Analogy For Robot Object Manipulation

Marcel Zak

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Master of Engineering
of the
University of Aberdeen.

Department of Computing Science

03-May 2019

Declaration

No portion of the work contained in this document has been submitted in support of an application for a degree or qualification of this or any other university or other institution of learning. All verbatim extracts have been distinguished by quotation marks, and all sources of information have been specifically acknowledged.

Signed:

Date: 03-May 2019

Abstract

This dissertation reports the development, testing, and evaluation of “Analogy", a research tool that serves for fast and easy testing of new ideas in robot object manipulation using an analogy. In comparison to the previous system, the required time to draw and test an analogy in a new scene was reduced by 10 times. Based on the evaluation the new analogy mapping and scoring algorithm seems to perform well in a variety of different scenarios. This tool is open sourced and well documented for future contributors.

Acknowledgements

I would like to express my sincere thanks and gratitude to all those who helped me with this project. First of all, I would like to acknowledge the unfailing support, expertise and enthusiasm of my supervisor, Dr. Frank Guerin. Then I would like to thank for all the support and care of my lovely wife Andrea Zak.

Contents

Chapter 1

Introduction

People are very good at manipulating and interacting with physical objects. Even young children are capable of manipulating and using everyday objects better than any currently available robot. With the recent increased demand in robotics, robots are expected to be deployed in many different situations and environments. For example, a robot called Pepper[1] is designed to be used in education, health care and it should be able to handle everyday tasks too. Robots may also help in countries such as Japan where the elderly population grow as younger population decline. This introduces an enormous challenge for the ability to adapt existing knowledge and skills in new scenarios and environments.

1.1 Motivation

Nowadays industrial robots are usually not exposed to the open environment [1]. However, this trend quickly changes with a new demand to deploy robots in many different scenarios. I and many other people have been waiting for robotic helpers that would take care of everyday tasks in a household. These everyday tasks that are considered to be easy for any human are notoriously difficult for a robot. There is not a single robot that could handle these standard household tasks. The main reason is that robots do not have a common seance or, in other words, general knowledge. The next problem is that skill adaptation is almost non-existent in robotics. It turns out that it is very difficult and researchers do not know how to do it yet. This will have to change if we want to have a future where robots can handle more complex tasks and do not require human guidance in an open unpredictable environment.

The important thing here is that a human can perceive an object differently based on the task that wants to execute. This is a very abstract explanation and the best way to understand this is based on a specific example.

If we look at Figure 1.1 we can see a regular spatula that is used in a kitchen. However, the way we perceive this tool depends on the task that has to be done.

1. Our task is to place a picture hook on a wall but we do not have a hammer. We can use the handle of the spatula to execute this task.

2. Our task is to cut a piece of cake but we do not have a knife. We can rotate the spatula by 90 degrees and use the sharp, thin edge of the spatula to cut the cake.

3. Our task is to flip a pancake. We can do this using the spatula as originally intended.

[1] https://www.softbank.jp/en/robot/

Figure 1.1: Spatula.

4. Our task is to spread butter on bread but we do not have a knife. We can use the edge of the spatula to do this.

5. Our task is to reach a ring that is under the sofa. We can use the spatula as an arm extension to do this task.

These are only a few examples of different use cases where the spatula can be used even though it was not originally intended to do so. This implies that depending on the task we want to do, it also depends on how we are going to handle the spatula. The underlying task has a direct influence on the ideal manipulation points and force vectors applied during the manipulation.

Another specific example is shown in Figure 1.2. There are 3 books on a shelf and we would like to manipulate the middle book such that it exposes the left and right sides and we can grab it. In this situation the ideal manipulation point to do this operation is shown in Figure 1.2 as a purple sphere with an arrow that shows the ideal force vector to be applied to this point.

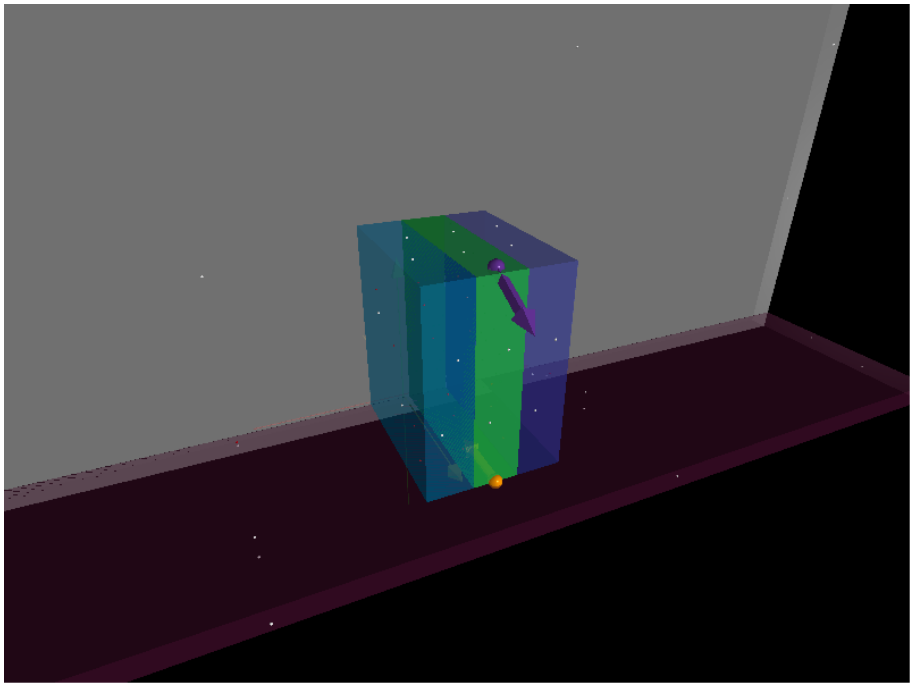

Figure 1.2: 3 books on a shelf. The middle one is the one we want to manipulate.

Having this knowledge we can use an analogy to solve a previously unseen situation that is

shown in Figure 1.3. There are 3 pizza boxes in a freezer and we would like to pick up the middle one. It is not immediately obvious that this scene is similar to the scene with books on the shelf in Figure 1.2. However, if we rotate the whole scene with pizza boxes by 180 ° clockwise. we can see some emerging similarities with the previous scene with books.

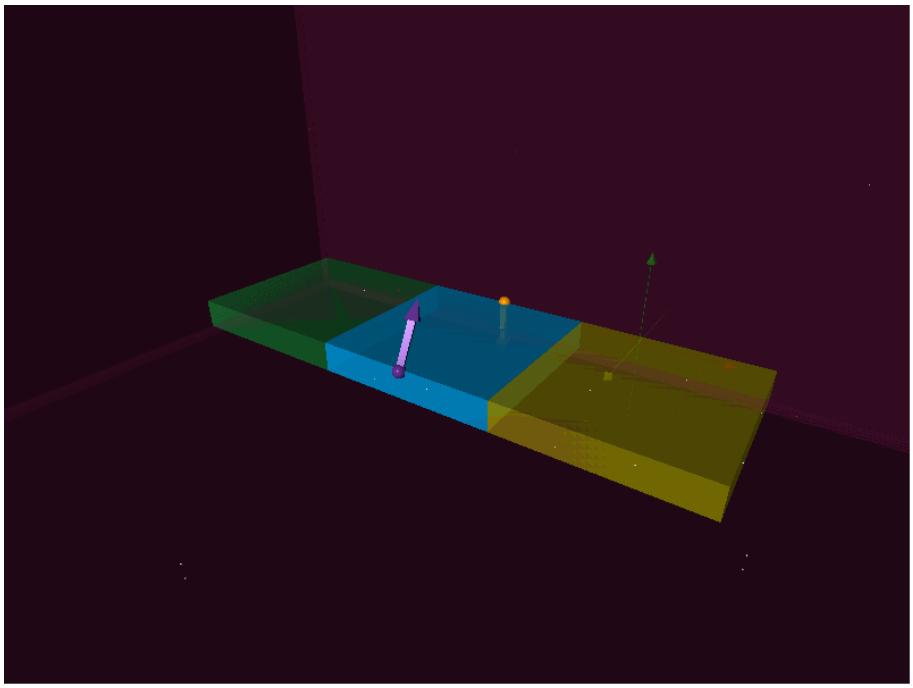

Figure 1.3: 3 pizza boxes in a freezer.

Using an analogy we can say that the scene with pizza boxes is similar to the one with books. Moreover, we can reuse the existing knowledge about the ideal manipulation point and force vector. We already know how to execute a pull operation the middle book from the source scene (3 books on the shelf) and we can apply this knowledge in the target scene (3 pizza boxes in the freezer). The result of such an analogy is shown in Figure 1.4.

This example clearly shows that depending on the task we want to execute it also depends on how we are going to interpret the surfaces of an object. In the scene with 3 books, we applied the manipulation point and its force vector to the top surface. But in the scene with 3 pizza boxes, we wanted to interpret the front surface of the pizza box as if it was the top surface.

This dissertation reports the development, testing and evaluation of “Analogy". It is a framework for researchers that want to explore the possibility of using an analogy for skills and knowledge transfer between different scenarios. Specifically, it focuses on how to transfer existing knowledge about ideal manipulation point and its force vector from a known scenario (source) to an unknown scenario (target). Currently, it supports two basic operations for push and pull of the target object. In addition, there is also one task using a spatula tool to demonstrate that it can be easily extended for other manipulation operations in the future.

1.2 Objectives

Given the current issues with knowledge and skills transfer (adaptation) presented above, the objectives of the project are:

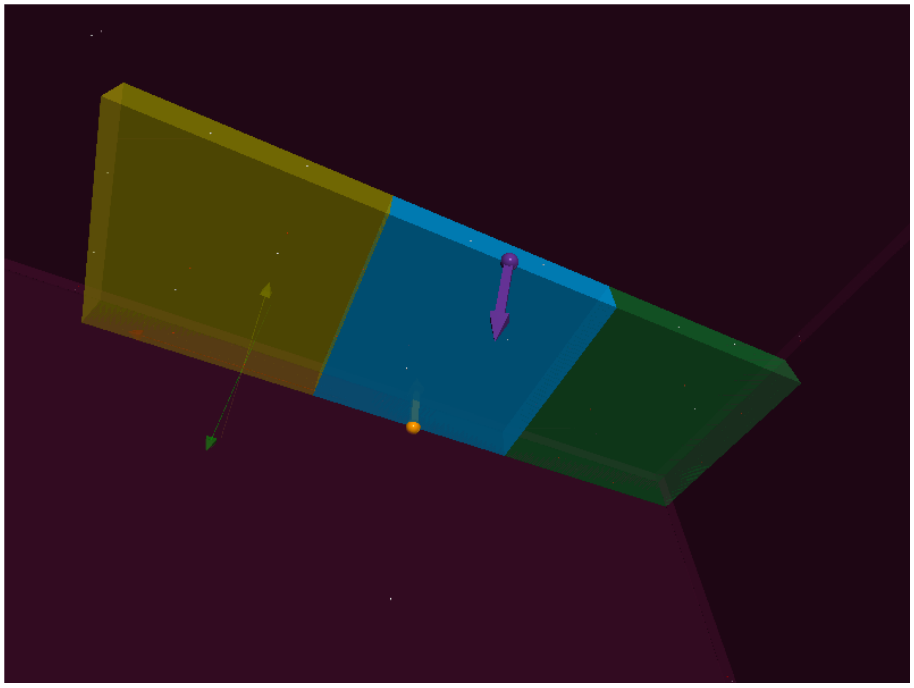

Figure 1.4: 3 pizza boxes in a freezer wit rotated view by 180 degree clockwise.

1. Development of a research framework that allows quick and easy testing of new ideas in robot object manipulation using an analogy.

2. Development of the core analogy algorithm that attempts to provide reasonable performance in knowledge adaptation between scenarios.

3. Testing and evaluation of the developed solution. The third goal is to test and evaluate the developed solution in terms of how well it adapts the existing knowledge to a new unseen situation.

4. Modularity and scalability to simplify future work. The system should be modular and possible to extend/reuse in other projects.

Chapter 2

Background & Related Work

This chapter is dedicated to background and work related to this project.

2.1 Robotics

Robotics is a place where pure theoretical computing science meets with the difficulties of a physical world. It is a notoriously difficult interdisciplinary area of research where small improvements often take several years. The word **robot** was first used in a science fiction play Rossum's Universal Robots written by Karel Capek. He was a Czech writer and in the Czech language, the word "robota" means labour [2]. In the mentioned play robots were machines that could work continuously without rest. This was a dream for many people for a long time.

Currently, we have robots that are relatively cheap, fast, efficient and very precise. They are replacing human workers in many places. The most common places where you can find robots are jobs that require repeating tasks, precise predictable movements, dangerous environment or poses other threads for human workers. All these places have one common thing. They are usually enclosed in well-controlled environments. This is because in robotics usually the hard things are easy and the easy things are hard. What I mean by that is that sub-millimetre accuracy of a movement is achieved very easily for a robot but pouring water into a cup is much more difficult. This is a general rule that can be applied in all computing science.

If we think about it, we can clearly see that robots have to work only in a well defined enclosed environment that is predictable. Majority of industrial robots are unable to cope with a dynamically changing environment. The problem of executing everyday manipulation tasks is for now out of the reach of current artificial intelligent robots. The main reason is that the robot has to be capable of coping with unseen situations, improvising, learning and reacting quickly. All these everyday manipulation tasks in an open environment have long tail distribution [1].

In order to overcome the long tail problem, the robot would have to be able to adapt its existing knowledge and experiences to unseen situations. To help to solve this issue, robots could use an analogy to apply the existing knowledge in new situations. But what exactly is an analogy?

2.2 Analogy

The Oxford dictionary says that analogy is "A comparison between one thing and another, typically for the purpose of explanation or clarification." [3]. This meaning is usually applied to language. For example, a sentence "Analogy is the motor of the car of thought.". So, we can write "analogy:thinking::motor:car" that we can read as "analogy is to thinking as a motor is to a car".

In more general notation we can write that an analogy is “A:B::C:D". This reduces the analogy to the standard view of proportional analogy that is known from IQ tests. A shoe is to a foot as a glove is to a hand etc. [4]. However, there are also other types of analogy. “A correspondence or partial similarity." is a more general and broader definition of analogy [3]. In this work, I am going to focus on analogy in robot object manipulation. To be more specific, I am going to focus on how can a computer (robot) store knowledge about ideal manipulation point with an associated force vector for a specific manipulation operation and then apply this knowledge using an analogy to a new unseen situation.

In the analogy research area, there are two main competing theories that propose an explanation of this phenomenon. The first one is Dedre Gentner’s Structure-Mapping Theory. The second one is Douglas Hofstadter’s High-level Perception theory.

Structure-Mapping Theory [5] describes analogy as a result of *structure-mapping*. This theory has the assumption that psychological concepts have structures. These should be the psychological representations of relations between perceptual and conceptual objects. Based on Structure-Mapping Theory an analogy (the ability to recognise that “one thing is like another") is a mapping and mapping is the core defining process of analogy. In other words, mapping takes two structures that are called *source* and *target* and computes one or more *mappings*. Then each mapping contains a set of *correspondences* between the *source* and *target* structures. Usually, a mapping also contains a numerical score that indicates its structural quality [6]. Gentner and others decided to create a computer model of this theory and created the Structure-Mapping Engine (SME) [7]. It turns out that it works very well when given a carefully constructed representation (structure). However, this approach is viewed as “horizontal" view of analogy. It means that it is capable of finding an analogy in already existing representation.

On the other hand, Douglas Hofstadter and his theory of High-Level Perception take a different approach to the explanation of analogy. The theory says that an analogy is a product of a general cognitive function called *high-level perception*. This is a process which constructs an underlying representation of a situation based on low-level perceptual processing. It means that a high-level representation of a situation is affected by these low-level processes. This theory may be easier explained by a simple example. Imagine a wine bottle. This may be perceived as a bottle, a weapon, a roller for dough or even as a candle stand. The way we perceive the wine bottle depends on the situation we are currently facing. Hofstadter goes even further and proposes a hypothesis that analogy is the core for cognition “The Cognition-core hypothesis" [4]. The Copycat Project designed by Douglas R. Hofstadter and Melanie Mitchell [8] is a model of analogy as high-level perception. It is a method that purpose is to determine insightful analogies. However, it works only in a simplified letter domain.

In the paper [9] Clayton T. Morrison and Eric Dietrich argue that the competition of both of the above-mentioned theories is ill-founded. They think that these theories explain two different aspects of analogy. Structure-Mapping Theory searches a horizontal view of analogy and High-Level Perception searches a vertical view of analogy. They propose that they should be unified in a single theory.

For the purpose of this project, I believe that Hofstadter’s theory is more suitable. Imagine a situation where we have a source object that we know how to manipulate and then we have a target

object that we do not know how to manipulate. Now, the source object has surface x as a top and the target object has a surface y as a top. However, we may want to view the target differently and treat one of the other surfaces as being like the surface x in the source. Which one we want to use we do not in advance. There is a search process involved that can change how we “see" the target. For example, we can “see" it sideways. This is a change of perspective. We would like to find a perspective that the designer did not decide in advance. I believe that the software should do the work instead of a human. In Gentner’s theory, the human sees the analogy in advance and codes it in predicates that gives the perspective the creator wants. Therefore, the analogy is obvious. There is not much for the computer to do. This approach is like GOFAI (“Good Old-Fashioned Artificial Intelligence"). It turns out that modern approaches beat GOFAI and that is the reason why the latest breakthroughs and innovations are not in GOFAI [10]. I would like to exploit the available computational power to explore and find the best analogy for a specific scenario.

2.3 Generalisation And Adaptation Of Knowledge In Robotics

A generalisation of knowledge is a very important part for robots that should work in open unpredictable environment. For example, we would not want to have a robot that is incapable to recognise and grasp a new cup that has cuboid shape instead of cylindrical. A human can immediately recognise and adapt existing knowledge in such a scenario. On the other hand, for a robot, it is a very difficult task.

In the paper, [11] authors proposed and evaluated a system that generalises knowledge about a known object via wrapping point clouds to the unknown target object. The results show that this approach is capable of recognising the key features of the target object. For example, it recognises what part of the object is a container, a rim and a handle. With the updated knowledge about the previously unknown object, the robot adapts motor control and trajectories adequately. It is an interesting approach to the generalisation of knowledge. However, I believe that this approach is computationally too expensive to be used in the real world for this moment. This does not mean that it won’t be a good solution in the future where we have more computational power thanks to Moore’s law[1].

In the paper [12] authors present an idea that transfer of analogy is happening in everyday manipulation tasks. However, when a source and a target scenario is too different the transfer of knowledge is difficult. Creativity and human guidance are necessary for a successful transfer of knowledge.

In the paper [13] authors have the same opinion about transferring existing skills as I advocated previously. They believe that the mapping is dependent on the task being performed and the roles the object plays in the task. This is similar to the example of perceiving a wine bottle as a candle stand as I mentioned previously. In addition, this paper presents results that human guidance can significantly improve the transfer of skill between two scenarios.

In the paper [14] authors explored the possibility of visual case retrieval for case-based interpretation of skill demonstrations. They tried to use SIFT feature-matching, SIFT feature-transformation and Fractal feature transformation. The results show that no single method works best for all case-based skill interpretation problems.

[1] https://en.wikipedia.org/wiki/Moore%27s_law

In the paper [15] authors faced problems with mobile manipulation. They decided to solve the problem by decomposing the manipulation problem into a symbolic and geometric part. Then these problems can be solved with a planer. This paper inspired me with the idea that the future system could split a complex manipulation task to smaller atomic manipulation tasks (push, pull, etc.) and then solve the problem using a planer.

I also tried to explore one of my ideas of adapting existing knowledge to a new situation with the help of looking at how different is the source and the target scenario. For example, rotating the target object that is represented as a mesh and trying to fit it to the shape of the source object. For this, I would need some measure to evaluate the mesh difference. Therefore, I read the paper [16] that proposes a new attribute deviation metric. The considered meshes contain geometrical and appearance attributes. These are for example material colour, texture, temperature, etc.

Then I also considered a different option on how to evaluate a mesh difference. In the paper [17] authors present a method that can quickly evaluate two meshes and return the numerical score that represents the approximation error.

The main reasons why I decided not to explore these above-mentioned methods further and apply them in this project are following. The methods are designed to compare two meshes after some mesh simplification and return the error value. In my case, the meshes are not going to be aligned and these methods would not work. In order to align two meshes I considered using principal component analysis [2]. Another option I considered was using Hausdorff distance[3]. However, I realised that this may be a work for a whole new project and it was out of the scope of this project. Even though it is worth mentioning for potential future improvements.

2.4 Collision Detection

Collision detection is something that is very easy for a human but very difficult for a computer. Before I started this project I did not have any prior knowledge about this problematic. Since I wanted to implement automatic detection of available surfaces to the system, I decided to investigate different approaches to do so.

As the first thing that I imagined was that the system could use a triangle to triangle collision detection for meshes in a scene. I explored this option and found the paper [18] from Oren Tropp. Also, I tried to read other well-known algorithms that can provide a solution to this problem. Be it [19] and [20].

While reading above mentioned papers I started to investigate what is the standard approach in game development for collision detection. It turns out that collisions are usually detected with a different approach. Because mesh to mesh collision detection is computationally expensive and not feasible. For example, in real time First-Person Shooter (FPS) games, game developers use different techniques. I started with learning how I could detect a collision in 2D environment [21]. Then I found this [22] well explained article by Mozilla for 3D collision detection. In the end, I decided to spend some time and really understand this problematic. Therefore, I read the book Mathematics and Physics for Programmers (GAME DEVELOPMENT SERIES) [23]. Specifically, Chapter 10 provides an excellent explanation of different techniques used for detecting

[2] https://en.wikipedia.org/wiki/Principal_component_analysis
[3] https://en.wikipedia.org/wiki/Hausdorff_distance

collisions between complex shapes. This provided me with solid knowledge.

I will try to explain the whole process of collision detection in simple words. Objects in a 3D scene are meshes that consist of triangles that consist of vertices with 3D space coordinates and defined edges between vertices. The computation process of a triangle to triangle collision detection is too expensive. Therefore, we usually approximate the shape of the object and wrap it to the axis-aligned bounding box (AABB). As the name suggests the box is axis-aligned. Also, it is a box with a minimal size where the object can fit. Every object in a scene has this AABB. Hence, we can quickly check if two AABBs overlap (collide). This way we can filter out all non-colliding objects in the scene. Now, we can focus only on the remaining that potentially collide. In addition to AABB, each object has associated collider points. Since we determined that two AABBs collided, now we have to check all collider points if they are inside another AABB. If that is true we know that the object has a collision at the specific point in the space where the collider is. It is important to keep in mind that this is not the only possible solution. There are different techniques and approaches. One example could be using Object-Aligned Bounding Box (OABB) instead of AABB. These different approaches bring some advantages and disadvantages that the developer has to consider in advance.

2.5 Existing Systems

The predecessor of this project was a system developed as a final year project by MSc student in Artificial Intelligence Stewart Duncan [24]. The project was an attempt to adapt the Copycat method [8] to object manipulation. The result was that the system was capable of using an analogy for suggesting a manipulation point to a target object in unseen situations. Originally I wanted to continue with the development using this existing system. However, after reading the source code, I realised that using the existing code would be a mistake. There were several issues that I found.

First of all, the code has poor quality, it is not reusable and it does not adopt any programming style. Then, all scenes had to be modelled manually as a python class. This means that the user supposed to specify each 3D space coordinate for each vertex then define edges between vertices. In the end, the user had to spend at least 45 minutes to create a single scene. This was unacceptable for the new system because I wanted to be able to test and prototype new ideas very quickly. In the end, the complexity of the mapping and scoring algorithm seemed to be unnecessary and I believed that I can improve, simplify and decrease the complexity of this algorithm. Therefore, I decided not to use thy system for my project.

Another similar project to this one was a final year project of BSc student in Computing Science Robin Graf [25]. This project focuses on knowledge adaptation using constraint-based movement in simulations for grabbing a book. The author experienced a lot of problems and drawbacks. For example, long run-time of simulations, an adaptation of skills did not work well in different scenes, a lot of problems with physics simulator and a very low success rate. Because I wanted to focus on adapting an existing knowledge using an analogy, I saw that using ROS[4] and Gazebo[5] or any other physics simulator would slow me down dramatically.

The last and most sophisticated system in comparison to previous systems was developed by

[4] https://wiki.ros.org/
[5] https://wiki.ros.org/gazebo

a PhD student Abelha and my supervisor Dr Guerin [26]. This project focuses on the transfer of tool affordance and manipulation cues with 3D vision data. What the system does is the following. Given a point cloud of a specific tool, it tries to fit super quadratics to the point cloud. Then based on the shape it can estimate which part of the tool is most suitable for a specific task. For example, a long and thin super quadratic would be suitable for cutting tasks. However, the system has to test each tool in the physics engine in order to evaluate the affordance of a tool for the given task. Further development of this system would require much more time than it was available for me. Therefore, I decided to create a completely new system that would allow researchers to quickly prototype and test new ideas in analogy research.

2.6 Summary

This chapter briefly described the challenges we face in robotics research. It also describes two competing theories that try to explain the phenomenon of an analogy. In addition, this chapter critically evaluates and discusses the cutting-edge research paper about generalisation and adaptation of knowledge in robotics. Following a brief explanation of the problematic of collision detection in computing science. The last section describes and discusses existing systems that are similar to this project.

Chapter 3

Requirements & Architecture

The functional and non-functional requirements are specified and justified in this chapter. Both of them are sorted from the most to the least important requirements. Moreover, the system architecture and technical decisions are described and justified against requirements.

3.1 Functional Requirements

The goal of this project is to create a solution for faster and easier research in analogy for robot object manipulation in 3-dimensional space (3D). This solution should provide easy means for testing new ideas and algorithms. In addition, the solution should be able to learn from the user what is an ideal manipulation point with an associated force vector for a specific manipulation task in a specific situation. Then, it should be able to adapt and apply this knowledge in proposing the ideal manipulation point with an associated force vector for a specific manipulation task in an unseen situation. To achieve this goal, I need functional requirements from FR1 to FR5.

FR1 - The user should be able to load a 3D scene in some well established, open source file format as an input for the solution.

In order to quickly and easily create new 3D scenes with objects, the user should use existing programs that serve this purpose well (Don't Reinvent The Wheel). In majority drawing programs, the user is able to save a 3D scene in some open source file format that is technologically independent. Then, the user should provide this file containing the scene as an input to the solution. This should be applied for both adding new knowledge to the solution and also solving an unknown situation.

FR2 - The user should be able to select the target object, ideal manipulation point and force vector for a specific task in the 3D scene.

In the scene, there may be several objects from which only a one should be the target object selected for manipulation task. Moreover, for adding new knowledge, the target object should have an ideal manipulation point with an associated force vector for a specific manipulation task.

FR3 - The user should be able to see the whole scene with all objects in it.

It includes the target object, ideal manipulation point with the associated force vector. This is very important for the fast and easy testing of new ideas in an analogy. The main argument for this is that a human can faster evaluate if an ideal manipulation point is at the correct place from the presented picture rather than from presented 3D space coordinates.

FR4 - The user should be able to save the scene with associated data about analogy into a persistent memory[1].

The knowledge about the ideal manipulation point and force vector for the target object should be saved. This information should be used by the solution while solving the unseen situation (target object in the scene).

FR5 - The user should be able to get the analogous point with associated vector for the selected target object in a new scene based on the current knowledge.

If the user wants to get a suggestion of ideal manipulation point in a scene. He should be able to get the result based on the current knowledge of the solution that is using an analogy. This project focuses only on 2 basic manipulation operations that are a push and pull operation. In addition, to these basic manipulation operations, there will be a single operation using a spatula tool.

Due to the nature of this research topic, the above functional requirements (FR1 - FR5) are sufficient for reaching the stated goal. This is an open-ended cutting-edge research topic and very detailed functional requirements would limit exploration possibilities.

3.2 Non-Functional Requirements

I list and justify non-functional requirements (NFR1-4) here:

NFR1 - The solution will be modular.

This means that the solution will be split into separate manageable chunks of associated code called modules. It improves readability and maintainability.

NFR2 - The solution will be scalable.

New knowledge will be added to the solution by the user. Therefore, the solution has to be able to save and use the new scenes and knowledge associated with the target object manipulation in it.

NFR3 - The solution will be able to run on Fedora 29 and Ubuntu 18.04 LTS

Linux is a very popular operating system (OS) among computer scientists and it is open source. This means that is widely available for everyone. However, there are many distributions and the solution cannot be tested on all of them. Instead, I decided to choose these two major Linux distributions. Consequently, it should be possible to run the solution on Windows 10 OS because it introduced a new feature called Windows Subsystem for Linux (WSL) [27]. However, the compatibility of WSL is not explicitly required.

NFR4 - All the technologies used in this project should be free of charge.

All the technologies and tools should be free. However, they do not have to be necessary open source.

All these functional and non-functional requirements were designed after I consulted them with a potential user of the system. The potential user of the system happens to be my supervisor Dr Frank Guerin who is also a researcher interested in this topic.

[1] https://en.wikipedia.org/wiki/Persistent_memory

3.3 System Architecture

The following section describes a high-level system architecture that is designed to fulfil the functional and non-functional requirements. I considered several system architecture patterns that may be suitable for this project. The most promising were Pipes and Filters [28], Publisher-Subscriber [29] and Model-View-Controller [30] (MVC). The first two patterns are widely used in robotics. However, the MVC pattern is mainly known from web development and graphical user interfaces. Even though, based on my thorough evaluation of all of them, I decided to use Model-View-Controller architecture. It is the best fit for the purpose of this project. The following subsections aim to briefly explain each considered architecture and justify the final decision to use MVC architecture.

3.3.1 Pipes and Filters

This architecture pattern aims to decompose a complex monolithic task into smaller independent elements that can be reused. It is a well-known pattern from the Linux environment where one output from a program can be an input to another program. For example, using the cat program on a file and then piping the output to another program or a chain of programs. [28]

For example:

```
$ cat file | command_1 | command_2
```

This approach can improve scalability, reusability, parallelism and performance. The architecture diagram in Figure 3.1 aims to demonstrate this pattern.

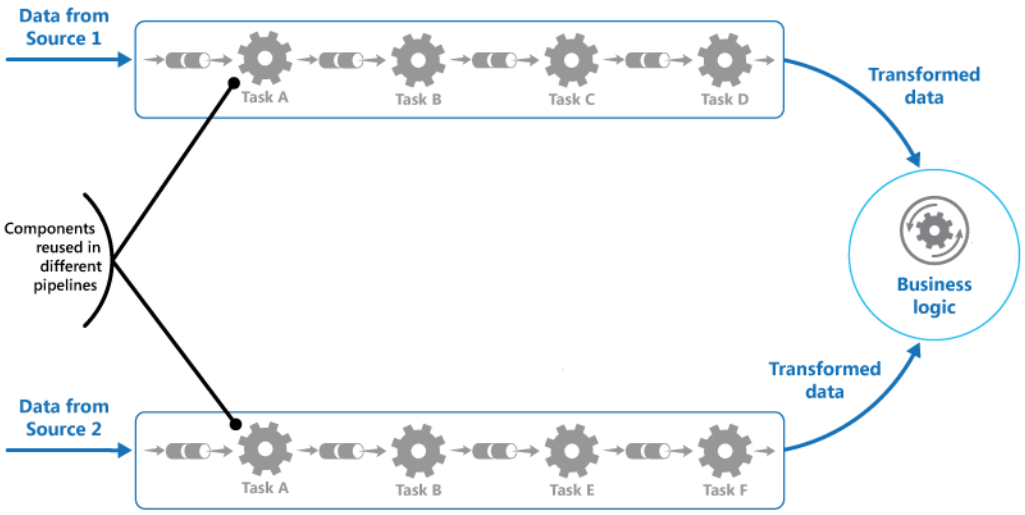

Figure 3.1: Pipes and Filters System Architecture [28]

As it is possible to see from the diagram in Figure 3.1 the data are processed in one direction. It means that after the “Task A" finishes then the “Task B" receives the input. This pattern is often used in robotics for sensor data processing. The robot can operate in a continuous manner. Receiving input from sensors, processing the input, planning actions and at the end acting.

However, this project focuses on the end user that is a researcher and not data processing from sensors. There is a fundamental difference. The solution should be able to dynamically interact with the user. Hence, there is not only one direction of data flow. Therefore, this architecture pattern is not suitable.

3.3.2 Publisher-Subscriber

It is also known as Pub/sub messaging. This pattern allows the publisher to announce an event (new data) to the interested subscribes asynchronously, without coupling them. Publisher and subscribers are usually independent processes. The advantage of using this pattern is that the system can scale up well, a subscriber cannot block the publisher and isolation of messages into channels. The disadvantages are that such a system is difficult to implement, one-directional communication, the order of messages is not guaranteed and complexity of the system. Therefore, we should always prefer to use existing systems rather than implementing our own. [29]

The Figure 3.2 shows the data flow in this architecture pattern.

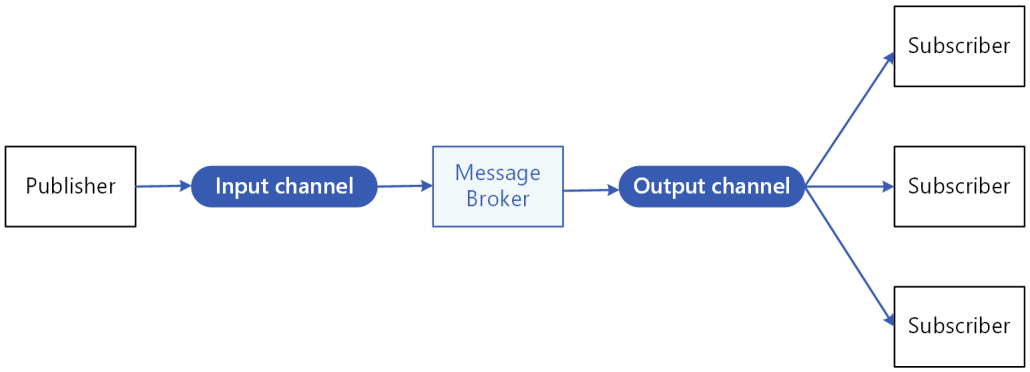

Figure 3.2: Publish-Subscribe System Architecture [29]

One of the widely used systems for robots, The Robot Operating System (ROS), is also using this architecture pattern [31]. However, for the purpose of this project this architecture is not suitable. It would be too complicated and time-consuming to implement it. If using ROS or other existing systems there would be introduced dependency and steep learning curve. In the end, this project is focused on the end user and should be easy to extend and edit.

3.3.3 Model-View-Controller

This architectural pattern nicely separates the code into a data processing part and a user interface processing part. In addition, this pattern is well established in web development and programs that use a graphical user interface. Also, it is simple to use and for fast development, this is very important. The architecture diagram 3.3 aims to explain the underlying structure of the system. Since the goal of this project is to create a research tool for a user that needs a visual output and interaction with the system, this architecture pattern is highly suitable. Further, I describe the responsibility of each architecture component. [30]

Model with Database

This component is responsible for data representation and storage. It should not depend on the controller or the view. In this case, it is responsible for representing the 3D scenes, target objects, ideal manipulation point, force vector and associated manipulation task. Also, it is responsible for storing and retrieving the data from a database (DB). This component is required to completely fulfil **FR4, NFR2** and partially fulfil **FR5**.

View

This component is responsible for representing the model's data to the user and sending

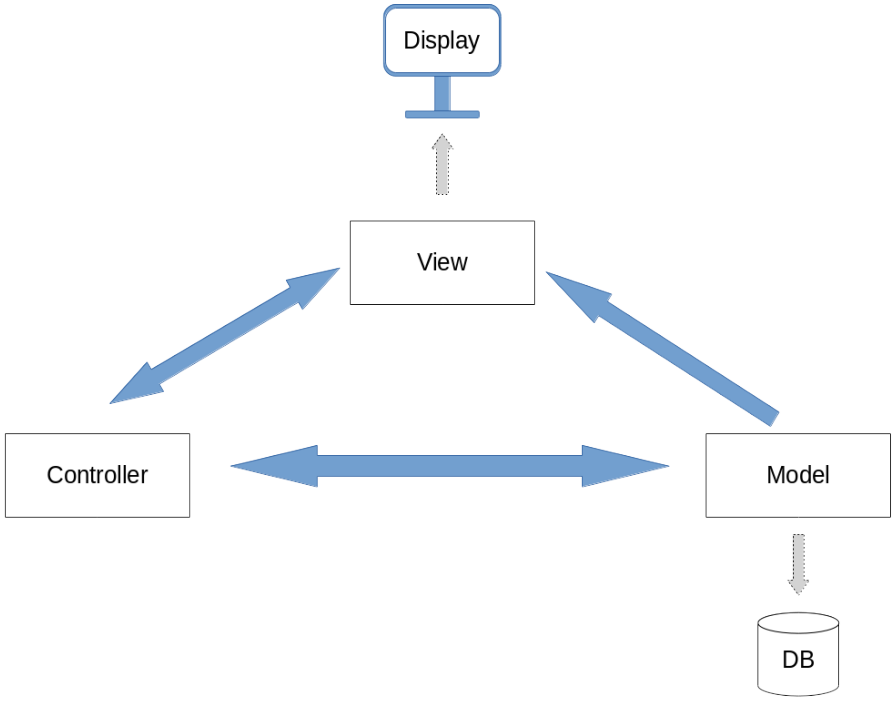

Figure 3.3: Model-View-Controller (MVC) System Architecture

the user actions to the controller. Again, it is an independent component of the model and controller. This component is required to completely fulfil **FR3** and partially fulfil **FR1, FR2 and FR5**.

Controller

This component is responsible for processing the user actions, validating them and communicating them to the model. In this case, it also contains the core of analogy algorithms. This component is required to partially fulfil **FR1, FR2 and FR5**.

3.4 Summary

This chapter described all considered architecture candidates and justified the choice of MVC system architecture that fulfils all the functional and one non-functional requirement **NFR2**. The remaining non-functional requirements are fulfilled in Chapter 4. In addition, Figure 3.3 describes the MVC system architecture that is used in the final solution. This includes specific reference to the functional and non-functional requirements.

Table 3.1: Table of all system architecture components and their contribution towards functional requirements fulfilment. Where ✓represents contribution and (-) represents no contribution.

Component	FR1	FR2	FR3	FR4	FR5	NFR1	NFR2	NFR3	NFR4
Model with Database	-	-	-	✓	✓	-	✓	-	-
View	✓	✓	✓	-	✓	-	-	-	-
Controller	✓	✓	-	-	✓	-	-	-	-

Chapter 4

Methodology, Technologies & Implementation

The methodology, technology, development tools and implementation are further described in this chapter. In addition, I also explain the major reasons behind the decisions.

4.1 Methodology

This project aims to explore the possibilities of using an analogy in robot object manipulation. As mentioned in Chapter 2, there are many different approaches trying to address an analogy in AI. Especially, this research is focused on abstract or invented tasks, but very little work in robotics that tries to exploit analogy. Moreover, in mainstream robotics, an analogy is almost nonexistent. Therefore, it is an open-ended, cutting-edge research topic in computing science. Traditional software development techniques are not suitable. Hence, I decided to adopt an exploratory programming approach [32], whereby I followed these steps:

1. Background literature reading and understanding the research topic.
2. Literature review of existing similar research projects.
3. Developing a project plan.
4. Learning about relevant software technologies, programming languages and tools used in this research area.
5. Establishing functional and non-functional requirements for the system.
6. Designing an over-arching architecture for the system which catered for the requirements.
7. Experimenting and exploring potential technologies, languages and tools suitable for the system.
8. Implementing and testing functionalities, integrating these with existing technologies.
9. Evaluating the developed system.

4.2 Technologies

This section describes the technology and tools that were used for the development of the system. It also presents the motivation for adopting specific techniques and tools.

4.2.1 3D file format

As the first thing, I had to decide which file format I am going to use for representing 3D scenes that will be the source information for the analogy system. There were several criteria that were crucial to this decision. The file format has to be open source, able to represent several objects in one scene, able to represent 3D objects and it should be some widely known file format. The candidates were these 8 most common 3D file formats: STL, OBJ, FBX, COLLADA, 3DS, IGES, STEP, VRML/X3D [33].

After applying the criteria and filtering out the unsuitable formats I was left with two options. The first one was using STL[1] format with ASCII encoding. The second was OBJ[2] format with ASCII encoding. Both would be suitable for this project. However, the STL format is older and does not support colours. This is not important for this project but I have to think about the future possibility to extend this project. On the other hand, the OBJ format is able to save information about colour and material texture. This information is stored in a separate file with the extension.MTL (Material Template Library). Therefore I decided to commit to OBJ file format.

4.2.2 3D drawing program

The following step was to decide which software I am going to use for drawing the 3D scenes. There were several requirements that had to be fulfilled. The software has to be free, export to OBJ format and easy to use. I considered and tested these options: 3D Slash[3], SculptGL[4], TinkerCAD[5], Wings 3D[6], Clara.io[7], Meshmixer[8], FreeCAD [9] and Microsoft 3D Builder [10].

After applying the requirements and evaluating all of them, I ended up with two final candidates. The first was FreeCAD and the second was Microsoft 3D builder. FreeCAD natively supports exporting to STL and OBJ files. However, it is more focused on Computer-aided design (CAD) and the development of scenes turns out to be more time consuming and complex than desirable [34]. Microsoft 3D Builder also natively supports export to OBJ file format and it is surprisingly free of charge [35]. Hence, I decided to use the Microsoft 3D builder for scenes drawing.

4.2.3 Database

As a next thing, I looked at several options for database technologies. There are two main types of databases. They are SQL and NoSQL. Also known as relational databases and non-relational databases. At the beginning of the project, I did not make any commitment to neither of them because I simply did not know which one is going to be more suitable. I did not know how I want to represent the knowledge, which data structures I am going to use or how the analogy algorithms are going to work. Therefore, I kept an open mind and considered them both until I saw the neat relations and how I could simply use a SQL database for representing the knowledge.

[1] https://en.wikipedia.org/wiki/STL_%28file_format%29
[2] https://en.wikipedia.org/wiki/Wavefront_.obj_file
[3] https://www.3dslash.net/index.php
[4] https://stephaneginier.com
[5] https://www.tinkercad.com/
[6] http://www.wings3d.com/
[7] https://clara.io/
[8] http://www.meshmixer.com/
[9] https://www.freecadweb.org/
[10] https://www.freecadweb.org/

Therefore, later on, I focus on choosing SQL database technology. I considered these candidates. They were PostgeSQL[11], MySQL[12] and SQLite[13]. Since the project should be a research tool and I prefer for this project simplicity and not performance, I decided that SQLite is going to be the best choice because it is lightweight, simple to deploy, widely used and it does not require running on a separate a server. Moreover, Python 3.7 provides a package for interfacing with SQLite API [36].

4.2.4 Scripting and Programming Language

Choosing the right programming language for a project is very important. This decision can improve readability, performance, future maintainability and decrease development time. For these reasons I carefully considered several languages that can be used for it. Firstly, I investigated what are the most popular programming languages. The most recent and complex survey that should represent this was Stack Overflow Developer Survey 2017 [37]. Because I adopted the exploratory programming technique that requires quick prototyping, I favoured languages that are more suitable for this approach. As the best language to use for the core development turns out to be the latest stable Python 3.7.3[14] to the date of writing this report.

Since the system is also going to utilise a Database for storing the knowledge, I am also using SQL language for communicating with SQLite.

4.2.5 Code Style

This aspect of software development is often underestimated. However, it is a very important thing for future development and cooperation with other developers. Because this project should serve as a research tool for others, it is important to establish the code style for naming conventions, package structures, comments, docstrings and many other important things. One of the most popular code styles for Python is Google Python Style Guide [38]. Using YAPF[15] that is set to use “google" style provides an automatic tool for quality assurance, formatting and style check.

4.2.6 Visualisation Of Scenes

After I knew that I am going to use Python as the programming language for this project, I looked for options on how to visualise the scene for the user. The requirement was that the user has to be able to interact with the scene (rotate, zoom in, zoom out, select target object, etc.). As the first thing I started to use PyOpenGL[16] and pygame[17] that is a widely adopted approach for visualisation of 3D objects with Python. However, after the first week of the software development cycle, I decided to change these libraries in favour of VPython 7[18]. This was mainly possible due to the wise choice of the software architecture that decouples the model from view. The main reason for this change was that using directly OpenGL API and dealing with low-level things such as frame buffers made the whole system excessively complex. On the other hand, VPython is presented as “3D Programming for Ordinary Mortals" and it simplifies many things. Therefore, I

[11] https://www.postgresql.org/
[12] https://www.mysql.com/
[13] https://sqlite.org/index.html
[14] https://www.python.org/downloads/release/python-373/
[15] https://github.com/google/yapf
[16] http://pyopengl.sourceforge.net/
[17] https://www.pygame.org/news
[18] https://www.glowscript.org/docs/VPythonDocs/index.html

decided to use VPython 7 for visualisation and user interaction.

4.2.7 IDE and Text Editor

For the development of the software, I considered using VIM[19] and Microsoft Visual Studio Code[20] (VSCode). I am using both of them every day. However, VSCode has good integration with Git version control that I also used in the development process. It also provides built-in debugger for Python programs, YAPF formatter and many other small features. This may seem a trivial thing, but the choice of the text editor can dramatically reduce the development time and increase the quality of the code. Hence, VSCode was the choice.

4.2.8 Source Control

As a source control, I decided to use Git[21] version control because it is stable an widely used in the industry. Moreover, I am familiar with Git as a daily user. For the development of the software, two major branches were set up. The first one was the master branch and the second was the dev branch. The master branch contains the latest stable version and features. On the other hand, the dev branch was used for the development of the new features and parts of the software. After every significant change, the repository was backed up to a remote location on Github.com.

4.2.9 Licensing

Considering which license is the right one for an open source software is very important. The reason is that many people write software and release it without any license. This software is therefore by default protected by copyright law and it cannot be copied, reproduced or included in other projects. Open source licenses, in brief, allow the software to be freely used, modified, and shared. I considered licenses that have the least limitations for future use. They were 2-Clause BSD [39], 3-Clause BSD [40] and MIT License [41]. 2-Clause BSD and MIT License are interchangeable. However, I did not want to allow to use my name or names of future contributors to endorse or promote products derived from this software without specific prior written permission. This is what is guaranteed in the 3-Clause BSD License. Hence, I chose the 3-Clause BSD License for this project.

4.3 Implementation

This section describes the implementation. As the first thing, I created a directory structure for the `analogy` package. This provides decoupling and the possibility to reuse this package in different projects.

4.3.1 Parsing OBJ File And Data Representation

Next, I created file parser for the OBJ file format. This was a fairly trivial task. However, once I was able to parse the file, I also needed to create new data structures for representing the data from the OBJ file. One option would be using data structures provided by VPython package. However, the view should be decoupled from the model. Therefore, I created new classes that represent the data from OBJ file. Docstrings (Google style) of these data structures and their attributes are presented and explained in Figures 4.1, 4.2, 4.3. For the full source code please see included file `analogy/mesh.py`.

[19] https://www.vim.org/
[20] https://code.visualstudio.com/
[21] https://git-scm.com/

Figure 4.1: Mesh class docstring in Python language.

```
class Mesh:
    """
    Mesh is a representation of a mesh in memory

    Attributes:
        name(str): Name of the mesh
        surfaces(list): A list of Surfaces that belong to the mesh instance.
        collision(bool): If the AABB has collision it is True.
        collided_objects(dict): Dict of all mesh names that the mesh collides
            with.
        aabb(AABB): Axis Aligned Bounding Box for the mesh
        color(list): A color of the mesh in RGB. Min value is 0.0 and max is 1.0.
            Default value is white that is [1.0, 1.0, 1.0].
    """
```

Figure 4.2: Surface class docstring in Python language.

```
class Surface:
    """
    Surface is a representation of a surface in memory

    Attributes:
        vertices(list): A list of vertices that belong to the surface instance.
        collider(list): 3D space coordinates of the collider for the surface.
            It is the centroid of the surface.
        collision(bool): If the surface has a collision it is True.
        collided_objects(dict): Dict of all names of meshes that the surface
            collides with.
    """
```

In order to improve efficiency and decrease the complexity of the whole system, I decided to calculate the Axis-Aligned Bounding Box (AABB) immediately while parsing the OBJ file. Hence, I also needed to create another data structure that is going to represent the AABB of each mesh object in the OBJ file. The docstring (Google style) of the AABB class and its attributes are presented and explained in Figure 4.4

4.3.2 Visualisation

As I already explained in Subsection 4.2.6, the first implementation was using OpenGL and pygame libraries for visualisation. These were later replaced for a single VPython library that is responsible for the view part of the MVC architecture. The biggest advantage of using VPython is that it provides high-level function calls and a variety of build in shapes that I could immediately use. However, to truly decouple the view from the controller and the model I implemented an adaptor (wrapper) around this library that source code is available in the file `analogy/vpython_drawings.py`.

Figure 4.3: Vertex class docstring in Python language.

```
class Vertex:
    """
    Vertex is a representation of a vertex in memory

    Attributes:
        pos(list): list of coordinates
    """
```

Figure 4.4: AABB (Axis-Aligned Bounding Box) class docstring in Python language.

```
class AABB:
    """
    AABB is a representation of a Axis Aligned Bounding Box in memory.

    Attributes:
        pos(list): list of [x,y,z] coordinates
        half_size(list): size of the AABB / 2 along x,y,z axis
            [width/2, height/2, depth/2]
        manipulation_points(dict): Dict of all known manipulation points.
        manipulation_vectors(dict): Dict of all known manipulation vectors.
        closest_surfaces(dict): Dict of 'side_name' and list of closest
    associated
            surfaces.
        collided_sides(dict): Dict of 'side_name' and encoding of collision
    .
            For eample, no collision = 0, partial collision = 1, full = 2.
        color(list): A color of the mesh in RGB. Min value is 0.0 and max
    is 1.0.
    """
```

4.3.3 Collision Detection

Implementation of collision detection plays a very important role in this analogy system. First of all, it plays a crucial role in the mapping algorithm for analogy. Then, it also provides information if a surface of the object is available for manipulation. In the previously implemented system [24] the user had to manually specify which surface of the object is collided and therefore not available for manipulation. For a human, it is a very straight forward process to determine if two objects collided. However, in computing science, it is a very difficult problem. Moreover, if this analogy package is used in a processing pipeline in a real-world robot, it would have to provide automatic collision detection. Since my goal was automating and simplifying the whole process for the user as much as possible, I considered it to be a necessary feature.

The first approach I decided to use was a triangle to triangle collision detection because a mesh is a set of triangles. These triangles are defined by vertices that have 3D space coordinates associated with them. The reason was that I believed there is no need for using axis-aligned bounding boxes (AABB). AABBs are mainly used because it is computationally less expensive to determine a collision of them.

My idea was that by using the object's mesh for determining a collision brings a better granularity and therefore accuracy. The triangles that are collided would be marked as unusable for manipulation and suggested manipulation points would be shifted to the nearest available triangle on the mesh. Hence, I looked at existing algorithms as mentioned in Chapter 2. As the first option

I tried this [42] implementation of “A fast triangle to triangle intersection test for collision detection" by Oren Tropp, Ayellet Tal and Ilan Shimshoni [18]. Unfortunately, there were problems with this implementation. Minor problems were the old programming style, loosing the underlying geometrical interpretation and using unconventional notations. The major problem was that in some instances it returned a wrong answer.

For example the triangles [a0,a1,a2] and [b0,b1,b2] are defined by the following points:

- a0 = (-21, -72, 63)
- a1 = (-78, 99, 40)
- a2 = (-19, -78, -83)
- b0 = (96, 77, -51)
- b1 = (-95, -1, -16)
- b2 = (9, 5, -21)

The result was that they do not intersect [43]. See the Figure 4.5 that shows these triangles.

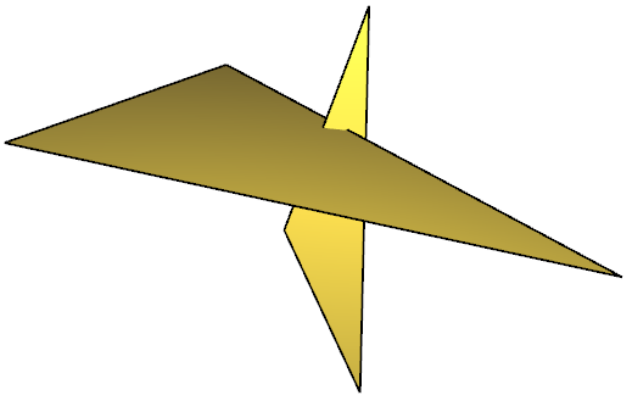

Figure 4.5: Triangle to Triangle intersection [43].

I also considered implementing this algorithm on my own. However, the goal of this project is not only implementing a collision detection algorithm. Therefore, I tried to look further for existing implementations. Because I could not find another open source implementation of this algorithm, I had to look at different options.

The next option was Moller’s algorithm [19]. I found this implementation in C [44] and used it. It turns out that it is working correctly. However, the performance of collision detection was very slow. Therefore, I wanted to try also Devillers algorithm [20]. I used this [45] implementation written in C. It also worked correctly on tested examples. In order to use those two algorithms (Mollers and Devillers) in this project, I had to write python wrappers for direct communication with C programs. Only for this reason the project also requires GCC compiler to compile those C source codes to shared object libraries. The whole process is automated by makefile that I prepared.

After an initial evaluation, I realised the drawback of using the triangle to triangle collision detection. First of all, it took too long. To calculate collision for a single object (1200 surfaces)

that collided with a second object (1200 surfaces) it took about 25 seconds. The result can be seen in Figure 4.6. Orange colour represents collided surfaces and red available for manipulation. In addition, if one object intersected deeper into the other one, the triangles (surfaces) that were inside the penetrated object were marked as not collided and therefore available for manipulation. This is shown in Figure 4.7 that is the inside view of the scene that is shown in Figure 4.6. The reason for that was that mesh is hollow inside and there is not any triangle to triangle intersection. Hence, without an axis-aligned bounding box, I could not determine if a vertex is inside an object. Even though I did not plan using the triangle to triangle collision detection anymore in this project, I kept the implementation in the analogy package as a ready to use tool for future improvements.

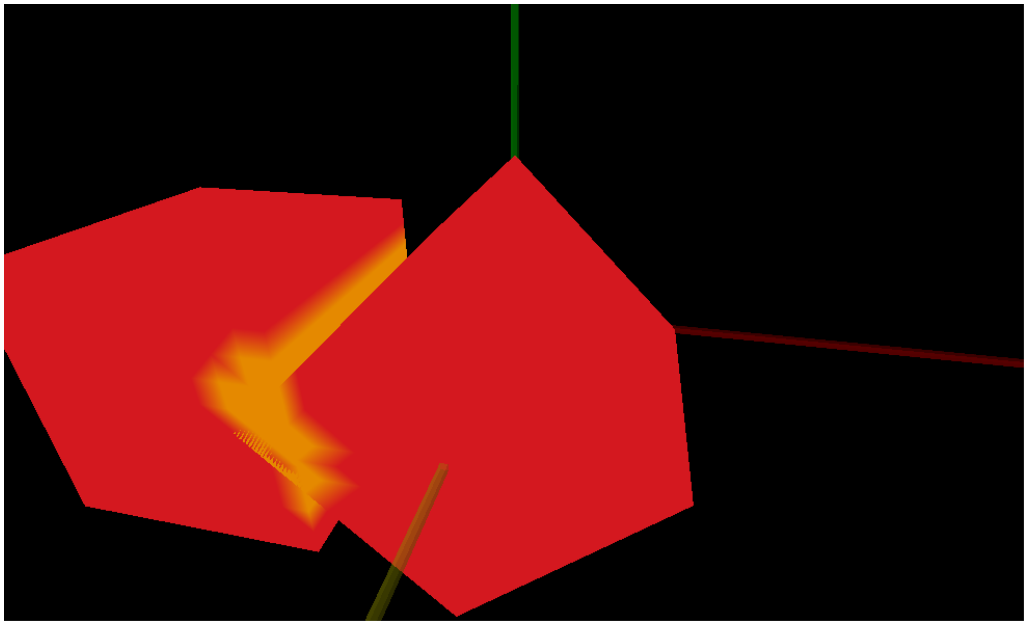

Figure 4.6: Collision two cubes, each with 1200 surfaces.

Figure 4.7: Collision two cubes, each with 1200 surfaces, inside view.

The solution was to using axis-aligned bounding boxes collision detection inspired by this [22] article from Mozilla. In addition, I wanted to know more about the problematic and I read Chapter 10 from the book Mathematics and Physics for Programmers (GAME DEVELOPMENT SERIES) [23]. The whole chapter focuses on “Detecting Collisions Between Complex Shapes". Since now I had solid theoretical knowledge about this problematic, I was able to implement the necessary algorithms on my own. I realised that there is a need for two collision detection algorithms. The first one is detecting if an AABB collides with another AABB. It has to be very fast and efficient because it tests every object in the scene against every other object in the scene. So the complexity of this algorithm is linear. See the implemented algorithm in the Figure 4.8. The second algorithm that I needed to implement was a vertex to AABB collision detection. As you can see the implemented algorithm in Figure 4.9, it is very similar and the complexity is linear.

Figure 4.8: aabb_intersect function in Python language.

```
def aabb_intersect(mesh_1, mesh_2, min_distance):
    # check X axis
    if abs(mesh_1.aabb.pos[0] - mesh_2.aabb.pos[0]) < (
            mesh_1.aabb.half_size[0] + mesh_2.aabb.half_size[0]) +
    min_distance:
        # check Y axis
        if abs(mesh_1.aabb.pos[1] -
               mesh_2.aabb.pos[1]) < (mesh_1.aabb.half_size[1] +
                                      mesh_2.aabb.half_size[1]) +
    min_distance:
            # check Z axis
            if abs(mesh_1.aabb.pos[2] - mesh_2.aabb.pos[2]) < (
                    mesh_1.aabb.half_size[2] +
                    mesh_2.aabb.half_size[2]) + min_distance:
                return True
    return False
```

Figure 4.9: aabb_intersect_vertex function in Python language.

```
def aabb_intersect_vertex(mesh, vertex, min_distance):
    # AABB (Axis Aligned Bounding Box)
    # check X axis
    if abs(mesh.aabb.pos[0] - vertex[0]) < (
        (mesh.aabb.half_size[0]) + min_distance):
        # check Y axis
        if abs(mesh.aabb.pos[1] - vertex[1]) < (
            (mesh.aabb.half_size[1]) + min_distance):
            # check Z axis
            if abs(mesh.aabb.pos[2] - vertex[2]) < (
                (mesh.aabb.half_size[2]) + min_distance):
                return True
    return False
```

The complete collision detection in the system works as follows:

1. For each mesh object in the scene creates an axis-aligned bounding box (AABB).
2. For each surface of the AABB create an associated collider. (It is a centroid[22] of the triangle

[22] https://en.wikipedia.org/wiki/Centroid

that belongs to the surface).

3. Iterate through each AABB in the scene and check if it collides with some other AABB in the scene.

4. If there is an AABB collision then for each surface of the first AABB check if the associated colliders (vertices) collide with the second AABB.

5. If all colliders of the surface have a collision with the second AABB mark the associated AABB surface as fully collided.

6. If there is a collider to AABB collision but there is at least one associated surface collider that does not collide then mark the surface as partially collided.

4.3.4 Database Design And Adaptor

The system design requires us to store knowledge into persistent memory. Consequently, my next step was to design the database. The initial design went through the normalisation process. As the first step towards normalisation, I applied rules to get it to the first normal form[46]. Then I continued and applied additional rules to get the database design to the second normal form [47]. In the end, I went all the way to the third normal form [48]. This means that every entry in each table should not have any transitive dependency. The advantages of removing transitive dependency are that the amount of data duplication is reduced and data integrity is achieved. The resulting database design is shown as UML diagram in Figure 4.10.

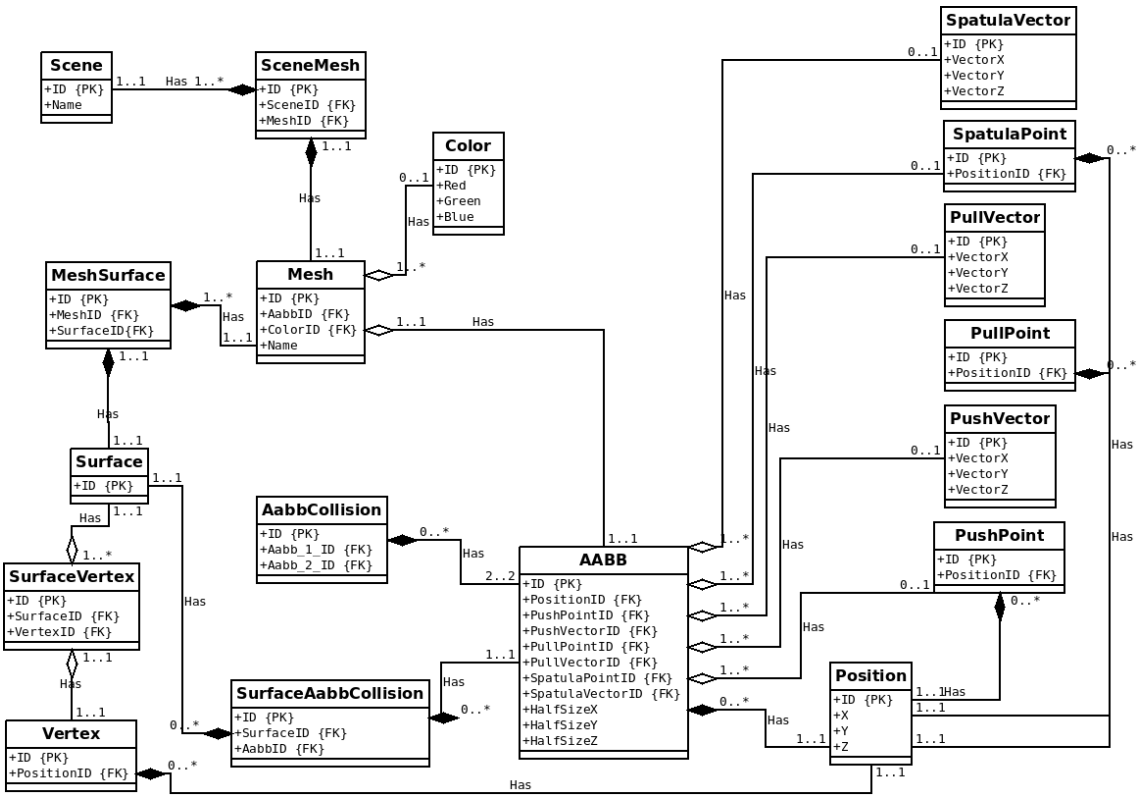

Figure 4.10: UML diagram of the first design of the relational database.

Later I realised that I can simplify the design if I do not want to reconstruct the whole scene from the SQL database. Therefore, I decided to separate the knowledge about the whole scene and the knowledge that is currently required for the analogy algorithm. The whole scene is stored as a pickle[23] object in the persistent memory. This allows the reconstruction of the whole scene in the case of future use. The knowledge that is currently required for the analogy algorithm is stored in the SQL database. The simplified UML design of the database in the third normal form is shown in Figure 4.11. There is the following advantage of using this approach. If the user decides to change the analogy algorithm and requires additional information that was not stored in the SQL database, it is possible to extract them later from reconstructing the whole scene while keeping the SQL database simplified.

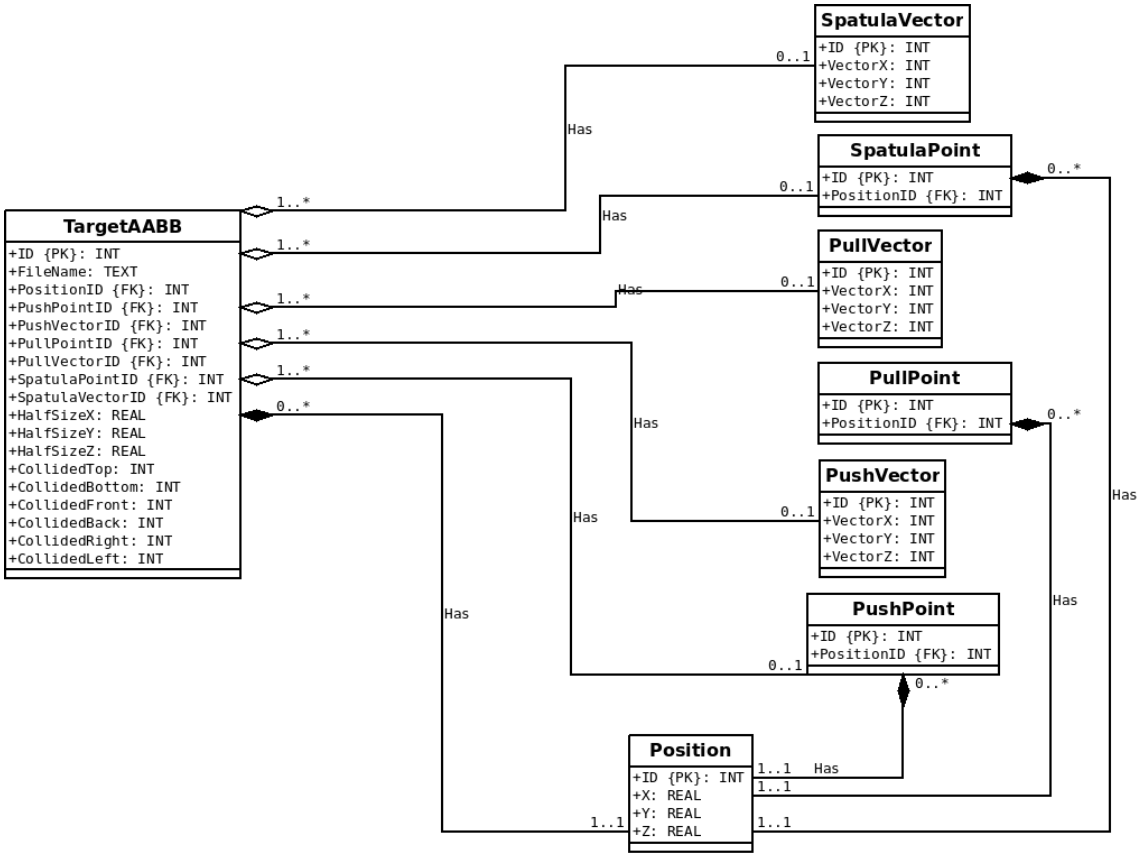

Figure 4.11: UML diagram of the used relational database.

After I had the database design I had to write Python functions that call SQL queries to the database (Create tables, get or save different data, etc.). They can be found in the file `analogy/storage/sqlitedb.py`. This file serves as an adaptor between the SQL database and the analogy system. If the user decides to change tables, edit SQL queries or change the underlying database design, the analogy system won't be influenced if the storage adaptor interprets it adequately.

[23] https://docs.python.org/2/library/pickle.html

4.3.5 Analogy Mapping and Scoring Function

The last piece to the puzzle was to design and develop the core analogy algorithm. I was inspired by “The Copycat Project: A Model of Mental Fluidity and Analogy-Making" [8] and its adaptation to object manipulation in the project “Robotic Automatic Task Oriented Manipulation" [24]. However, I saw the poor implementation that did not follow any programming style or convention. Moreover, the specific algorithm required knowledge about normals[24] of all surfaces that this new system could not provide from the generated OBJ files. Therefore, I decided to completely rewrite, simplify and decrease the complexity of the algorithm. It is important to say that the underlying idea of mapping a target cuboid object to a source object that has a known manipulation point remains the same. However, this may be considered the only common thing with the previously mentioned implementation. There are many differences and improvements. For example, ideal manipulation points associated with the goal operation (pull, push, using spatula), force vectors associated with manipulation points and the goal operation, etc.

As the first thing for the new analogy algorithm, I had to find out what is the rotation group of a cube. Considering only rotations that do not change its shape (multiples of 90° rotations about x,y or z-axis). There are several ways of calculating this. The way I did it was to number the vertices of the cube and observe the effect of each rotation on these vertices. See the Figure 4.12.

For example:

- x = rotation about x = (1, 4, 8, 5) (2, 3, 7, 6)
- y =rotation about y = (1, 5, 6, 2) (4, 8, 7, 3)
- z = rotation about z = (1, 2, 3, 4) (5, 6, 7, 8)

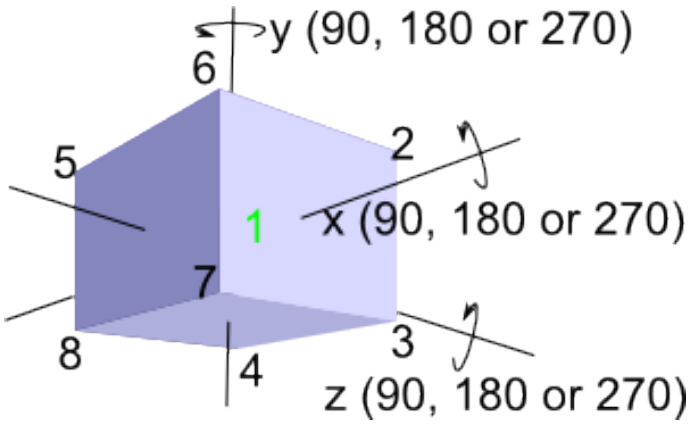

Figure 4.12: Vertex numbers of a cube and the direction of a cube rotation [49].

Now, I just had to find all the permutations of these rotations. The result is that there is 24 permutations including the identity element [49] [50]. Also, it is important to say that the effect of rotation about the z-axis can be achieved by a combination of rotations about x and y-axis. These permutation sequences can be seen in Figure 4.13. Sequences have to be read backwards. This is the standard notation in literature. For example, the sequence (x,x,y) means 90° rotation about y axis followed by 180° (90°+90°) rotation about x axis.

[24] https://en.wikipedia.org/wiki/Normal_(geometry)

```
('i'), ('x'), ('y'), ('x', 'x'), ('x', 'y'), ('y', 'x'), ('y', 'y'),
('x', 'x', 'x'), ('x', 'x', 'y'), ('x', 'y', 'x'), ('x', 'y', 'y'),
('y', 'x', 'x'), ('y', 'y', 'x'), ('y', 'y', 'y'), ('x', 'x', 'x', 'y'),
('x', 'x', 'y', 'x'), ('x', 'x', 'y', 'y'), ('x', 'y', 'x', 'x'),
('x', 'y', 'y', 'y'), ('y', 'x', 'x', 'x'), ('y', 'y', 'y', 'x'),
('x', 'x', 'x', 'y', 'x'), ('x', 'y', 'x', 'x', 'x'),
('x', 'y', 'y', 'y', 'x')
```

Figure 4.13: All 24 permutation sequences of a cuboid.

```
{
    ('i'): {'top': 'top', 'bottom': 'bottom', 'front': 'front',
        'back': 'back', 'right': 'right', 'left': 'left',},
    ('x'):{'top': 'back', 'bottom': 'front', 'front': 'top',
        'back': 'bottom', 'right': 'right', 'left': 'left',},
    etc.
}
```

Figure 4.14: Sample of 24 permutation sequences with resulting surface mappings as a Python dict.

Once I had all these 24 permutation sequences, I wrote a method `create_permutations` for a class `Mapping` that creates all surface mappings as a Python dictionary. The sample of the first two entries in the created dictionary is shown in Figure 4.14. It has the following form. {permutation sequence -> { surface -> surface after rotation}}

Finally, I was able to write the scoring algorithm that evaluates and assign scores for each permutation sequence.

The main parameters that play a role in the scoring algorithm are:

- Matching surface collisions after applied rotation on the target object. This parameter seems to be the most important because a collided surface cannot be used for manipulation. (Except using spatula.)
- Matching surface names after applied rotation on the target object. This parameter seems to be the least important. However, I think that sometimes the properties of top or bottom surface may play an important role in an analogy.
- xy, zy and xz size ratio differences between the source object and the target object after rotation. These properties turn out to be also important. While thinking about the analogy, I realised that the ratio between the tallness of an object and the size of a base is important for specific manipulation operations.

The high-level description of the algorithm is shown in Figure 4.15.

I believe that the algorithm can be best explained and understood, without any ambiguity, only by reading the implementation. Therefore, the implementation of this core algorithm for this project is shown in Listing A.1.

As you can notice on line 2 of the Figure 4.15, the importance of each of the parameters that play roles in the analogy can be adjusted by changing the values of the weights. Moreover, the parameters can be even penalised with negative values of weights, if necessary. The algorithm

```
def get_mappings_score(target_aabb, source_aabb):
    Initialise weights for exact_collision_weight, partial_collision_weight
    , no_collision_match_weight, top_match_weight, bottom_match_weight,
    front_match_weight, back_match_weight, right_match_weight,
    left_match_weight, no_surf_match_weight, xy_ratio_weight,
    zy_ratio_weight, xz_ratio_weight.

    for perm_sequence in all_permutations.keys():
        score =  1.0
        for surface in all_permutations[perm_sequence].keys()
            # Increment score for surface collision match
            if surface of the target aabb has the same type of collision
                (none, partial or full) as surface of the source aabb after
                rotation:
                score += 1.0 * exact_collision_weight
            elif surface of the target aabb  does not have none collision
                and surface of the source aabb after rotation does not have
                none collision:
            # this means that we match the partial and full collision of
            # surfaces
                score += 1.0 * partial_collision_weight
            else: # there is no match
                score += 1.0 * no_collision_match_weight # It will be + 0
                # score because of the weight

            # Increment score for surface name match
            if (surface == 'top' and
                all_permutations[perm_sequence][surface] == 'top'):
                score += 1.0 * top_match_weight
            elif (surface == 'bottom' and
                all_permutations[perm_sequence][surface] == 'bottom'):
                score += 1.0 * bottom_match_weight
            elif ...
                etc. for all surface names.

        for rotation in reversed(perm_sequence):
            rotate the target aabb accordingly. This rotates the target
            aabb to source scene.

        # Now, compare xy, zy, xz ratios of target and source aabbs after
        # rotation and penalise for difference
        xy_ratio_diff = abs((abs(vpython_vec_xyz.x) /
            abs(vpython_vec_xyz.y)) -
            (source_aabb.half_size[0] / source_aabb.half_size[1]))
        xy_ratio_diff *= xy_ratio_weight
        ... etc. for zy, xz ratios.

        score -= xy_ratio_diff + zy_ratio_diff + xz_ratio_diff
        mappings_score.append((score, perm_sequence))
    return mappings_score
```

Figure 4.15: Pseudocode for the core analogy algorithm for scoring of mappings.

was designed with this in mind for the potential future improvements and use of machine learning. The current values of weights shown in Listing A.1 yielded good performance during an evaluation process.

After receiving the list of tuples with `(score, perm_sequence)`, it has to be sorted from the highest to the lowest score. The system keeps only the best 3 permutation sequences with the highest score. This process of evaluating the best permutation sequence for mapping between the target object and the source object repeats for all source scenes that are added to the knowledge base. Then, the system picks the top 3 scenes with the highest score from all known scenes. Even though the system uses only the knowledge from the highest scoring scene and its permutation sequence, it still keeps in the memory the other options. It means that the system keeps track of the top 3 source scenes each with top 3 permutation sequences. The reason for it is that if the user or other entity (robot, physics engine, etc.) evaluates that the choice was not good, the system may propose other solutions to the manipulation problem. This approach of competing for best options was inspired by Douglas Hofstadter's presentation [4].

Once the system has the source scene and the specific permutation sequence, it has to adequately rotate the ideal manipulation points with their force vectors back from the source object to the target object. This process requires converting the source and target object's absolute 3D space coordinates to relative 3D space coordinates. Then, it has to reverse the best permutation sequence and rotate adequately the relative space coordinates of ideal manipulation points and their force vectors. Now, we end up with correctly rotated manipulation points and force vectors from the source object to the target object. However, there is the last step the system has to do. It is scaling the position of these points and force vectors because the source object may have different dimensions than the target one. This process is implemented and can be seen in the file `analogy.py` on lines from 262 up to 378.

4.4 Summary

This chapter discusses methodology, technologies and implementation. The methodology that I decided to adopt is an exploratory programming approach, whereby all the steps taken were described. Among many 3D file formats, I decided to use the OBJ file format. For the drawing of 3D scenes I used Microsoft 3D Builder. The majority of the code is written in Python 3.7 (Google Code Style) and algorithms for the triangle to triangle collision detection are implemented in C language. All of the code base was written in Microsoft Visual Studio Code. As a version control, I used Git. As storage for the scenes, I used SQLite. The whole project is released under the 3-Clause BSD license. All of the mentioned technologies and tools are free. Therefore, they completely fulfil **NFR4**. Moreover, the implementation is modular and scalable. Hence, it completely fulfils **NFR1 and NFR2**. Also, the system was developed on Fedora 29 and during the whole developed tested on Ubuntu 18.04. Therefore, the last requirement **NFR3** is completely fulfilled.

The Implementation Section describes the implementation and all crucial parts of the system. Especially, it focuses on how all parts work together. Also, the core analogy algorithm is described in details.

Chapter 5

Testing & Evaluation

This chapter describes tests conducted on the system. It also describes the tools and process used for the evaluation of the developed system. Further, all the results are presented, analysed, explained and discussed.

5.1 Testing

Since I used exploratory programming approach, I required very often to try new ideas and test them.

There were several prototypes that turned out to be a dead end. During this initial stage, the main approach for testing was using a diagnostic print out messages that reported intermediate values to the terminal. This is considered white-box testing of an internal structure of the functions. As the first functions started to emerge, I visualised the results using VPython on the screen. This approach allowed me to quickly evaluate the results and find bugs. It may seem like a not professional approach. However, I would argue that this way it is much faster and efficient to spot a problem than verifying a list of 1200 surfaces with 3D space coordinates. For example, see the previously mentioned collision detection problem in Figure 4.7. Visualisation of scenes also tested the existing functions as a black-box every time I launched the system. This provided me with quick feedback in the case that I accidentally broke the existing functionality by introducing a new one or changing the existing one. In such a scenario I used the diagnostic print out messages to debug the code.

The overall methodology for testing during the development was to test for boundary values, unexpected inputs and expected inputs. I performed this testing for each method and function I wrote. However, this system should be considered as a prototype and it is not an industrial quality code yet. The target user is expected to tinker with the system, adjust values and algorithms. Therefore, there is no need to write unit tests. These would be failing with any change in the code. Hence, they would completely miss the purpose of this project.

5.2 Qualitative Evaluation

As an evaluation process of the system, I decided to use qualitative evaluation. The main reason for this is that there are not nearly enough scenes for any meaningful statistics. Hence, I will be the judge that decides if the system's decision was correct or not. Moreover, I will directly compare the results with the previous project. The previous project was a final year project of an MSc student in Artificial Intelligence, "Robotic Automatic Task Oriented Manipulation" [24]. In the

end, I'll display the results next to each other.

For this reason, I had to remodel all the scenes from [24]. Using the same dimensions of objects and their placement in the scene, I created all 13 scenes. It is important to say that the old system is not capable of suggesting a goal-oriented manipulation points based on the desired operation (push, pull, etc.). Also, it is not capable of suggesting a force vector associated with a manipulation point. Moreover, it is not possible to use the old system with several source scenes. Therefore, knowledge is automatically limited to a single scene. All these limitations were removed in the new system and I consider them to be a huge improvement.

Specifically, I thought that the ability to choose among several source scenes should improve the system's performance dramatically. Therefore, I decided to do two evaluations. The first one with only one piece of knowledge in the database. See Figure 5.1. This piece of knowledge was the same as in the old system.

The second evaluation was conducted with knowledge about all scenes except the one that is being evaluated. For this purpose, I used the file `all_scenes.db` and created copies for each scene. Then, I simply removed the single entry of the scene that is being evaluated from the `TargetAABB` table. This guaranteed that the system does not have knowledge about the evaluated scene. It allows me to evaluate if the approach of using the knowledge base and multiple source scenes improves the performance in terms of better suggestions of manipulation points and vectors. This approach is similar to cross-validation in machine learning where we hold back the examples that are being evaluated.

5.2.1 Hardware And Software Used For Evaluation

Even though the hardware specification is not very important for this evaluation, I list it here. I wanted to show that the system can be used on older hardware and it does not resource demanding. My laptop is 8 years old HP EliteBook 8470p with the following parameters:

The evaluation was conducted using this hardware and software:

- Laptop Model: HP EliteBook 8470p
- CPU: Intel Core i7-3520M CPU
- Memory: Micron 8GB 2x4GB DDR3 1600 MHz
- OS Kernel: kernel-5.0.9-200.fc29.x86_64
- VM Swappiness: 0
- Python version: 3.7.3

5.2.2 Source Scene Books On A Shelf

In the first evaluation, I used the same source scene as in the old system. See Figure 5.1. The target object for manipulation is the middle book. On the left side, you can see the actual scene from the old system. As you may notice there is a problem with missing the back wall where the shelf is mounted. This mistake repeats in several other scenes. Be it Figure 5.2, 5.3, 5.4, 5.10 and 5.11. This mistake is not repeated in newly created scenes for the new system.

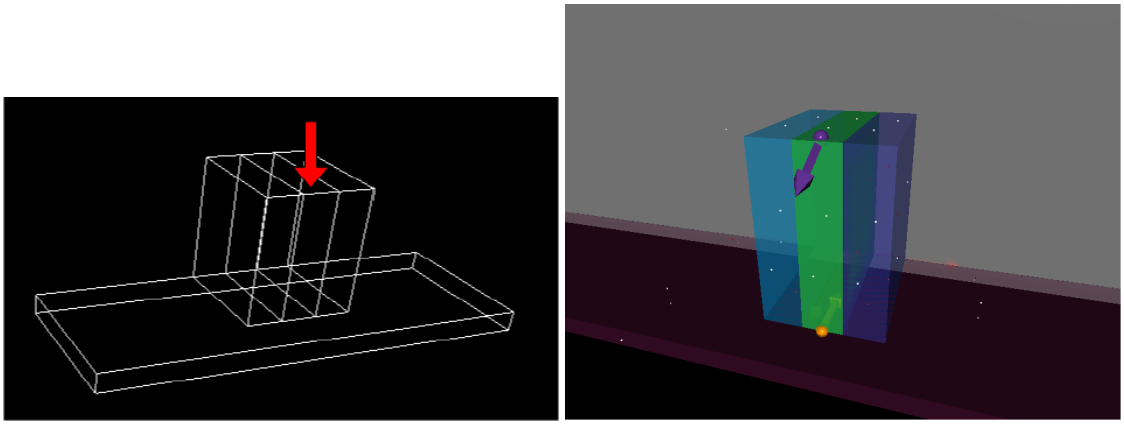

Figure 5.1: Books on a shelf. Left- old system with manipulation point. Right - new system with manipulation points and vectors.

After I read the source code of the old system, I understood that it does not play any role in the analogy mapping and scoring function. It is only a visual issue with the old system. The target objects in the affected scenes were correctly marked as if there were a collision on the back surface. However, while reading the source code of the old system, I found that the ideal manipulation point for the target object on the scene has limitations. It can be defined only at the edge of two adjacent surfaces. In addition, the point can be only exactly in the middle of the edge that is defined between two vertices. This is definitely not ideal because you do not want to have a manipulation point always only at the edge.

In Figure 5.1, on the left side you can see the red arrow that points to the ideal manipulation point. It is the middle of the edge between the front and top surface. The arrow does not represent a force vector. On the right side, there you can see the scene for the new system created in Microsoft 3D Builder. The purple sphere on the top surface of the book represents the ideal manipulation point for pull operation. The purple arrow that originates from this sphere represents a force vector that should be applied to the ideal manipulation point. The orange sphere at the bottom of the book represents the ideal manipulation point for using a spatula. The orange arrow that originates from this sphere represents a force vector that should be applied to the ideal manipulation point.

In the following scenes, you may also see the cyan sphere and arrow. This colour represents the push operation. However, you cannot see it in Figure 5.1 because the push operation would not make sense. Hence, in this scene, the ideal manipulation point and its force vector for the push operation is defined as `none`.

I used this colour scheme to distinguish among several manipulation operations:

- Purple colour = pull operation
- Cyan colour = push operation
- Orange colour = using a spatula operation

The criteria to pass the evaluation test for the old system is that it has to suggest a manipulation point that would achieve the stated goal for the scene after applying pull operation on the suggested manipulation point. Because the system is incapable of suggesting a force vector, I will

always consider the force vector in the best possible direction based on my judgement. The criteria to pass the evaluation test for the new system has to be the same. Therefore, I would consider a pass only if the new system suggests correctly the ideal manipulation point for pull operation. However, I'll consider only the proposed pull force vector. The remaining suggested manipulation points will not be considered for this evaluation. However, I will describe the results also for the other suggested manipulation points and vectors.

5.2.3 Target Scene Books On A Shelf 2

In the scene, shown in Figure 5.2, you can see three books on the shelf.

- **Goal**: The goal is to move the middle book such that it exposes the bottom surface without moving the left and the right book. This would allow someone to pick up the book.
- **Ideal manipulation point for the pull operation:** Back surface of the middle book.
- **Result for the old system:** Fail
- **Result for the new system with a single piece of knowledge:** Fail
- **Result for the new system with improved knowledge:** Fail

On the left side, you can see the suggested manipulation point by the old system. It is on the edge of the front-right surface. I consider this suggestion to be a failure. On the right side, there are suggested two manipulation points with their force vectors. The pull manipulation point is on the left surface of the book. The spatula manipulation point is on the right surface of the book. I consider the suggestion to be a failure too. However, if we look at the middle picture, we can see that the new system, containing more knowledge, suggested correctly the manipulation points for the spatula and push operation. Both operations would achieve the goal. The issue is that pull operation would fail. Therefore based on the previously stated evaluation criteria, the new system failed too in both trials.

I consider the ideal manipulation point for pull operation to be at the middle of the back surface with a vector straight towards the front surface. This operation would result in sliding the book towards the edge of the shelf. Hence, the bottom surface would be exposed and available for manipulation. The reason why the system did not suggest this option was that it did not have a similar situation in the knowledge base. The result of the analogy mapping and scoring algorithm was to match this scene "Boxes on a shelf" with rotation sequence: ('x', 'x', 'x', 'y'). This rotation sequence results in {'top': 'front', 'bottom': 'back', 'front': 'left', 'back': 'right', 'right': 'bottom', 'left': 'top'}. For example, the first pair says that top surface becomes front after applying this rotation and this continues for all the remaining surfaces.

5.2.4 Target Scene Boxes On A Shelf

In the scene, shown in Figure 5.3, you can see three boxes on the shelf.

- **Goal**: The goal is to move the front box such that it exposes the back surface without moving both boxes behind it. This would allow someone to pick up the box.
- **Ideal manipulation point for the pull operation:** Middle of the top surface of the front box.

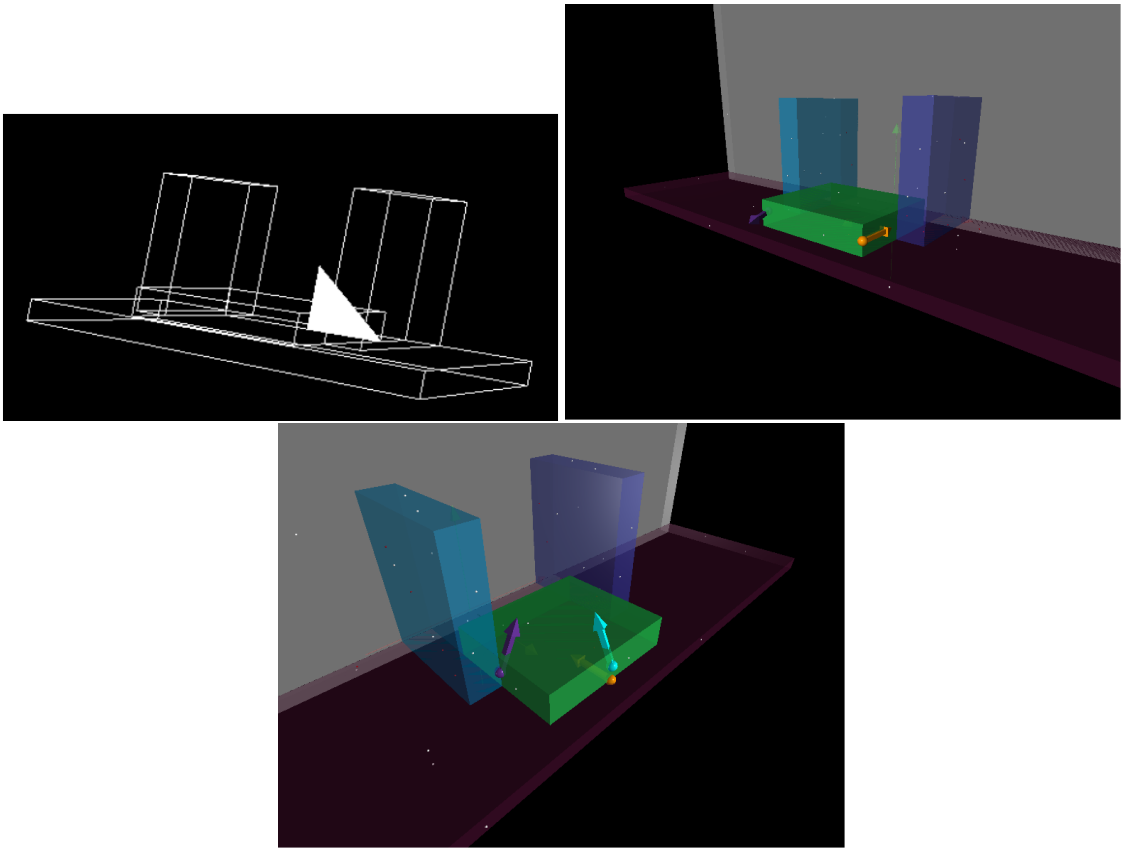

Figure 5.2: Books on a shelf 2. Left- old system. Right - new system with single piece of knowledge. Middle - new system with knowledge about all scenes except this one.

- **Result for the old system:** Fail
- **Result for the new system with a single piece of knowledge:** Fail
- **Result for the new system with improved knowledge:** Pass

On the left side, you can see the suggested manipulation point by the old system. It is on the edge of the top-right surface. I consider this suggestion to be a failure. On the right side, there are suggested two manipulation points with their force vectors. The pull manipulation point is on the top-right surface of the front box. The spatula manipulation point is on the bottom-right surface of the box. I consider the suggestion to be a failure too. However, if we look at the middle picture, we can see that the new system, containing more knowledge, suggested correctly all three manipulation points and their force vectors. All three operations would achieve the goal, including the pull operation. Therefore based on the previously stated evaluation criteria, the new system passes.

5.2.5 Target Scene Boxes On A Shelf 2

In the scene, shown in Figure 5.4, you can see three boxes on the shelf.

- **Goal**: The goal is to move the top box such that it exposes the bottom surface. This would allow someone to pick up the box.

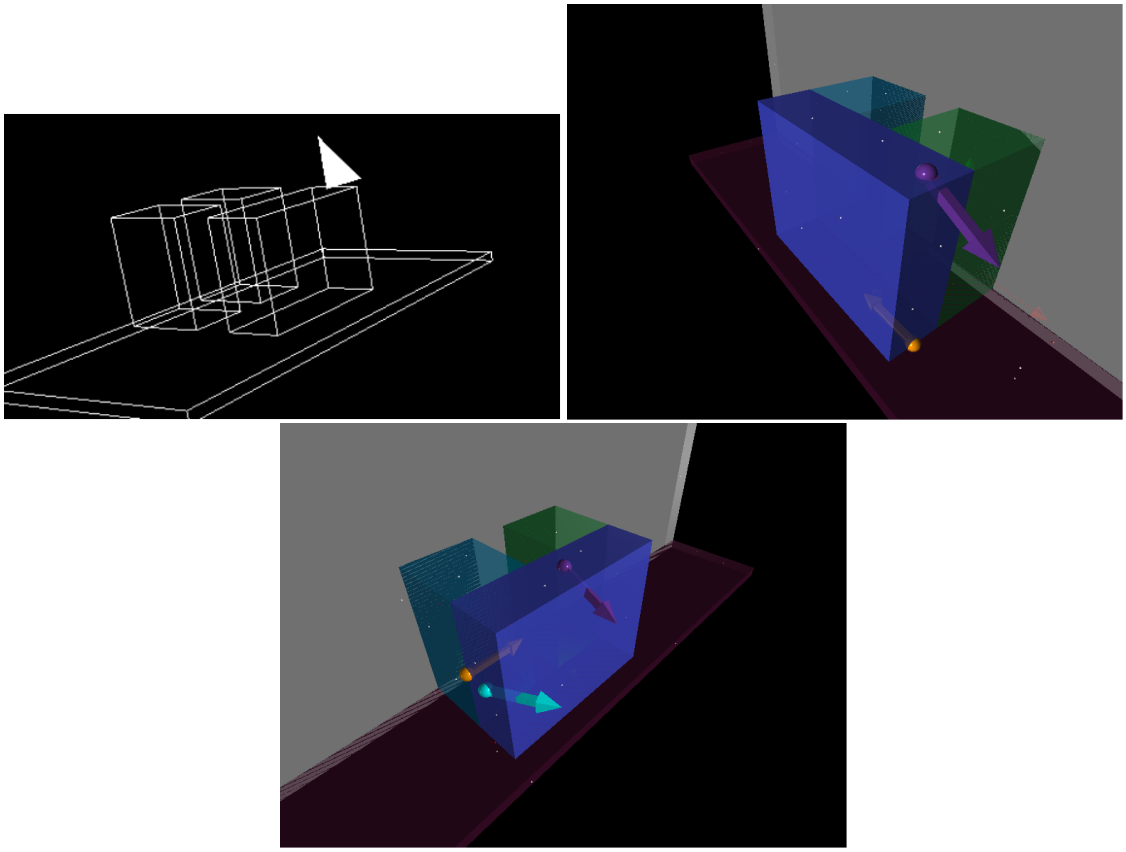

Figure 5.3: Boxes on a shelf. Left- old system. Right - new system with single piece of knowledge. Middle - new system with knowledge about all scenes except this one.

- **Ideal manipulation point for the pull operation:** Middle of the front surface of the top box.
- **Result for the old system:** Pass
- **Result for the new system with a single piece of knowledge:** Pass
- **Result for the new system with improved knowledge:** Pass

On the left side, you can see the suggested manipulation point by the old system. It is on the edge of the top-front surface. I consider this suggestion to be a good one because I could apply force on the edge and pull the box upwards. This would expose the bottom surface. On the right side, there are suggested two manipulation points with their force vectors. The pull manipulation point is on the left-front surface of the top box. The spatula manipulation point is on the right-front surface of the box. I consider the suggestion to be correct too. If we look at the middle picture, we can see that the new system, containing more knowledge, suggested correctly all three manipulation points and their force vectors. All three operations would achieve the goal, including the pull operation. Therefore based on the previously stated evaluation criteria, the new system passes.

5.2.6 Target Scene Box In A Corner

In the scene, shown in Figure 5.5, you can see one big box in the corner.

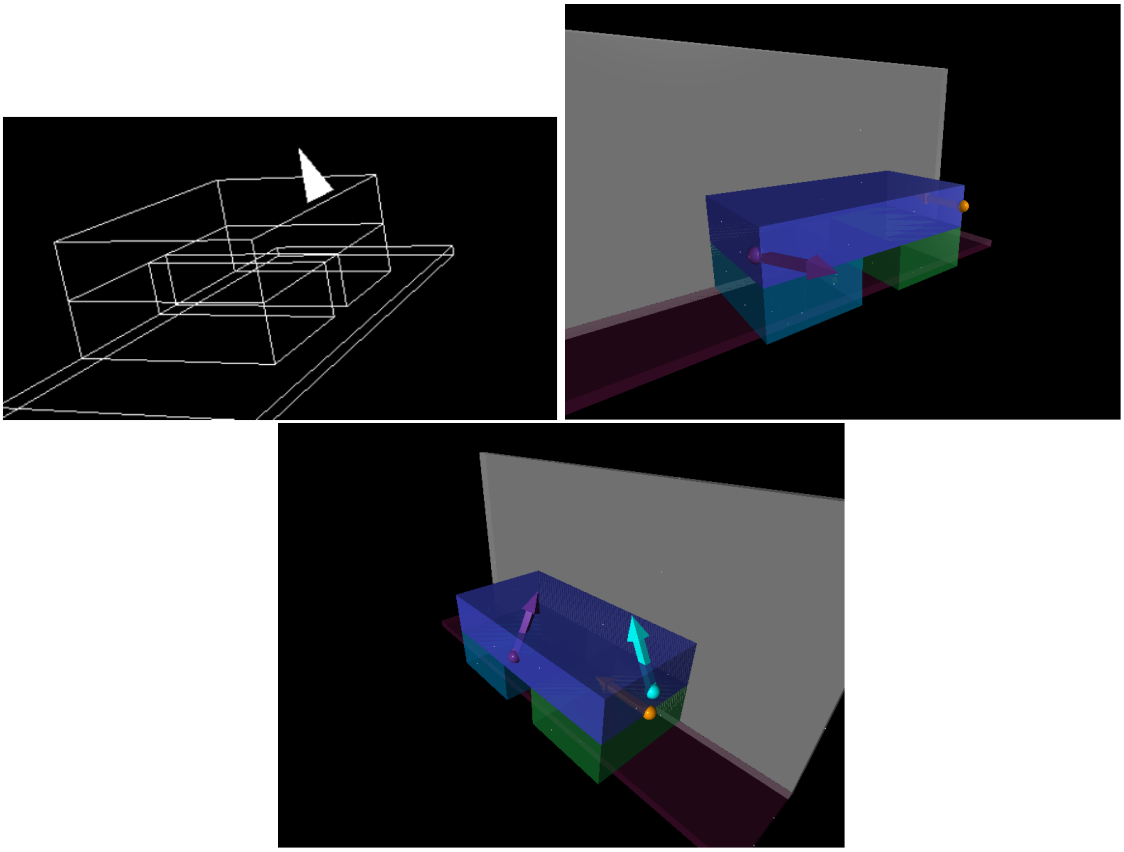

Figure 5.4: Boxes on a shelf 2. Left- old system. Right - new system with single piece of knowledge. Middle - new system with knowledge about all scenes except this one.

- **Goal**: The goal is to move the box such that it exposes one of the blocked surfaces from the walls (back and right surfaces). This would allow someone to manipulate the box.
- **Ideal manipulation point for the pull operation:** Front or left surface.
- **Result for the old system:** Fail
- **Result for the new system with a single piece of knowledge:** Fail
- **Result for the new system with improved knowledge:** Pass

On the left side, you can see the suggested manipulation point by the old system. It is on the edge of the top-left surface. I consider this suggestion to be a failure. On the right side, there are suggested two manipulation points with their force vectors. The pull manipulation point is on the top-front surface of the box. The spatula manipulation point is on the bottom-front surface of the box. Even though the spatula manipulation point is correct, I consider the suggestion for pull operation a failure. If we look at the middle picture, we can see that the new system, containing more knowledge, suggested correctly two manipulation points and their force vectors. Both operations would achieve the goal, including the pull operation. even though the push operation would fail, based on the previously stated evaluation criteria, the new system passes.

5.2.7 Target Scene Box In Corner 2

In the scene, shown in Figure 5.6, you can see one flat box in the corner.

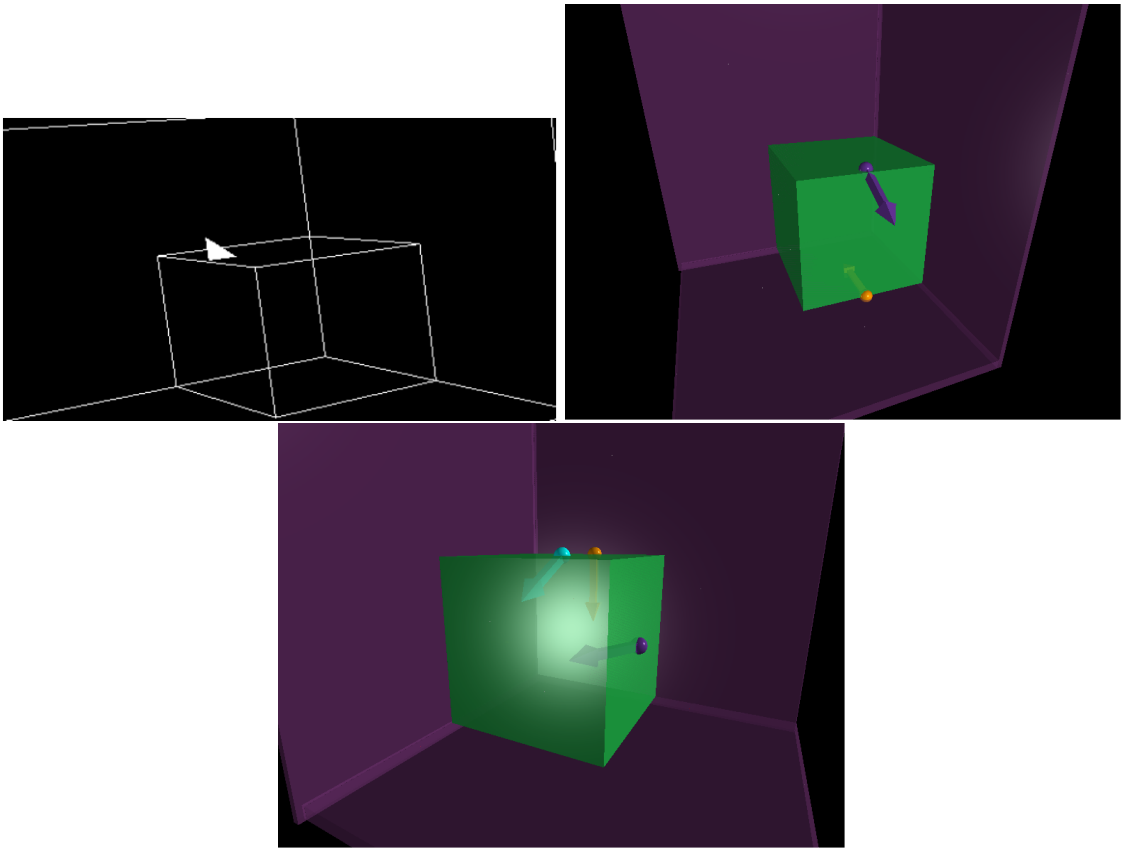

Figure 5.5: Box in a corner. Left- old system. Right - new system with single piece of knowledge. Middle - new system with knowledge about all scenes except this one.

- **Goal**: The goal is to move the box such that it exposes the bottom surface. This would allow someone to pick up the box.
- **Ideal manipulation point for the pull operation:** Front or left surface.
- **Result for the old system:** Pass
- **Result for the new system with a single piece of knowledge:** Fail
- **Result for the new system with improved knowledge:** Pass

On the left side, you can see the suggested manipulation point by the old system. It is on the edge of the front-left surface. I consider this suggestion to be correct. On the right side, there are suggested two manipulation points with their force vectors. The pull manipulation point is on the left-front surface of the box. The spatula manipulation point is on the right-front surface of the box. Even though the spatula manipulation point is correct, I consider the suggestion for pull operation a failure. If we look at the middle picture, we can see that the new system, containing more knowledge, suggested correctly all three manipulation points and their force vectors. All three operations would achieve the goal, including the pull operation. Therefore based on the previously stated evaluation criteria, the new system passes.

5.2.8 Target Scene Box In Corner 3

In the scene, shown in Figure 5.7, you can see one flat box standing in the corner.

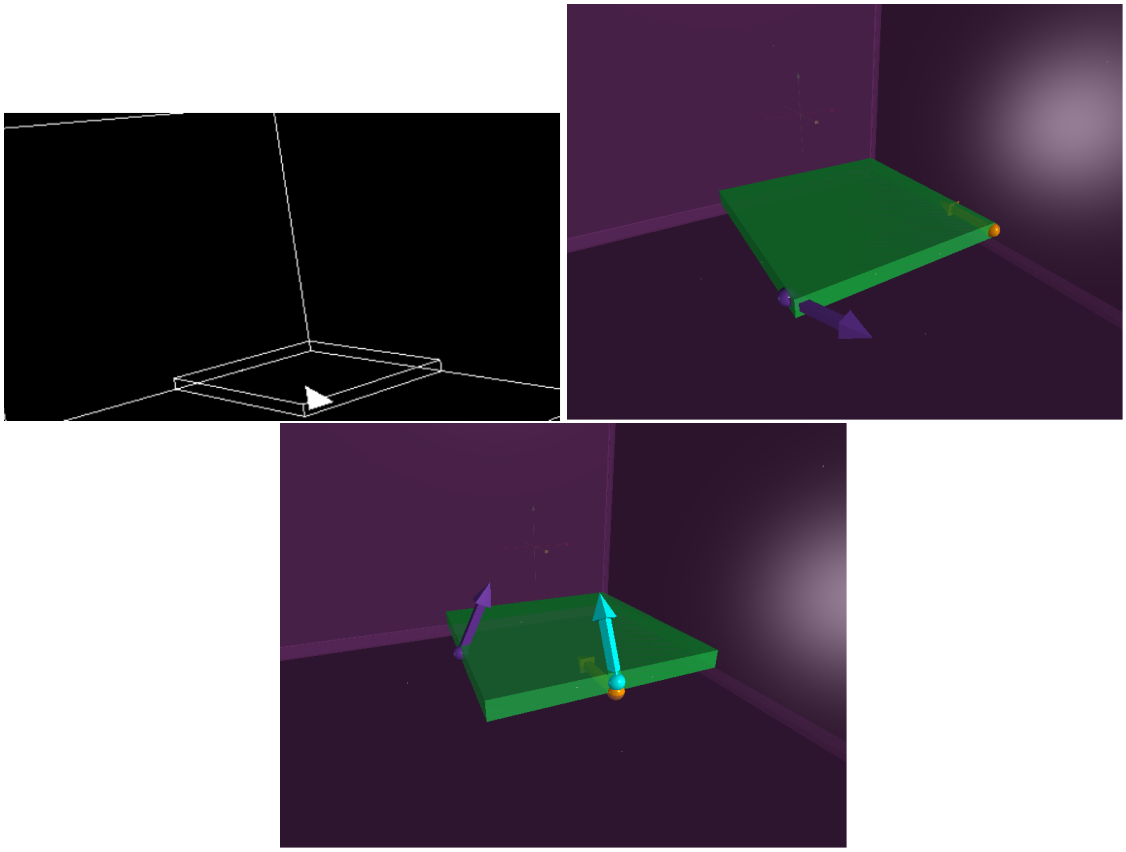

Figure 5.6: Box in a corner 2. Left- old system. Right - new system with single piece of knowledge. Middle - new system with knowledge about all scenes except this one.

- **Goal**: The goal is to move the box such that it exposes the right surface. This would allow someone to pick up the box.
- **Ideal manipulation point for the pull operation:** Top or front surface.
- **Result for the old system:** Pass
- **Result for the new system with a single piece of knowledge:** Fail
- **Result for the new system with improved knowledge:** Pass

On the left side, you can see the suggested manipulation point by the old system. It is on the edge of the top-front surface. I consider this suggestion to be correct. On the right side, there are suggested two manipulation points with their force vectors. The pull manipulation point is on the top-front surface of the box. The spatula manipulation point is on the bottom-front surface of the box. Even though the pull manipulation point is correct, the force vector is not. I consider the suggestion for pull operation a failure. If we look at the middle picture, we can see that the new system, containing more knowledge, suggested correctly all three manipulation points and their force vectors. All three operations would achieve the goal, including the pull operation. Therefore based on the previously stated evaluation criteria, the new system passes.

5.2.9 Target Scene Bread

In the scene, shown in Figure 5.8, you can see slices of bread in a horizontal bread bin.

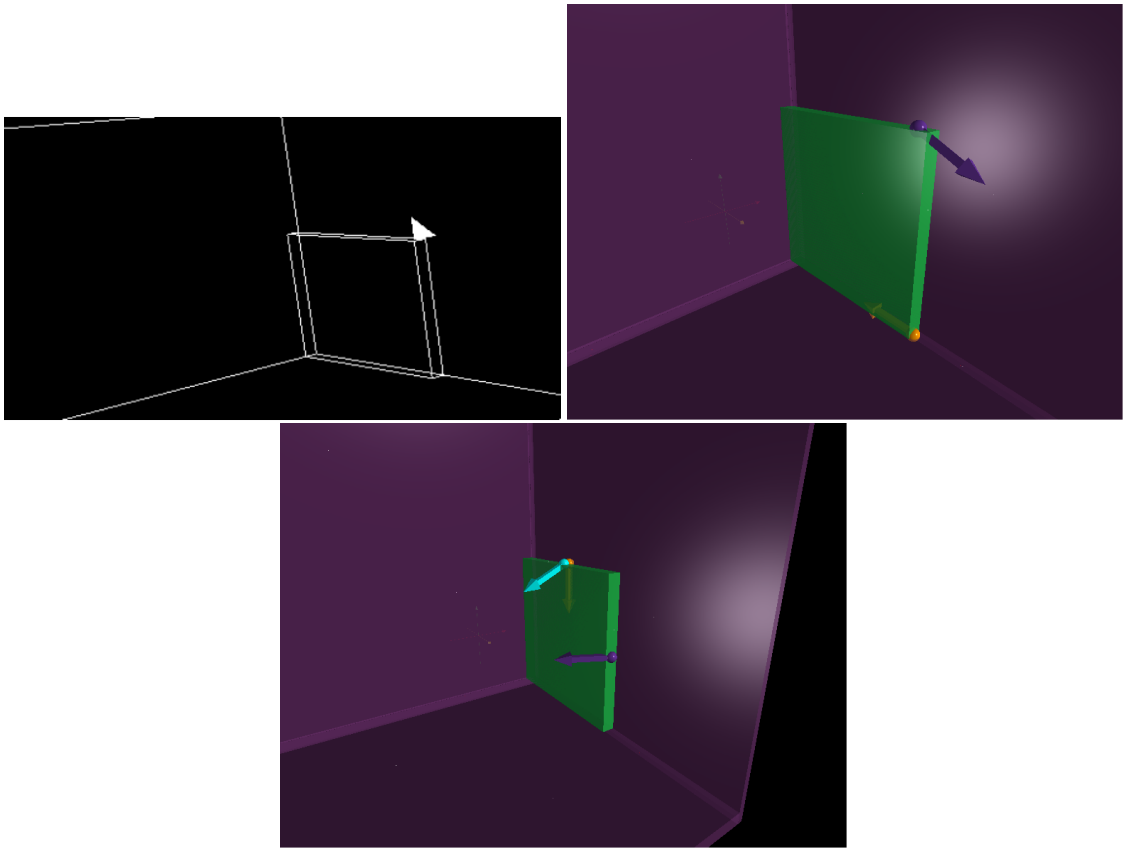

Figure 5.7: Box in a corner 3. Left- old system. Right - new system with single piece of knowledge. Middle - new system with knowledge about all scenes except this one.

- **Goal**: The goal is to move the middle slice such that it exposes the right and left surfaces. This would allow someone to pick up the slice.
- **Ideal manipulation point for the pull operation:** Top or front surface.
- **Result for the old system:** Pass
- **Result for the new system with a single piece of knowledge:** Pass
- **Result for the new system with improved knowledge:** Pass

On the left side, you can see the suggested manipulation point by the old system. It is on the edge of the top-front surface. I consider this suggestion to be correct. On the right side, there are suggested two manipulation points with their force vectors. The pull manipulation point is on the top-front surface of the slice. The spatula manipulation point is on the bottom-front surface of the slice. I consider the suggestion for pull operation correct. If we look at the middle picture, we can see that the new system, containing more knowledge, suggested correctly both manipulation points and their force vectors. Both operations would achieve the goal, including the pull operation. Therefore based on the previously stated evaluation criteria, the new system passes.

5.2.10 Target Scene Bread 2

In the scene, shown in Figure 5.9, you can see slices of bread in a corner of a bread bin.

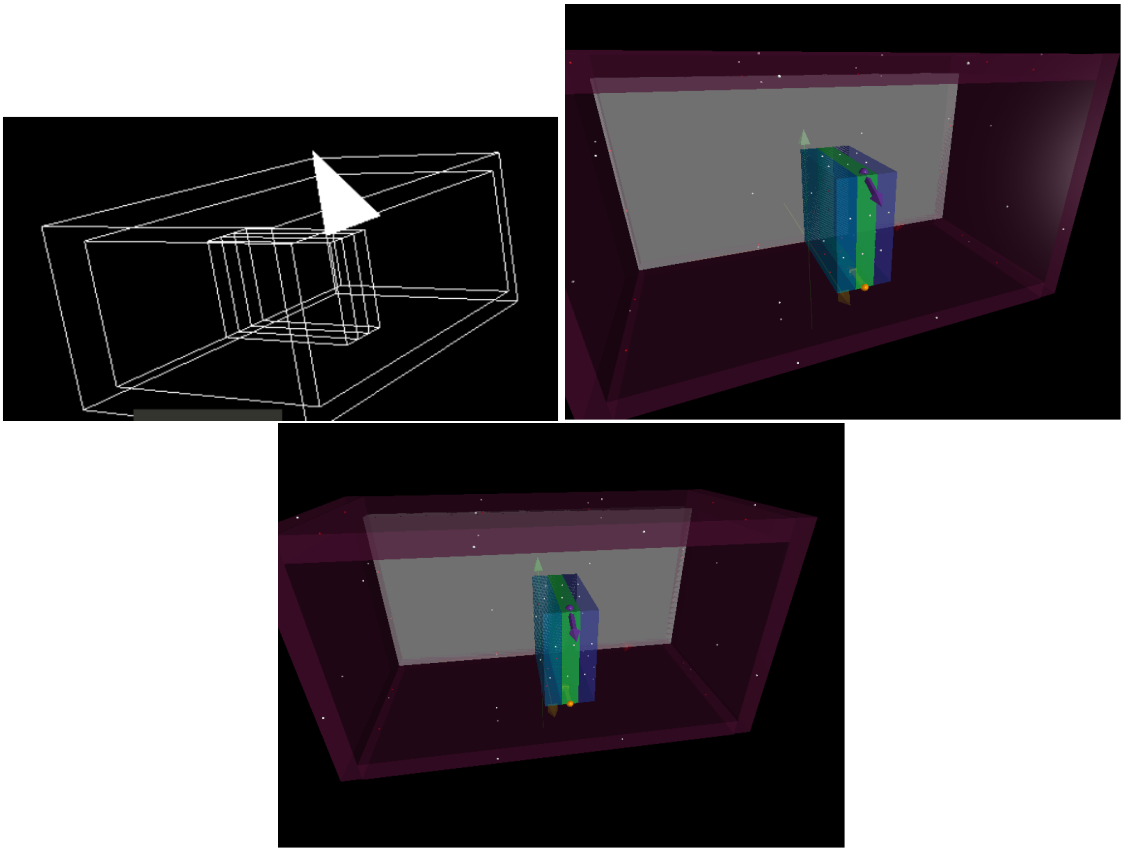

Figure 5.8: Bread. Left- old system. Right - new system with single piece of knowledge. Middle - new system with knowledge about all scenes except this one.

- **Goal**: The goal is to move the top slice such that it exposes the bottom surface. This would allow someone to pick up the slice.
- **Ideal manipulation point for the pull operation:** Front or right surface.
- **Result for the old system:** Pass
- **Result for the new system with a single piece of knowledge:** Pass
- **Result for the new system with improved knowledge:** Pass

On the left side, you can see the suggested manipulation point by the old system. It is on the edge of the front-right surface. I consider this suggestion to be correct. On the right side, there are suggested two manipulation points with their force vectors. The pull manipulation point is on the right-front surface of the slice. The spatula manipulation point is on the left-front surface of the slice. I consider the suggestion for pull operation correct. If we look at the middle picture, we can see that the new system, containing more knowledge, suggested correctly all three manipulation points and their force vectors. All three operations would achieve the goal, including the pull operation. Therefore based on the previously stated evaluation criteria, the new system passes.

5.2.11 Target Scene Cans On A Shelf

In the scene, shown in Figure 5.10, you can see three cans on the shelf.

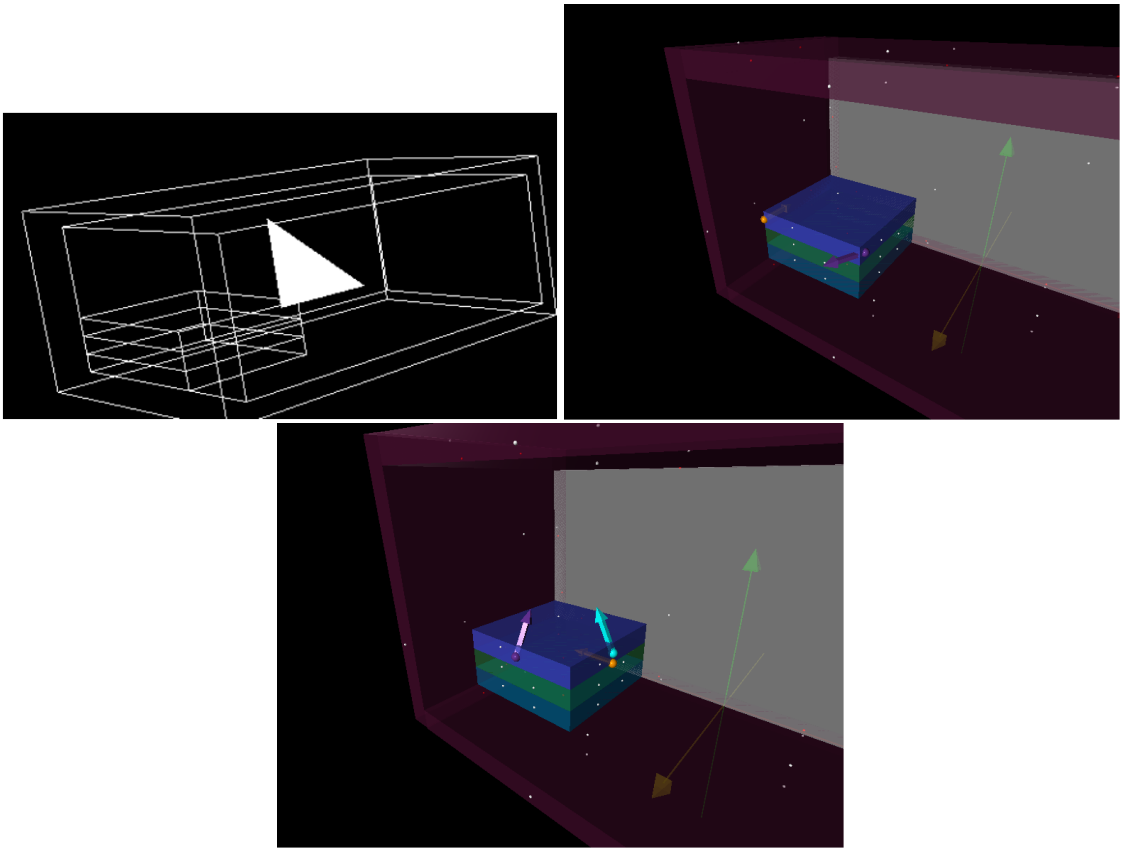

Figure 5.9: Bread 2. Left- old system. Right - new system with single piece of knowledge. Middle - new system with knowledge about all scenes except this one.

- **Goal**: The goal is to move the middle can such that it exposes left and right surfaces. This would allow someone to pick up the can.
- **Ideal manipulation point for the pull operation:** Back or top surface.
- **Result for the old system:** Pass
- **Result for the new system with a single piece of knowledge:** Pass
- **Result for the new system with improved knowledge:** Pass

On the left side, you can see the suggested manipulation point by the old system. It is on the edge of the top-front surface. I consider this suggestion to be correct. On the right side, there are suggested two manipulation points with their force vectors. The pull manipulation point is on the top-front surface of the can. The spatula manipulation point is on the bottom-front surface of the can. I consider the suggestion for pull operation correct. If we look at the middle picture, we can see that the new system, containing more knowledge, suggested correctly all three manipulation points and their force vectors. All three operations would achieve the goal, including the pull operation. Therefore based on the previously stated evaluation criteria, the new system passes.

It is important to mention that in this scene the cans are not touching the wall. Hence, the back surface is available for manipulation. It is possible to notice that the new system that has an extended knowledge takes advantage of this fact and apply the pull manipulation point and its force vector in the middle of the back surface.

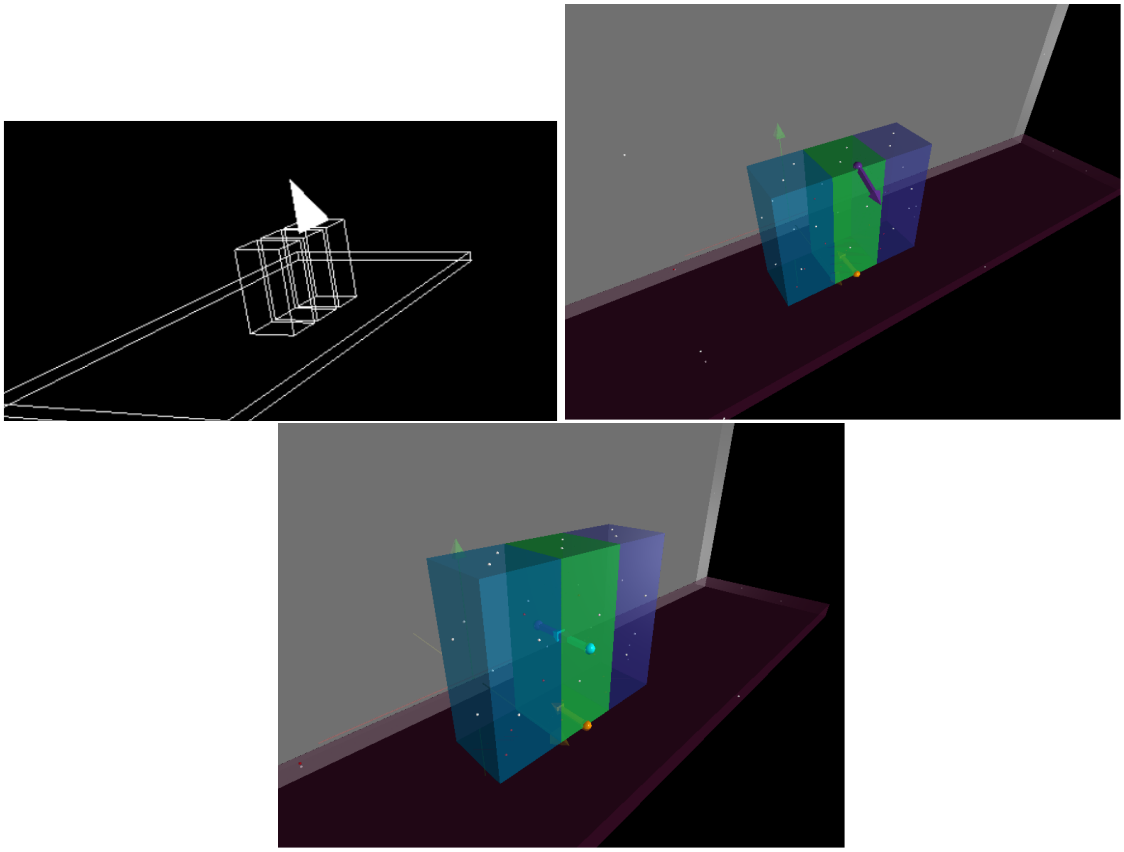

Figure 5.10: Cans on a shelf. Left- old system. Right - new system with single piece of knowledge. Middle - new system with knowledge about all scenes except this one.

5.2.12 Target Scene Cans On A Shelf 2

In the scene, shown in Figure 5.11, you can see 9 cans on the shelf.

- **Goal**: The goal is to move the top-middle can such that it exposes left and right surfaces. This would allow someone to pick up the can.
- **Ideal manipulation point for the pull operation:** Back or top surface.
- **Result for the old system:** Pass
- **Result for the new system with a single piece of knowledge:** Pass
- **Result for the new system with improved knowledge:** Pass

On the top-left side, you can see the suggested manipulation point by the old system. It is on the edge of the top-front surface. I consider this suggestion to be correct. On the top-right side, there are suggested two manipulation points with their force vectors. The pull manipulation point is on the top-front surface of the can. The spatula manipulation point is on the bottom-front surface of the can. I consider the suggestion for pull operation correct. If we look at the bottom-left picture, we can see that the new system, containing more knowledge, suggested correctly all three manipulation points and their force vectors. All three operations would achieve the goal, including the pull operation. Therefore based on the previously stated evaluation criteria, the new system passes.

I also experimented with editing the scene and moving all cans towards the wall. This caused the back surface to be not exposed and not available for manipulation. Also, I changed the shape of cans from cuboid to cylinder. This is not possible to see in the final picture because the system uses AABB for analogy. However, if you open the scene `scenes/cans-shelf-5.obj`, you can see that there were used cylindrical cans. Then, I tried the scene with the new system with extended knowledge. It correctly suggested both pull and spatula manipulation points and their force vectors. See the bottom-right picture in Figure 5.11.

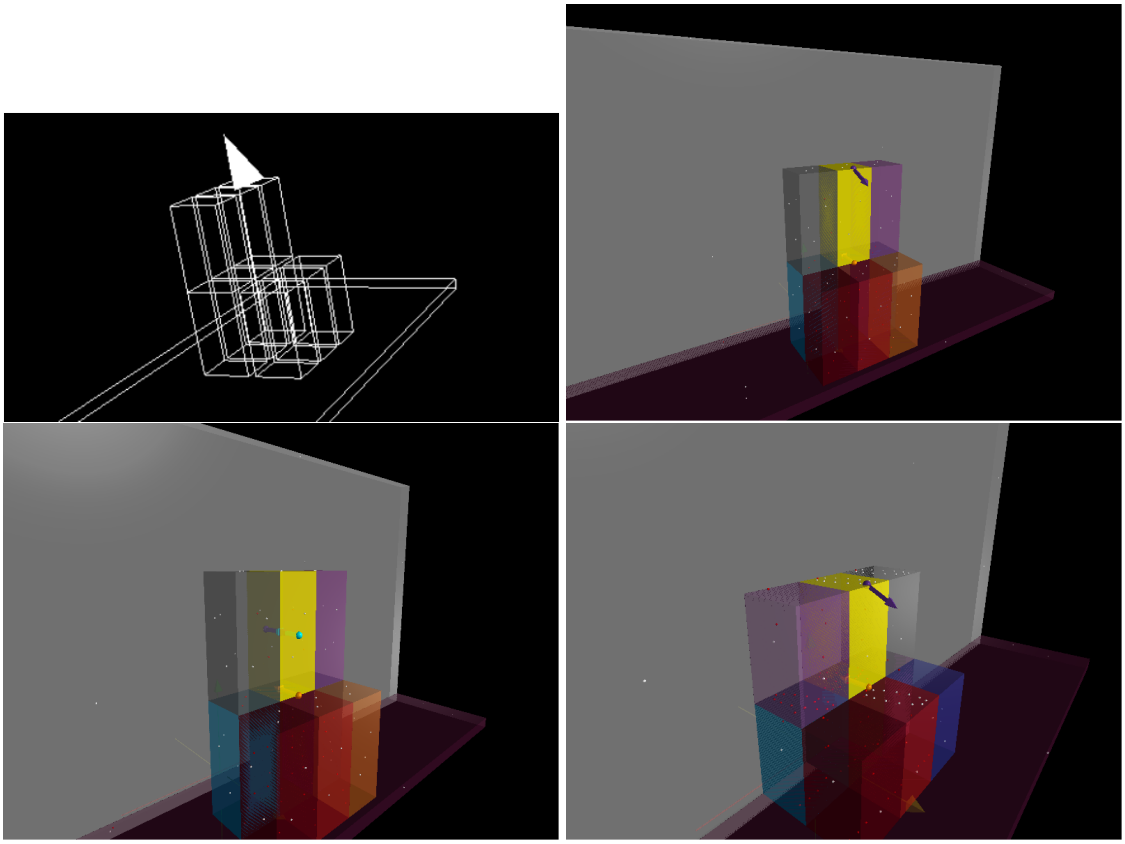

Figure 5.11: Cans on a shelf 2. Top Left- old system. Top Right - new system with single piece of knowledge. Bottom Left - new system with knowledge about all scenes except this one. Bottom Right - new system with knowledge about all scenes except this one, scene is corrected.

5.2.13 Target Scene Pizza In A Freezer

In the scene, shown in Figure 5.12, you can see three pizza boxes in the freezer.

- **Goal**: The goal is to move the middle box such that it exposes the bottom surface. This would allow someone to pick up the box.
- **Ideal manipulation point for the pull operation:** Front surface.
- **Result for the old system:** Pass
- **Result for the new system with a single piece of knowledge:** Pass
- **Result for the new system with improved knowledge:** Pass

On the left side, you can see the suggested manipulation point by the old system. It is on the edge of the top-front surface. I consider this suggestion to be correct. On the right side, there are suggested two manipulation points with their force vectors. The pull manipulation point is on the front-middle surface of the box. The spatula manipulation point is on the back-middle surface of the box. I consider the suggestion for both operations a success. If we look at the middle picture, we can see that the new system, containing more knowledge, suggested correctly both manipulation points and their force vectors. Both operations would achieve the goal, including the pull operation. Therefore based on the previously stated evaluation criteria, the new system passes.

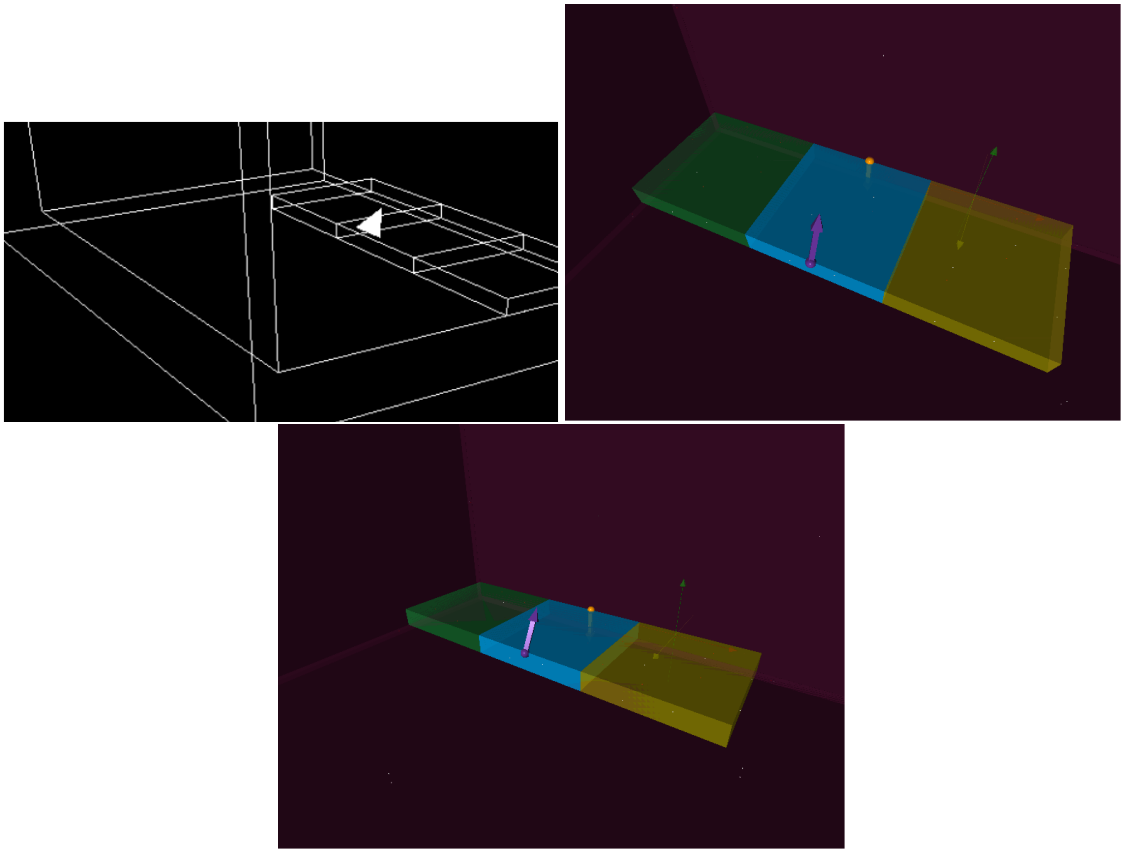

Figure 5.12: Pizza in a freezer. Left- old system. Right - new system with single piece of knowledge. Middle - new system with knowledge about all scenes except this one.

5.2.14 Target Scene Pizza In A Freezer 2

In the scene, shown in Figure 5.13, you can see three pizza boxes in the freezer.

- **Goal**: The goal is to move the top box such that it exposes the bottom surface. This would allow someone to pick up the box.
- **Ideal manipulation point for the pull operation:** Front or right surface.
- **Result for the old system:** Pass
- **Result for the new system with a single piece of knowledge:** Pass
- **Result for the new system with improved knowledge:** Pass

On the left side, you can see the suggested manipulation point by the old system. It is on the edge of the right-front surface. I consider this suggestion to be correct. On the right side, there are suggested two manipulation points with their force vectors. The pull manipulation point is on the right-front surface of the box. The spatula manipulation point is on the left-front surface of the box. I consider the suggestion for both operations a success. If we look at the middle picture, we can see that the new system, containing more knowledge, suggested correctly all manipulation points and their force vectors. All operations would achieve the goal, including the pull operation. Therefore based on the previously stated evaluation criteria, the new system passes.

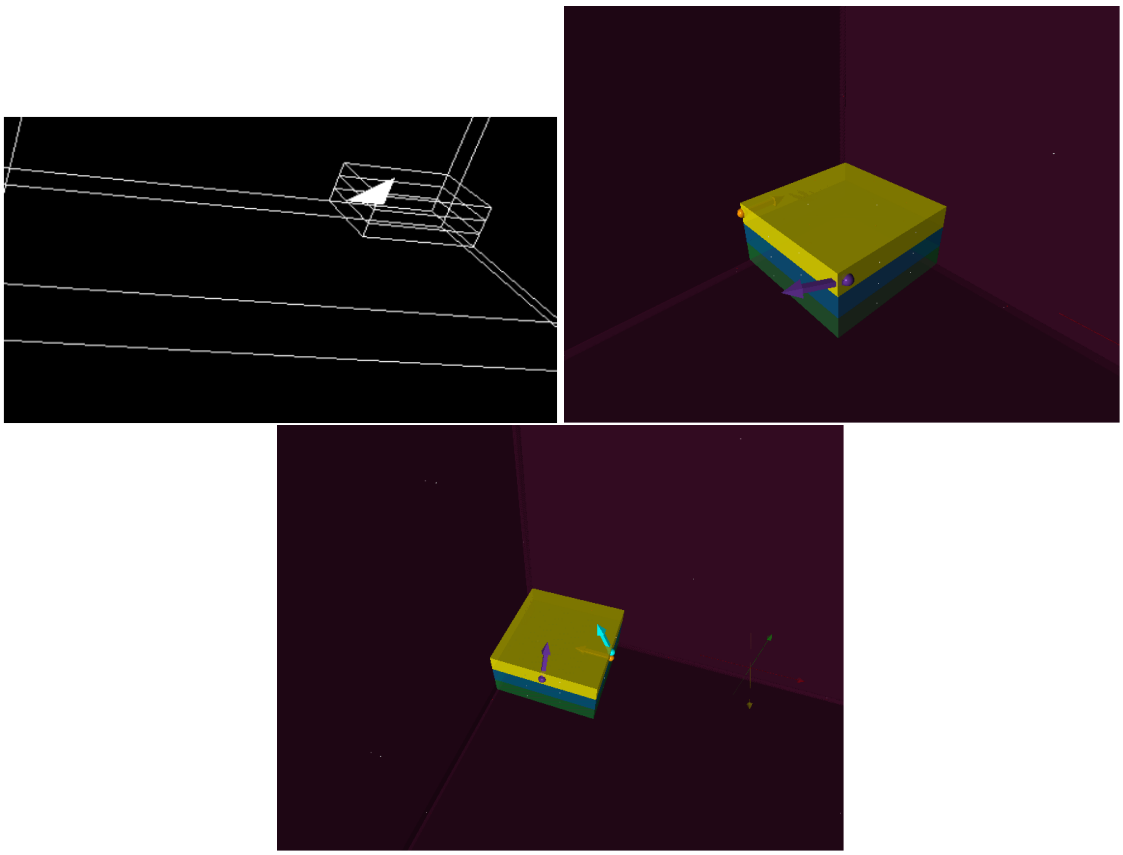

Figure 5.13: Pizza in a freezer 2. Left- old system. Right - new system with single piece of knowledge. Middle - new system with knowledge about all scenes except this one.

5.2.15 Results

It is important to summarise and discuss the collected results. As we could see, the old system is incapable of suggesting force vectors. Therefore, I evaluated only the correctness of the suggested manipulation point for pull operation. The results were that it suggested 9 manipulation points correctly and 3 were incorrect.

The new system is capable of suggesting also force vectors for manipulation points. In addition, it can evaluate all scenes in the knowledge base and pick the best one based on the highest score. The new system with a single piece of knowledge about the scene *Books on a shelf*. It correctly suggested only 7 pull operation. Then, there were 5 incorrect suggestions. These were 2 cases where the manipulation point was correct but the force vector was not. If I compared only the manipulation points and did not consider the force vectors, the result would be the same as the

old system.

At last the new system with extended knowledge about all scenes except the one that is being evaluated. In this case, we can see a big improvement. It suggested manipulation points with their vectors 11 times successfully and only 1 time incorrectly. Moreover, the suggestions were often more sensible based on my personal opinion. We can see the big difference between the new system with a single piece of knowledge and if the system has extended knowledge. It is incredible how this improved performance. However, I think that it is not possible to confirm this claim without a bigger study containing more samples. The Table 5.1 summarise the result of this evaluation.

Table 5.1: Evaluation results summary

Scene name	Old System	New System	New system - extended knowledge
Books on a shelf 2	Fail	Fail	Fail
Boxes on a shelf	Fail	Fail	Pass
Boxes on a shelf 2	Pass	Pass	Pass
Box in a corner	Fail	Fail	Pass
Box in a corner 2	Pass	Fail	Pass
Box in a corner 3	Pass	Fail	Pass
Bread	Pass	Pass	Pass
Bread 2	Pass	Pass	Pass
Cans on a shelf	Pass	Pass	Pass
Cans on a shelf 2	Pass	Pass	Pass
Pizza in a freezer	Pass	Pass	Pass
Pizza in a freezer 2	Pass	Pass	Pass
SUM Fails	3	5	1
SUM Passes	9	7	11

5.3 Other Improvements In The New System

There are several things that were improved in the new system in comparison to the old one. However, these improvements were not suitable for any kind of evaluation. Many of the improvements were already mentioned several times. However, here I would like to summarise all of them in one place.

These are improvements to the new system:

1. A new scene can be created within a few minutes. Usually, it took me about 3 - 5 minutes. With the old system, the author mentioned that creating new scenes were very time demanding and usually it took about 30 minutes up to 1 hour.

2. The new system has a modular design and the core algorithms and data structures are packaged in a separate package that can be easily reused in another system. In the old system, any portion of the code could not be reused. The code was written with a very low quality and highly coupled.

3. The new system automatically detects unavailable surfaces for manipulation. In the old system, this had to be manually specified during the process of creating a new scene.

4. Scenes are using standard and widely spread 3D file format. Therefore, it is possible to view the scenes without the need for specialised software. This was not possible with the old system. Scenes were created as Python classes.

5. The new system stores knowledge in standardised file format as SQLite DB file. Hence, it is easily accessible for other systems or analysis. The old system was incapable of storing knowledge.

6. The new system includes visual controls for displaying the scene. The user can rotate, zoom in, zoom out and walk through the scene. This brings a better user experience. In the old system, the user could not interact with the scene at all. It was only displayed as a static picture even though it was using OpenGL.

5.4 Summary

This chapter showed how the system was tested during the whole time of development. Furthermore, it also explained the hardware, software, tools and process used for the qualitative evaluation. Comparing only the suggested manipulation points with a single piece of knowledge in the database, the new system would perform the same as the old one. However, considering the performance of the new system with extended knowledge, it performs better than the old one. It shows that the chosen approach was a good decision. However, these results are not statistically significant because the old system was tested only on 12 scenes. In the end, all results are presented, explained and discussed.

Chapter 6

Conclusions, Discussion & Future Work

This chapter concludes the work of this project and discusses the developed system. In addition, it suggests the possible future work and improvements that can be done.

6.1 Conclusions

Overall, the developed system fulfils all the functional and non-functional requirements specified in Chapter 3. In addition, the core algorithms (analogy, collision detection, scene representation, knowledge retrieval, etc.) used in the system are separated as a Python package “analogy" that can be used in any other system. An interesting fact about this system is that the approach of using a knowledge base improved the performance of the system. Having only a few (17) examples in the knowledge base it was capable of suggesting the ideal manipulation points and their vectors in all scenes except a single one. In addition, it provides several solutions that are ordered based on numerical score and these may be used in the case that the first suggestion fails.

The most important goal of this project was to create a tool that could serve and accelerate research in robot object manipulation. I am convinced that this goal was fully achieved. In comparison to the previous system, the required time to draw and test an analogy in a new scene was minimally reduced by 10 times. The existing analogy algorithm seems to perform well but it would require to conduct a bigger study (with more examples) to confirm this conclusion. It is important to realise that the provided analogy algorithm is definitely not perfect or even complete. Hence, there is the ability to easily change the underlying analogy algorithm for your own. Moreover, I believe that using machine learning would improve the performance of the existing algorithm.

6.2 Discussion

From the beginning, this project was very ambitious and time-demanding. There were several aspects that made the project difficult. First of all, there is almost non-existent research in robot skill and knowledge adaptation using an analogy. As we could see in Chapter 2 all the existing research is mainly focused on a very specific domain or it is applied on invented tasks such as string manipulations. As a second aspect that made this project difficult was the topic on its own. It is an interdisciplinary topic that required a broad set of skills and knowledge that I had to gain during this time period. Some of them are a theory of analogy from the psychological point of view, 3D modelling and game development.

As a reflection on the development process, I have to say that I spent a substantial amount of

time by trying several prototypes. I encountered many dead ends. However, since I adopted the exploratory programming technique, this allowed me to produce better software.

In the end, I am very glad that I worked on this project. It allowed me to increase my spectrum of knowledge. I am hoping that this research tool will help some other people to test their ideas more quickly and efficiently. Also, I am hoping that there will be a small community that starts contributing to this project. Since I publicly released the project on my GitHub repository, there was a unique clone of this repository every day (only 3 until the date of writing).

6.3 Future Work

There are several things that can be added or improved in this system. Unfortunately, I was not able to do so because I had only a limited time period that I could spend on this project. These are the most important features that I think should be implemented in the future.

1. Create more scenes and let other people fill the knowledge base with ideal manipulation points and vectors for several manipulation operations. Then using this data and machine learning technique we could find optimal values for weights shown in Figure 4.15 on line number two. Currently, these values were initialised by myself and they are most likely not optimal. I would suggest using Normal equation[1] or Regression[2]. It is important to note that linear models may not yield the best results. There should be a proper evaluation using cross-validation. The mentioned machine learning techniques are only suggestions and there should be evaluated also different methods.

2. Change the existing code in `analogy.py` file such that it shifts the suggested manipulation points and their vectors from the surface of AABB to the actual mesh surface. This can be achieved easily by searching for the closest surface on the mesh from the suggested manipulation point.

3. Using real-world pictures that went through an image segmentation process. This may reveal some drawbacks of the current implementation. If this approach works the system could be used in a pipeline for a robot that is trying to solve a manipulation task. I believe that the ideal point in the pipeline for this system would be between image segmentation and physics simulation process. The system could suggest the ideal manipulation point and then the physics simulation would evaluate the suggestion. If it fails it could try the second suggestion etc.

These are only a few ideas or future development. However, if there is going to be at least a small community of contributors to this project in the future, they may have different ideas and requirements.

[1] https://en.wikipedia.org/wiki/Linear_least_squares#Derivation_of_the_normal_equations
[2] https://en.wikipedia.org/wiki/Regression_toward_the_mean

Appendices

Appendix A

Extras

```
def get_mappings_score(self, target_aabb, source_aabb):
    """
    Returns analogy scores for all mappings of two AABB objects.

    Args:
        target_aabb(AABB): The target AABB that we want to map to source AABB.
        source_aabb(AABB): The source AABB that from source scene.

    Returns:
        Analogy scores for all mappings (sequence permutations) of two AABB
        objects based on collided sides similarity, ratio of AABBs,
        matching surfaces.
    """
    # collision match weights
    exact_collision_weight = 0.99
    partial_collision_weight = 0.5
    no_collision_match_weight = .0
    # surface match weights
    top_match_weight = .02
    bottom_match_weight = .02
    front_match_weight = .01
    back_match_weight = .01
    right_match_weight = .01
    left_match_weight = .01
    no_surf_match_weight = .0
    # surface ratio difference penalties
    xy_ratio_weight = 0.1
    zy_ratio_weight = 0.1
    xz_ratio_weight = 0.1

    mappings_score = []
    for perm_sequence in self.all_permutations.keys():
        score = 1.0  # default score
        for surface in self.all_permutations[perm_sequence].keys():
            # scores for exact collision/partial/no-collision match
            if target_aabb.collided_sides[
                    surface] == source_aabb.collided_sides[
                        self.all_permutations[perm_sequence][surface]]:
                score += 1.0 * exact_collision_weight
```

```
            # match partially collided sides and fully collided sides
            elif target_aabb.collided_sides[
                    surface] != 0 and source_aabb.collided_sides[
                        self.all_permutations[perm_sequence][surface]] != 0:
                score += 1.0 * partial_collision_weight
            else:  # there is no match
                score += 1.0 * no_collision_match_weight

            # score for surface match
            if (surface == 'top' and
                    self.all_permutations[perm_sequence][surface] == 'top'):
                score += 1.0 * top_match_weight
            elif (surface == 'bottom' and
                  self.all_permutations[perm_sequence][surface] == 'bottom'
                 ):
                score += 1.0 * bottom_match_weight
            elif (surface == 'front' and
                  self.all_permutations[perm_sequence][surface] == 'front'):
                score += 1.0 * front_match_weight
            elif (surface == 'back' and
                  self.all_permutations[perm_sequence][surface] == 'back'):
                score += 1.0 * back_match_weight
            elif (surface == 'right' and
                  self.all_permutations[perm_sequence][surface] == 'right'):
                score += 1.0 * right_match_weight
            elif (surface == 'left' and
                  self.all_permutations[perm_sequence][surface] == 'left'):
                score += 1.0 * left_match_weight
            else:
                score += 1.0 * no_surf_match_weight
        # create vpython vec for xyz half size of the target object
        vpython_vec_xyz = vpython.vec(target_aabb.half_size[0],
                                      target_aabb.half_size[1],
                                      target_aabb.half_size[2])
        # rotate it based on rotation sequence
        for rotation in reversed(perm_sequence):
            if rotation == 'x':
                vpython_vec_xyz = vpython.rotate(
                    vpython_vec_xyz,
                    angle=vpython.radians(-90),
                    axis=vpython.vec(1, 0, 0))
            elif rotation == 'y':
                vpython_vec_xyz = vpython.rotate(
                    vpython_vec_xyz,
                    angle=vpython.radians(-90),
                    axis=vpython.vec(0, 1, 0))
        # compare xy, zy, xz ratios of target and source aabbs after rotation
        # and penalise for difference
        xy_ratio_diff = abs(
            (abs(vpython_vec_xyz.x) / abs(vpython_vec_xyz.y)) -
            (source_aabb.half_size[0] / source_aabb.half_size[1]))
        xy_ratio_diff *= xy_ratio_weight
```

```
92          zy_ratio_diff = abs(
93              (abs(vpython_vec_xyz.z) / abs(vpython_vec_xyz.y)) -
94              (source_aabb.half_size[2] / source_aabb.half_size[1]))
95          zy_ratio_diff *= zy_ratio_weight
96          xz_ratio_diff = abs(
97              (abs(vpython_vec_xyz.x) / abs(vpython_vec_xyz.z)) -
98              (source_aabb.half_size[0] / source_aabb.half_size[2]))
99          xz_ratio_diff *= xz_ratio_weight
100         sum_ratio_diff = xy_ratio_diff + zy_ratio_diff + xz_ratio_diff
101
102         # penalise for ratio difference
103         score -= sum_ratio_diff
104
105         mappings_score.append((score, perm_sequence))
106     return mappings_score
```

Listing A.1: Scoring algorithm for all 24 mappings from target to source object.

Appendix B

User Manual

System for research of analogy in robot object manipulation. For simplification, it is using axis-aligned bounding boxes.

- Design a scene in Microsoft 3D builder[1]
- Teach the system what are the ideal manipulation points and force vectors for specific tasks.
- Use the knowledge to solve the manipulation tasks in unseen situations using an analogy.

B.1 How to use the system

Assuming the installation is done. As a first thing activate venv with the following command.

```
source venv/bin/activate
```

Then you can run the system. There are two options. The first option is to add new knowledge to the knowledge base database. The second option is to solve or ask the system what are the best manipulation points and force vectors for a target object based on the current knowledge. You can have multiple different databases with different knowledge.

B.2 How to add new knowledge

This is a more general example of how to use the command.

```
./analogy.py add scene_name.obj knowledge_base.db
```

This is a specific example that will work immediately.

```
./analogy.py add scene_name.obj knowledge_base.db
```

If a file with the name of the knowledge base database does not exist, it will be created. If it exists, the knowledge will be added to the existing database.

You will see the scene shown in Figure B.1.

After the scene is opened and visualised in your web browser, select the target object for manipulation (click on it).

How to manipulate with the scene:

- Rotate the camera view: drag with the right mouse button (or Ctrl-drag left button).

[1] https://www.microsoft.com/en-gb/p/3d-builder/9wzdncrfj3t6?activetab=pivot:overviewtab

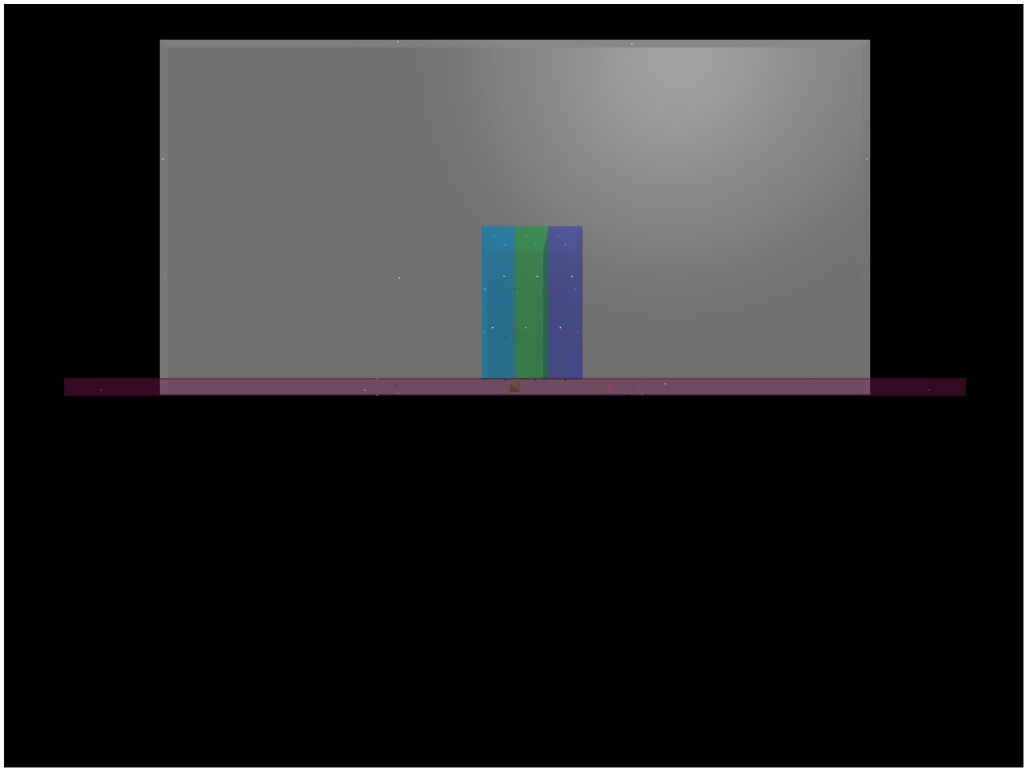

Figure B.1: books-shelf.obj scene without selected target object.

- Zoom: drag with left and right mouse buttons (or Alt/Option-drag or scroll wheel).
- Pan: Shift-drag.
- Touch screen: swipe or two-finger rotate; pinch/extend to zoom.

Figure B.2 shows how the target object changes opacity once it is selected.

Then follow the instructions in the terminal window. You will be asked to add the ideal manipulation points and force vectors for specific tasks. Currently, it supports only 3 tasks. The manipulation tasks are push, pull and using a spatula. In the end, knowledge is saved to the database.

The repository contains a few example scenes that you can find in `scenes` directory.

B.3 How to use the knowledge

This is a more general example of how to use the command.

```
./analogy.py solve scene_name.obj knowledge_base.db
```

```
./analogy.py solve scenes/books-shelf.obj all_scenes.db
```

After the scene is opened and visualised in your web browser, select the target object for manipulation (click on it). How to manipulate with the scene:

- Rotate the camera view: drag with the right mouse button (or Ctrl-drag left button).
- Zoom: drag with left and right mouse buttons (or Alt/Option-drag or scroll wheel).

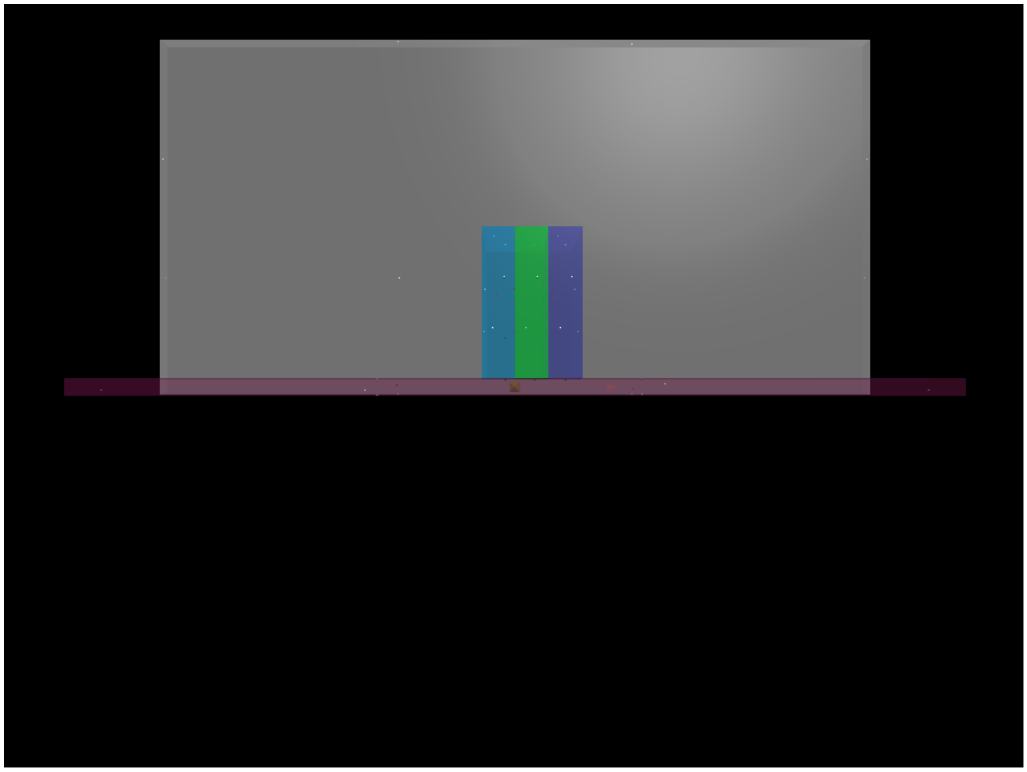

Figure B.2: books-shelf.obj scene with selected target object.

- Pan: Shift-drag.
- Touch screen: swipe or two-finger rotate; pinch/extend to zoom.

For example the middle book on the shelf. Then confirm the selection in the terminal window. You should now see the suggested manipulation points and their force vectors for specific tasks. This is presented in Figure B.3

White spheres are colliders for each axis-aligned bounding box. Based on the colour you can distinguish the manipulation tasks.

- Cyan colour = push
- Purple colour = pull
- Orange colour = using spatula

B.4 Analogy Example

It is interesting to see how the analogy works with limited knowledge. For example, `books-shelf.db` contains only knowledge about how to manipulate the middle book in scene `scenes/books-shelf.obj`. Now, if I want to solve a manipulation problem for a pizza box in a freezer in the scene `scenes/pizza-boxes-freezer.obj`, I just have to run this command.

```
1 ./analogy.py solve scenes/pizza-boxes-freezer.obj books-shelf.db
```

Notice that the last argument is the knowledge base we want to use. See Figure B.4

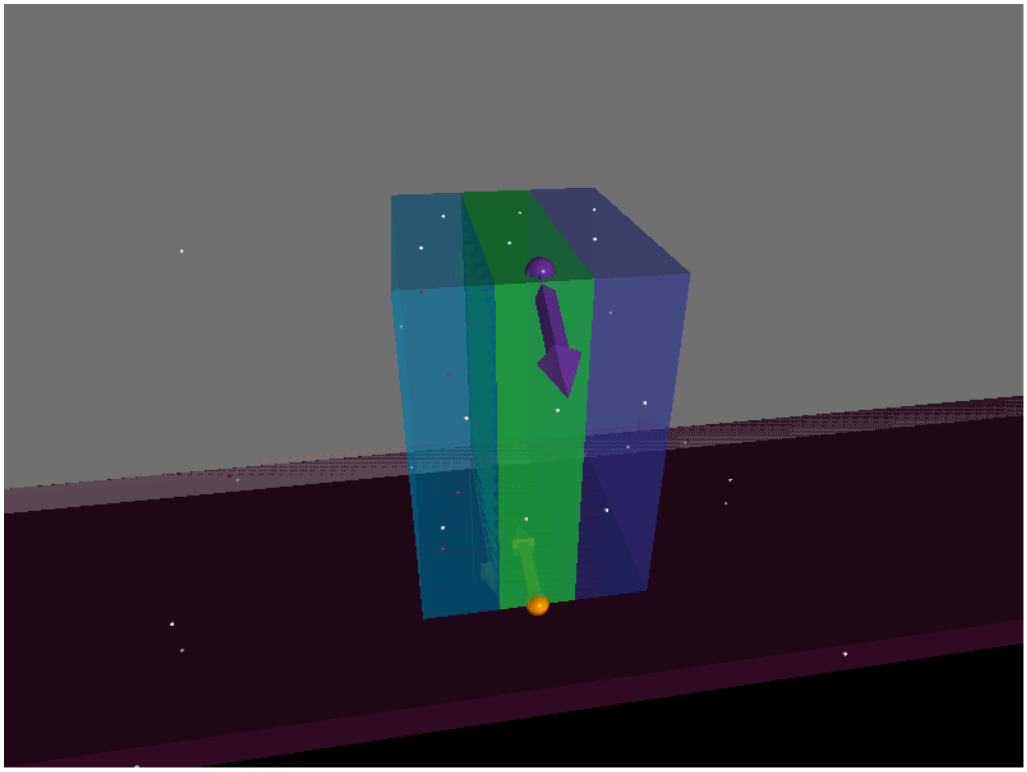

Figure B.3: books-shelf.obj scene with suggested manipulation points and force vectors.

Now you have to rotate the scene so you can select the target object (pizza box in the middle). See Figure B.5

After you confirm the selection in the terminal window you can see suggested points and force vectors. See Figure B.6

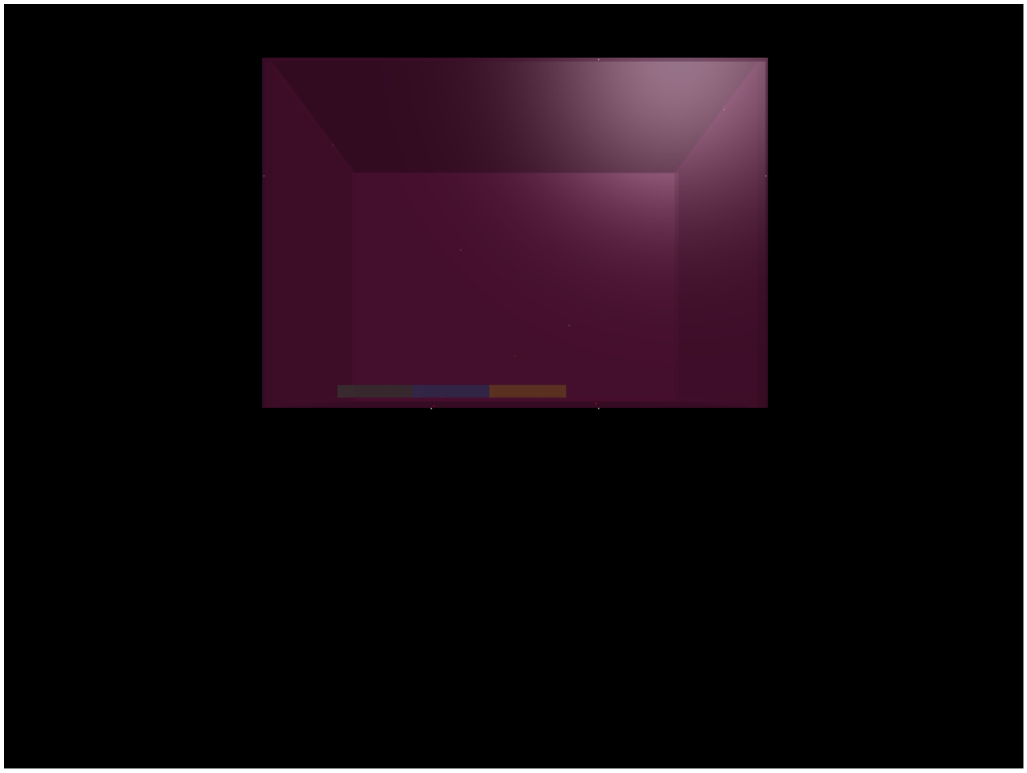

Figure B.4: pizza-freezer.obj scene without selected target object.

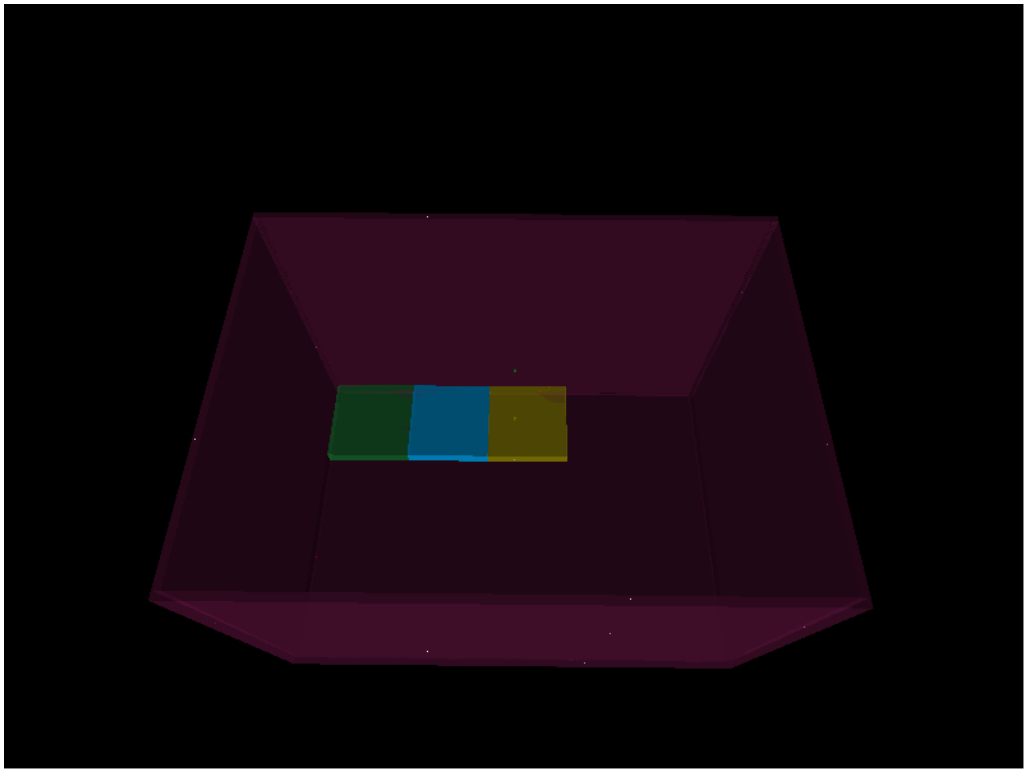

Figure B.5: pizza-freezer-2.obj scene rotated so it allows selection of the target object.

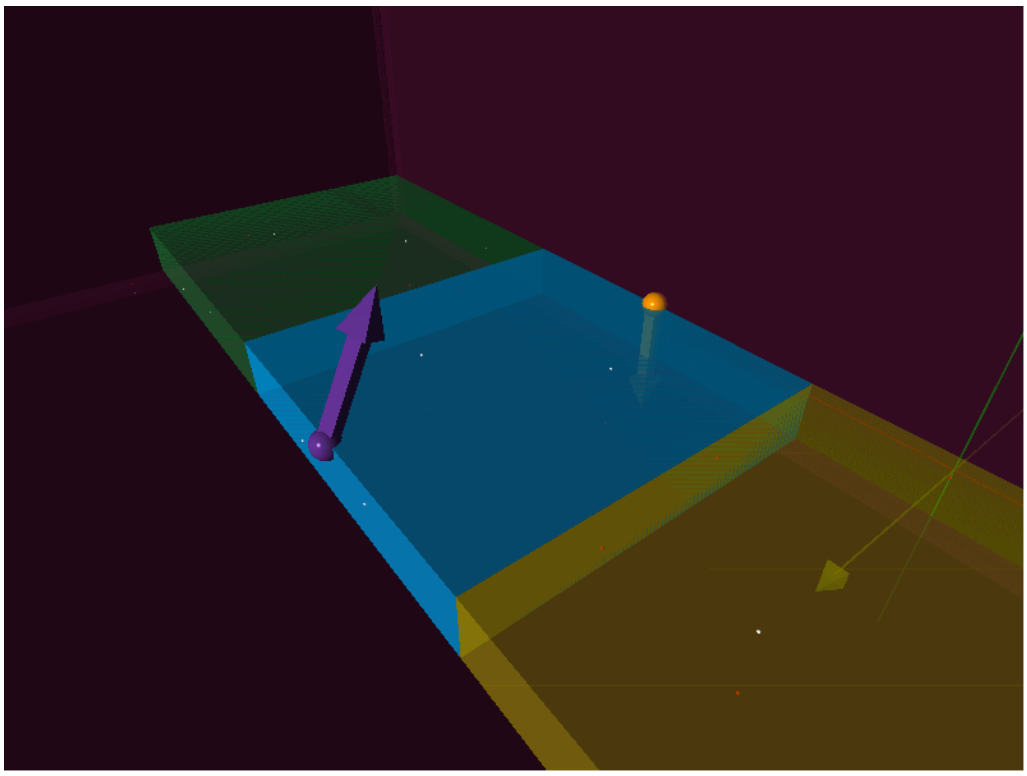

Figure B.6: pizza-freezer-3.obj scene solved using analogy.

Appendix C

Maintenance Manual

To use the developed system, it is necessary to fulfil the following requirements for the system:

1. PC with installed Fedora 29[1] or Ubuntu 18.04[2]. It may be also later versions too.

2. Installed a modern web browser of your choice. For example Firefox[3], Chromium[4] or Chrome[5]

3. Access to the Internet.

C.1 Installation

Open a terminal window and copy and paste the following commands. For Fedora OS follow Listing C.1 and for Ubuntu OS follow Listing C.2.

```
1 sudo dnf install git python3 gcc
2 git clone git@github.com:gandalf15/analogy.git
3 cd analogy
4 make
5 chmod +x analogy.py
```

Listing C.1: Terminal commands for Fedora 29

```
1 sudo apt update
2 sudo apt install git python3 gcc
3 git clone git@github.com:gandalf15/analogy.git
4 cd analogy
5 make
6 chmod +x analogy.py
```

Listing C.2: Terminal commands for Ubuntu 18.04

At this point the installation is done.

[1] https://getfedora.org/
[2] https://www.ubuntu.com/
[3] https://www.mozilla.org/en-US/firefox/
[4] https://www.chromium.org/
[5] https://www.google.co.uk/chrome/

C.2 Software Dependencies

There is only a single software dependency. It is VPython library and it can be downloaded from this URL: https://vpython.org/. However, there is no need to do this because I created a make file that automates the whole process.

C.3 Description Of Files

Figure C.1 shows the directory structure of the project.

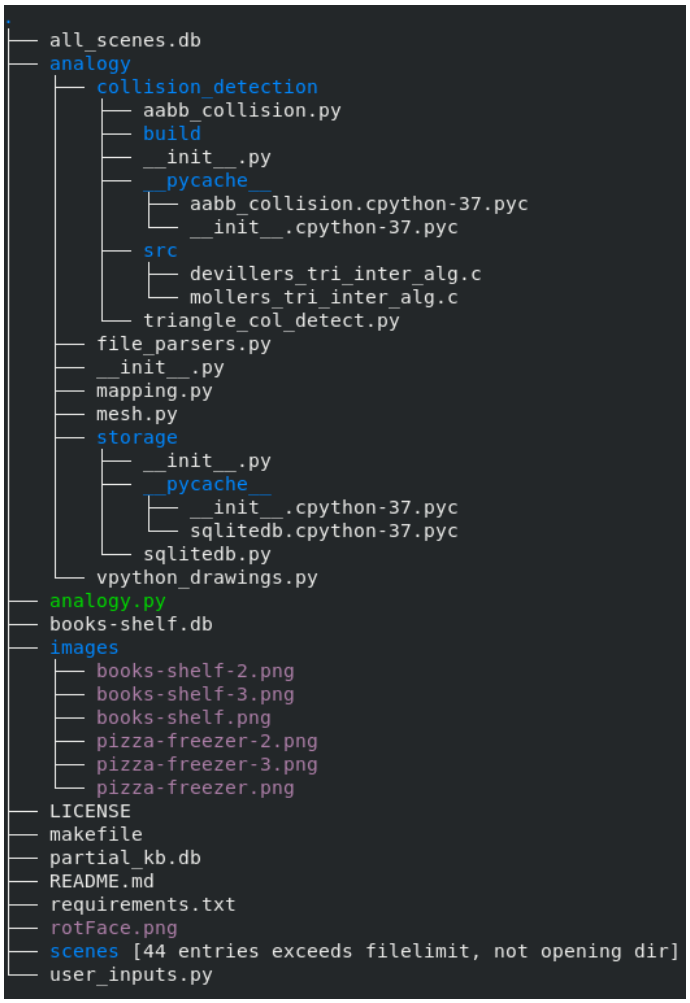

Figure C.1: File and directory structure of the project.

As you can see the names of the directories and files are self explanatory. Therefore, I am not going to describe every single file but only the most important parts.

- `all_scenes.db` - It contains knowledge about all scenes that are in `scenes/` directory.
- `analogy/` - This directory is the analogy package that can be reused in a different project. It contains all the necessary algorithms and data structures.
- `analogy/collision_detection/` - This directory algorithms for collision detection. This currently includes triangle-triangle collision detection using two different algorithms implemented in C programming language. The file `triangle_col_detect.py` is a wrapper for

these algorithms. The file `aabb_collision.py` contains the axis-aligned bounding box collision check algorithm.

- `analogy/file_parsers.py` - It contains file parsers. Currently supported only OBJ file format.
- `analogy/mapping.py` - It contains the analogy scoring algorithm and mapping algorithm for AABBs.
- `analogy/mesh.py` - It contains Classes for Mesh, Surface, Vertex and AABB.
- `analogy/storage/` - This directory should hold all connectors for storage options. Currently, there is implemented support only for SQLite database.
- `analogy/vpython_drawings.py` - It includes functions that use vpython library for drawing.
- `analogy.py` - It is the main file that puts all together.
- `books-shelf.db` - It contains knowledge only about a one scene and that is the `scenes/books-shelf.obj`.
- `requirements.txt` - List of all the python3 requirements.
- `scenes/` - This directory contains example scenes. This means OBJ files and their .mtl files.
- `user_input.py` - It contains functions for getting manipulation points and force vectors from the user.

Bibliography

[1] F. Guerin and P. Abelha, "Robot Manipulation in Open Environments: New Perspectives," vol. 13, no. 9, p. 7, 2014.

[2] "robot | Definition of robot in English by Oxford Dictionaries." [Online]. Available: https://en.oxforddictionaries.com/definition/robot

[3] "analogy | Definition of analogy in English by Oxford Dictionaries." [Online]. Available: https://en.oxforddictionaries.com/definition/analogy

[4] Stanford, "Analogy as the Core of Cognition." [Online]. Available: https://www.youtube.com/watch?v=n8m7lFQ3njk

[5] D. Gentner, "Structure-Mapping: A Theoretical Framework for Analogy," p. 9.

[6] D. Gentner and K. D. Forbus, "Computational models of analogy: Computational models of analogy," *Wiley Interdisciplinary Reviews: Cognitive Science*, vol. 2, no. 3, pp. 266–276, May 2011. [Online]. Available: http://doi.wiley.com/10.1002/wcs.105

[7] B. Falkenhainer, K. D. Forbus, and D. Gentner, "The structure-mapping engine: Algorithm and examples," *Artificial Intelligence*, vol. 41, no. 1, pp. 1–63, Nov. 1989. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/0004370289900775

[8] M. Mitchell, "Melanie Mitchell," *ACM SIGEVOlution*, vol. 8, no. 2, pp. 4–4, Mar. 2016. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2907674.2907678

[9] C. T. Morrison and E. Dietrich, "Structure-Mapping vs. High-level Perception: The Mistaken Fight Over The Explanation of Analogy," p. 5.

[10] R. Sutton, "The Bitter Lesson," Mar. 2019. [Online]. Available: http://www.incompleteideas.net/IncIdeas/BitterLesson.html

[11] S. Brandi, O. Kroemer, and J. Peters, "Generalizing pouring actions between objects using warped parameters," in *2014 IEEE-RAS International Conference on Humanoid Robots*. Madrid, Spain: IEEE, Nov. 2014, pp. 616–621. [Online]. Available: http://ieeexplore.ieee.org/document/7041426/

[12] T. Fitzgerald, A. Goel, and A. Thomaz, "Human-Robot Co-Creativity: Task Transfer on a Spectrum of Similarity," p. 8.

[13] ——, "Human-Guided Object Mapping for Task Transfer," vol. 7, no. 2, p. 24.

[14] T. Fitzgerald, K. McGreggor, B. Akgun, A. Thomaz, and A. Goel, "Visual Case Retrieval for Interpreting Skill Demonstrations," in *Case-Based Reasoning Research and Development*, E. Hüllermeier and M. Minor, Eds. Cham: Springer International Publishing, 2015, vol. 9343, pp. 119–133. [Online]. Available: http://link.springer.com/10.1007/978-3-319-24586-7_9

[15] C. Dornhege, M. Gissler, M. Teschner, and B. Nebel, "Integrating symbolic and geometric

planning for mobile manipulation," in *2009 IEEE International Workshop on Safety, Security & Rescue Robotics (SSRR 2009)*. Denver, CO, USA: IEEE, Nov. 2009, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/5424160/

[16] M. Roy, S. Foufou, and F. Truchetet, "MESH COMPARISON USING ATTRIBUTE DEVIATION METRIC," *International Journal of Image and Graphics*, vol. 04, no. 01, pp. 127–140, Jan. 2004. [Online]. Available: http://www.worldscientific.com/doi/abs/10.1142/S0219467804001324

[17] P. Cignoni, C. Rocchini, and R. Scopigno, "Metro: measuring error on simpli ed surfaces," p. 8, 1998.

[18] O. Tropp, A. Tal, and I. Shimshoni, "A fast triangle to triangle intersection test for collision detection," *Computer Animation and Virtual Worlds*, vol. 17, no. 5, pp. 527–535, Dec. 2006. [Online]. Available: http://doi.wiley.com/10.1002/cav.115

[19] T. Möller, "A Fast Triangle-Triangle Intersection Test," *Journal of Graphics Tools*, vol. 2, no. 2, pp. 25–30, Jan. 1997. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/10867651.1997.10487472

[20] O. Devillers and P. Guigue, "Faster Triangle-Triangle Intersection Tests," *Bulletin of Sociological Methodology/Bulletin de Méthodologie Sociologique*, vol. 37, no. 1, pp. 55–57, 2002. [Online]. Available: http://journals.sagepub.com/doi/10.1177/075910639203700105

[21] C. Moreno, "Efficient 2d Geometric Operations." [Online]. Available: https://www.mochima.com/articles/cuj_geometry_article/cuj_geometry_article.html

[22] "3d collision detection." [Online]. Available: https://developer.mozilla.org/en-US/docs/Games/Techniques/3D_collision_detection

[23] J. Flunt and D. Kodicek, *Mathematics & Physics for Programmers (GAME DEVELOPMENT SERIES)*, 2nd ed., 2012.

[24] S. Duncan, "Robotic Automatic Task Oriented Manipulation," p. 60.

[25] R. Graf, "Simulating adaptive object manipulation tasks," p. 49.

[26] P. Abelha and F. Guerin, "Transfer of Tool Affordance and Manipulation Cues with 3d Vision Data," *arXiv:1710.04970 [cs]*, Oct. 2017, arXiv: 1710.04970. [Online]. Available: http://arxiv.org/abs/1710.04970

[27] scooley, "Learn about the Windows Subsystem for Linux." [Online]. Available: https://docs.microsoft.com/en-us/windows/wsl/about

[28] dragon119, "Pipes and Filters pattern - Cloud Design Patterns." [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/patterns/pipes-and-filters

[29] alexbuckgit, "Publisher-Subscriber pattern." [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/patterns/publisher-subscriber

[30] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*.

[31] "ROS/Tutorials/WritingPublisherSubscriber(python) - ROS Wiki." [Online]. Available: https://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

[32] R. Mall, *FUNDAMENTALS OF SOFTWARE ENGINEERING*. PHI Learning, 2014.

[33] "8 Most Common 3d File Formats in 2019," Jan. 2019. [Online]. Available: https://all3dp.com/3d-file-format-3d-files-3d-printer-3d-cad-vrml-stl-obj/

[34] “Export to STL or OBJ - FreeCAD Documentation.” [Online]. Available: https://www.freecadweb.org/wiki/Export_to_STL_or_OBJ

[35] “Get 3d Builder - Microsoft Store en-GB.” [Online]. Available: https://www.microsoft.com/en-gb/p/3d-builder/9wzdncrfj3t6

[36] “sqlite3 — DB-API 2.0 interface for SQLite databases — Python 3.7.3 documentation.” [Online]. Available: https://docs.python.org/3.7/library/sqlite3.html

[37] “Stack Overflow Developer Survey 2017.” [Online]. Available: https://insights.stackoverflow.com/survey/2017/?utm_source=so-owned&utm_medium=social&utm_campaign=dev-survey-2017&utm_content=social-share

[38] “styleguide.” [Online]. Available: http://google.github.io/styleguide/pyguide.html

[39] “BSD 2-Clause “Simplified” License.” [Online]. Available: https://choosealicense.com/licenses/bsd-2-clause/

[40] “BSD 3-Clause “New” or “Revised” License.” [Online]. Available: https://choosealicense.com/licenses/bsd-3-clause/

[41] “MIT License.” [Online]. Available: https://choosealicense.com/licenses/mit/

[42] “Oren Tropp algorithm implementation.” [Online]. Available: https://cgm.technion.ac.il/Computer-Graphics-Multimedia/Software/TriangleIntersection/

[43] “algorithm - Triangle Triangle Intersection in 3d-Space.” [Online]. Available: https://stackoverflow.com/questions/1496215/triangle-triangle-intersection-in-3d-space

[44] “Mollers algorithm.” [Online]. Available: http://fileadmin.cs.lth.se/cs/Personal/Tomas_Akenine-Moller/code/opttritri.txt

[45] E. Haines, “Code developed for articles in the "Journal of Graphics Tools": erich666/jgt-code,” Feb. 2019, original-date: 2015-10-07T13:59:17Z. [Online]. Available: https://github.com/erich666/jgt-code

[46] “First Normal Form (1nf) of Database Normalization | Studytonight.” [Online]. Available: https://www.studytonight.com/dbms/first-normal-form.php

[47] “Second Normal Form (2nf) of Database Normalization | Studytonight.” [Online]. Available: https://www.studytonight.com/dbms/second-normal-form.php

[48] “Third Normal Form (3nf) of Database Normalization | Studytonight.” [Online]. Available: https://www.studytonight.com/dbms/third-normal-form.php

[49] M. Baker, “Maths - Cube Rotation - Martin Baker.” [Online]. Available: http://www.euclideanspace.com/maths/discrete/groups/categorise/finite/cube/index.htm

[50] “What is the rotation group of a cube?” [Online]. Available: https://www.physicsforums.com/threads/what-is-the-rotation-group-of-a-cube.505545/