UNIVERSITY OF TARTU

Institute of Computer Science

Computer Science Curriculum

Kalmer Kaurson

Graphical User Interface for Constellations Image Generator

Bachelor's Thesis (9 ECTS)

Supervisor: Tarun Khajuria

Tartu 2022

Graphical User Interface for Constellations Image Generator

Abstract:

This thesis describes the design and development of a graphical user interface

The aim of the thesis is to create an interface for an application where the user can insert an image and choose from which region of the image will be used to generate the constellation image. When an area is selected, the program generates a border from the object in the image. With the web application, it is possible to remove redundant lines from the generated images and choose how closely the points are both in the constellation and in the background.

The topic is important because the bachelor's thesis, UI application and source code will be made publicly available so that scientists can generate data for human and machine learning experiments.

Keywords:

Graphical user interface, Tkinter, web application

CERCS: P170 Computer science, numerical analysis, systems, control

Graafiline kasutajaliides tähtkujude pildigeneraatori jaoks

Lühikokkuvõte:

Lõputöö eesmärk on luua kasutajaliides rakendusele, kus kasutaja saab sisestada pildi ning valida, missugusest piirkonnast genereeritakse tähtkuju. Kui piirkond on valitud genereerib programm piirjoone pildil olevast objektist. Veebiliidesega on võimalik genereeritud piltidelt eemaldada üleliigseid jooni ning valida, kui tihedalt asuvad tähed üksteisest nii tähtkujus kui ka tagataustal.

Teema on oluline, sest bakalaureusetöö lõpuks läheb kasutajaliides avalikuks, et teadlased saaksid luua andmeid inim- ja masinõppe katseteks.

Võtmesõnad:

Kasutajaliides, Tkinter, veebiliides

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Table of Contents

1. Introduction

Machine learning can detect objects in images, but background noise can make it difficult for the program to identify objects. The program created by Computational Neuroscience group, University of Tartu is able to create a constellation from a given image as an input (see Figure 4), first creating a border from the object in the image and then turning the border into a dotted outline. Point noise is added to the background of the object's boundary, resulting in a constellation. Machine learning algorithms can learn such constellations and, as a result, identify objects inside the noise that would not otherwise be possible. During the generation of constellation images, in some cases the program cannot automatically detect the correct object from the image, or a situation arises where the program generates redundant lines when drawing a border. In this case, it is necessary to adjust the outlines by adding new lines or removing unnecessary lines. In addition to adjusting the lines, it could be possible to change the program parameters without modifying the program manually.

The aim of this bachelor's thesis is to create a graphical user interface for a constellation program, which can be used to modify the images created by the program, which would otherwise have to be done with a separate application. With the initial program it is possible to create many constellation images using specific program parameters but results may not be as the user wishes them to be. In these cases, the user has to either change program parameters by changing the code itself or have to use image editing software to remove unnecessary features from the image. In addition to image processing, the application must allow the user to change the spacing and number of the dots in the image.

The first part of the bachelor's thesis describes the function of the software and the requirements that must be met when developing a desktop application. The second part of the work describes the tools and software packages that were used during the development. The third part describes and analyzes the completed graphical user interface. The fourth part describes the challenges that occurred during this thesis and future possibilities regarding the interface.

The source code of the graphical user interface can be found at https://github.com/kalmerkaurson/Constellations_GUI.

2. Terms and definitions

OpenCV - Open Source Computer Vision is a library of Python bindings designed to solve computer vision problems.

Canny Edge Detector - OpenCV function for detecting the edges in an image.

ROI - A region of interest (ROI) is a portion of an image that you want to filter or operate on in some way.

CLIJ - ImageJ2/Fiji plugin for GPU-accelerated image processing.

CLIJPY - Bridge between CLIJ and Python via pyimagej.

COCO - Large-scale object detection, segmentation, and captioning dataset.

Mask-RCNN - Region-Based Convolutional Neural Network, it is a type of machine learning model that is used for computer vision tasks, specifically for object detection.

3. Similar Projects

This thesis project is quite unique when it comes to comparing it to similar projects. Projects that do image processing are similar enough to compare with the thesis project. This chapter focuses on three programs – OpenCV with Tkinter[1] (2016), ImagePy[2] (2021) and CLIJPY[3] (2020).

3.1 OpenCV with Tkinter

Adrian Rosebrock had created a program with Tkinter that converts an image into an outlined version of the image and the user interface displays both images after an image had been selected. (Figure 1)

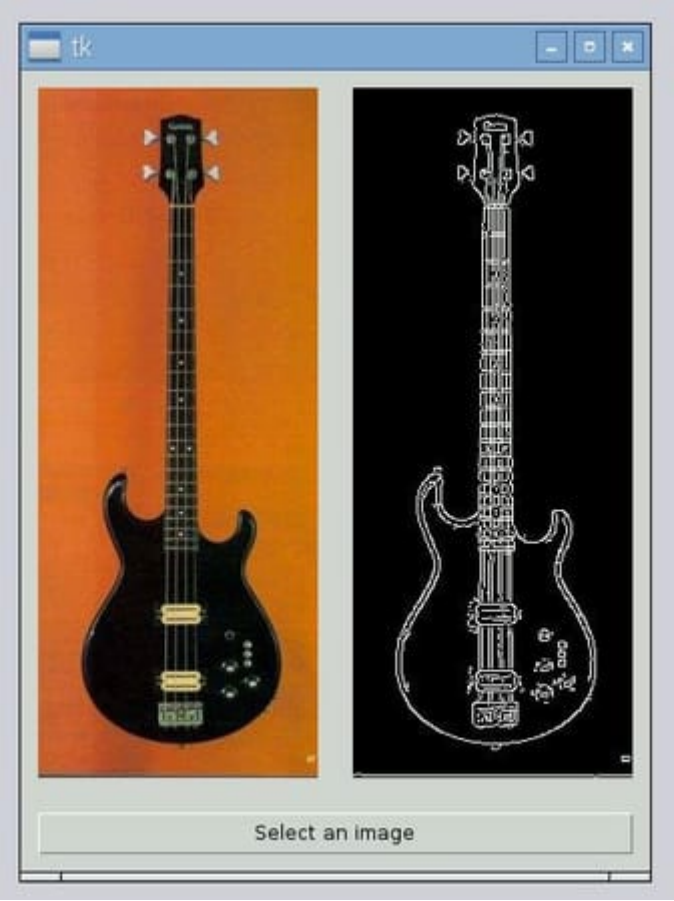

Figure 1: Loading an image, computing edges using the Canny edge detector, and then displaying the result using Tkinter and OpenCV.

[1] https://pyimagesearch.com/2016/05/23/opencv-with-tkinter/
[2] https://github.com/Image-Py/imagepy

[3] https://clij.github.io/clijpy/

The program made by Adrian Rosebrock has very limited functionality but does the same function of creating an outline of the selected image. This thesis project is an upgrade of Adrian's project because it allows the user to modify the edges by changing the parameters of the Canny Edge Detector or use an eraser tool to remove unnecessary lines.

3.2 ImagePy

ImagePy (Figure 2) is an open-source image processing framework written in Python. The program can read and save a variety of image data formats, supports ROI settings, drawing, measurement and other mouse operations. It can perform image filtering, morphological operations and other routine operations, and image segmentation. It is able to perform data analysis, filtering and statistical analysis.

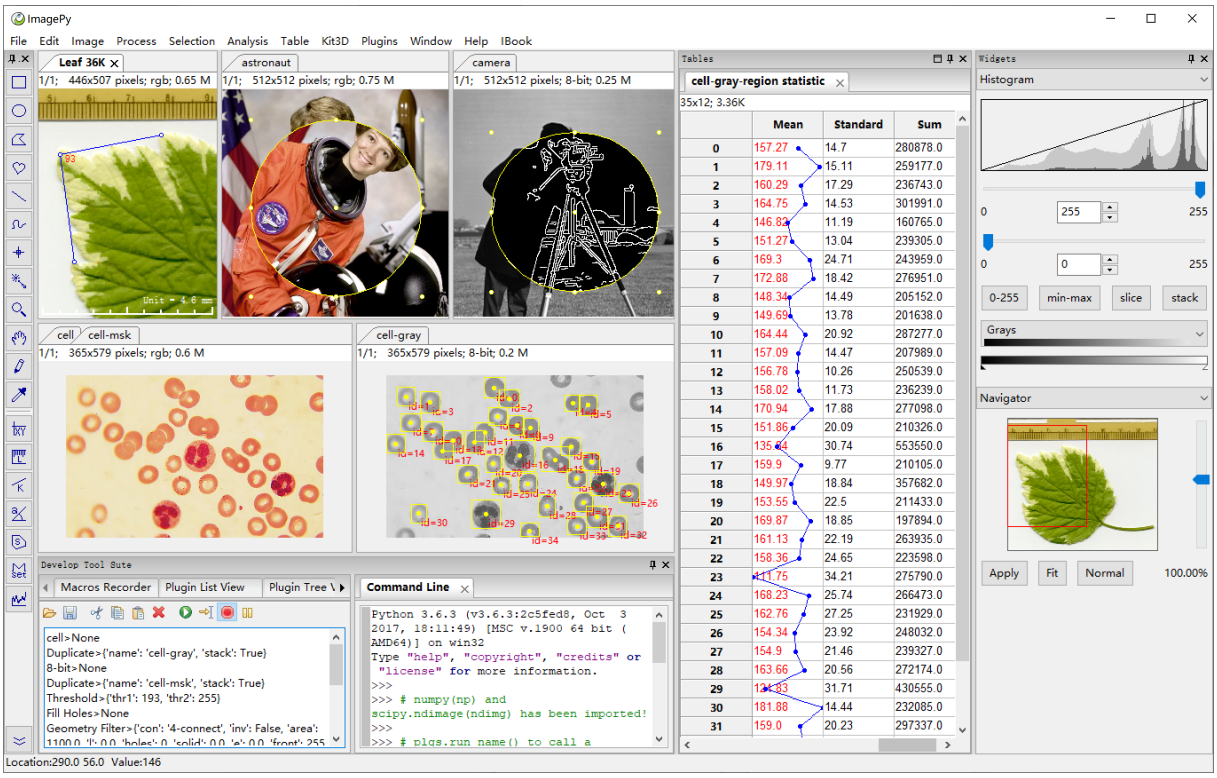

Figure 2. *ImagePy*. Overview, mouse measurement, geometric transformation, filtering, segmentation, counting, etc.

In comparison with this thesis project the *ImagePy* program can do image segmentations and edge detection but it can not create a dotted pattern out of the edge image. *ImagePy* has its own benefits and is mainly oriented for data analysis but for this project we require a simpler approach that is easier to understand.

3.3 CLIJPY

CLIJPY (Figure 3) is a GPU-accelerated image processing in python using CLIJ and pyimagej. Similarities with the thesis project are that *CLIJPY* was made using Python, *CLIJPY* is oriented for image processing, mainly for detecting dots.

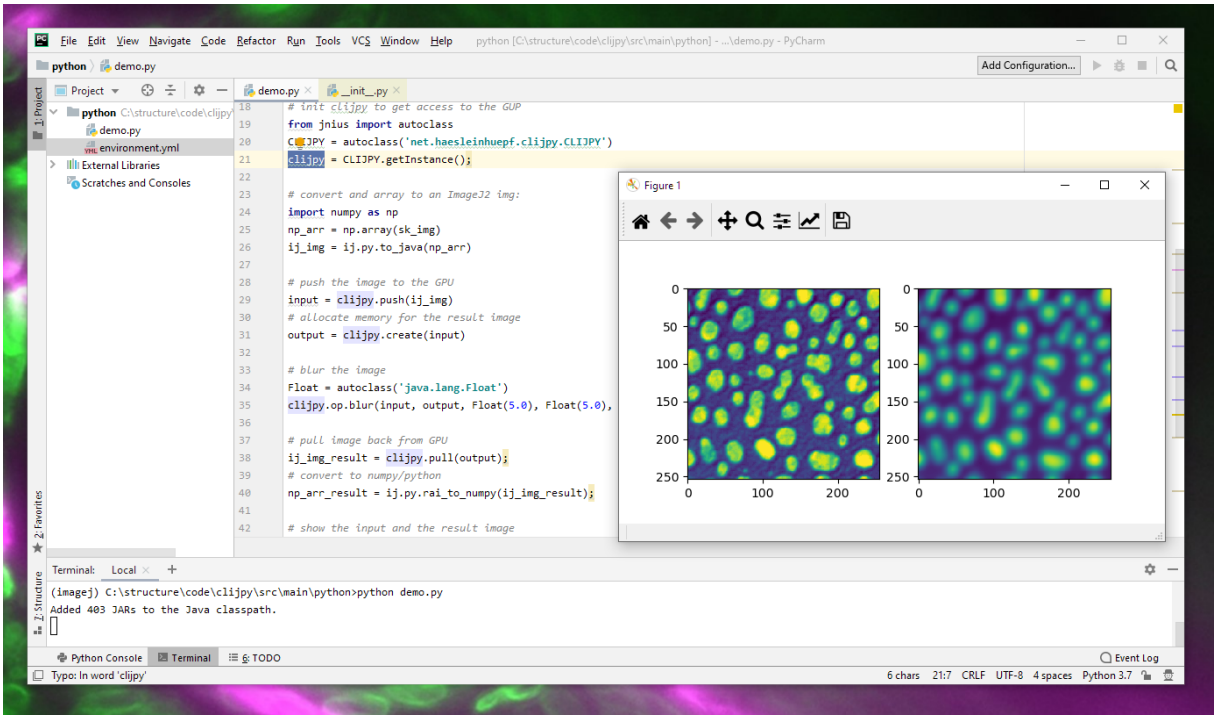

Figure 3. *CLIJPY*.

4. Software description and requirements

This chapter provides an overview of the software requirements for the bachelor's thesis.

4.1 Description

The main goal of this work is to create a graphical user interface for the application that allows users to create constellations created from images.

Even though there is a code to generate these images, some settings of the code need to be manually done and a chance to edit the outline improves the performance. So to facilitate that process and allow people who do not want to interact with the underlying technology (like psychologists and neuroscientists) this UI tool is just for the occasion.

In the interface, you can change the outline of objects in images, the spacing of the constellations, and the frequency of the dots. The interface must also allow the user to change the Canny detection function parameters.

4.2 Importance

Humans rely on generating hypotheses in order to identify constellations hidden inside clusters of dots. It is possible to identify these objects by recognizing a global pattern. After a global pattern is spotted, a hypothesis can be made to see if the suspected object is in the pattern. If the pattern is not a match for the hypothesis then a next hypothesis is made in order to identify the object. This process is repeated until the object is identified. The process of revising hypothesis until finding a solution is called „iterative inference“ [5].

Constellation images can help scientists study iterative inference in the minds of humans and machines. With a dataset of constellation images it is possible to not only study iterative inference but even create neural networks that could identify objects in the noise of dots.

The importance of this thesis is to create a graphical user interface that generates these constellation images. In order to make a dataset with thousands of images, there needs to be an interface to correct any mistakes created by the automated process.

The users of this program will be mostly psychologists and neuroscientists, who want to use their own images to be made in the constellation format and use it in their research.

4.3 Requirements

Functional requirements

The following is a list of functional requirements that must be met by the application.

1. The user must be able to select an image or a project to begin the image processing.
2. The user must be able to select between six different masks created from selecting an image.
3. The user must be able to change the Canny Edge Detector parameters.
4. The user must be able to remove and add lines in the outline image.
5. The user must be able to change the radius of the dots in the constellation image.
6. The user must be able to change the distance between the dots in the constellation image.
7. The user must be able to change the background noise of the dots in the constellation image.
8. The user must be able to save all of the images used in the application (original image, outline image, dotted outline, constellation image) in a single folder (Figure 4).

Non-functional requirements

The following is a list of non-functional requirements that must be met by the application.

1. All of the image editing tools should be located on a toolbar.
2. Every dropdown menu should have a tool description.
3. The user should be able to change between different views of the image processing (masks view, outline view and final result view).

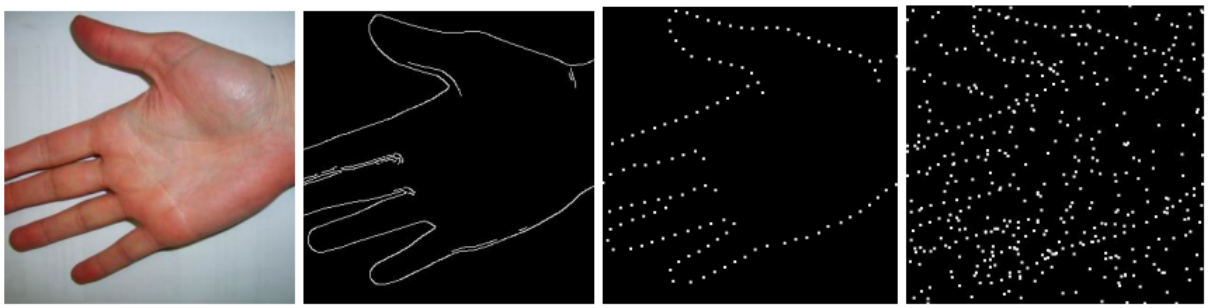

Figure 4. Program result describing view.

5. Implementation

This chapter provides an overview of the tools and software packages needed to create a web interface.

5.1 Tools

5.1.1 GitHub

The Git software repository was used for software version control. GitHub allows the user to create different branches, which allows to make changes to the code in one branch and then merge it with the main branch.

According to Coelho Jailton [2: 1], GitHub is good version control software because users can easily view project history information. J. Jiang [3: 1] states that making additional requests for forwarding a pull request would be adding tags. Adding additional information helps users better understand which versions of a project have a higher priority, and using tags makes it easier to classify versions.

Gitlab was considered as the alternative code manager but the author had more experience with GitHub and therefore GitHub was chosen for the sake of simplicity.

5.1.2 Python

The Python programming language was used to create the graphical user interface. The Pyinstaller package was used to make the web interface executable.

The reason why any other programming language was not used is that trying to access methods from Python scripts is complicated and would require time to understand.

5.1.3 Tkinter package

In order to create a graphical user interface, it is necessary to use a library that is simple and can achieve the desired result. D. B. Beniz and A. M. Espindola [1: 2] point out that the most convenient library for creating a graphical user interface is Tkinter. Tkinter is a good library because it has support for Windows, Linux, and MAC OS. The main reason for choosing the Tkinter tool is the placement of elements and the handling of events.

An alternative for Tkinter is wxPython which has its own pros and cons. WxPython needs to be downloaded and installed while Tkinter is part of Python. Tkinter is easy to learn, but can become overwhelming with complex interfaces. Because this project is not too big, Tkinter was easier to use than wxPython. For some Tkinter may not look as visually pleasing but functionality is more important than visuals.

The Tkinter package was used to create the web interface components. Among the options in Tkinter's tools are a number of layout methods that make it easier to place objects on the web interface. The user interface mainly uses the methods Frame and Canvas. The Frame method places the toolbar and the tools inside it. The Canvas method is used to place the images generated by the program.

5.1.4 C++ Build Tools

The coco_metric package used in this work requires C ++ Build Tools in addition to Python version 3.7. The Visual Studio C ++ build tool is a compiler that can be used to develop or debug C ++ code.

5.1.5 Thonny

Thonny is Python's integrated programming environment that is user-friendly and easy to use. Compared to other development environments, Thonny has the simplest debugging feature possible. Thonny already has Python 3.7 installed, which makes setting up the environment easier [4]. In Thonny's main view, there is a command-line that can be used to read error messages or run a program line by line. The command-line outputs the program’s text output.

Visual Studio Code was considered as an alternative because it has a wide variety of features that would benefit the development of this project. Visual Studio Code has Git embedded in it which would make version control easier. Visual Studio Code was not chosen for this project because changing Python versions took too long time while an existing version was already been used. Thonny did not require changing the Python version and compared to Visual Studio Code, managing packages was easier because it was possible to see all the Python packages and their versions installed from the Thonny interface.

5.2 Tool configurations

This chapter gives an overview of the exact settings and configurations of the tools used in this project.

For the base program to work the correct Python version must be used. The 64-bit version of Python 3.7 or older must be used for this program to work, otherwise it will not be possible to import the tensorflow version 1.15.2 package. The base program uses the COCO dataset to generate masks from the image. In order to use the COCO dataset C++ Build Tools must be installed. Microsoft Visual C++ 14.0 or greater is required for this to work.

5.3 Tkinter components

This chapter gives an overview of some of the Tkinter components used in this project.

To create a layout that can be reasonably modified, the Tkinter Frame widget was used for placement of all widgets. The toolbar components were all wrapped with Frames so if there is a need to add more components in the future it would be relatively simple by placing the component inside the main toolbar Frame. Having components wrapped with frames means adding new components does not change the previous layout at all. The select button and its footnote are both inside a frame which means neither component is affected by each other. The same can be said with other button components located in the toolbar. Using Frame to create the layout was more beneficial than using any other widget because it is much easier to build the layout using one component and not rely on other components.

All of the images are displayed on Tkinter Canvas widgets. The widget is useful when there is a need to actively change the images displayed on the widget. Some Canvas widgets have Tkinter Button widgets attached to them. Clicking a Button widget on a Canvas widget calls a function that hides the current Canvas components and displays new Canvas widgets. Three view changing Button widgets located in the toolbar frame use the same function of hiding currently displayed Canvas widget and displaying new Canvas widgets. More about view changing buttons is described in the design chapter.

Pencil tool and eraser tool both use the Tkinter Button widget. The Button widgets each have a function that is called when clicked. The function, when called, allows the user of the program to draw on the outline image canvas. The line is created through a function that detects if the left mouse button is pressed and creates ovals on the canvas region where the cursor is located. Example of the Tkinter oval creating function: canvas.create_oval((last_x, last_y, event.x, event.y), fill='white', outline='white', width=pencil_size_value). The drawing colour depends on the button that was clicked. The „Pencil“ button allows the user to create white lines while „Eraser“ button allows the user to create black lines.

The Tkinter Combobox widget allows the user to select one value in a set of values. In addition, it allows the user to enter a custom value. The Combobox widget was used so users would be able to change the parameters of the image processing methods. All Combobox widgets are located in the toolbox frame.

The Tkinter Label widget was used to create footnotes for all the tools in the toolbox.

6. The Design

This chapter describes the design used to create the interface.

6.1 Initial prototype interface views

The initial prototype was created by the author to understand how to use Tkinter and its main purpose was to know how the final user interface should look like. The initial prototype was used as a reference to understand what requirements were needed to be made.

The base program uses a pre-trained mask-RCNN [6] model which is accessed from a public Google storage bucket. The mask-RCNN model was used to generate masks out of the images. In order to use the pre-trained model it was necessary to download the metadata of the model from the Colab notebook where the base program was located. The downloaded model could then be used offline while the base program requires access to internet due to the fact that it requested the model from the google storage bucket each runtime.

The user can select an image file from a dropdown button „File“ (Figure 5) located on the top-left of the window. None of the tools is available until the user selects a file and a mask.

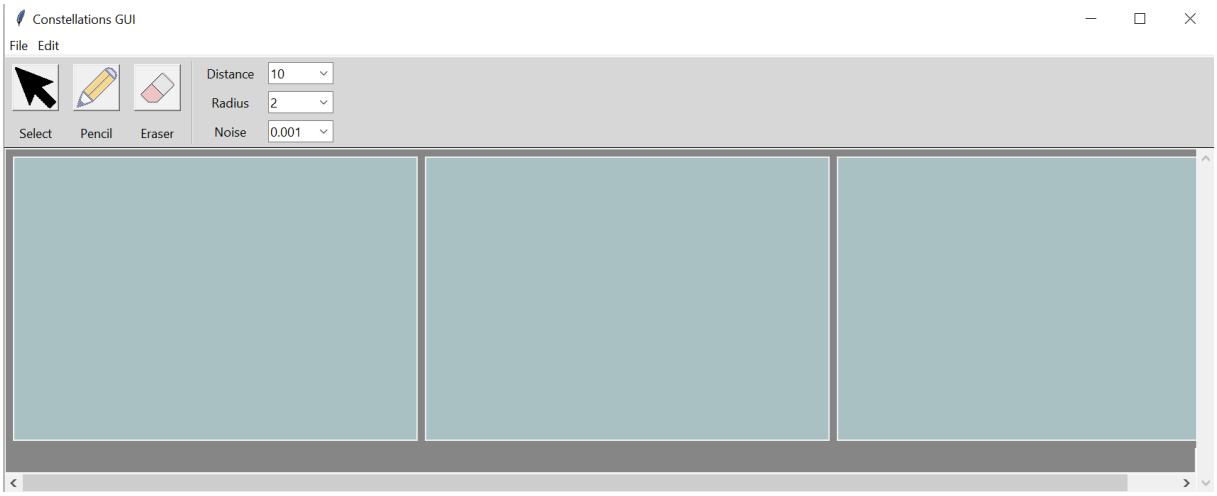

Figure 5. Initial program view after opening the program.

When the user has selected a file a new window appears with a selection of three masks created from the image. There are also two sliders for the user to change the Canny edge detector parameters (Figure 6).

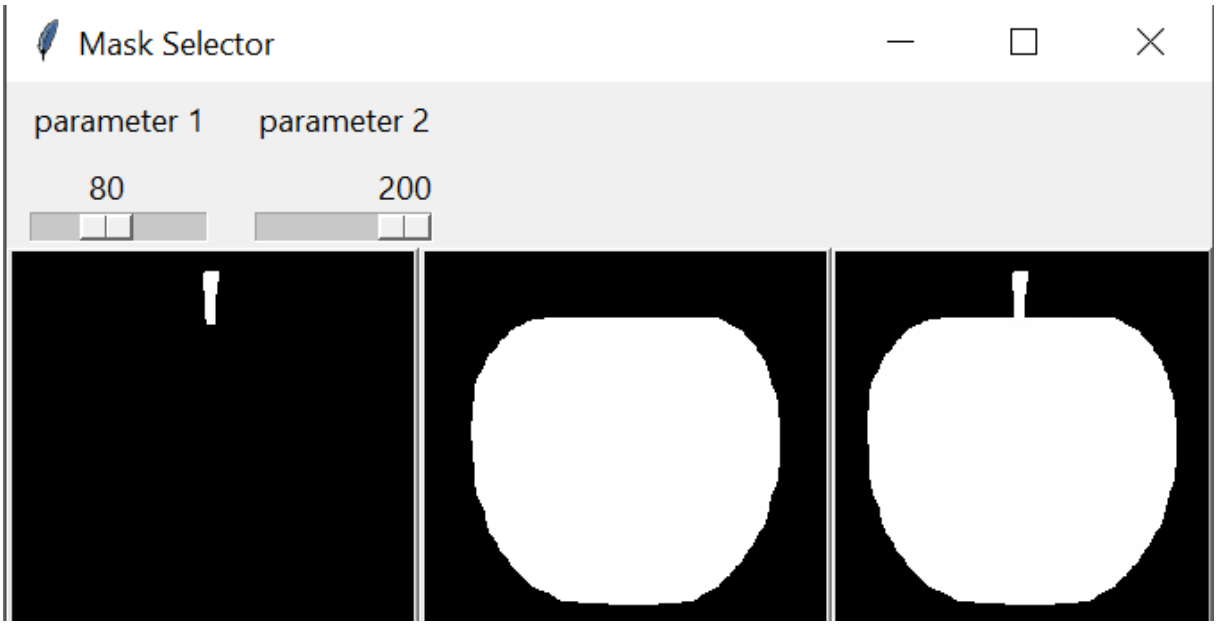

Figure 6. View of mask selection window.

The editing view is where the user can edit the images with selected tools or with three different parameters. After the user has selected a mask there will be three images generated from the mask and loaded on to the main frame (Figure 7). On the first image from the left is an outlined version of the mask that was selected in the mask selection phase. The outlined version of the image can be edited with either a pencil tool or an eraser tool. On the second image from the left is a dotted outline of the first image and it is updated after each edit of the first image. On the third image from the left is the constellation image made from the dotted outline image and it is also updated after each alteration of the first image.

The user can use the pencil tool to add more lines to the outline image (first image from the left on Figure 7). The eraser tool can be used to remove unwanted lines on the outline image. There are three dropdown boxes that can be used to change the constellation images. The „Distance“ dropdown box changes the distance of dots on the dotted outline image and the constellation image. The „Radius“ dropdown box changes the radius of the dots on the dotted outline image and the constellation image. The „Noise“ dropdown box sets the density of the dots that are added to the constellation image.

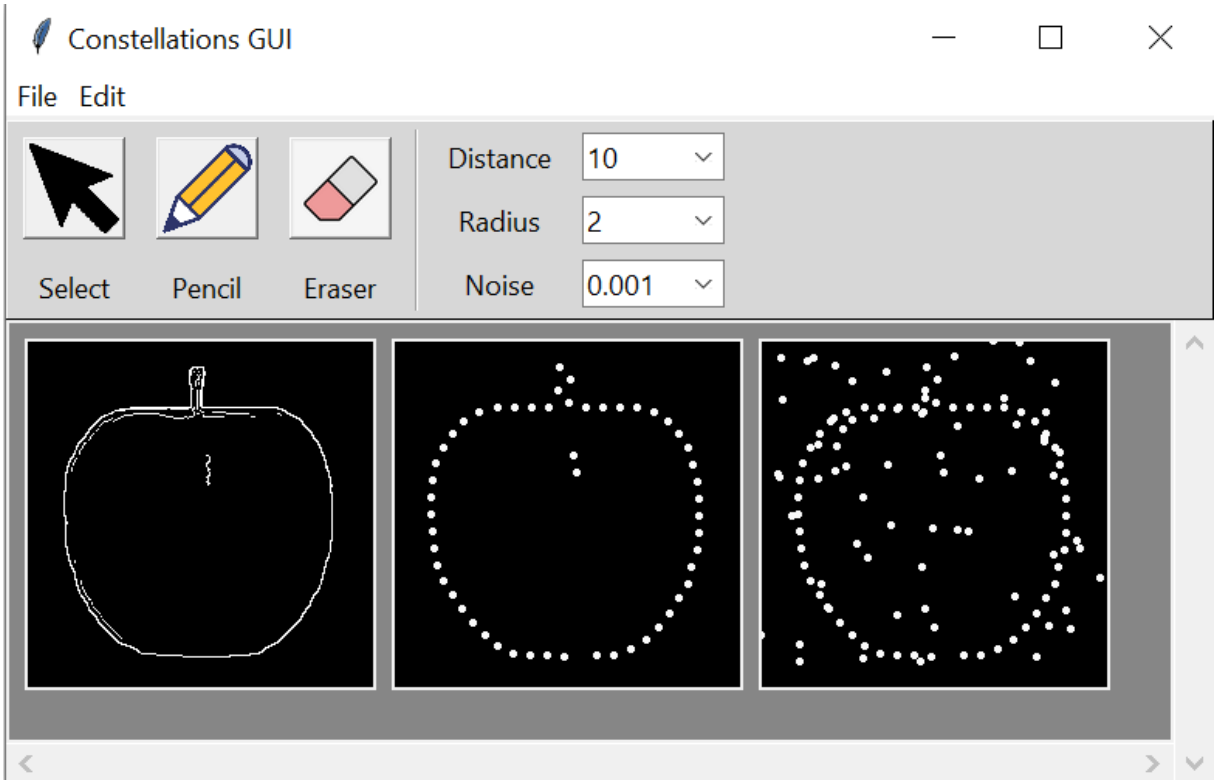

Figure 7. View of edge, dotted outline and constellation image.

After the user is satisfied with the result they can select a folder to save the files into. In total, four images are saved as a result: the initial image, the edge outline of the image, the dotted outline of the image and the constellation image.

6.2 Final user interface views

The final user interface was created so that the users can switch between different views which means when they accidentally choose a wrong mask they can use a button to return to the previous view. The new version of the interface is useful in cases where the user must modify bigger images. The edge image view (Figure 11) allows the user to experiment with different Canny edge parameters, the same result can also be seen on the dots image view (Figure 12) but having a separate view for only outline image means that only one image is updated when a parameter is changed. In the dots image view, all three images are updated when a parameter is changed (Figure 12). It takes seconds to process multiple images and that is why having an edge view is beneficial.

The final graphical user interface is not much different from the initial interface. Compared to the initial user interface there are more parameter options and the ability to switch from different views. The user can select a file the same way as it was done in the initial version.

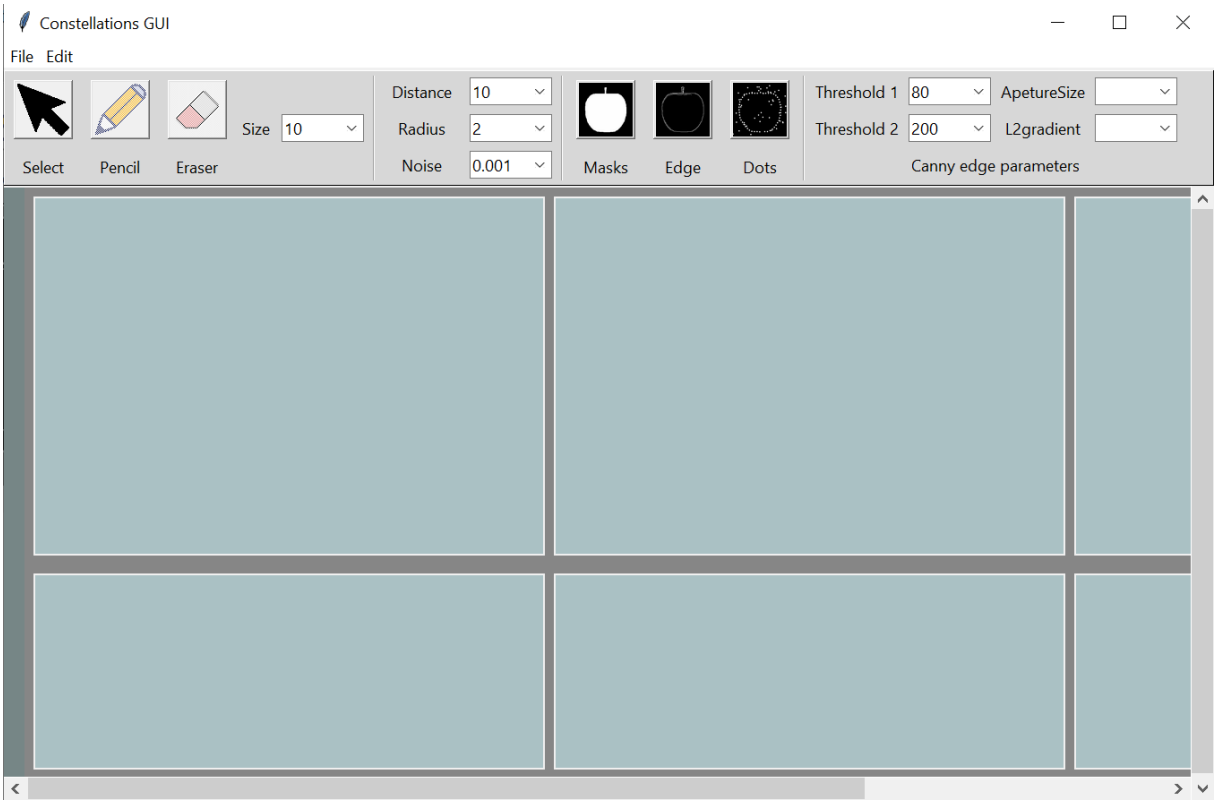

Figure 8. Final program view after the program has been opened.

For the sake of simplification an example image of an apple (Figure 10) is used to visualize the output views (Figures 9, 11, 12) of the graphical user interface.

The mask selection view has been moved from a separate window view to the main window. Compared to previous version of the interface there are six masks to select from. The first three masks on the top row (Figure 9) are segmentations detected by the program. If the program can only find less than three segmentations then only those masks are displayed that were detected. If three or more segmentations are detected then combinations in sets of two of the first three segmentations are displayed on the bottom row of the window (Figure 9).

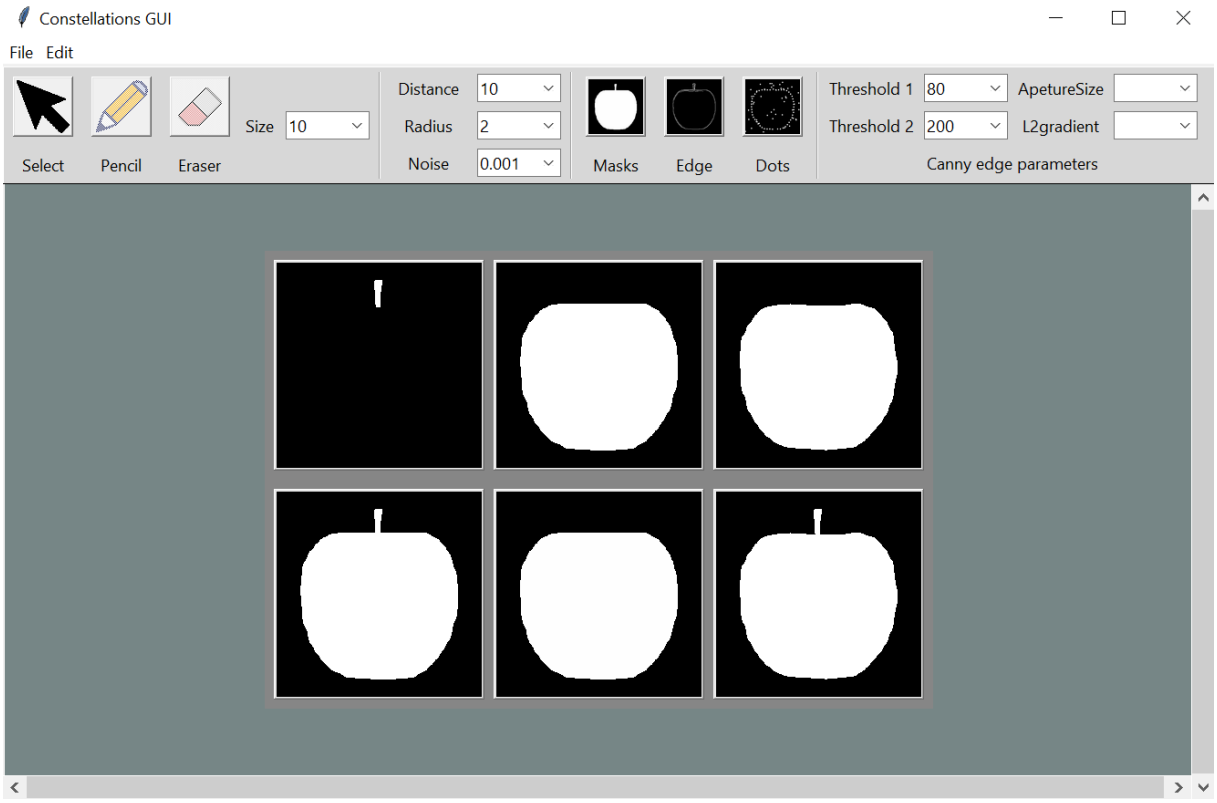

Figure 9. Mask selection view.

After the user has selected a mask from the masks view a new view opens up for editing the outline image. In this view the user can change the Canny edge parameters to get different results of the outline image. The parameters are located on the right side of the toolbar.

Figure 10. Apple image used for image processing.

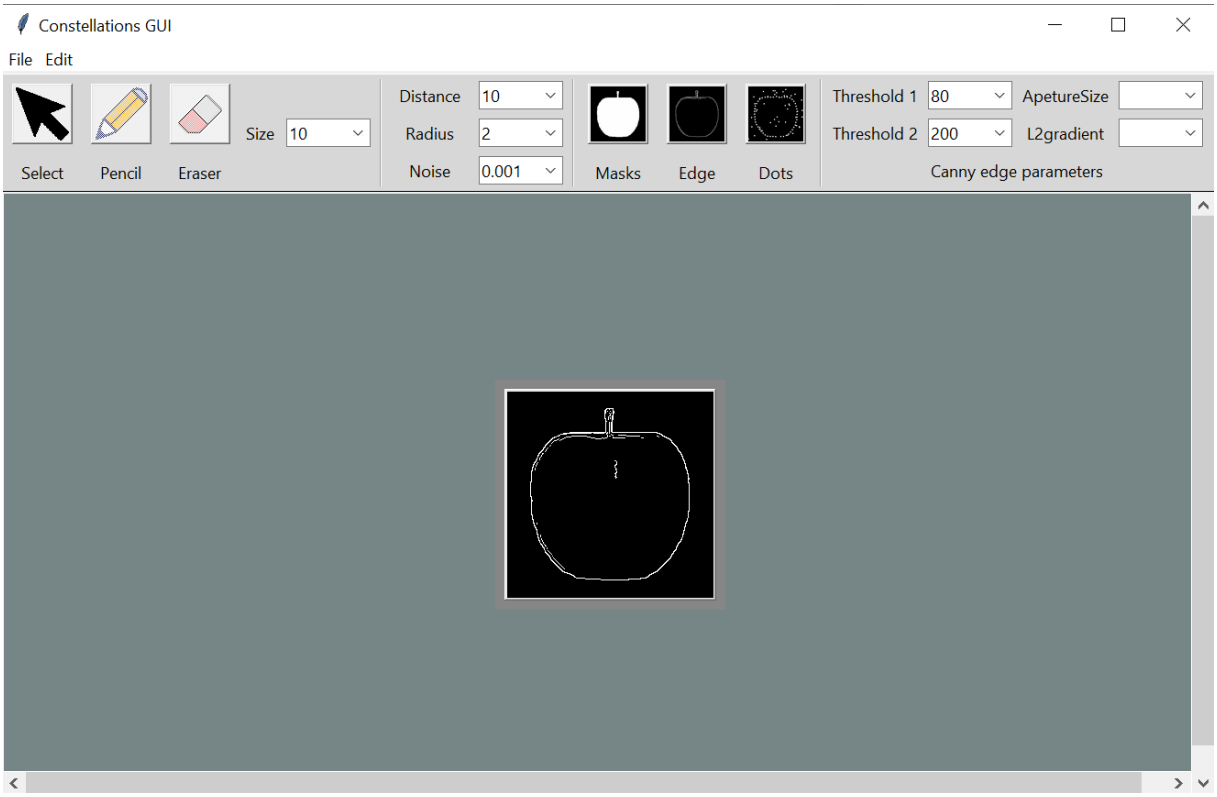

Figure 11. Edge image view for adjusting Canny edge detector parameters.

The constellation view (Figure 12) has the same features as described in the initial prototype interface chapter. Compared to the initial version of the interface there are options to change the size of the pencil and eraser which makes editing images easier. In this view the user can change Canny edge parameters which would update the outline image and erase any edits previously made to it. Changing „Distance“, „Radius“ or „Noise“ parameters does not erase the edits made to the outline image.

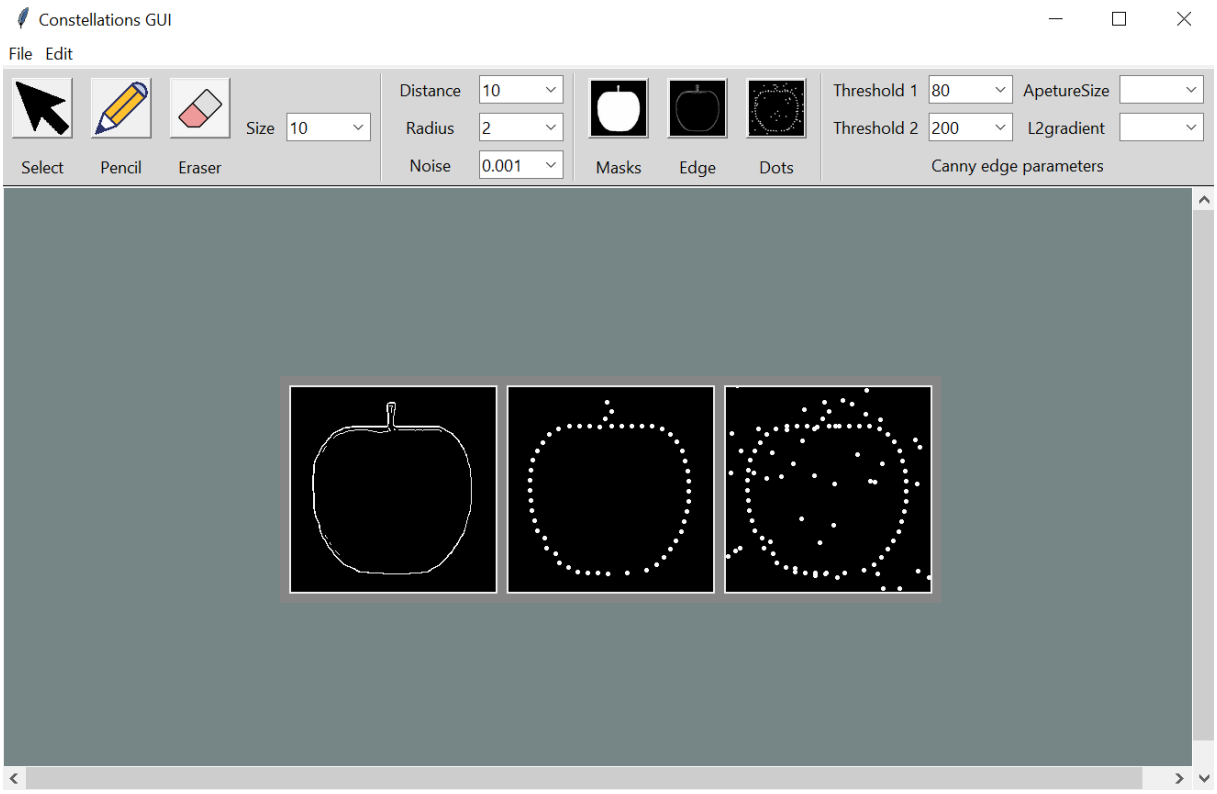

Figure 12. Constellation view for editing object outline image.

The user can use the three buttons on the toolbar marked as „Masks“, „Edge“ and „Dots“ to move between different views (Figure 12). If the user wants to save the images they can choose a directory and write a folder name for the images to be saved in. In the newly made folder, four images are saved including the initial image, outline of the image, dotted outline of the image and the constellation. The saving feature is located in the „File“ dropdown menu located on the top-left part of the window. The „File“ dropdown menu has options of „Save Project“, „Open File“, „Open Project“ and „Exit“. The „Open File“ option is meant for selecting a single image file to use for image processing. The „Open Project“ option allows the user to open a folder that contains a result of previously saved project images. The „Exit“ option closes the application.

7. Challenges and future work

This chapter describes the challenges, limitations and future expansion possibilities of this project.

7.1 Challenges

The main challenge of this project was trying to integrate the base program with the graphical user interface. The first issue that happened was trying to figure out which Python versions were acceptable for tensorflow version 1.15.2. When not being careful it is possible to dismiss a detail regarding what version was ultimately required. By default Thonny has Python 3.7 installed but it is a 32-bit version which conflicts with tensorflow version 1.15.2 which requires a 64-bit Python 3.7.

The second complicated issue was trying to extract the pre-trained model from google storage bucket. When using any other method other than Google Colab you will receive an error message when trying to load the model from the bucket. That means that Google Colab was necessary to extract the model. There were multiple ways to save the model but the only way the model would work offline was to save the metadata of the model.

The most problematic issue was trying to save the project in GitHub repository. One of the pre-trained model files exceeded the file size limit. For some reason the same issue did not occur afterwards.

7.2 Future work

In the future it is possible to improve the program in a number of ways. One of the main issues with the current implementation is that it is single-threaded and that means processing larger images takes a long time. This program is mostly linear, meaning it would be difficult to make it multi-threaded.

If there is a need to make many images in bulk it would be possible to add a feature to the program that selects a single folder and generates constellation images for each of the images located in the folder. After all the images are created the user can check for any mistakes made in the automatic process.

To make things easier for the user there could be a „Configurations“ option in the interface that allows the user to set the default values for the parameters in the program. That way the user does not need to change the parameter values each time or change the default values in the code itself.

8. Conclusion

The purpose of this Bachelor's thesis was to create a graphical user interface, which allows the users to create constellation images and change the parameters of the program using the interface. Without a user interface it would be much more difficult to remove any excess lines generated by the program.

In this thesis the author created a graphical user interface for generating constellation images. With the user interface it is possible to adjust the outline image by changing the Canny edge parameters and removing unwanted lines. It is also possible to change the density of the constellation background and the size and distance of the dots. Most of the functional features in the graphical user interface would otherwise have to be done manually.

The users of this program will mostly be psychologists and neuroscientists, who want to use their own images to be made in the constellation format. This allows them to generate such images for their research without requiring any working knowledge of the underlying python code.

The current version of the project is very limited and could be improved for better efficiency. With larger images, it takes considerably longer to generate the masks and update the constellation images after each modification of the outline image.

Depending on the need to expand the project, it is possible to add more functions that give the users more freedom to change other program parameters. The Canny Edge Detector function has multiple program parameters that can be used if needed.

References

[1] D. B. Beniz, A. M. Espindola, 2016, Using Tkinter of Python to Create Graphical User Interface (GUI) for Scripts in LNLS, 1-3 p.

[2] Coelho Jailton, Valente Marco Tulio, Milen Luciano, Luciana L. Silva, 2020, Is this GitHub project maintained? Measuring the level of maintenance activity of open-source projects, B.V. Elsevier, 1 p.

[3] Jiang Jing, Wu Qiudi, Cao Jin a, Xia Xin, Zhang Li, 2020, Recommending tags for pull requests in GitHub, 1 p.

[4] Thonny. https://thonny.org/ (09.12.2021)

[5] Ruben S van Bergen, Nikolaus Kriegeskorte, 2020 Going in circles is the way forward: the role of recurrence in visual inference. Current Opinion in Neurobiology 2020, 65:176–193, 181 p.

[6] Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick, 2017 Mask r-cnn. InProceedings of the IEEE international conference on computer vision, 2961-2969 p.

Appendix

1. License

Non-exclusive license to reproduce thesis and make thesis public

I, **Kalmer Kaurson**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Graphical User Interface for Constellations Image Generator,

   supervised by **Tarun Khajuria**.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.
3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.
4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Kalmer Kaurson

10/05/2022