POLITECNICO DI TORINO

Master's Degree
in Computer Engineering

Master Thesis

Development and evaluation of a platform for the automatic extraction and processing of PCB information from schematics and placement files

Supervisors
Prof. Paolo Bernardi
Ph.D. Giorgio Insinga

Candidate
Alessio Cappello

Academic Year 2023-2024

Did you realize
that you were a champion
in their eyes?

Summary

The invention of the Printed Circuit Board (PCB) led to different breakthroughs in the electronic world. PCBs are the crucial basis for virtually every electronic device and find application in several fields, ranging from medical to automotive and IoT to aerospace. An extensive number of components find a place on the topmost or the bottom layer, and they are connected following an intricate tangle of connections. Two types of files are essential for understanding how a PCB works: placement (describes the layout of components on one layer) and schematic (reports the connections and the pins for each component).
Manually testing a PCB requires looking for a specific component or connection in a placement or schematic file, which in turn requires the tester to look alternatively between the board and the manual. Avoiding looking at the manual for each component and connection would result in an acceleration of the testing phase, apart from reducing human errors: this can be achieved by synthesising all the information required for testing in an Augmented Reality (AR) application. The idea is to have a virtual layout that superimposes the board itself, which allows the tester to verify the correct functioning of the components without needing to inspect the manual.
Certainly, some information processing is needed to reach the goal: by just giving as input the placement and the schematic files of interest, the proposed program automatically extracts the information required for the AR testing phase. The application relies on text extraction packages, PDF manipulation tools and image processing libraries. The user has to upload the manual and select the placement or schematic files to analyse: then an ad hoc pipeline is launched distinguishing the operations based on the type of file currently processed. Some manual corrections may be required since components are obtained from placement files using an edge detection algorithm, that may encounter troubles and limitations.
At the end of the process, the extracted and elaborated information is written in some textual files respecting a format suitable for the AR application, keeping track of only those components appearing in placement and schematic files of interest.

Acknowledgements

Non posso che iniziare i ringraziamenti se non rivolgendomi alla mia famiglia: vi ringrazio per tutto l'amore e il supporto, nonostante spesso e volentieri non me lo meriti realmente. Seppur da un punto di vista scolastico e accademico sia stato il vostro orgoglio, non so se lo stesso può essere detto per il mio ruolo di figlio e fratello. Non si può essere perfetti, ma grazie per accettarmi per come sono.
Ci tengo a ringraziare il prof. Paolo Bernardi e Giorgio Insinga per l'opporunità datomi. Apprezzo che mi abbiate lasciato carta bianca per alcune scelte funzionali e stilistiche, ma non avrei mai ottenuto il risultato finale senza la vostra guida e il vostro riscontro.
Mentirei se dicessi che questi due anni a Torino non siano stati impegnativi e turbolenti talvolta: fortunatamente, però, ho incontrato tante persone con cui condividere momenti sereni e struggenti, e di questo ne sono profondamente grato. Ringrazio in particolare Francesca, Enf, Marco, Foros, Simone e Kaliroi: in modi diversi, siete state le persone che più mi sono state vicino e su cui ho potuto sempre contare.
Una menzione speciale sento di doverla fare nei confronti di Davide, l'amico con cui ogni sciocchezza può trasformarsi in una risata (vedi l'evento) e al tempo stesso posso avere un confronto costruttivo, e Angelo, la persona che più vedevo antitetica a me inizialmente nonostante la vicinanza in termini di provenienza, ma che si è poi rivelato molto simile a me.
Un supporto importante l'ho ricevuto anche da diverse persone del mio paesino sperduto di una qualunque provincia siciliana: un ringraziamento particolare va dunque ad Antonio, Nicolas, Mattia e Klevio, per non avermi fatto tagliare definitivamente i contatti con la nostra stupenda ma dannata terra.
Infine, ringrazio in generale tutte le persone che, in entità più o meno rilevante, hanno contribuito a percepire questo percorso meno difficile di quello che in realtà è. Non posso prevedere il futuro e non so quanti di voi saranno ancora al mio fianco, ma non posso che esservi riconoscente.

Contents

List of Tables

List of Figures

Finally free, the butterfly sheds light on
situations
That the caterpillar never
considered
Ending the internal struggle
Although the butterfly and caterpillar
are completely different
They are one and the same

Chapter 1

Introduction

1.1 PCB Testing

Since the invention of the Printed Circuit Board (PCB) in 1936 by the Austrian engineer Paul Eisler, the electronic world experienced a series of breakthroughs. PCBs enabled new levels of miniaturization, allowed the development of advanced circuits and made industrial production easier and more reliable. Since its invention, the density of components in a PCB has always been increasing. Nowadays, a PCB is a complex mix of disparate components connected by small stripes of copper called traces. A modern PCB has several interconnected layers, with traces that can freely change layers through vias.
The design of these boards requires a not indifferent amount of effort, but it is just the beginning: once the PCB has been manufactured, it should be fully tested to assess its correct behaviour. Connections between components need to be checked, as well as shortages between signals near each other. Most of the tests are often automatised in the mass production environment when the design is fixed and no debugging has to be performed. However, manual tests are essential during the prototyping and ramp-up phases.
To ease the testing phase, especially when manual intervention is needed, some test points that are easily accessible to the test engineer's oscilloscope probes are left on the PCB. When a component has been selected to be tested, the test engineer needs to find the right location to sample on the PCB under test, and finding the correct location on a complex board full of components and test points is not a trivial task.
The entire process can be summarised in the following steps:

- Select a component to test from the board's schematic
- Find the component's location on the board's placement file

- Finally, look for the component on the PCB in the location found in the placement file

These steps must be repeated for each component the test engineer has to test. With complex PCBs, a significant amount of time is spent looking for a specific component. Additionally, human errors are common and can lead to testing wrong components with related test quality issues.
Several solutions already exist in various augmented reality forms to mitigate the risks of testing the wrong components and to let the test engineer save some time. The simplest solution is to point a PC webcam to the PCB and have the augmented reality application display the results on the monitor Kowalke et al. [2024]: this is not convenient, though, as the information is projected on a nearby computer instead of having it overlaid directly on the board. Another solution requires a complex setup with a projector and several sensors to project the information directly on the physical board Ojer et al. [2020]. The drawback of this solution is that the setup is not easily portable, requiring a certain calibration effort.
Furthermore, all the methods present in the literature require access to the full design files of the PCB itself. The design files are available only to the designers of the PCB, but not to the customers who want to test their board or even the manufacturers who may receive only the production files.

1.2 Contribution

The innovative idea to speed up PCB testing is to use only the schematics and placement files: the necessary information is extracted and manipulated so that it can serve as the basis for an Augmented Reality application.
By equipping a headset, the test engineer would have access to all relevant information (such as components' location and connections) without having to move their eyes recurrently from the board to the manual and back again.
In this work, we show how information is extracted from the placement files and schematics and processed in a format that will later be required for the Augmented Reality layer. In this work, the term "placement" refers to the silkscreen, the topmost layer of a PCB, used as the reference that indicates the placement of the components on a PCB. This is, therefore, the first step towards achieving the proposed end goal. This work is structured as follows:

- In Chapter 2, a brief description of the electronic support involved is treated, talking about PCBs and related files.
- In Chapter 3, there is a short review of the existing software packages that support the application.

- In Chapter 4, the application's functionalities are broken down and explained in detail.
- In Chapter 5, conclusions are drawn and possible extensions and future work are delineated.

Chapter 2

Electronic Support

2.1 Printed Circuit Board

A Printed Circuit Board (PCB) is a fundamental element in nearly all contemporary electronic devices, representing simultaneously innovation and functionality in the electronic world. A PCB represents a well-organised complex of electronic components, including microprocessors, resistors, capacitors and connectors.
PCBs are practically everywhere. In consumer electronics, PCBs are found in smartphones, laptops, tablets and home appliances, providing efficient and reliable performance. In the automotive industry, they are used in engine management systems, navigation devices and sensors, enhancing vehicle safety and connectivity. Aerospace and defence sectors use PCBs in radar, communication systems and control units, benefiting from advanced PCB technologies like high-frequency and rigid-flex boards to ensure optimal functionality in demanding environments.
The development of a PCB starts with the design stage, during which engineers and designers carefully compose the circuit layout. This stage encompasses the creation of detailed schematics, which are subsequently transformed into a physical arrangement that maximises the operational efficiency and exploits the spatial limitations of the electronic device.
Through various chemical and mechanical techniques, the copper layer is worked to create the circuit pathways that will eventually accommodate electronic components. These pathways, known as traces, facilitate the essential electrical connections among different circuit elements, ensuring the proper routing of signals and power across the board. Furthermore, the PCB may incorporate vias, small plated openings that enable connections between multiple layers in multi-layer PCBs.
PCBs give benefits for their characteristic to enhance the assembly process and reliability of electronic circuits. By offering a solid plan for component placement and soldering, PCBs reduce the likelihood of errors and simplify the complex wiring

that would otherwise be necessary for a point-to-point assembly. This results in more compact, robust and economically viable electronic devices.

2.2 PCB Files

The design and manufacturing of a PCB depend on a variety of specialised files that describe every detail of the board's layout, structure and component placement. Among these files, it is possible to list: Gerber files, used to describe the PCB's layers, including copper traces, solder masks, silkscreens and the board outline, Excellon Drill files, used to specify the locations, sizes and depths of holes in the PCB, Bill of Materials, used as a list of all components used on the PCB, including part numbers, descriptions and quantities, essential for component sourcing and assembly. Two other files are of interest in this work, analysed in the two following subsections: schematics and placements files.

2.2.1 Schematic file

A schematic file is a detailed diagram illustrating the interconnections among various electronic components constituting an electrical circuit. Each component is drawn as a specific symbol that indicates its type and electrical properties such as value, polarity and pin arrangement.
Schematics allow designers to grasp the flow of signals, the power distribution and the interactions between components, thereby facilitating the identification of design issues before moving to the physical layout stage. Indeed, test engineers can perform simulations directly from the schematic, without the need for physical prototyping.
An example schematic image is shown in Fig. 2.1.

2.2.2 Placement file

A placement file acts as a map of the electronic components that need to be placed on the board: it details the exact locations, orientations and specifications of all components. This is a crucial file for automated assembly machines, providing precise coordinates for each component's placement on the PCB along with essential data such as component reference namings, rotation angles and side of the board (top or bottom) where the component should be mounted.
These files serve as a relevant reference during the inspection and testing phases, helping engineers check that components are correctly placed according to the design specifications: any discrepancies between the placement file and the assembled board can be quickly identified and corrected. An example placement image

is shown in Fig. 2.2.

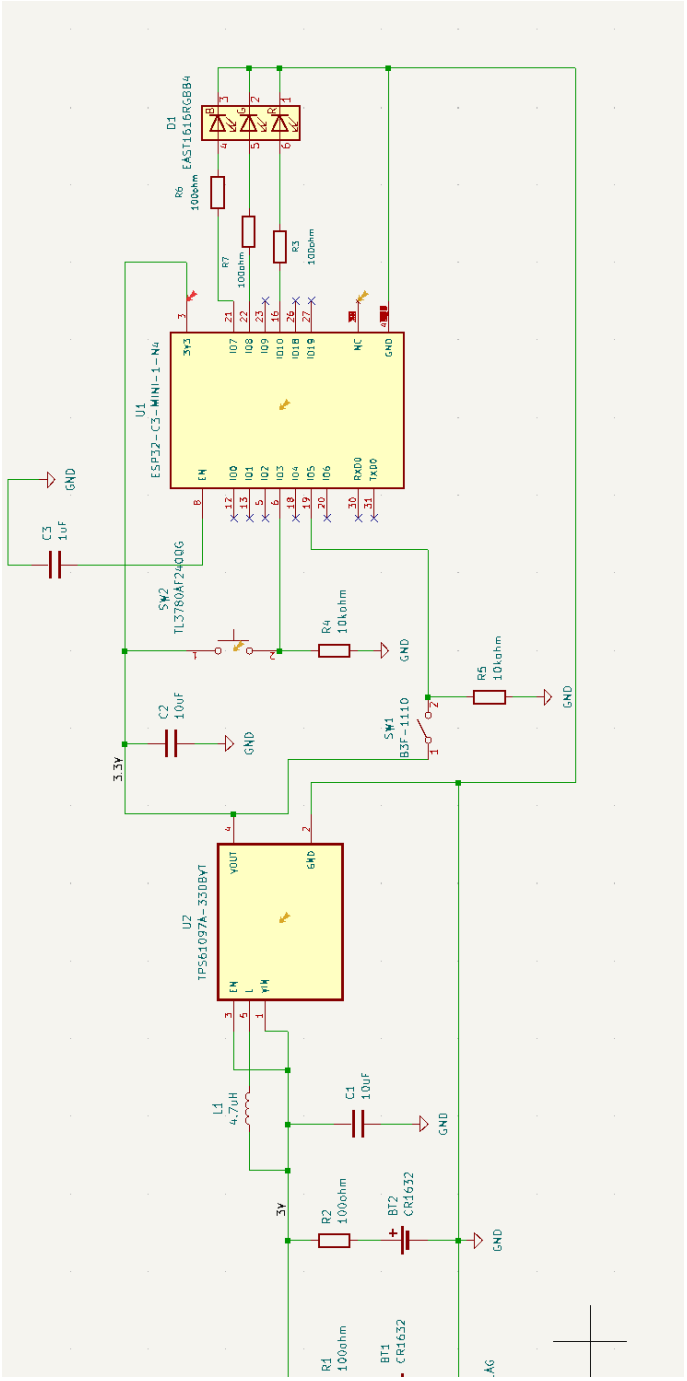

Figure 2.1. Schematic file image example

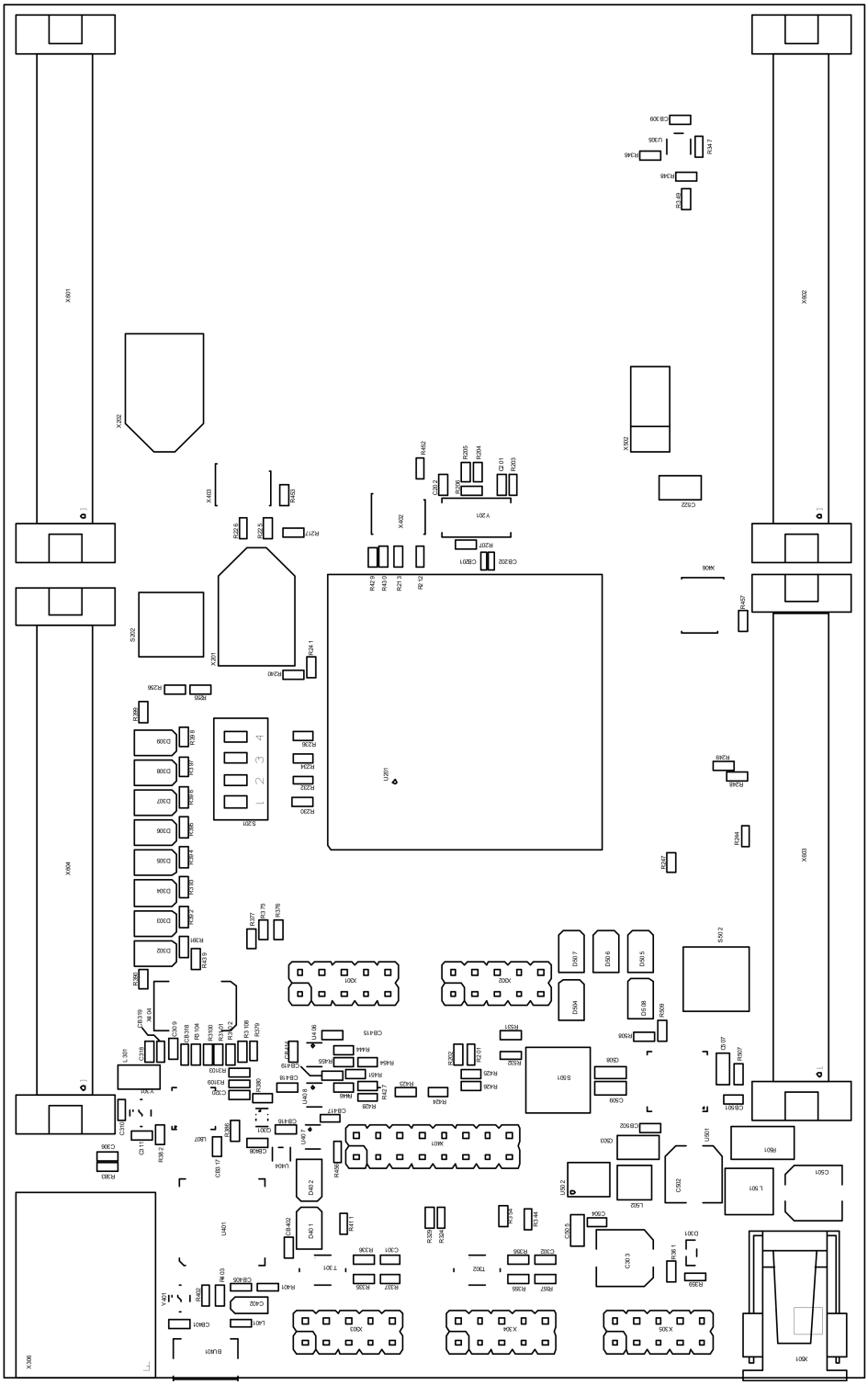

Figure 2.2. Placement file image example

Chapter 3

Software Support

3.1 Packages

The proposed tool has been written entirely in Python, with the support of different types of packages: PDF manipulation, image processing, and graphical user interfaces.

3.1.1 PDF manipulation

The following packages have been used to work on PDF files, a usual format for the boards' manuals: *PyMuPDF* (fitz), *pdfplumber* and *pdf2image*.
PyMuPDF Artifex [2024] is an open-source high-performance Python library for data extraction, analysis, conversion and manipulation of PDF and other documents. It is the reference package when the application needs to open, display and manipulate PDFs: for instance, when it is required to extract text from schematic files, this library is the one to be used since it provides the feature of extracting text blocks.
Pdfplumber jsvine [2015] is a package specifically designed for extracting data from PDF files, with high performance in obtaining text, tables, images and other contents present in PDF files. It is effective in PDF parsing, enabling some advanced functionalities. In the application, it is used to extract text from placement, since it can extract single words effectively.
Pdf2image Belval [2017] is a library designed to convert PDF pages into images. It acts as a wrapper around PDF rendering libraries, in this case Poppler freedesktop.org [2005], facilitating the conversion of each page of a PDF into high-quality images in formats like PNG, JPEG or TIFF. In the application it is used to convert the schematics and placement files into images, making them suitable for the following steps of image processing.

3.1.2 Image processing

The following packages have been used to open, manipulate and extract information from images: *PIL* and *OpenCV*.
PIL (Python Imaging Library) Clark [2015] is one of the most popular libraries for image processing. It provides several tools for opening, manipulating and saving different image file formats. It is the default choice when it is needed to only show images without performing peculiar manipulations.
OpenCV (Open Source Computer Vision Library) Bradski [2000] is an open-source library widely diffused for real-time computer vision and image processing tasks. It encapsulates advanced tools and algorithms that cover a broad range of image and video processing needs. Inside the application, it is used to draw on existing images or from scratch. For instance, it is used to visualise the outcome of the shape detection (treated later) after applying the Canny edge detection algorithm (treated in Appendix A).

3.1.3 Graphical User Interface

Every Graphical User Interface (GUI) has been developed using *tkinter* Lundh [1999], the standard GUI toolkit for Python used to create intuitive graphical user interfaces quickly and easily. It offers numerous widgets such as buttons, labels, text boxes, menus, canvas, and more. It is used to create every GUI of the application: manual loading, components manual correction and pin definition, placement and schematic components highlighter.

Chapter 4

Application overview

4.1 Execution flow

The application starts showing a GUI where the user has to upload the PDF board manual. Once opened, it is possible to go through the PDF pages and define several crops; for each crop, the user has to specify whether it is a placement or a schematic, and in the former case if it represents the top or the bottom layer. After this phase, each crop goes through the text extraction phase: the entire page where a crop has been taken is processed, but only the words retained inside the rectangle are kept. The outcome of this phase is a textual file per crop containing information about the extracted text.
At this point, crops representing placement files go through the shapes assignment phase: the Canny edge detector analyses the crop (converted from PDF to image) to extract edges to be composed into significant shapes (shapes with a tiny area are discarded). For each shape, the geometric centroid is calculated. A trivial algorithm assigns a component label previously extracted to the shape whose centroid is the closest. This phase is not error-free: a manual correction GUI has been set up such that the user can correct any mistakes or misses, other than defining the number of pins and geometry for a shape. The outcome of this phase is a textual file per placement file containing the updated information about the defined components.
Schematic crops are further processed after the user ends the manual correction phase for placement files: only the common components between placement files and schematics are kept in a textual file. Another textual file is produced about the pins: at the current state, the pins are extracted by looking inside the components that accommodate pins. A specific colour is used to define these components inside a schematic: in this way, only the labels inside areas of the given colour are

kept. Furthermore, a support file in the desired format for the AR phase is produced for each placement file.
In the end, a highlighting GUI allows the user to select components on a schematic crop by clicking the relative button. When the user switches to a placement crop, the components relative to the clicked buttons are highlighted.
The execution flow is schematised in Fig. 4.1.

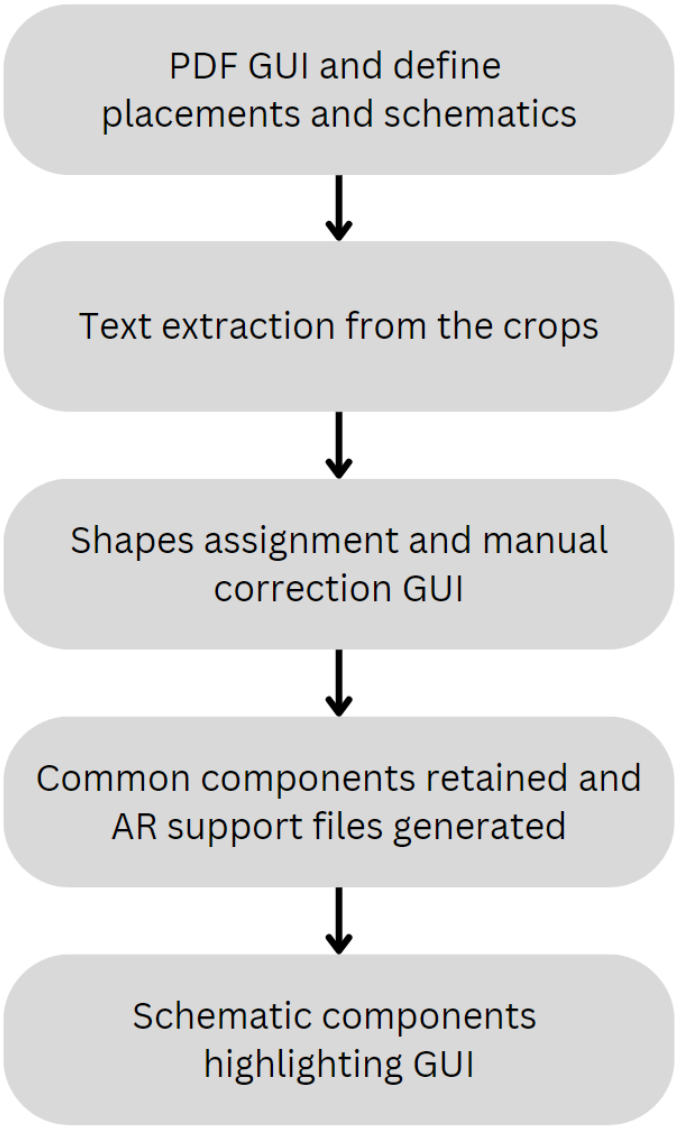

Figure 4.1. Execution flow diagram

4.2 Start menu

The application shows a small window asking for the action to be performed when launched. As shown in Fig. 4.2, there are three buttons: *Load a new manual*, *Correct centroids* and *Compare saved imges*.
Load a new manual is used to follow the entire execution flow, allowing the user to upload a PDF file of a board manual and select the crops of placements and schematics, as explained in Sec. 4.3.
Correct centroids allows the user to keep the crops defined in the last session and move directly to correcting the centroids and the pins, as explained in Sec. 4.5.2.
Compare saved images is used only to access the highlighting GUI to identify where a component is located on both placement and schematic, as explained in Sec. 4.7.

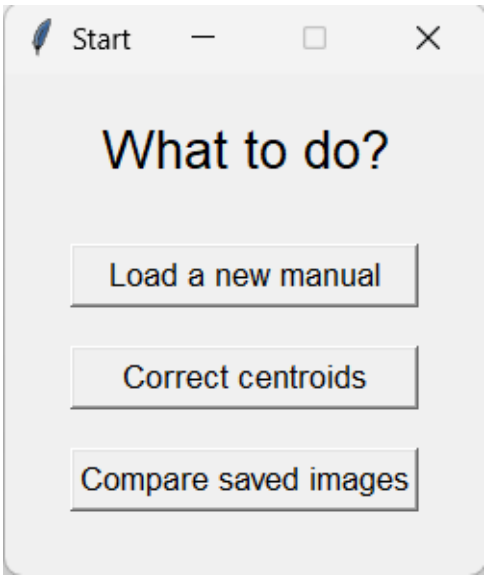

Figure 4.2. Start menu

4.3 PDF GUI

The application starts with an empty window, where some buttons appear as shown in Fig. 4.3. The *File* menu allows the user to load and display a PDF from mass storage.

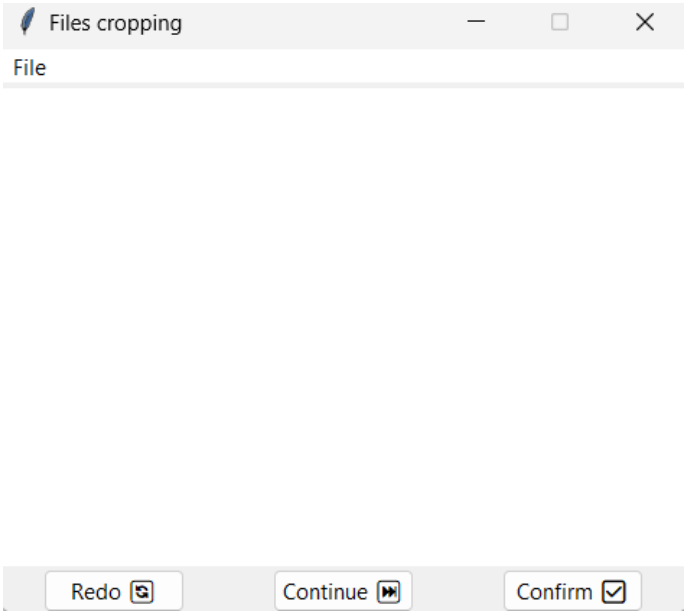

Figure 4.3. Initial GUI, no PDF loaded

After opening a PDF of a board, the first page is displayed, as shown in Fig. 4.4. A top bar is now available, containing arrow buttons to navigate the file and an entry field to input directly the page number to display after hitting the Enter button. The keyboard's arrow buttons are handled in a manner that the user can also navigate the document by pressing them.
At this stage, the user can define the placement and schematic crops to analyse. To do so, they can click and drag over the window to define a red rectangle: if mistakes are made, the user can click the Redo button or just click and drag again. An example of this action is shown in Fig. 4.5
To confirm a crop, it is sufficient to click on the Confirm button. A small dialogue window appears, asking to select whether the crop represents a placement or a schematic. In the former case, it also asks to choose between top or bottom. An example of the window is shown in Fig. 4.6.
Once all the crops of interest are defined, it is possible to pass to the next stage by pressing the Continue button at the bottom. The outcome of this phase is a textual file (referred to as GUI file) containing the absolute path of the input PDF

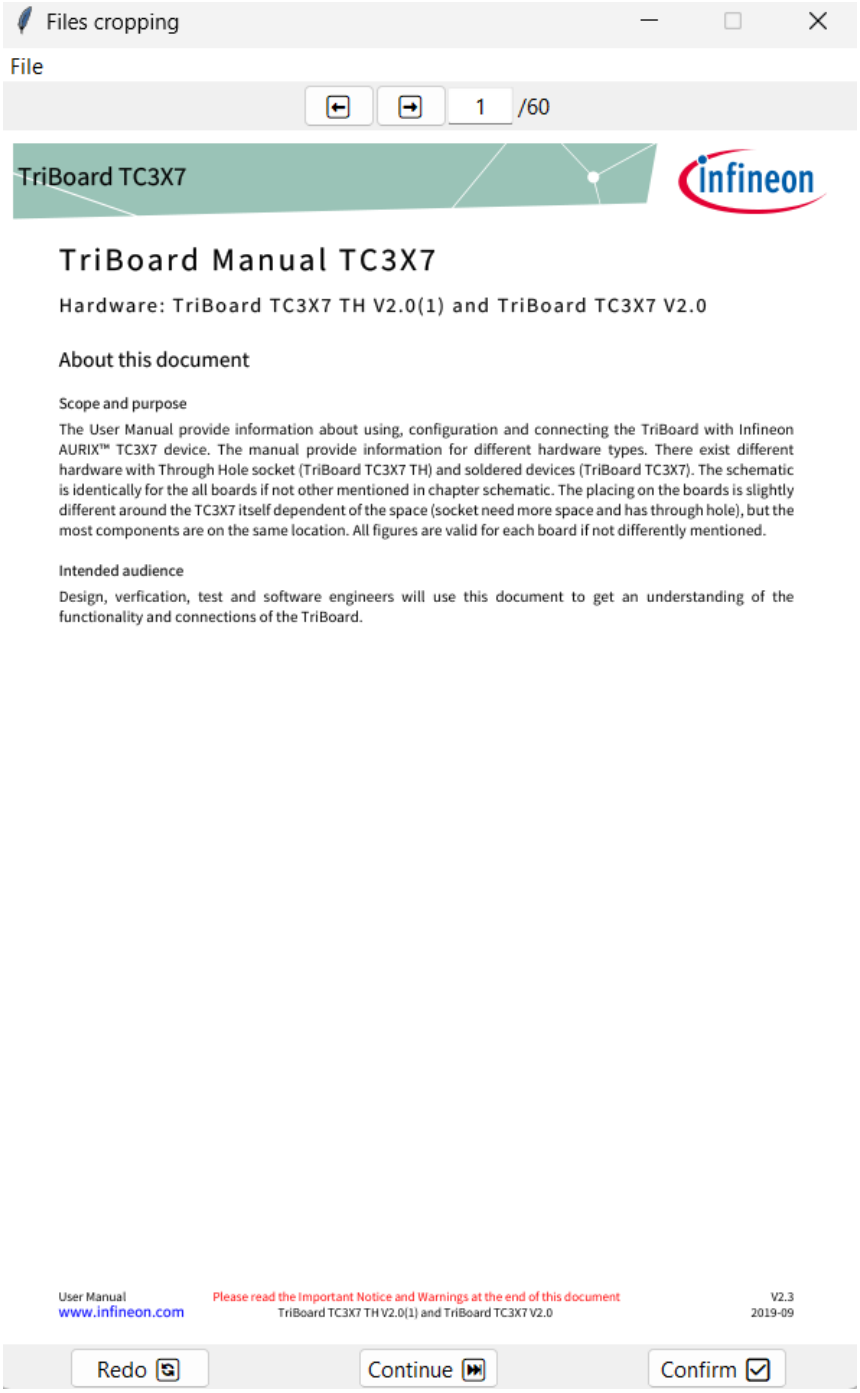

Figure 4.4. Initial GUI, PDF loaded

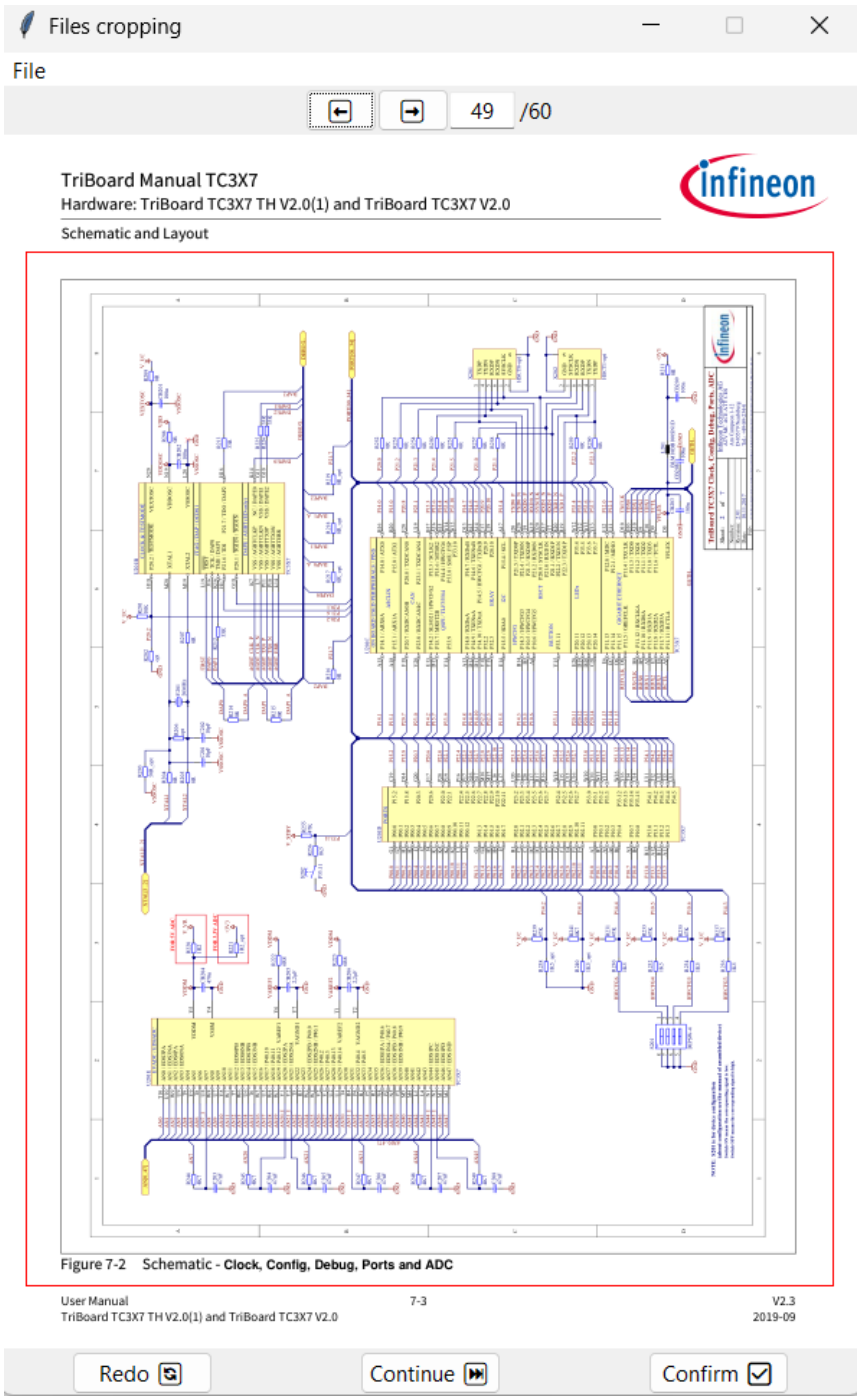

Figure 4.5. Initial GUI, red crop defined

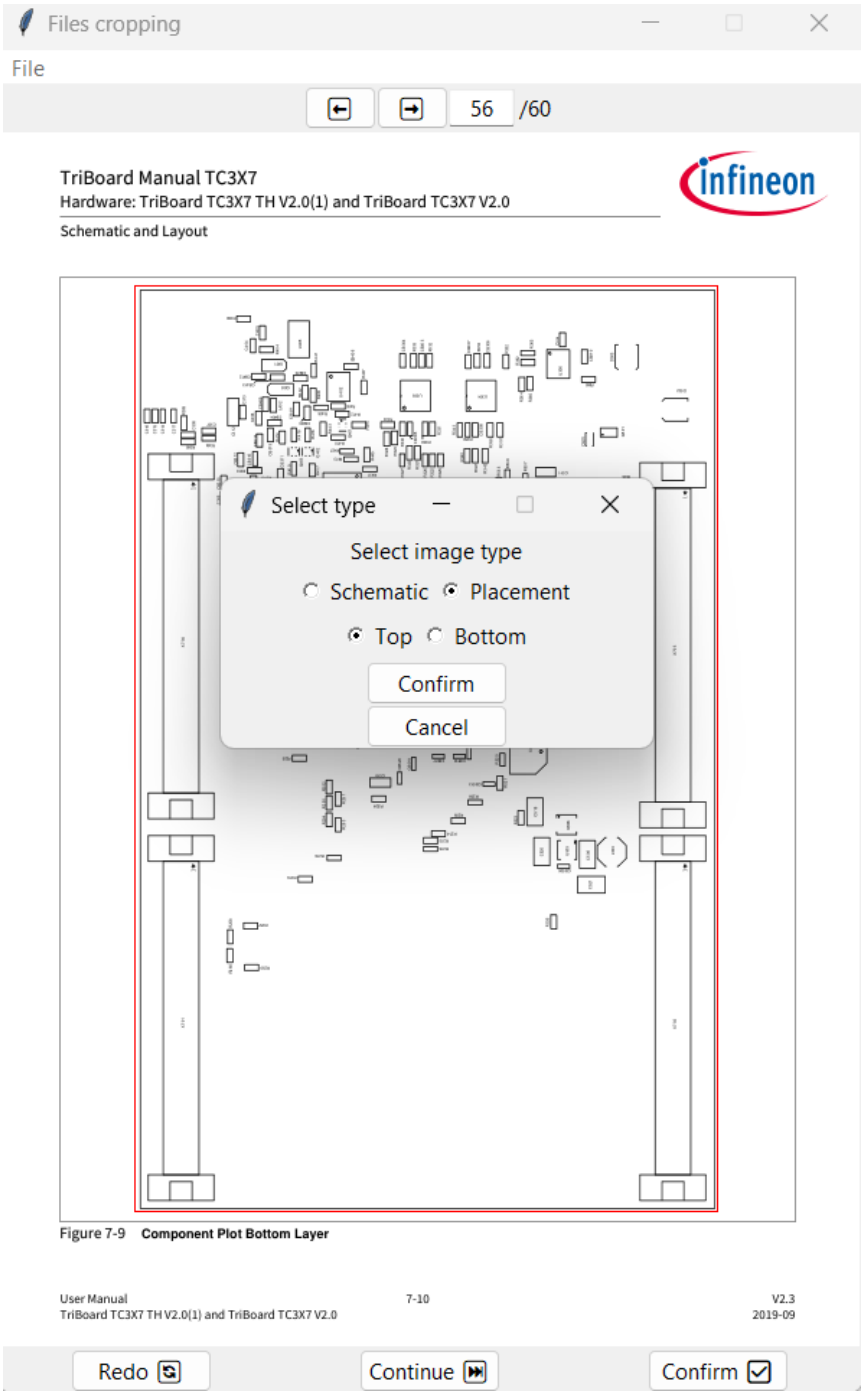

Figure 4.6. Initial GUI, crop confirmation

file and the crops defined by the user following the format:

Placement/Schematic $ Page # $ (x0, y0) - (x1, y1)

where # represents the page number, (x0, y0) represents the top-left corner of the crop and (x1, y1) represents the bottom-right one, all in PDF coordinates. In the case of a placement file, an additional field is added at the end that can be either *T* (top) or *B* (bottom).

4.4 Text extraction

The textual file produced at the end of the previous phase is the starting point for this one. Placement files are processed through pdfplumber, which can effectively extract single words from PDF files, while schematics are processed through PyMuPDF exploiting the word blocks' extraction. At the end of this phase, a textual file containing all the text extracted with coordinates is produced per each crop, following the format:

(Text, x0, y0, x0, y1)

where the coordinates refer to the resolution of the PDF file, which is annotated in the GUI file.

4.4.1 Board outline detection

Furthermore, the board outline is detected on the placement crops. It is performed here because it is needed for the next stage. The PDF page containing the placement crop is converted into a 1000 dpi image, and here is the first usage of the Canny edge detector applied inside the crop region. This is performed by the *find_outline* function, whose code is reported below.
The *findContours* function retrieves only the outermost contours (the mode is set to *RETR_EXTERNAL*), ignoring all the other nested edges that represent the components. At the end, it returns the found rectangle with the largest area, which coincides with the board outline.

```
1     def find_outline(image, tl_corner, br_corner):
2         roi = image[tl_corner[1]:br_corner[1], tl_corner[0]:br_corner[0]]
3         edges = cv2.Canny(roi, 50, 150)
4         contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
5                         cv2.CHAIN_APPROX_SIMPLE)
6         max_area = 0
7         max_rect = None
```

```
        for contour in contours:
            x, y, w, h = cv2.boundingRect(contour)
            area = h*w
            if area > max_area:
                max_area = area
                max_rect = ((x + tl_corner[0], y + tl_corner[1]),
                            (x + w + tl_corner[0], y + h + tl_corner[1]))
        return max_rect
```

Source code 1. The find_outline function

4.5 Placement components definition

The previous phase extracted the text labels from the placement files: now it is time to assign each component label to the closest shape. This phase is broken down into two steps: shapes assignment and manual correction GUI.

4.5.1 Shapes assignment

The shapes assignment step begins with converting the PDF page containing the placement crop of interest into a 1000 dpi image. During the processing of the first crop, an extra conversion into a 500 dpi image is performed. Every image processing elaboration is done on high-quality images to reach more accurate results; however, high-quality images can take some time to be displayed, so in this application, lower-quality images are used. This is not a concern: 500 dpi is more than enough to ensure good quality in a GUI, plus it helps speed up the image loading. The extra conversion performed is used to take note of the resolution of the 500 dpi version, as well as the 1000 dpi one.
The extracted text is covered using white rectangles: the coordinates are converted into the 1000 dpi image version to avoid any mistakes. It reduces the noise of the image, which is now ready to be processed through the Canny edge detector. The extracted edges are then filtered to retain only those falling inside the crop; these are then grouped into shapes stored in a dictionary. The code is reported below.

```
        # image has text covered with white rectangles
        gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        edges = cv2.Canny(gray, 50, 150)
        contours, _ = cv2.findContours(edges, cv2.RETR_TREE,
                        cv2.CHAIN_APPROX_SIMPLE)


```

```
        candidates = {}
        i = 0

        for contour in contours:
            epsilon = 0.005 * cv2.arcLength(contour, True)
            approx = cv2.approxPolyDP(contour, epsilon, True)
            line = []
            for vertex in approx:
                x, y = vertex.ravel()
                if y < (float(y1)/pdf_res[1]*image_res[0]) and y >
                        (float(y0)/pdf_res[1]*image_res[0]):
                    line.append((int(x), int(y)))
                if len(line) > 0:
                    candidates[i] = polygon.Shape(line)
                    i += 1
```

Source code 2. Snippet used to filter valid shapes.

The edge detector, this time, is set to analyse the entire image (the mode is set to RETR_TREE). The contours are approximated with vertices no more distant than *epsilon* from the original edge. Then, the components are stored in a namesake dictionary having the labels as keys and tuples containing coordinates and the centre of the labels as values. Before explaining the assignment algorithm, a table offering an overview of the two main data structures used is shown below.

Name	Key	Value format
candidates	incremental int	Shape
components	text label	(x0, y0, x1, y1, center)

Table 4.1. Data structures overview

The assignment algorithm iterates over all the candidates for each component: the candidate whose centroid is the nearest to the component centre will be assigned to that component at the end of the relative iteration. The code is reported below.

```
        for k, c in components.items():
            min = sys.maxsize
            min_index = -1
            for i, cand in candidates.items():
                if cand.centroid is None:
                    continue
                dist = polygon.distance_between_points(c[4],
```

```
                        cand.centroid)
                if dist < min:
                    min = dist
                    min_index = i
                elif dist == min and cand.area is not None and
                        (candidates[min_index].area is None or
                        cand.area > candidates[min_index].area):
                    min_index = i
            candidates[min_index].set_label(k)
```

Source code 3. Shapes assignment algorithm.

While populating the candidates dictionary, small shapes are filtered setting their centroid to None: this way, these are not considered during the process. At the end of the algorithm run, the components' labels are assigned to the closest shape with the largest area by setting the Shape label. The outcome of this step is a file containing the (potentially temporary) assignments of the centroids and an image displaying the labels and the selected shapes. It follows an example of placement given as input, the assignment image obtained and a comparison zoom.

As can be seen from the comparative zoom, some mistakes can be made during the assignment phase. For this reason, a graphical way to correct those mistakes needs to be provided, discussed in the next subsection.

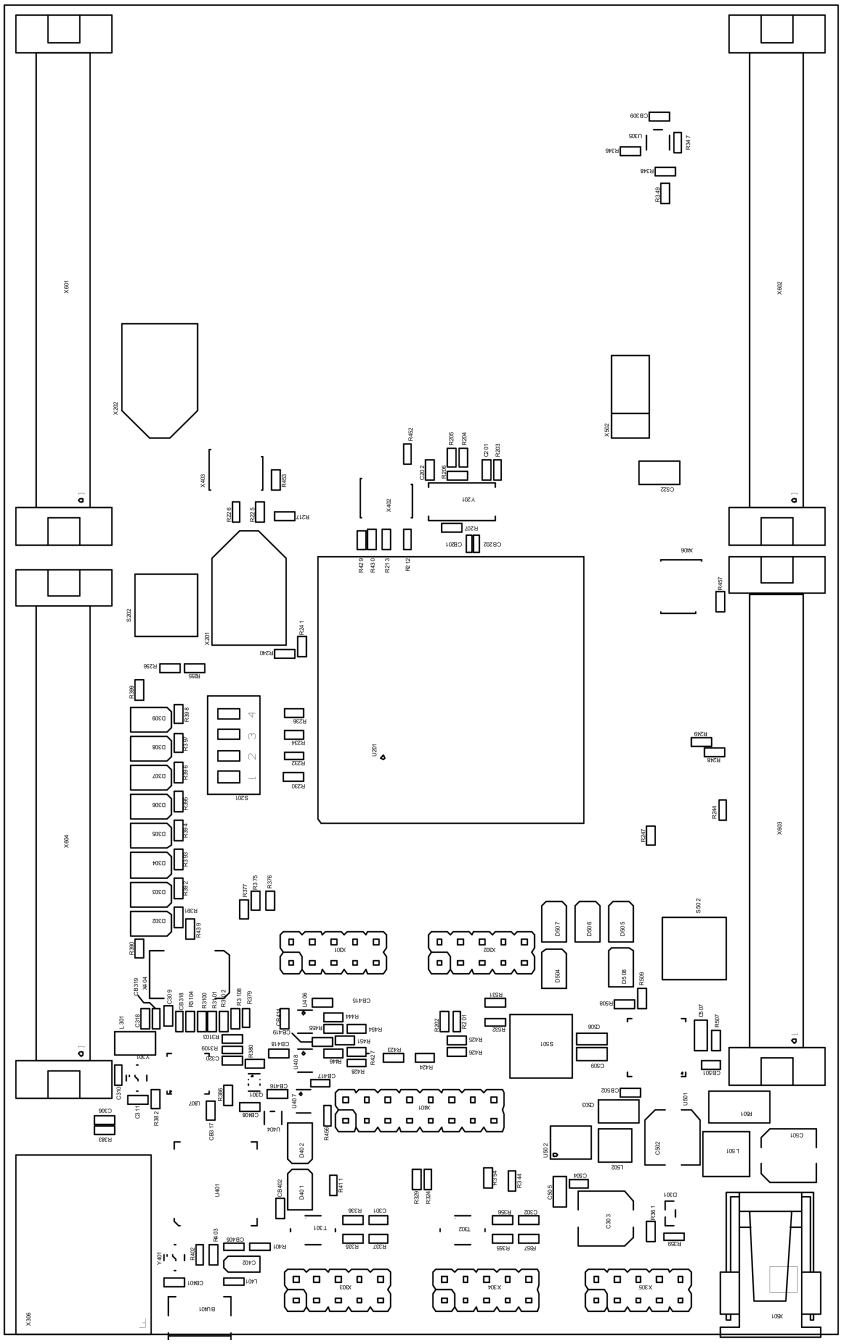

Figure 4.7. Placement file given as input

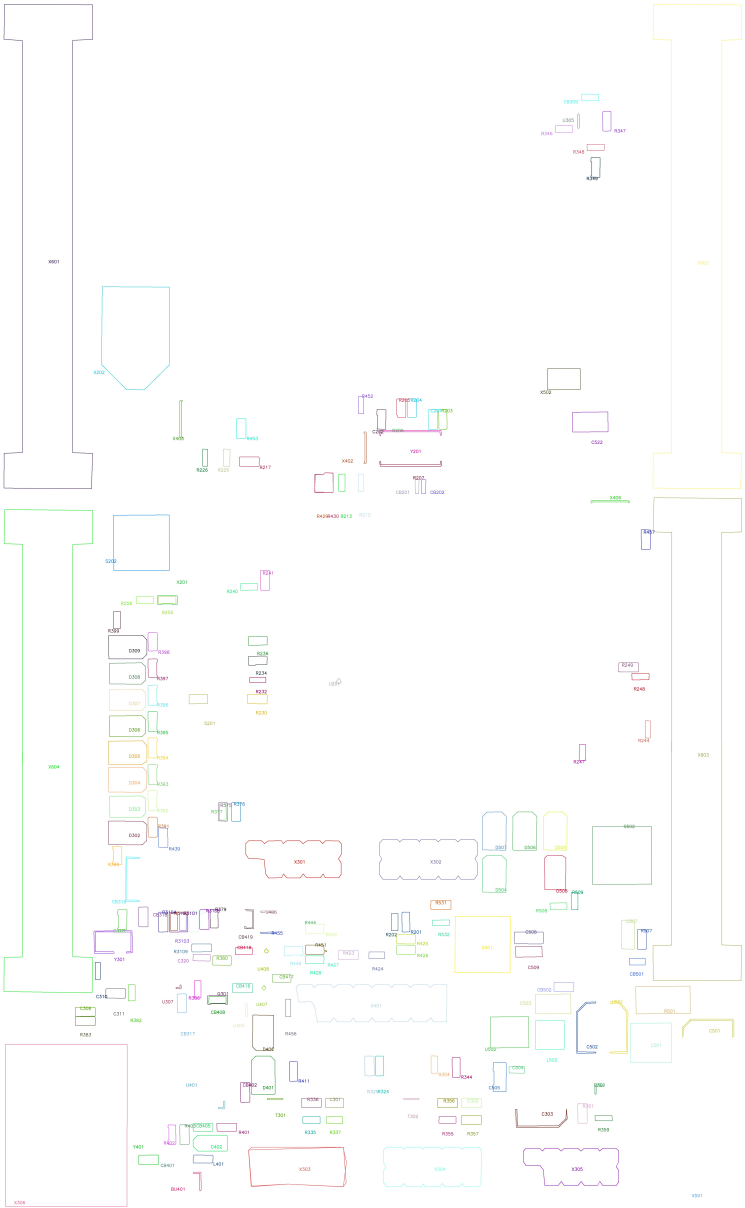

Figure 4.8. Assignment image obtained

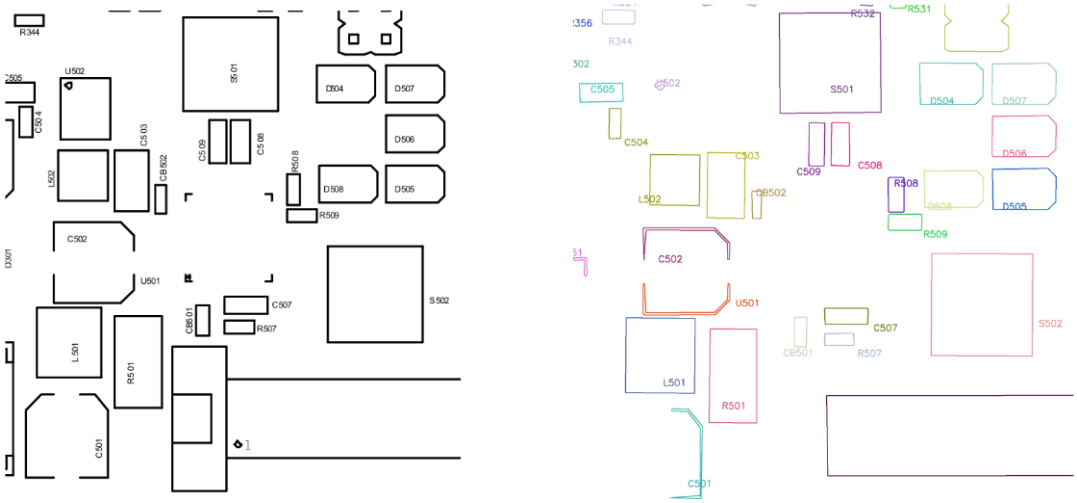

Figure 4.9. Comparative zoom of input and output

4.5.2 Manual correction GUI

During this phase, the user can correct mistakes and misses made during the shapes assignment stage. The GUI is launched once for each placement image. Each image has several levels of zoom that are pre-loaded to allow the user to zoom in and out. The GUI appears as shown in Fig. 4.10.
The labels' widgets are generated starting from the file generated during the assignment phase. It is possible to click on a label and drag it to the desired location, while the widget handles the coordinates internally. The labels are in the centre of a yellow circle to increase their visibility and more easily identify which components have been covered and which have not.
At the top of the GUI there is a bar hosting different buttons, starting from the left: zoom in, zoom out, add a new label, edit an existing label, delete a label, define pins for a component, hide labels and move to the next placement image.

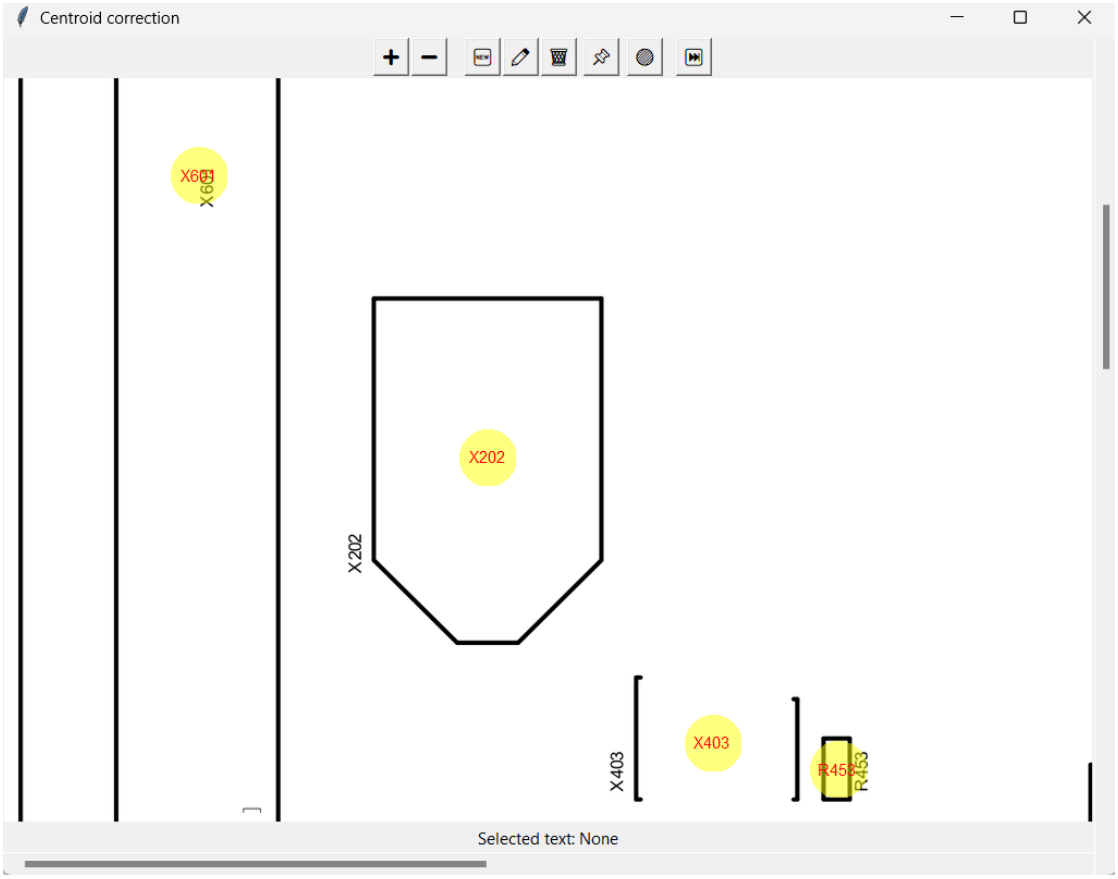

Figure 4.10. Centroid GUI, example

The add and edit buttons check the label does not exist before confirming to avoid duplicates. The define pins button makes a popup window appear after clicking a label. As shown in Fig. 4.11, it is required to indicate the total number of pins, how many pins per row and the enumerating order. The enumerating order offers two options: X or Y. Selecting X calculates the remaining pin positions following the incremental enumeration by rows, while Y does the same but following the order by columns.

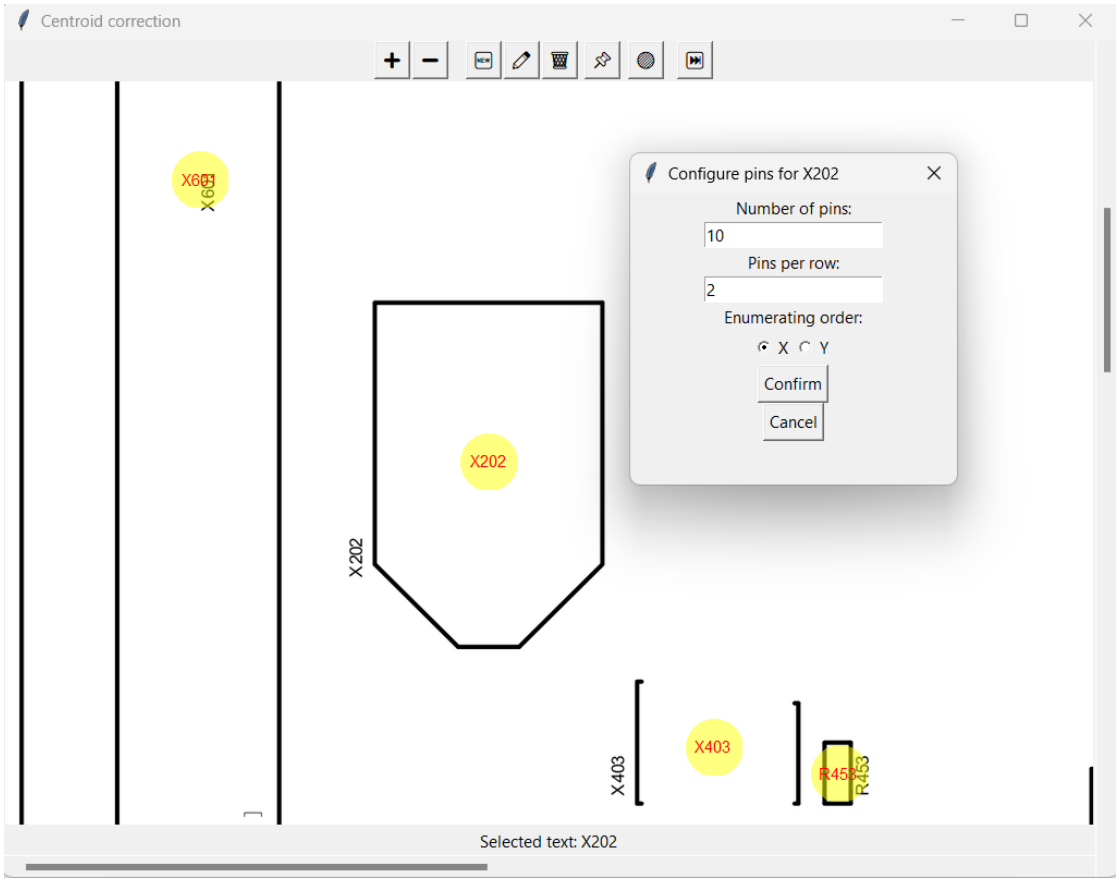

Figure 4.11. Centroid GUI, pins definition

Once confirmed, two green labels appear in the top-left corner: they represent the component's first and the last pin, following the format *label#number*. The user has to place them in the desired location, and then click on the confirm button at the bottom. An example is shown in Fig. 4.12.

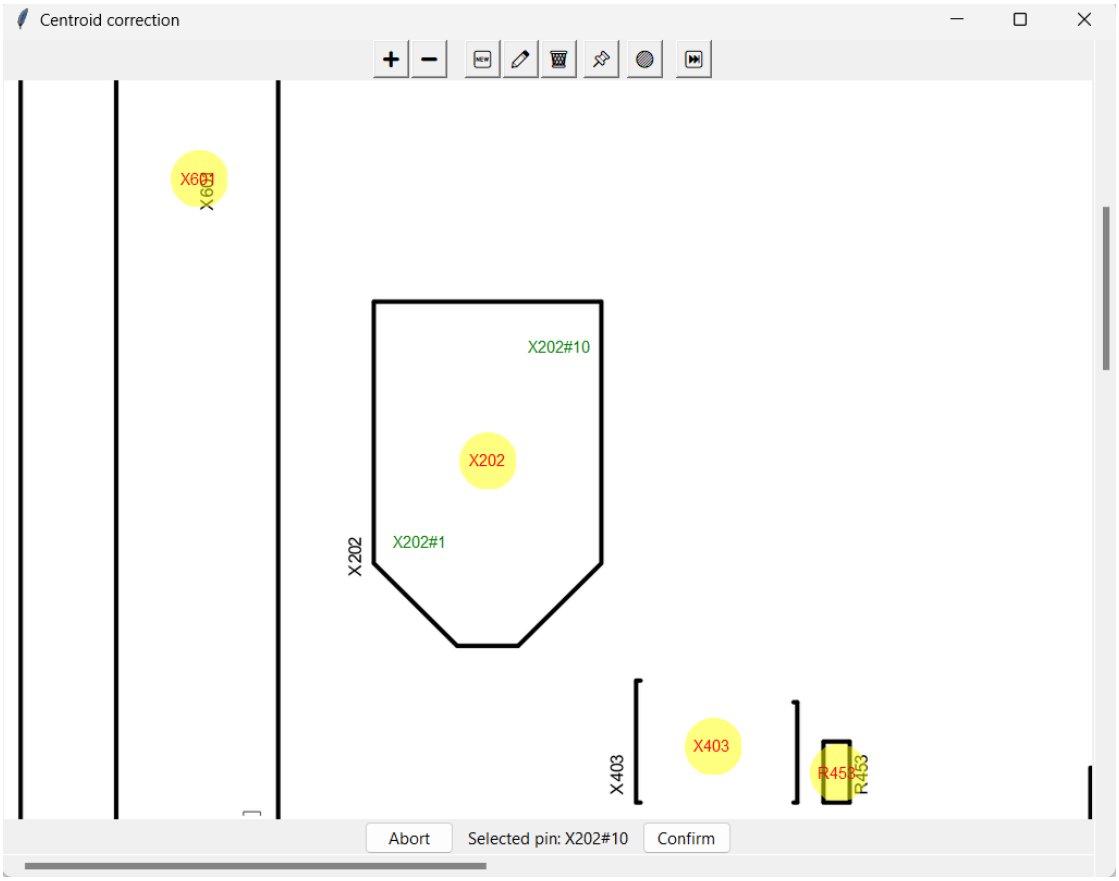

Figure 4.12. Centroid GUI, pins positioning

Once confirmed again, a simple code snippet calculates the remaining pins and creates a label for each, as shown in Fig. 4.13. The pins are now defined and cannot be moved anymore: to edit or delete them, is sufficient to repeat the process.

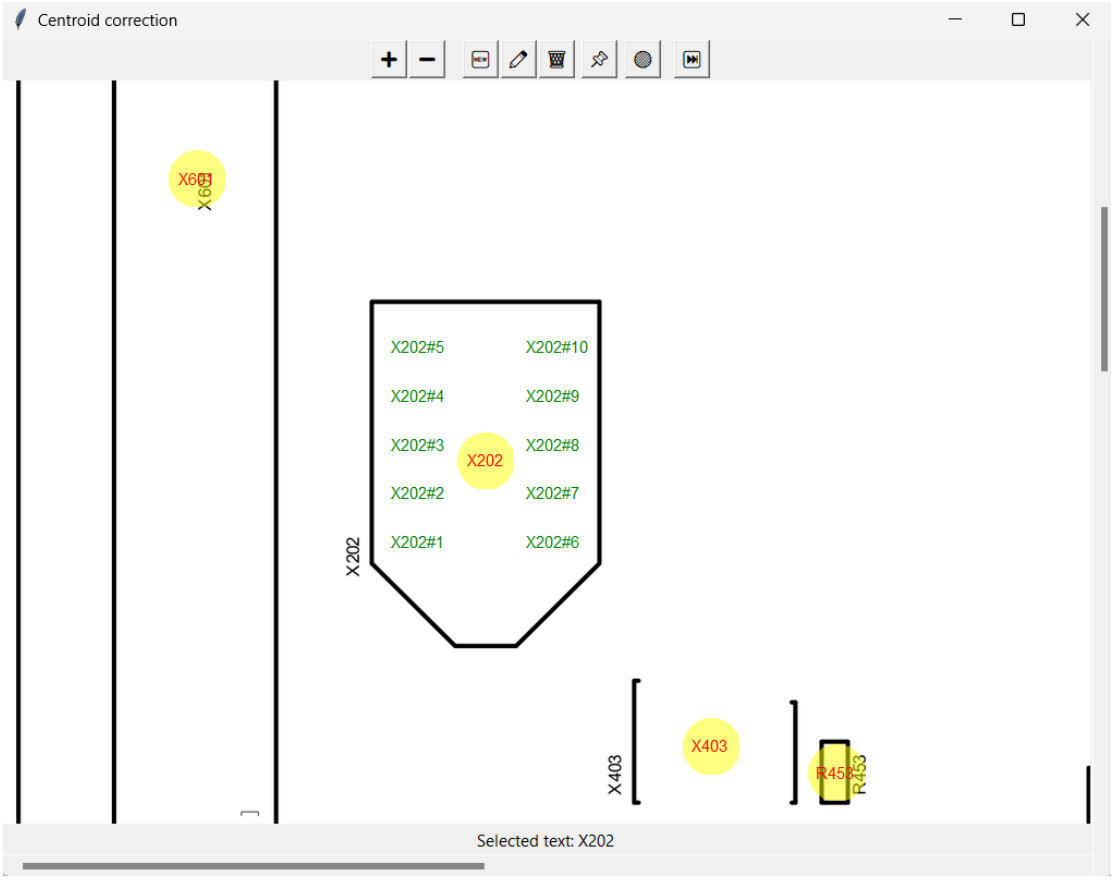

Figure 4.13. Centroid GUI, defined pins displaying

In areas with a high density of components, it may be convenient to display only the areas covered by the labels and to avoid overlapping labels. For this reason, a button has been provided to hide the labels and, by clicking it again, to display them again. An example is provided in Fig. 4.14.
Once all the corrections are made, the user can click the rightmost button to proceed with the next placement image. If all the placement images have been processed, the application moves on to the next stage.

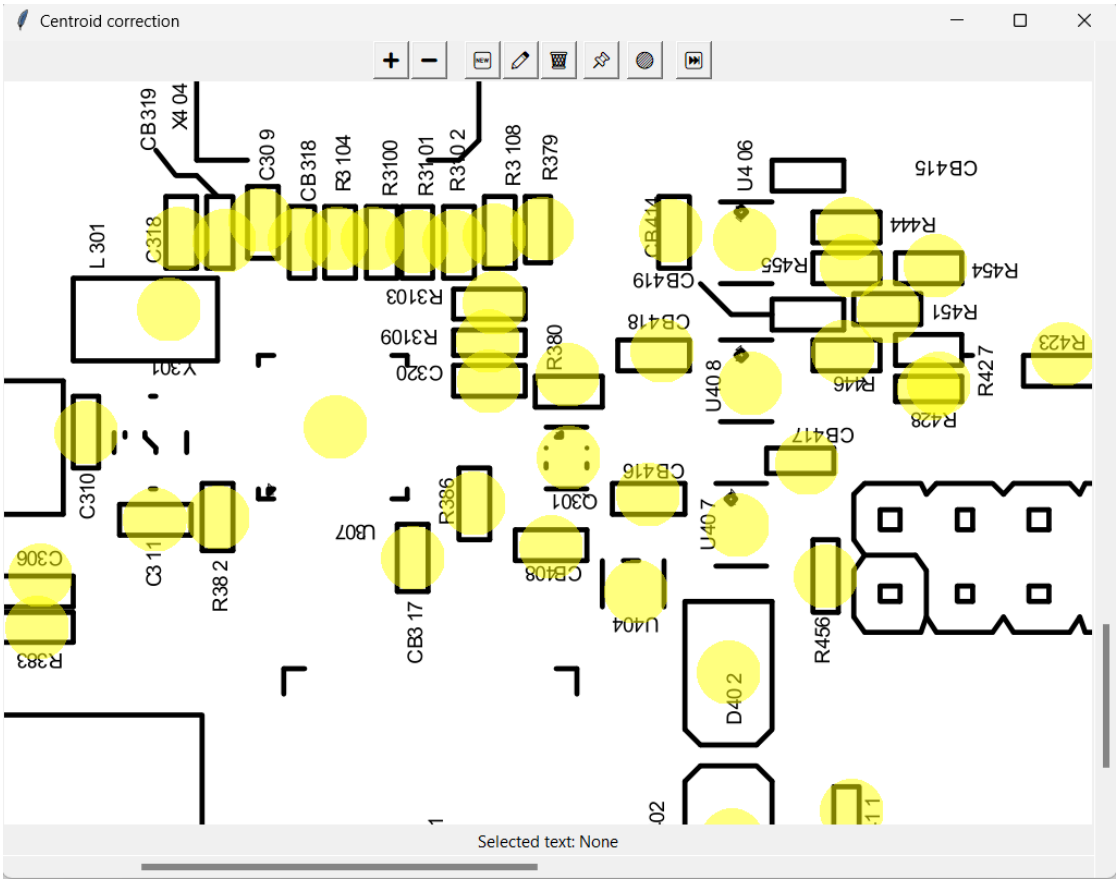

Figure 4.14. Centroid GUI, hidden labels

4.6 Matching phase

Several textual files are created during this phase where only shared components between placement files and schematics are kept. More specifically, every pair placement schematic gets a file. In the end, another version is produced for each placement file, where all the previous matching files related to it are concatenated. The files attend the CSV format for the AR testing application, where each row fills the following fields:

NAME,X-PL-CENTER,Y-PL-CENTER,X-SC-CENTER,Y-SC-CENTER,INDEX,TOP/BOTTOM

where: *NAME* is the component's label, *X-PL-CENTER* and *Y-PL-CENTER* are the coordinates of the label on the placement file calculated for the AR phase, *X-SC-CENTER* and *Y-SC-CENTER* the same as placement fields but for schematics, *INDEX* is the index of the page where it appears and *TOP/BOTTOM* can have as value T (top) or B (bottom).
More in detail, the coordinates are reported in a range (-0.5, 0.5), where (-0.5, -0.5) matches with the bottom-left corner. A special line is inserted immediately after the header filling only the first three fields with the label "OUTLINE", the outline width and height. The outline information flows from the outline extraction step to here.

4.6.1 Pins text extraction

Before moving to the final part of the application, a script is used to extract the pins' signals from the schematics. As will be shown in the next section, plenty of buttons will be overlaid over the schematics. At the current application state, the script exploits the components' colour as extra information. In this way, only text contained in regions of that colour will be extracted, hence signal names and pin numbers. As usual, a textual file is created as the outcome of this operation.

4.7 Highlighting GUI

Finally, a GUI is set up to display components on both schematic and placement files: the user can click a component, represented by a button, on a schematic, and the corresponding label will be displayed on the placement file where the component is placed. Before launching it, an image pre-loading phase is needed, similar to what is performed for the manual correction GUI. Once finished, an empty window is displayed as shown in Fig. 4.15.

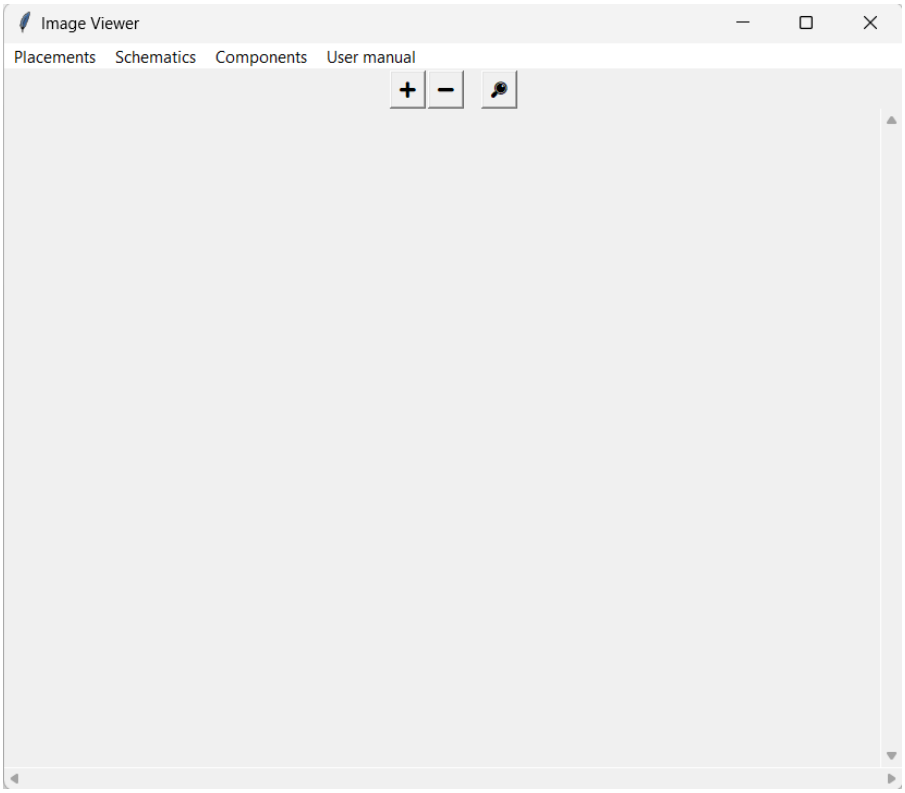

Figure 4.15. Highlighting GUI, empty

At the top, a menu hosts four sections: placements, schematics, components and user manual. The user manual section is just a shortcut to display the manual using a visualiser installed on the device. Once clicked, the placements and schematics show a cascade with all the placements and schematics analysed by the application. A button bar is shown under the menu, hosting the zoom-in, zoom-out and magnifier buttons, the last one used to look for a component inside the current image.
The user can select a schematic to display: it is shown in the window with all the buttons defined in the previous stage. An example is shown in Fig. 4.16. An extra button now appeared, allowing the user to unclick all the buttons at once.

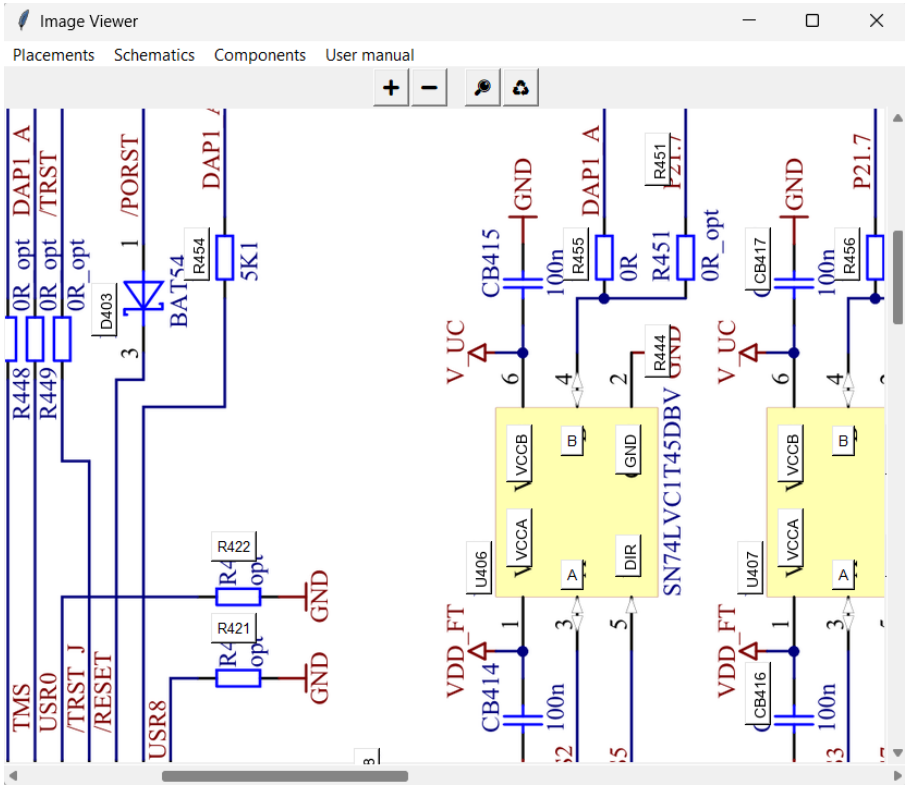

Figure 4.16. Highlighting GUI, schematic displayed

The user can click the buttons, which become yellow, and also can switch to another schematic image and do the same. An example of buttons clicked is shown in Fig. 4.17.

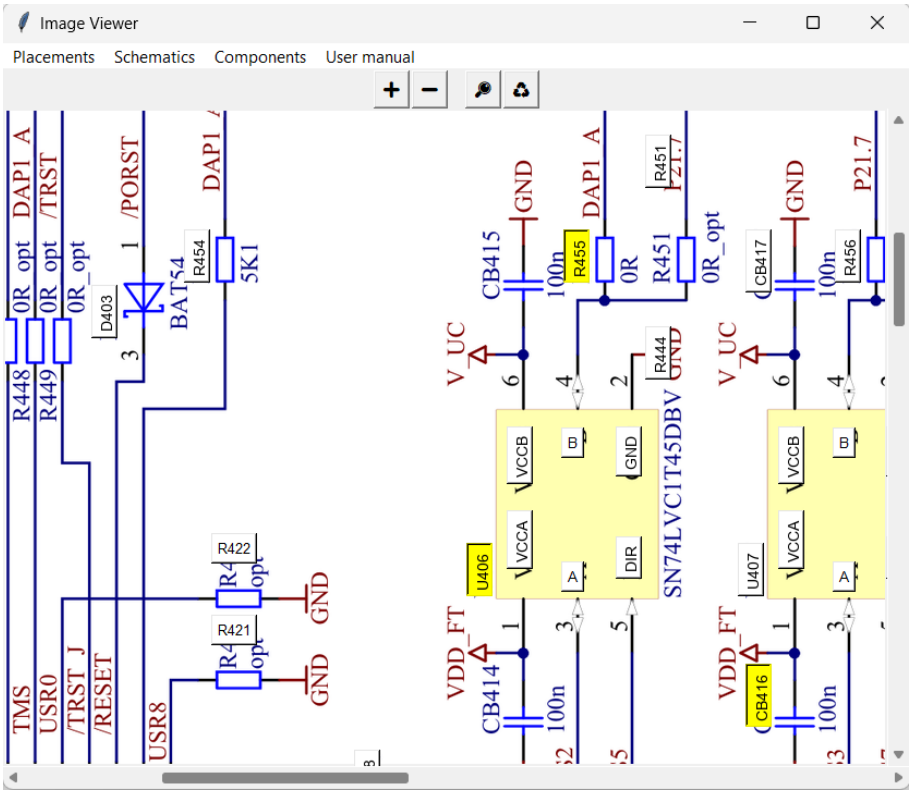

Figure 4.17. Highlighting GUI, buttons clicked

When a placement is displayed, all the selected components via buttons will show a label, equal to the one in the manual correction GUI. Plus, instead of having the unclick button, here is a button that hides all the labels, similar to the one already seen in the manual correction GUI. Following the previous example, in Fig. 4.18 the labels related to the clicked buttons are shown.

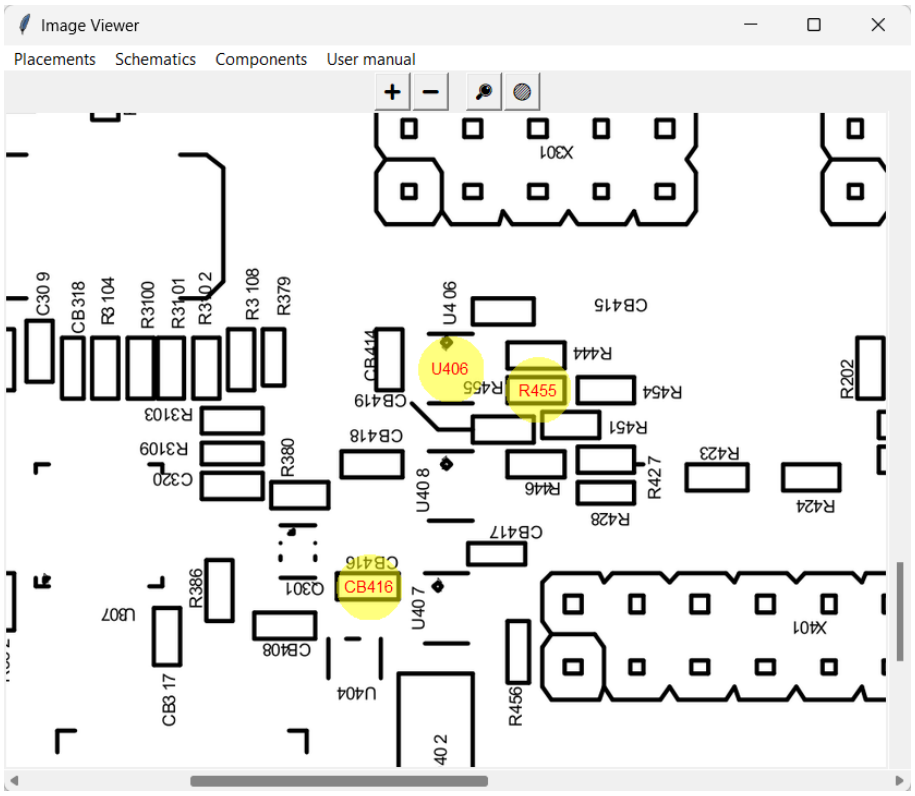

Figure 4.18. Highlighting GUI, placement highlighting

The components' menu section shows the list of displayed components in the current image, grouped by the initial letters. The user can click on a label and the image portion containing that component will be shown. Clicking on the magnifier button, a research window pops up. It works as the components section, also allowing the user to type and dynamically filter the labels based on what the user wrote. The research window is shown in Fig. 4.19.

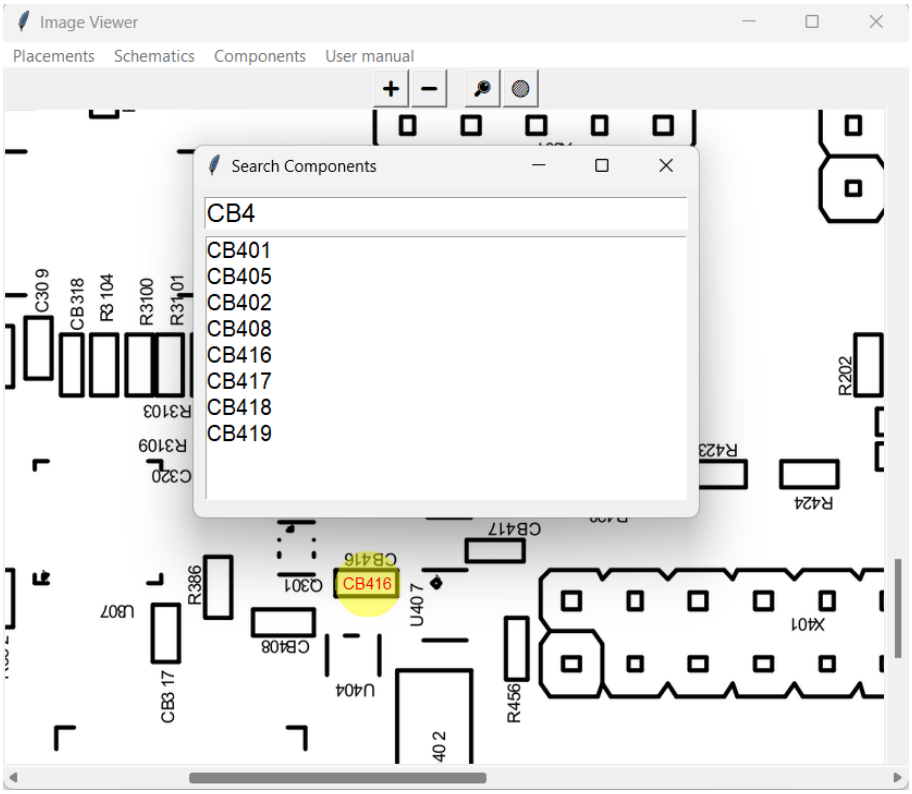

Figure 4.19. Highlighting GUI, research window

Chapter 5

Conclusions

This work is intended as a first step towards speeding up the PCB testing phase which currently relies on having the board manual always visible and human attention. There is still a long way to go, but a solid foundation has been built with this work. Of course, some choices had to be taken in order to achieve acceptable results, as shown in the shapes assignment section.

A combination of text extraction, image processing and GUI has been proposed to trace a path, still asking for little human intervention to adjust committed mistakes in the process. A way should be found to avoid relying on external information to take certain steps, such as pins' labels extraction before entering the highlighting GUI.

Future work may regard the connections graph extraction (briefly discussed in Appendix B) or automating some phases of the up-to-now shown process. For instance, one could think of embedding a neural network trained to distinguish between placement and schematic files when the user is cropping, avoiding the user specifying the file typology. Another idea could be to train a convolutional neural network to recognise the most diverse shapes: this could be beneficial especially to recognise open shapes. Another addition could be embedding Optical Character Recognition, in case text is present in the manual as lines or images.

Appendix A

Canny edge detector

The Canny edge detector Canny [1986] is an edge detection operator based on a multi-stage algorithm that detects a wide range of edges in images. It was developed by John F. Canny in 1986.
Canny edge detection is a technique to extract useful structural information from different vision objects and reduce the amount of data to be processed. The general criteria to achieve high-level performance by edge detectors are:

1. low error rate in edge detection, which means that the detection should not miss any edge present in the image

2. the edge points are well localised, which translates to minimising the distance between the edge points marked by the detector and the true centre of the edge

3. multiple responses to a single edge should be avoided, as well as fake edges created by image noise.

To satisfy these criteria, Canny used the calculus of variations, that is finding the function that maximises a given functional. In this problem, the functional takes into account the three criteria defined above, and the optimal function can easily be approximated by a Gaussian.
The algorithm goes through several steps:

1. **Noise reduction**
   A Gaussian filter is applied to reduce the image noise level, which may affect the detector's performance, such that the image is smoothed. Finding the proper kernel size for the filter is crucial because the noise sensitivity reduces, but the localization error increases as the size increases. A Gaussian filter

example can be:

$$G = \frac{1}{273}\begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix}$$

2. **Intensity gradient calculation**
   The intensity gradients represent the changes in pixel intensity values: large gradients refer to strong edges. The intensity gradients are calculated through derivative filters for the horizontal (G_x) and vertical (G_y) directions. These two values are used to calculate the gradient magnitude (G) and direction (θ):

$$G = \sqrt{G_x^2 + G_y^2}$$
$$\theta = \text{atan2}(G_y, G_x)$$

   where G is calculated through the hypot function and θ through the 2-argument arctangent.

3. **Non-maximum suppression**
   It is used as an edge-thinning technique to find the location with the sharpest change of intensity value. For each pixel, the edge strength of the current pixel is compared to the edge strength of the pixels in the positive and negative directions: if it has the largest intensity value, it will be preserved, otherwise discarded.

4. **Double thresholding**
   After the non-maximum suppression, some spurious edge pixels may be left due to noise and colour variation. A double thresholding technique is then used to further refine the result. A low and a high threshold are set: pixels with a gradient magnitude above the high threshold are classified as strong edge pixels, while the ones below the low threshold are discarded. The pixels in between are marked as weak edge pixels.

5. **Edge tracking by hysteresis**
   The strong edge pixels are confirmed to be part of the final image; the same cannot be said for the weak edge ones. It should be determined if a weak edge pixel comes from a strong edge or noise or colour variation, and keep weak edge pixels only if they belong to the first category. The idea is that weak edge pixels are connected to at least one strong edge pixel: the connections can be retrieved through connected-component analysis. The weak edge pixels connected to at least one strong edge pixel are kept, which can cause neighbouring weak edge ones to be kept in turn.

Appendix B

Connections graph extraction

This appendix represents a starting point for a possible future extension of this work. The aim is to obtain the connections between components from the schematics using connected-component analysis. The approach sketched out was to input different component symbols, as well as the colour of those containing pins. This way, an image including only the components of the schematic can be obtained through convolution and colour mask. Plus, one can extract the lines of connections based on information such as colour, and thus obtain a second image. An example is shown in the next pages: in Fig. 5.1 a schematic example is shown, in Fig. 5.2 the extracted components are shown and eventually in Fig. 5.3 the layout of the connections is shown.

The main problem is to handle the intersection between two connections: unless some specific symbol is reported (i.e. a full circle), the connections are separated. The idea is, again, to perform a convolution using the intersection template, remove them from the image, and store their coordinates. At this point, the connected-component analysis is used to identify unique connections: the result based on the reported example is shown in Fig. 5.4, where each connection is associated with a random colour. Near an intersection, four segments need to be paired top-bottom and right-left: the pairs should be considered as one connected component (or blob, to avoid ambiguities). Iterating the process for each intersection yields the actual connections, as shown in Fig. 5.5.

A first connections graph can be extracted using the blobs: for each component, the location is known, and a map containing the label as key and the near blobs as values can be built. Components that are connected share at least one blob.

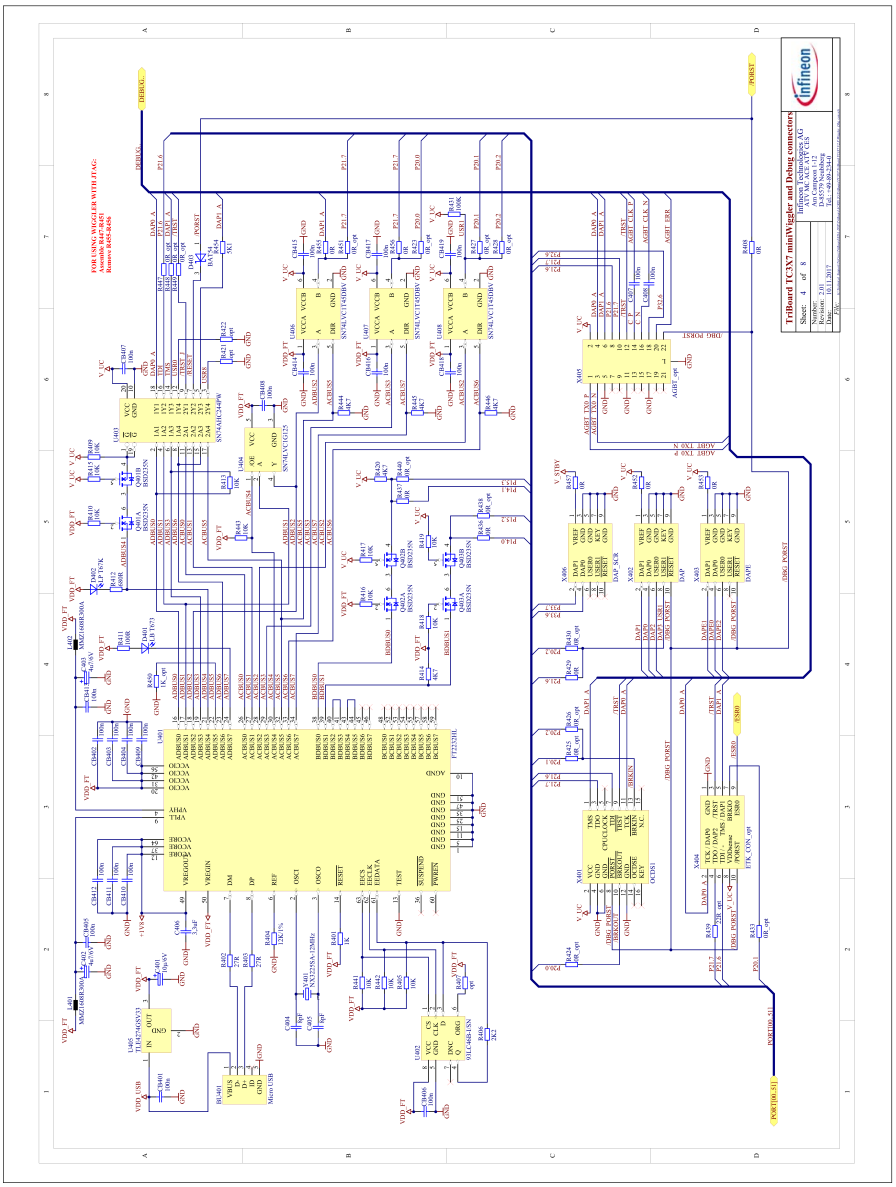

Figure 7-4 Schematic - miniWiggler JDS and Debug connectors

Figure 5.1. Connections extraction, schematic example

Figure 5.2. Connections extraction, extracted components

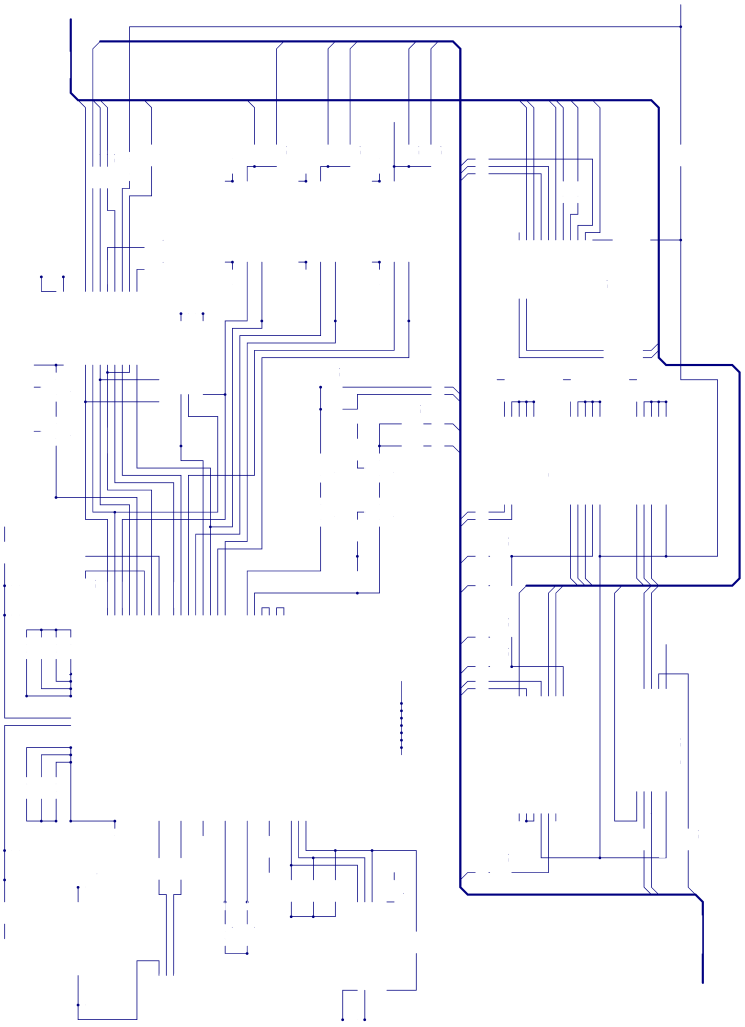

Figure 5.3. Connections extraction, extracted connections

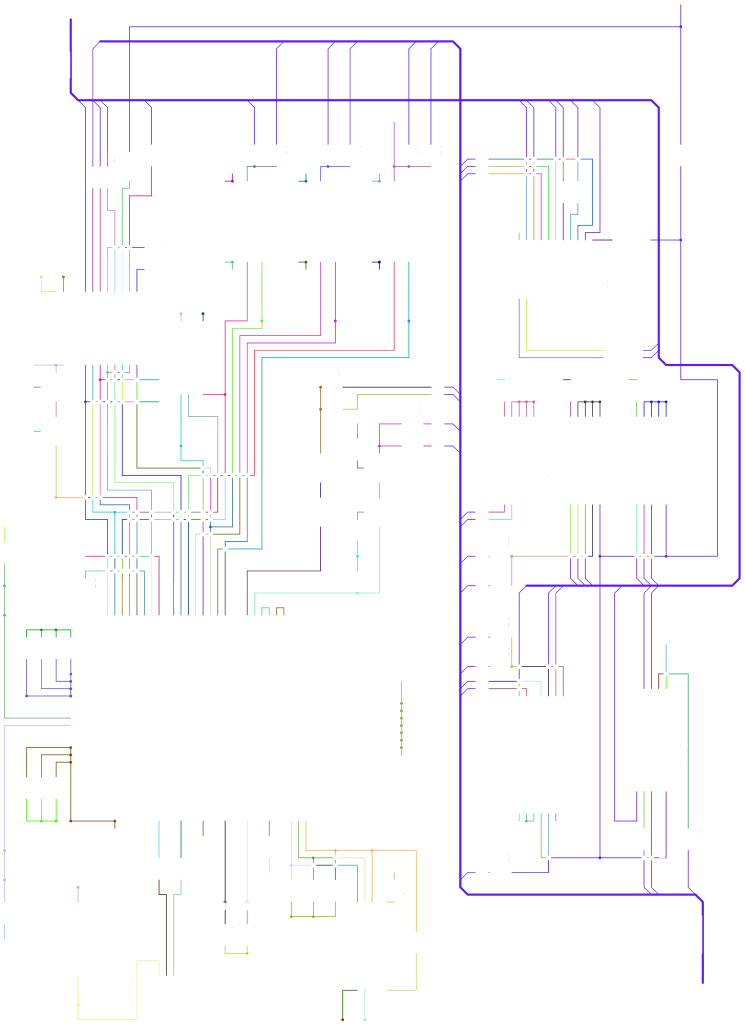

Figure 5.4. Connections extraction, first-stage connected components

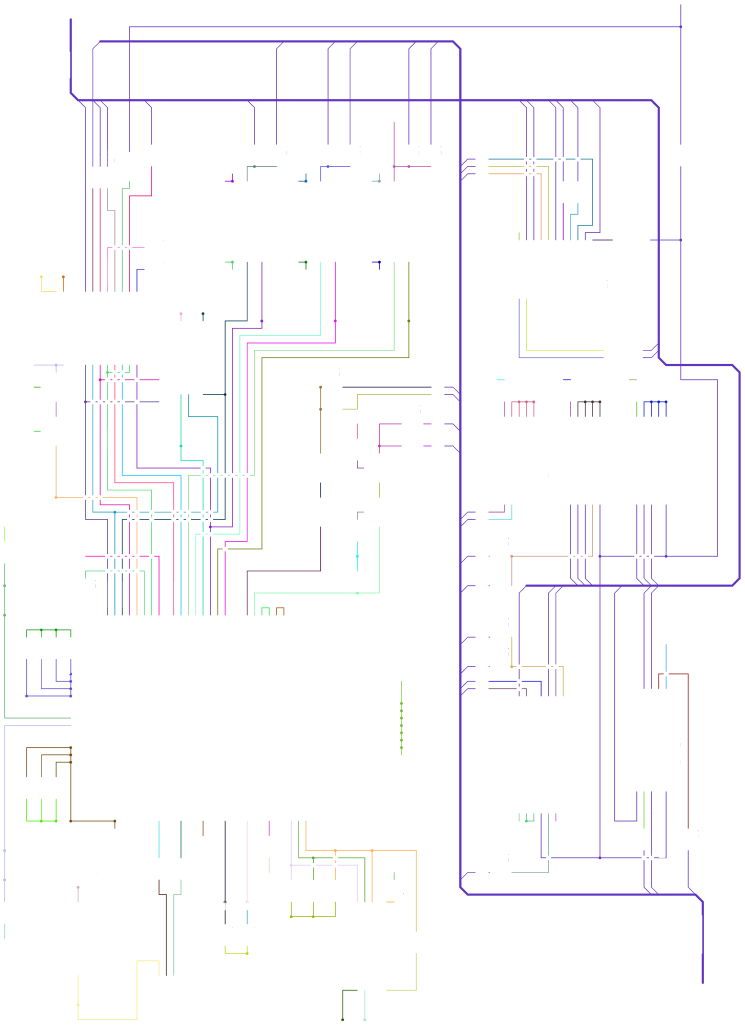

Figure 5.5. Connections extraction, final result

Bibliography

Artifex. Pymupdf documentation, 2024. URL https://pymupdf.readthedocs.io/en/latest/.

Belval. pdf2image. https://github.com/Belval/pdf2image, 2017.

G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986. doi: 10.1109/TPAMI.1986.4767851.

Alex Clark. Pillow (pil fork) documentation, 2015. URL https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf.

freedesktop.org. Poppler documentation, 2005. URL https://poppler.freedesktop.org/.

jsvine. pdfplumber. https://github.com/jsvine/pdfplumber?tab=readme-ov-file, 2015.

Wojciech Kowalke, Krzysztof Górecki, Przemysław Ptak, Liam Cadigan, Brian Borucki, Nick Warren, and Mario Ancona. A new system supporting the diagnostics of electronic modules based on an augmented reality solution. *Electronics*, 13(2):335, 2024.

Fredrik Lundh. An introduction to tkinter. *URL: www. pythonware. com/library/tkinter/introduction/index. htm*, 1999.

Marco Ojer, Hugo Alvarez, Ismael Serrano, Fátima A Saiz, Iñigo Barandiaran, Daniel Aguinaga, Leire Querejeta, and David Alejandro. Projection-based augmented reality assistance for manual electronic component assembly processes. *Applied Sciences*, 10(3):796, 2020.