TUM School of Computation, Information and Technology
Technical University of Munich

Semester Thesis

Fundamental Concept Development of a Robot Gripper Fingertip Design Machine Learning Framework

Supervisor Prof. Sami Haddadin

Advisor Johannes Ringwald

Author Pierre Quentin Ngandjui Tiako

Date March 8, 2024 in Munich

Disclaimer

I confirm that this Semester Thesis is my own work and I have documented all sources and material used.

Munich, March 8, 2024

(Pierre Quentin Ngandjui Tiako)

Abstract

Tactile robot-based assembly lines adaptation to new products are limited by the manual redesign, manufacturing, and exchange of the end-effector setup, given that this end-effector design often needs to be adapted to the object's geometry in order to allow the desired object manipulations. Machine learning makes it possible to automate the end effector design phase, generating fingertips with the right shape to hold the object or, in our case, automatically reproducing manual methods such as the projected surface representation and Bézier surface fit methods to generate the desired shape for the fingertip of the end effector. However, the use of machine learning requires a systematic search for appropriate data, which in this case is object data suitable for simulating manipulation tasks and for fingertip design. This thesis will therefore present a process for developing a database framework adapted to fingertip design with a databse technology adapted to our needs. Different functionalities of a database framework will be explored: the initialization of a database, the communication between the database and the training pipeline of the machine learning method and finally application of corrections on the object e.g its orientation in relation to the end effector.

Zusammenfassung

Taktile robotergestützte Montagelinien, die an neue Produkte angepasst werden, sind durch die manuelle Neuentwicklung, Herstellung und den Austausch der Endeffektoren begrenzt, da diese oft an die Geometrie des Objekts angepasst werden müssen, um die gewünschten Objektmanipulationen zu ermöglichen. Das maschinelle Lernen ermöglicht es, die Entwurfsphase des Endeffektors zu automatisieren und Fingerspitzen mit der richtigen Form zum Halten des Objekts zu erzeugen oder, in unserem Fall, manuelle Methoden wie die projizierte Oberflächendarstellung und die Bézier-Oberflächenanpassung automatisch zu reproduzieren, um die gewünschte Form für die Fingerspitze des Endeffektors zu erzeugen. Der Einsatz von maschinellem Lernen erfordert jedoch eine systematische Suche nach geeigneten Daten, in diesem Fall nach Objektdaten, die für die Simulation von Manipulationsaufgaben und für das Design der Fingerspitze geeignet sind. In dieser Arbeit wird daher ein Verfahren zur Entwicklung eines Datenbank-Frameworks vorgestellt, das an das Design von Fingerspitzen angepasst ist und eine an unsere Bedürfnisse angepasste Datenbanktechnologie verwendet. Es werden verschiedene Funktionalitäten eines Datenbank-Frameworks untersucht: die Initialisierung einer Datenbank, die Kommunikation zwischen der Datenbank und der Trainingspipeline der maschinellen Lernmethode und schließlich die Anwendung von Korrekturen am Objekt, z.B. seine Orientierung im Verhältnis zum Endeffektor.

Contents

Chapter 1

Introduction

As the physical interface between the robot and its environment, the end effector is the element which, almost by its very presence, defines the function of the robot [13]. In industrial assembly applications, the end effector must be adapted to the products handled by the assembly line. This adaptation for new products is limited by three stages: the manual re-design, manufacturing, and exchange of the end-effector [14].Our setup automatically changes the fingertip of the end effector to suit the object being processed. It can also be used to automate fingertip design using machine learning methods.

There are two types of fingertip design method [14]: **form closure methods** , which develop a fingertip adapted to the shape of the object and the location of the gripper fingers on the object must be sufficient to lock the object in a position in space, **force closure methods**, which involve determining the best position for the gripper's fingers and the friction forces between the gripper and the object in order to achieve an stable grip.

These two methods, or their combination, give rise to two categories of strategy [13]. **Analytical strategies** [5], which take a more mathematical approach to the shape of the object to be grasped and the forces to be exerted on it to ensure a stable grasping position, such as projected surface representation and Bézier fit surface, which enable fingertip design using different forms of form closure. **Data driven strategies**, which use CAD (Computer aided design) objects to help with fingertip design. These can be further divided into 2 subcategories, *simulation strategies*, as in [32], which use force closure methods to determine the best possible grasping positions (grasping sets) for a given gripper and object. More recently, there are *strategies that make use of machine learning* , which allow a variety of possibilities: reproducing analytical methods in the case of our setup, automatically generating the best grasping sets, or so-called generative design strategies such as Fit2Form [17], which can directly generate the CAD model of the fingertip adapted to the object.

However, in the context of our project, which uses machine learning strategies, one apparent limitation is the need to have an exhaustive database and to be able to use it in the most effective way.

So for my thesis in the context of our setup, in order to ensure that our machine learning approach works properly, research has been documented on fingertip design in general and in more depth on the operation of data driven approaches and machine learning approaches in particular. In order to resolve the limitations posed by machine learning approaches, we had to develop a flexible database and the tools to use it effectively. This involved researching and determining the most suitable database technologies, existing databases for supplying CAD models and determining which CAD format to use.

Chapter 2

State of the Art

The study and development of new design strategies for grippers has continued unabated since the modernization of the industry and the use of machines in assembly lines. These strategies have been modernized in particular with the use of CAD (Computer Aided Design), which are 3D models of objects ranging from mechanical parts to everyday objects. This has led to a differentiation in the methods used, which will be explored below in the different stages of finger design according to the strategies used. There has also been a resurgence in the number of CAD databases, whose file formats have themselves diversified. .

2.1 Parallel Gripper Finger-based Design

In the work of Honarpardaz et al. in [13], The gripper design process can be divided into three key processes: Grasp, Finger design, Experimental verification. The grasp process is divided into two parts: Grasp synthesis and Grasp analysis.

2.1.1 Grasp Synthesis and Analysis

Grasp, in the context of robotics, is the process by which an object is held to perform a specific task. This process is divided into 2 stages [13]:

- **Grasp Synthesis** is the search process that identifies which of the possible positions of grasps are closed grasps (equilibrium). Here, the stability of the position is not yet taken into account. This process differs from one strategy to another. These differences will be assessed below.
- **Grasp Analysis** is the process of checking the level of stability of the grasp positions identified during grasp synthesis. More specifically, we need to ensure that they are resistant to disturbance, and that they comply with force or form closure conditions. In this case, the stability assessment processes are not specific to one approach (analytical or data-driven) in particular. The conditions taken into account to ensure stability can be *general metrics* [10], linked only to the different forces that apply between the gripper and the object without consideration of the influence of the task to be carried out, *task specific metrics* [23] that take into account the constraints posed by the task to be carried out (specific position in which the object must be held, for example) and finally *uncertainty metrics* [3] that consider the different errors that can occur during the process of gripping the object (error in modeling the shape of the object, its location, etc.).

Analytical approaches

Analytical approaches are based on the mathematical modeling of the object, i.e. its geometry, but also the formulation of the kinematic and dynamic equations linked to the object when it is gripped. From this approach, 4 general directions can be drawn [13]:

- **Limited Contact Methods for Polyhedral Objects** considers the minimum number of contact points to ensure a closed force on the object as the operating condition. For example, Nguyen's work [29], which gives the minimum number of contact points to ensure balanced grasping, whether in the case of soft fingers or the absence of friction. An example of the execution process for this method can be found at fig. 2.1.

- **Unlimited Contact Methods for Polyhedral Objects** is an extension of the previous method when the number of contact points is not known. One possible method is that presented in the work of Liu et al [25] and shown in the example of fig. 2.3. The idea is to be able, for n contact points, to fix n-1 points without respecting a force closure equilibrium and to be able, by fixing the nth point, to find a force closure equilibrium.

- **Heuristic Methods** work with an unlimited number of contact points. The difference lies in the fact that random set grasps are first generated, or the contact points are chosen randomly, and then different rules are used to check the balance. In the example fig. 2.2, the rule is the same as that used for the previous case for an unlimited number of points, the condition to be met is that the origin of the wrench space lies precisely inside the convex hull of the primitive contact wrenches resulted by the contact forces at the fingers.

- **Methods for General Object Shape** is an extension of the above methods to more complex objects. Applying these methods to more complex objects would not be computationally efficient. Ding's work [9], for example, allows the heuristic method of fig. 2.2, to be generalized to complex objects by discretising the 3D model of the object into a cloud of points, with the normal of each of these points also stored. They show that this makes the search for contact points computationally more efficient.

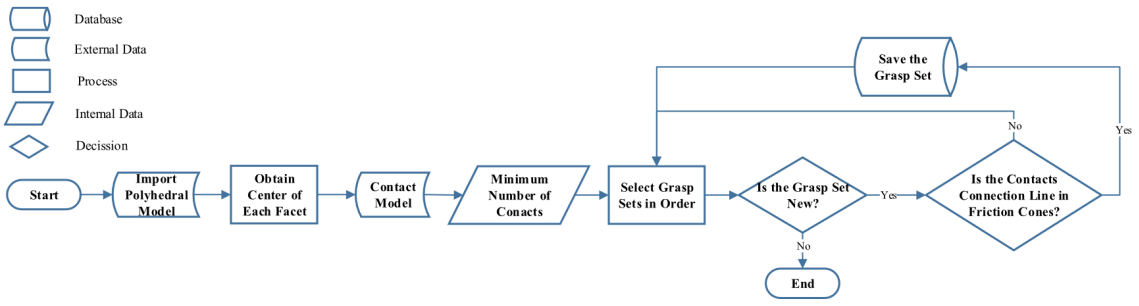

Figure 2.1: Flowchart of an example of limited contact methods for polyhedral objects Taken from [13]. It gives an example of the process to be followed in this case to determine the grasp sets that ensure a certain force closure, hence the check to see if the contact points are in friction cones to ensure that the object is gripped.

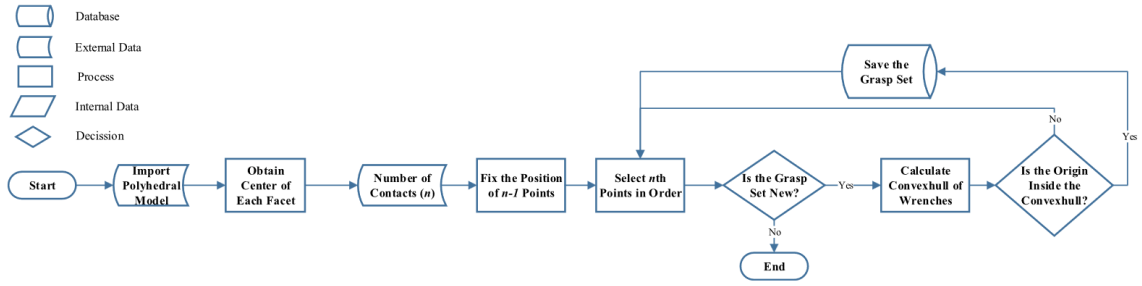

Figure 2.2: Flowchart of an example of unlimited contact methods for polyhedral objects Taken from [13]. This is an example that draws on Liu's work [25] to determine an equilibrium position in force closure by adding a point to n-1 others that do not meet the force closure condition. A necessary and sufficient condition to ensure this equilibrium is that the origin of the wrench space lies precisely inside the convex hull of the primitive contact wrenches resulted by the contact forces at the fingers..

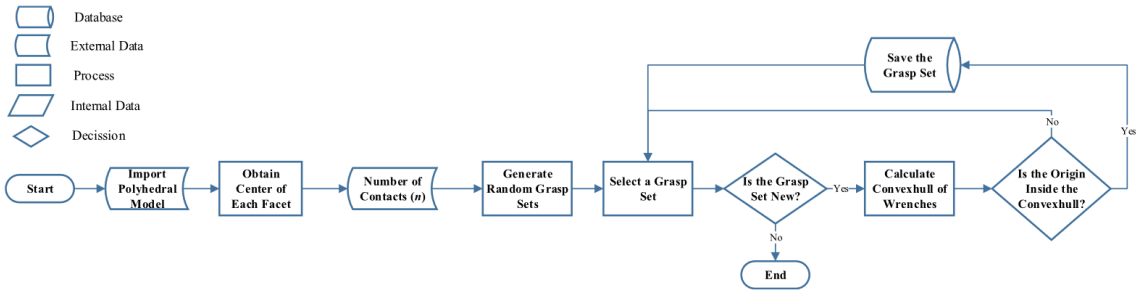

Figure 2.3: Flowchart of an example of heuristic methods. Taken from [13].

Data driven approaches

Data driven approaches make full use of CAD to generate grasp sets. Instead of relying on mathematical formulations and equations, it is possible to check the balance of each grasp using simulation. Ramasubramanian's work [32] shows that it is possible to use CAD software to automate the grasp synthesis process using simulation, and even more so, because it is also possible to test the stability of these grasps, i.e. perform grasp analysis with them.

To go further, simulation methods can be integrated into the entire design process, not just the grasp phase. In Schwartz's work [36], the aim is, as on, to generate fingers with the right shape by starting with monolithic fingers and seeing the imprint left by the object on the gripper's fingers. Simulation not only generates the grasp sets and imprints, but also plays a part in the optimization phase, as can be seen in . Also following the same principle, Wolniakowski et al [40] have worked on a similar approach, with design differences that can be seen on the left-hand side of fig. 2.4.

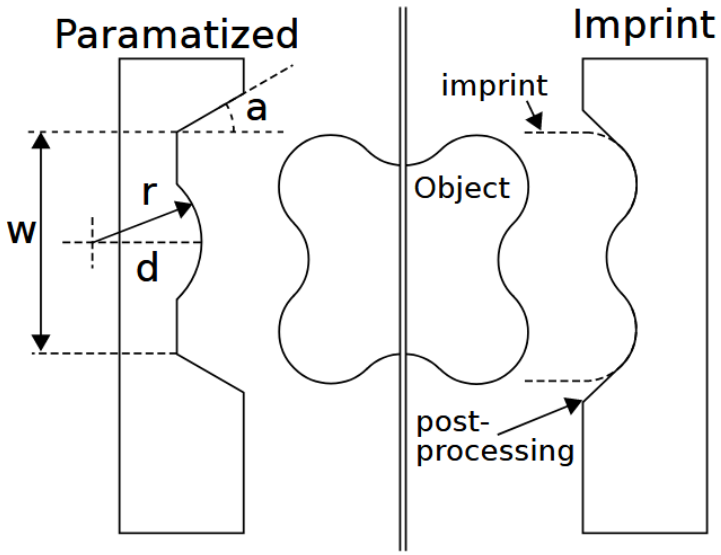

Figure 2.4: The two different ways of designing the finger. The first on the left is by parametrization by Wolniakowski et al [40] and the second on the right is by the method with imprints by Schwartz et al. Taken from [36] .

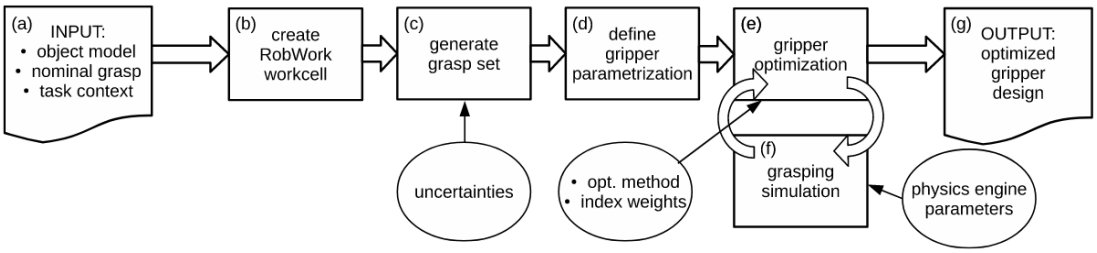

Figure 2.5: Workflow of method Taken from [36]. It allows simulation to be used throughout the finger design process. it can use simulation to take into account parameters related to the task. .

2.1.2 Methods of finger design

According to [13], thinking about gripper design can take 3 forms: **modular design**, **reconfigurable design** and **customized design**.

Modular design consists in most cases of having a set of existing fingers ready for a given gripper. The aim is to be able to adapt the gripper's fingers to suit the object. This is often done by simplifying the geometry of the object (using cylinders, cubes, etc.) in order to determine the most suitable fingers, as was done by Brown et al. [6] and Friedman et al [11]. However, other approaches are possible, such as Sanfilipino et al. [34], whose aim is to have a gripper with several degrees of freedom, one per finger, and to be able to control each of these degrees of freedom to obtain the ideal gripper for the given object.

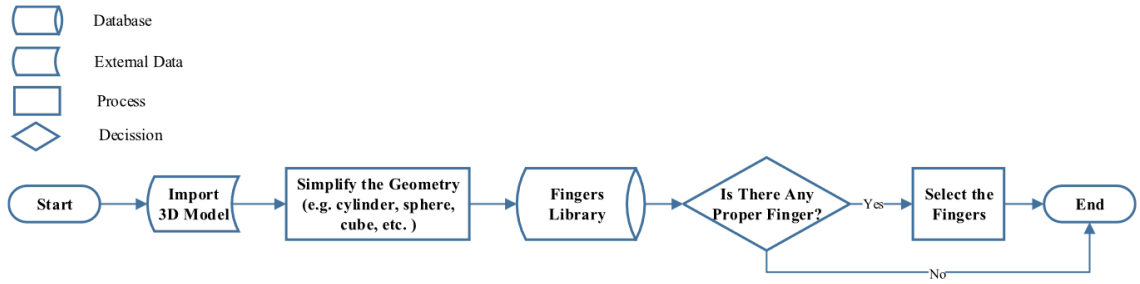

Figure 2.6: Flowchart of an example of modular finger design Taken from [13] .

Re-configurable design approaches are based on methods that, for a fixed gripper, determine the most stable way of holding the object. Many methods employ simulation software such as grasp it [27] are used in this framework to determine the most stable grasp. Learning algorithms are used extensively in this design approach. They allow continuous improvement as new objects are encountered, the notable difference being the parameter that is learned. In the case [31], the algorithm learns from a dictionary of prototypical grasp-predicting parts. And in the [12] case, the shape of the objects most suitable for gripping by this gripper is learned.

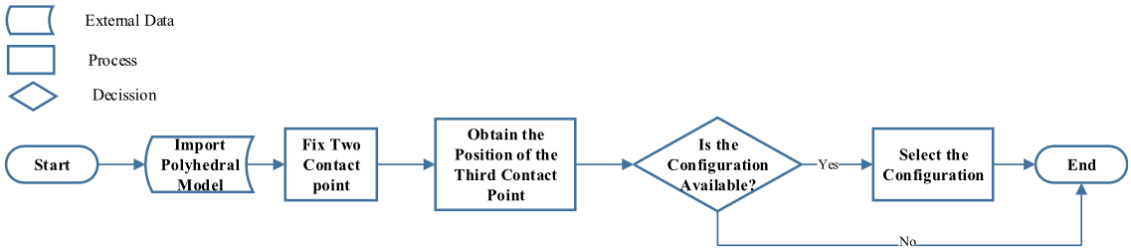

Figure 2.7: Flowchart of an example of re-configurable finger design Taken from [13] .

Customized design approaches aim to create a specific gripper finger for a work-piece. In a similar way, based on the shape of the object, the work of Wolniakowski et al [40], Ringwald et al [33] and Schwartz et al [36] all have a customized design approach. The first two have a parametrised version of the shape of the object and Schwartz et al use the footprint of the object instead, see fig. 2.4.

2.2 Automatic Gripper Finger Design

2.2.1 Approach Overview

To improve the adaptation of Tactile robot-based assembly lines to new products, it is necessary to be able to automate the redesign, manufacturing and changing of the end effector. The last two stages are relatively straightforward, as shown by Ringwald's strategy [33], in which the fingertip is manufactured using 3D printing. However, finger design strategies such as modular and re-configurable require either a simplification of the object geometry, see fig. 2.6, or are applicable to polyhedral, see fig. 2.7. These design strategies do not generalise well to objects of variable geometry. It is therefore necessary to be able to automate the custom design strategy.

There are two general concepts involved in automating finger design: form closure and force closure. In developing a process to automate finger design, these two concepts can be

considered independently or in combination. Form closure develop a fingertip adapted to the shape of the object and the location of the gripper fingers on the object must be sufficient to lock the object in a position in space [14]. Force closure is used to design a gripper based on the friction forces between the gripper and the object in order to maintain a stable grip. As these two concepts are independent, the aim is to be able to automate the choice of one of these concepts for the design, as well as the different stages of gripper design discussed above. Traces of this desire to automate grippers' design using the form closure approach can be found as far back as the 90s, with the article [38]. They start with a gripper made up of 2 monolithic fingers and their shape is designed to correspond to the shape best suited to handling the given object.

One group of researchers whose work on this subject has been particularly prolific is Honarpardaz et al. with the Generic automated finger design [14]. Their work has made it possible to determine a process that can be fully automated, fig. 2.8, to produce suitable fingertips. The work of Honarpardaz et all has recently played a key role in developing an automatic process for designing fingertips. This GAFD process was further developed and modified in their subsequent studies. Methods were developed to use pre-existing samples of form closure and to use contact points to generate fingertips, speeding up the process and giving the Fast generic automated finger design (FGAFD) [26]. A more optimized version of this process, Generic optimized finger design (GOFD), has also been created [16], see in Appendix A. The notable differences are that it takes into account the experimental verification stage described above. Also, the grasp synthesis and analysis stage uses a genetic algorithm where the list of solutions obtained is derived from a set of individuals that have evolved using processes such as succession and crossover. The finger design stage has also been improved, in particular to update the database of possible grasp sets. They have also turned their attention to developing such a process for soft robotics in their recent work [15].

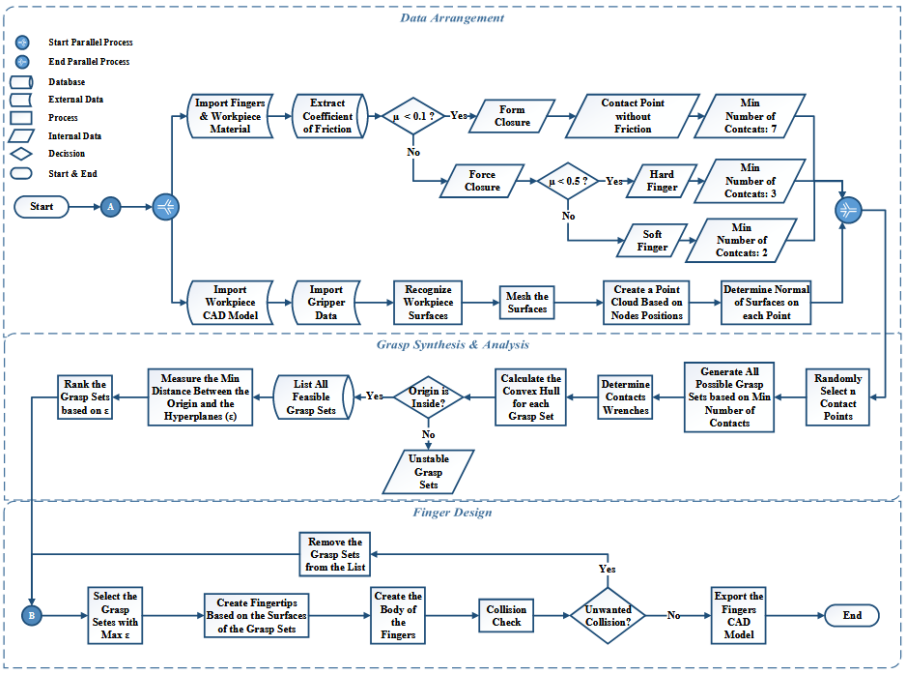

Figure 2.8: GAFD flowchart Taken from [14]. It structures a process for determining the optimum gripper for an object. The first phase of the data arrangement consists of analysing the coefficient of friction to determine the optimum number of contact points for gripping the object and the technique (form closure or force closure) to be used. The second phase, grasp analysis and synthesis, uses the information from the first phase to generate a list of the different ways of grasping the object, keeping only those that are stable and ranking them in order of stability. Finally, the fingertips are generated to meet the various conditions defined.

2.2.2 Machine Learning applications in design

Machine learning algorithms are extremely useful for improving process computation times. In the case of design in general, as the complexity of the systems to be designed increases, excessively long computation times are necessary [30]. This is where machine learning algorithms come in to improve computation times. Considering a fingertip design problem, where the objective is to generate the fingertip surface adapted to the shape of the object in order to ensure form closure. Mathematically, the solution surface for this problem is a two-dimensional function that is tangent to the object at any point adjacent to it. However, constraints can be applied to this surface to ensure a force closure, for example, or constraints linked to the task to be carried out with this object. Thus, parts of the design process can be likened to optimization problems. Thus, [30] proposes various machine learning algorithms that are used in these optimization problems.

A further step in this initiative is to enable the 3D object required for the design to be generated directly. In this case, theories such as topology optimization are require. This makes it possible to optimize the material layout in the design domain defined by [24]. This is done by adding conditions to the initial design optimization problems. In general, the objective of topology optimization is to minimize the strain energy of the structure to be synthesized [24]. Different strategies can be used for topology optimization, and in the context of [24] it is applied to finite elements. Combining topology optimization and machine learning makes it possible to develop generative design methods. As a result, a method such as MLGen [21] was created, combining memory neuron long-short term networks (LSTM) with a topology optimization method using finite elements. Other examples of the use of generative design methods in [18] use a system with a generator and a discriminator to optimize the designs generated for particular tasks.

To return more specifically to the case of finger design for grippers, two applications of machine learning can be found. On the one hand, its exclusive use in the Grasp phase. As in the work of J.Varley et al [19], algorithms can be used to speed up this phase by generating the various possible grasp sets for the given object as seen in fig. 2.10. On the other hand, the other application of machine learning in design is, as seen above, generative design, which can also be used differently depending on the situation. In the case of Ai et Al work [2], the aim is to optimize a predefined shape of gripper, and this involves evaluating the various constraints of the optimization problem. The gripper before and after optimization can be seen on fig. 2.11. The most interesting case is Fit2form [17]. The design process is completely automated. It consists of a generator network that generates fingertip models adapted to the input object and a fitness network that measures certain criteria of grasp quality. The two are optimized together to ensure the best possible fingertips and an accurate measurement of grasp quality as seen in fig. 2.9.

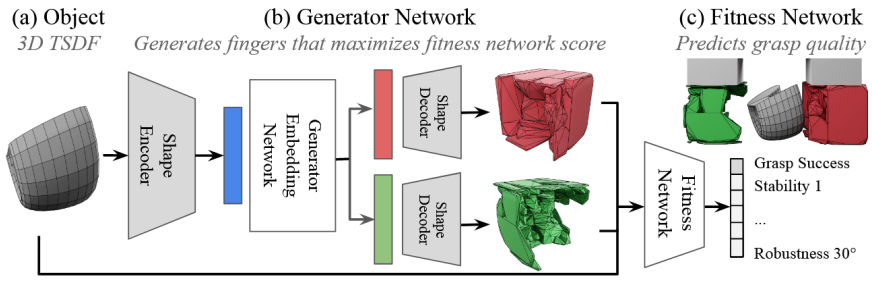

Figure 2.9: Fit2Form framework taken from [17]. It describes how does fit2form work. It takes a 3D objects as input encodes it shapes and the generator create fingertips that will maximize the fitness score that is measure with the fitness network. The generator and fitness network are train together to maximizes their efficiency.

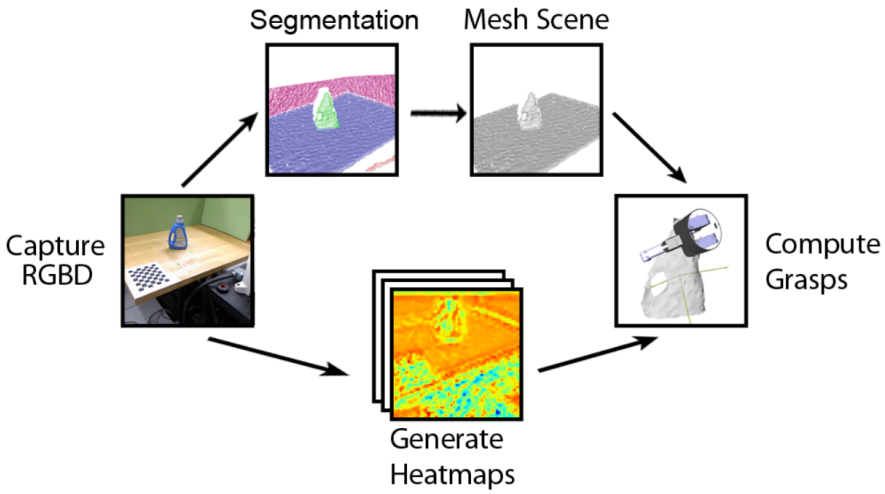

Figure 2.10: Overview of the Grasp Generation System taken from [19]. After an image is captured, the scene is segmented and meshed while the heatmaps are generated in parallel. The heatmaps give the possible locations of the finger on the identified object. The meshed scene and the heatmaps are used within GraspIt! [27] to plan grasps..

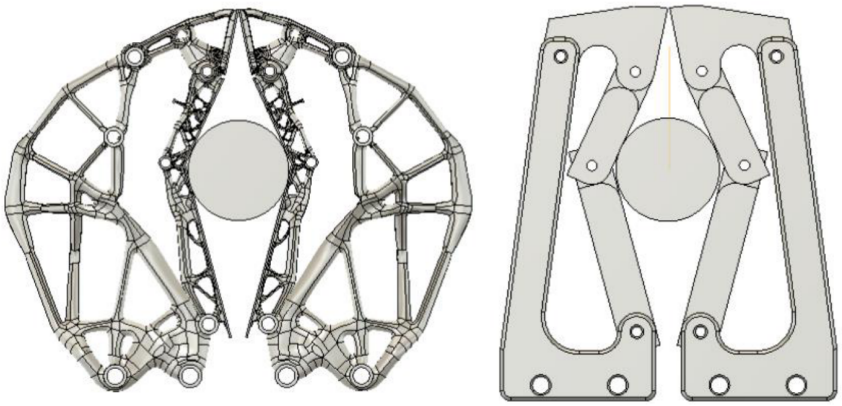

Figure 2.11: View of the gripper before (right) and after (left) optimisation taken from [2]. The aim is to use the gripper on the right to develop a gripper that combines optimisation of the force closure (the forces exerted on the object) and optimisation of the topology.

2.3 CAD Database

In finger design machine learning applications, particularly in generative design cases such as Fit2form [17], it is necessary to have a database of 3D models for model training. In this section, we will look at ways of building a CAD database, including where to find them and existing database technologies.

2.3.1 Existing databases for 3D object models

In order to use data-driven finger design methods, whether through the use of machine learning or otherwise, it is necessary to have databases containing 3D models of the elements that

will be manipulated. These elements are called Computer Assisted Design or CAD. The popularization of computer-aided design has led to the development of these databases, which are accessible via the Internet. They can be distinguished by their accessibility, the CAD file format used and the different objects inside.

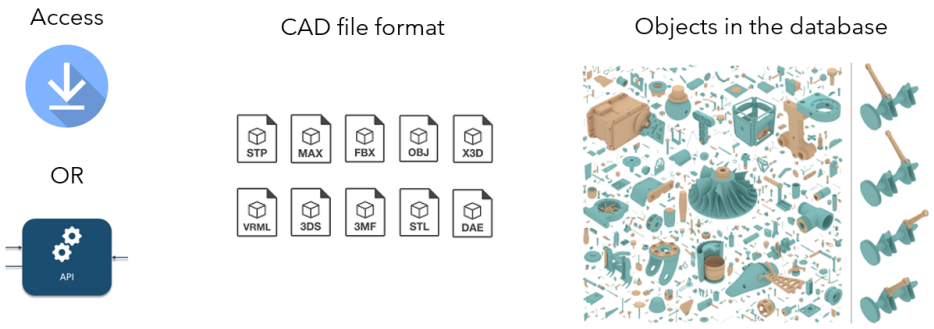

Figure 2.12: Differentiation criteria of sources databases. the CADs are either downloadable or accessible via an API, there are different types of file formats and objects.

The databases most frequently used in scientific articles include those listed in table 2.1 below.

Table 2.1: List of source databases and their charateristics

Database's name	Reference	File format	Types of objects	Availability
ABC	[22]	meta, step, para, stl2	From mechanical parts to everyday objects	Downloadable by chunks
Fabwave	[4]	step, stl, f3d	Mechanical parts	Downloadable by chunks
Grabcad	[37]	depend on the CAD	From basic CAD parts to complex CAD assemblies	Downloadable
Onshape	[35]	step, parasolid, jt,catia, nx	From basic CAD parts to complex CAD assemblies	API and Downloadable
Fit2Form DB	[17]	stl, obj	Complex geometrical objects	Downloadable
Bigbird	[1]	obj	Everyday objects	Downloadable
Shapenet	[8, 41]	obj	Everyday objects	API and Downloadable
Ycb Obj	[7]	stl, obj	Everyday objects	Downloadable

2.3.2 Databases' technologies

In the context of a project on the scale of an individual or a group of individuals, in order to organize the data to be used correctly, there are a multitude of technologies available to achieve this. Several classifications of these technologies are possible. The first is the difference between technologies that use the Structured Query Language (SQL) model and those that do not. SQL is a very popular query language based on tables and operations between them.

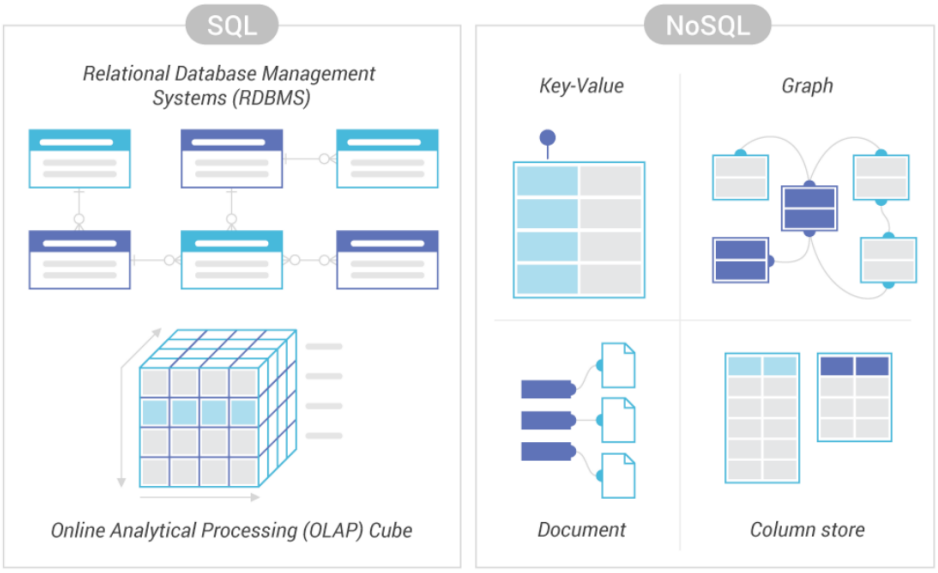

Figure 2.13: Classification of database technologies. There are those using the SQL system and those which doesn't. As well as storing data in SQL tables, technologies that do not use SQL language each have a way of storing and accessing data. Data can be stored in the form of a document with a unique identifier, and each document can have attributes similar to others, enabling documents to be aggregated to provide access to multiple data sets that meet certain characteristics. On the other hand, for highly relational data (for example, to store data representing a social network, i.e. people and their relationships), technologies using graphs will be the most appropriate.

In the fig. 2.13, when it comes to technologies that don't work on the SQL system, there are several kinds. Some are based on schema that describe the relationships between data, such as the graph model with examples like Neo4j. On the other hand, systems based on the document model, such as MongoDB, store data almost independently in document form. These can then be combined into collections.

An important theorem in the use of these technologies for databases is the Consistency Availability Partition tolerance (CAP) theorem [39]. The CAP theorem explains why a distributed database cannot guarantee both consistency and availability in the face of network partitions. The theorem says an application can guarantee only two of the following three features at the same time:

- Consistency — the same answer is given to all
- Availability — access continues, even during a partial system failure
- Partition tolerance — operations remain intact, even if some nodes can't communicate. A "node" typically refers to a single computing entity within the distributed system. In larger systems with bigger databases, all the data may not be stored on the same

server, or may be processed by different machines depending on the request made to the database.

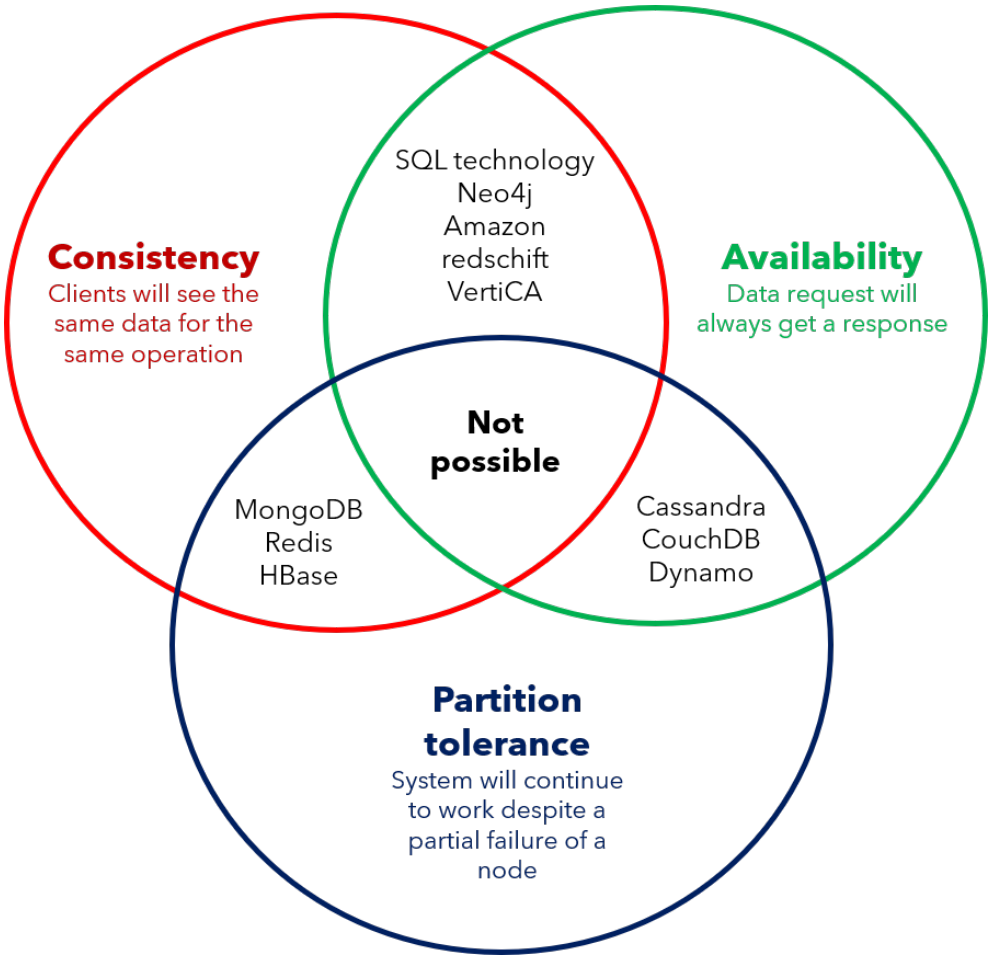

Figure 2.14: Classification of databases technologies by concern to the CAP theorem. In line with the CAP theorem, some technologies, such as mySQL and Neo4j, tend towards Consistency Availability, while others, such as MongoDB, tend towards Consistency Partition Tolerance.

2.3.3 CAD Format

With the development of computer-aided design methods, there has been an emergence of different 3D object formats. These serve different purposes. Some are more suited to encoding curves, and others can be more easily re-meshed. As a result, their use varies between those specialized for simulation and others more suited to 3D printing. Some CAD formats are used more frequently and more universally than others in the various tab1 databases, such as obj widely used in 3D geometry, stl commonly used for 3D printing and rapid prototyping and step, an ISO standard for exchanging data between CAD systems. Others are specific to databases or have been created by companies, such as f3D, which is specific to Fusion 3D and developed by Autodesk, catia, developed by Dassault, and NX, developed by Siemens PLM software. In order to establish an exhaustive CAD database, it is simplest to focus on the 3 most popular and widely used formats: stl, step and obj. From [20], a comparison table of those CAD formats has been drawn up table 2.2.

Table 2.2: Comparison between the STL, STEP and OBJ formats in terms of the way in which object geometry is stored, the various object properties and their usecases

File format	Geometry	Properties	Usecase
STL	Collection of triangular facets facet = vertices coordinates + surface normal	Simple and lightweight	3D printing, rapid prototyping, and computer-aided manufacturing
STEP	Support a wide range of geometric entities (curves, surfaces, solids, assemblies)	Store metadata related to product structure, assembly hierarchy, materials, tolerances, and more	- Data exchange between different CAD systems - Archiving of design data - Virtual prototyping
OBJ	A list of vertices and a list of faces (triangles or polygons) that reference those vertices	Can reference external material library files to define material properties	3D printing, visualization, and game development

Table 2.3: The limitations of each CAD format

File format	Limitations
STL	**Limited Geometry Representation**: approximation of curved surfaces and loss of detail particularly for complex or organic shapes **Lack of Metadata**: No material properties, color information, or assembly structure **No Support for Parametric Modeling**: such as history-based design or constraints
STEP	**Complexity**: contain a large amount of data, handling and processing large STEP files can be resource-intensive and may require specialized software.
OBJ	**Surface Representation Only**: no information about solid bodies, assembly structure, or parametric features **Limited Material Properties**: the material properties they support are relatively basic compared to formats like STEP **No Parametric Data**: like STL

Chapter 3

Analysis and Projects requirements

3.1 State of the art analysis

As part of the project on which this thesis is based, the aim is to make robot-based tactile assembly lines as adaptable as possible to new products. This involves automating the 3 stages: re-design, manufacturing and end effector exchange. However, the cases [2, 33, 40] clearly show that with 3D printing, automating end effector manufacturing is not a problem. All you need is a fixed part of the gripper and a variable part, the fingers, which can easily be replaced. At the same time, automatically replacing the end effector using the method described above is just as trivial. This is how modular design [6, 11] usually works, the aim of which is to have a library of fingers available and to be able to install on the gripper the most appropriate fingers for each product. The work by Ringwald et al [33] also describes, among other things, a mechanism for replacing these fingers.

So, to achieve the goal of adapting tactile robot-based assembly lines to new products, the critical point will be to automate the design of the end effector, or more precisely the fingers or fingertip of the end effector. As seen in the previous chapter, there are two types of approach to design: **analytical** and **data-driven**. **Data-driven** approaches use databases of 3D object models in a variety of ways. They can be used for simulation [32, 36], or to train machine learning methods [17, 19]. On the other hand, **analytical approaches** are based on mathematical and dynamic considerations and formulations, and are essentially manual methods. But they can be automated using machine learning. J.Varley's work [19] takes a similar approach to heuristic methods, using convolutional neural networks to generate heatmaps from the image of a scene in which an object needs to be grasped, representing the possible positions of the gripper's fingers on the object in order to grasp it. So recent developments in this field show that machine learning can automate**analytical approaches** and even turn them into **data-driven approaches**.

In the theme of being able to take rules from analytic approaches and use them in a design automation process, we should take a closer look at the work of Honarpadaz et al [14, 16]. Their GAFD makes it possible to combine force and form closure, and use the rules established by analytic approaches to evaluate which is the best approach. Indeed, in the GAFD figure, in the first phase of their process they use the rules established here fig. 2.8 to determine, in relation to the shape of the gripper and the material of the object, which is the best approach to adopt, force or form closure, in particular by the number of contact points to be used. With this information, it can generate the different possible ways of holding the object, keep the most stable ones, and use the most stable of these to determine what shape the gripper's fingertips should have, while checking that there are no collisions between them.
As shown in the Appendix A, Honarpadaz et al. [16] have used GAFD to create a more optimized process the Generic Optimized Finger Design (GOFD), integrating Experiment ver-

ification and replay optimization levels at each stage of the process, taking assemblies into account (a set made up of several workpieces), and geometric analysis, which allows sufficiently large surfaces of objects to be taken into account from the outset, without modeling surfaces that are too small. Finally, the grasp planning and analysis phase generates better grasp sets.

By comparison, the approach adopted in this project uses machine learning to automate the projected surface representation and Bézier fit surface methods, which are basically manual techniques. The use of these techniques ensures form closure as long as the base surface exceeds a certain size. This eliminates the need for GAFD's grasp analysis phase. And where GAFD does not take account of the task to be performed, which is corrected by GOFD, as long as the surface on which the object is held is of sufficient size, our method makes it possible to fix the object's orientation from the outset (putting the object in the limited position defined by a task) and to directly output the appropriate fingertip.

However, as a machine learning technique, like Fit2Form [17] or J.Valey [19], it is necessary to have an appropriate database, or to be more precise, to create an environment consisting of the database and the communication tools to use it and connect it to the rest of the training pipeline. In order to build up this environment, there are already several free CAD packages available on the Internet, as shown in table 2.1. Each CAD database contains different types of objects, ranging from mechanical parts, to assemblies made up of several different CAD, to 3D models of everyday objects. This database search has also shown that there are different CAD formats, each with its own characteristics, uses and limitations, see table 2.2. The choice of this file format has an impact due to the first process these CAD have to go through. In fact, on the fig. 3.1, the CAD will first be transformed into a process base. During this process, depending on the level of detail of the CAD format, it is possible that the approximation of the base process is poor, as there were not enough points to represent the surface of the object. The CAD format used must either have a good level of detail, or be easily remeshed to increase its level of detail. Finally, to set up the environment, it is necessary to choose an appropriate technology. It must be possible to adapt this model to different objects according to different criteria (their size, type of object, the industrial tasks in which they are most employed). As these different criteria can vary from one object to another, it's best to use a technology that doesn't have a rigid schema, i.e. you can add or remove tags for each piece of data without having to rethink the entire database schema.

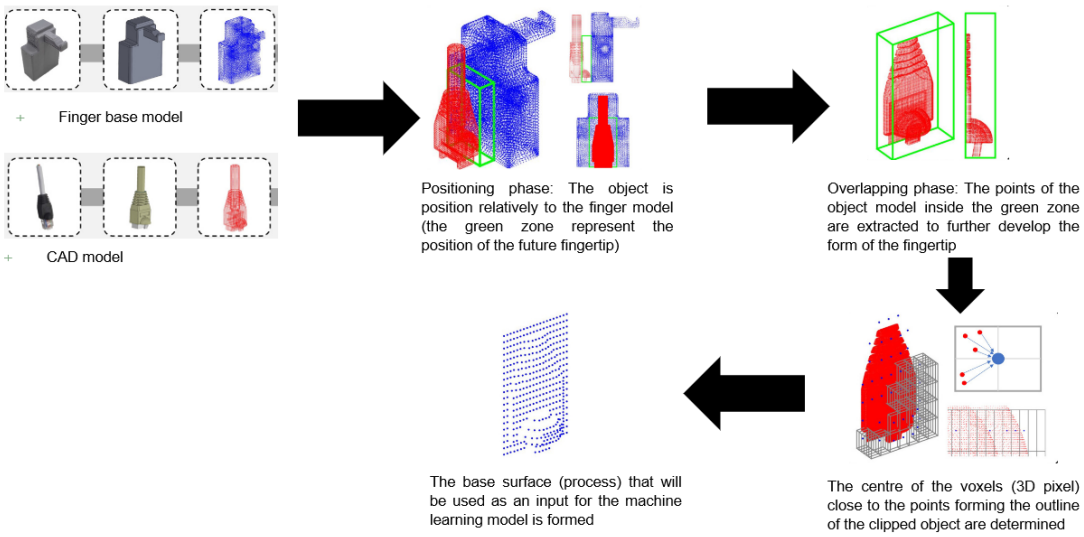

Figure 3.1: Process for obtaining the Base process that will be used as input data for training the machine learning model.

3.2 Project requirements

1. Use a CAD format with a sufficient level of detail, or one that is easy to resize and therefore easy to add levels of detail.

2. Use a consistent database technology with a flexible / no schema. For example, if a group of objects was initially categorized as mechanical parts, we may need to add a “screw” attribute to specify that all these objects are screws. With strict schema technology, such as MySQL tables, if there isn’t a dedicated column for this type of attribute, you’ll need to rethink the entire structure of the table..

3. Develop the tools needed to instantiate the database, modify the database and connect it to the database process pipeline

4. The database must store CAD references and the tools must enable CAD to be retrieved, whether stored locally or remotely. In this case, where the database stores the CAD references, i.e. the path to the machine location where the CAD is stored, these tools must be able to load the CAD regardless of whether the database and CAD are on the computer where the rest of the pipeline is located (locally) or whether they are stored on a completely different computer/server (remotely). In the local case, all you have to do is call the CAD reference in the code to load the CAD. In the remote case, you first need to connect to the remote machine, and decide whether it’s easiest to load the CAD remotely, or to download it to the local machine and then load it.

5. Data storing the reference of a CAD file must also have tags in its attributes that provide information about the object.

Chapter 4

Concept Development

The aim is to develop the foundations of a scalable environment consisting of a database and the tools for linking it to the pipeline, while respecting the requirements defined above and the information obtained in retrospect from the first tests carried out with this model. This information includes, for example, poor approximations of the base process due to the level of detail of the CAD used, or the fact that the pipeline is developed on MATLAB, and so the tools for extracting data from the database must be developed on MATLAB as a minimum.

4.1 CAD format Identification

The choice here will be made on the basis of the comparison in table 2.2 and table 2.3 and the requirements for the project. For the OBJ format, many of the properties such as color and texture are not useful for this project, although the encoding of the geometry using also polygons and not just triangles as facets offers a better level than STL as the curves and more complex surfaces are better approximate with a variety of polygon rather than just triangles.

For the STEP format, although it has the best capacity to encode complex geometry curves and surfaces without without discretization via triangular or polygonal facets, it also contains properties such as the material of the modeled object, and other parametric information such as the history of modification of the CAD. These properties would be useful in the case of a force closure and in case the CAD has to be re-design multiple times. But the STEP format is also the most complex one, and its processing of a large CAD file by the rest of the pipeline could be resource-intensive.

STL is the format with the simplest geometry and is already used in rapid prototyping. However, the simplicity of this geometry means that even the most complex surfaces and curves can be poorly approximated. On the other hand, this simplicity makes it easy to remesh if necessary. What's more, the two previous formats are convertible to this one, but it's not possible to convert STL and OBJ files into STEP files. So, in terms of the number of CAD, it will be simpler to have as many CAD as possible in STL format.

So, given that the format and geometric approximations of the STL format can be adjusted, and that the current model doesn't need information such as the object's material, and given that it's the format with which the greatest number of CAD will be obtained, the choice of CAD format will therefore be STL. But if the project evolves and requires additional information on the object represented by the CAD, you need to keep in mind the other two formats presented.

4.2 Selection of source databases

Once the CAD file format has been identified, the databases from which the files will be extracted must be determined. The first source database will be selected here to lay the foundations for this environment. The other databases not selected will be useful for completing the database of the environment to be set up, but the aim here is to have a first data set for testing the environment to be designed. The origin of each file must also be kept in mind, to ensure the project's continuity. This will enable you to know which source databases have already been used when you need to increase the size of the project database.

As shown in the table 2.1 and fig. 2.12, source databases offer two ways of accessing CAD, via API that allow you to load the CAD you need directly into the program, or to download them directly. Since the aim is to be able to store CAD to create a par database, the easiest way is to download them to have a better control over the references to be stored in the database, or even in the queries. What's more, the API of these platforms are user-linked. Given that the database to be created for this thesis will be for collaborative work within the laboratory, it's simpler to actually have the data from these CAD than to load them via an API.

With the help of table 2.1, the following two diagrams, fig. 4.1 and fig. 4.2, have been drawn up to determine the first databases that can be exploited. Looking at these diagrams, the source databases that are of interest for this project are: ABC, Fabwave, Grabcad and YCB object. OR, because ABC and Fabwave allow CAD files to be downloaded in chunks of several hundred, they are easier to use in the design of the database. For a future scale-up, YCB object and Grabcad could also be considered.

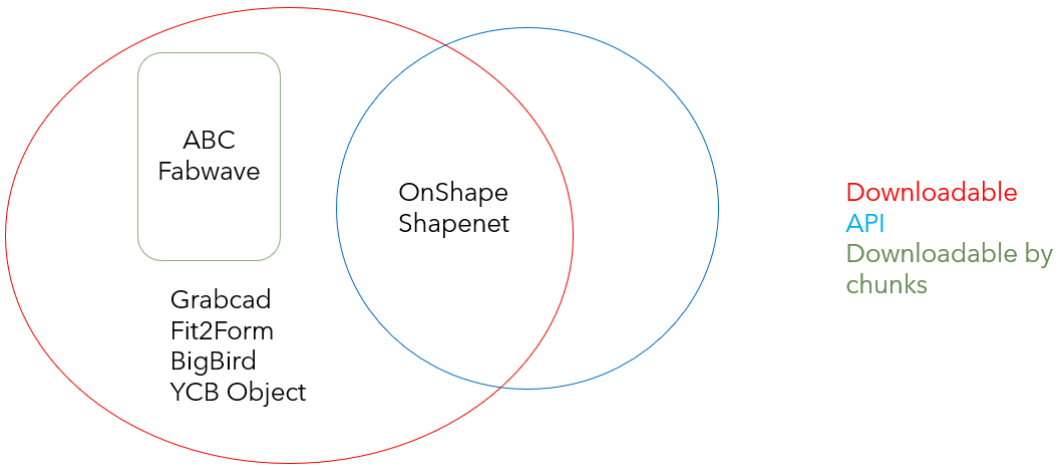

Figure 4.1: Visual classification of source database by the form of availability to their CADs.

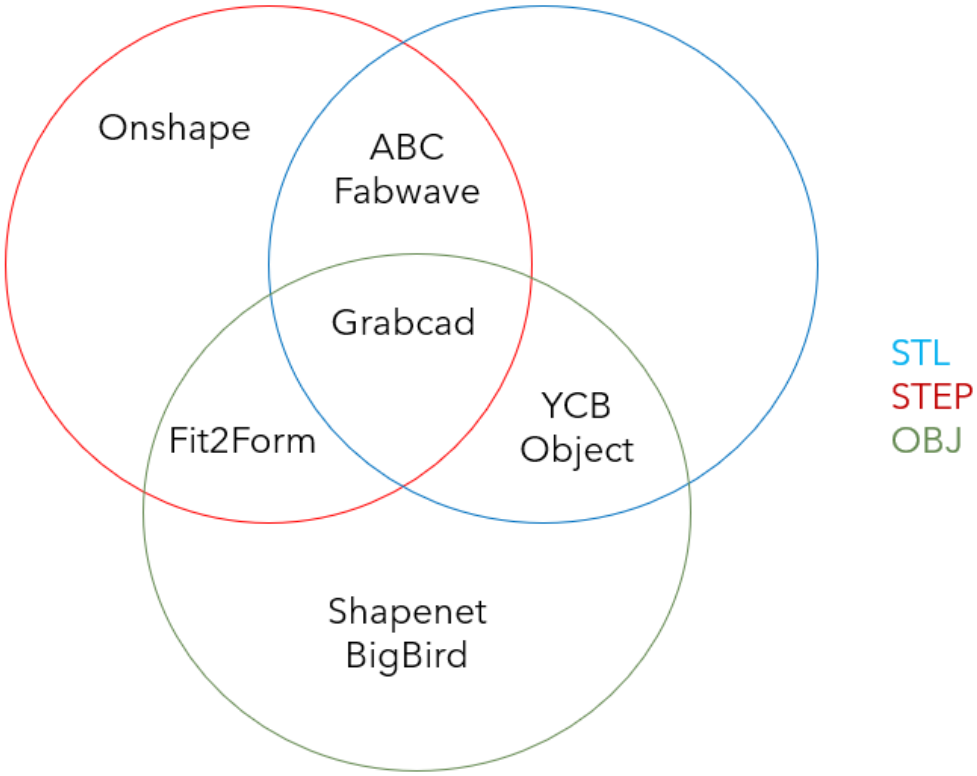

Figure 4.2: Visual classification of source databases by the format of files in it considering the most frequently used formats.

4.3 Choice of database technology

As a reminder, the requirements for database technology are a flexible schema, or even the absence of a schema, and are consistent according to the meaning given by the CAP theorem described above. But also a byproduct of the project that reduces the number of possibilities is that since the base process pipeline is done on MATLAB. Each database technology, apart from those that work with the SQL system, has its own language, a specific way of storing and rendering data, such as Mongosh for MongoDB or Cipher for Neo4j. To avoid having to develop from scratch tools that enable MATLAB to communicate with a database technology, and to be able to read, interpret and modify it, it's simpler to turn to database technologies for which MATLAB has a base of integrated tools for communicating with it. The fig. 4.3 and fig. 4.4 show database technologies that respect these characteristics. This makes it possible to reduce the pool of database technologies compared to those presented here fig. 2.14 from which to choose to those shown in fig. 4.3 and fig. 4.4: *MongoDB*, *Apache Cassandra*, *Neo4j* and the SQL technologies *MySQL*,*PostgreSQL* and *SQLite*. Given that, in the CAP theorem, Apache Cassandra does not ensure consistency and that SQL-based technologies have a fairly rigid schema, this limits us to two solutions: **MongoDB** and **Neo4j**.

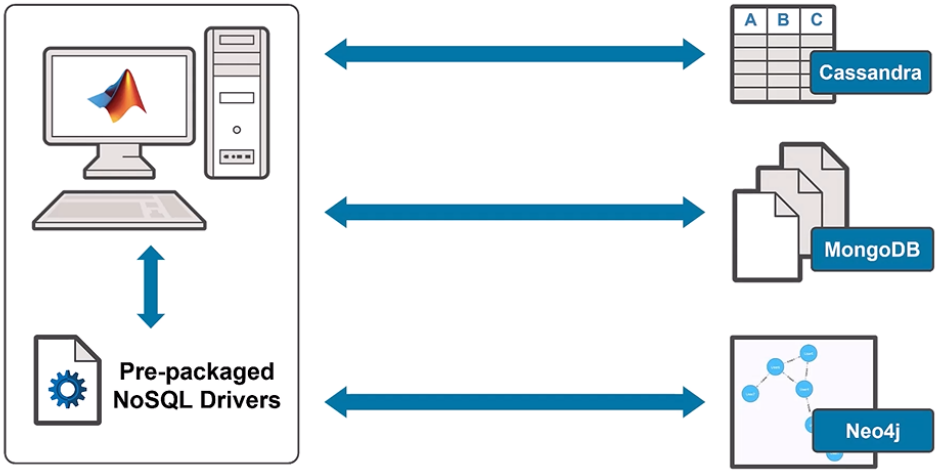

Figure 4.3: Non-SQL databases technology compatible with MATLAB.

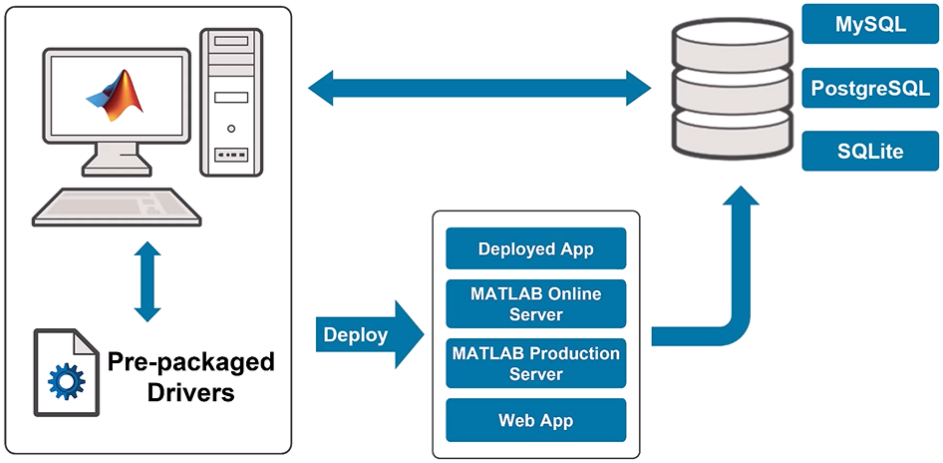

Figure 4.4: Non-SQL databases technology compatible with MATLAB.

MongoDB is a document-based database technology, fig. 2.13. The data is in the form of JSON like documents themselves organized in the form of collections, with no schema required to establish a collection. Each document can be completely different from the others.

Neo4j is a graph-based system (see fig. 2.13) made up of nodes, which are entities, and edges, which are the relationships between entities. Each node or entity has one or more properties. These properties are the data that the system stores.

And so, given the highly relational nature of Neo4j, the choice of database technology will be **MongoDB**.

Chapter 5

Implementation

The established work environment is made up of different parts, the relationships between which can be seen in the diagram Figure 11:

- MongoDB server: the technology used to store CADS references and other CADS information in the form of documents with a structure similar to JSON. It will make it easy to select CAD according to predefined criteria.
- MongoDB Compass: a graphical interface for performing operations on the database without using code, but it is preferable to make only small test modifications. More importantly, it allows you to visually observe modifications being made to the database.
- The various tools coded in python and MATLAB, in order to instantiate the various data in the database after its creation, but also to load and upload data and apply modifications to the database

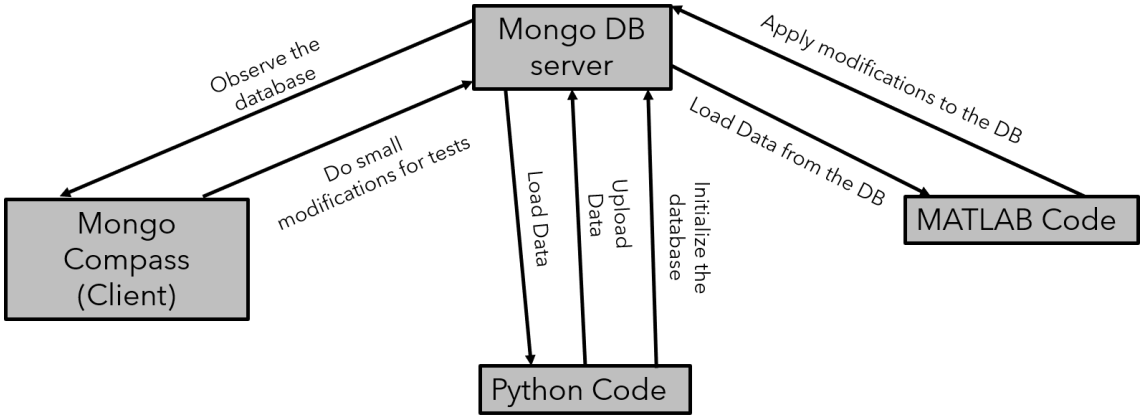

Figure 5.1: Functional architecture. It represents the different elements of the environment and how they interact with each other.

5.1 MongoDB server and functionalities

5.1.1 MongoDB Server installation

Installing MongoDB is fairly straightforward, especially locally on a computer. This is the same process as installing it on a server. It may be necessary to modify the installation port,

depending on which ports are open and available on the server. The default port is 27017, but this can be changed with the command: mongod –port {NumPort}. The appendix B shows the various installation steps for a Windows machine/server. It's best to keep the default settings given during installation, but it is possible to make modifications: not to install it as a service, to modify where logs and data will be stored, among other things. The various installation modules can be found here [28].

5.1.2 MongoDB Query

This specific part is to understand how to interact with a MongoDB-based database, which works by a Query system like SQL. But the big difference lies in the way these queries are written. SQL queries are written in the form of sentences (SELECT * FROM table) with operators such as SELECT and FROM. Unlike SQL queries, MongoDB queries are written in the form of JSON like strings, as shown in **??**. THE function for applying a query on MATLAB is *find*, so a query on matlab looks like this:

$$\text{find}(\text{collection}, \text{query filter}, \text{Projection} = \text{fields}, \text{Sort} = \text{field}, \text{Limit} = \text{l}) \tag{5.1}$$

where:

find the Matlab function to retrieve documents from the database,
collection the set of documents from which to search for the desired documents,
query filter defines the conditions that documents must meet to be included in the result set,
Projection specifies which fields should be included or excluded in the returned documents,
Sort defines the order in which the returned documents should be sorted,
Limit specifies the maximum number of documents to be returned by the query.

Query Filter

It comprises one or more key-value pairs where the keys represent field names, and the values represent the criteria against which the field values are compared. There are also operators that exist to precise the query formulation. 3 types of operator are presented here:

- **Logical Operators**. Examples:
  - $and : Joins query clauses with a logical AND and returns all documents that match the conditions.
  - $or : Joins query clauses with a logical OR and returns all documents that match any of the conditions.
- **Element Operators**. Examples:
  - $exists : Matches documents that have the specified field.
  - $type: Selects documents if a field is of the specified type.
- **Array Operators**: Examples:
  - $in: Matches any of the values specified in an array.
  - $all: Matches arrays that contain all elements specified in an array.

Examples of query filters:

Simple query: We want to remove documents whose field *tags* contain the value *"Mechanical Component"*.

$$\{\text{tags} : [\text{"Mechanical component"}]\} \tag{5.2}$$

where:

tags is the key / field name that which will be observed,

"**Mechanical Component**" is the value that is which is sought in the field.

Array search query: The *tags* field is associated with an array in the database. We want to retrieve documents whose array values *["Mechanical component", "Pipes"]* are all in the array associated with *tags*.

$$\{\text{tags} : \{\$\text{all} : [\text{"Mechanical component"}, \text{"Pipes"}]\}\} \tag{5.3}$$

Projection

It allows to shape the structure of the output documents by specifying the fields that are interesting for the task.

Example: In the document, we only want to extract the CAD references, i.e. the *filename* and *fileDirectory* fields, not even the id of the document which is included by default.

$$\{\text{filename} : 1, \text{fileDirectory} : 1, \text{id} : 0\} \tag{5.4}$$

In general, when a projection is made, all fields that are not explicitly written into the projection will be set to zero, i.e. not included. Only the id must be explicitly excluded.

Sort

It typically consists of one or more fields along with the sorting direction (ascending or descending). The way to write it is the same as for projection, except that for an ascending order we associate the value 1, and for a descending the value -1.

Writing MongoDB queries requires a longer learning curve than for SQL, not only to know the meaning but also to master the position that operators must have. That's why, when setting up this environment, some of the queries needed for the start have been programmed with obvious function names so that they can be used directly without any knowledge of MongoDB Query.

5.2 MongoDB Compass

MongoDB Compass is an interface, shown in **??** and **??**, whose use is highly recommended. Not only does it allow you to visually demonstrate what code would do on the database, it also allows you to run small tests on the database and observe the results without having to do any programming. After installing the MongoDB server, you can also easily instantiate databases and new collections. For example, in **??**, two database instances (dbSTL and mydbSTL) have been created via the interface, and the dbSTL database contains the Data and Test collections, themselves created via this interface.

5.3 Python Code

Once the MongoDB server has been installed and the database created on the server using MongoDB Compass, the next step is to place the data in the database. Certain decisions have been made for the initial setting up of the environment. Only the CAD file references will be stored, i.e. the path to access them locally or remotely. They will also all be stored in the same collection. In a future evolution of the project, different collections according to other criteria (applications adapted to different environments for example) could be envisaged. Finally, the CAD will be sorted using a system of tags. These represent different types of information about the object represented by the CAD, the name of the object, whether it is a mechanical component, etc.

The first condition, which is to store the file references, leads to a second one, which is to ensure that these references remain constant. In order to ensure this, a particular organization of the files has been created, which can be seen in fig. 5.2 and fig. 5.3.

Figure 5.2: Organising CAD files to ensure that the code runs smoothly. Files are stored in folders named after their original database (Fabwave, ABC). These folders are themselves placed in a general folder that serves as the root.

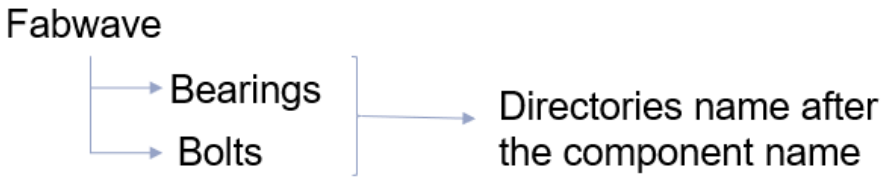

Figure 5.3: Organising files from the same original database in the folder. Some databases have a classification of their CAD and it is useful to keep it. In the case of Fabwave, the classification was made from the name of the object represents by the CAD.

Because of this organization, it is possible to find all the CAD files from the root folder path. The constraint is therefore reduced to ensuring that the root file path is invariant. This organization also has another advantage. By reading the path of a CAD file, a great deal of information about it can be deducted, including its origin. Also, if this root database provides a classification of CAD files, information about this classification can be retrieved by reading the path and used as tags.

At initialization of the database, the fields in a document that store a the reference of a CAD will be: **originDatabase**, **tags**, **fileDirectory**, **filename**. An important point is that MongoDB documents are represented in the form of Python dictionaries. So to initialize the database with documents that have the fields created above, you need to create a list of Python dictionaries with the keys of each dictionary corresponding to the document fields.

The fig. 5.4 shows a UML of the object created for this Python code.

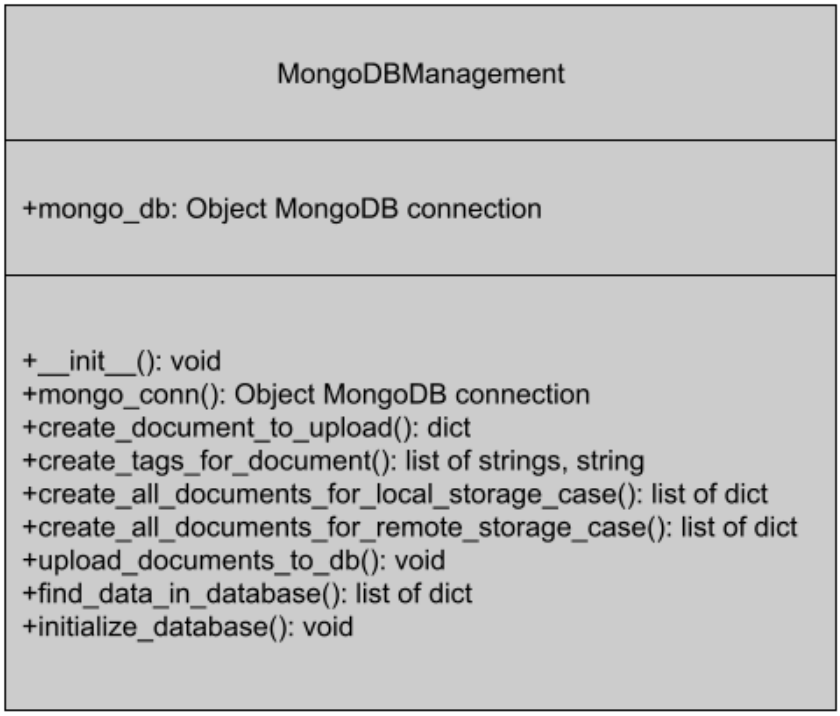

Figure 5.4: UML representation of the MongoDBManagement object for python code. It contains a public attribute (represented by the + sign) mongo_db, which is an object that enables connection to a MongoDB database. It also contains 9 public methods, each of which returns a certain type of variable, void and if the method returns no variable.

The purpose of this MongoDBManagement object is to establish a connection with the previously installed database, to browse the files whose organization has been described here, and through the path fig. 5.2 to this file to extract the name of the database from which it originates, as well as the characteristics fig. 5.3 added in the field *tags* to form the documents that will be stored in the database. Finally, it allows you to load files from the database, as well as upload them.

The **mongo_db** attribute is used to store the object used to connect to a MongoDB database.

Each method plays a role in achieving the objectives given for this object:

- **__init__**(mongo_hostname, mongo_port, mongo_dbname): is the object's constructor, and returns nothing (void). Above all, it instantiates the database connection and therefore takes as parameters the IP/address to access this database *mongo_hostname*, the port to connect to this database *mongo_port* and the database name *mongo_dbname*. It does this by calling the **mongo_conn** method.

- **mongo_conn**(hstname, prt, db): takes as parameters the IP/address to access this database *hstname*, the port to connect to this database *prt* and the database name *db* to test and instantiate a connection to a MongoDB database.

- **create_document_to_upload**(original_database, file_directory, filename, tags): takes the key values defined above, original_database for *originDatabase*, file_directory for *fileDirectory*, filename for *filename*, tags for *tags* as parameters and outputs the dictionary corresponding to a document in MongoDB.

- **create_tags_for_document**(actual_dir): allows you to deduce the source database

from the file path *actual_dir*, and also to define the tags to be given to the CAD. Its construction is such that, for each source database that has been previously studied, it is possible to add in the tags which type of element is contained in the original database. For example, in the fig. 5.3, Fabwave CAD represents all mechanical components, which can be added to the initialization tags. In addition, Fabwave CAD was classified according to element type, so it is possible to read from the path of a Fabwave CAD what type of mechanical component it is (e.g. Pipes) and add it to the tag list.

To ensure that this method works properly, two things are required: for each new source database, a condition must be added to the function to identify whether a CAD belongs to it, and if the CAD path can be used to obtain information on the type of elements, there must be a variable for each source database listing the type of elements that can be identified via the file path.

- **create_all_documents_for_local_storage_case**() and

  create_all_documents_for_remote_storage_case() serve the same purpose, create a list of dictionaries that will be uploaded to the database as database documents. The principle is to browse the root directory fig. 5.2 until a CAD file is found. When the path of a CAD file is found, the **create_tags_for_document** method is called to determine the tags and source database of the document, and then all these elements are given to the **create_document_to_upload** function to create the dictionary representing the document to be uploaded.

  The difference between the two lies in where these files are stored and the python libraries for accessing them. If these files are stored on the computer running the code, the os library is used. Otherwise, the pysftp library is used to connect remotely to the machine where the CAD files are located and to browse the root directory.

- **upload_documents_to_db**(posts, dir_name): upload one or more documents *posts* to the database , in a specific collection *dir_name*.Normally, python drivers for MongoDB have functions to do this job, but this one has a function for uploading one document and another for uploading several. This method makes it possible to agglomerate the two by checking in input whether *posts* is a dictionary list or a single dictionary.

- **find_data_in_database**(query, collection, fields_to_include={}, isremote=False, localpathtodownload=“”): load the datas from a specific *collection* that respect the *query* conditions. We can precise which fields of the documents we want (do a projection) (fields_to_include). Also if the datas are stored remotely *isremote=True*, they are downloaded to the specific local path to download *localpathtodownload*.There’s also a function for loading data from the database. The contribution of this function is that if the CAD files are stored on a machine other than the one on which this code and the rest of the pipeline will be run, the CAD files are automatically downloaded to the local machine.

- **initialize_database**(dir_name, islocal=True): depending on whether the CAD files are stored locally or not *islocal* (on the machine on which this code is run or not), it calls the appropriate method to create the list of documents to be uploaded and uploads all these documents to the specified database and collection *dir_name*.

5.4 MATLAB Code

As the core of the pipeline is written in MATLAB, this part is the main tool for establishing communication between the database and the pipeline. Since MATLAB has tools for communicating with a database, at first sight these tools could be used without overlaying to achieve the desired result. However, between writing the query filter, and the way MATLAB imports data from the database (which we'll look at below), it's necessary not only to add overlays to MATLAB's existing methods, but also to have code that enables the re positioning step from fig. 3.1. This code consists of 5 objects: **QueryManager**, **JSONWriter**, **ImportData**, **AutomatePositionning** and **AddBoundariesLimitsToDataInDB**.

QueryManager is used to write MongoDB Queries. In MATLAB, queries must be passed in the form of strings of characters in JSON format.For example, the query filter eq. (5.3) would be manually written as:

$$\text{"{"tags" : {"\$all" : ["Mechanical component","Pipes"]}}"} \tag{5.5}$$

But writing these strings by hand can be quite time-consuming and syntax-sensitive, i.e. knowing when to use braces {} rather than brackets []. A query filter is written with the same type of variable as a document that has been loaded from a MongoDB database, i.e. a dictionary which ensures case-resistance.A MATLAB structure works on the same principle as a Python dictionary. It's a variable with keys, each of which is associated with a value. Since it operates on the same principle as a query filter, it is possible to use a structure to represent a query filter. Furthermore, there is a function for converting a structure into a character string. However, for the structure, which is made up of keys with associated values, the field/key names cannot contain special characters ($@_ and others). However, we saw earlier that a field can be an operator and therefore contain the $ character. The solution is to write a query as a structure in MATLAB, removing the $ character from the operators and then passing it to a function which will convert it to a JSON string and add the missing character. As a result, the QueryManager object has as an attribute the possible operators for recognising them and a method which takes as a parameter the structure which represents the $ character. So to write this filter query, you first need to write a structure on this model:

$$\text{{"all" : ["Mechanical component","Pipes"]}} \tag{5.6}$$

where:

all is the operator without the sign $ and the key of the structure

Then encapsulate this structure in another, this time with a key that corresponds to the field *tags* in the document:

$$\text{{tags : {all : ["Mechanical component","Pipes"]}}} \tag{5.7}$$

This structure will then be passed to the QueryManager object, which will first transform it into a JSON like string in the form of a char array:

$$\text{"{"tags" : {"all" : ["Mechanical component","Pipes"]}}"} \tag{5.8}$$

Finally, it will identify operators that are between "" without their $ sign, and add this sign at the appropriate position. In this example, this results in the query filter eq. (5.5).

ImportData is an object for managing the import of data from the database. An import function already exists for loading data, except that the type of output may vary depending on the data. If all the data has exactly the same fields, it is imported in the form of an array

of structures, otherwise it is loaded in the form of a cell of structures. However, cell and array are used in different ways. The aim is to have the most flexible database model possible, so some data will have a similar structure and others will not. The **ImportData** object therefore exists so that we don't have to worry about what type of variable we're going to get as output, and so that we can put everything in a cell array of structures, allowing to keep fields that may not be present in all documents. In addition to what we saw in the previous paragraph, given that each piece of data is imported in the form of a structure, we need to make sure upstream that the data fields do not contain any special characters.

AutomatePositionning is used to reposition the object in relation to the fingerbase. A simple repositioning, which does not necessarily take the context into account, has been decided as in Figure 1. This positioning just consists of centering the object in relation to the fingerbase and aligning their main axis (the largest axis) and making it protrude slightly on the z axis. Finally, a json file is generated with the CAD reference and the operations required for positioning.

JSONWriter simply takes a structure and writes it in the form of JSON to a json file. It is used in particular in **AutomatePositionning** to generate the file with the CAD reference and the positioning operations.

AddBoundariesLimitsToDataInDB is used here to add the CAD dimensions to the data in the database. Having the approximate dimensions of the object can be used as filter criteria for working specifically with a particular size of object. This is what **AddBoundariesLimitsToDataInDB** does: it loads all the data, calculates the object's boundaries on the X, Y and Z axes, adds a field boundaries to the data in the database and updates it with the calculated values.

In addition to these objects, there are two files which bring together the objects mentioned in an algorithm for which the aim at the end is to have json files with the references of the CAD files and the positioning operations which will be supplied to the rest of the pipeline:

- Main_import_remote_datas.m
- Main_import_local_datas.m

The difference between the two is the ability to process files stored locally and remotely. In particular, when files are stored remotely, they are downloaded to the computer and the reference that will be stored afterwards is the path to access the file.

Chapter 6

Evaluation

There are 3 points to checking that the environment is working properly:

- Ensure that the server is correctly installed, which can be done either via the command line or the MongoDB Compass interface.
- Ensure that the python code is correctly connected to the server and that the database is initialized via this code. Initialization can simply be checked via MongoDB Compass by verifying that the number of CAD references imported by Python is equal to the number of data on the server. MongoDB Compass makes it easy to see the total number of data items in a collection.
- Make sure that the MATLAB code is correctly connected to the server, that for a given filter the number of data items collected is correct and that modifications made by the MATLAB code on the database are correctly applied.

Most tests and verification are limited to local work, i.e. the MongoDB server and the data are on the same computer, under a Windows operating system. Once the server has been installed, it defaults to port 27017, and it is possible to connect via this port. In MongoDB Compass, the default connection address is entered directly if the software detects the presence of MongoDB on the computer. fig. 6.1 is the windows that MongoDB Compass shows that the connection to the server has been established. For this evaluation, a CAD database with a collection of files has been created, fig. 6.1.

6.1 Python code test: Initialization of the database

This initialization will be made using only the Fabwave data for classification purposes, which will make it possible to automatically assign initial tags. Verification will be based on a comparison between the number of documents created in Python and the number of documents currently uploaded to the database. This will also allow to test the data upload function on Python. The fig. 6.2 and fig. 6.3 show that all the documents created in Python have been imported. Also, the process is very fast (<1sec) for 3693 data.

This fig. 6.3 shows the form in which documents are stored at initialization. They are stored as Binary JSON. BSON (Binary JSON) is a binary-encoded serialization of JSON-like documents. For each document, we have keys / fields associated with a value. Also, when an ID is not specified when the document is uploaded, a unique identifier is automatically assigned to the document. After initialization, a document/data is in the form of a JSON with fields:

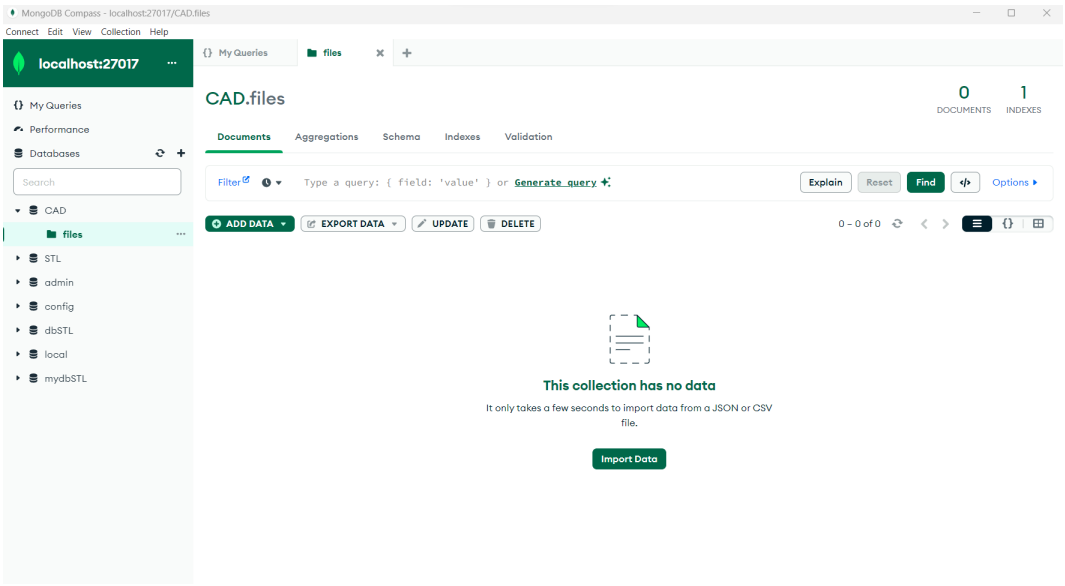

Figure 6.1: New database CAD with collection files created via Mongo Compass.

```
mongo_db_management = MongoDBManagement(mongo_hostname=hostname, mongo_port=port, mongo_dbname=dbname)
nb_files = len(mongo_db_management.create_all_documents_for_local_storage_case())
print(nb_files)
mongo_db_management.initialize_database(dir_name="files")
```

Figure 6.2: Code for evaluation. The object that contains the different tools are created and after that the number of documents is evaluated to compare with the number of documents actually implemented in the database.

- **_id**: unique identifier for each document
- **originDatabase**: the CAD file's original database name stored as a string
- **tags**: array of strings characterizing the object represented by the CAD file.
- **fileDirectory**: Folder in which the corresponding CAD file is stored
- **filename**: CAD file name

This configuration is not immutable, as we'll see when we test the MATLAB code. It is always possible to identify a document in order to modify it, by adding new fields or modifying existing fields (adding new tags to the list of existing tags, for example).

6.2 Python Code test: Data loading from database

For the data load, the example of 'Pipes', which is a component category in Fabwave, will be used. This category contains 95 files. Using the query tags: "Pipes", documents with the pipes tag can be filtered. In fig. 6.4, the tests show that the load test gives the same number of documents for this query and that the objectId of the first document retrieved is the same.

6.3 MATLAB Code test: Updating documents in the database

It works in the same way as Python as far as loading data from the database is concerned. The final tests concern the ability to modify the data in the database and the generation of the

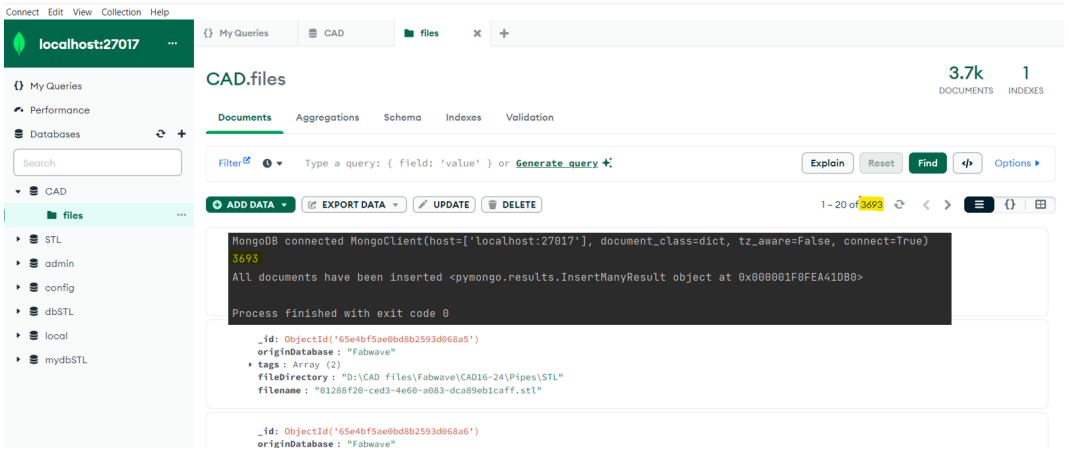

Figure 6.3: Observation of the comparison of documents created and in Python and uploaded on the database (the interesting numbers are highlighted).

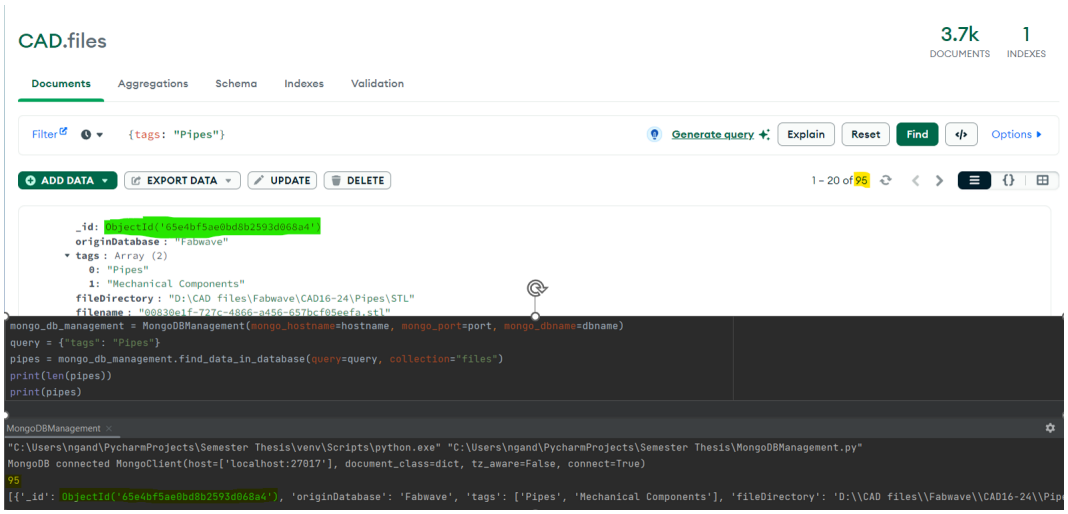

Figure 6.4: Test of data loading via Python. The number of data filter for the query {tags: “Pipes”} is the same).

Figure 6.5: Comparison of two objects before and after adding the boundaries to the documents.

JSON files that will be given to the remaining part of the pipeline. The ability to modify the data in the database will be tested using the **AddBoundariesLimitsToDataInDB** functions, the aim of which is to load all the data in the database, calculate the object boundaries and update the corresponding boundaries for each document.

By its very nature, the process is a little slower as each CAD file has to be loaded into MATLAB to retrieve the boundaries via functions. But the modifications are applied for the better and correspond well to the CAD when the boundaries are calculated directly by loading the CAD file without going through the database. Finally, the function for generating JSON files with information such as the path to the file and the operations required for the positioning phase, allows you to set parameters for operations that would be entered manually to replace those calculated automatically. And so two cases are possible, no operation is passed as a parameter and in this case operations as defined above will be determined.

Chapter 7

Discussion

The environment created as part of this project must meet certain criteria:

1. Setting up a database with flexible or no scheme
2. Store CAD file references
3. Set up tools to communicate with this database. They must enable you to upload data to the database, load data from the database and modify data in the database.
4. These tools must be functional, regardless of whether the data is stored on the machine on which the code (tools or pipeline) is running, or whether these CAD files are located remotely.

The tests in the previous chapters have shown that these objectives have been achieved. Some of the code was also written in Python, with the intention of potentially using it to code the pipeline. But as the project pipeline is still on MATLAB, it's this code that has the most functionality. For example, the **AddBoundariesLimitsToDataInDB** and **AutomatePositionning** objects use MATLAB libraries used by the rest of the pipeline, but also functions employed by the pipeline. So, since the functions used are the same, we keep the same interpretation of the CAD files, and the approximations made by these functions remain the same. If these objects had been made in Python, there would be differences in the logic of the functions which would distort the results.

The MATLAB part has often been the knot and the limiting factor in certain choices that have been made. The main type of variable for representing the format of MongoDB documents on MATLAB is the structure. This type of variable will be the one under which the database data will be loaded. And the structure has limitations, particularly on field names, which cannot have special characters, so the names of the fields to be defined must obey this constraint. But the real limitation comes from another point. On MongoDB, when documents are uploaded, a unique identifier is assigned to each document if this is not done in advance. It can be useful to have this identifier even from a code point of view. The purpose of the field use case for example would be to encode an industry-specific task requiring the object to be in a specific position that cannot be modified. This can be achieved simply by using a string to uniquely encode the task designation. If different use cases, indicating different positions for the same CAD file, are defined, this leads to different documents which are similar in many respects except for the identifier, use case and position. To avoid using the logical AND operator on the use case and positioning to uniquely designate one of the documents under consideration, the simplest solution is to use the identifier. This is represented by the "_id" field. Although it is loaded with the document in a structure variable, in MATLAB this identifier is inaccessible because of its name.

Most CAD gave satisfactory results for the generation of their process base. But there were also special cases where the process base was not precise enough or the CAD file was read as being empty, even though, thanks to different software, these CAD files did contain an element. In MATLAB, to read the CAD file, the various points of the mesh are taken to form a point cloud in MATLAB. This is related to the CAD meshing and the density of vertices, and therefore proportionally to the number of facets used to encode the object. In fact, it can happen that, at the surface of the object, there are not enough facets used to encode the object, and therefore not enough vertices that will be sampled to form the point cloud used later. This would require algorithms to re-mesh the CAD, or else to check whether things are better with a switch to python.

Finally, other possibilities for data storage have been explored. Depending on the size of the data, it is possible to store files directly on MongoDB (if < 16 MB) or via GridFS (if $>$ 16 MB). GridFS is a MongoDB module that can be used to store larger files by splitting them into several chunks, creating a reference document and several other documents that store the chunks of the file and refer to each other in order. However, this aspect of MongoDB is not present on MATLAB, requiring the use of external drivers such as Java. Also, accessing these files requires working with streams either to download the said files or to be able to read the chunks directly in the code, which is far too complex, especially on MATLAB. This requires not only importing drivers from other programming languages, often Java, and using them with MATLAB, as MATLAB does not support this MongoDB extension. This results in a transcription of the code that should be used on Java into MATLAB. After some testing, this transcription is proving to be arduous. In addition, more research needs to be done on how MATLAB interprets streams, and how to decode with MATLAB a stream of binary data representing an STL file. So, although storage by reference works very well, it requires precise organization of the files, but simplifies the work at code level. Storing directly on MongoDB would make it easier to manage the files, since everything would be managed by MongoDB, but would make the task more complex in terms of code.

Chapter 8

Conclusion and Outlook

Fingertip design has long been a key issue in industry, particularly in the development of machines used in assembly lines for pick and place work. The aim is to improve the adaptability of these machines to new products, and this means automating the design of the end effector so that it can be perfectly used with the new product. These methods enabled the development of processes to automate fingertip design, involving a balance between force and form closure. These methods were also increasingly based on CAD, i.e. 3D modeling of the objects to be manipulated. Machine learning has also played an increasingly important role in this process, from helping to generate grasp sets to the possibility of directly generating the appropriate fingertip. The development of data-driven methods such as these has led to an increase in the number of platforms storing these CADS, some of which are accessible to all. As a result, in order to develop a data-driven method, it is necessary to have a suitable working environment, centered on a database for which various more or less flexible technologies exist.

This work has enabled us to set up such an environment by choosing MongoDB for its flexibility in particular, but also for its compatibility with MATLAB. The CAD files used are of the STL extension because of its simple geometry and its very frequent presence in different databases. This environment is divided into four parts. The MongoDB server itself and its graphical interface MongoDB Compass, python code and MATLAB code. The python code, in addition to being able to load and upload documents to the database, is used to initialize the database, i.e. to upload all the documents containing the references of the CAD files and various initial tags. The MATLAB code is used to update documents already in the database and also to load data from it. It also uses load documents to create JSON files that provide information about the CAD reference and certain operations that will be carried out on it to the rest of the pipeline.

Although it's functional for a start, the work done on this environment still shows areas for improvement that can be addressed in future work. For example, maintaining the constancy of CAD file references is still based on trust in the users in the case of teamwork, and so this area is still vulnerable to poor handling. One solution would be to have a very simple database, i.e. an array that would store for each file name the path it should have to the root directory. And to complement this, a two-part code: one would go through all the files with the right orgarnization to record for each file the right path to access it, the other would go through the root directory and put the files in their place if modified. This solution doesn't prevent human errors linked to the organization used, but as long as the error of executing the code's function to record the correct path is not made in a haphazard way, it at least ensures the correction of bad manipulations of file organization.

In addition, only a few queries' operators essential to this project have been taken into account. For the future, it would be advisable either to extend the list of operators taken into account, to learn as much as possible about writing queries or to switch to a system like

python where writing queries is simpler. Finally, the most pressing problem is the re-meshing of CAD files. We need to find algorithms that can automate this phase while retaining the basic essence of the file.

Appendix A

GOFD presentation

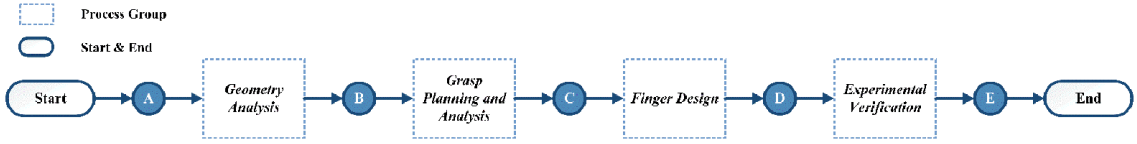

Figure A.1: Overview flowchart of the most recent method of Honarpadaz et al the Generic Optimized Finger Design taken from [16].

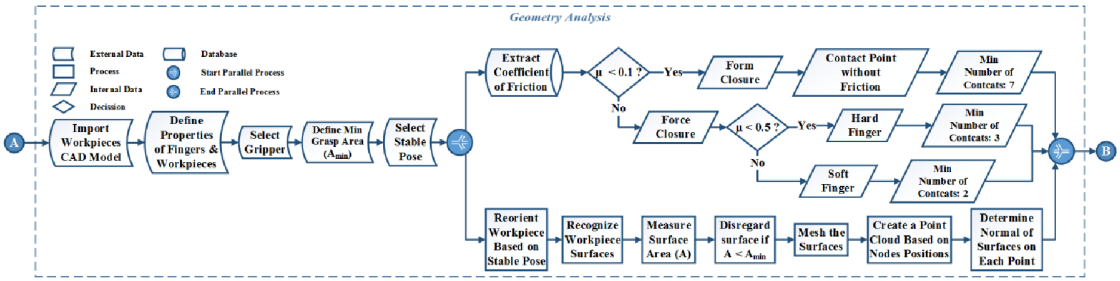

Figure A.2: Flowchart of the Geometry Analysis process for the GOFD taken from [16]. Compare to fig. 2.8, It adds a surface analysis to model only large enough surfaces where it is conceptually possible to grap the object.

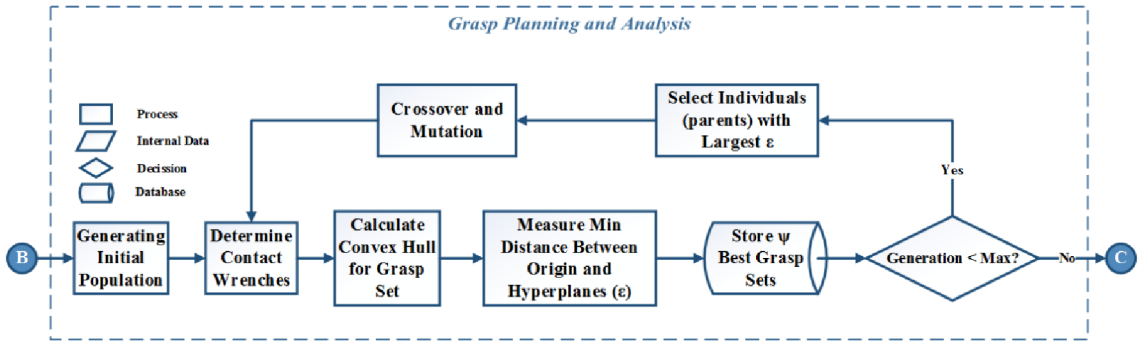

Figure A.3: Flowchart of the Grasp planning and analysis process for the GOFD taken from [16]. Compare to fig. 2.8, It adds a procedure to check that you've generated as many grasp sets as possible and that you've generated the best possible grasp sets. This involves Crossover procedures, which combine the finger positions of two parent grasp sets to create a new child grasp set, and Mutation, which applies random changes to the child grasp set to explore more possibilities.

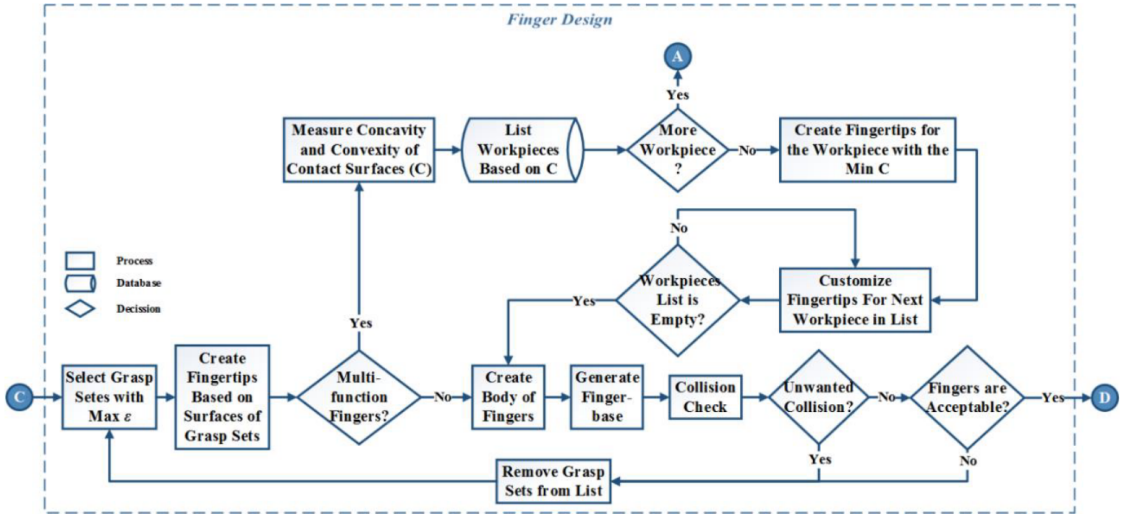

Figure A.4: Flowchart of the Finger design process for the GOFD taken from [16]. Compare to fig. 2.8, the case where an assembly made up of several objects / Workpieces is taken into account in order to generate the grasp sets of all the workpieces in the assembly before generating the fingers at the same time and checking that there are no collisions between them.

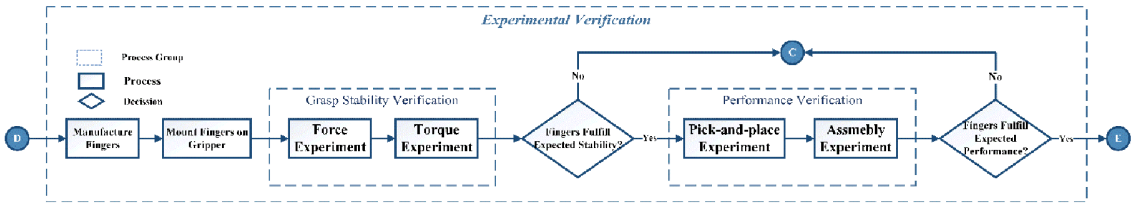

Figure A.5: Flowchart of the Experiment verification process for the GOFD taken from [16]. A new process has been added to the GAFD fig. 2.8 to check whether the fingers generated work in practice.

Appendix B

MongoDB installation

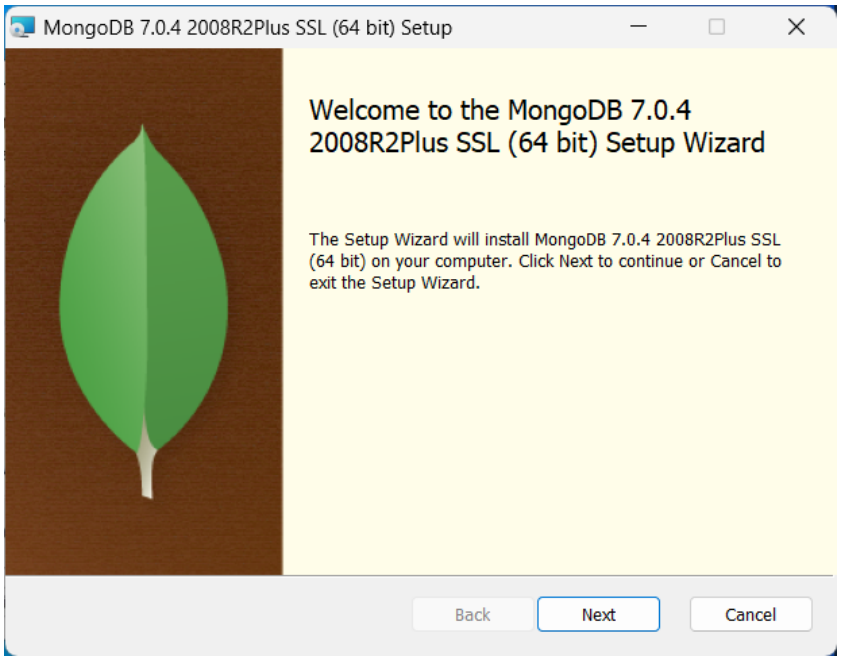

Figure B.1: Step 1 of the MongoDB installation. After downloading the executable and running it, this window will open.

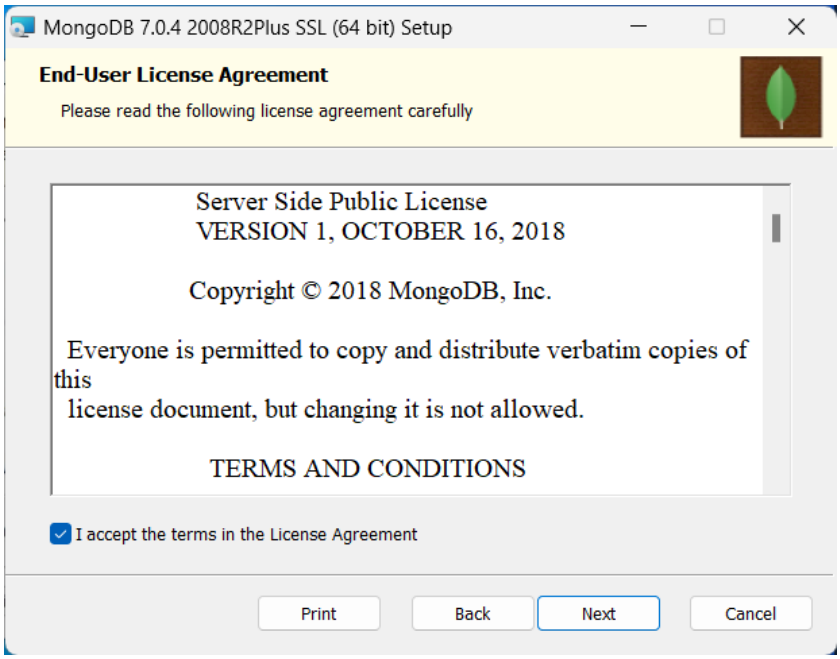

Figure B.2: 2nd step of the installation, read and accept the terms of use.

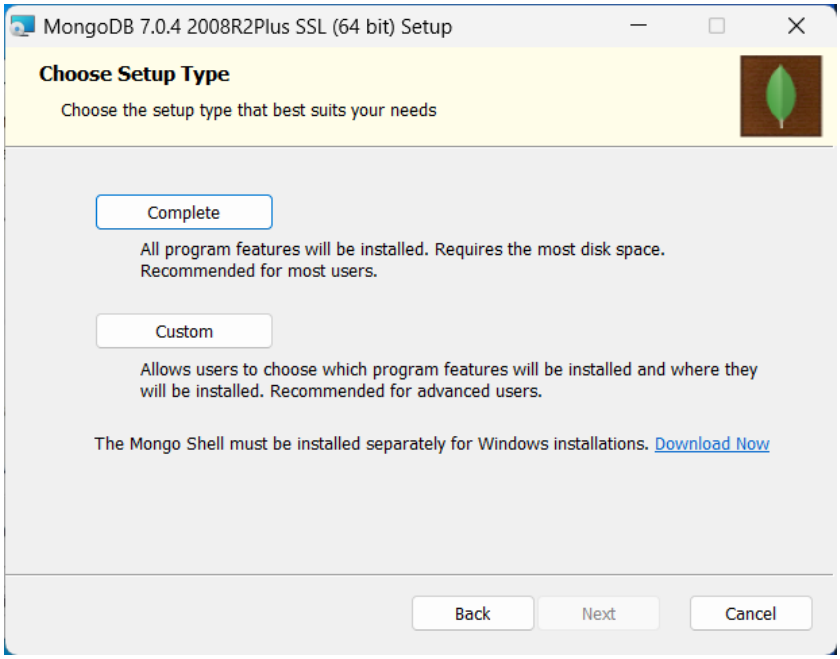

Figure B.3: 2nd step of the installation, read and accept the terms of use.

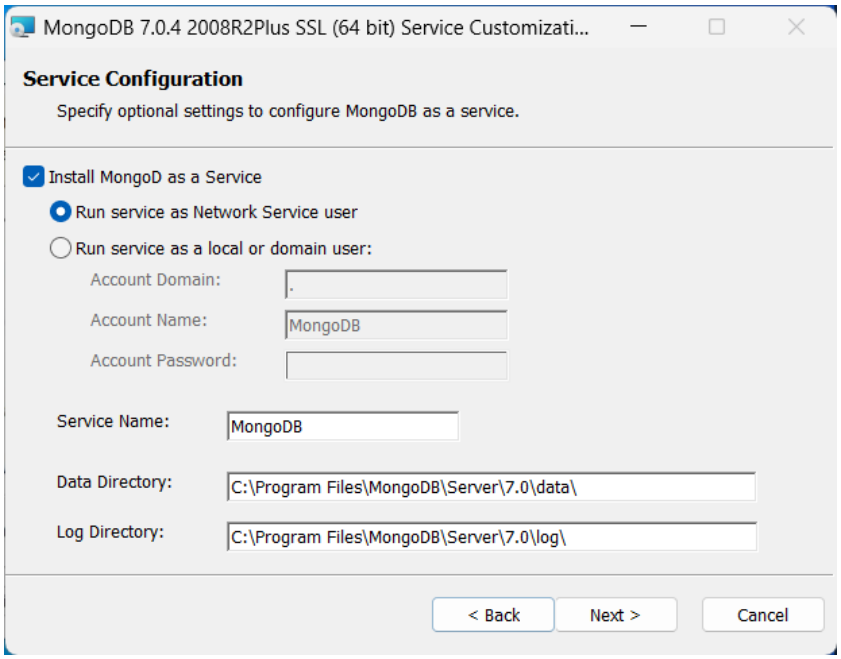

Figure B.4: Step 4: Choose MongoDB installation mode. If installed as a service, it will start when the machine/server is launched and run in background. Otherwise, it will have to be started manually each time and run in foreground. In addition, the terminal window must be kept open at all times to keep MongoDB active. By default, and especially when working with a group, it's best to install it as a service. When you run a service as the Network Service user, the service runs under the security context of the built-in Network Service account. Using the Network Service user is often a good choice for services that need network access but don't require extensive access to system resources. Running a service as a local or domain user means the service runs under the security context of the specified user account. It's often used for services that require access to resources beyond what the Network Service account can provide, such as accessing network shares, databases, or other systems.Except in special cases, keep the default run as a Network service user.

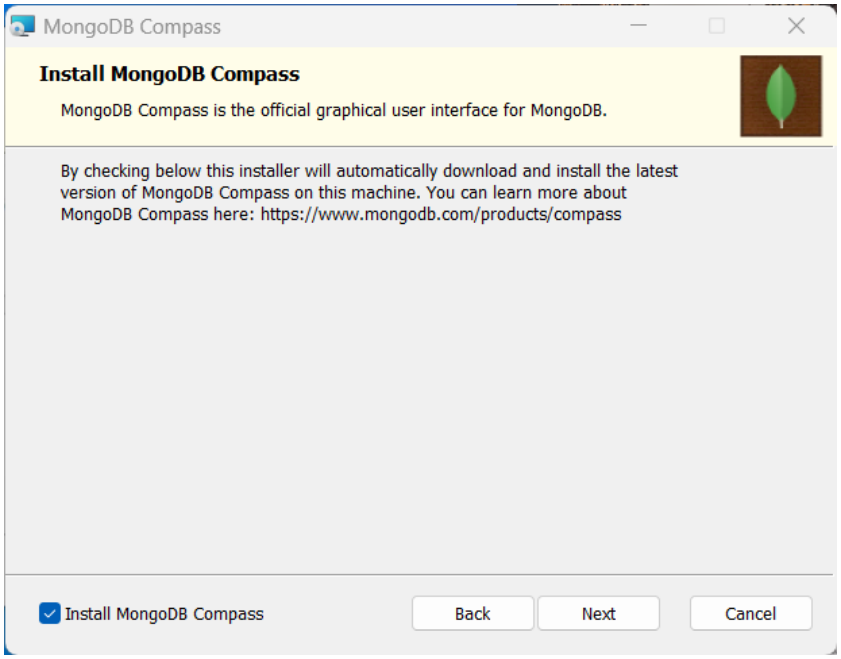

Figure B.5: Step 5: install MongoDB Compass automatically or not.

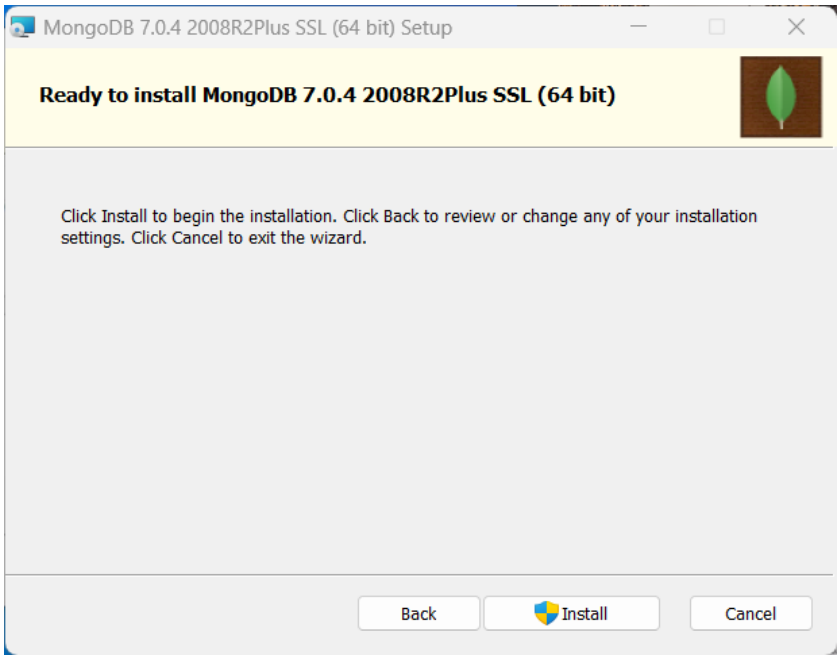

Figure B.6: Step 6: Begin the installation.

Appendix C

UML representation of MATLAB objets in code

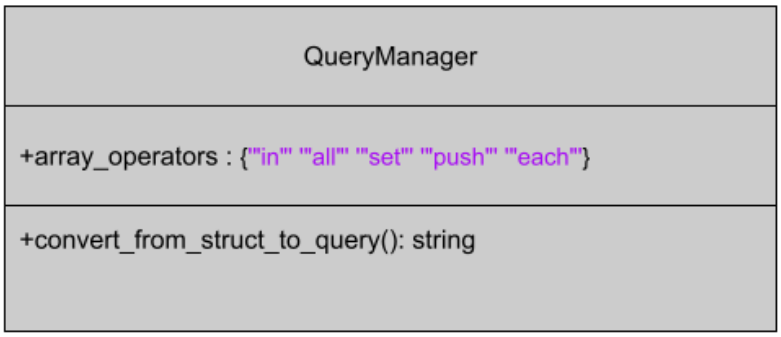

Figure C.1: UML representation of the QueryManager object. A method whose goal is to take a structure representing a query filter, transform it into a JSON-like string, and add the $ character to the query filter operator.

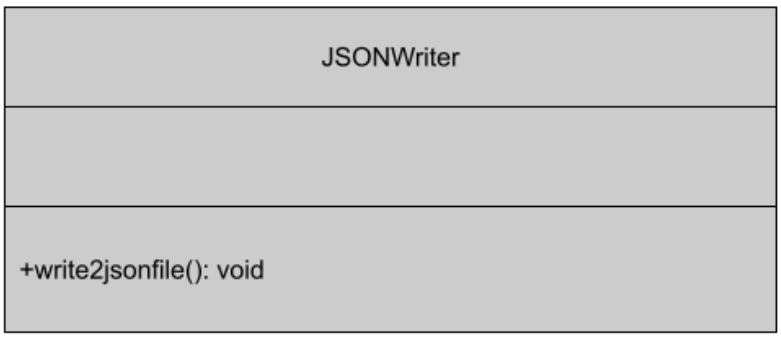

Figure C.2: UML representation of the JSONwriter object. The method allows you to take a structure and write it as a JSON-like string in a JSON file.

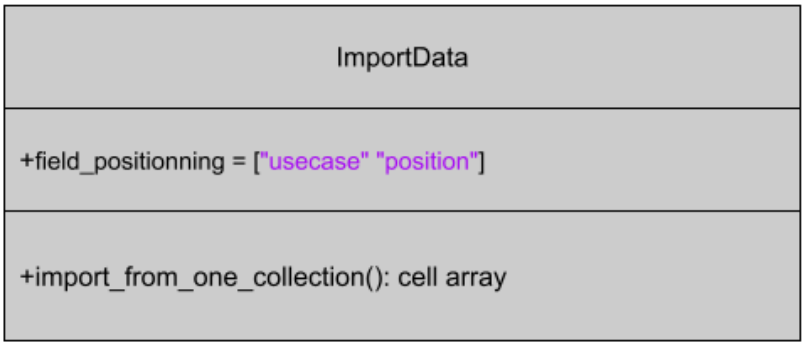

Figure C.3: UML representation of the ImportData object. This method is an overlay of MATLAB's existing method for importing data from a database collection. The problem with this existing method is that the type of variable given as output can vary between a struct array and a cell array (depending on whether the load documents in the database have the same fields or not). This function ensures that the output variable type is always a cell array.

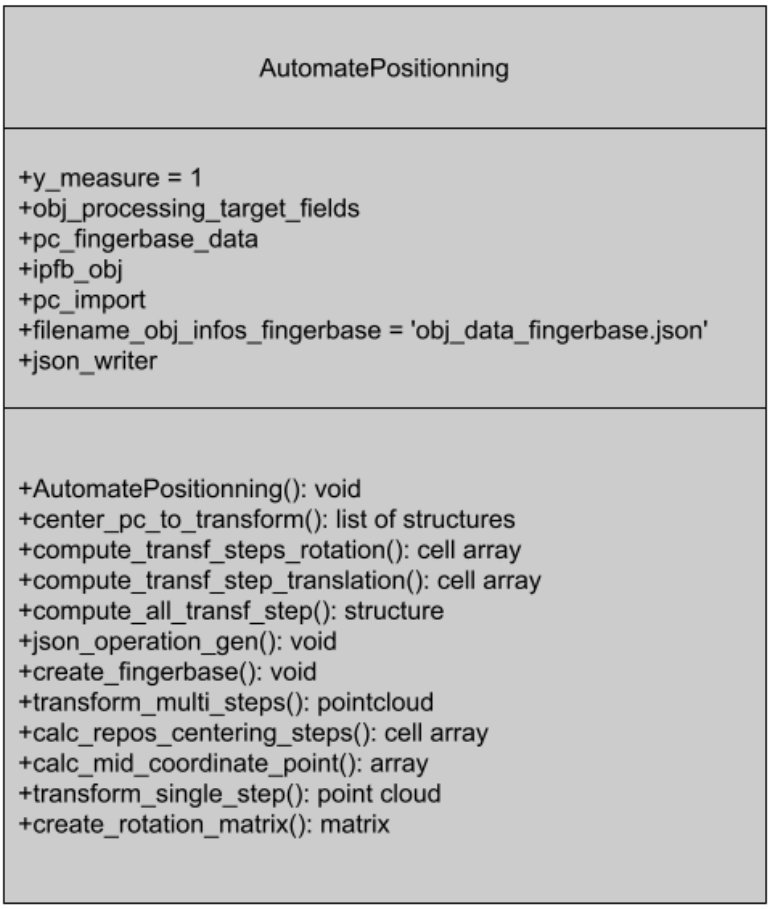

Figure C.4: UML representation of the AutomatePositionning object. This method allows you to modify the object's position, placing it at a certain position relative to the fingerbase, as in the process for the base process. create_fingerbase() loads data from the fingerbase's CAD file. json_operation_gen() generates a JSON file containing the modifications to be made to the object's CAD so that its position is that desired. To do this, this method first calls calc_repos_centering_steps(), to align the object's center with that of the fingerbase, then compute_transf_steps_rotation() to calculate the rotations and finally compute_transf_step_translation() for the translations to be performed on the CAD. Each transformation is first actually applied to the CAD via transform_multi_steps() before any other modification is applied.

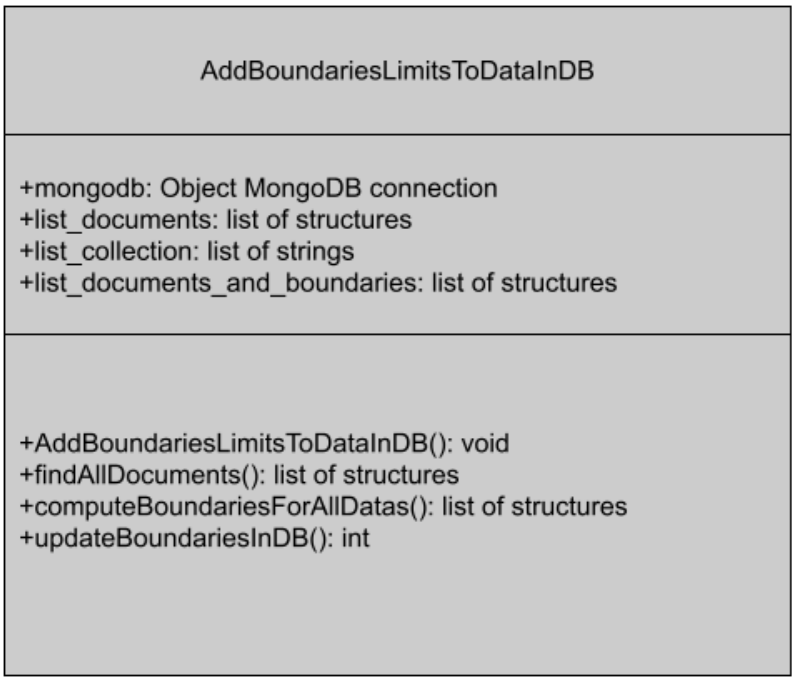

Figure C.5: UML representation of the AddBoundariesLimitsToDataInDB object. TIt loads all the data and references of all the CADs currently stored, calculates their boundaries on the 3 spatial axes and adds this boundary information to the corresponding documents in the database.

Bibliography

[1] A. Singh, J. Sha, K. S. Narayan, T. Achim, and P. Abbeel. “BigBIRD: A large-scale 3D database of object instances”. In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 509–516. ISBN: 1050-4729. DOI: 10.1109/ICRA.2014.6906903.

[2] Ai, R., Pilapil, M., Shi, R., Badugas, J., and Cheng, S. “Computer-Aided Design & Applications, 20(1), 2023, 44-55 © 2023 CAD Solutions, LLC, http://www.cad-journal.net 44 Design and Optimization of an Adaptive Robotic Gripper using Finite Element Analysis and Generative Design”. In: *Computer-Aided Design and Applications* (2022), pp. 44–55. DOI: 10.14733/cadaps.2023.44-55.

[3] Balan, L. and Bone, G. M. “Automated Gripper Jaw Design and Grasp Planning for Sets of 3D Objects”. In: *Journal of Robotic Systems* 20.3 (2003), pp. 147–162. ISSN: 0741-2223. DOI: 10.1002/rob.10076.

[4] Bharadwaj, A., Xu, Y., Angrish, A., Chen, Y., and Starly, B. “Development of a Pilot Manufacturing Cyberinfrastructure With an Information Rich Mechanical CAD 3D Model Repository”. In: *Proceedings of the ASME 14th International Manufacturing Science and Engineering Conference 2019*. New York, NY: The American Society of Mechanical Engineers, 2019. ISBN: 978-0-7918-5874-5. DOI: 10.1115/MSEC2019-2882.

[5] Bicchi, A. and Kumar, V. “Robotic grasping and contact: a review”. In: *IEEE international conference on robotics and automation, ICRA 2000*. Washington: IEEE, 2000, pp. 348–353. ISBN: 0-7803-5886-4. DOI: 10.1109/ROBOT.2000.844081.

[6] Brown, R. G. and Brost, R. C. *A 3-d modular gripper design tool*. 1997. DOI: 10.2172/437676.

[7] Calli, B., Singh, A., Walsman, A., Srinivasa, S., Abbeel, P., and Dollar, A. M. “The YCB object and Model set: Towards common benchmarks for manipulation research”. In: *2015 International Conference on Advanced Robotics (ICAR)*. IEEE / Institute of Electrical and Electronics Engineers Incorporated, 2015, pp. 510–517. ISBN: 978-1-4673-7509-2. DOI: 10.1109/ICAR.2015.7251504.

[8] Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. *ShapeNet: An Information-Rich 3D Model Repository*. URL: https://arxiv.org/pdf/1512.03012.

[9] Ding, D., Liu, Y.-H., and Wang, M. Y. “On computing immobilizing grasps of 3-D curved objects”. In: *Proceedings 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation, 2001*. IEEE / Institute of Electrical and Electronics Engineers Incorporated, 2001, pp. 11–16. ISBN: 0-7803-7203-4. DOI: 10.1109/CIRA.2001.1013165.

[10] Ferrari, C. and Canny, J. “Planning optimal grasps”. In: *Proceedings*. Nice, France: IEEE, 1992, pp. 2290–2295. ISBN: 0-8186-2720-4. DOI: 10.1109/ROBOT.1992.219918.

[11] Friedmann, M. and Fleischer, J. “Automated Configuration of Modular Gripper Fingers”. In: *Procedia CIRP* 106 (2022), pp. 70–75. ISSN: 22128271. DOI: 10.1016/j.procir.2022.02.157. URL: https://syncandshare.lrz.de/download/MlV5Tm9uRVRYVlZoU1U0WVFyNjNB/Literature/Gripper-Finger-Design/Grasp_Quality_Evaluation_Network_for_Surface-to-Surface_Contacts_in_Point_Clouds.pdf?inline.

[12] Herzog, A., Pastor, P., Kalakrishnan, M., Righetti, L., Bohg, J., Asfour, T., and Schaal, S. “Learning of grasp selection based on shape-templates”. In: *Autonomous Robots* 36.1-2 (2014), pp. 51–65. ISSN: 1573-7527. DOI: 10.1007/s10514-013-9366-8. URL: https://link.springer.com/article/10.1007/s10514-013-9366-8.

[13] Honarpardaz, M., Tarkian, M., Ölvander, J., and Feng, X. “Finger design automation for industrial robot grippers: A review”. In: *Robotics and Autonomous Systems* 87 (2017), pp. 104–119. ISSN: 0921-8890. DOI: 10.1016/j.robot.2016.10.003. URL: https://www.sciencedirect.com/science/article/pii/S0921889015303171.

[14] Honarpardaz, M., Tarkian, M., Sirkett, D., Ölvander, J., Feng, X., Elf, J., and Sjögren, R. “Generic Automated Multi-function Finger Design”. In: *IOP Conference Series: Materials Science and Engineering* 157.1 (2016), p. 012015. ISSN: 1757-899X. DOI: 10.1088/1757-899X/157/1/012015. URL: https://iopscience.iop.org/article/10.1088/1757-899X/157/1/012015.

[15] Honarpardaz, M. “A methodology for design and simulation of soft grippers”. In: *SIMULATION* 97.11 (2021), pp. 779–791. ISSN: 0037-5497. DOI: 10.1177/00375497211018743.

[16] Honarpardaz, M., Ölvander, J., and Tarkian, M. “Fast finger design automation for industrial robots”. In: *Robotics and Autonomous Systems* 113 (2019), pp. 120–131. ISSN: 0921-8890. DOI: 10.1016/j.robot.2018.12.011. URL: https://www.sciencedirect.com/science/article/pii/S0921889018304123.

[17] Huy Ha, Shubham Agrawal, and Shuran Song. “Fit2Form: 3D Generative Model for Robot Gripper Form Design”. In: *Conference on Robot Learning* (2021), pp. 176–187. ISSN: 2640-3498. URL: https://proceedings.mlr.press/v155/ha21b.html.

[18] J. Hu, J. Whitman, M. Travers, and H. Choset. “Modular Robot Design Optimization with Generative Adversarial Networks”. In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 4282–4288. DOI: 10.1109/ICRA46639.2022.9812091.

[19] J. Varley, J. Weisz, J. Weiss, and P. Allen. “Generating multi-fingered robotic grasps via deep learning”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 4415–4420. DOI: 10.1109/IROS.2015.7354004.

[20] Jaycon. *Exporting 3D Files: STL vs. OBJ vs. IGES vs. STEP*. 2023. URL: https://www.jaycon.com/exporting-3d-files-stl-vs-obj-vs-iges-vs-step/.

[21] Kallioras, N. A. and Lagaros, N. D. “MLGen: Generative Design Framework Based on Machine Learning and Topology Optimization”. In: *Applied Sciences* 11.24 (2021), p. 12044. DOI: 10.3390/app112412044.

[22] Koch, S., Matveev, A., Jiang, Z., Williams, F., Artemov, A., Burnaev, E., Alexa, M., Zorin, D., and Panozzo, D. “ABC: A Big CAD Model Dataset for Geometric Deep Learning”. In: *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, uuuu-uuuu, pp. 9593–9603. ISBN: 978-1-7281-3293-8. DOI: 10.1109/CVPR.2019.00983.

[23] Li, Z. and Sastry, S. S. “Task-oriented optimal grasping by multifingered robot hands”. In: *IEEE Journal on Robotics and Automation* 4.1 (1988), pp. 32–44. ISSN: 08824967. DOI: 10.1109/56.769.

[24] Liu, C.-H., Chiu, C.-H., Chen, T.-L., Pai, T.-Y., Hsu, M.-C., and Chen, Y. "Topology Optimization and Prototype of a Three-Dimensional Printed Compliant Finger for Grasping Vulnerable Objects With Size and Shape Variations". In: *Journal of Mechanisms and Robotics* 10.4 (2018). ISSN: 1942-4302. DOI: 10.1115/1.4039972.

[25] Liu, Y.-H. "Computing n-Finger Form-Closure Grasps on Polygonal Objects". In: *The International Journal of Robotics Research* 19.2 (2000), pp. 149–158. ISSN: 0278-3649. DOI: 10.1177/02783640022066798.

[26] M. Honarpardaz, M. Meier, and R. Haschke. "Fast grasp tool design: From force to form closure". In: *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*. 2017, pp. 782–788. ISBN: 2161-8089. DOI: 10.1109/COASE.2017.8256199.

[27] Miller, A. T. and Allen, P. K. "GraspIt!" In: *IEEE Robotics & Automation Magazine* 11.4 (2004), pp. 110–122. ISSN: 1558-223X. DOI: 10.1109/MRA.2004.1371616.

[28] Mongo DB. *MongoDB Community Server Download*. URL: https://www.mongodb.com/try/download/community.

[29] Nguyen, V.-D. "Constructing force-closure grasps in 3D". In: *Proceedings. 1987 IEEE International Conference on Robotics and Automation*. Institute of Electrical and Electronics Engineers, March 1987, pp. 240–245. DOI: 10.1109/ROBOT.1987.1088014.

[30] Peri, D. *Machine Learning Algorithms in Design Optimization*. URL: https://arxiv.org/pdf/2203.11005.

[31] R. Detry, C. H. Ek, M. Madry, and D. Kragic. "Learning a dictionary of prototypical grasp-predicting parts from grasping experience". In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 601–608. ISBN: 1050-4729. DOI: 10.1109/ICRA.2013.6630635.

[32] Ramasubramanian, A. K., Connolly, M., Mathew, R., and Papakostas, N. "Automatic simulation-based design and validation of robotic gripper fingers". In: *CIRP Annals* 71.1 (2022), pp. 137–140. ISSN: 00078506. DOI: 10.1016/j.cirp.2022.04.054.

[33] Ringwald, J., Schneider, S., Chen, L., Knobbe, D., Johannsmeier, L., Swikir, A., and Haddadin, S. "Towards Task-Specific Modular Gripper Fingers: Automatic Production of Fingertip Mechanics". In: *IEEE Robotics and Automation Letters* 8.3 (2023), pp. 1866–1873. ISSN: 2377-3766. DOI: 10.1109/LRA.2023.3241757.

[34] Sanfilippo, F., Salvietti, G., Zhang, H. X., Hildre, H. P., and Prattichizzo, D. "Efficient modular grasping: An iterative approach". In: *2012 4th IEEE RAS and EMBS International Conference on Biomedical Robotics and Biomechatronics*. Ed. by Author, I. C. IEEE, 6/24/2012 - 6/27/2012, pp. 1281–1286. ISBN: 978-1-4577-1200-5. DOI: 10.1109/BioRob.2012.6290693.

[35] Scala, R. M., Cabanellas, J. M., and Baltasar, L. "Part 3D Design and Draft According to CSG (Constructive Solid Geometry) and Its Application to Three CAD Software: SolidEdge ST 2021, Onshape, AutoCad". In: *ADVANCES IN DESIGN ENGINEERING III*. Ed. by Cavas-Martínez, F., Marín Granados, M. D., Mirálbes Buil, R., and Cózar-Macías, O. D. de. Lecture Notes in Mechanical Engineering. [S.l.]: SPRINGER INTERNATIONAL PU, 2023, pp. 1023–1063. ISBN: 978-3-031-20324-4. DOI: 10.1007/978-3-031-20325-1{\textunderscore}80.

[36] Schwartz, Lukas Christoffer Malte Wiuf, Wolniakowski, A., Werner, A., Ellekilde, L.-P., and Krüger, N. "Designing Fingers in Simulation based on Imprints". In: *Proceedings of the 7th International Conference on Simulation and Modeling Methodologies, Technologies and Applications*. SCITEPRESS - Science and Technology Publications, 2017. DOI: 10.5220/0006441003040313.

[37] Stratasys. *Grabcad*. 2015. URL: https://grabcad.com.

[38] V. B. Velasco and W. S. Newman. "Computer-assisted gripper and fixture customization using rapid-prototyping technology". In: *2013 IEEE International Conference on Robotics and Automation*. 2013, 3658–3664 vol.4. ISBN: 1050-4729. DOI: 10.1109/ROBOT.1998.681393.

[39] Wiling, B. "Scientific Study of CAP Theorem and Understanding its Different Implementation Methods". In: *Mathematical Statistician and Engineering Applications* 71.1 (2022). DOI: 10.17762/msea.v71i1.55.

[40] Wolniakowski, A., Miatliuk, K., Gosiewski, Z., Bodenhagen, L., Petersen, H. G., Schwartz, L. C. M. W., Jørgensen, J. A., Ellekilde, L.-P., and Krüger, N. "Task and Context Sensitive Gripper Design Learning Using Dynamic Grasp Simulation". In: *Journal of Intelligent & Robotic Systems* 87.1 (2017), pp. 15–42. ISSN: 1573-0409. DOI: 10.1007/s10846-017-0492-y. URL: https://link.springer.com/article/10.1007/s10846-017-0492-y.

[41] Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. "3D ShapeNets: A Deep Representation for Volumetric Shapes". In: 2015, pp. 1912–1920. URL: https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Wu_3D_ShapeNets_A_2015_CVPR_paper.html.