# PARAMETRIZATION AND AUTOMATION OF THE CREATION OF A 3D MODEL

Master's thesis

Alexandre PERIE

Industrial supervisors: Benoit GELY & Prasad MAHAJAN
University supervisor: Lukas KAZDA

Study program: Automotive Engineering
Department: Department of Technical Engineering

03/07/2023 – 22/12/2023

# Table of Contents

**Acknowledgements:**

For the great experience during this internship, concluding long studies in engineering, I would like to thank:

## My parents

All along my studies, they have been by my side supporting each of my decisions but also helping me when I needed guidance. They knew how to keep me motivated and they have been a real source of inspiration.

## My friends

During these studies and even during this internship, I discovered wonderful people, making my life even better each day. A special thanks to Prasad, and I guess “Congratulations” are even in order since he invited me to his wedding in India, showing how friendly and kind the people at Capgemini were.

## My internship supervisors

This internship has been full of challenges, but Benoit, Prasad, and Lukas knew how to support me and provide me with constructive feedback. I always tried to make the most of their remarks, and thanks to them I have been able to achieve my goals.

# Abstract

The goals of my internship were divided into two main tasks, parametrizing an entire bike frame model, and automating the creation of a CATIA model from the data of its technical drawing. Since literature is short on information about the automation process, one major obstacle would be to define this process for our case. Make it as clear and structured as possible would allow us to open it up to a wide field of applications. Because of the client constraints and the company preferences, I would have to work on software like CATIA and Python.

By studying CATIA features, I have been able to complete the parametrization of the bike frame. I also had to work on CATIA scripts so I could begin to develop automation tools and create a quick animation of the possibilities of the parametrization.

To automate the process of creation of a 3D model, I had to code on Python using software like Visual Studio Code and GitLab. Additionally, I ventured into AI, employed for contour recognition on technical drawings. Thus, I delivered a functional tool using data from the technical drawing as input and returning a sketch of the views on a CATIA format file. My work would allow a precious time reduction in design operations, and a better result in CAD models.

# List of figures

# 1 Presentation of the company

In 1967, Serge Kampf founded his enterprise management and data processing company called Sogeti. This company will keep expanding and merging with other Information Technology (IT) companies to become the one we know today: Capgemini. In 1974, Sogeti bought the American company Gemini Computer Systems and merged with CAP (Centre d'Analyse et de Programmation) which will give the name of CAP Gemini Sogeti. Today, the company is known as Capgemini and Sogeti is still one of its subsidiaries [1].

Capgemini Engineering is a subsidiary of Capgemini created from the acquisition of Altran Technologies in 2019. It is now a global leader in engineering and R&D services with more than 65.000 engineers and scientists across the globe. The company helps clients across a wide range of industries, including automotive, aerospace, defense, energy, healthcare, life sciences, manufacturing, and telecom, to innovate and develop the products and services of the future.

With 55 years of experience, Capgemini Engineering offers a comprehensive portfolio of services, including digital, software and mechanical engineering and has a dedicated R&D team that is constantly developing innovative technologies and solutions to help clients stay ahead of the curve. In 2022, the Group reached the whooping figure of €22 billion sales [2].

# 2 Problem statement

If we want to realize a 3D model of an existing product, the first step would be to get a representation of this object, whether an image or a drawing of it. Even if most of the creation is done today by Computer Aided Design (CAD), creating the 3D model requires a non-negligible amount of time since it must be step by step. Automating this operation could save a lot of time for the employee as well as money for the client. If we want to create a new product, we don't have the existing drawing of the part, so our process would be applied to retro-engineering. In our case, our motivation started from the fact that constructors often deliver drawings of old parts, and not 3D models, created when CAD didn't exist. We are nowadays living in a digital era, where we want to have all files and parts on a digital support. Recreating the 3D model from old drawings, which rarely include dimensions, has naturally become an essential step in the automation process.

It has been the driver of multiple studies, trying to find a way to convert a 2D drawing into a 3D model. Today, we can find a lot of articles describing the process of the analysis of a technical drawing. For example, Prasad A. and Harish A. defined an algorithm generating a 3D solid from different views of an object [3]. However, their algorithm does not describe how they store and represent their data used to create the 3D model. Also, Lapo G., Rocco F., Matteo P., and Yary V. found a means to transform 2D orthographic drawings into 3D models. They specify how their image treatment works, how they retrieve clean curves from the drawing [4]. But they don't specify their type of data, how they reconstruct the 3D model from it, and it is the heart of the subject. Moreover, none of the work done today is applied on CATIA, that is why my objective during this internship would be to give a clear process from the data structure to the creation of the CATIA file. To get to this, three notions need to be understood:

- How is it possible to automate CATIA operations?
- How can we represent the data structure through a precise ontology?
- How can we extract contours from our drawing with a clear image process?

We proceeded step by step so we would not rush into anything, and we wanted to make sure we mastered each stage before moving on to the next. To achieve every goal, the start involved getting familiar with CATIA and especially with its automation tools. This would be done by creating the parametrization of the bike frame and realizing an animation covering its possibilities (not detailed in the report because of the format of the animation).

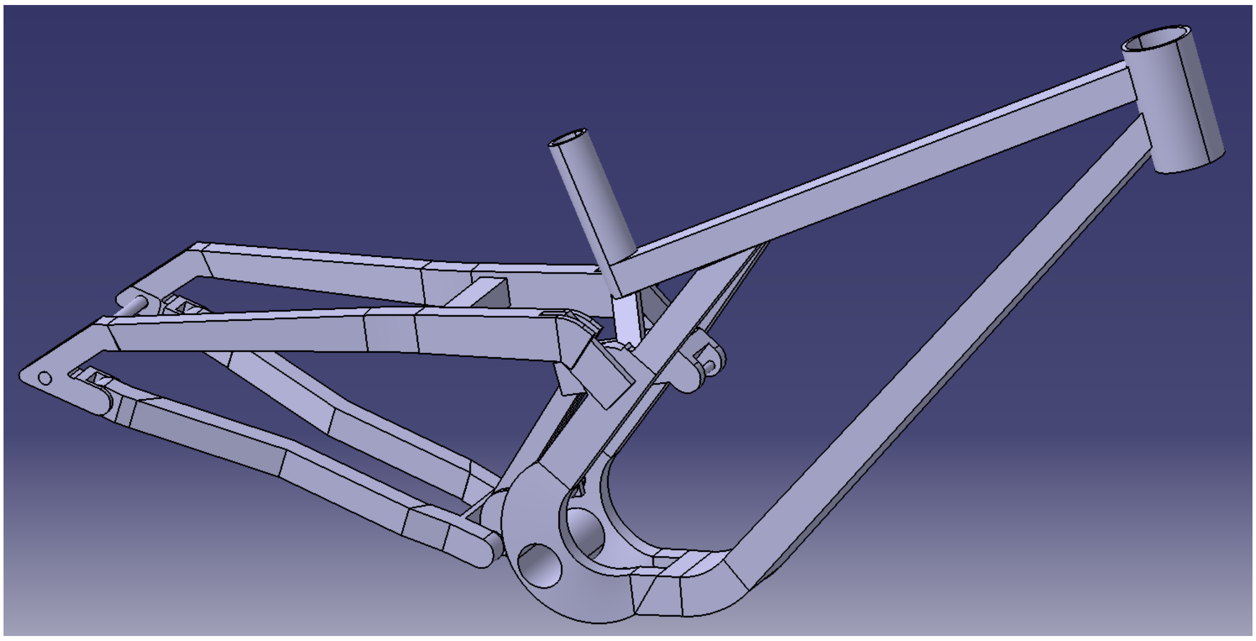


Figure 1: Assembly of the bike frame

Then, writing a script automating the creation of a simple part seemed the logical way to approach. Defining a structured format for data as our input appeared as the natural step to lead to the creation of more complex sketches.

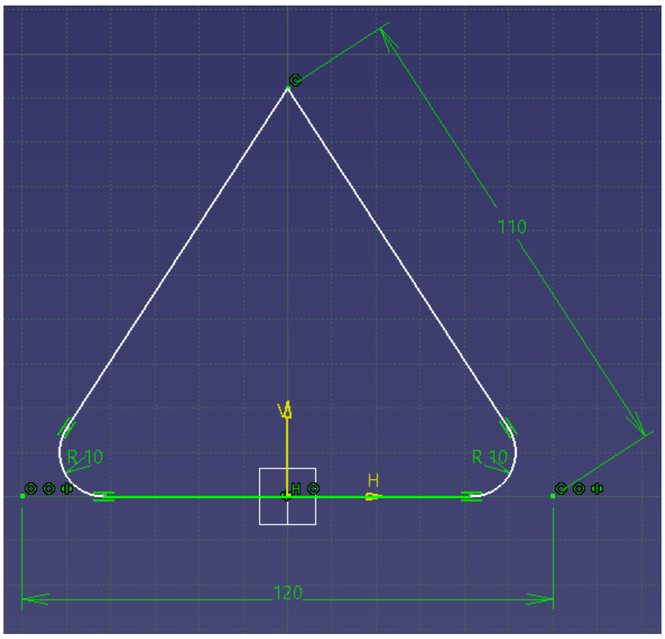


Figure 2: Example of a sketch for a simple shape

After getting this process cleared, we realized that the contour recognition algorithm was flawed and needed some adjustments. By correcting and adding new functions, we have finally been able to get close to the result desired.

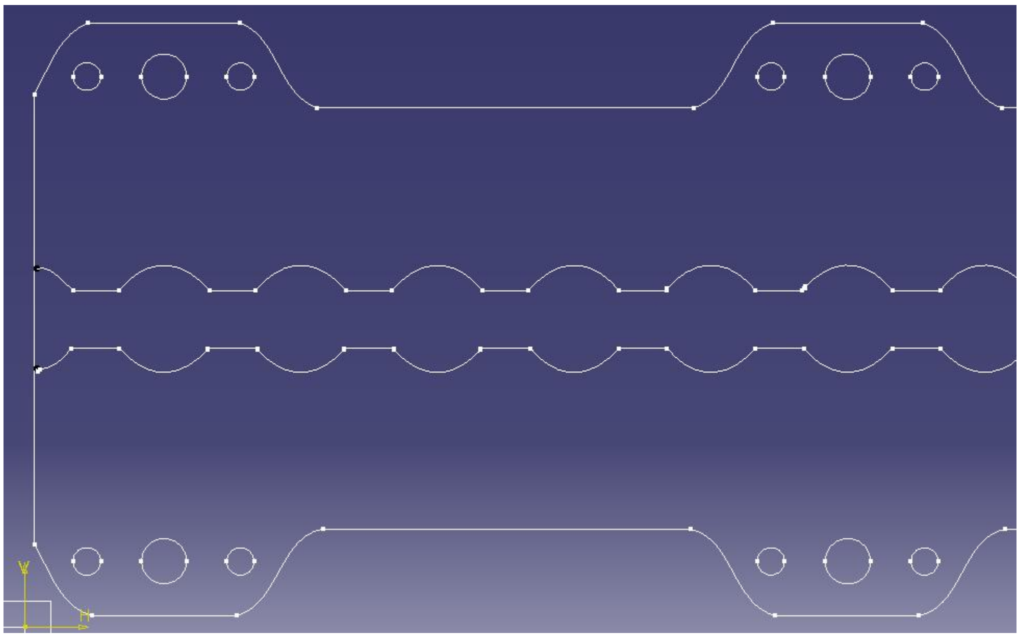


**Figure 3: Resulting sketch after improvements of the algorithm**

# 3 State of the art

As explained in problem statement (2), work is related to the CATIA automation and 3D model creation from the industrial drawing. Bibliography has been done based on the CATIA scripting in Python and image processing.

## 3.1 CATIA Automation

Given industrial constraints about the subject, CATIA seemed to be the only conceivable tool for the 3D modelling part of the problem. CATIA is a software used in CAD, Computer Aided Manufacturing (CAM) or Computer Aided Engineering (CAE). It offers a wide range of features such as 3D modelling and design, and product analysis [5]. Developed by Dassault Systèmes, CATIA stands out as a leading non-open-source software package widely used in the engineering world. Its extensive user base and popularity within diverse industries provide a strong foundation for applications and innovations, and building on this support would open up a wide field of applications [5]. Many operations can be automated by using macros, scripts that the user can define and write in the software. It is also possible to realize this automation through a module implemented in Python. The preference for this final possibility is due to its simplicity and ease of understanding, attributed to the clear and straightforward syntax of Python.

### 3.1.1 CATIA script

The tool present in CATIA is CATScript, its code language is called Vbscript, and it is the same as Visual Basic (language used in Microsoft Excel macros, and one of the most popular in the world). The way CATIA Automation works is object-oriented. It means that our variables are represented as objects that can be included in classes or to which we can apply attributes and methods [6].

For example, to create a simple sketch we would have to create a part beforehand. These two objects, the sketch, and the part, would belong to two classes. Like their names, they would be given attributes, and we can apply methods to them. We can call a function in the sketch class like one that would create 2D elements or one that would create a constraint between two elements of the sketch.

### 3.1.2 Pycatia

As mentioned before, Python is a better option for the automation. For this, we would need a module that could call and use CATIA objects as needed. This module is called “Pycatia”. It offers a direct approach to accessing and manipulating CATIA objects such as parts, assemblies, and geometries, enabling precise parametrization of designs. Its module structure is like the one of CATIA Automation, with the primary distinction lying in a slightly different syntax [7]. We would see later that we would separate the CATIA operations from the main code to help for a better understanding of the algorithm. This approach, calling a specialized script from a main one, is indeed simpler on Python than on CATScript.

Pycatia offers on Python almost all the features available on CATIA. We can create assemblies, parts, sketches, geometries, and constraints. Even if it will be mentioned again later in this report, it is necessary to give examples of the possibilities of this module in this section. So, a script realizing a simple extrusion must follow the same logic we would do on the software. To create the pad, we need to draw the sketch beforehand. So, the first step is to create the part, then the sketch, then the constrained geometry by creating each element individually, and then we can do the extrusion. All the functions behind take the same inputs as the ones we would enter in the software. For example, to create a closed circle we would have to enter the coordinates of the centre and the radius. A sketch must be created from a reference plane and inside a part.

We can see the logic for the creation of a simple cube in the following process:

- Import Pycatia (as "**caa**")
- Create a new document with "**documents = caa.documents**"
- Create a new part with "**part1 = documents.add("Part").part**"
- Create a new body from that part and a geometrical set: **body1 = part1.bodies.item("Geometrical set.1")**
- Create a new sketch inside that body from a reference plane with "**sketch1 = body1.sketches.add(reference plane)**"
    - Create the 2D elements library by opening the sketch edition mode: **factory2D1 = sketch1.open_edition()**
    - Create the lines for the square base of our cube: **factory2D1.create_line(x_start, y_start, x_end, y_end)**
- Close the sketch edition mode: **sketch1.close_edition**
- Update the part: **part1.update**
- Create the extrusion for the cube: **part1.shape_factory.add_new_pad(sketch1, length)**
- Update the part: **part1.update**

To use Pycatia, CATIA needs to be installed on the computer, however it doesn't need it to be opened. This "bash mode" execution allows us to save time in the process. We can then save the file created in a repertory on the computer. Each element created with Pycatia is an object. It means that we can access the document as well as the different elements of the geometry of a sketch as objects and get their attributes. We can also apply functions to them. For example, we can update or save the document.

## 3.2 Ontology

In the IT world, ontology is a discipline that aims to formally represent the knowledge of a specific domain. It relies on a taxonomy to organize entities into classes and subclasses, and semantic tables to describe the properties and relationships between these entities. It supplies a semantic structure and a common language for describing entities and their interactions within the domain. Such an approach makes it possible to capture and model domain knowledge in such a way as to make it explicit and comprehensible to both machines and humans.

In practice, an ontology is represented in the form of graphs, such as the one shown in Figure 4, where a concept can be described by several attributes and may have one or more relationships (hierarchical, partial, association) with other concepts or systems [8].

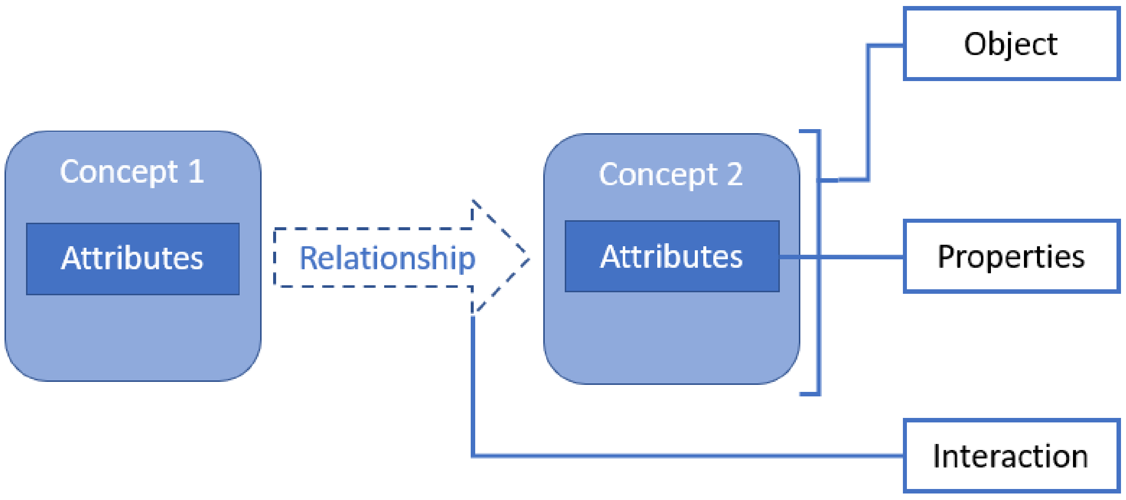


**Figure 4: Explanation of the ontology [8]**

The use of ontologies in IT offers several advantages. Firstly, it eases the understanding and exchange of knowledge between different applications and systems. By using a common ontology, developers can ensure that the various parts of the system in question interact coherently. Ontologies also help information retrieval and data access. By describing concepts and relationships in a structured way, it becomes possible to perform precise queries and extract relevant information from heterogeneous data sources [8].

## 3.3 Image processing

In generating a CATIA model from the technical drawing data, we depend on the functionality of the contour recognition algorithm. It is imperative to understand the workings of the image processing involved. In our approach, the Python module named cv2 is employed to execute operations for extracting contour data and manipulating the input image [9]. For example, this library allows us to convert the input image into a black and white picture using a threshold for the pixel's intensity. The threshold value is an integer between 0 and 255, 0 being a white pixel and 255 a black pixel. This functionality will be very useful when we will process drawings which are not clear enough or maybe even blur. If the threshold is too low, we might extract too many contours and if it is too high, we might lose some.

After transforming the colours and getting a clearer image, cv2 can extract the contours of the shapes on the image by using the function "findContours". The retrieval method for the contours depends on the attributes entered in the function. It also returns us the hierarchy matrix of the contours, we will explain this notion in detail later in this report, but it essentially represents the relationship between the contours. The hierarchy will not be the same depending on the retrieval method.

Cv2 is even able to extract the dominant points of the contours with "approxPolyDP". Thanks to this function, we can approximate the shape of a contour depending on the number of points composing it. However, we lose data if a shape is rounded for example. This method would only extract the starting and ending points, and we will not recognize an ark but a segment.

Also, cv2 enables us to get other attributes of the contours such as their areas or their perimeters, but we will not use it. We studied some ways to use them to delete the problematic contours, but it led us on wrong tracks.

The contour recognition method using the approximation of the number of points composing the contour is called a "morphological method". Another one is the "predictive method", using an Artificial Intelligence which simply recognizes the shape and categorizes it into a class. This last method needs the AI to be well trained, which is not really the case of the one we use for our algorithm. This caused us some problems regarding the nature of the contours we studied and their representations.

We need now to mention that the input for our work will be a clear image of the technical drawing. It means that the technical drawing has been cleared from all the dimensions and notations on it, we simply have the shape of the part. The algorithm which cleans the drawing will not be studied in this report since it concerns a different approach of the subject. In the future, it may become necessary to consider it to perfect the way our algorithm could work or how the technical drawing data could be more accurate.

# 4 Parametrization of a CATIA model

The first goal of the internship was the parametrization of an entire CATIA model so it would help for the optimization of the design. The bike frame is already on the market, but the study would help for a future product. To get started, we could rely on the model of the existent frame. From it, it is possible to retrieve the dimensions and the geometry for a simplified model. To achieve this objective, we would proceed in four steps:

- Create the model from assumptions we would make about the geometry
- Define the parameters
- Get as much relations as possible between them to decrease the number of inputs
- Control the parameters from an Excel sheet and a macro.

## 4.1 Creation of all the parts of the assembly and definition of the parameters

The first step to parametrize the bike frame assembly is to realize the whole assembly. To do that, we need to make assumptions about the geometry of the frame, decompose it into segments. Each segment corresponds to a part of the frame that has no radical changes in geometry, volume or simply corresponds to a piece of the frame. That way, we can isolate each interesting part and simplify its geometry.

We try to stay true to the original geometry as best we can. We can approach the section of the segments to rectangular or circular ones. When there is a noticeable change in dimensions of the section along the segment, we realize an extrusion with a variable section. For each part, we try to name the parameters of the geometry, so it will be easier to recognize them during the parametrization step. Since the parts are created as volumes, we then need to extract the surfaces for the realization of the final part as 2D element. What we mean by a 2D element is like a beam with a defined section and a surface thickness. Then, we deactivate the volume and keep the resulting part. We can see the process on Figure 5:

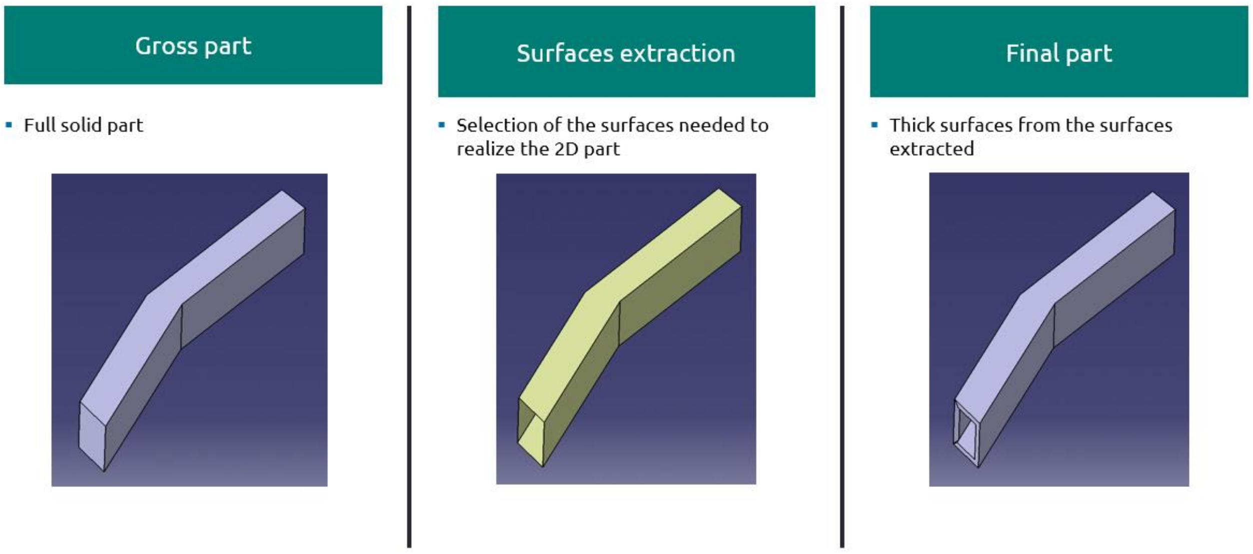

Gross part
Full solid part
Surfaces extraction
Selection of the surfaces needed to realize the 2D part
Final part
Thick surfaces from the surfaces extracted

**Figure 5: Process of the creation of a part**

We need to make sure the joints between the parts fit correctly, we can also adapt these joints with our own values to be sure the assembly will fit. For example, some parts of the frame are just beams linked to other parts with a special function, like receiving a pivot. This link is a flush-mounted liaison as the two parts are melded together. This kind of connection needs accurate measurements since the symmetry of the whole geometry could depend on it.

After completing the production of all components, the next phase involves assembly. If the measurements are correct, the parts should seamlessly come together. Otherwise, the small gaps or errors are readjusted. It is crucial to precisely set up the assembly constraints, ensuring that when the frame is in motion, it adheres to the bike's actual dynamics. For this kind of assembly, only pivot and flush-mounted connections are employed.

The bike frame can be divided into two main parts: the front and the rear parts. The front can represent the frame itself, the elements not moving and all mounted together. The rear part is the most interesting and will be the focus of our research for parameters. The rear part will move with the suspension of the bike. A shock absorber connects the rear and the front parts (not represented on Figure 6). It is coupled to a connecting rod system so the "rear triangle" keeps a circular translation movement (parallelogram JGOI on Figure 7). The main characteristics that we would like to act on are the position of the top pivot liaison (point G), the angle at the rear triangle ($\alpha_1$), and the space between the two rear triangles where the rear wheel is. And we will change the angle at the front of the frame on the bar connecting the pedal masterpiece and the cylinder where the bicycle fork comes.

Also, all the sections parameters like the widths, the heights, and the thicknesses will be monitored. We will not change them, but we include this possibility so that we can present more options when the optimization of the frame needs to be done.

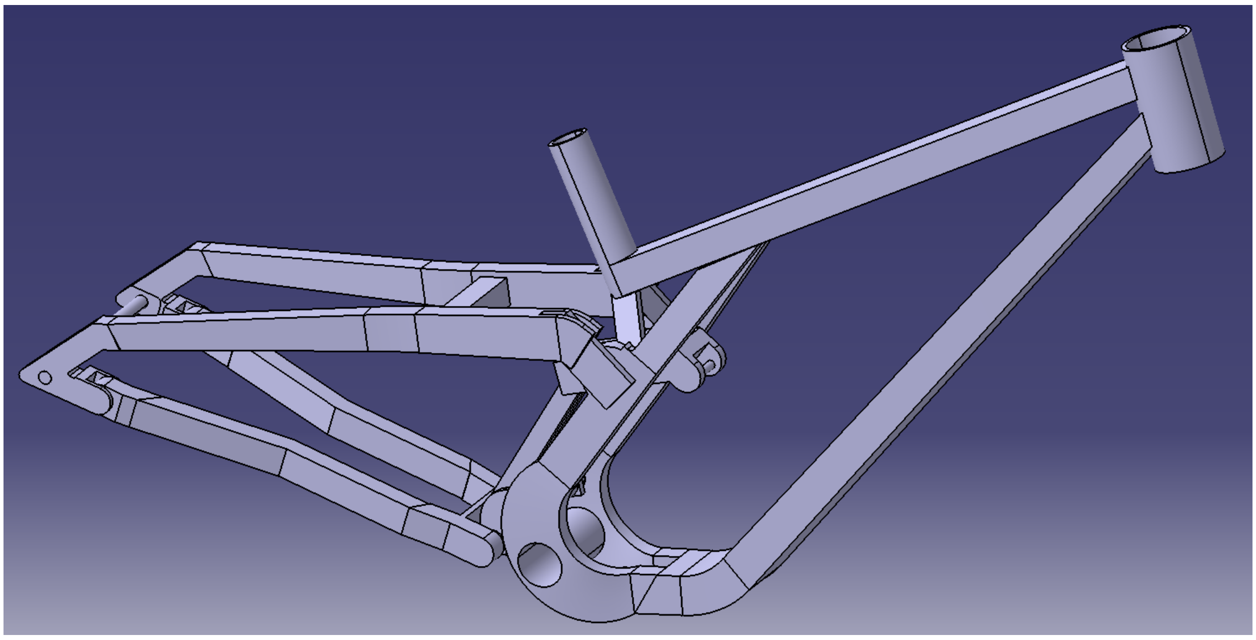


**Figure 6: Complete bike frame model**

## 4.2 Construction of the equations of the problem

Between the assembly phase and the creation of the parts, it became necessary to find relationships between the measurements, so it fits correctly. This implies mathematical and geometrical relations between the parameters defined previously during the creation of the parts. All these relationships result in a problem with multiple variables, some unknowns, and others we need to impose. For that, we define which parameters are our inputs, what geometry we want to change, and which parameters are affected. For example, if we want to change the angle at the rear of the frame, it will imply that the lengths of the parts at the rear would change as well as the position of pivot liaisons.

The first assumption is that all the parameters concerning the sections would be entered manually. Then, we decide to impose the angle alpha1 (Figure 7), and the distances between the symmetrical parts (Figure 8) at the rear as our inputs. All the other parameters affected will have to be calculated and actualized.

In identifying the relationships, we initially mapped out the problem on paper and deduced the necessary equations. To simplify the complexity, we broke down the problems into smaller components, isolating the unknowns. Recognizing the persistent difficulty, we opted to stabilize the geometry by fixing it in a predefined position, termed as the initialization position, thereby reducing the number of unknowns. Utilizing these equations, we obtained parameter values crucial for the main equations of our problem. Employing geometric closures and fundamental relations within triangles, we established sufficient connections to effectively address and solve the problems. After simplifying the equations, we were able to get the following results, the main parameters can be seen in the figures 7 and 8, the other ones are just "transition" parameters:

- Geometric closure inside GOIJ projected on $\vec{y}$:

$$-\left(L_{pivot} + |HO|\right)sin(\alpha_0) - |OI|cos(\varphi_1) + L_{bielette}cos(\alpha_I + \varphi_1) + |JG|cos(\varphi_3) = 0 \tag{1}$$

- Geometric closure inside GOIJ projected on $\vec{z}$:

$$-\left(L_{pivot} + |HO|\right)cos(\alpha_0) - |OI|sin(\varphi_1) + L_{bielette}sin(\alpha_I + \varphi_1) + |JG|sin(\varphi_3) = 0 \tag{2}$$

- Geometric closure inside ABCDEFGOI projected on $\vec{y}$:

$$\begin{gathered} -|AB| - |BC| + |CD|cos(\alpha_1) + |DE|cos(\alpha_4 + \alpha_1 - \pi) + |EF|cos(\alpha_2 + \alpha_4 + \alpha_1) + |FG|cos(\alpha_3 - \varphi_3) \\ - \left(L_{pivot} + |OH|\right)sin(\alpha_0) - |OI|cos(\varphi_1) - |IA|cos(\varphi_1 - \alpha_5 + \pi) = 0 \end{gathered} \tag{3}$$

- Geometric closure inside ABCDEFGOI projected on $\vec{z}$:

$$\begin{gathered} |CD|sin(\alpha_1) + |DE|sin(\alpha_4 + \alpha_1 - \pi) + |EF|sin(\alpha_2 + \alpha_4 + \alpha_1) + |FG|sin(\alpha_3 - \varphi_3) - \left(L_{pivot} + |OH|\right)cos(\alpha_0) \\ - |OI|sin(\varphi_1) - |IA|sin(\varphi_1 - \alpha_5 + \pi) = 0 \end{gathered} \tag{4}$$

- Triangle relation:

$$\alpha_I + \alpha_J + \varphi_1 - \varphi_3 = \pi \tag{5}$$

- Vector relations:

$$|DE_x| = |DE_{x0}| + (Voie_{arrière} - Voie_{arrière0}) - (Ecart_{central} - Ecart_{central0}) \tag{6}$$

$$|AB_x| = |AB_{x0}| + (Voie_{arrière} - Voie_{arrière0}) - (Voie_{centrale} - Voie_{centrale0}) \tag{7}$$

Since Equations 1, 2, 3, and 4 are not linear, we used the solver from Excel. This method relies on least-squares averaging calculations, so it is possible the values are correct but not physically possible. In our case, the results were plausible enough, so we kept this method.

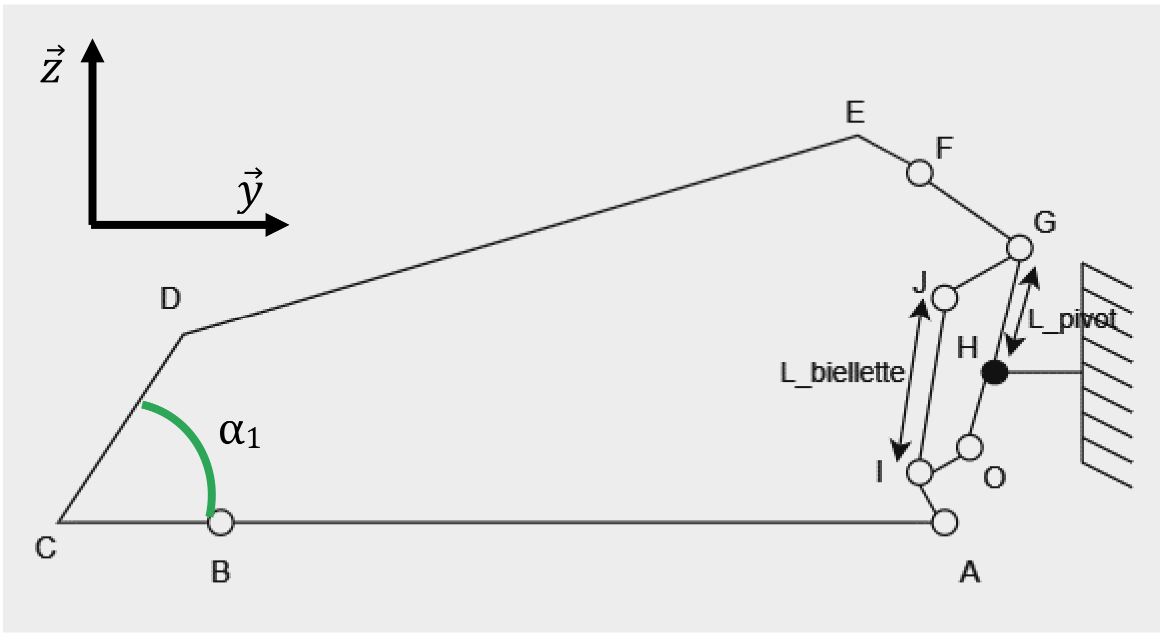


Figure 7: Scheme of the problem at the rear of the frame

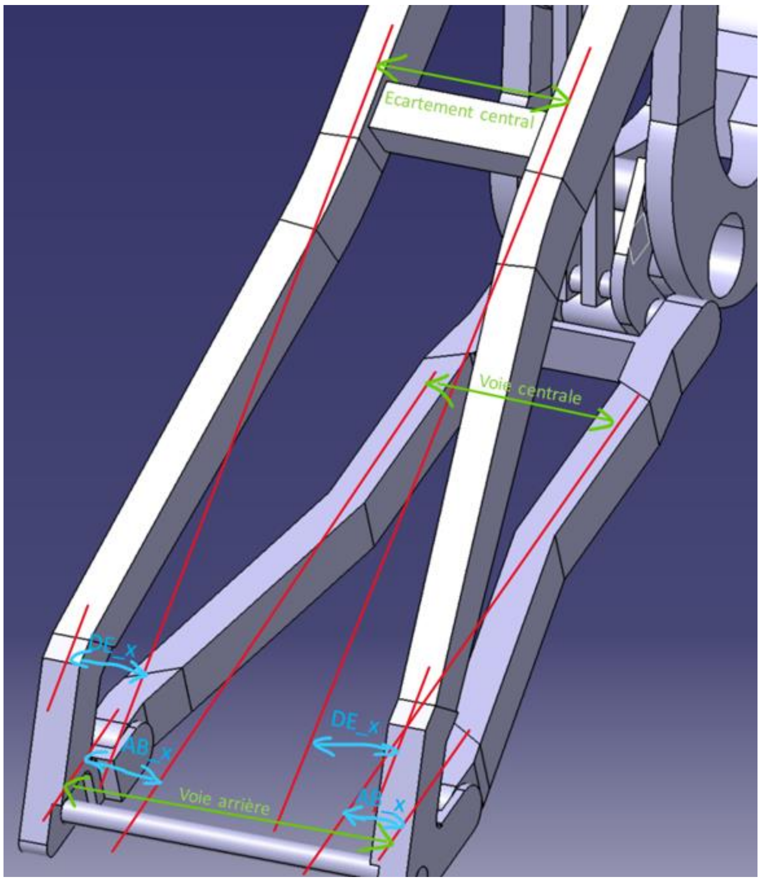


Figure 8: Scheme of the problem of the symmetry of the parts at the rear of the frame

## 4.3 Control of the parametrization

Once the assembly is done and the equations defined, the last step is the parametrization. For this, we will use the built-in tool of CATIA: the design table. The benefit of having already named our parameters as we wanted is that now they are recognizable among all the parameters the design table tool presents us. In parallel to this, we create an Excel file in which each parameter is represented in a column. Then, the design table on CATIA is smart enough to automatically make the connection between the values of the parameters in the Excel files and the parameters we selected before.

To streamline this process across different configurations, a concise algorithm has been developed using CATScript. This not only automates the procedure but also enables us to generate animations based on the various configurations.

The layout of the Excel file needs to be specific because we need to have the parameters' names on the first line, then all the values and nothing else in the table. In our case, we have another sheet where we put the equations of our problems so when we change a value in the first sheet, we launch the algorithm which will update the impacted parameters according to our equations. Also, we must be careful with the units because CATIA will read them in the International System if they are not mentioned in the name's cell. Because of the important number of parameters, almost a hundred, we put our focus on the ones defined previously such as the rear angle $\alpha_1$, the distance of the pivot located in G and the distances between the symmetrical parts around the rear wheel.

# 5 Use of Pycatia to automate the creation of a part

After we understood the basics of the CATIA Automation process during the realization of the script for the parametrization, it became possible to go to the next step: directly automating the creation of the part. We would rely on clean pictures of technical drawings, simple shapes at first, then we would apply our algorithm to more complex shapes. However, as mentioned earlier, we lack documentation about the process from the 2D drawing analysis, the data storage, to the creation of the 3D model. It would be our role to define this road map, and make it as clear and structured as possible. And since much of the work would have to be done on Python, we needed to find the correct module to make the link with CATIA. Pycatia was the solution. We would try to structure the code as logical as it could be, separating the Pycatia part and the main code involving more Python in-built methods.

## 5.1 Reading data from an Excel file

Since during the parametrization we were using an Excel sheet to define a configuration of our model, we decided to keep this method to define a configuration for our part. In our case, the configuration would concern the geometry of the part like the fillet radius or distances for example.

Our model is a lug, so we know its geometry and it is a good start to develop a method that will be applied to any part. We create a sketch on which we know all the constraints and the names of our geometric elements (Figure 9). In the Excel file we would create three columns with the names of our parameters: the side length, the base length, and the fillet radius.

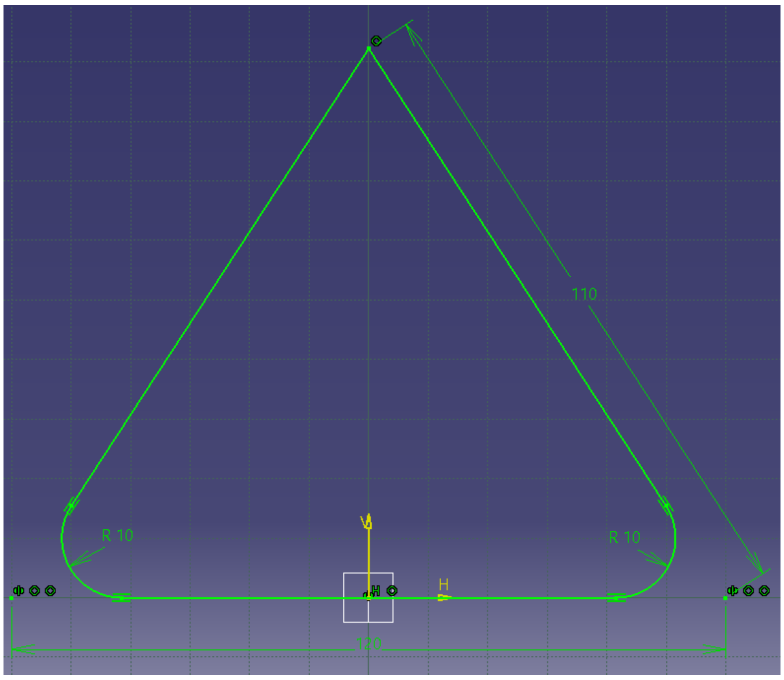


**Figure 9: Sketch of the lug model**

To read the Excel file, we use the module in Python called "pandas"; it extracts the Excel values as a table and automatically recognizes the names of our columns. Then, we catch the correct value corresponding to the configuration we want and apply it to the model. We update it and since the geometry is correctly constrained, when we change the distances there is no problem in the sketch.

## 5.2 Library of functions

With the example of our lug, since we know the specifics of the part we are creating, we know which operations to make and in which order. So, the first script we realized was specific to that case. To aim for a broader field of application, we decided to divide our operations. It means that we would have our main code realizing the logical operations and another code in which we would define functions that would be focused on CATIA operations like sketch creation, geometry's elements, constraints creation… To realize this script, which could be like a library of functions, we decided to match its ontology to the one of Pycatia. We have our part and our sketches represented as classes so we can get their attributes and call their own methods.

For example, we create a part, then a sketch, then we create a Factory2D element associated to this sketch. This Factory2D element allows us to create points, lines, circles and other 2D geometric elements in our sketch. Then, we create a Constraints element associated to our sketch. This Constraints element will allow us to create the horizontal, perpendicular, parallel constraints between the 2D elements we created before, and that we can find in our sketch. Finally, we can update or save the part thanks to methods in our Part class.

Incorporating 3D operations, such as pads, holes, or pockets, into our script represents a final crucial step. Prior to implementing these operations, it is imperative to ensure the integrity of the sketch. The sketch must be closed and without any self-intersections. To address potential issues, we created a function capable of detecting gaps within a geometry. It is designed to work seamlessly on geometries composed exclusively of segments. This function finds gaps between the endpoint of one segment and the starting point of the next, calculates the intersection point, and generates two more segments to effectively fill these gaps.

For now, the main code is structured as we know how to create the part. We know which operation to make, we simply separated the CATIA functions from the logic of conception. Our objective is now to adapt our main script to any sketch it could encounter. So, it means that we need to define a precise type of input to represent the data of our sketch. We could keep the Excel format, but we decided to change to a JSON file because the data representation was simpler and easier to construct.

# 6 Creation of a sketch from JSON file

As mentioned before, since we knew the geometry and the constraints of the part we were studying, the input was known. But, to realize a script that could be applied to any sketch, we needed to define an input easily understandable and that could also be done easily by the contour recognition code. The JSON format seemed appropriate since we needed to store attributes, values, or names into our file.

## 6.1 Proposition of a structure for JSON file

To make the input adaptable to any sketch, it was imperative to identify and specify the necessary parameters, universal data applicable to any model. Once these parameters were defined, we organized them into a format that mirrored the structure of our existing library of CATIA functions. This structured approach ensures consistency and compatibility throughout the implementation.

The skeleton of the JSON file would be as followed:

- Sketches/Sketch1
  - Geometry
    - Points
      - Point1: {x, y}
    - Lines
      - Line1: $\{x_{start}, y_{end}, x_{end}, y_{end}\}$
    - Arcs
      - Fillet1: $\{x_{centre}, y_{centre}, radius,\ start\ point\ name, end\ point\ name\}$
    - Circles
      - Cirlce1: $\{x_{centre}, y_{centre}, radius\}$
    - Splines
      - Spline1: $[start\ point\ name, control\ point\ 1, control\ point\ 2, end\ point\ name]$
  - Constraints
    - Horizontality
      - $[line1\ name, line2\ name, \dots]$
    - Verticality
      - $[line1\ name, line2\ name, \dots]$
    - Parallelism
      - $\{First\ element\ name, second\ element\ name\}$
    - Coincidence
      - $\{First\ element\ name, second\ element\ name\}$
    - Symmetry
      - $\{First\ element\ name, second\ element\ name, symmetry\ element\ name\}$
    - Distance between points
      - $\{First\ element\ name, second\ element\ name, value\}$
    - Tangency
      - $\{First\ element\ name, second\ element\ name\}$
    - Radius
      - $\{Circle\ or\ ark\ name, value\}$
    - Fixed
      - $[element1\ name, element2\ name, \dots]$

## 6.2 Adaptation to structure of given JSON file

The contour recognition code does not yield the precise data needed for our CATIA creation script. Consequently, we decided to store an extensive amount of data, aiming to align with the predetermined frame example as closely as possible. Notably, the output from the contour recognition code diverges from our first concept, as it captures not only the coordinates of all points but also keeps information on all contours and constraints, all at the same hierarchical level.

### 6.2.1 Data from contour recognition code

The contour recognition code is separated in various parts. There is a first part in which the technical drawing is cleaned of all the dimensions or notations and gives us a clean image of the technical drawing. But we will not study this part since it does not involve any matter that could affect our results.

Then, there is the extraction of the contours and the categorization of the nature of these contours. These two steps are especially important because they both relies on sensitive methods of detection end classification. The contours extraction is done using the method findContours from cv2 module. And the determination of the nature of the contours is done using an Artificial Intelligence (AI).

For each contour, the findContours method will give us a list of all the points ordered as we follow the contour in a clockwise direction. There exist different methods of contour recognition inside this method, and each of them would give us a different result. Also, findContours gives us the hierarchy of all the contours. The hierarchy will be explained later in this report, but it is important to know that it is a criterion used to sort the useful contours of the shape.

After having detected and deleted the useless contours, the AI processes the remaining contours to figure out their nature. It can be “Triangle”, “Rectangle”, “Circle” or “Complex”. According to their nature, the algorithm processes the contour points and gives us all the components of the contours sorted in different lists:

- Points (or nodes)
- Segments
- Curves
- Circle parameters (centre point coordinates and radius)

Of course, these lists can be empty depending on the nature of the contour. For example, if it is a rectangle, we would only have data in the nodes and segments lists.

From these data, we build a JSON file with a structure as close as the one defined before. It would store the hierarchy, all the contours and their points, the constraints between all the elements and a more structured list of the contours, all this at the same level in the dictionary represented by the JSON format.

The more structured list of the contours would include data such as:

- Identification number
- Points coordinates
- Nature of the contour
- Lists of the segments, curves, or circle parameters

## 6.2.2 Processing data to be used with Pycatia

The second and main objective of the internship was to convert data from the technical drawing into a sketch on CATIA. To achieve that, it needs a good comprehension of the data given by the contour recognition algorithm and what is needed for the functions of Pycatia.

We previously established how Pycatia works, so we need now to explain how we enter our data from the JSON file into our functions. But firstly, it is necessary to mention that in the algorithm creating the sketch, we browse each list of geometric elements for each contour in the JSON file and call the corresponding Pycatia function. For example, for each contour, we extract the segments, curves, and circles and then we create them without necessarily respecting the direction of the contour. Concerning how the data is processed between the extraction and the creation, we would only focus on three elements, each of which requiring a different process.

### 6.2.2.1 Points and segments

The points and the segments are quite simple examples because they need little modifications. Since we already have the list of our points represented as lists of coordinates, it is quite easy to extract the data and enter it into our function responsible of the creation of a 2D point.

We quickly realized that we do not need to create the points from the list of the nodes that we can find in the JSON file. For each element inside a contour, like a segment or a curve, we would have the points coordinates of its geometry. For example, the list of segments in a contour would include for each segment its identification number and the coordinates of the starting and ending points. So, for each geometry we have access to the points, and we would create them before creating the main component: the line, the curve, or any other geometry.

### 6.2.2.2 Splines

The splines are a more complex example because there is a difference between how they are interpreted in the contour recognition code and how they are parametrized in CATIA. First, the algorithm recognizes the curves as Bezier curves, which means it finds control points, but the curve does not pass through these. Whereas in CATIA, a spline is defined also by control points but passes through these points. The JSON file stores the curves as lists of points, with the control points and the starting and ending points at the beginning and the end of the list.

Since the representation is not the same in CATIA, we must recalculate the Bezier curve from the control points, extract points that are on the curve and send them into our function creating the spline. However, one minor problem persists. The contour recognition algorithm is not able to configure the entire curve correctly, it is cut at some points by segments. From the moment it detects a vertical or horizontal slope or with a value of 1 or -1 in the curve, it cuts it to represent a segment. This is why in the result on CATIA in Figure 10 we do not have one but nine curves.

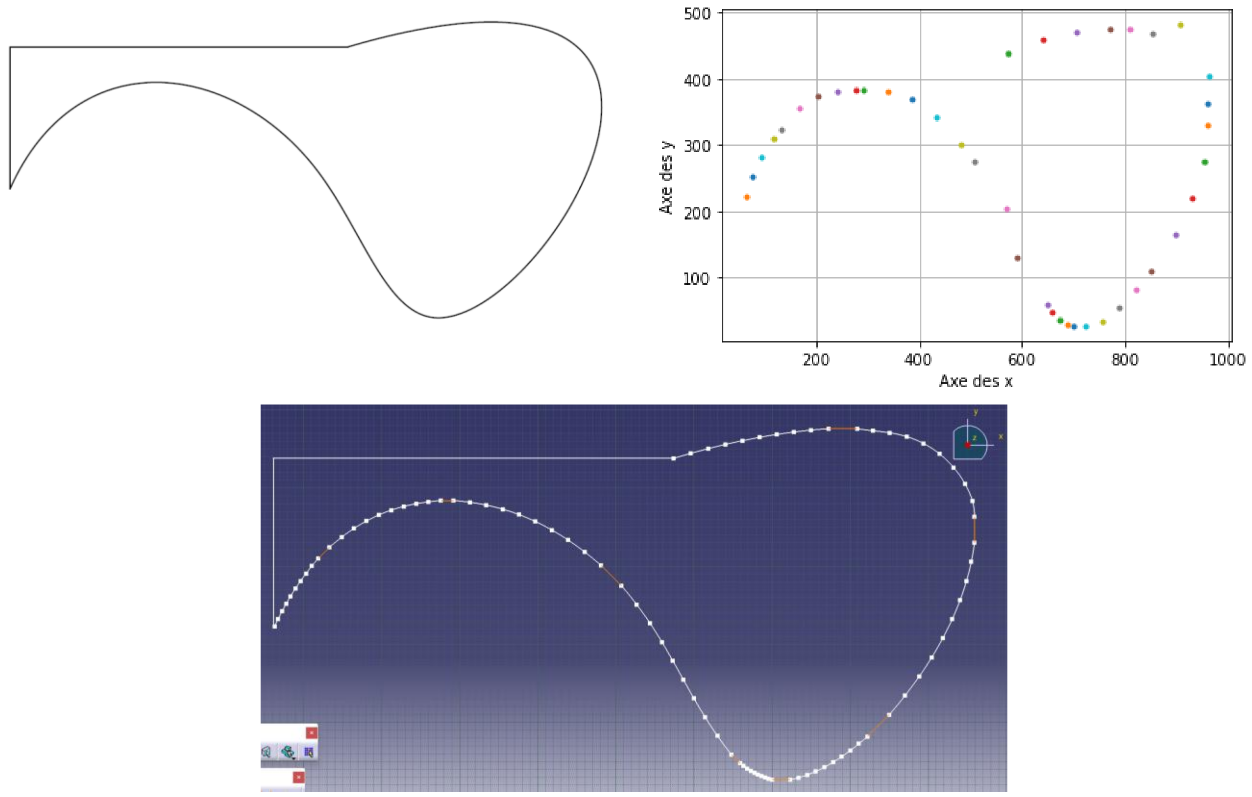


**Figure 10: Original shape, control points of the Bezier curves and result**

### 6.2.2.3 Constraints

In the JSON file, the constraints are stored in a particular format. The name of the constraint corresponds to the nature, and it is numbered. The data inside depends on the nature, but we would normally find all the data of the different elements involved. For example, a constraint called "parallel_segments_0" would be a parallelism constraint and we would have all the data of the two segments involved (their identification and the coordinates of the starting and ending points).

In CATIA, we select the components to constraint them, we do not really see how they are processed. After having registered a macro while doing a constrained sketch, we were able to understand the process and which functions to call with which parameters.

In our algorithm creating the sketch, the constraints come last, after the creation of every geometrical element. During the creation of the 2D elements, we take care to store the created object in a dictionary with their identification number as identification key. Thus, when it comes to the constraint creation, we have access to every 2D objects and their names.

To create the constraint, first we must create "references" objects from the elements involved. To do that, we get the identification of the elements and get the corresponding 2D object in our dictionary previously mentioned. It is important to get that type of object because the Pycatia function only takes in this type of variable. When we have the references, we apply to them the function corresponding to the constraint encountered, then we activate the constraint.

# 7 Multiple contours shapes

After the algorithm for the creation of the sketch proved to be successful on simple shapes, we tried it with shapes involving multiple contours. We have mentioned before that our algorithm browses each contour data in the JSON file, but a few changes had to be made because we discovered some problems in the way the data were stored in the JSON. After rectifying that we were able to get the results in CATIA.

## 7.1 Problems and hypotheses

The first results were remarkably close to the original image, and it was very promising. However, we soon realized abnormalities in the contours. Even if the global shape were close to the original, when we zoom in, we could see a slight gap between the lines, as though there were two overlapping contours. To highlight this problem, we hid the double elements, using the Show/hide tool on CATIA. This way, we were able to see the main contours and the hidden elements in two different pages (Figure 11).

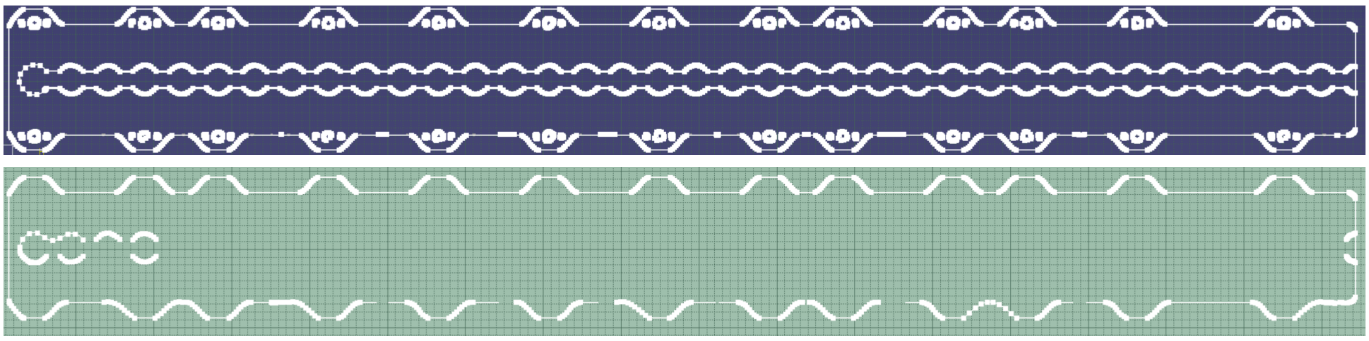


Figure 11: Sketch of the main and hidden contours on the case of a seat track

Another problem was obvious: the accuracy of the contour lining. Some contours were clearly not well represented. We could notice major changes in the curvature of a line while in the original image the contour was simple, like a segment (Figure 12).

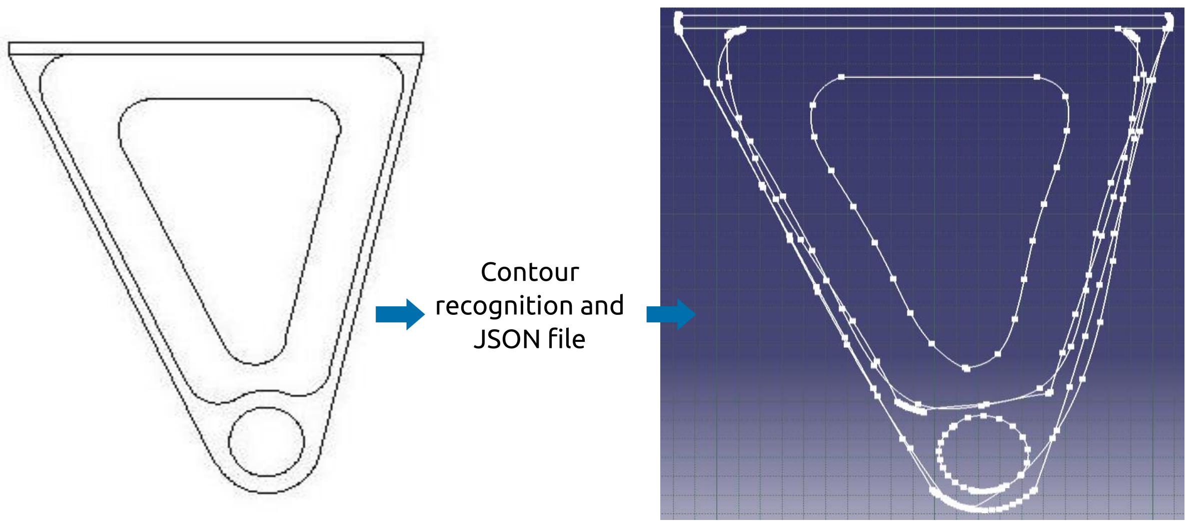


Figure 12: First try for the lug face process

### 7.1.1 Double contours

As mentioned before, we can notice that some contours are doubled. To overcome this problem, we made some assumptions about its origin. Maybe it was an error in the algorithm which could consider the same contour twice in a certain case. After revision of the code, it was not due to this, so we made tests with an image on which we could have the control of the contour thickness.

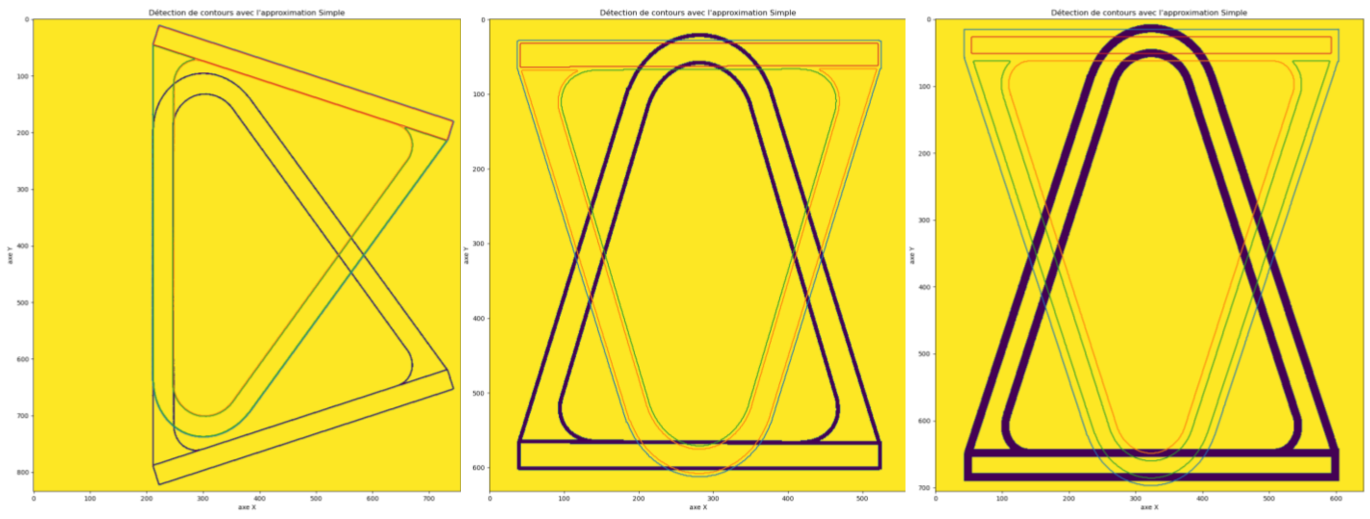


**Figure 13: Study of the double contour problem**

As we increase the thickness of the contour, we notice the contours become distinct and they clearly stand out from each other (Figure 13). It means the root of the problem is the method used for the contour detection: findContours.

Indeed, the method detects the limits of a shape with the same intensity [10]. That is why we must define well the threshold in the parameters of the method. If we have an enclosed shape, we will detect the external and internal contours of the shape, twice the same contour. In this case, we can delete the double contour with a method we developed, relying on the hierarchy of the contour. This method will be explained later.

But if we have two different contours sharing the same boundary (example of the lug Figure 12), it means that one of the contours could be a combination of the other contours. This is more difficult to detect and to prevent, it can be the subject of a paper that would stand by itself.

### 7.1.2 Wrong recognition of the contour by AI

The second problem was the quality of the contour's shape, it was far from the original shape. To solve this, one solution is to train the algorithm of the AI more. This method, using an AI, is called a predictive approach.

Another solution we thought would change the result was to mix a predictive and a morphological approach. The last one is a method in which we realize a simplification of the contour with its main points and from these points we deduce its shape. For example, a square would have four main points and since it has four points, the morphological approach would recognize it as such. While the predictive approach would only tell it is square because the algorithm recognizes a shape it has already seen before and back then we said it was a square.

The morphological approach only relies on mathematics: if it has five sides then it is a pentagon and the same for any polygon. But the limit to this method is when we have circles or rounded shapes, we do not have sides as such. One way is to put a limit to our case disjunction, and from a threshold we consider it to be a circle, but that only works for closed circles. If we had rounded shapes, we would not be able to retrieve the rounding data.

For now, we only use the AI algorithm to define the nature of the contour we set as input. The problem is that even when it is recognized as a rectangle or triangle, there is still actually curves in the contour data. So, it is possible that a complex shape could be recognized as a rectangle. It is not important because the geometry will be created the same way whatever the shape is, but if we want to perfect the process of the contours in the future, depending on their nature, it would be better to get the correct geometry from the first try.

## 7.2 Improvement of contour recognition

As mentioned earlier, challenges arose when applying the algorithm to shapes with multiple contours. Certain issues can be addressed by directly changing the algorithm's approach, while others require a higher-level adjustment in the sequence of actions to proactively prevent these problems. We will focus on two improvements that helped getting a result closer to the reality.

### 7.2.1 Circle detection

Due to the AI's inability to accurately discern the nature of contours, the algorithm treated every contour as a complex shape, regardless of its actual nature. Consequently, both circles and rectangles were processed uniformly, with circles being represented as shapes composed of Bezier curves. This representation introduced challenges, particularly in achieving a closed shape, as illustrated in Figure 14. This example is drawn from the case of the seat track in Figure 11.

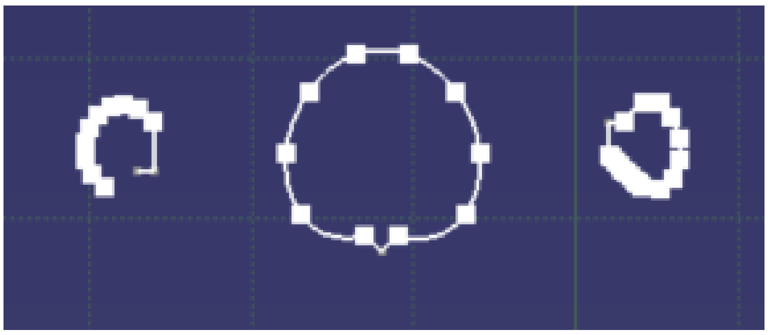


**Figure 14: Detail of the seat track**

With the aim of accommodating future requirements that involve processing each contour based on its unique characteristics, we have opted to introduce a verification step in the shape recognition process, specifically for circles.

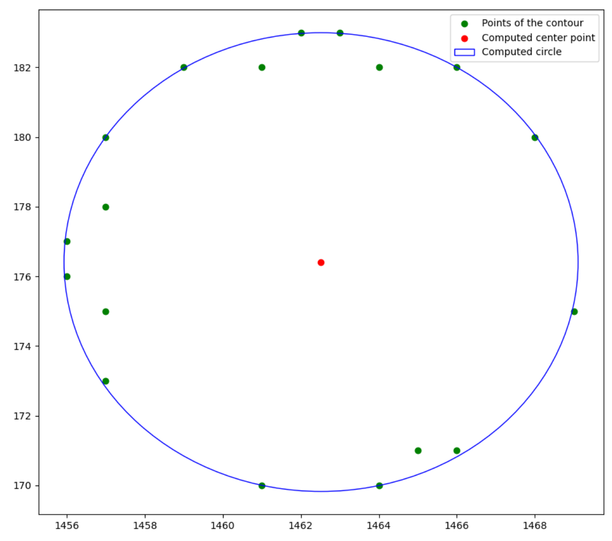
After having recognized the nature of the contour, the algorithm creates a contour with the correct parameters by calling the corresponding class in another script, and it is at that step that we short circuit the algorithm by treating every shape as a complex one.

The improvement would involve calling the circle contour definition at this step. Also, we would have to slightly change the Circle class, so it considers the data we enter as input and gives us in return the correct output. For example, we want to enter the same data for the contour as we have for any contour, but it returns empty lists of segments and curves, and a list for the circle parameters (centre point coordinates and radius).

But to call this class, we need to detect the contour as a circle, or check if the circle detected is really a circle. That is where the verification step comes in. In the shape recognition script, after we get the nature of the contour, we check for each of them whether its nature is a circle or not. We do it for every contour because as we mentioned before, the AI is not able to classify them correctly. But, as the AI improves, it will become necessary to perfect it.

For the verification step, we retrieve the points of the contour and calculate the assumed centre point as if it was a circle and an average radius (Figure 15). Then, we loop again through the points and check if the distance between them and the assumed centre point lies within a range of values around the radius. If it is not the case for at least one point, it means it is not a circle, and we redefine the nature of the contour as complex.

Otherwise, we keep the calculated values for the coordinates and the radius and the nature as a circle. We can see on Figure 16 the result obtained after cleaning manually the double shapes (problem seen on Figure 8). The circles are detected correctly, and we have a sufficient accuracy as they are still aligned.



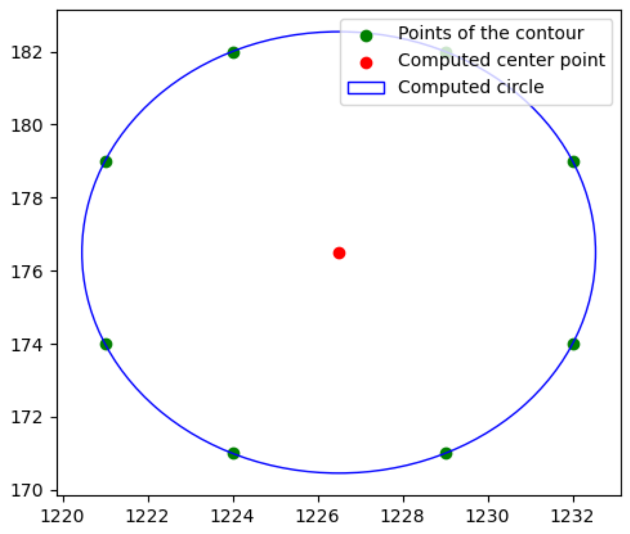


Figure 15: Examples of checked contours (complex on the left, circle on the right)

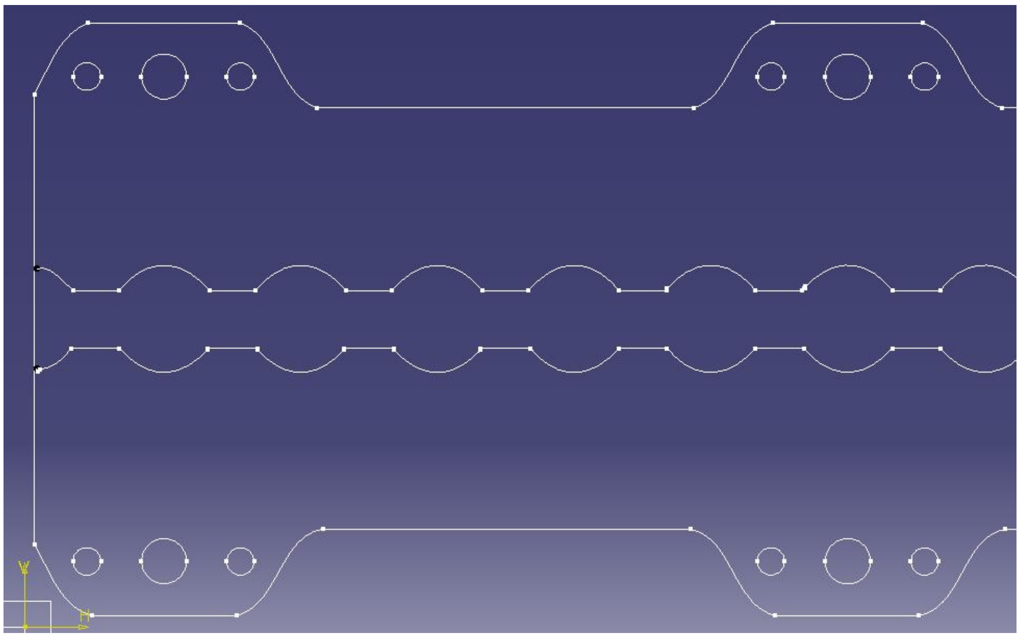


Figure 16: Cleaned result with closed circles

## 7.2.2 Removing double contours

As mentioned earlier, the contour recognition algorithm detects every zone limit in the picture. Thus, if a shape is closed, like a simple circle, we would detect the external and internal contours of the circle; we would have two contours representing the same shape. Beforehand, it needs to be understood that the contour retrieval method also detects the boundaries of the picture. Thus, in the contours data, the first contour (number 0) corresponds to the external shape of the picture, a rectangle, and will be ignored in the rest of the analysis since it will be automatically deleted.

A first approach to avoid this double contour problem was to find a way to fill some zones of the drawing. That way, the algorithm would not detect the internal contours since they do not exist anymore. For this, we used the library cv2 again and its function floodFill. However, this function automatically inverts the contrast of the picture, which means that we do not detect the image boundary anymore, but the drawing becomes white on a black background.

Another solution was to select the zones to fill, but it was more complex since there is no function able to directly do this. We needed to define a mask on the picture, apply it and superimpose it on the original image. Obviously, the results were not as good as expected so we looked for another solution.

That is why we developed a function which deletes the redundant contours based on their hierarchy. This concept is crucial to understand since it depicts the ontology between the contours, and it is the heart of the function which already existed to delete the redundant contours. The hierarchy is represented as a matrix of numbers, each line would correspond to a contour and each column specifies its relationships with the other contours: [Next, Previous, First child, Parent].

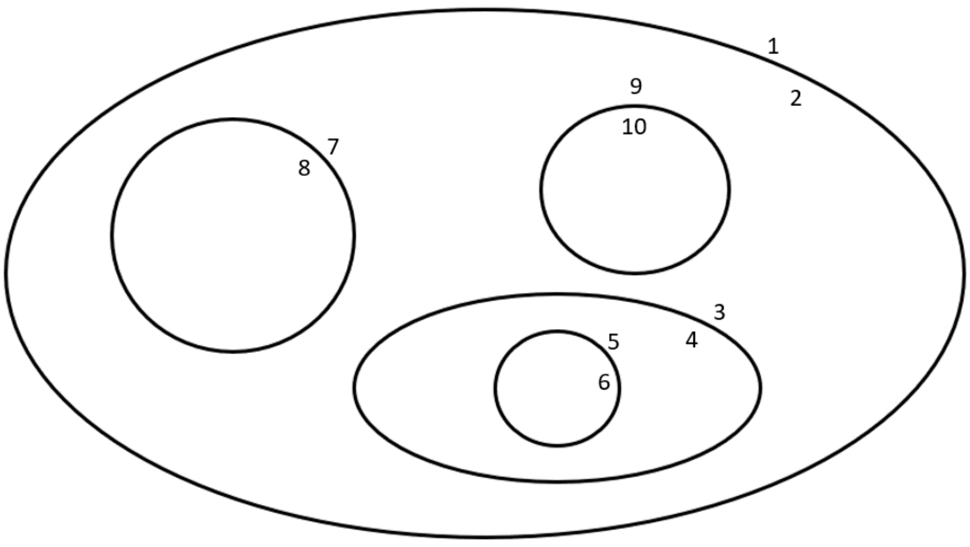


**Figure 17: Reference picture for hierarchy study**

On Figure 17, contours 2, 4, 6, 8 and 10 are internal and are the ones we want to remove. The hierarchy of the contours depends on the contour retrieval mode we set in the method findContours. We briefly described earlier how the contour retrieval mode works, so we will now focus on the hierarchy.

In the case of Figure 17, it would return this hierarchy matrix:

$$\begin{bmatrix} -1 & -1 & 2 & 0 \\ -1 & -1 & 3 & 1 \\ 7 & -1 & 4 & 2 \\ -1 & -1 & 5 & 3 \\ -1 & -1 & 6 & 4 \\ -1 & -1 & -1 & 5 \\ 9 & 3 & 8 & 2 \\ -1 & -1 & -1 & 7 \\ -1 & 7 & 10 & 2 \\ -1 & -1 & -1 & 9 \end{bmatrix}$$

If we take the example of contour C3 (third line in the matrix), we have this: [7 -1 4 2]. It means that contour C7 is the next one in hierarchy, which is obvious since they are inside the same shape, contour C2, which is also their parent. Then, we have -1 as the value for the previous contour, meaning there is no previous contour. C4 is the first child of the contour since it is the internal contour of the ellipse [11].

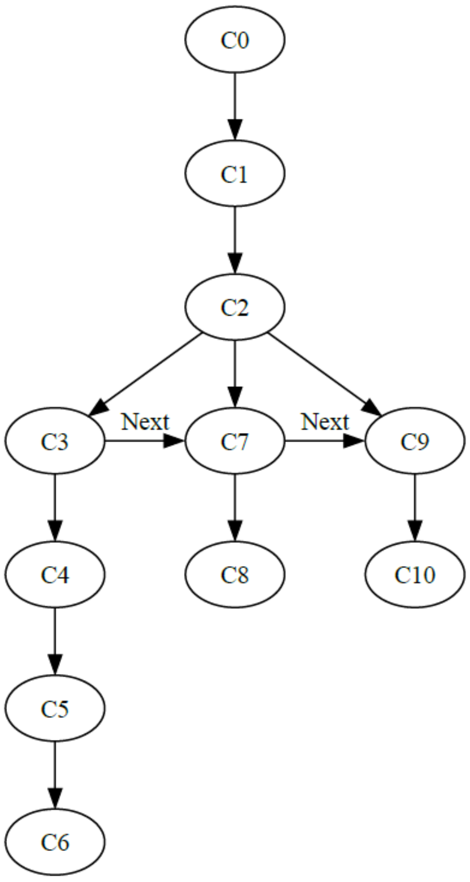


Figure 18: Ontology of the contours of the reference picture

From this we can draw a quick ontology of the contours (Figure 18). We can notice contour C0 which corresponds to the limits of the picture, and which is ignored, that is why it does not appear in the hierarchy matrix. We are also able to see that the contour we want to remove are alone at their rank in hierarchy, which means that they have no next nor previous contours, so their hierarchy begins with [-1 -1 _ _].

We wanted to change the function detecting the redundant contours since it was not correctly configured. Indeed, we noticed that it deleted only the contours having no next, no previous and no child contours. If we take the example of Figure 17 and the scheme in Figure 18, it will mean that it removed only the contours at the end of each branch (6, 8 10). So, this function would not delete contour C4 even if it is a redundant contour.

However, to differentiate contour C7 from the contours to delete, we need to make sure that we are not dealing with a "parent" contour. Indeed, the external contours act like parents, they contain other contours.

So, in our algorithm we define a "parent state" that we read during each loop and redefine according to the contour we are currently analysing and the one we analysed the loop right before. The logic states that every first child of a parent contour corresponds to an internal contour. So, if the one before was a parent, and if the one we are reading has no next nor previous contours, it means we are dealing with a double contour and we remove it (result in Figure 19).

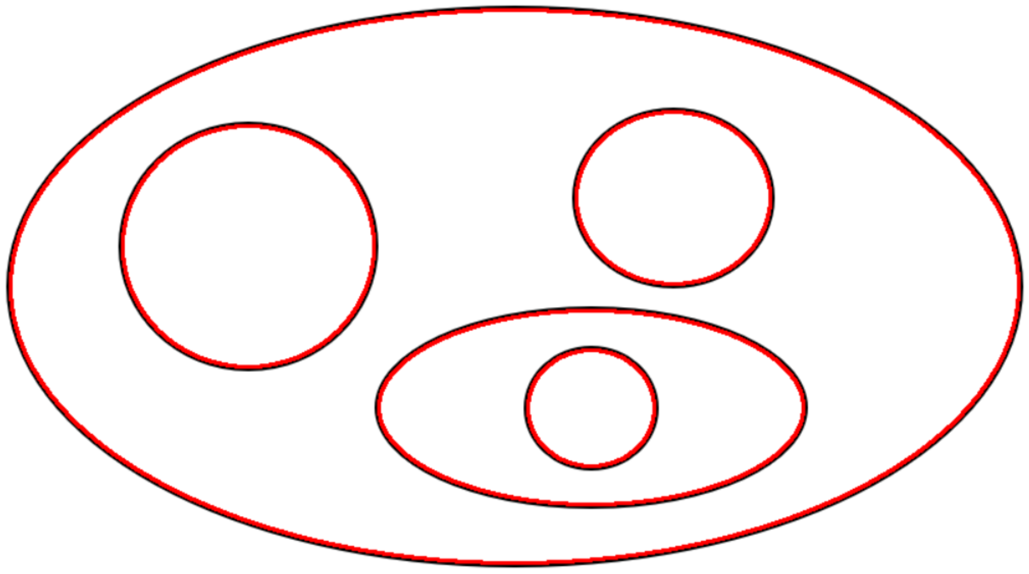


**Figure 19: Contours to remove after the analysis of the hierarchy**

We notice that only the internal contours are deleted and contour C7 has been recognized as a parent contour, and so has been kept. This function will allow us to get better results in the representation of the technical drawing. We will not lose data because of a bad understanding of how the contour retrieval mode works.

The next step would be being able to detect the combination between the contours. For example, two contours could be inside the same parent, share some limits with the parent while having a different shape. We would have twice the same boundary drawn on the sketch while having two distinct contours. This, however, represents too much work to be studied in this subject.

# 8 Conclusion

In the context of my studies in Engineering at the ENSTA Bretagne and Prague Technical University, I realized my end-of-study internship at Capgemini Engineering in Blagnac, France. Working in the research department was for me a rich experience because I worked on mechanical subjects while applying automation tools. By participating actively in the company life and bringing a new point of view on the subjects, I have been able to complete my goals and return a satisfying result to the team.

Indeed, my internship has been driven by the objective of parametrizing and automating the creation of CATIA models. After being able to parametrize an entire bike frame from a control sheet on Excel and a CATIA script, I could apply this knowledge to the automation process. This latter goal has been more challenging. The logical sequence of action to create the model was simple to define, but we realized that the input data were not structured or clear enough. So, I had to work on image processing and on the contour recognition algorithm to get the correct results.

Working in such various fields made me put in perspective what I would expect to do in the future. Combining both programming and mechanics was thrilling, working on Python to get a result on CATIA was something special. It allowed me to increase my knowledge in both those tools but also in automation methodology, with a touch of AI experience.

Thanks to this experience, I now have a clearer idea of what I want to do after my studies. Working in an IT company on subjects involving mechanics was a first for me, and I really enjoyed it. I discovered a new way to work on subjects I have always been interested in, but with tools that embody the future of engineering. And again I would like to thank all the people at Capgemini or at any point during my higher education years who helped me along my way.

# 9 Sources

[1] ‘Capgemini’, *Wikipédia*. Aug. 08, 2023. Accessed: Oct. 26, 2023. [Online]. Available: https://fr.wikipedia.org/w/index.php?title=Capgemini&oldid=206772009

[2] Capgemini, ‘Capgemini Engineering’. Accessed: Aug. 11, 2023. [Online]. Available: https://www.capgemini.com/fr-fr/notre-groupe/nous-connaitre/nos-marques/capgemini-engineering/

[3] A. B. Harish and A. R. Prasad, ‘Photo2CAD: Automated 3D solid reconstruction from 2D drawings using OpenCV’, 2021, doi: 10.48550/ARXIV.2101.04248.

[4] L. Governi, R. Furferi, M. Palai, and Y. Volpe, ‘3D geometry reconstruction from orthographic views: A method based on 3D image processing and data fitting’, *Comput. Ind.*, vol. 64, no. 9, pp. 1290–1300, Dec. 2013, doi: 10.1016/j.compind.2013.02.003.

[5] ‘CATIA’, Dassault Systèmes. Accessed: Dec. 04, 2023. [Online]. Available: https://www.3ds.com/products/catia

[6] ‘CATIA Automation Home Page’. Accessed: Oct. 26, 2023. [Online]. Available: http://catiadoc.free.fr/online/CAAScdBase/CAAScdAutomationHome.htm

[7] ‘Pycatia documentation’, GitHub. Accessed: Nov. 27, 2023. [Online]. Available: https://github.com/evereux/pycatia/tree/master/examples

[8] C. EL BEZZAR, ‘Reconstruction de topologies 3D à partir d’informations de coupes ou projection 2D’.

[9] ‘OpenCV: Structural Analysis and Shape Descriptors’. Accessed: Nov. 30, 2023. [Online]. Available: https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a

[10] ‘Python | cv2 findContours() Method - Java2Blog’. Accessed: Nov. 22, 2023. [Online]. Available: https://java2blog.com/cv2-findcontours-python/

[11] ‘OpenCV: Contours Hierarchy’. Accessed: Nov. 22, 2023. [Online]. Available: https://docs.opencv.org/4.x/d9/d8b/tutorial_py_contours_hierarchy.html

## About Capgemini Engineering

As the world leader in engineering and R&D services, Capgemini Engineering applies in-depth sector knowledge and mastery of the latest digital and software technologies to support the convergence of the physical and digital worlds. Combined with the Group's full range of capabilities, Capgemini Engineering helps its clients accelerate their transformation towards the Intelligent Industry. Capgemini Engineering employs 65,000 engineers and scientists in over 30 countries, and operates in sectors such as Aerospace, Defense, Naval, Automotive, Rail, Infrastructure & Transportation, Energy, Utilities & Chemicals, Life Sciences, Communications, Semiconductors & Electronics, Manufacturing & Consumer Goods, Software & Internet.

Capgemini Engineering is part of the Capgemini Group, a responsible, multicultural global leader with 360,000 people in over 50 countries. A strategic partner to companies as they transform their businesses by harnessing the full power of technology, the Group is guided every day by its raison d'être: unleashing human energies through technology for an inclusive and sustainable future. With 55 years of experience and extensive industry expertise, Capgemini is recognized by its clients for meeting all their needs, from strategy and design to operations management, leveraging innovations in the ever-evolving fields of cloud, data, Artificial Intelligence, connectivity, software, digital engineering, and platforms. The Group achieved sales of 22 billion euros in 2022.

Get the Future You Want | www.capgemini.com/engineering