter introduces an insight into how two algorithms, empirical and analytical, are designed from the programming point of view. For better understanding, the programs' workflow is illustrated in a scheme. There are also presented several support programs required for the algorithm to work in the best possible way.

4.1 Project's structure

Each essential utility functions module, class, or wrapper is stored in a corresponding Python module in the project folder. To understand the basic principles of the introduced algorithms, it is necessary to get acquainted with the relevant utility modules:

- `Hand_analyt.py` and `Hand_emp.py` contain information relevant to the application about the gripper and stores its local parameters. It is special for each type of gripper. The IK solver configuration occurs here as it is inseparable from connecting the gripper's local points to the points defined in the world coordinate system.
- `Motionplanning.py` is a set of functions, which build every part of the path of the robot: transit, transfer and retract.
- `TransitUtility.py` and `TransferUtility.py` are modules that design a configuration space (*CSpace*) of the robot for the current operation. Here can be activated the collision checking between all structures in the simulation world. The user can adjust *Cspace* to the current task, e.g., turn off the collision checking between objects while building a structure with multiple elements.
- `Tk_Wrapper.py` is responsible for the algorithm's startup GUI window.

4.2 Empirical algorithm

The first introduced algorithm is a dataset-driven approach based on brute-force searching and attempting to grasp an object. Work with the algorithm is divided conceptually and practically into two main scripts: offline (`dataset_generation.py`) and online (`online_ask.py`) parts. This algorithm's typical workflow consists of creating a dataset of successful grasps by running `dataset_generation.py` and then using them in the online situation considering only the environment's current constraints.

4.2.1 Dataset generation

In the offline program, a dataset is created individually for each object. The central concept is that there is a virtual bounding box sampled into a set of points. Every point is considered as a place for potential grasping. The points, which are outside of the mesh of the object, are filtered out. We create an IK solver triangle constraint discussed in Section 3.2.1 in the remaining points iteratively. This triangle plane is then rotated by all Euler angles with respect to a fixed coordinate system placed in each point. Therefore, we try every translation and orientation of the robot's end-effector inside the virtual bounding box. Each step has a fixed offset from the previous one. The concept of the algorithm is illustrated in the next pseudocode:

Algorithm 1: Brute Force search for a grasp

```
Result: Array with successful grasps
Define the bounding box;
Define the step between angle samples;
for each roll angle do
    for each pitch angle do
        for each yaw angle do
            for each point do
                Define IK constraint;
                Solve IK task;
                if the task is solved then
                    Store the grasp
                end
            end
        end
    end
end
```

For any current constraint, the output of the IK solver is binary. Hence, we can store the successful grasp defined by the IK constraint in a structure. To make the algorithm more flexible, the constraint should be specified with respect to the object's coordinate system. Therefore, it is translated to the origin of the world coordinate system by subtracting the object position from each point forming the constraint. In the algorithm, the data storage

structure is a NumPy array with `3xNxM` dimensions, where `N` is a number of point samples, and `M` is a number of orientation samples. Once the whole bounding box is examined, we store the grasps in a `.csv` file. This file is going to be used in the real-time program.

4.2.2 Using the dataset in a real-time application

The `online_task.py` is designed in a way that a user can adjust it to any application. The objects with the corresponding dataset can be loaded through the world file or on the run. Initialization of the program includes loading the dataset of grasps from the `.csv` file and sorting them by the cumulative distance from each simulation's geometry element. Sorting the array increases the possibility of finding the successful grasp faster as we first try the ones remoted from potential environment constraints. As the grasps are predefined, we only have to avoid those that lead to collisions with the environment's current state. The first suitable non-collision grasp is executed.

To summarize the empirical approach, the diagram below illustrates the way algorithm is designed to be used:

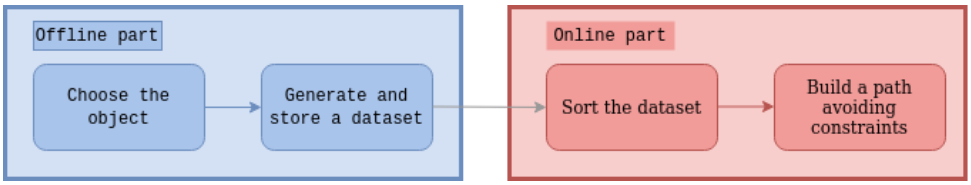


Figure 4.1: The workflow of the empirical algorithm

4.3 Analytical algorithm

In the second part of the thesis, the goal was to eliminate the need to create datasets. Usually, the problem of finding a suitable grasp has a bad computational complexity. The computational time can be reduced by applying geometric filters, thereby avoiding trying by-default infeasible grasps.

Section 3.1 introduced the way 3D objects are represented by a polygon mesh (in this case, triangle polygon mesh). The polygons are used as a feature for planning grasp positions for the robot's end-effector. The central concept is to select the faces of the object suitable for creating contact with the robot's gripper. That means that the algorithm differs depending on the number of fingers of a gripper. For example, to create a stable contact with each finger of a two-finger gripper, a pair of faces should be found. As a result, the complexity of the algorithm increases with the increasing number of contacts needed.

The first crucial step is to compute the needed parameters and geometry features of the object.

4.3.1 Mesh geometry calculation

The orientation of a face of an object is defined by its normal vector. It is a vector that is perpendicular to the surface of a polygon. To analyze and continue working with the mesh, a normal vector should be computed for each face element. The required information for the computation is the position of three vertices $v_1,\ v_2,\ v_3$ forming the polygon in the world coordinate system.

$$\vec{a} = \vec{v_3} - \vec{v_1} \tag{4.1}$$

$$\vec{b} = \vec{v_2} - \vec{v_1} \tag{4.2}$$

$$\vec{n} = \frac{\vec{a} \times \vec{b}}{|\vec{a} \times \vec{b}|} \tag{4.3}$$

Similarly, the center of mass of a face is extracted from the placement of vertices. It is an average of all three coordinates.

$$c\vec{m} = \frac{1}{3} \cdot (\vec{v_1} + \vec{v_2} + \vec{v_3}) \tag{4.4}$$

Working with the object in simulation involves translation and changing its orientation. Hence, the position of all geometry features with respect to the world coordinate system must be aligned with the object's position to conduct the operation correctly. However, on the input, the position of vertices is defined in the object's coordinate system. Therefore, the coordinates of each vertex should be transformed by a corresponding translation and rotation matrix. The rotation matrix R represents the orientation of the object relative

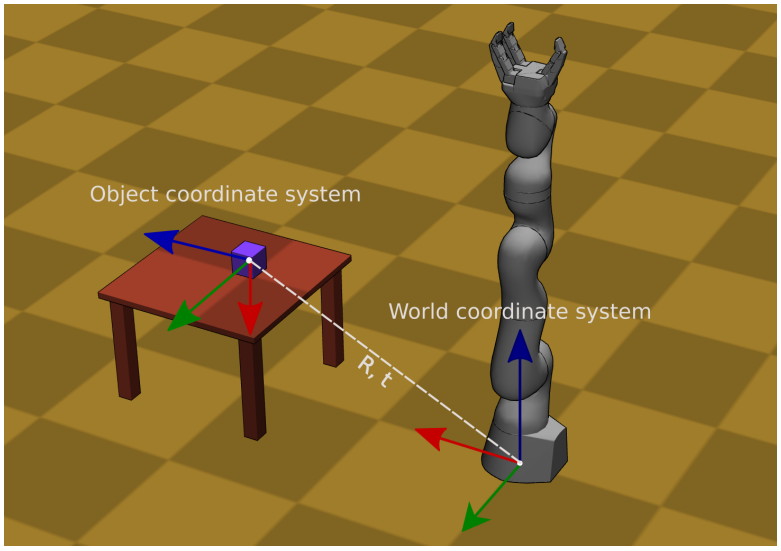


Figure 4.2: Transformation between coordinate systems

to the world coordinate system. Likewise, the position of the object's origin represents the translation $\vec{t}$. The following computation:

$$\vec{v}_{world} = R \cdot \vec{v}_{obj} + \vec{t}, \tag{4.5}$$

where $\vec{v}_{world}$ is a position of a vertex in the world coordinate system and $\vec{v}_{obj}$ is a position of the same vertex in the object coordinate system. Each face is saved as a Python object, and all geometry features and polygon color are object variables.

4.3.2 Heap creation

Now that all faces with their geometry features are available, they are used to distinguish the polygon groups suitable for creating a contact. An array of possible polygon combinations must be built and sorted. For this purpose, heapsort is chosen among the other sorting algorithms due to its efficiency in sorting large data. With the increasing number of array elements, the time required to perform heapsort increases logarithmically, and the algorithm performs equally decent results in the worst, average, and best cases.

Heap for two-finger gripper

The goal of this part of the program is to find suitable pairs of faces. In order to achieve it, the combinations of two faces are examined with the use of the `itertools` Python module. The next step is filtering out the pairs based on conditions:

- The color of the faces - red-colored faces indicate the areas of the object forbidden for the grasp;
- The direction of the faces normal vectors - the faces suitable for contact generation must be oriented in the opposite direction;
- The level of parallelism of normal vectors - to ensure the grasp is stable, the faces must be parallel;

The faces' direction is determined by calculating the dot product of the normal vectors.

$$\vec{n_1} \cdot \vec{n_2} = c \tag{4.6}$$

The operation's output is a scalar c, which must be negative for the faces to be oriented in the opposite direction. Without this condition, the faces aimed in the same direction will not be filtered out, and the gripper may get an infeasible goal configuration.

The cross product defines the parallelism of the faces. Two collinear vectors, i.e., vectors that lie on the same line, have zero cross product.

$$\vec{n_1} \times \vec{n_2} = \begin{cases} \vec{0}, & \text{for collinear } \vec{n_1}, \vec{n_2} \\ \vec{n_3}, & \text{for other } \vec{n_1}, \vec{n_2} \end{cases} \tag{4.7}$$

In the program, the cross product of the vectors is considered zero if the norm of the output vector is equal to zero with predefined tolerance.

If all conditions are met, the pair of faces is appended to the Heap. The sorting factor is the distance between the centers of two faces. It helps to prioritize the polygons located on the same line amongst all other parallel pairs.

Heap for three-finger gripper

The foundation for this part is the analysis of the shape of the parallel three-finger gripper discussed in Section 2.3.1. Each finger is attached to the circle-shaped gripper body. The angle between the radii connecting every two fingers to the center of the circle is 120 degrees.

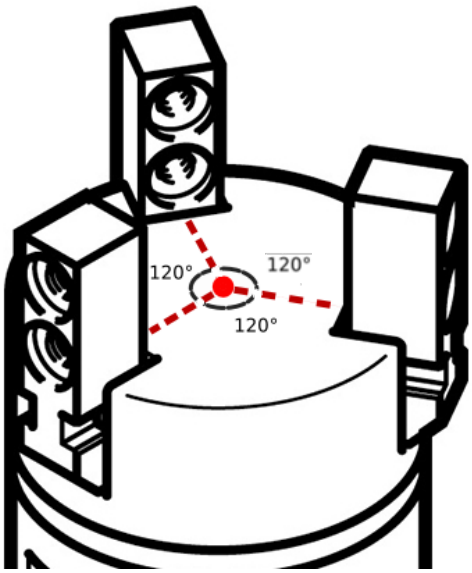


Figure 4.3: Angles between each finger of the FESTO DHDS gripper

Due to this geometry construction, the algorithm is searching for groups of three faces, which also have 120 degrees between each other.

To reduce the computational time, the algorithm does not iterate through the combinations of three faces, but similarly, as in the case of the two-finger gripper, it starts by searching the combination of two polygons. If the angle between two faces' normal vectors is equal to 120 degrees within the specified tolerance, the mesh is examined again to complete the group with the third face.

When the whole group of three faces that satisfy the condition is found, the Heap can be initialized. The Heap is sorted by the area of the triangle formed by the centers of the three faces to ensure that they lie on the same plane.

The tolerance is not a fixed parameter as it is closely connected with the structure of the mesh. The less elements the mesh has - the higher the tolerance should be. On the other hand, to filter out more irrelevant face pairs, the tolerance can be lowered.

Unwrapping the Heap

Once the Heap is initialized, the program is ready and waiting for the user to define the goal position, i.e., the position of the object at the end of the transfer. When the goal is set, the most prioritized group of faces is popped

from the Heap to design an IK constraint. Aside from the unique gripper parameters, such as the offset between the contacts and the gripper body, the concept of the IK constraints definition is also different for two and three-finger grippers.

As the pair of faces in Section 4.3.2 define a single axis, the IK constraint is formed by two points lying on it. That means that one rotation is left free for the IK solver. The principle of the constraint design is discussed in Section 3.2.1.

Even though the two points are more flexible for the solver, this method cannot be used when working with a three-finger gripper. In this case, the group of three faces lies on a plane that has its fixed rotation and position. The IK constraint is defined by three points that belong to that plane.

Summary

The diagram below summarizes all stages described earlier.

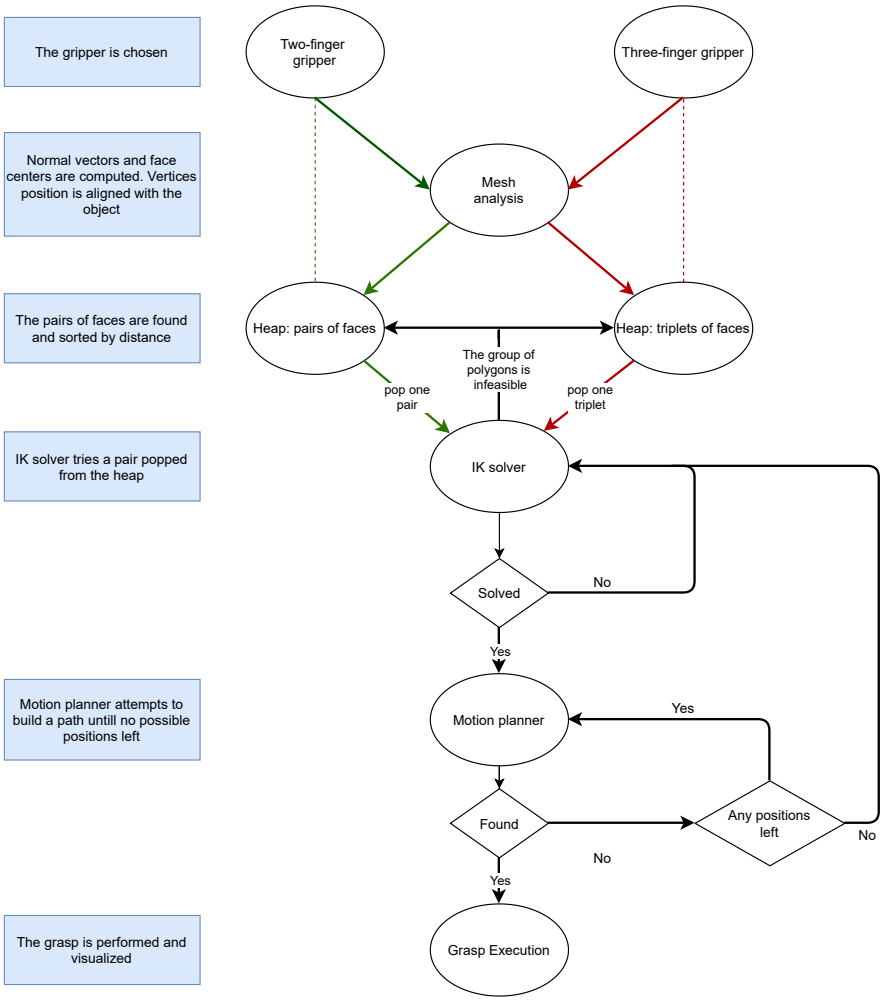


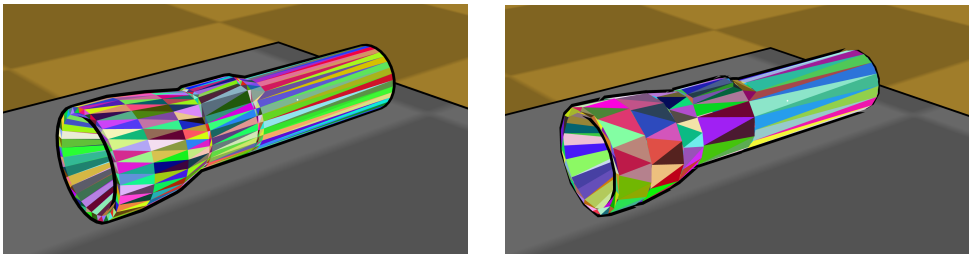
Figure 4.4: Analytical algorithm summary

4.3.3 Subroutines

The main problem with the concept of the algorithm is that the computational time increases polynomially with the number of polygons the mesh has. In contrast, some of the objects, especially those formed from primitive shapes, may lack flexibility in the positioning of the gripper. In other words, to achieve the most desirable results, the mesh should be preprocessed to reach the uniform state. There are several techniques implemented in the algorithm beneficial in various cases.

Mesh decimation

Frequently, the mesh consists of a huge number of polygons. It significantly affects the time required to find a suitable group of faces. Furthermore, most of the triangles would have too little area, which lowers the precision of the contact generation. The solution is to reduce the number of polygons, i.e., decimate the mesh. The mesh decimation within the algorithm is conducted with the use of `PyVista` Python API [27]. It offers a powerful tool for reducing the number of polygons with several implemented methods. The decimation rate is defined by a real number in the range from 0 to 1, which is the ratio between the new and old number of polygons. The Figure 4.5 illustrates the flashlight used as an example of a mesh decimation. Each face is colored in random color to emphasize the decimate effect.

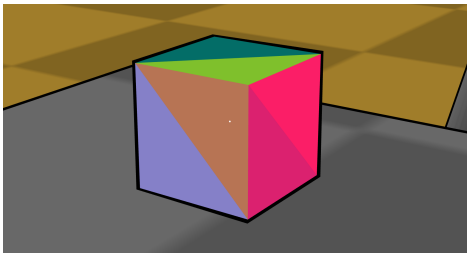


(a) : Original mesh **(b) :** Decimated mesh

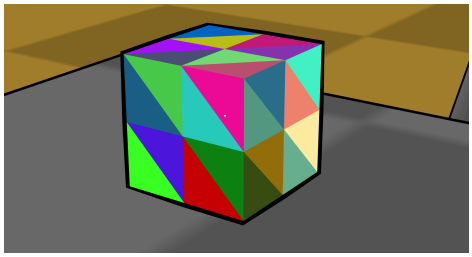
Figure 4.5: Flashlight mesh decimation: 69% of polygons is reduced

Subdivision

Primitive shapes bring up the problem that extensive triangle areas only have one possible position for a finger. The number of potential grasps along one polygon plane can be increased by the `PyVista` subdivision tool that splits the polygon into several new ones. In addition, they have the same orientation as their predecessor, meaning the normal vector is preserved. Trimesh library mentioned earlier also has a subdivision tool, but it is far less manageable.



(a) : Original mesh

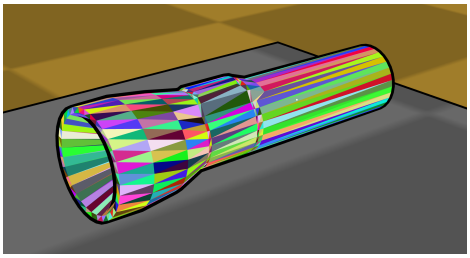


(b) : Subdivided mesh

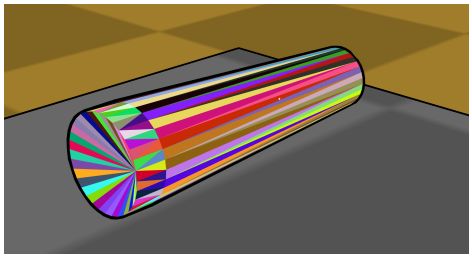
Figure 4.6: Cube mesh subdivision: each polygon is divided into four new ones

Convex Hull

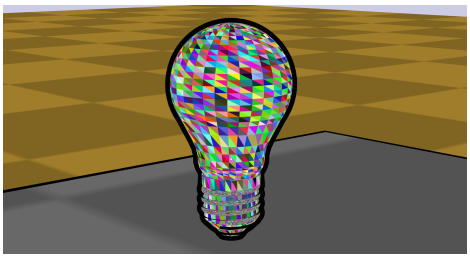
For the objects with too complex structure, the strong decimation might not help. Moreover, it can worsen the quality of the mesh and lead to some visible defects. These defects may affect the direction of normal vectors and cause the undesired behavior of the program. For this reason, the mesh of an object can be simplified to its convex hull. This method significantly reduces the number of faces. The case that benefits the most is when the object's polygons are not evenly distributed. For example, most of the faces are concentrated in the cap of a bulb, and much fewer of them in the glass bulb itself. With the use of a convex hull, the polygons become evenly spread. One of the possible side effects can be that the fragile area might not be easily distinguished, *e.g.*, Figure 4.7d.



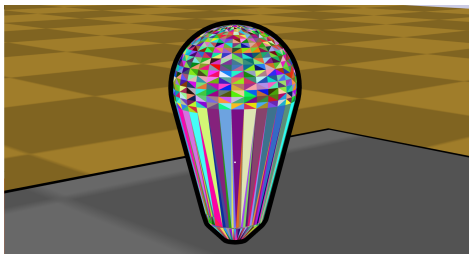
(a) : The original mesh of the flashlight



(b) : A convex hull of the flashlight mesh



(c) : The original mesh of the bulb



(d) : A convex hull of the bulb mesh

Figure 4.7: Creating the convex hull of the objects can significantly reduce the number of polygons

4.4 Fragile and forbidden areas

Some of the objects with complicated or heterogeneous structure require more delicate handling than usual. It implies that some fragile, brittle parts of the object should not be interacted with. As for working in simulation, those parts should be visualized and avoided during the process of contact generation. Therefore, the most suitable way is for the user to colorize some of the faces, marking the forbidden area for grasp with a predefined color. The color chosen for this purpose is red[1]. That means that whenever the program discovers the red color of a face, the face can no longer participate in grasp planning. During the simulation, the red color is associated with collisions on each stage of the trajectory building. The easiest and most common way to colorize the object is to use the Blender application [28]. Blender is a popular and complex software that is applied for every additional mesh upgrade in the project. Using a simple script `blender_colors.py` those colors can be converted into an array and used inside the simulation. A step-by-step guide on emphasizing fragile areas in Blender is included in Section A.4

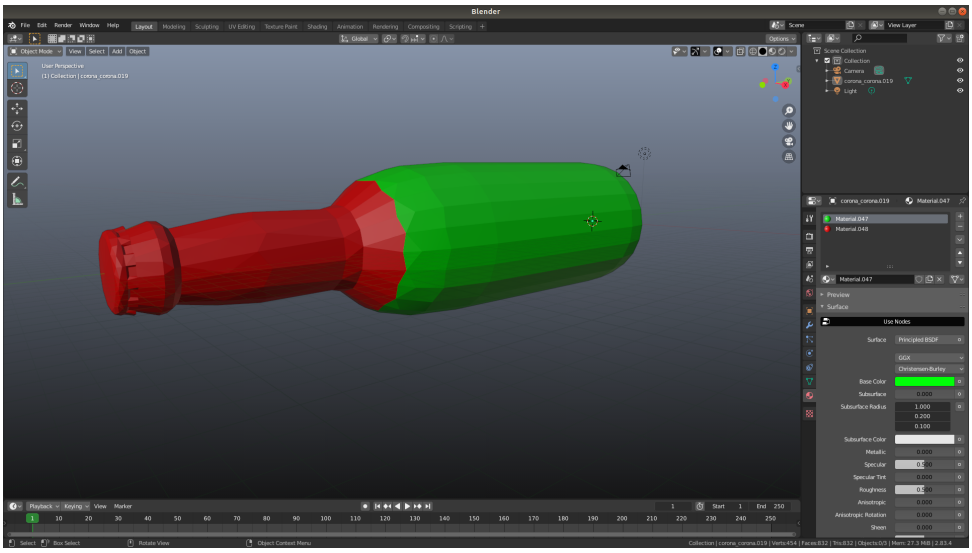


Figure 4.8: The colorized object in Blender ready to be uploaded

[1] In the algorithm red color is represented as RGBA array [1.0, 0.0, 0.0, 1.0]

4.5 Tkinter Interface

Tkinter [**?**] is the standard Python package, which allows to create GUI windows and integrate them with the project. It has been used to build a GUI application, which simplifies the process of starting the algorithm and uploading the objects. For the most common type of application, a user can create a world with one of the presented grippers; change, preprocess and upload several objects into it. The interface is connected to the Blender application[2] in case the user wants to define some forbidden areas of the object or conduct a more complex mesh upgrade. The interface is depicted in Figure 4.9.

[2]For correct use the Blender must be installed.

Figure 4.9: The Tkinter interface helps to easily upload the objects into the world.

Chapter 5

Experiments

The chapter introduces the possible use of the algorithms. Firstly, the average execution time is examined in different cases. The results give a deeper understanding of how the algorithms work and what can be expected while working with them. Secondly, some examples of possible applications are introduced to present possible simulation scenarios. Note that the quality of the final path received from the motion planner is not a priority. Consequently, the time was not spent on optimizing the trajectory. Each object involved in grasp planning is manipulated by a suitable gripper. For this reason, the cylinder and sphere are grasped by the three-finger parallel gripper, while the flashlight, bottle, and bulb are grasped by the two-finger adaptive one.

5.1 Experiment: Dataset generation

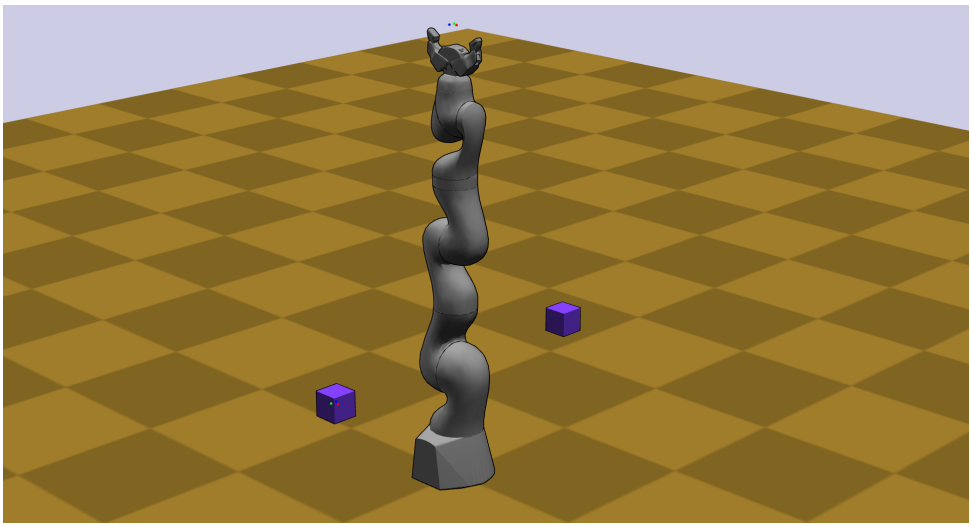


Figure 5.1: Creating dataset with the objects located at both sides of the robot to achieve grasp configurations from both sides.

To initialize the experiment, the empirical algorithm should be prepared, i.e., datasets must be created for all objects. There are many ways to create a

complete dataset of grasps. In this case, the simulation configuration depicted in Figure 5.1 was selected.

The computational time was measured based on ten equivalent dataset generations for each object. Average, worst, and best cases are introduced in the Table 5.1.

Two-finger adaptive				
Object	Computational Time			Valid grasps
	min $[s]$	average $[s]$	max $[s]$	
Cube	35.6101	38.2553	40.7109	89
Flashlight	31.1756	32.87	34.1096	82
Bottle	15.3425	18.2547	21.4158	45
Light Bulb	34.3231	36.7529	40.0723	58
Three-finger parallel				
Object	Computational Time			Valid grasps
	min $[s]$	average $[s]$	max $[s]$	
Cylinder	28.0765	38.3561	42.4624	78
Sphere	21.0843	23.3132	25.4140	52

Table 5.1: Dataset generation time for each object with the use of different grippers

5.2 Experiment: Computational time

The goal of this subsection is to show the results of the analytical algorithm in differently complicated scenarios and compare them with the dataset approach in terms of time efficiency. For the equivalent input, the results may differ, showing stochastic behavior due to numerical inconsistency. Within the experiment, three different worlds are generated. The experiment is performed on various types of objects and two grippers presented in the thesis. For each situation, ten iterations are conducted, and the task execution time is saved.

While working with the analytical algorithm, the task compounds of two main parts, for which the time is measured separately. The first part is the Heap initialization, and the second one is a combination of the transit, transfer, and retract path planning, including the IK problem solution. On the other hand, as the datasets for the empirical algorithm have been precomputed, there has to be found a feasible grasp among them in the online part.

5.2.1 Simple scenario

In this scenario, the task is relatively straightforward. The aim is to examine the diversity level among the simulations with the same primitive input. The situation is captured in Figure 5.2.

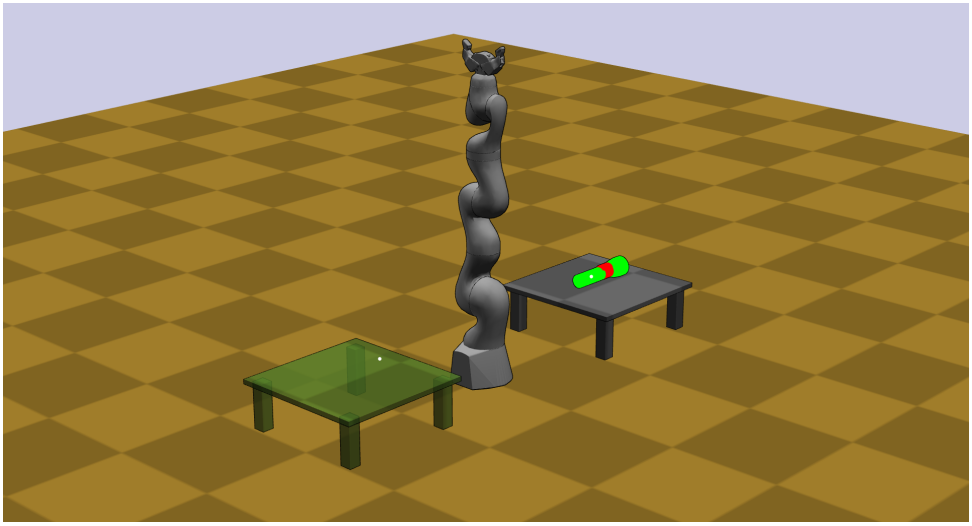


Figure 5.2: Simple scenario: the object must be transferred from one white point to another.

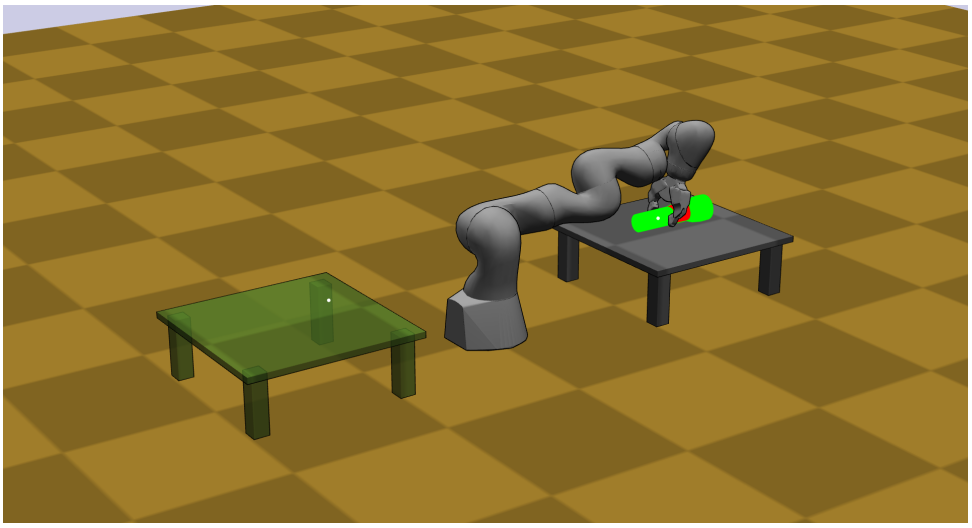


Figure 5.3: The successful grasp is found

The time required to plan the whole task using the analytical algorithm for each object is presented in Table 5.2. The results achieved with the use of datasets in the empirical approach are displayed in Table 5.3. Then, the total time spent on searching for a feasible grasp is compared in Figure 5.4.

Two-finger adaptive							
Object	Number of faces	Heap Creation			Grasp Search		
		min $[s]$	average $[s]$	max $[s]$	min $[s]$	average $[s]$	max $[s]$
Cube	48	0.0275	0.0362	0.0575	1.0192	1.973	3.2091
Flashlight	788	6.6034	6.7547	7.419	0.9797	1.3947	1.662
Bottle	832	4.9176	5.2628	6.2031	0.9746	1.6547	2.1789
Light Bulb	811	12.5136	12.6318	12.9757	0.7458	1.3664	2.2448
Three-finger parallel							
Object	Number of faces	Heap Creation			Grasp Search		
		min $[s]$	average $[s]$	max $[s]$	min $[s]$	average $[s]$	max $[s]$
Cylinder	128	1.4294	1.589	1.9715	0.935	1.2788	1.9469
Sphere	184	23.5981	25.5957	31.2299	1.1595	1.6583	2.4828

Table 5.2: Analytical algorithm performance in simple scenario

Two-finger adaptive			
Object	Grasp Search		
	min $[s]$	average $[s]$	max $[s]$
Cube	0.9064	1.338	1.8306
Flashlight	0.804	1.4147	2.1532
Bottle	0.8879	1.503	1.9527
Light Bulb	1.1527	1.7414	2.7707
Three-finger parallel			
Object	Grasp Search		
	min $[s]$	average $[s]$	max $[s]$
Cylinder	0.9356	1.4737	2.0159
Sphere	0.7423	1.2435	2.9839

Table 5.3: Empirical algorithm performance in simple scenario

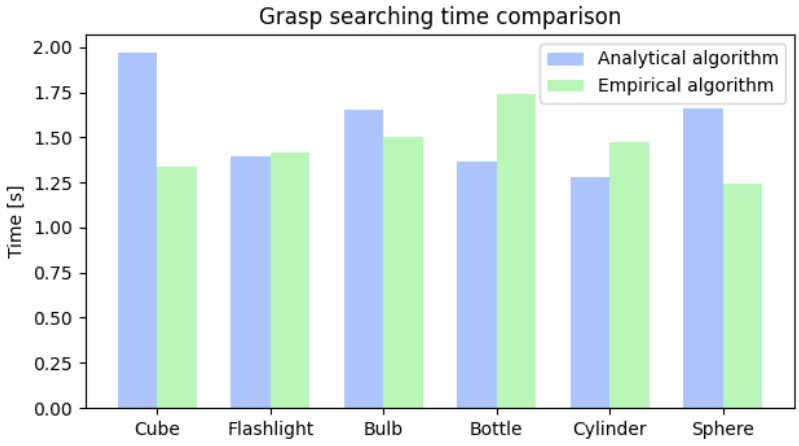


Figure 5.4: Task execution time comparison between the algorithms in simple complexity scenario

5.2.2 Average complexity scenario

The task is more complex as the robot has to achieve a more unnatural position in joint coordinates. Moreover, the possible grasp is located near the wall, which complicates the possible end effector orientation. The situation is captured in Figure 5.5.

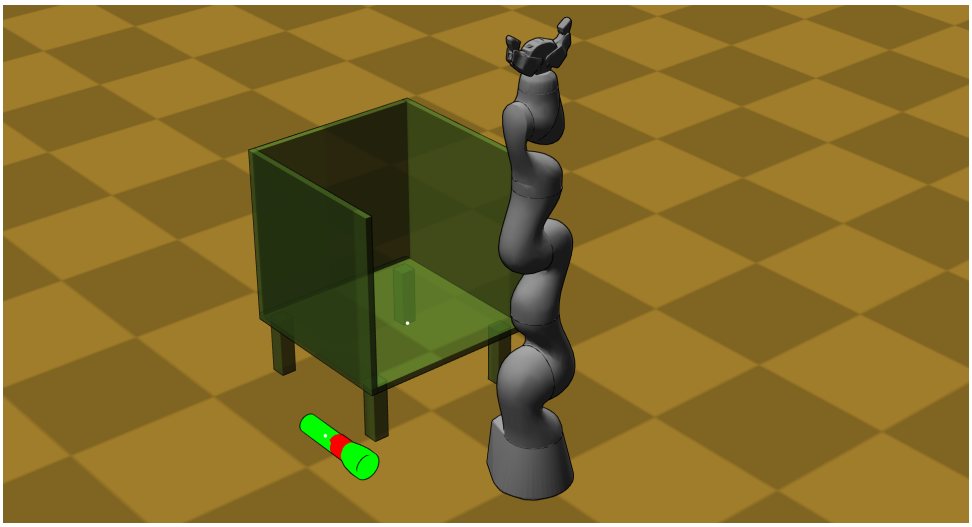


Figure 5.5: Average scenario: the object must be transferred from one white point to another.

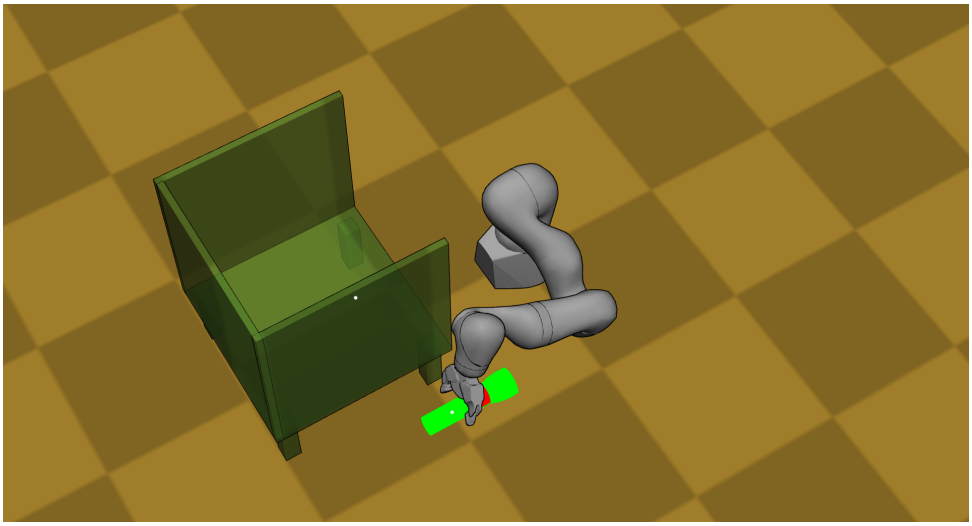


Figure 5.6: Successful grasp is performed.

The execution time of the analytical algorithm is presented in Table 5.4 and compared with the results achieved by using the predefined dataset displayed in Table 5.5.

Two-finger adaptive							
Object	Number of faces	Heap Creation			Grasp Search		
		min $[s]$	average $[s]$	max $[s]$	min $[s]$	average $[s]$	max $[s]$
Cube	48	0.0278	0.0313	0.0547	1.372	3.228	6.7185
Flashlight	788	6.4394	6.5067	6.6389	1.1761	3.028	4.3791
Bottle	832	4.9113	5.8035	6.5252	1.9871	7.1452	19.0238
Light Bulb	811	12.6542	12.7789	12.9685	1.2329	1.921	2.8054
Three-finger parallel							
Object	Number of faces	Heap Creation			Grasp Search		
		min $[s]$	average $[s]$	max $[s]$	min $[s]$	average $[s]$	max $[s]$
Cylinder	128	1.6261	2.2135	3.3801	1.1451	2.2129	3.3603
Sphere	184	20.8357	23.1187	29.0583	1.1931	2.133	3.0299

Table 5.4: Analytical algorithm performance in average complexity scenario

Two-finger adaptive			
Object	Grasp Search		
	min $[s]$	average $[s]$	max $[s]$
Cube	3.5046	4.9682	6.8654
Flashlight	1.7575	3.3571	5.2328
Bottle	2.1485	3.5008	4.57
Light Bulb	2.3949	4.2311	6.4692
Three-finger parallel			
Object	Grasp Search		
	min $[s]$	average $[s]$	max $[s]$
Cylinder	1.1734	1.8645	2.4885
Sphere	0.8649	3.9637	7.9895

Table 5.5: Empirical algorithm performance in average complexity scenario

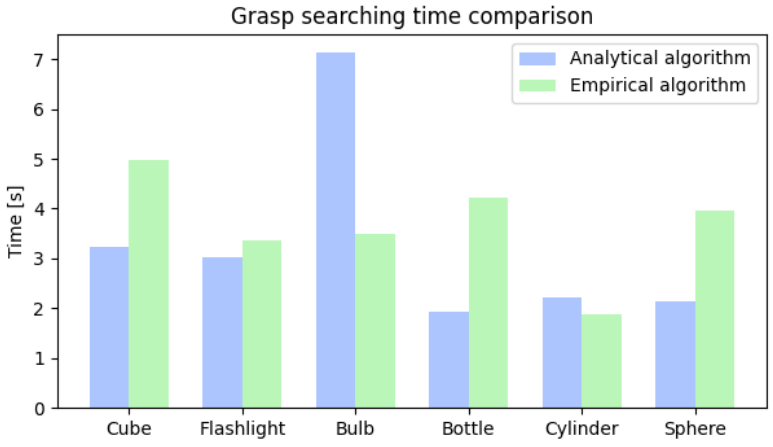


Figure 5.7: Task execution time comparison between the algorithms in average complexity scenario

5.2.3 Hard complexity scenario

The object is located inside the box, limiting the number of possible solutions for the IK problem. The last links of the robot have to be oriented perpendicularly to the table's surface to grasp the object without causing a collision with the box. The situation is captured in Figure 5.8.

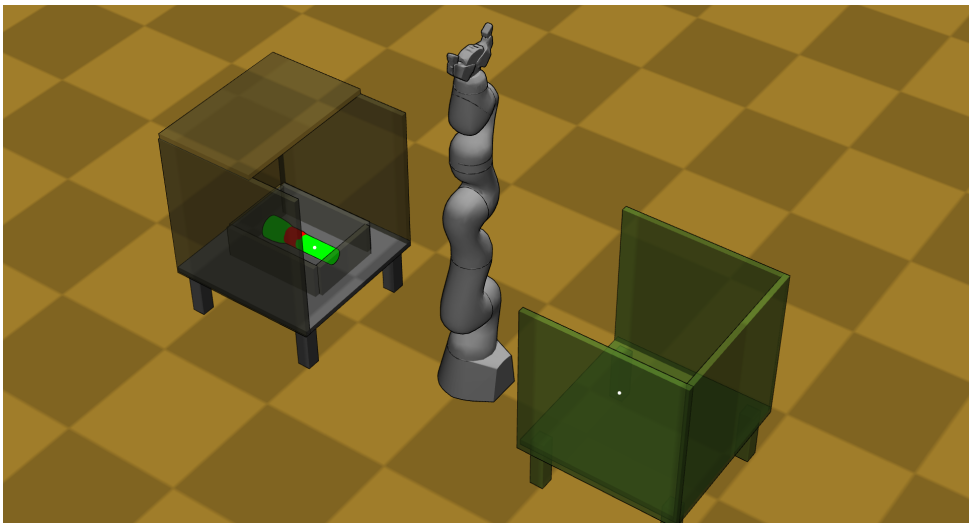


Figure 5.8: Complicated scenario: the object must be transferred from one white point to another.

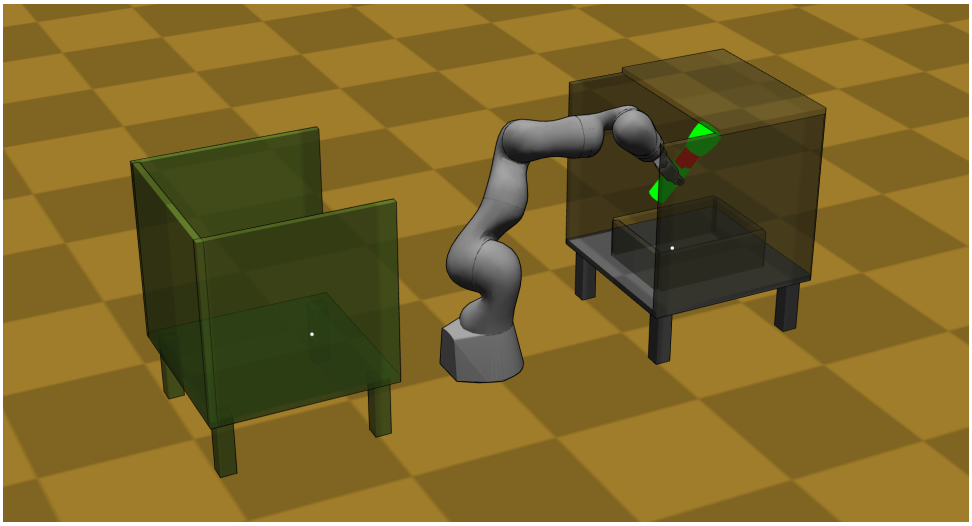


Figure 5.9: The arm carefully retrieves the object from the boxes

The execution time of the analytical algorithm is presented in a Table 5.6 and then compared to the task performance time of the empirical algorithm presented in Table 5.7.

Two-finger adaptive							
Object	Number of faces	Heap Creation			Grasp Search		
		min $[s]$	average $[s]$	max $[s]$	min $[s]$	average $[s]$	max $[s]$
Cube	48	0.0275	0.0294	0.0418	7.4163	12.0252	21.7062
Flashlight	788	6.5287	7.1078	8.7429	7.4	13.3271	22.1319
Bottle	832	5.1008	5.8233	7.0272	8.9978	14.0996	22.2311
Light Bulb	811	12.418	12.778	14.9742	4.2864	8.8352	16.8706
Three-finger parallel							
Object	Number of faces	Heap Creation			Grasp Search		
		min $[s]$	average $[s]$	max $[s]$	min $[s]$	average $[s]$	max $[s]$
Cylinder	128	1.5469	1.729	2.4684	2.0392	5.5146	9.8367
Sphere	184	21.2435	21.6097	22.2232	4.8747	8.9268	12.9684

Table 5.6: Analytical algorithm performance in hard scenario

Two-finger adaptive			
Object	Grasp Search		
	min $[s]$	average $[s]$	max $[s]$
Cube	2.1734	4.0905	5.6801
Flashlight	4.1126	9.125	15.6189
Bottle	3.4374	6.4495	12.2287
Light Bulb	5.5681	12.6943	15.3445
Three-finger parallel			
Object	Grasp Search		
	min $[s]$	average $[s]$	max $[s]$
Cylinder	4.8143	6.8166	11.267
Sphere	3.4985	7.1985	13.1791

Table 5.7: Empirical algorithm performance in hard scenario

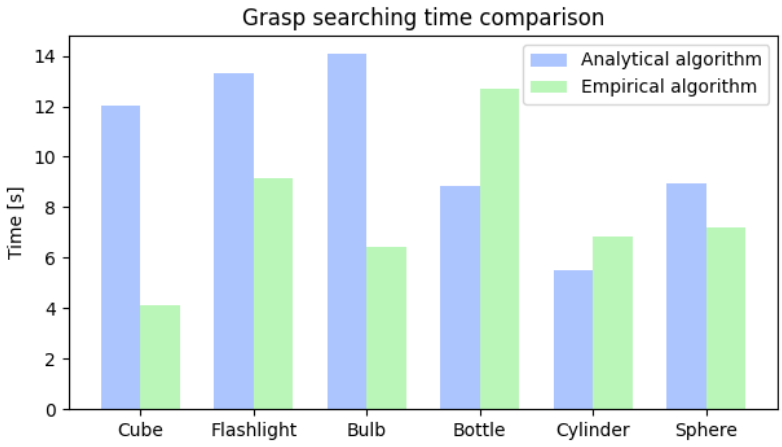
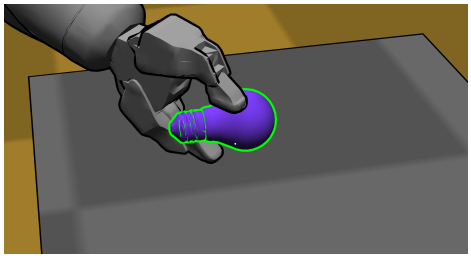


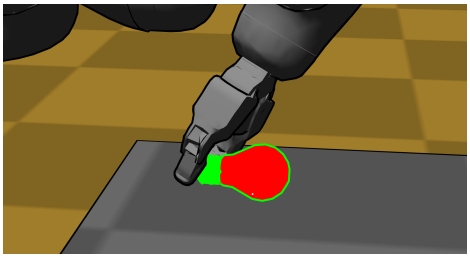
Figure 5.10: Task execution time comparison between the algorithms in hard scenario

5.3 Experiment: Fragile areas

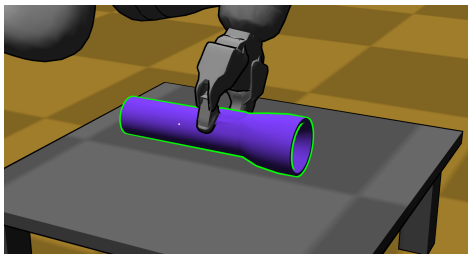
In this experiment, the objects are grasped with and without fragile area definition. The goal is to examine the performance of the colorizing method. Hence, a grasp is planned in identical conditions on the same objects to compare whether the arm detects the forbidden area. The results are pictured in Figure 5.11.



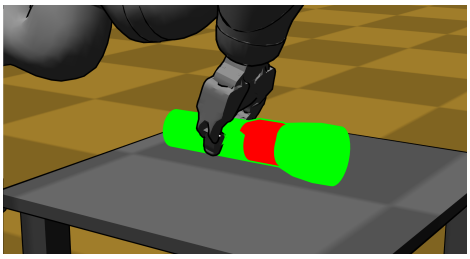
(a) : The bulb is grasped by the glass area



(b) : The fragile area is avoided, the bulb is grasped by the cap



(c) : The grasp is placed at the button of the flashlight



(d) : Button is avoided and the grasp is placed on the body of the flashlight

Figure 5.11: Grasping object with and without colorizing

Within the experiment, the first feasible grasp is executed. In both cases, the fragile area is detected and avoided.

5.4 Application: Building a simple sturcture

The experiment is designed to show the basic concept of manipulation with objects in the robot workspace using the analytical algorithm. The structure is surrounded by walls, which complicates the motion planning task as all the robot-terrain collisions must be detected and avoided. It is crucial to remark that the motion planning algorithm used in the experiment is `Lazy-Prm*` as it affects the computational time as it was discussed in Section 3.3. This algorithm executes an acceptable quality trajectory after a small number of iterations. The robot managed to find a suitable pair of faces to grasp, solve the IK task, and plan the motions to perform the task. The blocks were picked up and placed at the top of each other. The whole experiment run is displayed in Figure 5.12.

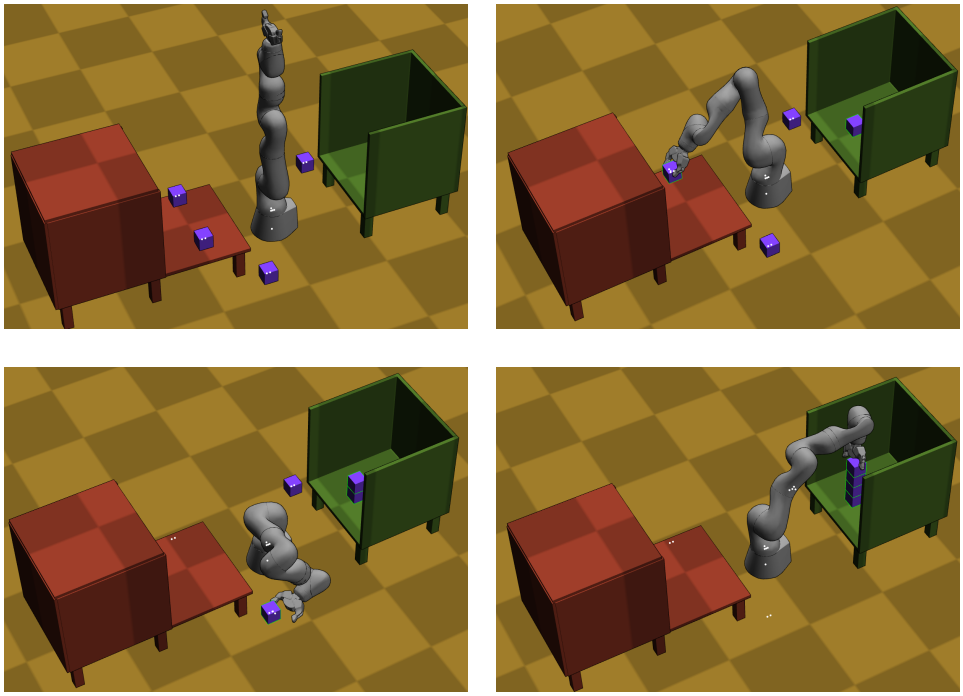


Figure 5.12: The run of structure building application based on analytical algorithm

The task was also conducted using the dataset approach. As a continuation of the previous experiment, the dataset for the cubes is the same described in Section 5.1.
Despite the fact that the task was completed, the grasp quality was slightly lower than in the analytical algorithm. The situation captured in the picture 5.14 is an example of a grasp that might cause problems with force closure in a physical simulation.

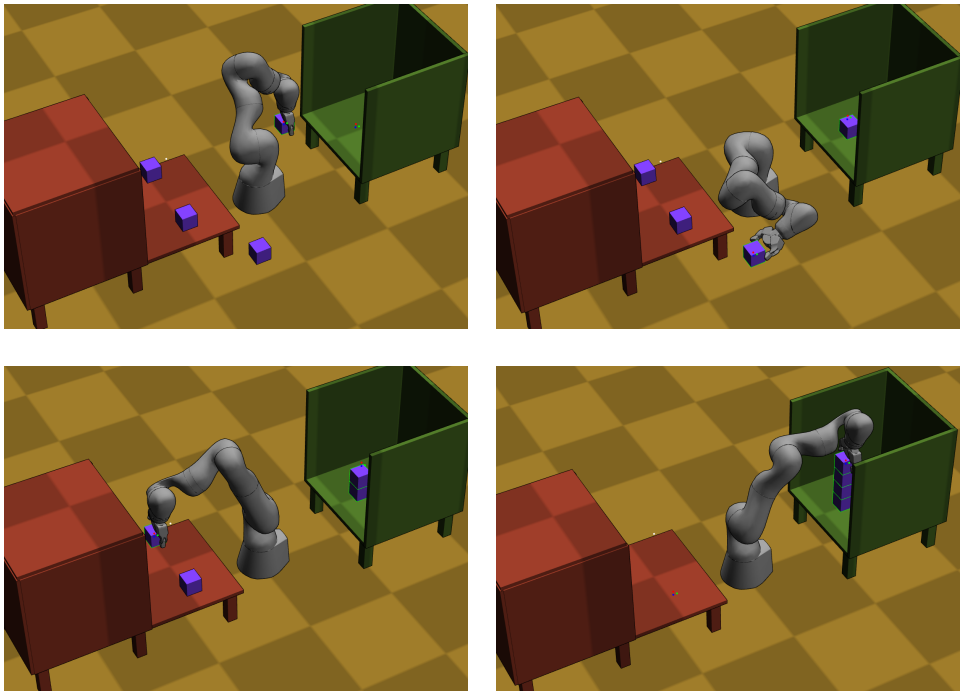


Figure 5.13: Building structure with dataset approach

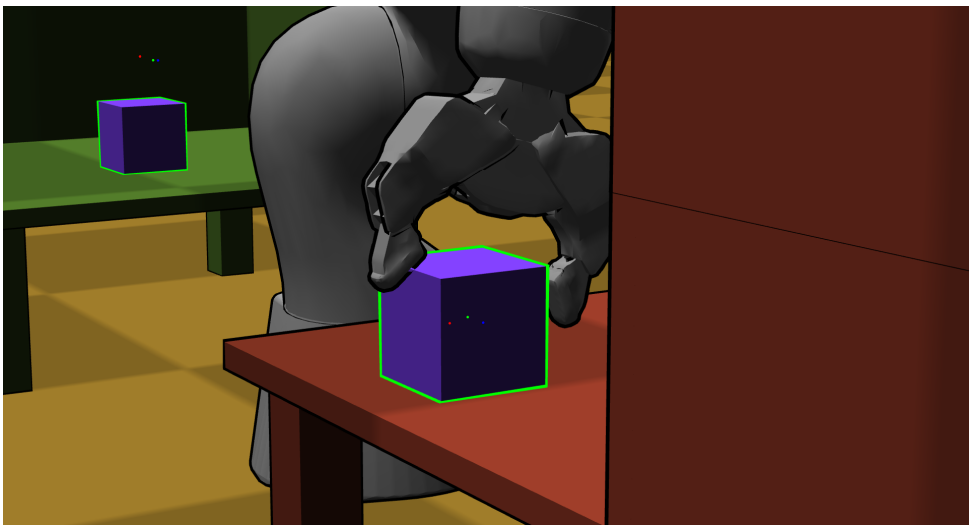
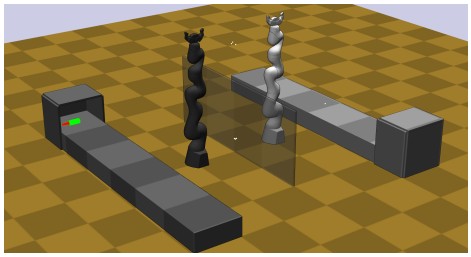


Figure 5.14: A low quality grasp found by the dataset approach

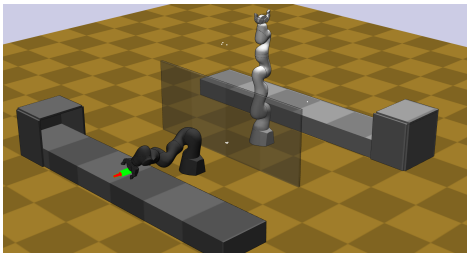
5.5 Application: Industry line

In this task, a simplified version of an industrial line is visualized. The experiment aims to present the possibility of multi-robot cooperation within the application. Specifically, the path is built for the object to transport it from one industrial line to another using two KUKA LBR iiwa robots.

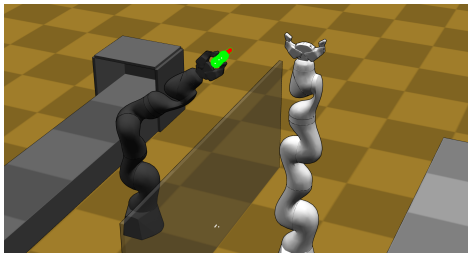
Automatically computed grasps and paths are constructed by the analytical algorithm and `Lazy PRM*` motion planner accordingly. The main milestones of the experiment are captured in the Figure 5.15



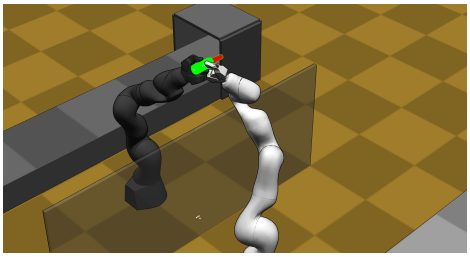
(a) : The object is transported to the arm



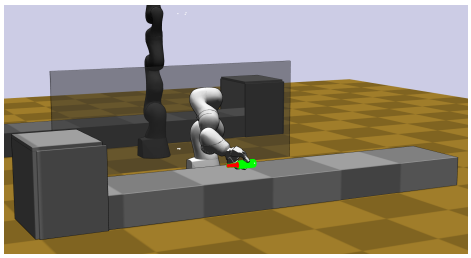
(b) : The first grasp is performed



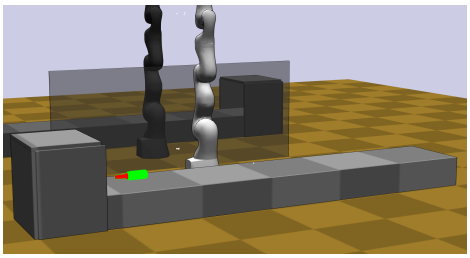
(c) : The object is transferred to the common point



(d) : Second collisionless grasp configuration is found



(e) : The object is placed at the second industrial line



(f) : The object continues its movement as the pipeline continues

Figure 5.15: The main milestones of the visualization of industrial line application

Chapter 6

Analysis and discussion

This chapter analyzes the results achieved in the experiments. The algorithms' behavior is interpreted in terms of advantages and downsides. Based on that, there can be made an assumption on which is the most suitable situation for each approach. It is also necessary to underline the limitations of the project to preserve its efficient performance in practical applications. Later in the chapter, a few potential improvements are discussed.

6.1 Experiment analysis

To investigate the outcome of the computational time experiment conducted in Section 5.2 each individual table should be examined. The algorithms performed differently and show diverse tendencies with the increasing complexity of a task. Firstly, the measurements are analyzed separately for each algorithm and, secondly, compared with each other.

6.1.1 Heap generation results

As displayed Tables 5.2, 5.4, 5.6, the program is divided into two parts: heap creation and grasp searching. The initialization of a heap shows a strong trend towards increasing time associated with a growing number of polygons. However, this is not the only parameter affecting the computational time. Operating with various shapes leads to a different number of groups of faces that meet the set conditions. In turn, the more groups of faces are suitable, the slower the heap is constructed. Table 6.1 displays the number of suitable grasp groups found during each heap generation:

The numbers may vary depending on the initial mesh shape and the level of decimation or subdivision applied. For example, the bottle is introduced as an extreme case of mesh preprocessing. The initial structure of the mesh is quite heterogeneous as most of the polygons are located near the cap. Following reduction of some of them may affect undesirable areas resulting in loss of potential grasp candidates. Consequently, the heap length is notably smaller. Despite the successfully found grasps within experiments, the risk of grasp search failure is higher.

Object	Heap length
Cube	192
Flashlight	9092
Bulb	1366
Bottle	26
Cylinder	768
Sphere	3420

Table 6.1: Heap length for each object

On the other hand, generating too many potential grasp groups may result in a bad computational complexity of the second part of the program, the task itself. Heap prevents this from happening by sorting the array based on the distance between two faces so that the parallel polygons lying on the same line have a higher priority.

A complicated mesh structure of an object is an important problem, which should be respected and solved when uploading a new object into a simulation.

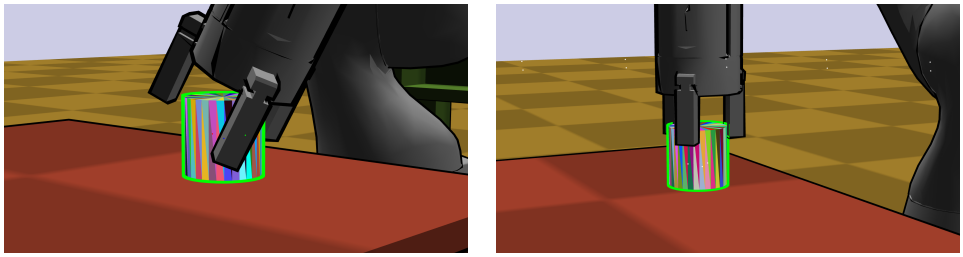
6.1.2 Dataset generation

The performance of the empirical algorithm utterly depends on the quality of the predefined dataset. While creating it, the position of an object in the world should be neither too distant nor too close. The object can be searched for grasps from several orientations (in Section 5.1 only two were used). Therefore, the time needed to create a new dataset can differ from the results displayed in Table 5.1.

6.1.3 Behavior comparison

Both algorithms were tested on the numerical inconsistency of the simulation. The numbers enlisted in the tables prove that even performing the task with the same input in the same simulation world results in unique outcomes. In some cases, it can lead to a result similar to what was witnessed in the grasping the bottle in the average complexity scenario. The more prioritized feasible grasp failed the grasp planning and the algorithm continued searching for another one. Generally, the computational time difference is not critical, as in both simple and average complexity scenarios the algorithms perform similarly. Nevertheless, the variety between the best and the worst computational time in the dataset approach is slightly lower, as opposed to the analytical algorithm. The reason for this phenomenon is that fewer mathematical operations are needed to plan a grasp, leaving less space for numerical inconsistencies. Moreover, more complicated situations are solved much faster. It is connected with the fact that all of them are already suitable and sorted based on the distance from the other objects. Thus, the only reason for grasp failure is an environmental constraint. The time is not wasted on trying infeasible by default grasps. Assuming a whole and carefully created dataset, the

computational results are more constant than the ones achieved by the analytical algorithm.



(a) : Bad quality grasp executed by empirical algorithm

(b) : The grasp performed by the analytical algorithm is stable

Figure 6.1: Comparison of two grasps found by different algorithms

The disadvantage of the dataset approach is grasp quality. In non-physical visualization, the outcome of the grasp cannot lead to failing force closure, because the mass of an object and gripper force are not taken into account. Consequently, some of the grasped in the dataset may have low quality. One of those grasps was shown in the application described in Section 5.4. For a slightly more expensive computational cost, the analytical approach plans grasp at positions fulfilling the set conditions, therefore, providing their higher quality. Compare the two situations in Figure 6.1. In such a case, it is evident that the grasp from the dataset will lead to an undesirable outcome.

The analytical approach benefits from the lack of preliminary preparation. There is no need to compute anything in advance. This feature makes this algorithm far more flexible than the empirical one. Grasp quality and flexibility of the analytical algorithm make it a universal tool in numerous applications, e.g., bin-picking, structure assembling, and common object manipulation. Unless the empirical method is supported with some physical filtering criteria, it is only applicable in some particular cases to quicken the task execution.

6.2 Fragile area detection analysis and application

Finally, one of the most potent tools presented in the thesis is the method allowing to forbid any area of the object for grasp planning. The experiments 5.3 and 5.5 show the user-defined area is detected and correctly interpreted. Considering the method from another point of view, colorizing the object can also be used to highlight specific places for grasp planning with a special color in applications, where the precision of a grasp is a priority.

6.3 Possible improvements

To achieve lower computational time, both algorithms could be run in parallel on more computer cores. Such parallelisation is possible because individual computations are independent. Generally, the hardware setup can also be upgraded.

For the empirical approach, the best improvement would be a generation of a dataset considering the physical aspects of a simulation. This way, the unstable grasps will be filtered out.

Bibliography

[1] FAN, Yongxiang, Masayoshi TOMIZUKA. Efficient Grasp Planning and Execution With Multifingered Hands by Surface Fitting. IEEE Robotics and Automation Letters. 2019, 4(4), 3995-4002. ISSN 2377-3766. Available from: doi:10.1109/LRA.2019.2928210

[2] AYDIN, Yahya, Masayuki NAKAJIMA. Database guided computer animation of human grasping using forward and inverse kinematics. 1999, 23(1), 145-154. ISSN 00978493. Available from: doi:10.1016/S0097-8493(98)00122-8

[3] LI, Ying, Jiaxin L. FU a Nancy S. POLLARD. Data-Driven Grasp Synthesis Using Shape Matching and Task-Based Pruning. IEEE Transactions on Visualization and Computer Graphics. 2007, 13(4), 732-747. ISSN 1077-2626. Available from: doi:10.1109/TVCG.2007.1033

[4] IBERALL, Thea. Human Prehension and Dexterous Robot Hands. The International Journal of Robotics Research. 1997, 16(3), 285-299. ISSN 0278-3649. Available from: doi:10.1177/027836499701600302

[5] Presence: Teleoperators and Virtual Environments. 5. 1996. ISSN 1054-7460. Available from: https://direct.mit.edu/pvar/article/5/4/416-430/92620

[6] ZHAO, Wenping, Jianjie ZHANG, Jianyuan MIN and Jinxiang CHAI. Robust realtime physics-based motion control for human grasping. ACM Transactions on Graphics. 2013, 32(6), 1-12. ISSN 0730-0301. Available from: doi:10.1145/2508363.2508412

[7] BIERBAUM, Alexander, Matthias RAMBOW, Tamim ASFOUR and Rudiger DILLMANN. Grasp affordances from multi-fingered tactile exploration using dynamic potential fields. 2009 9th IEEE-RAS International Conference on Humanoid Robots. IEEE, 2009, 2009, , 168-174. ISBN 978-1-4244-4597-4. Available from: doi:10.1109/ICHR.2009.5379581

[8] SCHLESINGER, G, R.R DUBOIS, R RADIKE and S VOLK. Der mechanische Aufbau der künstlichen Glieder (the mechanical building

of artificial limbs). Ersatzglieder und Arbeitshilfen für Kriegsbeschädigte und Unfallverletzte. Julius Springer-Verlag, 1919.

[9] HAUSER, Kris. Klamp't. Intelligent Motion Laboratory [online]. [cit. 2021-04-04]. Available from: http://motion.cs.illinois.edu/klampt/

[10] HAUSER, Kris. Fast Interpolation and Time-Optimization on Implicit Contact Submanifolds. Robotics: Science and Systems IX. Robotics: Science and Systems Foundation, 2013, 2013-06-23. ISBN 9789810739379. Available from: doi:10.15607/RSS.2013.IX.022

[11] HAUSER, Kris. Klamp't Manual [online]. [cit. 2021-04-04]. Available from: http://motion.cs.illinois.edu/software/klampt/latest/pyklampt_docs/

[12] LBR iiwa [online]. [cit. 2021-04-04]. Available from: https://www.kuka.com/en-at/products/robotics-systems/industrial-robots/lbr-iiwa

[13] Trimesh Python API [online]. [cit. 2021-04-04]. Available from: https://trimsh.org/trimesh.html

[14] SÁNCHEZ, Gildardo and Jean-Claude LATOMBE. A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking. Robotics Research. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, 2003-6-30, 403-417. Springer Tracts in Advanced Robotics. ISBN 978-3-540-00550-6. Available from: doi:10.1007/3-540-36460-9_27

[15] HAUSER, Kris. Lazy collision checking in asymptotically-optimal motion planning. 2015 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2015, 2951-2957. ISBN 978-1-4799-6923-4. Available from: doi:10.1109/ICRA.2015.7139603

[16] KARAMAN, Sertac and Emilio FRAZZOLI. Sampling-based algorithms for optimal motion planning. The International Journal of Robotics Research. 2011, 30(7), 846-894. ISSN 0278-3649. Available from: doi:10.1177/0278364911406761

[17] MILLER, A.T. and P.K. ALLEN. GraspIt!. 2004, 11(4), 110-122. ISSN 1070-9932. Available from: doi:10.1109/MRA.2004.1371616

[18] LEÓN, Beatriz, Stefan ULBRICH, Rosen DIANKOV, et al. OpenGRASP: A Toolkit for Robot Grasping Simulation. Simulation, Modeling, and Programming for Autonomous Robots. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, 2010, , 109-120. Lecture Notes in Computer Science. ISBN 978-3-642-17318-9. Available from: doi:10.1007/978-3-642-17319-6_13

[19] LEÓN, Beatriz, Antonio MORALES and Joaquin SANCHO-BRU. Robot Grasping Simulation. From Robot to Human Grasping Simulation. Cham: Springer International Publishing, 2014, 2014-9-29, 33-65.

Cognitive Systems Monographs. ISBN 978-3-319-01832-4. Available from: doi:10.1007/978-3-319-01833-1_3

[20] BONILLA, Manuel, Cosimo Della SANTINA, Alessio ROCCHI, et al. Advanced Grasping with the Pisa/IIT SoftHand. Robotic Grasping and Manipulation. Cham: Springer International Publishing, 2018, 2018-07-15, 19-38. Communications in Computer and Information Science. ISBN 978-3-319-94567-5. Available from: doi:10.1007/978-3-319-94568-2_2

[21] HAUSER, Kris. Robust Contact Generation for Robot Simulation with Unstructured Meshes. Robotics Research. Cham: Springer International Publishing, 2016, 2016-04-23, , 357-373. Springer Tracts in Advanced Robotics. ISBN 978-3-319-28870-3. Available from: doi:10.1007/978-3-319-28872-7_21

[22] LIANG, Beatrice. Robot Learning in Simulation for Grasping and Manipulation. Columbia University, Department of Computer Science.

[23] PRZYBYLSKI, Markus, Tamim ASFOUR and Rudiger DILLMANN. Unions of balls for shape approximation in robot grasping. 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2010, 1592-1599. ISBN 978-1-4244-6674-0. Available from: doi:10.1109/IROS.2010.5653520

[24] Graphical User Interfaces with Tk [online]. Python Software Foundation, 2021 [cit. 2021-04-12]. Available from: https://docs.python.org/3/library/tkinter.html

[25] Three-point gripper DHDS. FESTO, 2020/02. Available from: https://www.festo.com/cat/en-gb_gb/data/doc_engb/PDF/EN/DHDS_EN.PDF

[26] 3-Finger Adaptive Robot Gripper Instruction Manual [online]. Robotiq, 2020 [cit. 2021-04-12]. Available from: https://assets.robotiq.com/website-assets/support_documents/document/3-Finger_PDF_20201208.pdf?_ga=2.122470219.1354482457.1618261049-1070373188.1618261049

[27] PyVista [online]. [cit. 2021-04-18]. Available from: https://docs.pyvista.org/

[28] Blender [online]. [cit. 2021-04-19]. Available from: https://www.blender.org/

Appendix A

User Guides

A.1 Creating a new world

The world for an application can be easily created based on the pattern of the existing words located in the corresponding folder at the root of the project. The main terrain can be easily uploaded from the terrain file in the `objects` folder.

```
<terrain file="objects/terrains/plane.off">
        <display color="0.3 0.3 0.2 0.5"/>
</terrain>
```

The terrains are constructed from primitive blocks, so that adding a new one would be:

```
<terrain file="objects/terrains/cube.off"
         scale="0.4 0.02 0.15"
         translation="0.565 0.145 0.2">
    <display color="0.3 0.3 0.2 0.5"/>
</terrain>
```

Adding a robot to the world with the specified color, position, and orientation in the world coordinate system:

```
<robot name="kuka" file="robots/kuka/kuka.rob"
       translation="0 0 0.01">
</robot>
```

Rigid objects can also be specified in the world file. However, to correctly upload them into the world, running one of the algorithms is advised.

A.2 Using the dataset algorithm

- In the folder project, run the script named `dataset_generation.py`. The dataset is going to be stored in the grasping_database folder.
- Choose the object to create the dataset from. The program will start to compute grasps and store the successful ones into an array. At the end of the program, the array is written into the specified directory.
- The offline part is finished, so now the main application can be started by running `online_task.py`. Choose the world file prepared for the application or one of the default ones.
- Run the script and select the objects in the Tkinter window. Each uploaded object must have its dataset in the `grasping_database` folder.

A.3 Using the analytical algorithm

The application supported by the analytical algorithm can be visualized by:

- Run the script named ***"analytical_algorithm"*** with the way to the world specified in the main function.
- In the Tkinter window, choose the number and type of objects.
- Select the object and use a keyboard[1] to specify the final position for the transfer. Everything, including the transit, transfer and retract paths, will be constructed automatically.

[1] To choose the object, press the number of the object, e.g., press *"1"* to choose the first uploaded one. Press one of the letters: *"a"*, *"b"*, *"c"*, *"d"* to select the goal.

A.4 Uploading and colorizing a new object

The guide assumes that the input for this part is a 3D object with `.obj` extension. Blender **version 2.83** used in the thesis is installed in advance.

A.4.1 Blender: material assignment

To define a fragile area for a new object, a few steps should be completed:

- Import the object represented by an `OBJ` file.
- Add new material and change the base color of it into red (RGB settings are advised)
- Select the faces that must be marked, and assign the newly created material to them.
- Add the *Triangulate* modifier to make sure that all polygons have a triangle shape.
- Export the object as an `OBJ` file into the corresponding folder.

A.4.2 Blender: face colors and vertex array creation

Now that the color materials are attached to the faces, they have to be correctly imported into the simulation. Therefore, the colors, corresponding faces, and vertices are saved and stored in the `NumPy .npy` extension file in the object folder. This process is implemented in the utility script `blender_colors.py`. It is located in the `utilities` project folder. All the arrays must be saved into the folder inside the object directory. There are two ways the script can be executed:

- Using the `bpy` library, the Python API for Blender, installed in the current virtual environment
- Using the Python editor inside the Blender.

Note that the Python **version 3.6.0** of the project's virtual environment was not compatible with the API at the time of writing this thesis. Consequently, the Blender built-in editor was used.