User Manual - Image Color Analysis Tool
Group 7 - Ariana Beeby, Max Figura, Jason Jopp
CSC443 Semester Project "Computer Vision"
Manual by Max Figura - 2025/05/06

Contents

1 Introduction

This is the result of the semester project by Group 7 in CSC443 Software Engineering. The scope of this project was to create a nontrivial application over the course of the semester, using development structure and techniques learned in the class.

The project is centered around OpenCV, a python library that contains many tools for image processing and computer vision. The result is an application that can detect objects within an image, given a color to search for. This is achieved through a Graphic User Interface (GUI) which allows a user to select different input options and view and save the results of analysis. Object detection can be achieved with both 'blob' detection, where like-colored pixels

are grouped locally, and contour detection, which searches for the edges of objects. Functionality also includes color range input selection and limited video analysis.

While designed to have a functional and intuitive interface, the product is not in its current state well-suited for use by most clients. Object detection in photographs is not reliable and can require a significant amount of fine-tuning. Rather, the software allows for an academic exploration into the methods of computer vision and image processing. With further development, it could also serve as the foundation for more sophisticated object detection and tracking in video, which in turn could be used as part of a stablisation engine.

2 Development

2.1 User Stories

- **Process for Analysing an Individual Image: As a user I want an outputted image which shows the estimated locations of the object(s) of interest based on their color so then I can see where those colored objects are in an image.**
  Functionality was added with OpenCV to mask an image according to color, then use blob detection to locate an object through the mask. Different kerneling sizes can be used to generate the mask, and measures are taken in blob detection to avoid issues with objects located on the edge of an image.

- **Graphic User Interface: As a user, I want to see a graphic interface of the image and functionalities so that I can more easily operate the program.**
  A GUI has been successfully implemented using Pyside6 (Qt). It allows for graphic file selection, input options, and analysis, and it functions consistently across platforms.

- **User Color Selection: As a user, I want to choose a color to detect within the image so that I can try to find a specific object.**
  A special color picker interface was made, tailored to the project requirements. It allows the user to enter numerical inputs in RGB or HSV form, as well as sample a color from the screen. It also allows for the user to select a range of colors, and displays the resulting color space.

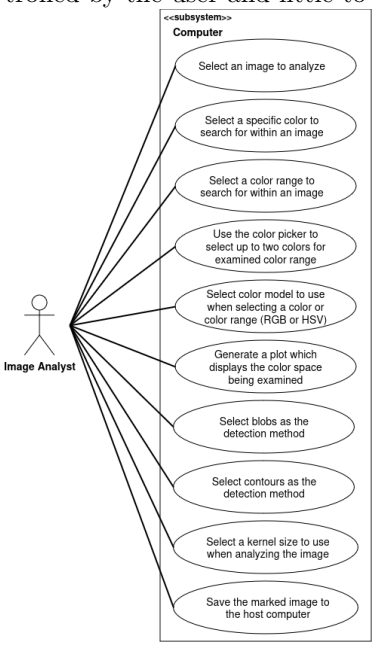
- **User File Saving: As a user I want to have options to save a processed image so that I can keep track of the resulting files.**
  The program by default writes its output to a hard-coded directory, overwriting with subsequent runs. Options were added to the GUI to allow the user to pick a file location and save the image where desired.

- **Contour Analysis: As a user, I want to find an object using contour detection so that I can explore alternate location methods.** Functionality was added that uses OpenCV contour detection, also locating the centroid of a detected object to determine its approximate location in the image. Options were added to the GUI to enable contour detection and select a point such that the nearest detected object can be determined.

- **Color Selection in Video: As a user I want to detect and select a color in a video and track it as the video progresses so that I can eventually track full objects.**
  Functionality was added to read, process, and write videos frame-by-frame. The GUI was updated to accomodate video, working by displaying the first frame of the file.

2.2 Use-Case Diagram

The use-case of the program is very simple, with all functionalities being controlled by the user and little-to-no backend necessary:



2.3 Implementation

The program is entirely implemented in Python, and achieves its various funcionalities with the use of additional libraries. OpenCV is the primary functional library, used for most image processing and it relies upon NumPy for array structures. On the user end, PySide6 is used to create and manage a Qt interface. Pandas and Plotly play a minor role in the application, used to generate color

space maps for range selection. All code was written by the developers or else adapted from official documentation for the various libraries.

In its current state, the project is stored and distributed using GitLab and is managed with Git. Development was completed across different operating systems and software tools, and the application is designed to function consistently regardless of environment.

2.4 Testing

[Please see Full Test Descriptions and Logs, appended to this manual.]

2.5 Risk Register

Risk	Risk Category	Likelihood	Impact	Mitigation Strategy	Risk LVL (1-5)	Notes
Lack of technical expertise	Technical Risks	High	High	Refer to documentation when creating program, communicate with industry professionals for advice.	4	
Difficulty integrating disparate libraries	Technical Risks	Medium	Medium	Have all functionalites routed through a "main" controller using default data types when possible	2	
Access to nondesired files of user	Security Risks	Low	Medium	Set strict bounds for what files can be opened (non image file)	2	
Overwriting of user files	Security Risks	Medium	Medium	Allow the user to set the location files are saved at	2	
Minimal QA	Process Risks	Low	Low	Due to a lack of exterior customers, most of our operational use is QA.	1	Due to low overall operating risk of program, if something goes wrong it is not dangerous to the team as a whole.
Project scope drift/mismatch	Project Management Risks	Medium	Low	Coordinate regularly so developers understand the direction of the project and progress within the scope of the project.	2	If the scope of the project is not clear, developers could work on functionalities which are not actually used in the program.
Incapacitation of team member	Human Risks	Low	High	Sufficiently document own work	2	
Cost estimation errors	Financial Risks	Low	High	Keep low costs (none)	1	Note, we have no cost estimated overall
The team will be liquidated by May 9	Organizational Risks	High	High	Plan multiple acceptable end states and keep track of what is attainable before the deadline	5	
Major changes to libraries used	External Risks	Low	Medium	Continue use of older version, if necessary	1	The project is not likely to be long-lived enough for features to become unsupported

3 Deployment

3.1 Installation

The project does not at this time have a proper distribution set up; instead, files must be cloned or downloaded from a Git repository remotely stored at GitLab. Due to Python's interpretive nature, the application can be used on any system that has the language installed with only minute differences in behavior. To ensure the necessary libraries are available, a virtual environment should be created and `pip` used to install the items listed in `requirements.txt`. Once this has all been done, the application can be launched by running `main.py` with Python.

3.2 Usage

The usage of the application is largely simple and straightforward. Once the program is open, the user can import an image or video file using the 'Select File' button in the top left. A preview of the image will then appear in the top right. They may select the color of the object to detect in a number of ways:

- Simply enter a single set of RGB values
- Select 'Input HSV' and enter a single set of HSV values
- Select 'Sample', thenk click a point on the screen (presumably clicking the object within the image)
- Select 'Use two-color range selection', use one of the above options to pick a color for both inputs, then select 'Update Range' to view the color space

The user can next choose a processing kernel radius; a larger value will allow for better detection of an unevenly-textured object but may miss smaller objects. One may also use the 'Use Contours'/'Use Blobs' button to toggle between blob detection and contour detection.

Once the desired inputs are set, the user can select 'Analyse', after which a copy of the input image will appear, marked up to show the located object. If desired, the user may adjust input values to improve the results and re-analyse. When the desired output is achieved, the user can use the 'Change Location' button in the bottom left to select a file save name and location, then select 'Save Output' to print the image there. The application can be exited at any time by closing the window.

4 Future Development

Because of the deadline and limited resources for completion of the project, further work could include a number of both improvements to functionality and newly-implemented features. Examples include:
Fixes/improvements

- General improvements to image detection and reliability
- Fix of color space display bugs - ordering and scaling
- Prevent non-image/video files from being opened
- Integrate media player within application for video viewing

New features

- Full implementaiton of location selection when using contour detection
- Full implementation of image save resizing
- Exporting of detection metadata
- Direct camera integration
- Full continuous object tracking in video
- Video stablisation through object tracking
- QR code recognition and processing

5 Appendix: About the Developer

Max Figura is an undergraduate student studying Computer Science at Augsburg University. He has some preceeding experience with small- and medium-scale self-directed software development projects. In addition to creating an arcade-style video game from scratch and designing a few basic custom audio-visual tools, he has implemented an existing stochastic mathematical model as part of an undergraduate research experience. Max does have more limited experience with collaborative development and team management, but previously took an administrative and documentative role as part of FIRST Tech Challenge team.

As part of Group 7, Max was primarily responsible for the design and integration of the GUI. They worked primarily with the PySide6 library and contributed almost all of the code for the interface, along with managing the "controller" to connect all functionalities to that event-driven loop. Max also served as scrum leader during one of the project's sprints, although external circumstances resulted in a lower productivity across the team during that time. However, the team generally had a well-balanced structure and would regularly reassess the project's status, making consensus decisions regardless of the current assignment of scrum leader.

The project was chosed with a relatively tight scope that could be expanded as desired and according to the resources availble. This in turn required effective communication and agreement between the team members on what proposed features and functionalities should be implemented next. For example, during the initial development of the UI, it was found that the default color selection widget in Qt was not well-suited for the project. The team agreed that at the time it was sufficient as a method of inputing color values and left it unchanged for the first iteration. However, once it was decided that two-color range selection would be added, creating a custom color-picker became a greater-priority task. By then, the other developers also had a more refined conception of what features would and would not be appropriate for the tool, and so the color picker could be better tailored to fit those requirements.

Project test plan:

Test 0: Initial setup

This initial setup must be performed in order to properly test the program. As the testing team should already have the program set up for development purposes, this section will only loosely describe the initial setup process. For more information, please contact someone from the development team.

Steps:

1. Launch a command line program such as terminal or command prompt.
2. Navigate to a local directory that you can clone the project's git repo into.
3. Clone the project repository onto your local system.
4. Verify all required files are present by referencing the included project README.md.
    a. Ensure the "requirements.txt" file is present for venv setup.
5. Create a python virtual environment using command: "python3 -m venv <venv_name>"
6. Activate the venv command "source <filepath to venv activate file in venv directory>".
7. Install the required libraries using command "pip install -r requirements.txt"

At this point, your system should be ready to perform the following tests in this outline. For all the next tests, ensure that you are within the venv you have just set up in these initial steps.

Note:

- All commands using "python" may need to use "python3" on some operating systems.
- Each of these tests must be performed on a Linux, MacOS, and Windows system.

Test 1: Basic program launch and closure

This test is to ensure that the primary program can be launched using the libraries installed from the requirements.txt file from within a virtual environment.

Steps:

1. Activate your virtual environment.
2. Launch the program using command "python main.py"
3. Ensure the program has launched with no error messages appearing in the terminal.
4. Close the program using the x in the corner of the GUI, again verifying no error messages have appeared in the terminal.

Test 2: Searching for a single color using the color picker function

This test is designed to make sure that the color picker functions properly, and that the program can identify an object of a single color in an image. Test also verifies an image can be loaded.

Steps:

1. Activate your virtual environment.
2. Launch the program using command “python main.py”
3. Choose the “select file” option in the GUI.
4. Select and open “images/shapes.png” image file.
5. In the “colors to Detect” section of the GUI, select the “Sample” button.
6. Select the red square to select a color from.
7. Select the “Analyze” button in the bottom left corner of the GUI.
    a. At this point a marked image should appear which displays a green circle surrounding most of the red square. This verifies that the program was able to find that color of object within the image.
8. Close the program.

Note: Use of shapes.png has been known to result in a warning of "qt.gui.imageio: libpng warning: sBIT: invalid". Any output of this form can be ignored.)

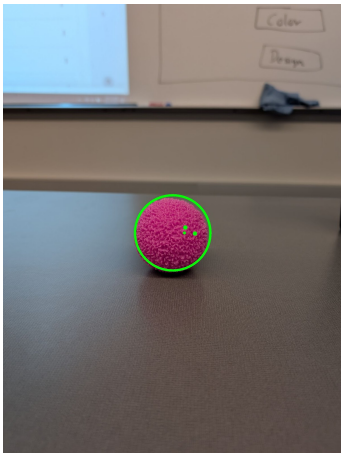
Test 3: Searching for a real-world object using an HSV color range with blobs and contours, saving a marked image to cache

This test is designed to ensure that we are able to search for a real world object using an HSV color range with a lower and upper bound. This test also verifies that the visualization which shows the selected color range is working properly. This test also verifies that kernel selection is working properly by being able to detect a real-world object with a mixed color appearance.

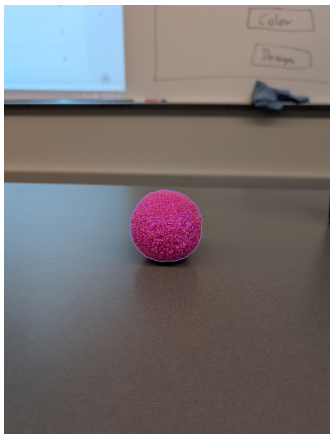
Steps:

1. Activate your virtual environment.
2. Launch the program using command “python main.py”
3. Choose the “select file” option in the GUI.
4. Select and open “images/pink_ball.jpg” image file.
5. Select the “Use two-color range selection” button in the GUI.
    a. At this point a plot should appear with a single black dot in the middle.
6. Select the “Input HSV” button to switch from RGB to HSV color entry.
    a. At this point the color fields should have their labels swapped to Hue/Saturation/Value from Red/Green/Blue.
7. Enter the following color ranges into the GUI:
    a. Hue: 330 - 340
    b. Saturation: 80 - 255
    c. Value: 50 - 255

8. Select the “Update Range” button and examine the plot, verify that it now displays a variety of pink values and saturations, with a narrower hue selection.
9. Select “Analyze”.
    a. At this point a new marked image should be generated which has small green circles on it where the program identifies colors that fall within the given range. There should only be green circles on the pink ball itself and not the background. There should not be a large green circle around the pink ball yet because kernelling is still set to zero.
10. Next, raise the “processing kernel radius” to 1, and reanalyze the image.
    a. There should be no change to the marked image at this point.
11. Continually raise the kernel from 1 to 3 to 5 to 7.
    a. Note that with each increase to the next odd kernel radius, the program is identifying different sections of the pink ball as being the correct color.
    b. At a kernel radius of 7, the program should be able to identify the ball as a whole object, and draw a circle around the entire ball itself.
    c. Expected result:



12. Select the “Use contours” button to switch from using blobs to contours.
13. Select “Analyze” to display the marked image using contour detection.
    a. Expected result:



14. Select the “Save Output” button to save the marked image.

15. In your file browser or terminal, navigate to the images folder within the project and verify that there is now the same marked image named “pink_ball-out.jpg” in the directory.
16. Close the program.

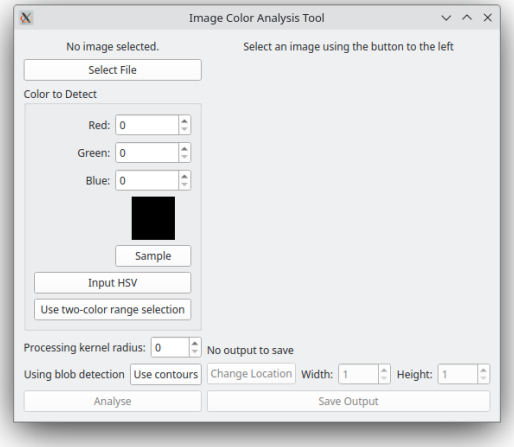
GUI Tests:

These tests are designed to ensure that the GUI displays and functions properly across different platforms. By the very nature of the GUI, most results will have to be confirmed with visual inspection and comparison with the included images. Note that Qt automatically matches the "theme" of applications to the user's window display system, but any substantial difference in sizing, layout, spacing, or behavior (as opposed to differences in color palette or typeface) should be noted.

Test 4: Window Appearance and Behavior

These tests ensure that the window can be manipulated by the user in ways that are consistent with standard window management behavior.

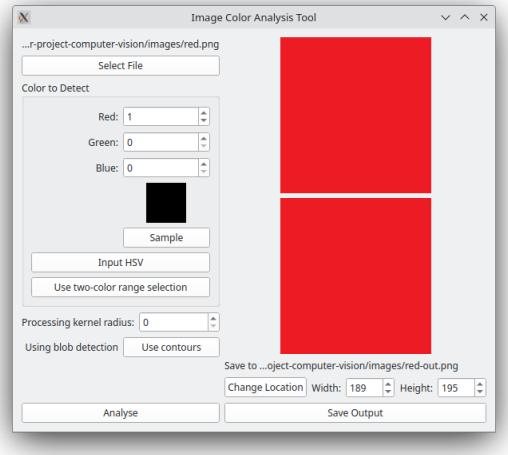
1. Start the program by activating the virtual environment and running main.py with python. The following window should appear



   Verify that the window theme is consistent with the OS/display system appearance, and that "Image Color Analysis Tool" appears in the header along with 'minimise', 'maximise', and 'close' buttons.

2. Click "Select File", which should cause a file dialog consistent with the operating system to appear. Choose and open "red.png" from the "images" folder. The window should increase in size to accommodate the image display.

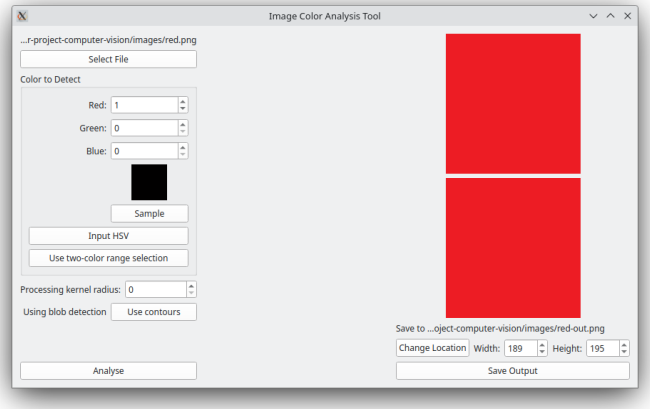
3. Enter any color value within the color picker and click "Analyse". The window should increase in height only to accommodate the output display, appearing as shown.



4. Check each of the following (tests are independent and can be verified in any order)

    a. Click within the window header and drag to another place on the screen. The window should move to the dragged location.

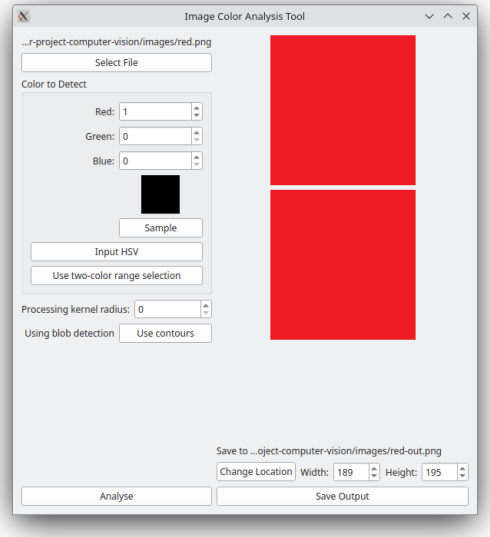
    b. Widen the window by hovering over the right edge of the window until an arrow icon appears, then clicking and dragging to the right. The two columns (identified by the "Analyse" and "Save Output" buttons at the bottom) should maintain their size and separate, leaving a blank space in the middle as shown.



    Reset the width by hovering over the right edge, clicking, and dragging back to the left. The window should return to its minimum width, as in the first picture.

    c. Lengthen the window by hovering over the bottom edge of the window until an arrow icon appears, then clicking and dragging down. A blank space should appear between the "use contours" and "Analyse" button, as well as between the output display and the "Save to..." indicator, will all other features remaining

compact, as shown.



Reset the height by hovering over the bottom edge, clicking, and dragging back up. The window should return to its minimum height, as in the first picture.

d. Click the 'maximise' button in the window header. The window should expand to fill the screen, with elements within the window retaining their size and staying compact as when the window was lengthened or widened. Click the 'restore' button and the window should return to its normal size.

e. Click the 'minimise' button in the window header. The window should disappear, while still being listed in the display's taskbar. Click the icon, and the window should reappear.

5. Click the 'close' button in the window's header. The window should disappear and the program exit, with no print output to the terminal having occurred.

Test 5: Color Picking

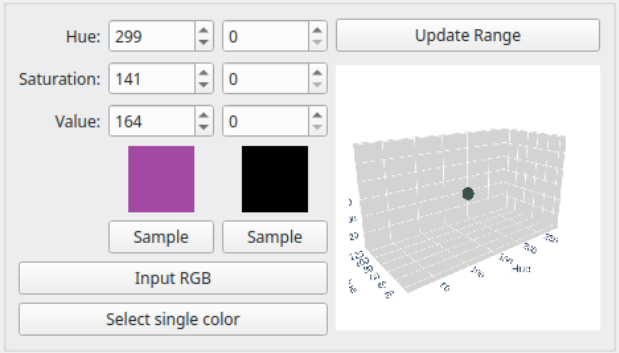
These tests ensure that the user can input color values for detection in a consistent manner.

1. Start the program by activating the virtual environment and running main.py with python. Click "Select File" and choose and open "shapes.png" from the "images" folder.

2. Click the "Sample" button and then click on the purple circle from the preview of shapes.png. The numerical values should change to Red: 163, Green: 73, Blue: 164, with the box underneath showing the same color as the circle.

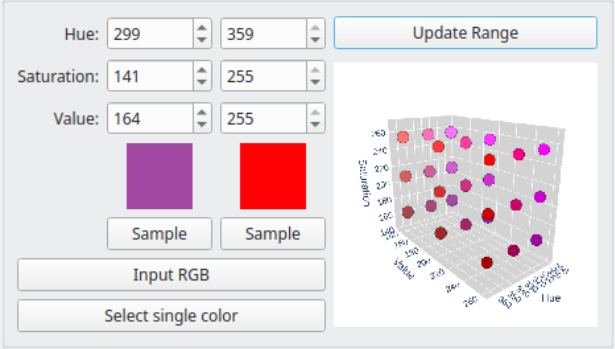
3. Click "Input HSV". The text of the button should change to "Input RGB", and the numerical values should change to Hue: 299, Saturation: 141, Value: 164.

4. Click "Use two-color range selection". After a few seconds, additional options should appear, with the color picker now looking as shown.



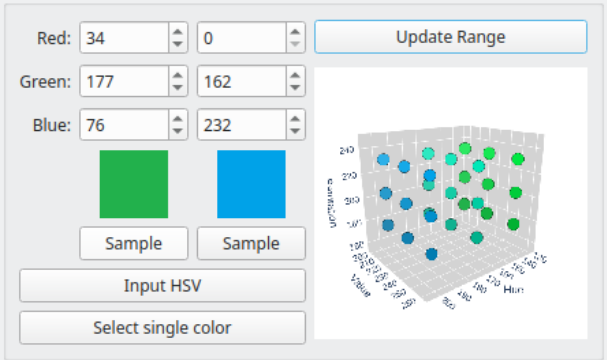
5. In the second column of numerical inputs, manually enter Hue: 359, Saturation: 255, Value: 255. The box underneath should now be a bright red.

6. Click "Update Range". After a few seconds, the color space preview should update to appear as shown.



7. Click "Analyse". A result image should appear with the purple circle and bright red square circled.

8. Click "Input RGB". Use the sample buttons to set the first color to that of the green circle, and the second to that of the blue circle. Click "Update Range". The color space preview should update as shown.



9. Click "Select single color". The extra options should disappear, leaving a blank column in the middle of the window.

10. Click "Sample", then hit the [Escape] key. The cursor should return to normal, with the color not having updated.

11. Click "Sample, then click a point outside the window. The selected color should update to match that color.

12. Click "Use two-color range selection". After a few seconds, the second color and range display should reappear, with the same blue value as earlier.

13. Close the window. The program should exit, without any output printed to terminal. (**Note:** use of shapes.png has been known to result in a warning of "qt.gui.imageio: libpng warning: sBIT: invalid". Any output of this form can be ignored.)

Test 6: General GUI Functions

These tests ensure that the standard behaviors of the GUI can be executed consistently and reliably.

1. Start the program by activating the virtual environment and running main.py with python. Verify that the "Analyse", "Change Location", and "Save Output" buttons, as well as the save size inputs, appear disabled and cannot be used.

2. Click "Select File". Use the file dialog to select and open the image "pink_ball.jpg" from the "images" folder. The image should appear, scaled to a size that is readable but does not exceed the screen size.

3. Sample the image of the ball for an input color, choosing a mid-tone point on the object. Verify that the "Analyse" now appears enabled.

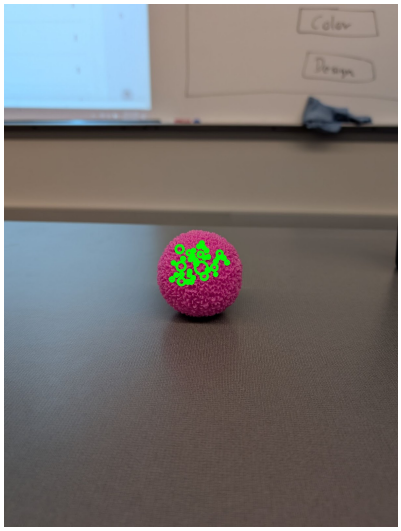
4. Set "Processing kernel radius" to 20 and then click "Analyse". An output image should appear showing several detected parts of the ball as circled with green. Verify that the window has not exceeded the screen size. Verify also that the save settings now appear enabled.

5. Set the "Width" of the save image to 1000. Verify that the "Height" automatically updates to 1328.

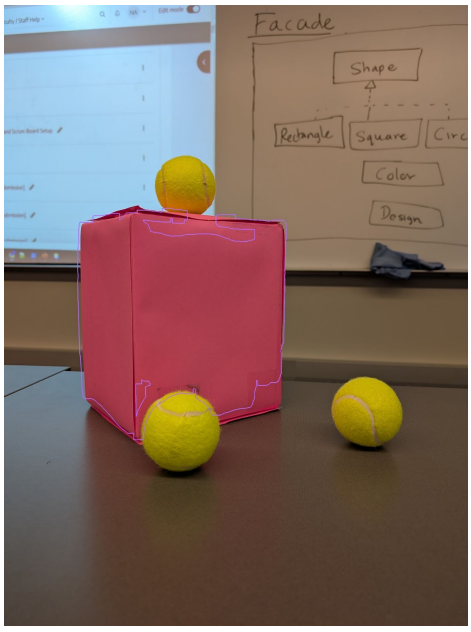
6. Click "Change Location". Use the dialog to select a new path and file name ending with "jpg".

7. Click "Save Output". Find and view the resultant image outside of the program and verify it appears similar to below (**Note:** image save resizing is not actually implemented as of

this test being written)



8. Click "Select File" and open "green_3_pink_box.jpg" from the "images" folder. The top image should change to match, with the lower image still showing the previous output.
9. Sample a mid-tone point from the box. Set the kernel radius to 60. Click "Use contours"; an additional point selection option should appear.
10. Click "Select a point in the image". Click the pink box, approximately at the center of the image. The text should update to a value of approximately (0.5,0.5).
11. Click "Select a point in the image". Click a point outside the image. The coordinates should not update.
12. Click "Analyse". The lower image display should update to show the box highlighted in purple, as shown.



13. Close the window. The program should exit, without any output printed to terminal.

Test 7: Input bounds

These tests ensure that the values of each numerical input cannot exceed the limits imposed.

1. Start the program by activating the virtual environment and running main.py with python. Click "Select File" and choose and open "red.png" from the "images" folder.
2. Click "Use two-color range selection". Verify that for each of the six color inputs that the spinbox arrows cannot be used to decrease the value past 0.
3. Verify that each of the six inputs cannot be set higher than 255 by attempting to type '256', then typing '255' and attempting to use the spinbox arrows to increase.
4. Click "Input HSV". Verify that each input cannot be set lower than 0 as in step 2.
5. Verify as in 3 that both "Hue" inputs cannot be set larger than 359, and that the "Saturation" and "Value" inputs cannot be set higher than 256.
6. Verify for "Processing kernel radius" that the spinbox arrows cannot be used to lower the value past 0. Verify that values higher than 99 cannot be entered by typing or using the spinbox arrows.
7. Click "Analyse". Enter '1' for the "Width" save size, and verify that the "Height" also displays 1. Verify that the spinbox arrows cannot be used to lower the value of either input.
8. Enter '0' for the "Width", and verify that it corrects back to 1.
9. Enter '9999' for the "Width", and verify that it corrects to 9692, with "Height" being 10000. Verify that the spinbox arrows cannot increase "Height", and that they can increase "Width" to 9694 before being corrected back to 9692.
10. Close the window. The program should exit, without any output printed to terminal.

After Testing Reports:

Tester:	Max	**Operating System:**	Kubuntu (Plasma desktop environment)
Date:	2025/04/30	**Version (last commit):**	Latest at time (c06794eb)
Test(s) Run:		4, 5, 6, 7	
Results: Note any discrepancies with expected outcomes, including step number and image if applicable		All tests completed with results matching expectations	
Other Notes:		GUI 3.7 - Multiple attempts at picking a color, running analysis, and saving file were required to generate comparable image.	

Tester:	Jason	**Operating System:**	macOS Sequoia (15.4)
Date:	2025/05/05	**Version (last commit):**	Latest at time (73a34e7e)
Test(s) Run:		0, 1, 2, 3, 4, 5, 6, 7	
Results: Note any discrepancies with expected outcomes, including step number and image if applicable		All tests completed with results matching expectations	
Other Notes:		n/a	

Tester:	Ariana	**Operating System:**	Windows 11 Home
Date:	2025/05/05	**Version (last commit):**	Latest at time (73a34e7e)
Test(s) Run:		0, 1, 2, 3, 4, 5, 6, 7	
Results: Note any discrepancies with expected outcomes, including step number and image if applicable		Test 3 - Slight freeze when selecting 2 color range, blob circle slightly off center Test 5 - When sample purple, then switch to hsv, and switch back to rgb does not stay the same color/values.	
Other Notes:		n/a	