Resistor Value Detector

Computer Science NEA

2020 - 2022

James Gray

Contents

Analysis

Introduction

Project Overview

The application that my client has requested is a resistor detector that identifies the values of a resistor from an input image. A resistor is a device that is frequently used when constructing circuits and its purpose is to oppose the flow of electrical current. When building circuits, using the correct resistor is essential, so a series of coloured bands that represent specific values are printed on them so the resistor value can be identified. The picture below shows a typical resistor with its colour bands and what they represent.

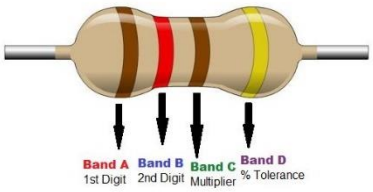


Figure 1 – A 4 band resistor diagram. (Braza,

One way of deciphering these bands is to look them up in resistor value tables, however, using these tables to work out values manually can be a time-consuming task that is prone to errors. The purpose of my project is to produce an application that provides easier alternative for resistor value identification.

I will develop a program that:

- Has a user-friendly UI that also allows for manual input.
- Returns accurate auto-detected values for the majority of the time (has over 80% accuracy).
- Allows for manual overrides of the auto-detected values, ensuring the application output is always correct.
- Is very accessible, i.e., being able to be used on multiple operating systems and device types.

Background

In Reading School, resistors from the electronics stores are used for building circuits in the Electronics labs. Due to time limitations, after labs have finished, the resistors are usually returned unsorted to the electronic stores. Since many resistors may be used in a lab, a considerable amount of time would need to be spent identifying and sorting these unsorted resistors so they may be stored correctly.

Figure 2 – The current state resistors are stored after use

Despite having organized stores for resistors, figure 2 highlights the current storage for many of the used resistors.

By creating an application that is accurate, very accessible, and easy to use, I hope that students can aid the teacher in the storage of these resistors. At the end of the lesson instead of dumping all these used resistors into a tub, both the students and teacher could launch this application and identify the values of the resistors they used. After identifying the value of these used resistors, it would then be possible to place these resistors in the resistor store that is labelled with the value that corresponds to the resistor.

Client

My client is a teacher at Reading School who teaches both Electronics and Computer Science. He has requested an application that can quickly identify resistor values from an image, for both him and his students to use.

I have planned my interview questions so that I can get a good understanding of the problem and have an idea for the system that the client wants. This will allow me to specifically create my objectives around the client's desires.

I have documented the interview questions and their answers below.

Question	Do you have any experience with existing resistor calculation apps, if so, explain your experience?
Answer	No, not currently. But I would find resistor calculator app useful.

Question	What would the use case for a resistor calculator be for you?
Answer	We have a lot of resistors which must end up in small parts cabinets. This process is currently tedious and time consuming as all the resistors must be labelled before being sorted. A resistor calculator would allow teachers or students to sort any resistor, labelled or not, without having to remember any colour codes.

Question	What is the basic outline for what you want the system to do?
Answer	On the most basic level, I would like you to create an accessible application that allows the user to input an image of a resistor and have its values output.

Question	What are your specific demands for the app?
Answer	• That the application works with resistors with any number of bands. • A clear, simple UI with sensible scaling. • That the application can locate the resistor anywhere in an image. • That the application is accessible to as many people as possible. • That the program warns the user when there is an invalid output. • That the application can work both online and offline. • That the application is more accurate than the other applications you showed me. • That the application is accessible on different types of devices and operating systems as the user's device will use is unknown.

After the interview with the client, I gained a much clearer insight into what the system I developed was to be. I have used the information from these interview questions along with the further research I conducted on the client within The Client's Thoughts on Existing Systems section to help me create good objectives for my system.

In addition to this interview, throughout the project, I have asked the client for their thoughts. I will also ensure that I ask my client for their feedback at the end of my project to ensure that they are pleased with the system I have created and if they can identify any improvements or issues.

Research

Studying Existing Systems

By documenting the reviews of existing systems, I could identify their appeals and flaws. I could then design my system around the positive aspects of existing systems, which would allow me to create the best system possible.

I have used the search terms: “resistor calculator” and “resistor scanner” to find the apps that I have documented below. For each app that I have studied, I have made a note of the aspects of the application which I personally find positive and negative, the average review score for the application and the top reviews for that application.

The rating that is stated is the average rating rounded to the nearest star and most screenshots are from their respective Google Play store pages.

Application 1: Resistor color code calculator (Google Play Store, Resistor color code calculator, 2021)

Average Rating

- ★★★★★

Positives

- Application works for resistors with different numbers of bands.
- The UI is simple and clear and relatively easy to use.
- Allows the user to check the history of the calculated resistor values.
- Allows the user to share the resistor bands and its result.

Negatives

- Does not allow the user to calculate resistor values from an image.
- Entering resistor values in a drop-down menu could be hard to use on smaller devices.
- The application cannot calculate resistor values from an image.

Top Reviews

- “Just starting to learn electronics. This app is a big help – awesome!” – Oswald Mann (★★★★★)
- “This useful for lay man to understand resistors. It provide search facility and color code information too.” – Kamram Hassan (★★★★★)

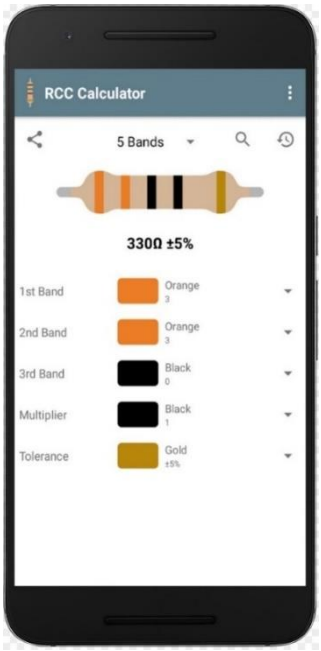


Figure 3 - Application 1

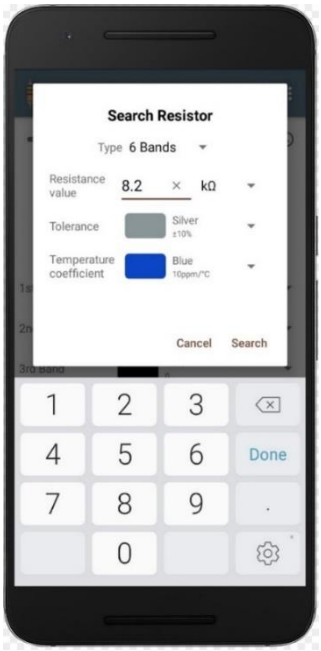


Figure 4 - Application 1

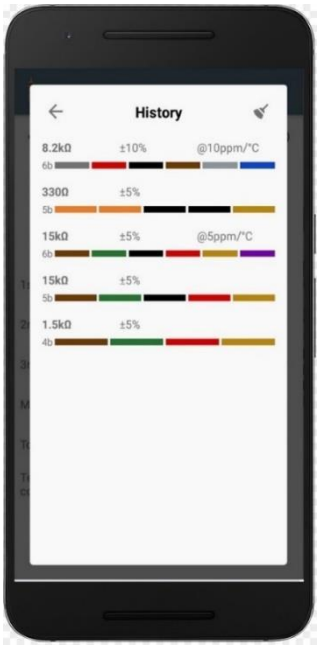


Figure 5 - Application 1

Application 2: Resistor Scanner (Google Play Store, Resistor Scanner, 2021)

Average Rating

- ★★★☆☆

Positives

- Allows for the input of an image.
- Allows for calibration of the colours.
- Allows the user to adjust camera settings.
- Works for resistors with different numbers of bands.
- Allows for manual input of a resistor.

Negatives

- Does not allow for manual override.
- Application output is mostly inaccurate or does not output at all.
- The UI is not very user friendly – it is very cluttered, making the app hard to use.
- Processing is locally done and can be quite resource intensive.

Top Reviews

- “In the rare event that it was able to scan a picture, it took longer than it would take to simply look at a chart. Oh, and it was wrong, every single time.” – Mike M (★☆☆☆☆)
- “It hardly works even after multiple tries to calibrate the colors. Good idea but the execution needs to be improved.” – kbelbeisi (★☆☆☆☆)
- “AMAZED I tried 3 resistors, and it gave me the CORRECT resistance for each of them… actually excellent application.” – Kyprianos Iracleous (★★★★★)

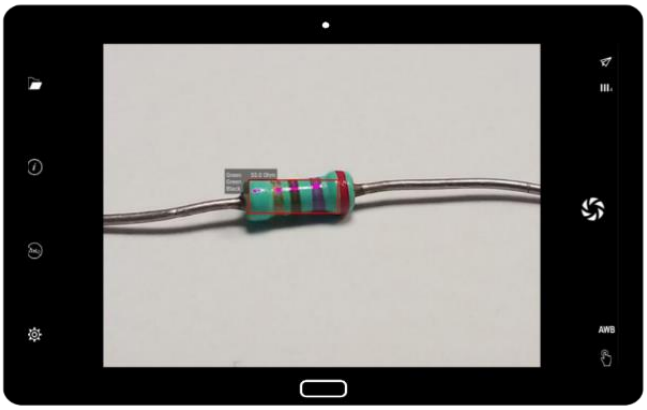


Figure 6 - Application 2

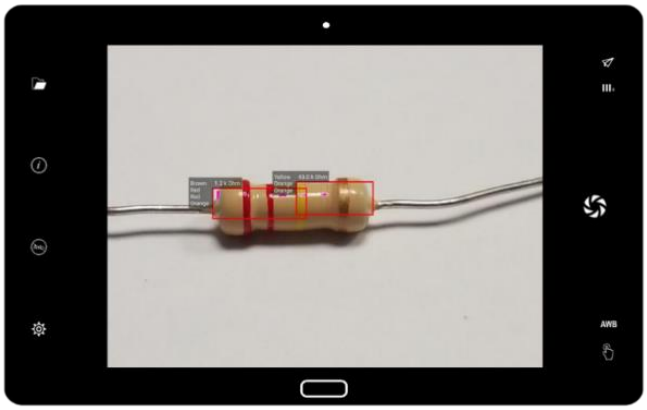


Figure 7 - Application 2

Application 3: Resistance Calculator (Google Play Store, Resistance Calculator, 2021)

Average Rating

- ★★★★★

Positives

- Application works for resistors with different numbers of bands.
- The UI is very simple and clear. It is very intuitive and easy to use.
- Allows different types and formats of resistors.
- Shows a visual representation of the resistor.

Negatives

- Does not allow the user to calculate resistor values from an image.
- Does not allow the user to store calculated values.

Top Reviews

- “Very helpful and easy to use.” – sl249900 (★★★★★)
- “Nice and easy to use app.” – A Google user (★★★★☆)

Figure 8 - Application 3

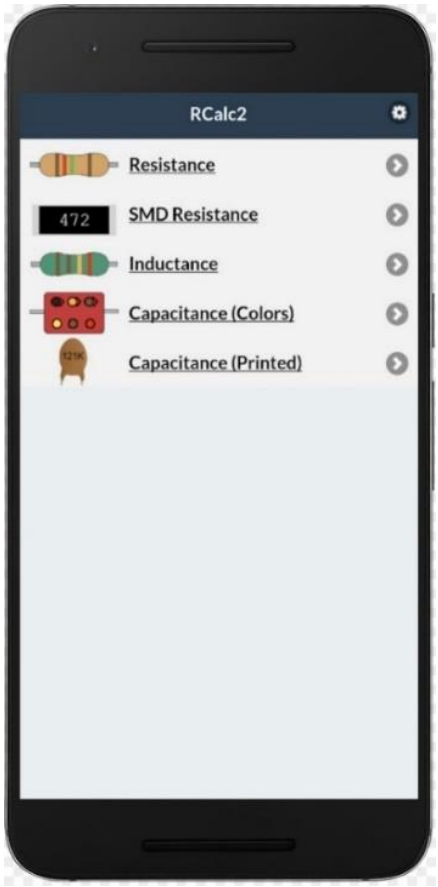


Figure 9 - Application 3

Application 4: Resistor Scanner (beta) (Google Play Store, Resistor Scanner (beta), 2021)

Average Rating

- ★★★★★

Positives

- Application works for resistors with different numbers of bands.
- The UI is very simple and clear. It is very intuitive and easy to use.
- Allows different types and formats of resistors.
- Shows a visual representation of the resistor.

Negatives

- Does not allow the user to calculate resistor values from an image.
- Does not allow the user to store calculated values.

Top Reviews

- “Very helpful and easy to use.” – sl249900 (★★★★★)
- “Nice and easy to use app.” – A Google user (★★★★☆)

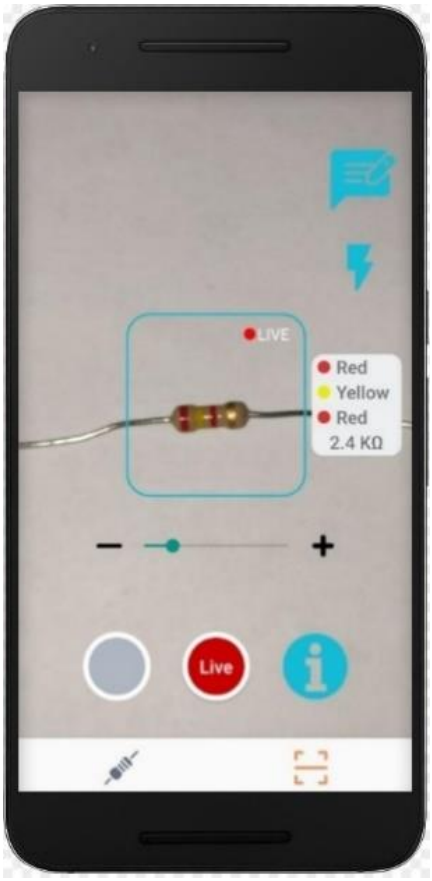


Figure 10 - Application 4

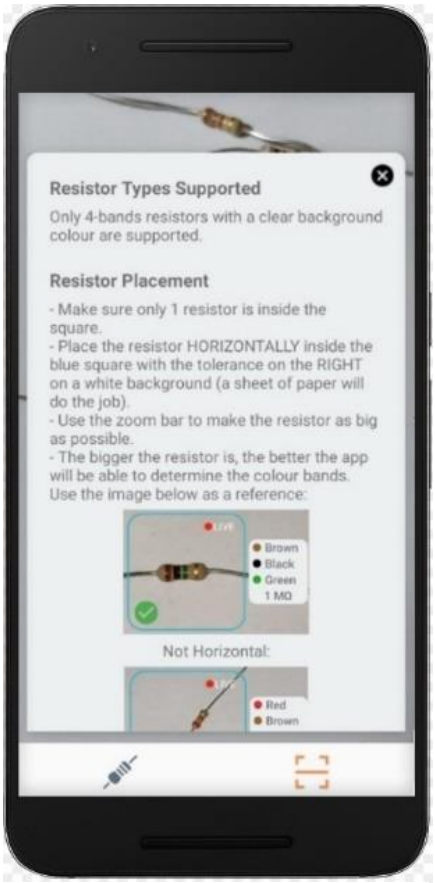


Figure 11 - Application 4

Conclusion of Research

Upon studying these reviews, I could find out what features appealed to users.

Users gave high ratings to applications which:

- Produced an accurate and desired outcome.
- Contained a GUI that is straight to the point.
- Were easy to use.

Lower rated applications tended to implement the ability to calculate resistor values from an image. Although this feature is highly desirable, it has currently been poorly implemented in existing systems with these applications providing inaccurate or undesired outcomes for the user.

The Client's Thoughts on Existing Systems

In addition to the interview, I conducted further research on the client which allowed me to gather their opinion on two separate types of resistor calculator apps. The chosen applications were "Resistor Scanner" (Google Play Store, Resistor Scanner, 2021) and "Resistor Calculator" (Google Play Store, Resistance Calculator, 2021) (application 2 and application 3 that can be seen in the previous section).

I asked the client to state their thoughts while they used the application and below, I have documented both the positive and negative points that the client found.

If the thoughts of the client were like that of the reviews of the existing applications, I could ensure the application was

Positives:

- UI is clear, simplistic, and straight to the point.
- App contained other features such as capacitor value calculator – which were a nice bonus

Negatives:

- Some buttons were too small to press easily.
- The app could benefit from some sort of tutorial so that the is aware of the application's features.

Screenshots and context:

- Figure 12 and figure 13 are screenshots of the Resistor Calculator application that help illustrate the points made by the client.
- The client's complaint about the resistor type buttons can be seen below at the bottom of figure 12.
- The client's compliment about having multiple modes can be seen below in figure 13.

Figure 12 - Application 3 resistor calculator screen

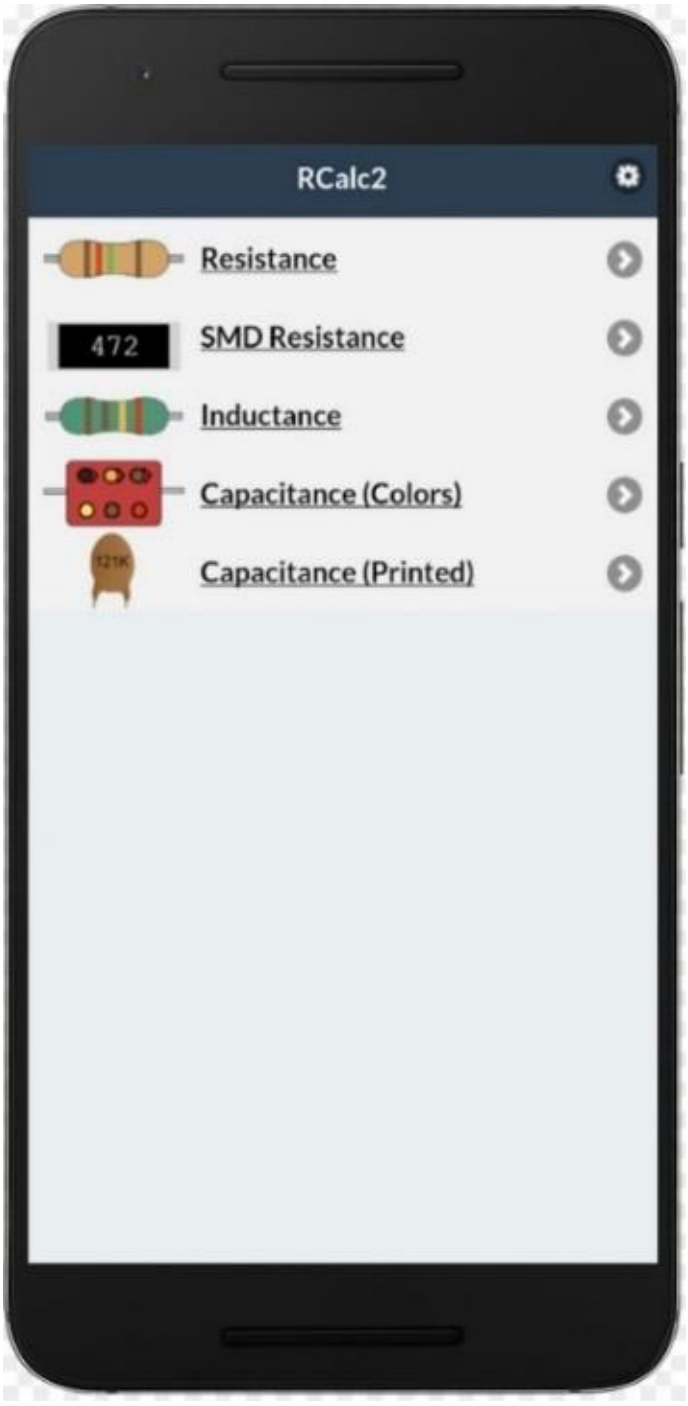


Figure 13- Application 3 mode select screen

Resistor scanner (Google Play Store, Resistor Scanner, 2021)

Positives:

- Good concept which has potential if the output was more immediate and accurate.
- Comes with tutorial.

Negatives:

- UI is cluttered and unclear – not as good as “Resistance Calculator”.
- Using and navigating the app was difficult, even after looking at the tutorial.
- Inaccurate output, even in optimal conditions.

Screenshots and context:

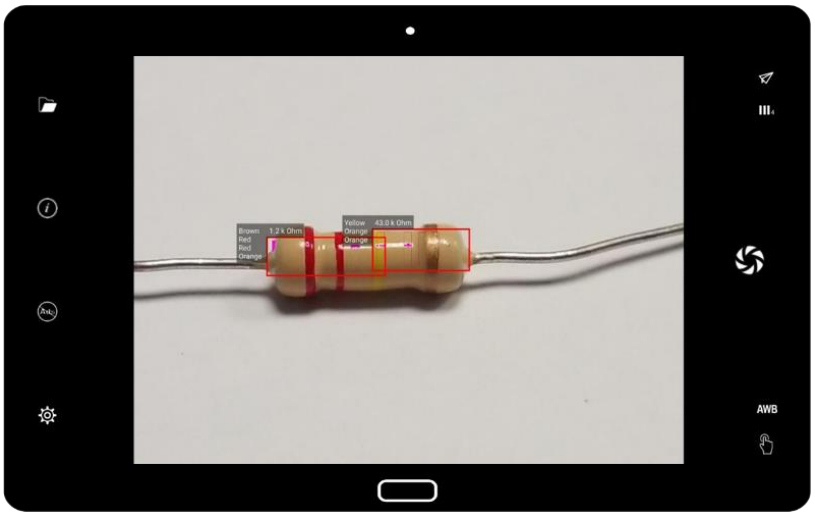


Figure 14

Figure 14 is a best-case scenario for the resistor scanner app. Resistor bands are clear, the resistor is on white background and the picture has been taken under white light meaning the colours are accurate.

Some correct colours have been identified, however the application has incorrectly identified the band locations and that are 2 resistors in the image.

It appears that the reflection caused by the light has caused the position of the resistor bands to be incorrectly identified and 2 resistors have been identified.

The orange being incorrectly identified is due to the application failing to distinguish between the bands and the resistor background colour.

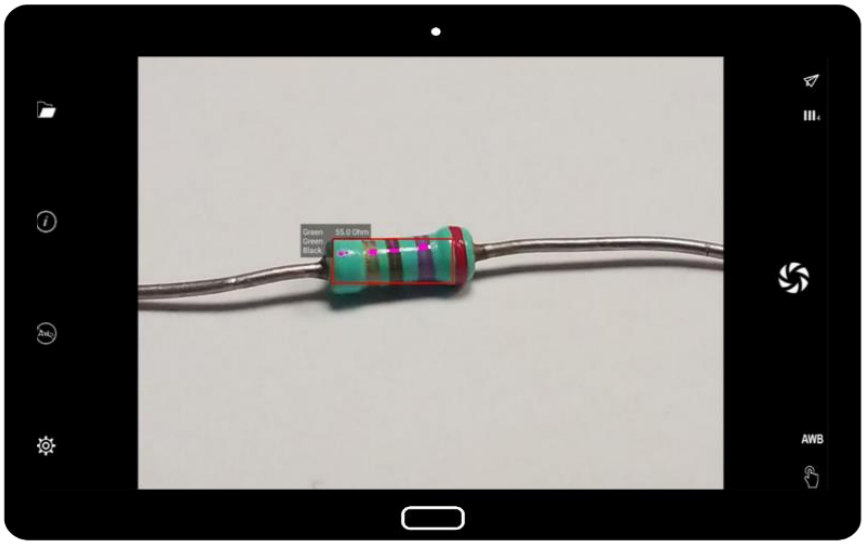


Figure 15

Figure 15 is a difficult scenario for the resistor scanner app. Although, conditions for the resistor are optimal – as described in the previous screenshot analysis, the colour of the resistor bands have faded into the resistor background colour.

Once again, the reflection due to the light has caused incorrect band location detection, with only 3 of the 4 bands being recognized.

Observations

The client's overall opinion on each app supported that of the reviews. The "Resistor Scanner" (Google Play Store, Resistor Scanner, 2021) app was summarized by the client as *"Unresponsive, cluttered and inaccurate."* while the "Resistor Calculator" (Google Play Store, Resistance Calculator, 2021) was summarized by the client as "*Precise, clean, and smart.".*

I will ensure to address the flaws identified by the reviews of the resistor scanner app and my client.

For the processing, I will need to ensure that my application can:

- Correctly identify the location of a resistor
- Correctly distinguish between resistor bands and the resistor background colour.
- Correctly locate and identify the resistor band colours.

For the GUI, I will need to ensure my application is:

- Easy to use.
- Has sufficient features to assist the user in producing the most accurate output possible.
- Resembles that of the "Resistor Calculator".

Image Processing

This problem will require me to process an image of a resistor to identify the colour of its bands. I did some research into the areas of image processing I think are needed for my project.

Image Processing Techniques

The two areas of image processing think will be needed for my resistor band detector are:

- Detecting the location of the resistor and its bands.
- Detecting the colour of the bands.

These are described in more detail below.

Boundary and Object Detection

One of the earliest stages of visual processing is where the human brain undergoes a computationally complex process to recognize an object's edges or boundaries. Identifying an object's edges reduces the amount of data that needs to be processed by the brain as the necessary structural elements of the image can be preserved while irrelevant information can be filtered (Encyclopedia of Clinical Neuropsychology, 2021).

In the system I create, I will need to recognize the boundaries of the resistor within the image so I can identify its location. Not only will this reduce the amount of processing that the image needs to undergo, but it will also ensure correct part of the image is processed, and that accurate results are returned.

Colour Processing

The brain possesses the ability to identify colours under different lighting conditions. If there was a yellow light on a red object, it can still be identified by the brain as red. However, to a computer, the yellow light would change the colour of an object causing inaccurate recognition.

To compensate for this, I could ensure that the user uses the flash and takes a photo of the resistor in white light (at daytime or under a powerful white lamp). By doing this, the colours will be normalized and accurate colour detection can take place (Your brain is lying to you - colour is all in your head, 2021).

I have also done some research into different colour spaces, that allow me to carry out more accurate colour definitions/comparisons. This is highlighted later in the Colour Spaces section.

Machine Learning

Advances in computer vision and machine learning mean that there are many algorithms available that can decipher the contents of an image or identify features within data. These algorithms can be applied to images to identify the position of objects after being trained with samples of images or to identify features of data.

In the system I can think of multiple uses for where machine learning could be implemented. One of these uses is to locate the resistor and doing this will ensure that the band-location can then be

done on the correct part of the image – giving the application higher accuracy. Machine learning could also be used to intelligently differentiate identify resistor bands within the image.

I have done some research into different possible candidates for the machine learning within my project.

K-Nearest Neighbours

K-nearest Neighbours is a supervised, machine learning algorithm which is trained with labelled input data so that it can learn and produce an output for new unlabeled data.

The way the K-Nearest Neighbours works is that it assumes that similar things are near to each other. The KNN algorithm can capture this idea of similarity using mathematics by capturing the distance between points. However, this algorithm can also be finer tuned to account for other factors such as the weighting of certain features. (Machine Learning Basics with the K-Nearest Neighbors Algorithm, 2021)

Supervised machine learning algorithms such as KNN are generally used to solve classification or regression problems. This makes it unsuitable to implement within my system as I need to locate the object rather than classify it.

The algorithm

Firstly, calculate the distance between the new data point and all the surrounding pre-labelled data points. This is done using the typical Euclidean distance formula:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Next, after computing all the distance find k number of closest neighbours.

After a k number of closest neighbours have been identified, vote for labels using majority voting. The best match is then returned as the label for the data point.

Limitations of the KNN algorithm

The KNN algorithm works best with data that has a lower number of features. This is because that when there are a high number of features, you can encounter problems such as overfitting.

Increasing the dimensionality can also lead to this problem of overfitting. To avoid this, the amount of data will need to increase exponentially based on the number of dimensions. This problem of increasing the dimensionality is known as the “Curse of Dimensionality”.

“Principal component analysis” can be done before applying on any data to determine whether it is suitable to use the KNN algorithm on data. If the data is not suited, you must find another algorithm, as crucial parts of the KNN algorithm such as calculating Euclidean distance become meaningless for the data.

Deciding the number of neighbours (K) in KNN

Since there is no universal optimal value for K for all algorithms, K must be determined for the data.

Firstly, the magnitude of K must be determined. Smaller values of K will mean that the result is influenced more by “noise” in the data while larger values of K make the algorithm much more

computationally taxing. In addition to this, the outputs when small values of K are used can be described as having low bias but high variance while the output when using many neighbours will have much lower variance but a higher bias.

Next, you must choose whether K is even or odd. Generally, it is good to choose an odd value of K if the number of classes is even.

A method such as the Elbow method can be used to determine the best value of K after trying multiple values. (KNN Classification, 2021)

Cascade Classifier

Cascade classifiers are a supervised machine learning algorithm that is trained from many positive and negative images. Training the system produces an xml file that contains a lot of feature sets that then allows it to detect trained objects in other images.

Cascade classifiers work using the concept of Cascade of Classifiers where instead of applying the vast number of features on a window, the features are instead grouped into different stages of classifiers and applied one-by-one. Windows that fail the first stage are discarded and the remaining features on it are not considered. Windows that pass ill then apply the second stage of features and continue the process. A window which passes all stages is the region that contains a desired object. (opencv-python, 2021)

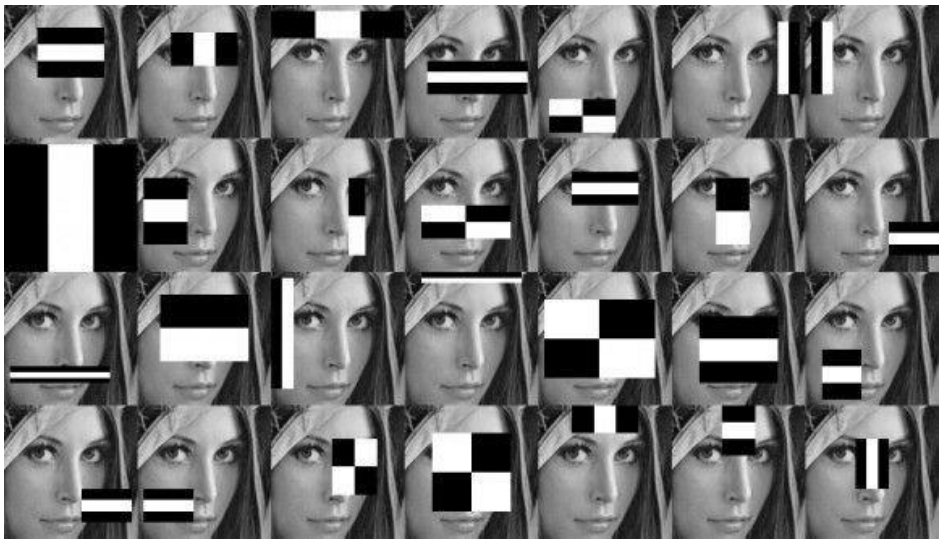


Figure 16 - Representation of Training a Haar Classifier (Cascade classifier example, 2021)

Implementing a cascade classifier into my system would prove useful in initially locating the resistor. I think the similarities in the shape of the resistor body will allow me to successfully train a cascade classifier.

K-means

Summary

K-means is an unsupervised machine learning algorithm that splits data into clusters that contain certain similarities.

Each cluster is determined by related each point of data with the centroid with the nearest mean, resulting in data known as Voronoi cells.

Clusters

Take the points:

[5, 6], [7, 5], [4, 5], [20, 30], [21, 29], [19, 28], [10, 25], [13, 24], [12, 20], [11, 23], [17, 4], [18, 5], [17, 6]

Plotting it on a graph gives:

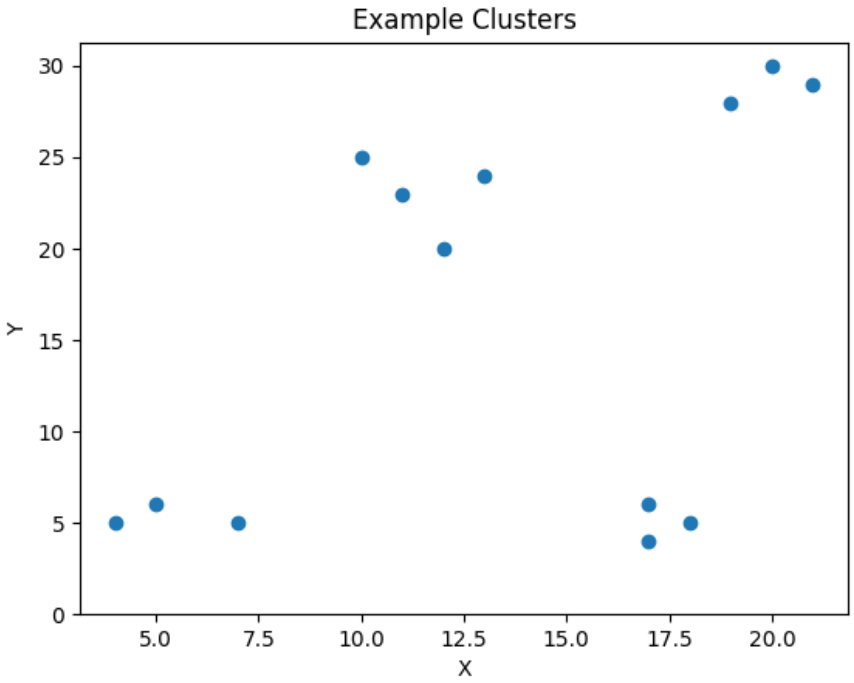


Figure 17 – Plotted points

As we can see from graph in figure 17, the data falls into 4 distinct clusters.

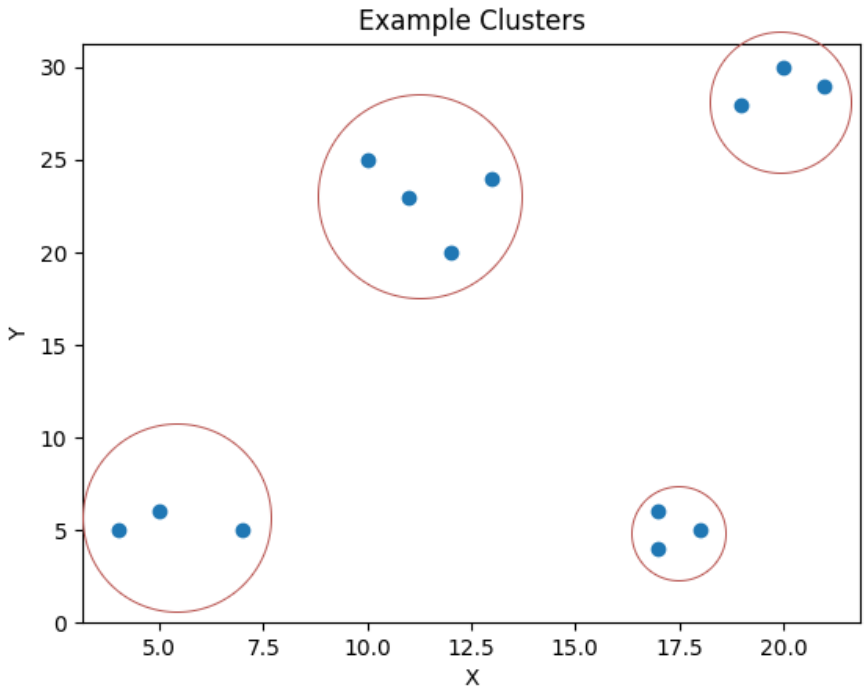


Figure 18 – Plotted points with clusters circled

Inter-cluster and Intra-cluster Distance

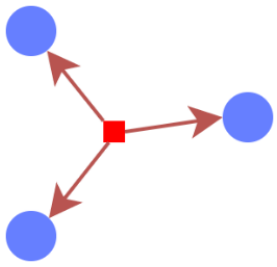


Figure 19 - Intra-cluster distance

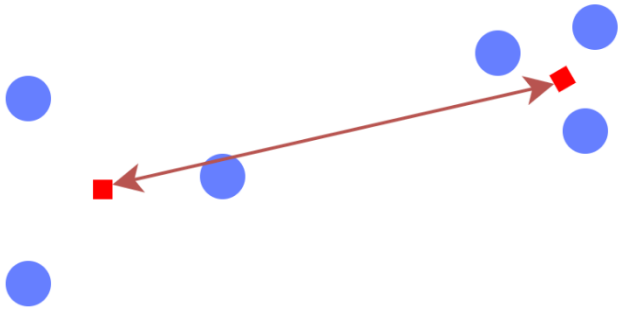


Figure 20 - Inter-cluster distance

The red arrows in figure 19 highlight the intra-cluster distance between the defined centroid (red square) and the points (blue circles).

The red arrow in figure 20 highlights the inter-cluster distance between two centroids (red squares) in different clusters.

Both the intra-cluster and inter-cluster distances are calculated using the Euclidean distance formula:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Inertia

After calculating the Euclidean distance from each point to the centroid, the inertia can be calculated.

Inertia is the sum of all the inter-cluster distances between the centroid and the points.

For a good cluster, the centroid must be adjusted so the inertia is as low as possible. At this lowest inertia value, the centroid is at the optimum distance away from all the points in a cluster.

Dunn Index

Calculating the inertia for a cluster does not factor in the fact that different clusters must be as different from each other as possible.

The Dunn Index considers the distance between two clusters as well as the distance between a cluster's centroid and points.

The formula for the Dunn index:

$$Dunn\ Index = \frac{minimum\ inter\ cluster\ distance}{maximum\ intra\ cluster\ distance}$$

The Dunn Index is the ratio of the minimum inter cluster distance to the maximum intra cluster distance.

To achieve the best possible clusters, you should maximize the Dunn index.

This largest Dunn index value is achieved by making the numerator as large as possible, and the denominator as small as possible. This means the clusters will be compact and far apart.

By calculating the clusters for a variety of values for K, you can then find the optimal value that gives you the best Dunn Index value.

K-Means++

K-means++ is an algorithm that works out the optimal initial cluster centers before moving forward with the standard k-means clustering algorithm.

Rather than randomly picking all the cluster centroids, only the first cluster centroid is chosen at random from the data points.

For each data point compute the distance from the nearest, previously chosen centroid(s).

Select the next centroid from the data points such that the probability of choosing a point as centroid is directly proportional to its distance from the nearest, previously chosen centroid. This means that the point that has the maximum distance from the nearest cluster centroid is likely to be selected as the next cluster centroid.

The algorithm

1. Choose the number of clusters that best represents the data. This can be determined through methods such as the “Elbow method” or by calculating the Dunn Index for various values of K.
2. Next, compute seeds that the initial iteration of the K-means algorithm will work off. The original K-means algorithm determines these seeds by randomly selecting several data points equal to the defined number of clusters. However, using this method can lead to errors in the final centroids as the chosen seeds may be within the same cluster. Instead, an algorithm such as K-means ++ can be used.
3. All the data points can then be assigned to the closest cluster centroid.
4. After assigning all the data points to their closest cluster centroid, the centroids can be recomputed.

K-means can be iterative or recursive meaning that the program uses the previous output to keep iterating through the steps of assigning all the data points to their closest cluster centroid and recomputing the centroid based on the assigned data points, until it reaches certain criteria. The criteria can be:

- Centroids of newly formed clusters do not change.
- Data points remain in the same cluster.
- A defined maximum number of iterations is reached.

There is no point of iterating through the algorithm more if the centroids of newly formed clusters are not changing as it is a sign that the algorithm is not learning any new pattern.

If the points all remain in the same cluster after multiple iteration, it is a sign that they have all been assigned correctly, and therefore you can stop iterating through.

You could also just set a limit on the number of iterations that can occur in the first place to ensure that the algorithm does not run for too long.

Colour Processing and Classification

There are many colour spaces available that can be used to represent different colours. The standard for colour spaces is RGB, where a red, green, and blue value are all assigned a value out of 255 that represents their strength. Despite this, I have decided to research other colour spaces as image processing problems tend to stray away from these standards such as RGB.

Colour Spaces

RGB

RGB (or BGR) is the industry standard for colour representation. It uses 3 values, each representing the amount of red, green, or blue a colour has.

Within RGB, a pixel with a value of 255, 0, 0 is the reddest colour you can get as it has the maximum amount of red and no green or blue.

The issue with classifying RGB values is when you realize that just because the values of the colour are close to another colour – it does not mean that they are close to the human eye. This means that it is hard to find continuous ranges for a colour to define it.

HSV

In image processing, colour spaces such as HSV tend to be used over standards such as RGB (or BGR).

This is because HSV separates luma (image intensity) from chroma (the colour information). When processing images, a range of hue and saturation values can define the colour you want, while the luma can be defined separately to allow for a tolerance for a colour's darkness.

This means that a continuous range for a HSV value can be used to define the colour within an image.

Conclusion

I am therefore choosing to use the HSV colour space for my colour detection. This is due to its ability to separate a colour's luma from the chroma and its ability to define a continuous range that defines a colour.

Sort Algorithms

It is definite that I will need to sort lists within my project and shall write my own algorithms to carry out this operation.

Merge Sort

Algorithm

Merge sort is a moderately efficient sorting algorithm that makes use of a technique known as divide and conquer. This is highlighted within the image below, where the initial input list is split up until it is singular data items.

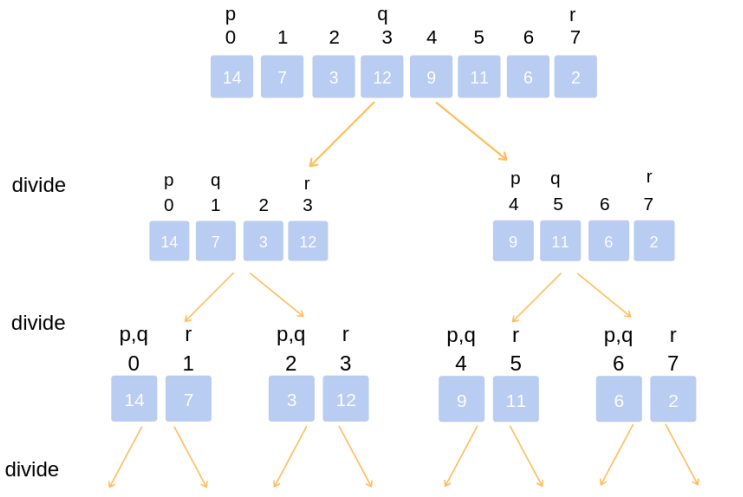


Figure 21 - Splitting up the data for merge sort (Merge Sort Algorithm, 2021)

After the list has been split up into singular data items, pairs of elements are then compared, sorted, and merged. This process is repeated until the full list has been recompiled. This can be seen within the image below.

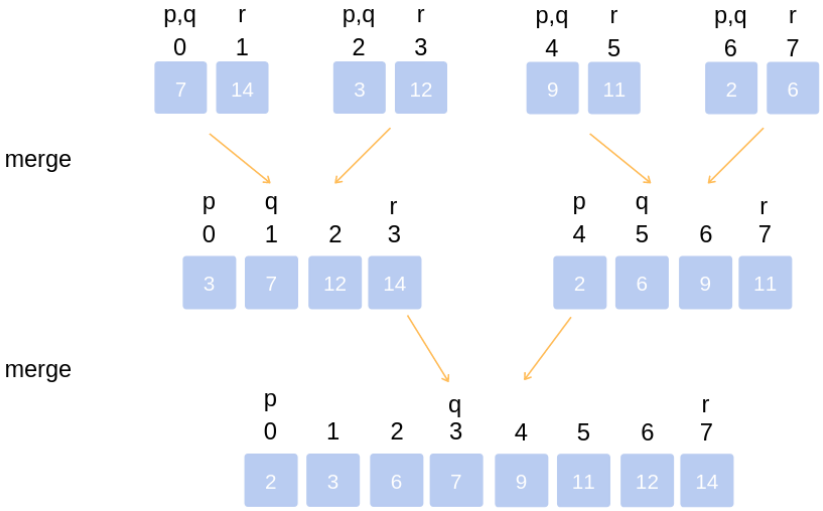


Figure 22 - Sorting and merging pairs of lists (Merge Sort Algorithm, 2021)

Pseudocode

Can be seen in the design section.

Cases for Merge Sort

Assuming that the input list is longer than 1 in size, in big O notation, the time complexity for merge sort to run is O(n*log(n)) in all cases due to its divide and conquer nature.

Bubble Sort

Algorithm

Bubble sort is a simple sorting algorithm that steps through a list and compares items that are next to each other. If the items are in the wrong order, the items will be swapped around. This process repeats until the list is sorted.

Cases for Bubble Sort

Worst Case

In the worst and average cases, in big O notation, $O(n^2)$ comparisons will occur, and $O(n^2)$ swaps will occur. This means that the algorithm is much less efficient than merge sort and that I will not use it within my project.

Best Case

In the best case, in big O notation, O(n) comparisons will be made and O(1) swaps will be made.

Software and Requirements

Programming Languages

I have researched various possible programming languages that could potentially be used for the application. I have written a brief description of each language in the tables below and then have justified the language I will be using.

GUI

Language	Description
HTML and CSS	HTML and CSS are declarative languages that allow for the creation of sophisticated, aesthetically pleasing UIs with much customization and integration with JavaScript. Using HTML and CSS would make the system platform independent as the system could run on any device with a browser installed. Although HTML and CSS have many advantages, integrating it with the program may prove much more difficult that using an in-built Python library such as Tkinter and require additional programming languages to be learnt.
Tkinter	Tkinter is a python library that allows for the creation of a very basic UI with a very simple implementation in my best programming language, Python. Using Tkinter would make the created system platform specific as it would need python installed to run, which is undesirable for this use case.
Android GUI	Android is an open-source Linux-based operating system that primarily runs on Android smartphones. Android GUI makes use of XML and Java files to declare screen layouts, allowing for the creation of a sophisticated, aesthetically pleasing, and easy-to-use UI (Android User Interface, 2021). Implementing Android GUI would may prove very difficult as multiple additional programming languages must be learnt.

Choice of Language for GUI

The task at hand requires me to pick a GUI that is accessible to different types of devices. The students and teachers using my system have different devices available to them and they may be running different operating systems.

The best choice of programming language to create my GUI is therefore HTML and CSS due to their platform independence and ability to create a sophisticated, aesthetically pleasing, and easy-to-use GUI. It will allow my application to be accessed from any device that can run a web-browser.

Processing

Language	Description
Python	Python is an interpreted, object-oriented, high-level programming language with access to a vast number of libraries, such as those that will help manipulate resistor images or create a webserver. It is my native language
Java (Android SDK)	Java is a high level, object-oriented, platform independent language that can be used to develop software such as Android applications (Java Summary, 2021). Java has access to libraries that allow me to manipulate resistor images. Using the Android SDK would make my system machine specific – with it only being able to run on Android devices, making it unsuitable for the task.
C++	C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming (C++ Quick Guide, 2021). Just like Java and Python, C++ has access to libraries that will help me manipulate resistor images.

Choice of Language for Processing

The task at hand will require me to detect the location of a resistor in an image as well as the position and colour of its bands. The programming language that I choose should have access to libraries that allow me to manipulate an image and to create a webserver due to the choice of HTML and CSS to create the GUI.

Python has access to a vast number of libraries such as OpenCV and Flask that will allow me to manipulate resistor images and create a webserver. This will allow me to create a platform independent application that can run on a webserver that contains the HTML and CSS GUI. In addition to this, Python is a familiar language to me – in which my A-Level studies is based. However, due to its nature of being an interpreted language, using Python may cause the application to run slower and for less efficient use of the server's memory.

Despite having access to image processing libraries such as OpenCV4Android, using Java to create an Android application is unsuitable. Developing an Android application will restrict the types of devices which my application can run on.

C++ is a compiled language meaning that runs more efficiently and quickly than an interpreted language such as Python– which may make the application seem more responsive and is the original language in which libraries such as OpenCV and TreeFrog were written. This means that applications that make use of these libraries run very efficiently. OpenCV will allow for the manipulation of the resistor image and TreeFrog allows me to create a webserver for the application. In the case of this task, the user of a webserver is essential due to the choice of HTML and CSS to create the GUI.

Although C++ is the optimal language for the application, the language I have chosen to write my application in is Python. The familiarity of Python to me will allow me to develop my application more efficiently and with less mistakes. This means the client can use my application sooner and that the final system will be more polished. The only downside of Python compared to C++ is the efficiency of the program, which is not a concern in this use case as the application will be ran on a powerful server.

Final Choices and Requirements

As I have chosen to use HTML and CSS for the GUI and Python with Flask integration for the processing and the creation of a webserver. Due to the use of a webserver, JavaScript will be needed to respond to requests between the webpage and the webserver and dynamically update elements of an HTML page when appropriate (such as when a button needs to show as selected). The table below explains my choices.

Language	Reason
Python	• Python is a familiar language – in which my A-level studies is based. • Python has access to many libraries, that allow the data to be manipulated better – leading to more accurate band and colour detection.
Flask (Python Library)	• Allows the application created to be fully device independent. • Allows the use of HTML and CSS for the user interface. • The bulk of the processing is done on the server side rather than on the end user's device. This means the application will be more responsive. • The application will not require the end users to install anything before using it.
HTML	• HTML page will be needed to format and display elements correctly. • To accept user inputs.
CSS	• To style HTML elements so that the UI can be clear and look nice.

Language	Reason
JavaScript	• To manage inputs, and processes on the HTML page. • To handle requests to and from the Flask webserver. • To manipulate the HTML page. • To perform basic functions.

Below is another table that is showing the requirements for my application to run.

Requirement	Reason
Platform	• The app will be platform independent due to the use of a webserver.
Hardware	• If a camera is available, it can be used to take a picture of the resistor. • The camera is not compulsory to calculate resistor values; the application will also allow for manual input as well as an option to upload a pre-existing image. • The app minimum specifications will be low as the bulk of the processing is done on the server side. • The device will have to be able to connect to the internet (to use the full program).
Prerequisites	• The app will require the device to have a web browser to run.

These requirements are low and will allow the application to run on a vast number of devices – which is essential in the use case of the system.

I have outlined later in the UI design section how the compatibility of older devices may affect the system's UI usability.

Client Demands

I have made a note of the client's demands as well as stating a way to achieve the objectives set by the client.

That the application is accessible on different types of devices and operating systems as the user's device will use is unknown.

Number	Demand	How I will achieve it
1	That the app calculates a resistor value for resistors with a variety of bands.	• Ensure the program can calculate the values for all resistor types. • Implement algorithms that can identify the number of bands if the user wants to find resistor values from an image. • Allow the user to declare the number of bands to the program. • I will include an option that will allow the user to choose the number of bands that the resistor has if they use manual input.
2	A clear, easy to use UI (with sensible scaling).	• I will use CSS and HTML to create the main UI and will test it on multiple devices to ensure it is working and displaying correctly.
3	That the application can locate the resistor anywhere in an image.	• Ensure the created algorithms for resistor location are versatile. Make sure it is not just tailored around the test data.
4	That the application is accessible to as many people as possible.	• By creating an application that makes use of a webserver, my application will be accessible to many device types via a browser.
5	That the program warns the user when there is an invalid output.	• Output to the user if the resistor is a standard resistor value. • Use try excepts to tell the user as to why an output is invalid (know where and why the program went wrong).

Number	Demand	How I will achieve it
6	That the application can work both online and offline.	• Do resistor value calculations on the webpage locally using JavaScript, so the HTML page can be used as a standalone application.
7	That the application is more accurate than the other applications you showed me.	• I will test my program thoroughly and ensure that it has above an 80% detection rate. • Ensure my program passes all tests.

SMART Objectives

Creating Good Objectives with User Stories

After researching how to write good objectives, I have found out that a commonly used way to define objectives is to capture them as “user stories”. (A Short Guide to Writing Software Requirements, 21)

The template for a user story is:

- *As a <user>, I want <some goal>, so that <some reason>.*

Using user stories allows good objectives to be created as it is essentially the same as asking the questions:

- Whose benefit are we doing this for?
- What is being done?
- Why are we doing is? (optional)

By using this simple template, you allow the objectives to define the problem without implying a particular implementation.

Below I have written the user stories along with a corresponding objective number. These objective numbers will correspond to tests in the testing table later in the document.

System Objectives

The table of objects contains the list of objectives I have identified from the client demands.

Each objective is linked back to the client’s demands so you can easily verify that all demands made by the user are covered by the objectives.

Objective	1
User Story	**As a user I want to be able to take a photo of a resistor and have the resistor value calculated for me.**
Demand	**1**

Objective	2
User Story	As a user I want the bands colors that have been automatically detected from the resistor image to be displayed
Demand	1

Objective	3
User Story	As a user I want to be able to override the band colors that were detected so that I can make sure the resistor calculation is accurate
Demand	1

Objective	4
User Story	As a user I want to be able to enter the resistor band colors manually (without needing an image) and have the value calculated for me.
Demand	2, 6

Objective	5
User Story	As a user I want the calculated resistor value to be displayed next to the resistor bands.
Demand	2

Objective	6
User Story	As a user I want the system to tell me when the set of bands does not correspond to a standard resistor value.
Demand	5

Objective	7
User Story	As a user I want the displayed resistor value details to include the important resistor details. 1. Resistance in Ohms 2. Tolerance in percent 3. Temperature constant in ppm/K
Demand	1, 2

Objective	8
User Story	As a user I would like to be able to use the system in an offline mode.
Demand	6

Objective	9
User Story	As a user I would like the system to work when the resistor is located anywhere in the image.
Demand	3

Objective	10
User Story	As a user I would like the system to work when the resistor is smaller or larger in the image.
Demand	3

Objective	11
User Story	As a user I want the system to be able to calculate the resistance for resistors with a different number of bands.
Demand	1

Objective	12
User Story	As a user I would like to be able detect resistors using different devices.
Demand	4

Objective	13
User Story	As a user I would like to be able detect resistors on devices with different operating systems.
Demand	4

Objective	14
User Story	As a user, I would like the application to work with all resistor colours. 1. Black 2. Brown 3. Red 4. Orange 5. Yellow 6. Green 7. Blue 8. Violet 9. Grey 10. White 11. Gold 12. Silver
Demand	1, 7

Flowchart

Diagram and Explanation

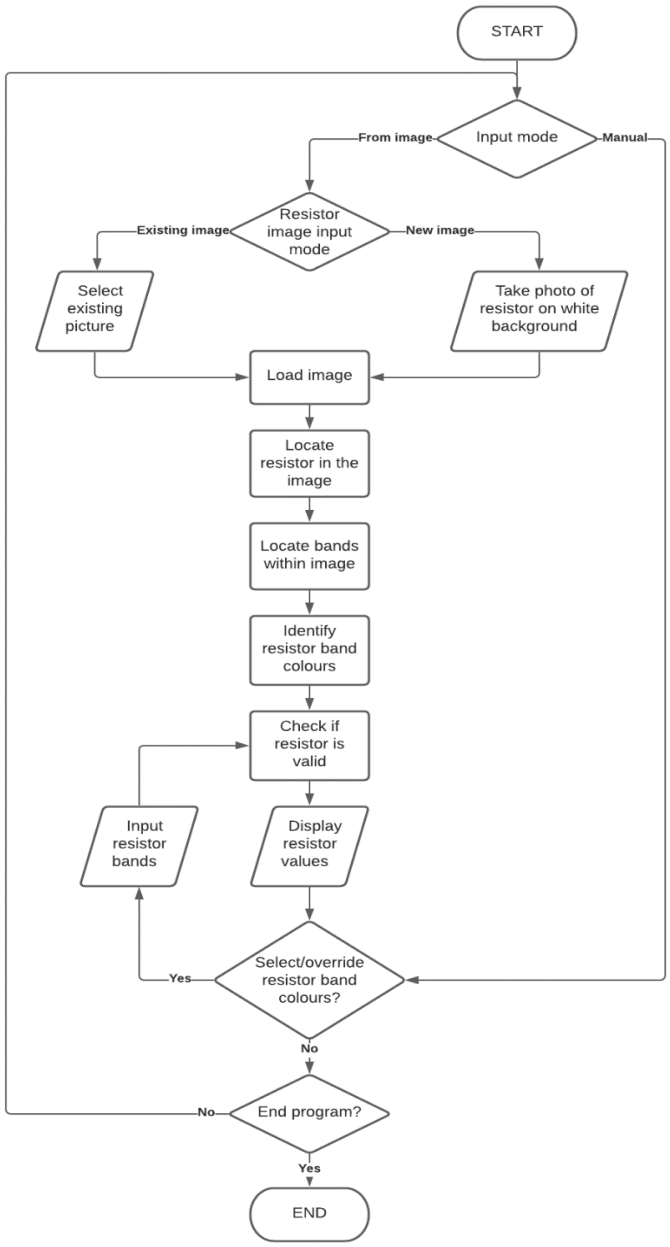


Figure 23 – High-level flow chart showing the process flow.

The diagram on the left is a very simplified overview of the system in the form of a flowchart. It shows the very basic principles for operation of the system.

This flowchart highlights the flow of processes that I expect to program within my final program.

Firstly, the user should be able to choose between manual colour input or image input.

If the user chooses to use image input, they should be able to override/correct the output of the program – this will ensure that the output of the program will always be correct and useful to the user.

The program should output whether the resistor values fall within the standard resistor values, and let the user know when the result is likely to be inaccurate/entered in wrong.

Critical Path

I have decided the split the development into stages. For each stage in development, I have created a critical path diagram representing a different stage in completion for the project.

Although I might stray away from this plan, it will ensure that I am keeping up with progress for my project.

Analysis

Critical Path Diagram and Explanation

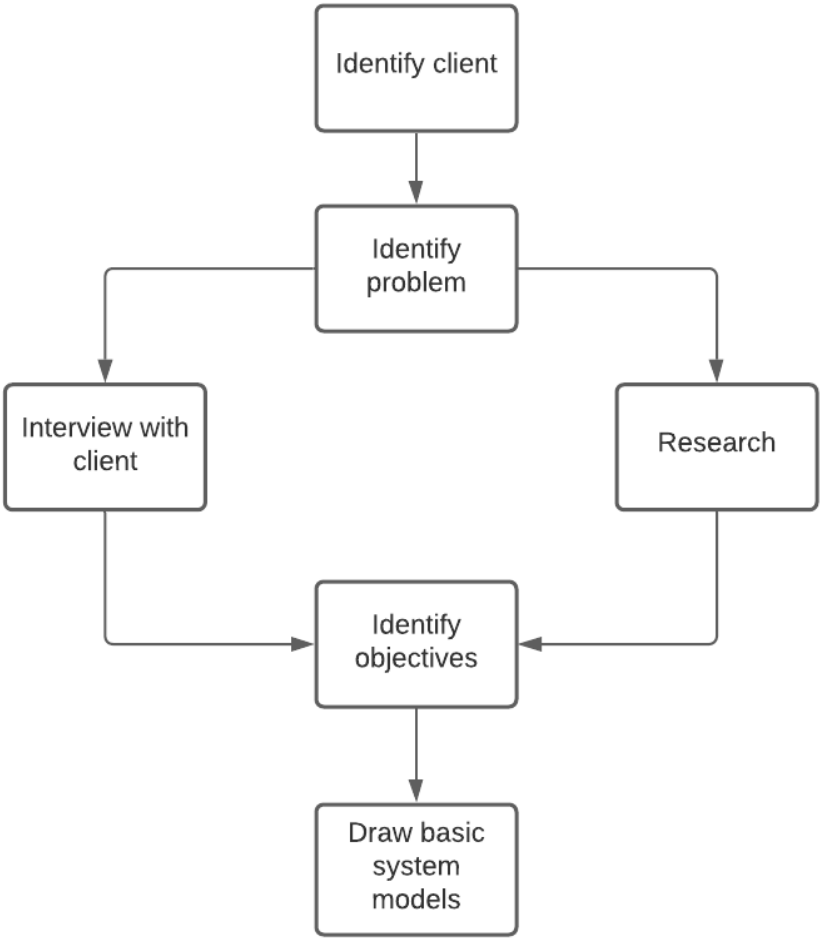


Figure 24- Critical path diagram 1

In the analysis stage of the project, I will develop a strong foundation for which the rest of the project will be developed.

I will do this by thoroughly researching my problem and ensuring that the system objectives are well established.

Doing this will ensure that I am aware of what the program's end point is to look like.

I will make sure that I gather much information from my client about what they want from the system so that I can make good objectives from. Doing this is so crucial as it will determine the basis on which the development of the project happens.

Prototype

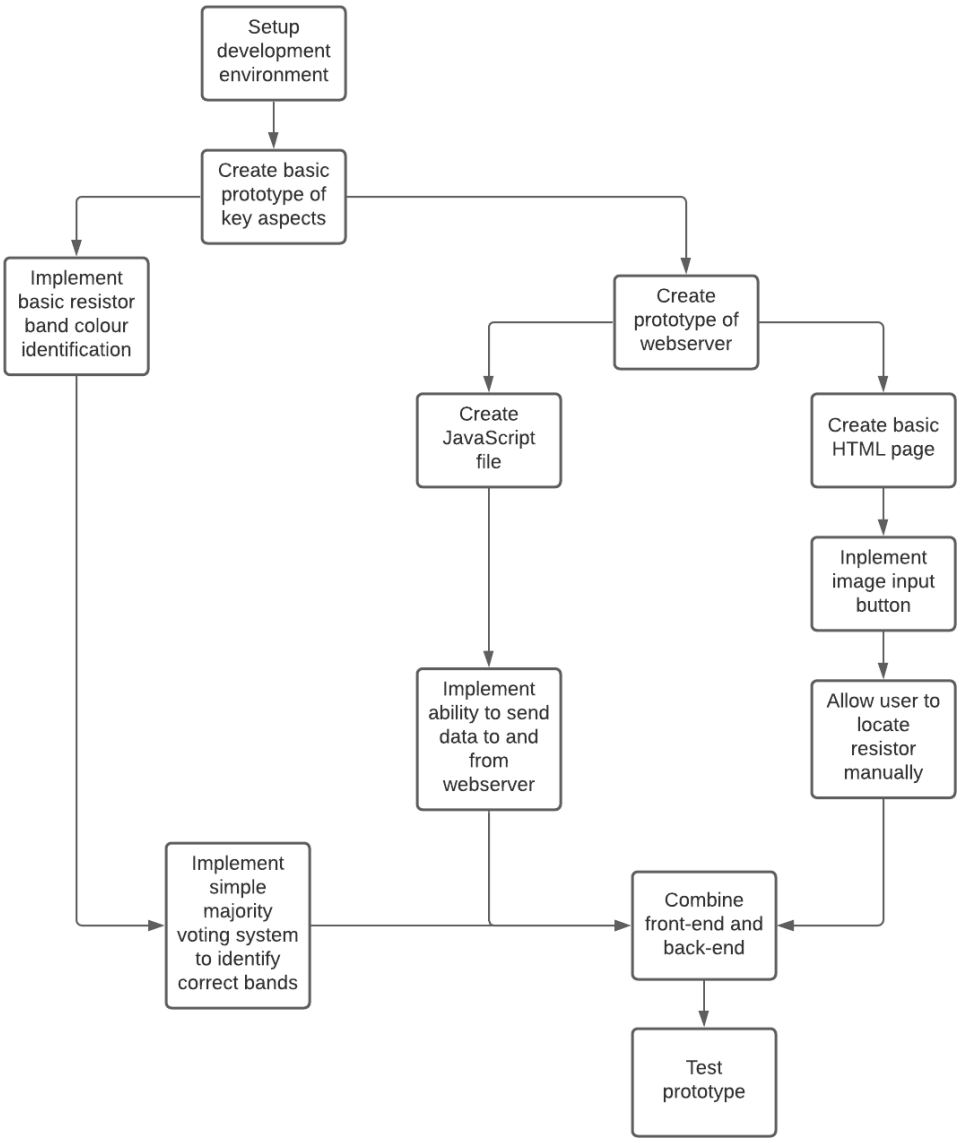


Figure 25 - Critical path diagram 2

In the prototype stage of the project, I will create a basic system that will function as a skeleton for the final program.

This prototype will do the bare minimum needed for the system to function and will not contain many features that will be present in the final system – such as resistor band colour buttons and automatic resistor location detection.

I will ensure I will test the separate components of the system individually before combining them to try and reduce the number of bugs or errors present.

Developing Final Program

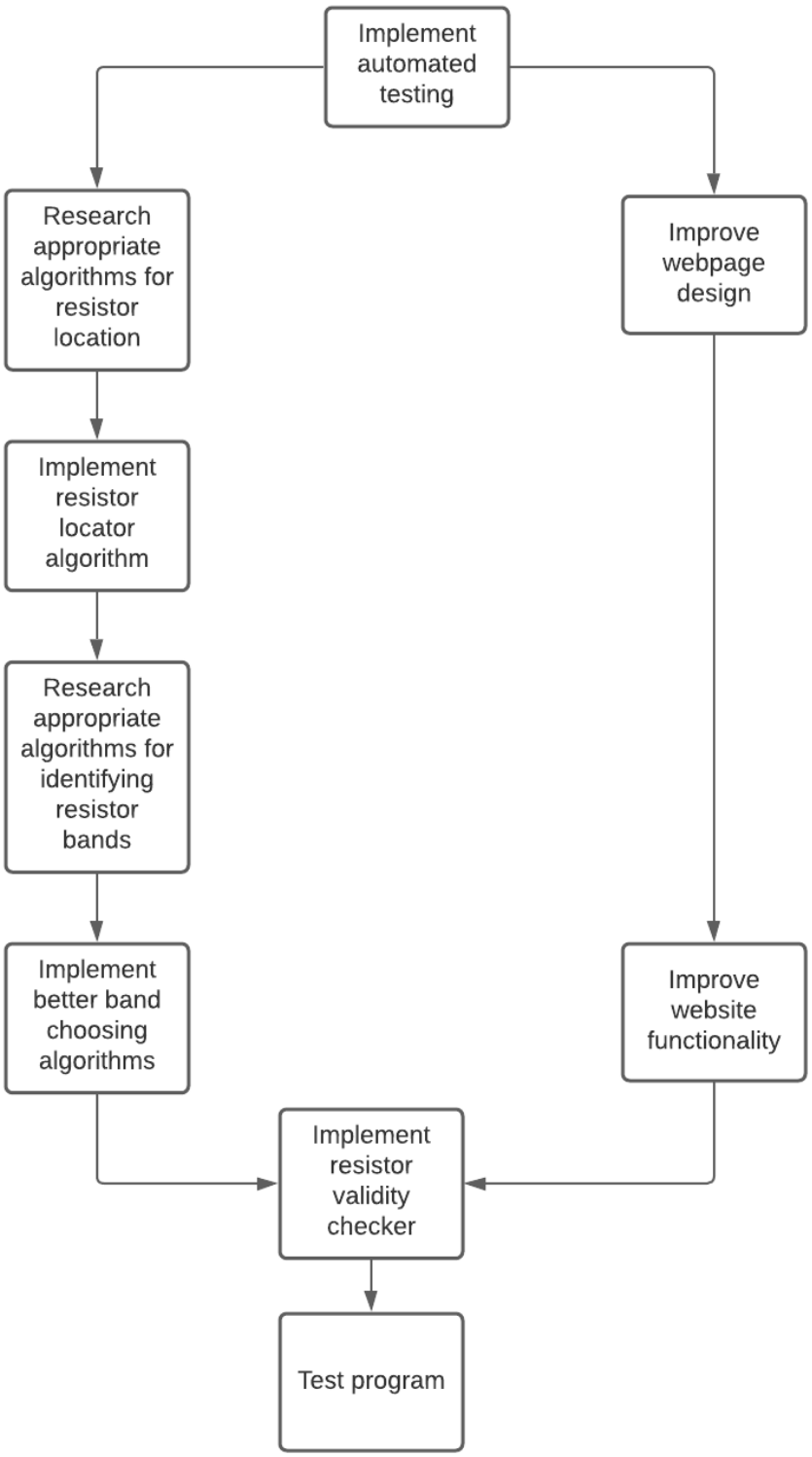


Figure 26 - Critical path diagram 3

In the final development stage of the program, I will ensure that the UI resembles the UI that is pictured in the design stage of my project and will implement automatic resistor location detection.

I will develop 2 methods of identifying a resistor's location and will identify which is better through automatic testing.

By "other algorithm", I am referring to methods such as finding a way of using image processing with open-cv to isolate and extract the resistor body.

Prototyping

Overview

My program will have 4 main components:

- Webserver.
- Resistor body location.
- Band location.
- Band colour identification.

For each of these stages, I will create a basic prototype of a couple of my ideas for my system.

Locating the Resistor Body

Haar Classifier

I tried using a Haar Classifier to locate the resistor body, however I found out that I did not have enough training data to train the Haar Classifier to work with a reasonable accuracy.

Even though it is possible to train Haar Classifiers to recognize any types of objects within an image, they are usually used for facial recognition. This is another reason why it has struggled identifying the resistor bodies within images as a resistor body is less distinct than a face.

Cropped Resistor Body Images

For the prototype stage of my program, I have decided I will manually locate the resistor body by cropping the original resistor image to be just the resistor body. I will experiment later with alternative methods of locating the resistor body within the image.

Band/Colour Location

I tried multiple approaches for identifying the band colours and locations.

Initial Approach

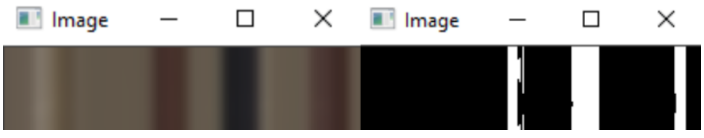


Figure 27 - Side by side comparison of a blurred BGR image and its inverted monochrome result

My initial approach involved converted a vertically blurred image to monochrome using the adaptive threshold operation within cv2.

From the image above, we can observe that converting the image to an inverted monochrome image correctly converted the bands to distinct white rectangles for the brown and black bands. However, gold proved to be problematic, not being represented at all in the monochrome image.

Although only gold proved to be problematic in this case, there are more problematic colours: yellow, orange, and white. The reason these colours are problematic is due to the defined monochrome thresholds that I set.

This is an issue that cannot be avoided as the purpose of the monochrome threshold was to convert only the resistor background colour to black while leaving the bands as white. However, since gold, orange and yellow are very close to the background colour, they are also converted to black. On top of this white is converted to black due to its nature of being a very light colour.

For the bands which did show up on the inverted monochrome image, I could take their contours, find the bounding rectangle that encloses those contours. With this bounding rectangle, I could then find the average colour within the section of the image that falls within the bounding rectangle and look up that average colour against known values to obtain the colour for a band.

Limitations

A limitation of this colour look-up method that I quickly discovered was that it was very difficult to compare my obtained value to a BGR value. After researching colour spaces, I discovered that I could easily check for a colour by converting the BGR colour value fell within an HSV range.

After finishing this method of colour detection, I realized that I could very easily improve the colour detection if I skipped the step of converting to monochrome, as it caused me to lose data and accuracy. In my next approach, I explain it.

Colour Masking

It wasn't necessary for me to know where the bands were before identifying their colour. Instead, I could just mask the certain HSV ranges that a certain colour is expected to lie in and then take the contours and bounding box of that mask. This means I would obtain the colour and the location of a band at the same time.

By masking the certain HSV ranges that represented a colour, I could detect whether that colour was present. However, this information would not suffice as the resistor colour band order matters greatly in determining its value. By taking the bounding rectangle of the colour mask for a certain colour, I could then order the resistor by using merge sort on the bounding box x coordinates.

If all the colours were accurate to their real colours, it would be very easy to define the HSV ranges that need to be masked for a colour, however, due to factors such as glare and the inaccuracy of resistor band colours in the first place, I had to create my own HSV ranges for each colour rather than just searching up the HSV colour definitions for colours online.

I discovered these HSV ranges by implementing a tool within my program that allowed me to click on any pixel of the image and obtain the HSV value for that pixel. I added some leeway to accommodate for slightly differing conditions.

However, this leeway and the inaccuracy with the colours in the first place meant that the colour ranges of certain colours overlapped. Examples of similar problematic colours are brown, orange and gold.

In addition to these similar problematic colours, the H values for white, black, and grey do not fall within specific ranges as they can span from 0-255. This means I will have to think of a way to compensate for this wide range and ensure it does not override other bands that are detected within the same position.

Obtaining the HSV Ranges

These HSV values are in the cv2 format meaning that the H range is from 0 to 180, and the S and V ranges are from 0 to 255. The data from clicking through all the colours on my images can be seen below.

Colour	Range
Black	[[0, 0, 7], [120, 36, 7], [120, 36, 7], [0, 0, 7], [9, 81, 22], [15, 73, 7], [0, 0, 5], [90, 51, 5], [90, 51, 5], [111, 98, 26], [113, 104, 22], [117, 135, 17], [117, 135, 17], [113, 128, 18], [113, 128, 18], [116, 120, 17], [158, 34, 30], [158, 38, 27], [140, 45, 17], [150, 36, 14], [160, 55, 14], [150, 20, 13], [0, 0, 7], [0, 0, 6], [120, 43, 6], [0, 0, 5], [0, 0, 5], [0, 0, 6], [75, 85, 6], [75, 85, 6], [90, 51, 5], [90, 43, 6], [120, 43, 6], [0, 13, 19], [0, 0, 9], [120, 36, 7], [0, 0, 7], [120, 36, 7]]
Brown	[[6, 170, 30], [6, 170, 30], [6, 173, 28], [5, 177, 26], [5, 186, 26], [4, 173, 25], [5, 191, 24], [176, 105, 34], [176, 109, 35], [0, 117, 37], [176, 105, 39], [0, 114, 38], [178, 83, 40], [5, 148, 55], [6, 137, 54], [7, 142, 54], [7, 137, 54], [8, 139, 55], [8, 137, 56], [8, 137, 56], [9, 114, 47], [8, 125, 47], [9, 136, 45], [9, 161, 46], [9, 172, 43], [6, 154, 48], [7, 156, 49], [8, 159, 48], [7, 154, 38], [6, 163, 39], [7, 170, 39], [6, 174, 38], [7, 174, 38], [6, 185, 33], [7, 188, 34], [0, 121, 40], [4, 137, 41], [6, 157, 39], [7, 168, 38], [7, 168, 38], [7, 177, 36], [7, 174, 38]]
Red	[[4, 163, 50], [5, 163, 50], [5, 167, 49], [4, 163, 50], [4, 157, 52], [5, 165, 51], [5, 165, 51], [4, 167, 52], [5, 167, 52], [177, 147, 78], [177, 149, 77], [178, 163, 78], [177, 182, 80], [177, 181, 83], [177, 178, 83], [177, 179, 84], [177, 161, 87], [177, 171, 88], [178, 181, 90], [178, 182, 91], [176, 157, 83], [176, 179, 84], [176, 181, 86], [177, 192, 61], [179, 199, 64], [179, 194, 67], [179, 194, 67], [178, 162, 82], [178, 166, 86], [178, 158, 89], [178, 157, 91], [179, 119, 79], [179, 143, 82], [179, 141, 87], [178, 147, 87], [178, 192, 61], [179, 203, 59], [178, 207, 59], [179, 188, 57], [178, 210, 63], [179, 202, 58], [178, 211, 58], [179, 208, 60], [179, 207, 53], [179, 207, 53], [179, 212, 65], [179, 210, 62], [178, 213, 61], [178, 207, 75], [177, 201, 75], [177, 209, 72], [178, 208, 71]]
Orange	[[11, 194, 80], [11, 197, 83], [11, 202, 87], [11, 205, 92], [11, 203, 94], [11, 201, 95], [12, 203, 94], [12, 156, 93], [11, 177, 98], [10, 175, 102], [10, 174, 104], [9, 166, 106], [10, 181, 110], [10, 186, 114], [10, 168, 117], [10, 165, 113], [11, 171, 112], [11, 172, 108], [11, 164, 106], [7, 193, 86], [9, 189, 81], [8, 201, 81], [8, 206, 78], [8, 207, 74], [8, 207, 74], [9, 217, 87], [8, 213, 85], [8, 217, 81], [8, 220, 80], [8, 200, 116], [8, 206, 114], [8, 211, 111], [8, 210, 108], [11, 177, 95], [11, 185, 98], [12, 194, 105], [11, 178, 106]]
Yellow	[[24, 190, 121], [24, 193, 119], [24, 194, 117], [25, 194, 113], [24, 195, 114], [23, 179, 118], [24, 193, 116], [24, 192, 114], [23, 170, 108], [24, 191, 108], [24, 194, 112], [25, 170, 123], [24, 169, 122], [24, 170, 120], [24, 171, 119], [25, 170, 114], [25, 172, 111], [25, 174, 107], [24, 146, 101], [24, 162, 112], [23, 167, 122], [24, 165, 124], [26, 163, 111], [26, 165, 113], [26, 171, 122], [25, 169, 127], [24, 166, 129], [24, 166, 115], [24, 164, 112], [25, 167, 104], [25, 167, 98], [25, 163, 94], [24, 169, 119], [23, 161, 108], [24, 168, 105]]

Colour	Range
Green	[[52, 111, 62], [49, 110, 65], [49, 112, 66], [47, 84, 61], [44, 86, 62], [52, 119, 62], [51, 125, 63], [49, 111, 69], [54, 107, 57], [67, 150, 63], [67, 138, 70], [67, 147, 71], [67, 131, 74], [67, 145, 72], [67, 131, 74]]
Blue	[[106, 170, 66], [105, 163, 64], [104, 200, 65], [104, 202, 63], [104, 200, 65], [104, 205, 66], [104, 207, 64], [104, 199, 68], [107, 223, 71], [107, 208, 71], [107, 227, 72], [107, 227, 73], [107, 228, 75], [107, 203, 74], [107, 218, 75], [107, 197, 75], [107, 208, 76]]
Violet	[[141, 125, 43], [142, 113, 43], [143, 119, 43], [144, 118, 41], [143, 124, 41], [143, 128, 40], [143, 131, 39], [155, 65, 47], [155, 71, 47], [146, 95, 43], [143, 107, 43], [141, 101, 43], [143, 100, 46], [143, 94, 49], [145, 90, 48]]
Grey	[[60, 4, 69], [30, 15, 69], [90, 4, 68], [60, 4, 69], [60, 4, 69], [18, 18, 72], [150, 4, 68], [0, 0, 68], [120, 4, 70], [0, 4, 70], [0, 0, 71], [0, 4, 72], [130, 13, 58], [135, 9, 58], [130, 13, 58], [130, 13, 57], [135, 18, 57], [128, 17, 59], [120, 9, 59], [128, 17, 60], [120, 13, 60]]
White	[[9, 27, 131], [9, 27, 131], [9, 27, 130], [9, 26, 127], [9, 27, 125], [12, 27, 124], [11, 24, 117], [10, 26, 116], [10, 27, 115], [13, 27, 114], [11, 25, 112], [12, 27, 123]]
Gold	[[17, 110, 79], [16, 115, 73], [17, 113, 68], [17, 121, 61], [16, 117, 59], [16, 121, 55], [16, 123, 52], [17, 126, 73], [17, 123, 52], [18, 124, 70], [18, 130, 57], [18, 132, 56], [16, 133, 50], [17, 128, 50], [17, 128, 50], [14, 117, 37], [13, 112, 41], [14, 113, 43], [15, 109, 47], [16, 109, 49], [15, 109, 56], [15, 109, 61], [15, 112, 64], [15, 102, 75], [14, 98, 73], [15, 105, 75], [15, 108, 73], [15, 92, 78], [18, 95, 99], [18, 93, 126], [21, 90, 142], [19, 88, 101], [17, 95, 91], [15, 96, 82], [16, 93, 71], [17, 95, 67], [16, 93, 66], [16, 93, 66], [13, 117, 35], [13, 113, 36], [15, 112, 80], [15, 113, 111], [15, 113, 68], [17, 94, 68], [16, 86, 71], [16, 96, 56], [17, 89, 57], [17, 96, 53], [13, 72, 57], [13, 85, 57], [14, 89, 69], [14, 84, 79], [13, 86, 86], [18, 90, 68], [19, 90, 68], [19, 89, 69], [19, 78, 78], [19, 87, 79], [18, 82, 106], [18, 83, 77], [19, 81, 76], [18, 83, 80], [20, 66, 93], [18, 82, 90], [12, 117, 48], [13, 105, 39], [13, 113, 36], [18, 125, 57], [17, 123, 56], [21, 109, 115], [20, 125, 110], [16, 128, 48], [15, 130, 61], [13, 142, 43], [16, 124, 99], [16, 107, 95], [17, 108, 85], [15, 139, 33], [17, 126, 83], [16, 133, 71], [15, 124, 66], [17, 117, 50], [20, 102, 50], [18, 127, 56], [17, 95, 70], [15, 95, 59], [15, 88, 58]]
Silver	No data (could not find a resistor with a silver band).

I then determined sensible HSV ranges from this data for all the colours apart from, white, black, and grey.

Since white, black and grey do not have definite H ranges, I have decided to mask the full 0 – 180 range for the H ranges of these colours. As mentioned earlier, I will have to think of an algorithm that will allow me to ensure that these lenient colours do not override other colours that are detected in the same position.

Limitations

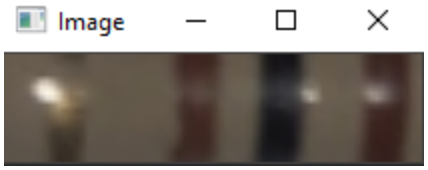


Figure 28 - Resistor body closeup

Glare was a big issue when masking the colours within the resistor. Random white blobs would be detected as bands and obscure other HSV colour masks. These random white blobs would then be included in the results for the resistor bands, which greatly decrease the detection's accuracy and the other bands that are detected are less easily identifiable as distinct bands.

In addition to this, I must develop a way to identify the best match for each band at each position as some HSV ranges may overlap and cause conflicting colours at each band position.

Addressing Glare

Initial Approach

Within this prototype, I have combatted glare using a strong vertical blur.

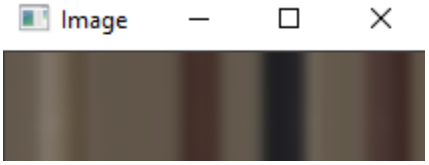


Figure 29 - Strongly vertically blurred image of resistor body

In the image above, you can see that blurring the image both makes the bands more rectangular as well as removing the glare blobs at the cost of lightening the columns which the glare was in.

The colour masks could then be applied to this blurred image, and the result would not contain white blobs.

Limitations

However, there are issues with this approach:

- The gold band, which is common on resistors, becomes almost indistinguishable from some resistor backgrounds.
- Blurring the glare in some places can create "ghost bands" which are the result of the glare being too bright.

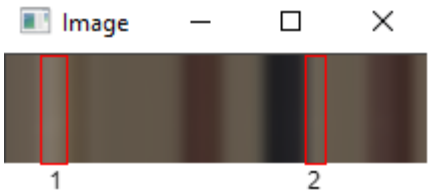


Figure 30 - Strongly vertically blurred image of resistor body with possibly problematic "ghost bands" labelled.

In the image above, the potentially problematic ghost bands have been labelled. It is very possible that both these bands get picked up by the colour mask as white/yellow colour bands.

I will have to adjust this approach for the final implementation of my program as it limits the detection rate of gold bands and has the possibility of creating extra "ghost bands", however, for the prototype stage of the program, this blurring allows me to mask the colours successfully and accurately (assuming it is not gold).

Webserver

I have developed applications making use of a webserver before but have never developed applications that allow for an image input.

I will make a very basic prototype of my webserver, that allows for a user to input an image.

The code for my webserver prototype can be seen below.

Code

Python (Flask)

```
import json
import os.path
from os.path import join, dirname, realpath

from flask import Flask, request, render_template, jsonify

from prototype.detection.Detector import Detector


UPLOADS_PATH = join(dirname(realpath(__file__)))
app = Flask(__name__, template_folder='resources', static_folder='resources/static/')


# Returns the UI of the page.
@app.route('/ui', methods=['GET'])
def ui():
    return render_template('Resistor.html')


# Calls the detection program.
@app.route('/api/detect', methods=['POST'])
def detect():
    file = request.files['file']

    location = f'{os.getcwd()}\\prototype\\images\\{file.filename}'

    with open(location, 'wb') as target:
        file.save(target)

    detected = Detector().detect(location)
```

```
    return jsonify(detected=detected)
```

HTML and JS

```
<!DOCTYPE html>
<html lang="en">
<meta name="viewport" content="width=device-width, initial-scale=1"/>
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">

<body>
<h1>Resistor Calculator</h1>

<div class="w3-container">
    <div class="w3-container">

        <input type="button" id="select_button" value="Select Resistor" class="w3-btn w3-black"
               onclick="document.getElementById('select_input').click();"/>

        <input type="file" id="select_input" style="display:none;" accept="image/*"/>

        <input type="button" id="scan_button" value="Scan Resistor" class="w3-btn w3-black"
style="display:none;"/>

    </div>
</div>
</body>

<script>
    var select_input = document.querySelector('#select_input');
    var scan_button = document.querySelector('#scan_button');

    select_input.onchange = function () {
        file = select_input.files[0];
        scan_button.style.display = 'inline'
    }

    scan_button.onclick = function () {
        let file = select_input.files[0];
        console.log(file)
        uploadAndResponse(file);
    }

    function uploadAndResponse(file) {
        let form = new FormData();
        let xhr = new XMLHttpRequest();

        form.append('file', file);
        xhr.open('post', '/api/detect', true);
        xhr.send(form);

        xhr.onreadystatechange = function () {
            if (xhr.readyState === XMLHttpRequest.DONE) {

                let detected = JSON.parse(xhr.responseText)

                let detected_colours = detected['colours'];

                console.log('COLOURS: ' + detected_colours)
            }
        }
    }

</script>
</html>
```

Result

The HTML code produces the following – a very basic webpage that allows for image input.

Figure 31 - Webpage

Clicking the select resistor button opens a file select window. This can be seen below.

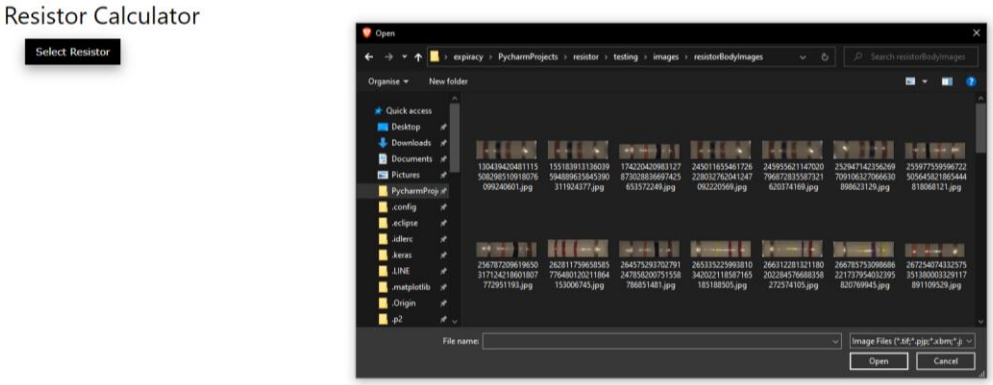


Figure 32 - Webpage with file select window

After selecting a file, a scan button appears.

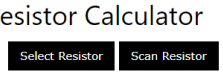


Figure 33 - Webpage with scan button

After clicking the scan button, the image is successfully received by the program. This is the highlighted file that can be seen below within the prototype directory. This shows my webserver prototype is working properly.

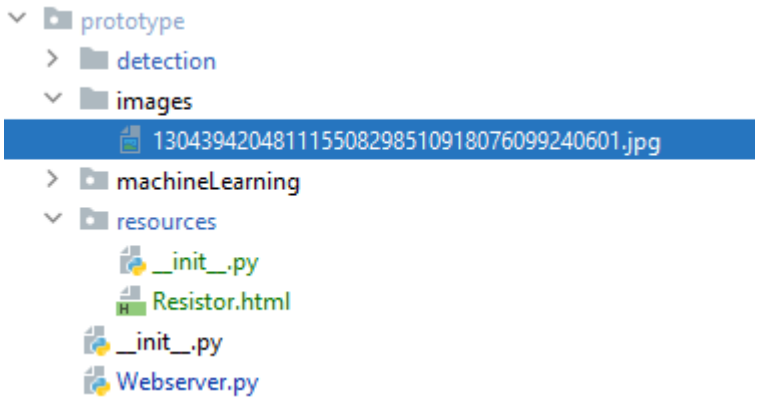


Figure 34 - File structure of prototype directory

Design

This section of the document will explain both the design of the program, before and after it has been completed.

System Overview

Overall problem

As outlined in my analysis, I hope to solve the problem of quickly and easily identifying resistor values by developing an application that will allow the identification of a resistor's values from an image while addressing the flaws or limitations of already existing apps.

The application I have created is a Python program that will run on a Flask webserver that has an HTML, CSS, and JavaScript front-end.

Compatibility

The application that I create will need to be platform independent as I will not know the operating system of user's device. By using a webserver, the application I create is platform independent.

The UI of the application may not be as easy to use on devices with particularly low resolutions as it may display incorrectly. However, the application should remain fully functional if the device has an internet connection and a compatible browser.

An offline part functional mode will be available if the user wishes to download the webpage files and to run them locally. This will require them to download the HTML, CSS, and JavaScript files.

Designing the Graphical User Interface

Wireframes

A wireframe is a schematic or blueprint that is useful for helping you, your programmers and designers think and communicate about the structure of the software or website you're building. (What Are Wireframes?, 2021)

In documentation for professionally made software, I noticed that many used wireframes to show both the basic principles and placement of elements within the UI as well as showing the flow of the program as it flows between different menus.

Wireframing is beneficial in many ways (Beginners' guide to wireframing UIs: benefits & best practices, 2):

- Define information architecture and content.
- Minimize rework and revisions.

- Facilitate client and stakeholder feedback.
- Get the user experience right from the get-go.
- Start defining and validating requirements.

On the next page (figure 20), is the wireframe UI that I have constructed for my system.

Designing the Wireframe UI

After researching software to create my wireframe UI with, Balsamiq seemed like a suitable choice. Figure 21 – which can be seen 2 pages over – is my wireframe UI.

Inspiration

Figure 35 - The main inspiration for my UI.

The element placement within the wireframe was inspired by the high-rated resistor calculator applications such as Resistance Calculator (Google Play Store, Resistance Calculator, 2021) as their design concepts are intuitive and clear, with both the client and reviews praising their good design.

I have incorporated the Resistor Calculator's resistor colour band button placement into my wireframe UI and have placed the output windows at the top of the screen as it is hard to miss. I have also ensured elements are kept in-line and have kept spacing consistent.

The UI on Different Devices

The placement of the elements was considered within a mobile device as this will be the worst-case scenario for the UI. I want to try and ensure that UI will fit inside the device horizontally without scrolling and that the buttons are sufficiently big for the user to click on by allocating them most of

the screen. On larger screens such as computer monitors – the UI will display the same, in the centre of the screen and background on either side of the main UI.

Ease of Use

To ensure the UI is easy to use for the user, I made the UI flow with the fewest number of screens possible. I have achieved this by doing most things via a main screen that can update depending on the user input.

Explaining the wireframe UI

- The arrows show the flow between each screen of the UI if something is pressed.
- The smiley face is a placeholder for a camera input.

I have created 2 possible versions for the UI when an image input is needed:

- Version 1 of the UI takes the user to a separate window such as the camera application to take and then input a resistor photo.
- Version 2 of the UI will use the camera via an embedded camera feed on the webpage to submit many resistor images until one is successful.

I hope to implement version 2 into my final application as it will have better detection rates and feel more seamless to the user.

The better detection rates will be result of the live camera feed allowing multiple images to be sent to the server until there is result and the increased seamlessness will be since there are less pages to navigate through. However, time may limit my ability to implement this functionality. I will discuss my final implementation in the evaluation.

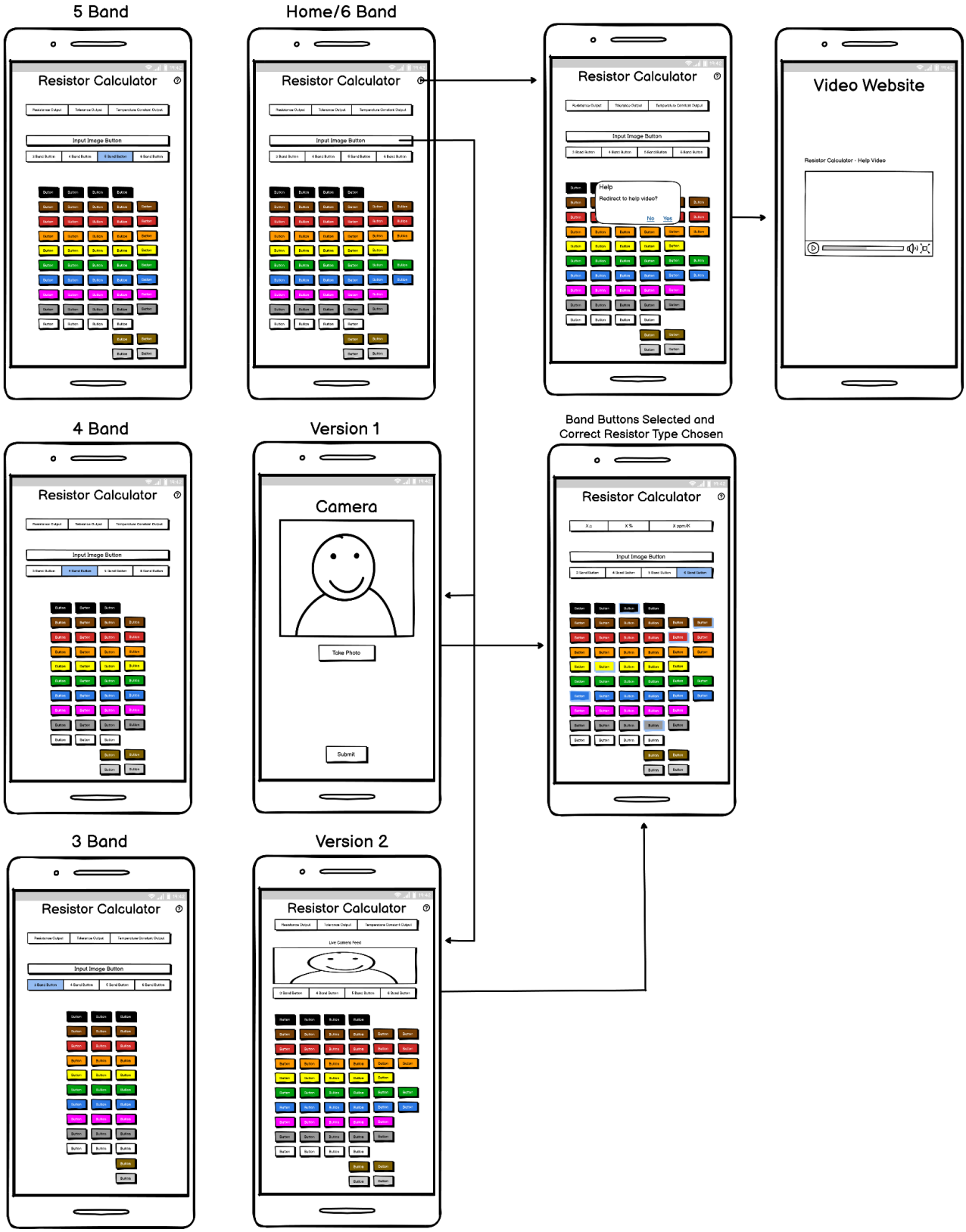


Figure 36 – Wireframe UI

GUI Design

As outlined in the wireframe UI, I will design a UI that is based on the Resistor Calculator, however, I intend to implement a more modern/minimalistic style.

As outlined by uxdesign.cc (A guide to the Modern Minimal UI style, 2021), a minimal UI should have:

- Subtle roundness on UI elements.
- Big, readable headings.
- Thoughtful use of colors.
- Focus on contrast.
- Limited use of effects.
- Small details which are often illustrated.

As stated by uxdesign.cc, the use of colours must be thoughtful. I must therefore pick a suitable colour palette that contains complementing colours.

As white is a resistor band colour, a contrasting background needs to be used to allow the white to show better. A dark mode style will be able to achieve this, however I will have to ensure the dark mode background differs in shade enough from the black and greys of other elements.

After researching colour palettes, I stumbled upon the Discord UI colour palette. I think the colours

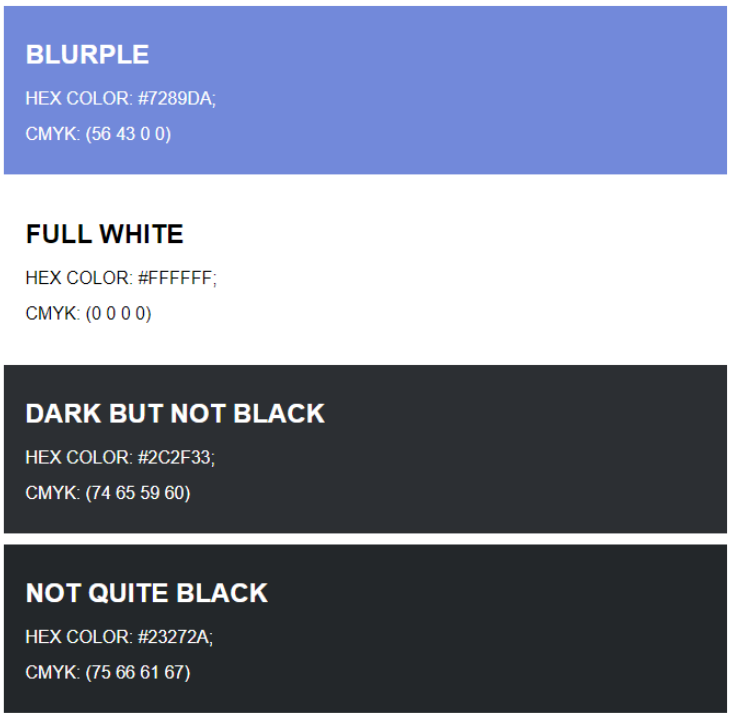


Figure 37 – The discord colour palette (Brand, 2021)

of this palette are very suited for my UI. The background colour named “dark but not black” and that can be seen in figure 19 will allow the resistor colour buttons to be clearly visible to the user.

I will incorporate this colour palette into the design of my UI.

Final UI Design

Application Home Page

Figure 38

- Figure 20 shows the application home page of my UI.
- I have built upon the wireframe UI by adding colour and consistent styling within the buttons and output windows.
- The user input buttons that allow you to input the image or choose the number of bands are a consistent colour and are grouped together so that it is clear to the user that they are input buttons.
- The output windows are white with black text as this is very typical of output windows; it will be clear to the user. The output when nothing is selected will be N/A.
- I have also chosen a modern Google font for the text within the UI as these fonts are free and are very easy to implement within an HTML page.
- I have added a gradient effect to both the gold and silver buttons to ensure that they do not get confused with other colours such as yellow, orange, or grey.
- I have added titles to the resistor band colour columns to make the app more understandable for those less familiar with resistors.
- There will be a help button in the top right of the application.

Application Home Page with Resistor Band Colour Buttons Selected

- The choice of background allows for a black outline to be used.
- The black button is dark grey so that the black outline is visible when it is selected.
- The output windows will show the units of the outputs to ensure the user understands what output window represents what.
- Selecting a resistor type that has less bands than the current type will cause all the buttons to be deselected.
- Selecting a resistor type that has more bands will not cause any deselection.

Detection Overview

Below I will briefly outline how I tackle the problem of finding a resistor's values from an image. All these following processes are explained in much more detail in the implementation section.

Locating the Resistor

My final application must be able to identify the location of the resistor body within an image.

After experimenting with cv2 operations, I will do this with a combination of using adaptive monochrome thresholding and eroding.

I find the amount of eroding I need to do using coordinate geometry to find the knee of a curve.

When the location of the resistor body has been identified, the image can then be cropped so that the image only contains the resistor body.

Band Location and Colour Identification

As highlighted in the prototype, I after experimenting with other options, I have determined HSV masking as the best way to locate the resistor bands.

By masking the HSV ranges for each possible resistor band colour, I find out whether the colour is present within an image and obtain the areas of the image at which the colour is present.

To increase the number of detected bands I get to work with, I have split the original image into 20 image slices. After masking all these image slices separately, I obtain much more data for the colours of the bands within the image of a resistor body.

Filtering the Bands

After identifying the colour and location of bands, I can then filter out the bands that don't meet certain criteria.

The criteria for a band to be kept is:

- That the band is above a certain height and width.
- That its x centre (x coordinate + half the width of the band) falls within a certain deviation of one of the x centres centroids identified by K-Means.

Identifying the Resistor Bands

After obtaining the filtered bands, I have used a combination of weighting and majority voting to identify the final resistor bands.

Checking Orientation

After obtaining the list of final resistor bands, I check the validity of the resistor in each orientation (when the list of resistor bands is and isn't reversed) and pick a valid orientation to be output to the final program.

Output

Below I will explain how the output of the project works.

As there are resistors with different numbers of bands, I must ensure that format the resistor colour values in the correct way before sending them to the webpage. For example, a 3-band resistor does not use the first 3 bands of the resistor; it uses the first 2 and then the 4th band. This means that I must leave a placeholder or blank value to the 3rd value to indicate that it is not in use for the processed resistor.

The colour values acquired from the processed resistor will be sent to the webpage via a response to the initial HTTP request. After sending the colour values, I must automatically input and select the correct resistor band colour buttons on the webpage so that the user can see a result that they can then change/modify if it is incorrect.

Error Handling

Try Excepts

I must ensure that errors within my program are appropriately handled so that the program understands what to do when there is an error. I can do this by incorporating the use of try and excepts.

If there is an error which will disrupt the processing of the program, I need to raise the exception to the level above as carrying on with erroneous data can cause the program to stop or provide a useless output.

If one of these fatal errors occurs, I will ensure that I output the error to the user on the webpage to let them know that there is an error with the processing.

Outputting the Values

Due to the nature of the possible colours for resistor bands, not all colours are available for all resistor bands. For example, gold cannot be selected on the first resistor band. If a certain colour that doesn't exist for a certain band is received by the webpage, a try catch should make sure that the value is ignored, and an appropriate error should be given to the user.

Libraries

It would be extremely difficult to develop an application that requires image processing that produces an accurate output without the use of libraries – especially with limited time.

There is also no need to develop something that a good existing library already does; it would be solving a problem which has already been solved well. The time which would be spent developing something a previously existing library already does could be spent developing my own project in other ways that will improve its functionality.

Although I have written a K-Means implementation, its efficiency is not good enough to be considered acceptable for the glare identification. I have used the SciKit implementation of K-Means for this part of my program specifically.

The main libraries I have used, and their purpose are listed in the table that can be seen below.

Library	Purpose
OpenCV (opencv-python, 2021)	• Will allow to carry out operations on the image to manipulate it. This includes operations such as monochrome thresholding and showing the image.
Flask (Flask, 2021)	• Make the application platform and device independent. • To host the webpage – the GUI.
Numpy	• For mathematical operations, such as finding the mean of a list or square rooting numbers. • For list operations, such as finding a list of differences.
SciKit	• For K-Means (on large very matrixes), has been used within the program for the glare identification as it is operating on a 2d image matrix that is multiple thousands of items long.

Pseudocode Algorithms

Webserver

```
CHECK FOR HTTP Request

IF HTTP Request RECEIVED THEN
    CHECK endpoint

    IF endpoint IS "UI" THEN
        SEND RESPONSE(DISPLAY UI webpage)

    ENDIF

    IF endpoint IS "api" THEN
        PROCESS request
        EXTRACT image from request

        resistor = Detector.detect(image)

        SEND RESPONSE(resistor)

    ENDIF

    IF endpoint IS "validate" THEN
        PROCESS request
        EXTRACT resistor colours from request

        valid = validate(resistor colours)

        SEND RESPONSE(valid)

    ENDIF

ENDIF
```

Resistor Value Calculations

```
SUBROUTINE
getDigits(band_1, band_2, band_3)
    digits <- dictionary(
        'BLACK':'0'
        'BROWN':'1'
        'RED':'2'
        'ORANGE':'3'
        'YELLOW':'4'
        'GREEN':'5'
        'BLUE':'6'
        'VIOLET':'7'
        'GREY':'8'
        'WHITE':'9'
        )

    IF band_1 NOT none THEN
        digit_1 <- digits(band_1)
    ELSE
        digit_1 <- ''

    IF band_2 NOT none THEN
        digit_2 <- digits(band_2)
    ELSE
        digit_2 <- ''

    IF band_3 NOT none THEN
        digit_3 <- digits(band_3)
    ELSE
        digit_3 <- ''

    RETURN digit_1 + digit_2 + digit_3

ENDSUBROUTINE

SUBROUTINE
getTolerance(colour)
    tolerances <- dictionary(
        'BLACK':0
        'BROWN':1
        'RED':2
        'ORANGE':3
```

```
        'YELLOW':4
        'GREEN':0.5
        'BLUE':0.25
        'VIOLET':0.1
        'GREY':0.05
        'GOLD':5
        'SILVER':10
        )

    IF colour NOT none THEN
        RETURN tolerances(colour)
    ELSE
        RETURN none
ENDSUBROUTINE

SUBROUTINE
getTemperatureConstant(colour)
    temperature_constants <- dictionary(
        'BLACK':0
        'BROWN':100
        'RED':50
        'ORANGE':15
        'YELLOW':25
        'BLUE':10
        'VIOLET':5
        )

    IF colour NOT none THEN
        RETURN temperature_constants(colour)
    ELSE
        RETURN none

ENDSUBROUTINE
```

K-Means

I have split the K-Means algorithm into multiple crucial parts.

Finding the distance between 2 points

Using the equation for Euclidean distance, I can input 3d coordinates point_1 and point_2 and find the distance between them.

```
SUBROUTINE
find_distance(point_1, point_2[0])
    dx_squared <- (point_1[0] - point_2[0]) ^ 2
    dy_squared <- (point_1[1] - point_2[1]) ^ 2
    dz_squared <- (point_1[2] - point_2[2]) ^ 2

    distance <- (dx_squared + dy_squared + dz_squared) ^ 0.5

    RETURN distance

ENDSUBROUTINE
```

Finding the centroid seeds

Below the pseudocode for the K-Means++ centroid selection algorithm can be found.

1. Choose the first centroid at random.
2. Obtain the distances from each item's closest centroid.
3. Out of these distances, find the data point with the largest distance to its closest centroid.
4. Select the data point that causes the largest distance to its closest centroid and set it as the next centroid.

```
SUBROUTINE
initialize_centroids(data, number_of_centroids)
    number_of_items <- LENGTH(data)

    centroids <- EMPTY LIST

    APPEND(data[RANDOM_INT(0, number_of_items - 1) TO centroids])

    FOR centroid_number <- 0, number_of_centroids - 1
        minimum_item_centroid_distances <- EMPTY LIST

        FOR item IN data
            minimum_item_centroid_distance <- INFINITY

            FOR centroid IN centroids
                item_centroid_distance <- find_distance(item, centroid)
```

```
                minimum_item_centroid_distance <- MIN(minimum_item_centroid_distance, item_centroid_distance)

            ENDFOR

            APPEND(minimum_item_centroid_distance TO minimum_item_centroid_distances)

        ENDFOR

        index_of_highest_distance <- INDEX(MAX(minimum_item_centroid_distances) IN minimum_item_centroid_distances)

        centroids[centroid_number] <- data[index_of_highest_distance]

    ENDFOR

    RETURN centroids

ENDSUBROUTINE
```

Adjusting the seeds to be centroids

Below, the pseudocode for the K-Means fit algorithm can be seen. This algorithm is responsible for adjusting the input centroids to be the centroids.

1. Input “seeds” (initial centroids into the algorithm).
2. Assign all the data points to the closest cluster centroid.
3. Group all the data points by their closest cluster centroid.
4. Find the mean of each grouped data points, these are the new centroids.
5. Rerun the algorithm if certain conditions are not met.

```
SUBROUTINE
fit(data, centroids, iteration, max_number_of_iterations)

    intra_cluster_distances_for_centroids <- EMPTY LIST

    FOR centroid IN centroids

        intra_cluster_distances_for_item <- EMPTY LIST

        FOR item IN data
            APPEND(find_distance(centroid, item) TO intra_cluster_distances_for_item)

        ENDFOR

    ENDFOR
```

```
    labels <- EMPTY LIST

    FOR intra_cluster_distances_for_item IN intra_cluster_distances_for_centroids

        lowest_distance <- INFINITY

        FOR distance IN intra_cluster_distances_for_item

            lowest_distance = MINIMUM(lowest_distance, distance)

        label <- FIND_INDEX_OF(lowest_distance IN intra_cluster_distances_for_item)

        APPEND(label TO labels)

        ENDFOR

    ENDFOR

    data_grouped_by_label <- LIST CONTAINING(EMPTY LIST FOR EACH centroid IN centroids)

    FOR index <- 0 TO LENGTH(labels)
        APPEND(data[index] TO data_grouped_by_label[labels[index]])

    ENDFOR

    FOR index <- 0 TO data_grouped_by_label
        x_total <- 0
        y_total <- 0
        z_total <- 0

        FOR item IN data_grouped_by_label[index]
            x_total <- x_total + item[0]
            y_total <- y_total + item[1]
            z_total <- z_total + item[2]

        ENDFOR

        x_mean <- x_total / LENGTH(grouped_data)
        y_mean <- y_total / LENGTH(grouped_data)
        z_mean <- z_total / LENGTH(grouped_data)

        new_centroids[index] <- [x_mean, y_mean, z_mean]
```

```
    ENDFOR

    centroid_differences_sum <- 0

    FOR index <- 0 TO new_centroids
        centroid_difference <- find_distance(centroids[index], new_centroids[index])
        centroid_differences_sum <- centroid_differences_sum + ABS(centroid_difference)

    ENDFOR

    iteration += 1

    IF centroid_differences_sum != 0 OR iteration = 1 THEN
        IF iteration <= max_number_of_iterations THEN
            RETURN fit(data, centroids, iteration, max_number_of_iterations)

        ELSE
            RETURN new_centroids, seeds

        ENDIF

    ELSE
        RETURN new_centroids, seeds

    ENDIF

ENDSUBROUTINE
```

Finding the optimal value of K

Below is my pseudocode algorithm to find the optimal value of K (the number of centroids).

```
SUBROUTINE
find_optimal_k(data)
    dunn_indexes <- EMPTY LIST

    FOR centroid_amount <- 3 TO 6

        number_of_centroids <- centroid_amount

        seeds <- initialize_centroids(data)

        centroids, labels <- fit(data, seeds)
```

```
        data_grouped_by_label <- GROUP data BY label

        intra_cluster_distances <- find_intra_cluster_distances(data_grouped_by_label)

        maximum_intra_cluster_distance <- MAX(intra_cluster_distances)

        inter_cluster_distances <- find_inter_cluster_distances(data_grouped_by_label)

        minimum_inter_cluster_distance <- MIN(inter_cluster_distances)

        dunn_index <- maximum_intra_cluster_distance / minimum_inter_cluster_distance

        APPEND(dunn_index TO dunn_indexes)

    ENDFOR

    optimal_k <- INDEX(MAX(dunn_indexes))

    RETURN optimal_k

ENDSUBROUTINE
```

Merge Sort

Below the pseudocode for a recursive version of merge sort can be found. The sort is initiated by calling the sort subroutine which will make recursive calls to sort and merge the data.

```
SUBROUTINE
sort(data)
    IF LENGTH(data) <= 1 THEN
        RETURN data

    ENDIF

    middle_index <- FLOOR(LENTH(data), 2)

    left <- LIST UP TO middle_index
    right <- LIST AFTER middle_index

    left <- sort(data)
    right <- sort(data)

    sorted_and_merged_list <- sort_and_merge_lists(left, right)
```

```
    RETURN sorted_and_merged_list

ENDSUBROUTINE

SUBROUTINE
sort_and_merge_lists(left, right)
    left_index <- 0
    right_index <- 0

    sorted_and_merged_list <- EMPTY LIST

    WHILE left_index < LENGTH(left) AND right_index < LENGTH(right) THEN
        IF left[left_index] < right[right_index] THEN
            APPEND(left[left_index] TO sorted_and_merged_list)

            left_index <- left_index + 1

        ELSE
            APPEND(right[right_index] TO sorted_and_merged_list)

            right_index <- right_index + 1

        ENDIF

        EXTEND(sorted_and_merged_list WITH REMAINING items IN left)
        EXTEND(sorted_and_merged_list WITH REMAINING items IN right)

    ENDWHILE

ENDSUBROUTINE
```

Designing and Developing the Algorithms

Within the evaluation section of this document (page 169-173), I have highlighted much information about how I developed and designed parts of my program along with my thoughts on each of them.

I have explained the different approaches I have made to solve different parts of the program along with their issues/limitations and how I have addressed them.

Class Diagrams

Image, Greyscale, Annotation, BGR and HSV Classes

Below is a class diagram for the image class.

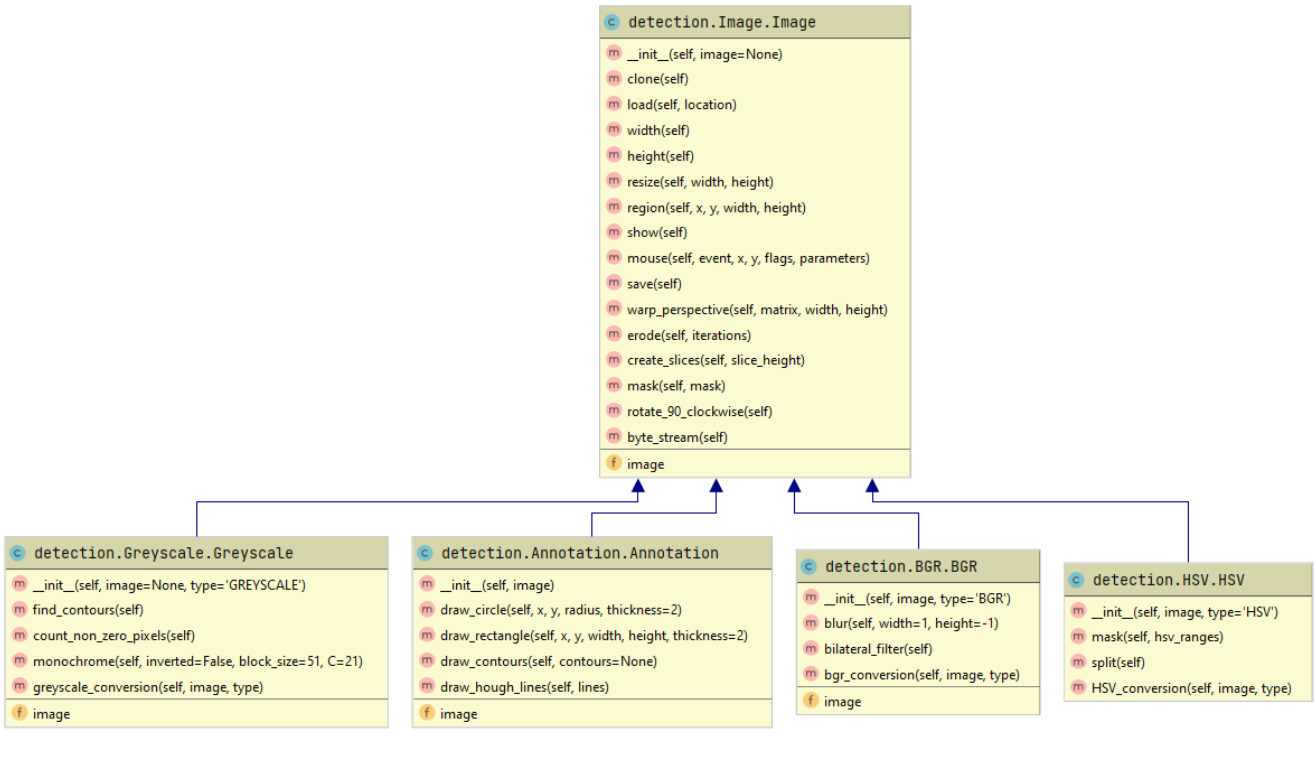


Figure 39 - Image class diagram

Overview

The image class is the parent class of 4 other image classes which are, Greyscale, Annotation, BGR and HSV. Although I originally had all the image functions within the Image class, I decided to split them up. Certain image operations can only be carried out on certain types of images; therefore, I have decided to split the classes up accordingly to what operations work on what image.

Class Relationships

As the Greyscale, Annotation, BGR and HSV class inherit from the Image class, they all have access to the Image methods and inherit the image attribute from their parent Image class.

Class Purposes

Image

The Image class contains generic operations that work on all types of images.

Greyscale

The Greyscale class inherits from the Image class and contains operations that only work on greyscale type images. To initiate an instance of the Greyscale class, you must give the image and the current image type so that the image can be converted.

Annotation

The Annotation class inherits from the Image class and contains operations that annotate the image. It initiates a from just an image and works for all image types.

BGR

The BGR class inherits from the Image class and contains operations that only work on BGR type images. To initiate an instance of the BGR class, you must give the image and the current image type so that the image can be converted.

HSV

The HSV class inherits from the Image class and contains operations that only work on HSV type images. To initiate an instance of the HSV class, you must give the image and the current image type so that the image can be converted.

ResistorBand, Resistor, SliceBand and SliceBands Classes

Below is the class diagram that shows the relationship between the ResistorBand, Resistor, SliceBand and SliceBands classes.

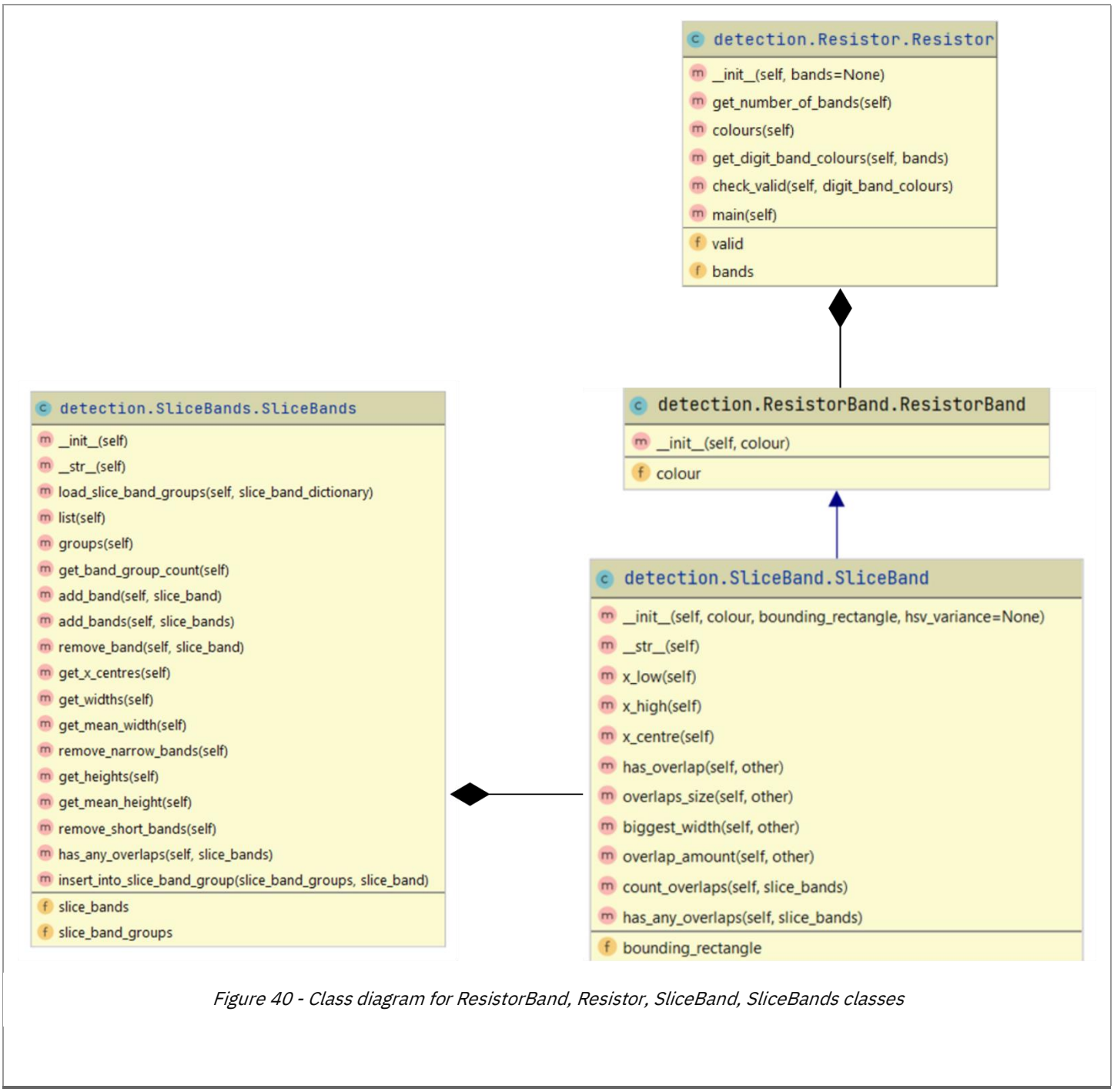


Figure 40 - Class diagram for ResistorBand, Resistor, SliceBand, SliceBands classes

Class Relationships

- The SliceBands class is composed of multiple instances of SliceBand. Without these instances of the SliceBand class, the SliceBands object cannot exist.

- The ResistorBand object is the parent class of the SliceBand object which inherits the colour from the ResistorBand.
- The Resistor class is composed of multiple instances of ResistorBand. Without these instances of the ResistorBand class, the Resistor object cannot exist.

Class Purposes

Below, I will explain the purpose of each class.

SliceBands

The purpose of the SliceBands class is to encapsulate a list of SliceBand instances and carry out operations on all the slice band instances or to modify the SliceBands list attribute (the list of slice band instances).

SliceBand

The purpose of the SliceBand class is to encapsulate the bounding rectangle and colour, carry out basic operations on the SliceBand and return attribute information.

ResistorBand

The ResistorBand class is just an encapsulation class. It has no methods, and its only purpose is to encapsulate a colour.

Resistor

The purpose of the Resistor class is to encapsulate a list of ResistorBand instances and to format the data from the bands attribute and check the validity of the resistor bands that were used to initiate the class instance.

Colour Class (Enumeration)

Below, the class diagram for the colour class can be seen.

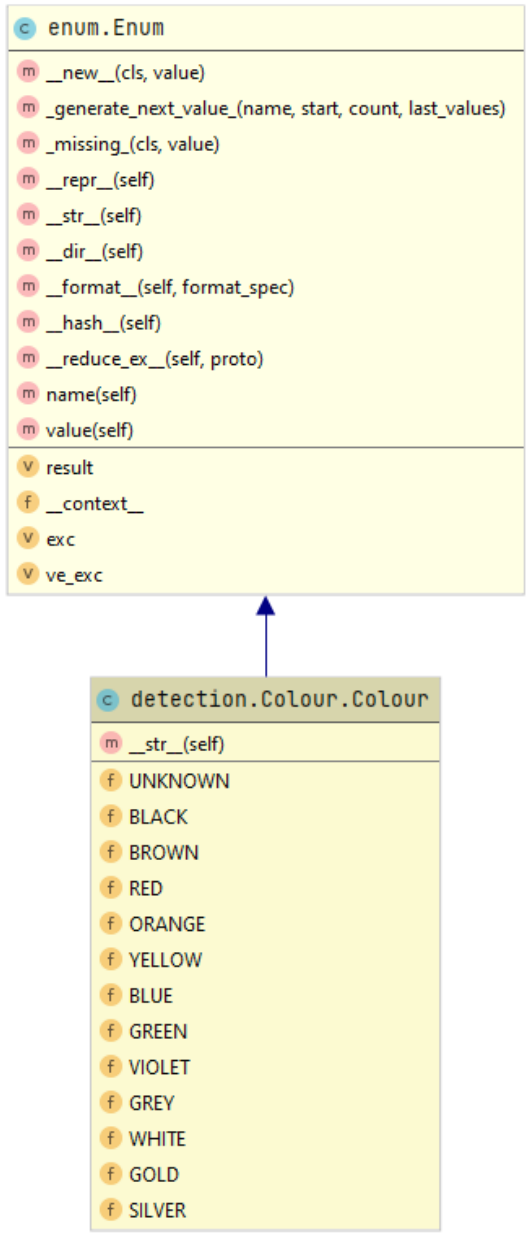


Figure 41 - Colour Enum class

Class Relationships

The Colour class inherits from the Enum class that I have imported.

Class Purpose

I will explain the purpose of the colour class within my program.

Colour

The purpose of the Colour class is to encapsulate all the possible colours in one place, allow for enumeration through the colours and to make it easier when referencing colours within my code.

ResistorLocator Class

The class diagram for the ResistorLocator class can be seen below.

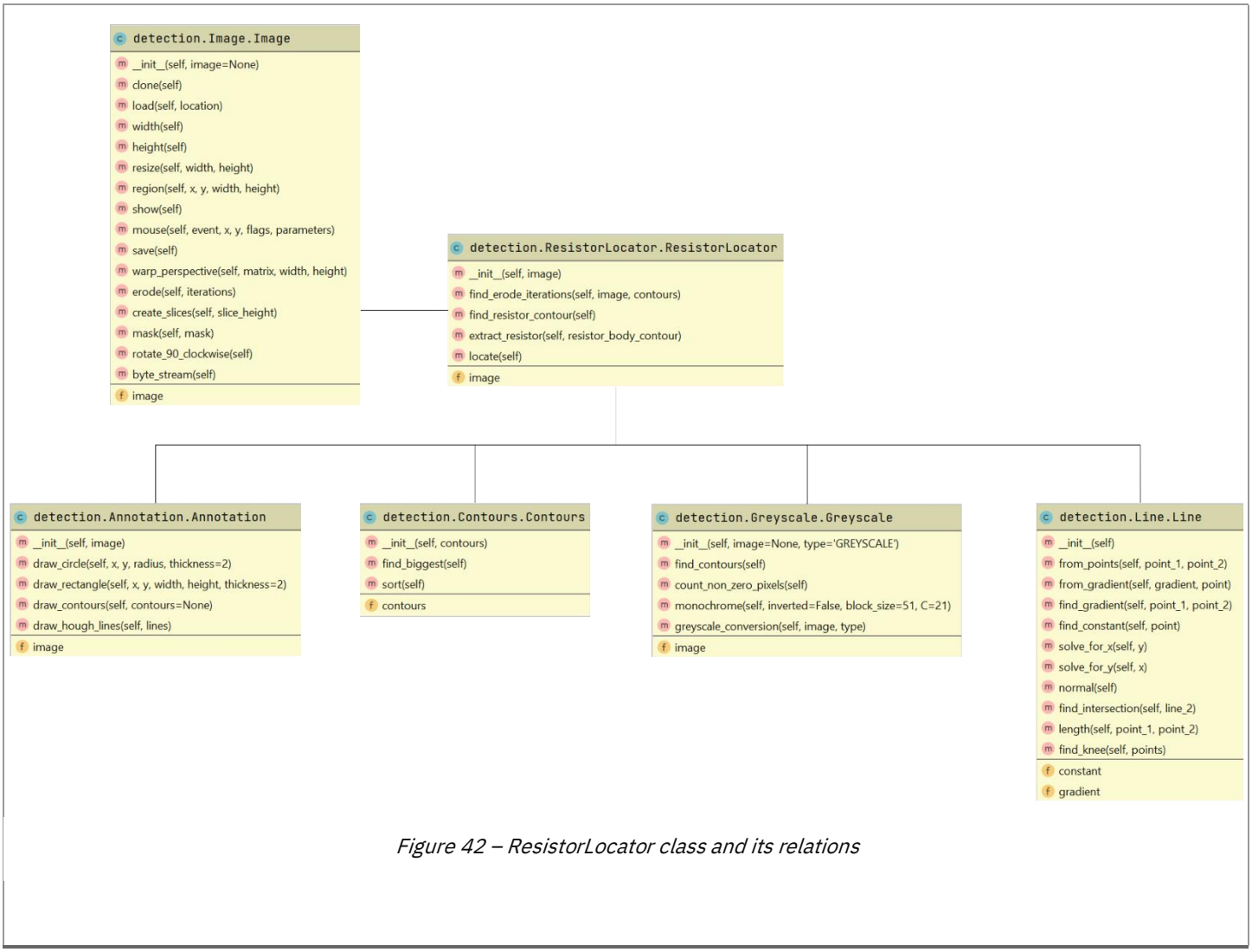


Figure 42 – ResistorLocator class and its relations

Class Relations

The ResistorLocator class is initialized with an instance of the Image class and is associated with the Annotation, Contours, Greyscale and Line classes.

Class Purposes

Below, I will explain the purpose of each class.

Image

An instance of the Image class whose image attribute contains the matrix that represents the resized original resistor image. It is also used for image operations such as warp_perspective.

Annotation

The Annotation class is used to draw the contours onto the resistor image to fill in the resistor body.

Contours

The Contours class is used to sort find the biggest contour. This function is used to identify the contour of the resistor as it is the biggest contour within the image.

Greyscale

The Greyscale class is used to allow me to use image operations that work on greyscale type images. This includes find_contours and monochrome.

Line

The Line class is used to allow me to use coordinate geometry to help me identify the knee of a curve to determine the number of erode iterations that are needed.

ResistorLocator

The purpose of the resistor locator is to crop out a horizontally oriented image of the resistor body that can be used within later algorithms.

GlareRemover Class

Below is the class diagram for the GlareRemover class.

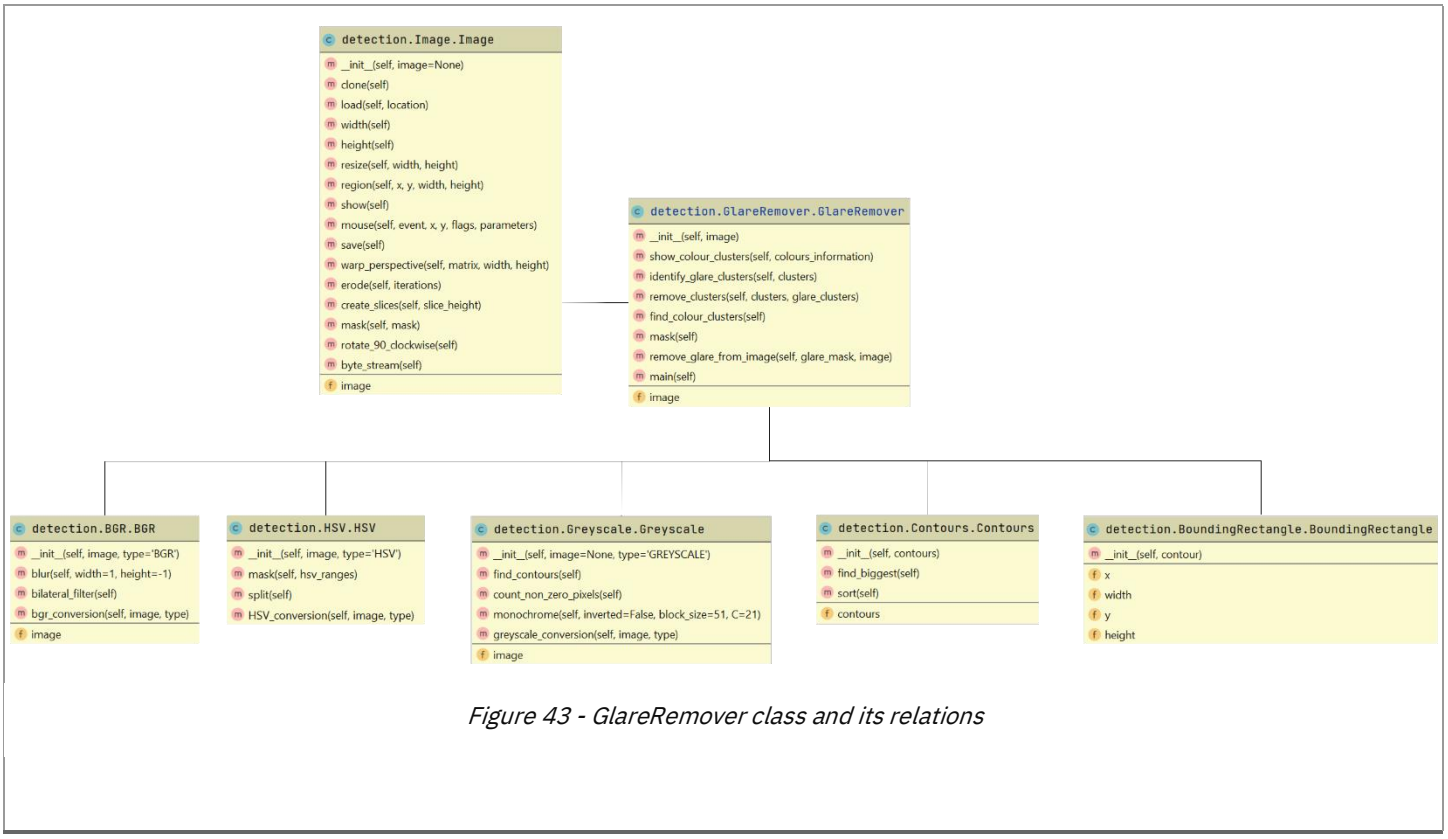


Figure 43 - GlareRemover class and its relations

Class Relations

The GlareRemover class is initialized with an instance of the Image class and is associated with the BGR, HSV, Greyscale, Contours and BoundingRectangle classes.

Class Purposes

Below, I will explain the purpose of each class.

Image

The Image class's image attribute contains the matrix that represents the resistor body image and allows me to perform generic image operations such as resize and region on the image.

BGR

The BGR class is used for operations that work on BGR type images such as blur.

HSV

The HSV class is used for operations that work on performed on HSV type images. Within the GlareRemover class, the HSV class was used to convert pixel values and to mask HSV colour ranges.

Greyscale

The Greyscale class is used for operations that work on Greyscale images such as find_contours.

Contours

The Contours class is used to sort find the biggest contour. This function is used to identify the biggest remaining region in an image.

BoundingRectangle

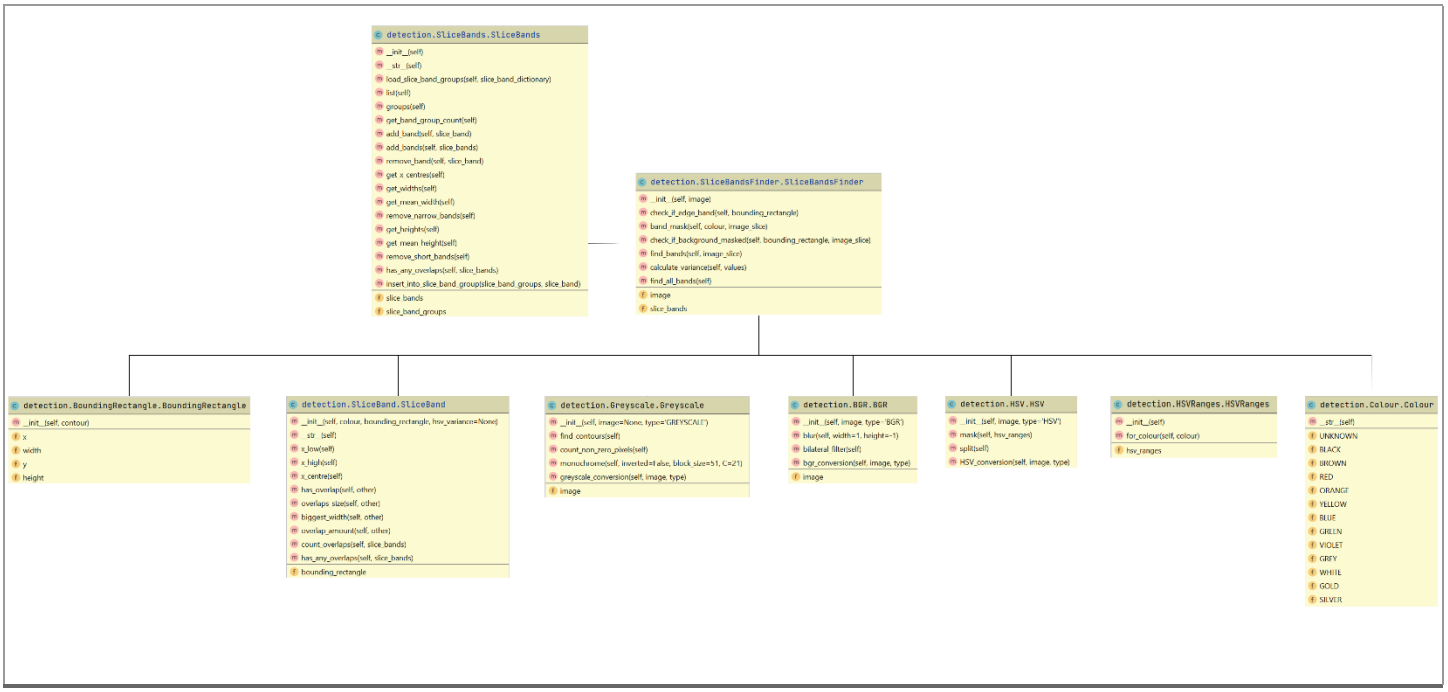
The BoundingRectangle class is an encapsulation class that initializes from a contour allows me to group together the bounding rectangle attributes.

GlareRemover

The purpose of the GlareRemover class is to return an image of the resistor body that does not contain glare and can be used for later algorithms.

SliceBandFinder Class

Below is the class diagram for the SliceBandFinder class.



Class Relations

The SliceBandFinder class contains an instance of the SliceBands class within its attributes and is associated with the BoundingRectangle, SliceBand, Greyscale, BGR, HSV, HSVRanges and Colour classes.

Class Purposes

Below, I will explain the purpose of each class.

SliceBands

An instance of the SliceBands class is stored within the SliceBands slice_band attribute. The SliceBandFinder will update this instance of SliceBands with the slice bands that are found.

BoundingRectangle

The BoundingRectangle class is an encapsulation class that initializes from a contour allows me to group together the bounding rectangle attributes.

SliceBand

Every time a slice band has been located, a new instance of the SliceBand class will be initialized with the slice band's bounding rectangle and colour.

Greyscale

The Greyscale class is used to carry out operations that only work on greyscale type images. This includes functions such as find_contours and count_non_zero_pixels.

BGR

The BGR class is used to carry out operations that work on BGR type images such as blur.

HSV

The HSV class is used to carry out operations that work on HSV type images such as the mask operation.

HSVRanges

An encapsulation class that contains a dictionary that contains all the HSV ranges for all the possible resistor band colours. These ranges can be searched using the for_colour function.

Colour

The colour enumeration class that contains all the possible resistor band colours.

SliceBandsFinder

The SliceBandsFinder class is responsible for adding slice bands that meet certain criteria to the list within the slice bands instance class attribute after masking the image by masking the resistor with certain HSV ranges.

SliceBandFilter Class

Below is the class diagram for the SliceBandFilter class.

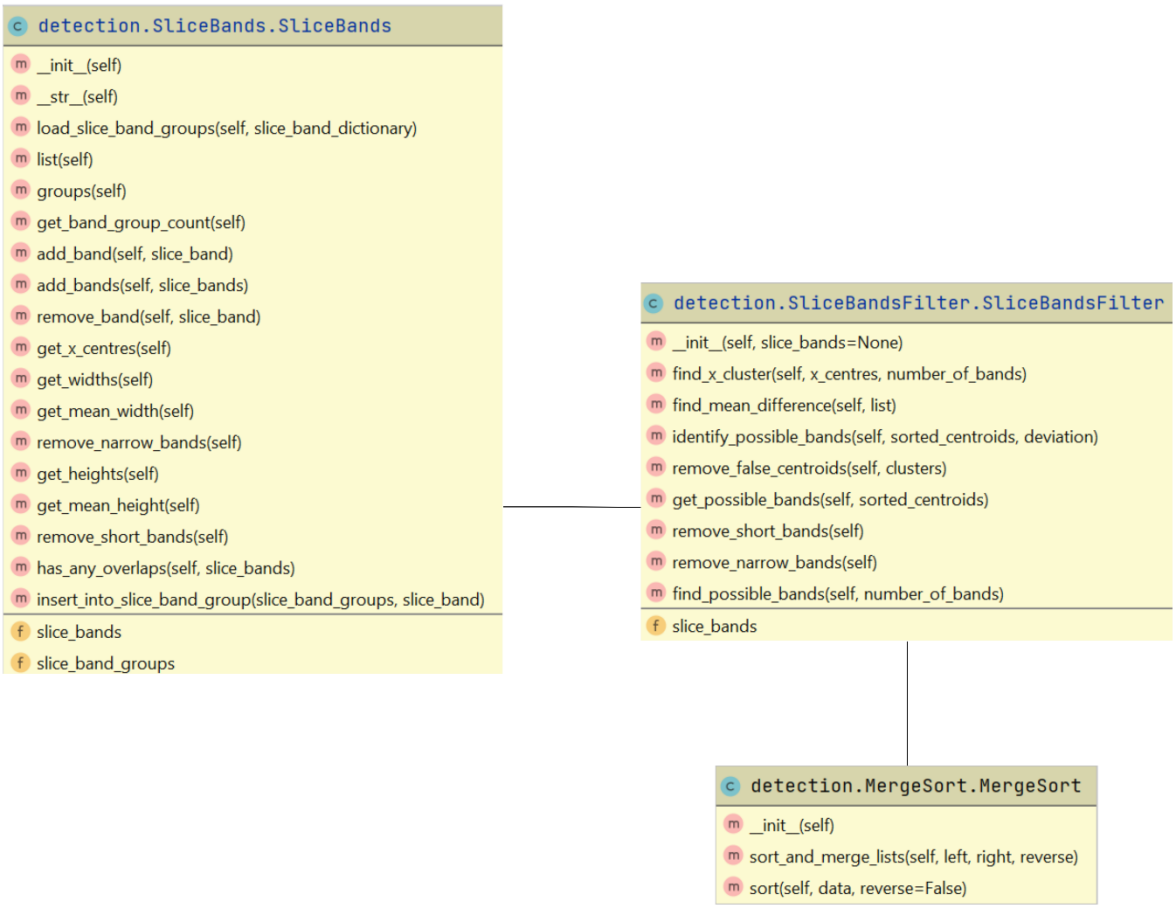


Figure 44 - SliceBandFilter class and its relations

Class Relations

The SliceBandsFilter class is initialized with an instance of the SliceBands class. The SliceBandsFilter class is associated with the MergeSort class.

Class Purposes

Below, I will explain the purposes of each class.

SliceBands

The SliceBands instance that I have used to initiate the SliceBandFinder with has an attribute which is a list of SliceBand class instances (found by the SliceBandFinder).

MergeSort

The MergeSort class is used to sort unordered data which needs to be ordered such as the centroids of the x coordinates of the band centres.

SliceBandsFilter

The SliceBandsFilter class is initialized with an instance of the SliceBands class. Its purpose is to then filter out any slice band that does not meet certain criteria.

BandIdentifier Class

Below is the class diagram for the BandIdentifier class.

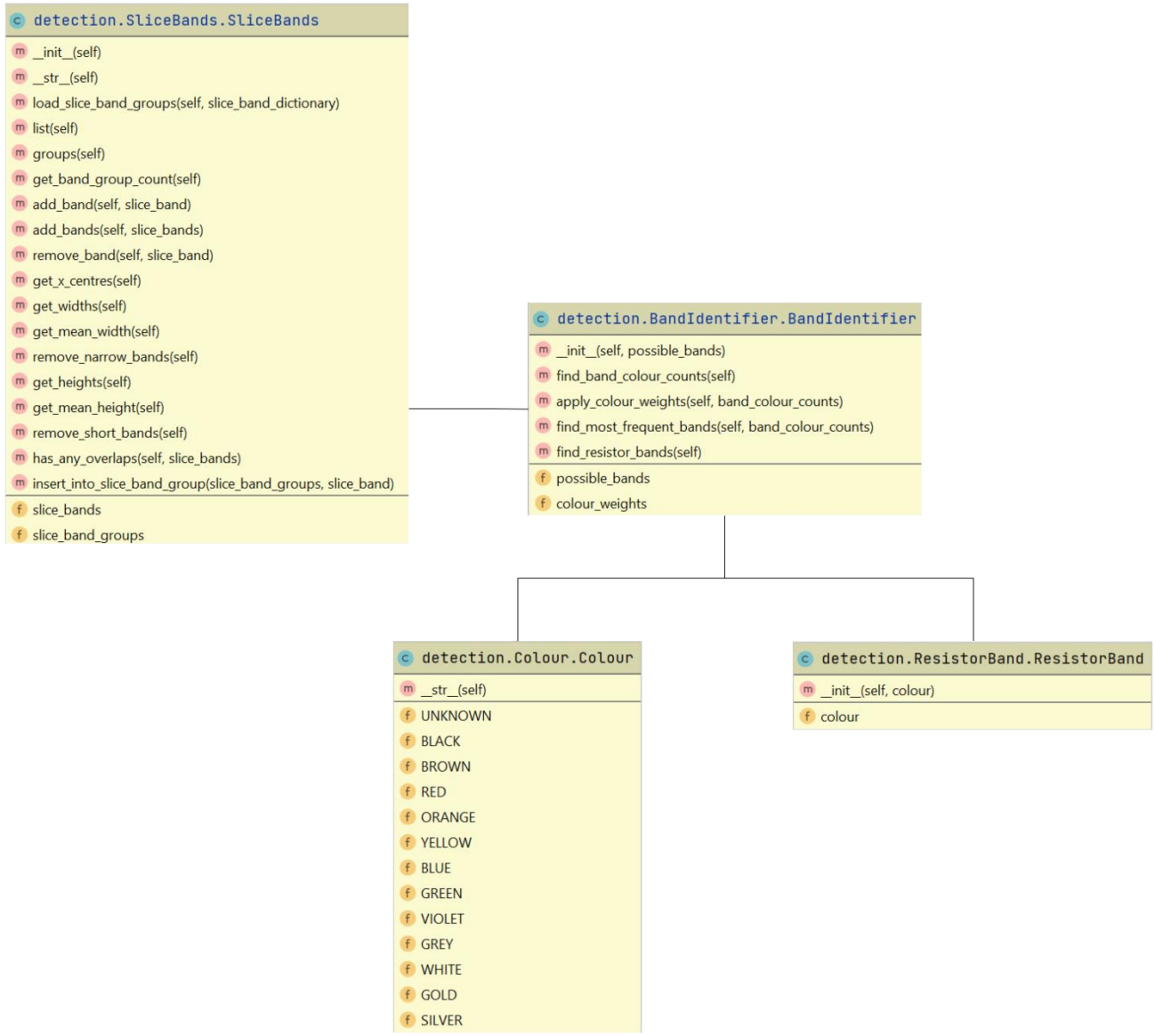


Figure 45 - BandIdentifier class and its relations

Class Relations

The BandIdentifier class is initiated with an instance of the SliceBands class and is associated with the Colour and ResistorBand classes.

Class Purposes

Below, I will explain the purpose of each class.

SliceBands

The instance of the SliceBands class that is used to initialize the BandIdentifier has an attribute slice_band_groups. This slice_band_groups attribute is a list of lists that contain all the possible SliceBand instances for each band.

Colour

The Colour enumeration class is used for referencing colours.

ResistorBand

For every possible band colour that is identified, an instance of the ResistorBand class is created with that colour and appended to the resistor_bands list.

BandIdentifier

The BandIdentifier class is used to select the most likely colour from each slice band group by using a combination of weighting and majority voting.

Detector Class

Below is the class diagram for the Detector class.

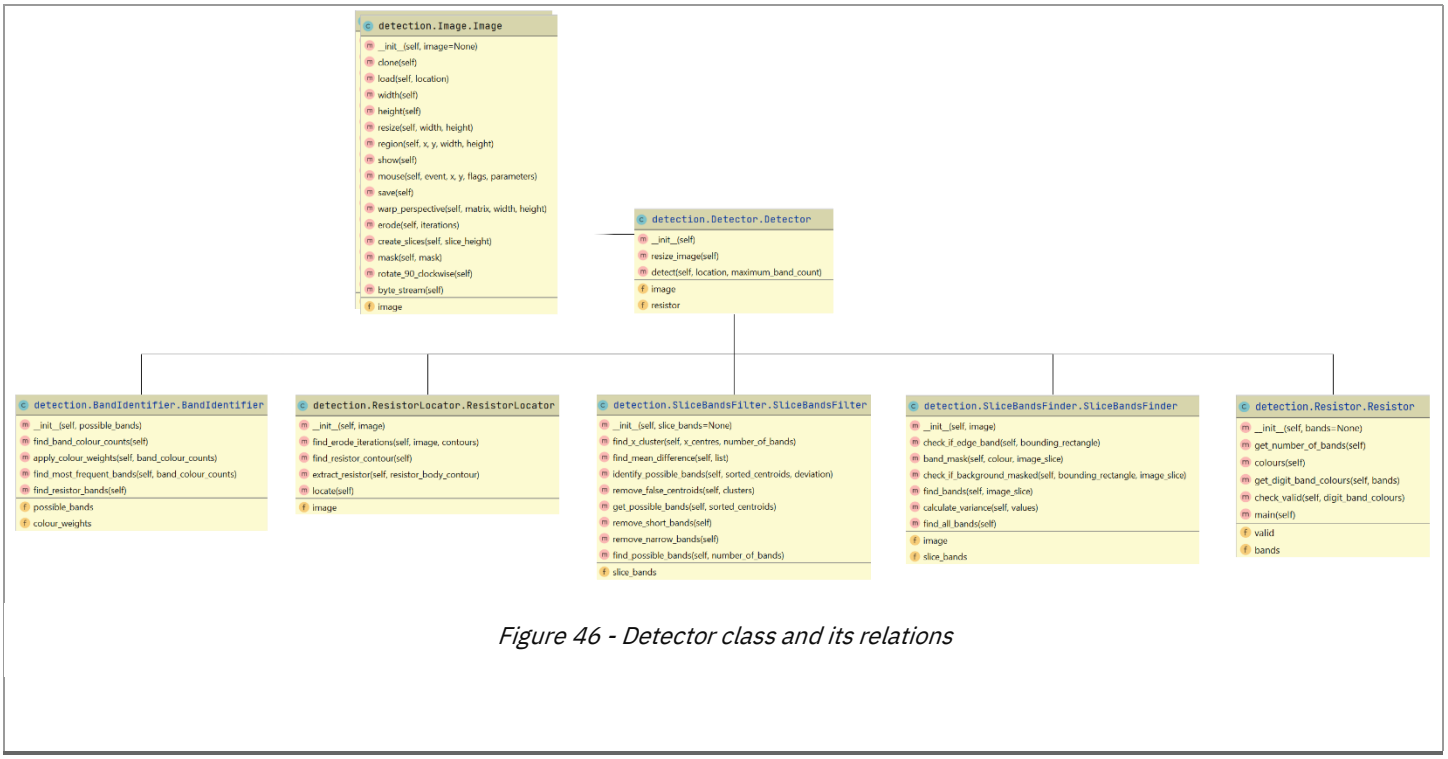


Figure 46 - Detector class and its relations

Class Relationships

The Detector class is initialized with an instance of the Image class and is associated with the BandIdentifier, ResistorLocator, SliceBandsFilter, SliceBandsFinder and Resistor classes.

Class Purpose

All classes but the Detector class have already been explained above.

Detector

The purpose of the Detector class is to act as the main function of the resistor value detector program. It is responsible for running all the classes that it is associated it in the correct order, to obtain an instance of the Resistor class that contains the detected resistor colours and validity.

System Relationships

Below is a diagram that shows the relationship between the different parts of the system and how they participate within the system and what each item will be responsible for.

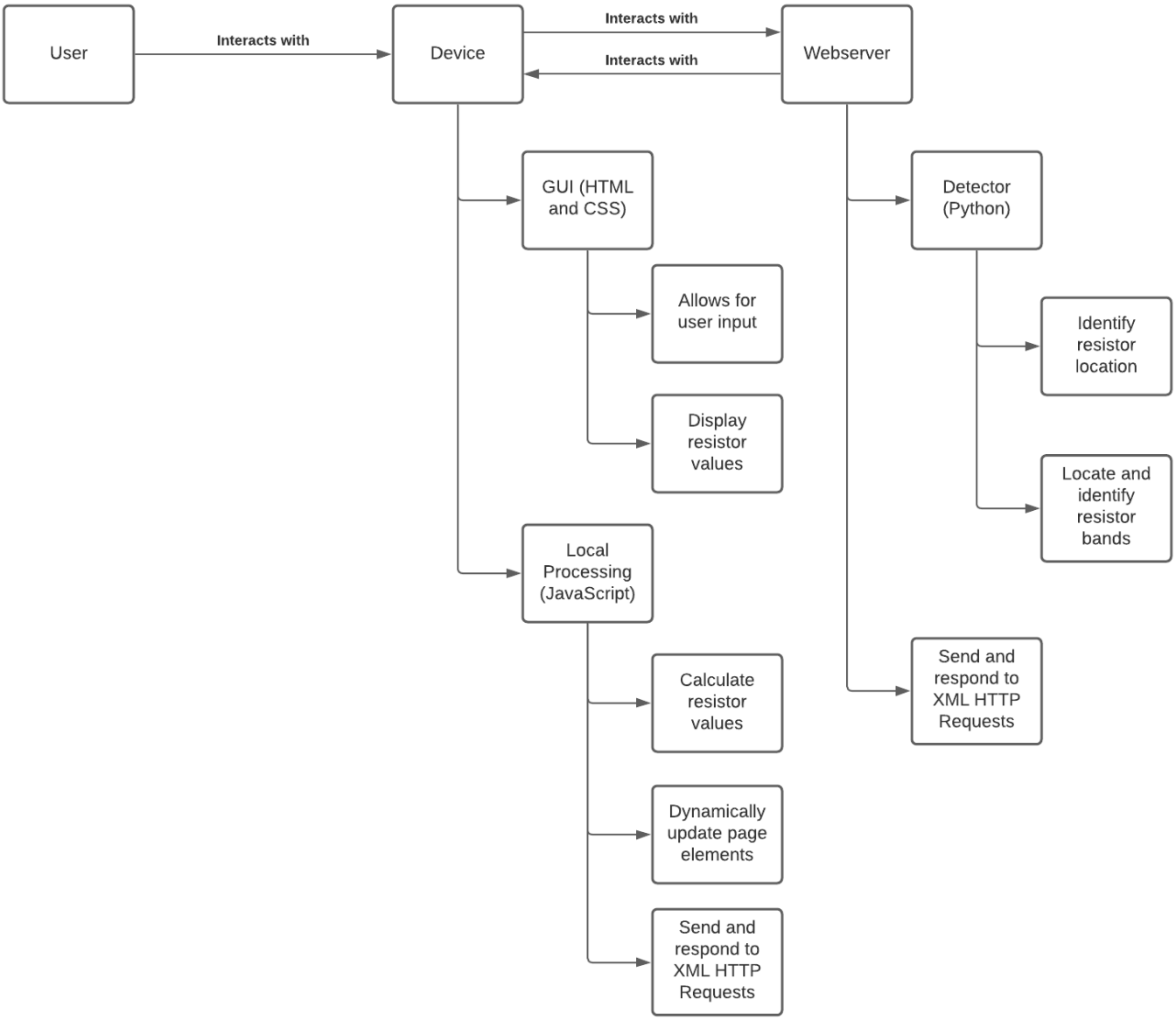


Figure 48 - System relationships diagram

The user will input into their device:

- If the user manually selects the resistor colour bands into the webpage, the processing will be done locally on the device – within JavaScript. Resistor band colour button elements will then be updated to appear selected.

- If the user inputs resistor images, the image will be sent to the webserver where it can be processed using Python. The result will then be returned – with the resistor band amount and band colours being selected on the webpage. At this stage, the user can then manually select other bands to correct the resistor values.

Data Flow Diagram

Explanation

Data flow diagrams map out the flow of information within a system and they can be created to represent the system with different amount of abstraction.

(Levels in Data Flow Diagrams (DFD), 2021)

Data Flow Diagram Symbols

Below is a key for what the symbols within a data flow diagram represent.

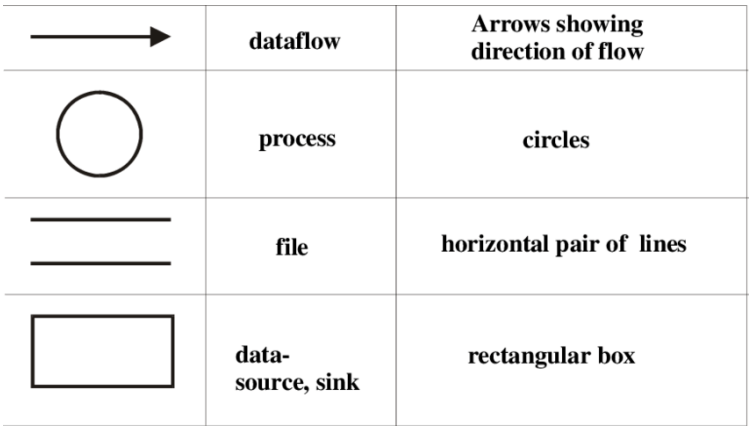


Figure 49 - Data flow diagram symbols (ResearchGate, 2021)

Although it is not on this key, stacked rectangles represent an external entity (such as the user).

Webserver

The Webserver class is responsible for connecting both the webpage and the program together.

From the webpage, 2 API calls are made, these are the validate and detect app routes.

Detector

Below is the data flow diagram that shows how data flows from the user's input to a result.

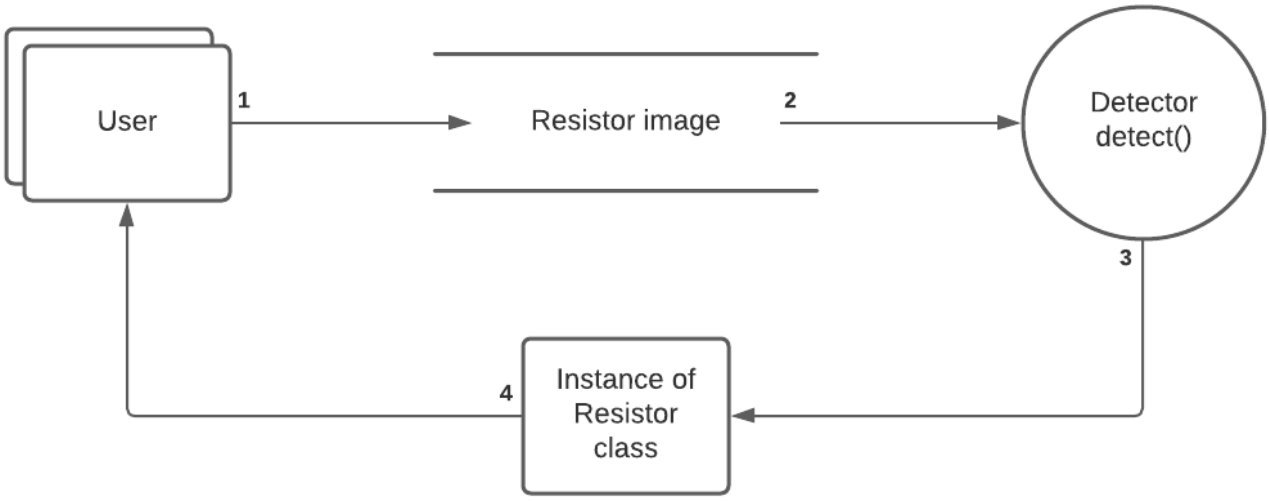


Figure 50 - Data flow diagram showing data flow between webserver and detector

1. The user inputs a resistor image into the page.
2. This resistor image is saved to a directory and the location that the image was saved is passed into the Detector class's detect function.
3. The result of the Detector class's detect function is an instance of the Resistor class that contains the detected colour values for the resistor as well the validity. These values are output on the webpage for the user to see.

Validate

Below is the data flow diagram that shows how the flow of data for the user's resistor band button inputs to be validated.

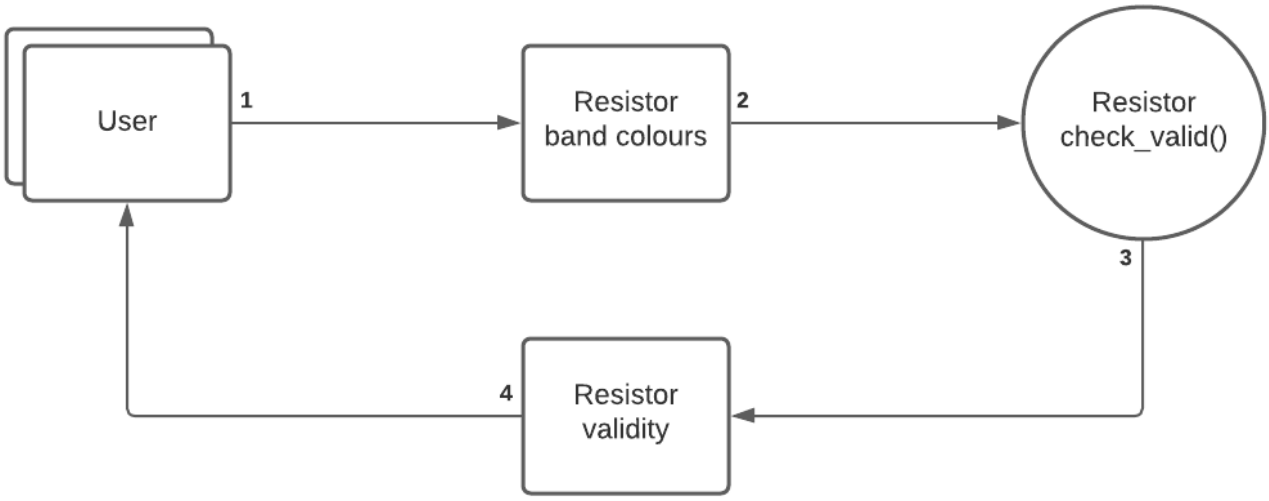


Figure 51 - Data flow diagram showing data flow between webserver and validator

1. The user overrides/selects a resistor band colour button on the webpage.
2. The selected bands make up the resistor band colours which are then sent to the Resistor class's check_valid function.
3. The result of this function is the validity of the resistor, which shows up on the webpage to the user.

Detector

The Detector class acts as the "main" class for the program and is responsible for managing for running the other classes within the program. I have created a data flow diagram for the flow of data for this class, this can be seen below.

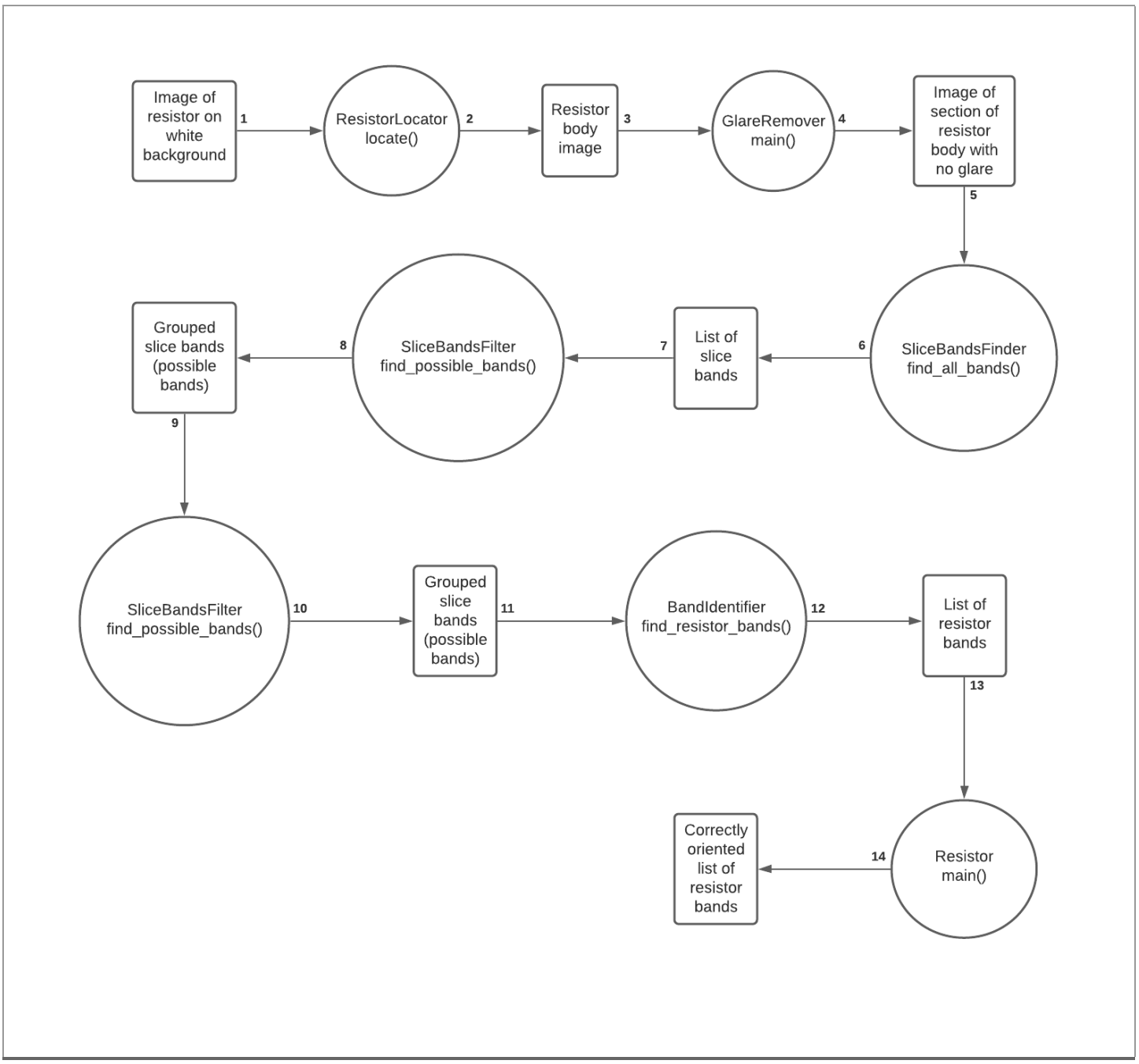


Figure 52 - Data flow diagram for the detector (main) class

1. Firstly, I pass an instance of the Image class whose image attribute contains a matrix that represents the initial resistor image into the ResistorLocator class and run the locate (main) function.
2. I then monochrome and erode the image to obtain the image of the resistor body.
3. The image attribute of the instance of the image class now contains an image of a resistor body. I pass this instance of the image class into the GlareRemover class and run the main function.

4. I identify glare within the resistor body image using K-Means to identify glare pixels. I then remove pixels that are marked as glare to obtain the no glare resistor body image.
5. I pass this no glare image into the SliceBandsFinder class and run the find_all_bands (main) function.
6. Using HSV masking, I obtain the masks for different resistor colours, which I take the bounding rectangles of. I initialize instances of the SliceBand class with these bounding rectangles and colours and add these instances to the list attribute of the SliceBands class.
7. I pass the instance of the SliceBands class that contains the list of SliceBand instances into the SliceBandsFilter class and run the find_possible_bands function.
8. I remove obvious outliers and then use K-Means to identify the x coordinates at which there are clusters of slice band x centres (bounding rectangle x + half the bounding rectangle width). I identify SliceBand instances within the list of SliceBands that meet certain conditions and add them to their corresponding slice band group (where each group represents the possible slice bands for a resistor band).
9. I pass these grouped slice bands into the BandIdentifier class and run the find_resistor_bands (main) function.
10. I count the colour frequency of the slice bands in each group and then apply weights to these frequencies. After obtaining these colour frequencies, I use majority voting on each group to find the colour that represents a group. For each colour, I initialize and instance of the ResistorBand class and append it to a list of resistor bands.
11. I pass the list of resistor bands into the Resistor class and run the main function.
12. After finding the orientation of the resistor that is valid, the list of resistor bands within the Resistor class instance contains a list of correctly oriented resistor bands. This instance of the resistor class is returned to where the detector class detect function was called.

Test Plan

All tests correspond to an objective and for each test I have described the test to be carried out, the data type used and the expected outcome. I have ensured that I have covered all objectives with these tests.

These tests will be identical for the tests used for the final testing for my program and they can be seen below. If my system is fully functional, all or almost all tests must pass.

Test Number	Objective	Description	Data / Type	Expected Outcome
1	1	Input images of resistors by selecting pre-existing files and scan.	Normal, from pre-existing files.	Webpage updates with correct resistor value displayed.
2	1	Input images of resistors via camera input and scan.	Normal, picture taken on the spot.	Webpage updates with correct resistor value displayed.
3	1	Input image of blank page and scan.	Erroneous, no resistor in image.	Webpage updates with error saying no resistor could be in the image.
4	2	Input images of resistors by selecting pre-existing files and scan.	Normal, from pre-existing files.	Webpage updates with correct resistor band colour buttons selected.
5	2	Input images of resistors via camera input and scan.	Normal, picture taken on the spot.	Webpage updates with correct resistor band colour buttons selected.
6	2	Input image of blank page and scan.	Erroneous, no resistor in image.	Webpage updates with error saying no resistor could be located within the image.
7	3	Input image of resistor via camera roll, scan and then change the selected resistor band colour buttons.	Normal, from pre-existing file.	After scanning, webpage updates with returned resistor values from initial image. After correcting bands, observe that the resistor value and validity status are updated.
8	3	Input image of resistor via camera roll, scan and then change the	Erroneous, returned resistor value is not	After scanning, webpage updates with returned resistor values from initial image. After

Test Number	Objective	Description	Data / Type	Expected Outcome
		selected resistor band colour buttons.	fully accurate.	correcting bands, observe that the resistor value and validity status are updated.
9	4	Select various valid and invalid combinations of resistor values using the resistor band colour buttons.	Normal.	Resistor value and validity status are updated when the resistor band colour values change.
10	5	Input images of resistors by selecting pre-existing files and scan.	Normal, form pre-existing files.	The resistor values and resistor band colour buttons are both output after scanning the image.
11	6	Input images of resistors by selecting pre-existing files and scan.	Normal, form pre-existing files.	Output resistor value matches the resistor when it is in its correct orientation with its validity being verified.
12	6	Input images of resistors via camera input and scan.	Normal, picture taken on the spot.	Output resistor value matches the resistor when it is in its correct orientation with its validity being verified.
13	6	Input pre-existing image of problematic resistor and scan.	Erroneous, returned resistor value is not fully accurate.	Incorrect output resistor value has its validity being declared as wrong.
14	6	Input image of blank page and scan.	Erroneous, no resistor in the image.	Output values are reset to their default, all resistor band colour buttons are deselected and the validity status is reset to invalid.

Test Number	Objective	Description	Data / Type	Expected Outcome
15	7	Select various valid and invalid combinations of resistor values using the resistor band colour buttons.	Normal.	Output resistor values are displayed with their correct unit.
16	8	Open the HTML page without running the webserver and select various resistor values using the resistor band colour buttons.	Normal.	Output resistor values are recalculated when the resistor band colour buttons are changed.
17	9	Input pre-existing resistor images that have the resistors in different locations and scan.	Normal, from pre-existing files.	Output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
18	9	Input images of resistors via camera input with the resistors in different locations and scan.	Normal, picture taken on the spot.	Output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
19	10	Input images of resistors via camera input with the resistor being different orientations and scan.	Normal, picture taken on the spot.	Output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
20	11	Input pre-existing resistor images that have different amounts of bands and scan.	Normal, from pre-existing files.	Output resistor values are correct, verified as valid and the resistor band colours selected correspond to the

Test Number	Objective	Description	Data / Type	Expected Outcome
				resistor in its correct orientation.
21	11	Input images of resistors that have different amounts of bands via the camera input.	Normal, picture taken on the spot.	Output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
22	12	Access application on desktop computer, input pre-existing resistor image and scan.	Normal, from pre-existing file.	Webpage displays correctly, output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
23	12	Access application on phone, input pre-existing resistor image and scan.	Normal, from pre-existing file.	Webpage displays correctly, output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
24	12	Access application on phone, use camera input to input resistor image and scan.	Normal, picture taken on the spot.	Webpage displays correctly, output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
25	13	Access application on device running Windows, input pre-	Normal, from pre-existing file.	Webpage displays correctly, output resistor values are correct, verified as valid and the

Test Number	Objective	Description	Data / Type	Expected Outcome
		existing resistor image and scan.		resistor band colours selected correspond to the resistor in its correct orientation.
26	13	Access application on device running Android, input pre-existing resistor image and scan.	Normal, from pre-existing file.	Webpage displays correctly, output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
27	13	Access application on device running Android, use camera input to input resistor image and scan.	Normal, picture taken on the spot.	Webpage displays correctly, output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
28	13	Access application on device running iOS, input pre-existing resistor image and scan.	Normal, from pre-existing file.	Webpage displays correctly, output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
29	13	Access application on device running iOS, use camera input to input resistor image and scan.	Normal, picture taken on the spot.	Webpage displays correctly, output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.

Test Number	Objective	Description	Data / Type	Expected Outcome
30	14.1	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure black resistor bands are correctly detected over 80% of the time.
31	14.2	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure brown resistor bands are correctly detected over 80% of the time.
32	14.3	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure red resistor bands are correctly detected over 80% of the time.
33	14.4	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure orange resistor bands are correctly detected over 80% of the time.
34	14.5	Run automated testing from within the program on a collection of pre-existing images that contains resistors that	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure yellow

Test Number	Objective	Description	Data / Type	Expected Outcome
		have all the possible resistor band colours.		resistor bands are correctly detected over 80% of the time.
35	14.6	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure green resistor bands are correctly detected over 80% of the time.
36	14.7	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure blue resistor bands are correctly detected over 80% of the time.
37	14.8	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure violet resistor bands are correctly detected over 80% of the time.
38	14.9	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure grey resistor bands are correctly detected over 80% of the time.
39	14.10	Run automated testing from within the program	Normal, from pre-	Application console prints out expected

Test Number	Objective	Description	Data / Type	Expected Outcome
		on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	existing files.	resistor values next to detected resistor band values. Ensure white resistor bands are correctly detected over 80% of the time.
40	14.11	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure gold resistor bands are correctly detected over 80% of the time.
41	14.12	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure silver resistor bands are correctly detected over 80% of the time.

Implementation

Skills Used

Skill Used	Location
File(s) organized for direct access.	103-107
Complex mathematical model.	113-121, 123-130,
Merge sort or similarly efficient sort.	108-110
Recursive algorithms.	108-110, (125-128)
Complex user-defined use of object-orientated programming (OOP) model.	62-77, Appendix
Client-server model.	149-150

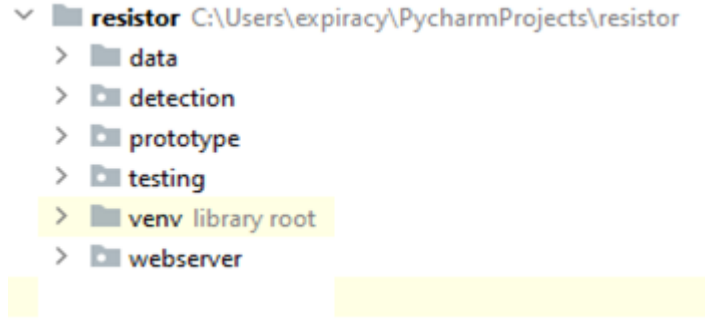
The main complexity of my program can be found within the resistor locator and band filter. Both make use of mathematical models to help solve problems, with the resistor locator making use of coordinate geometry to find the knee of the curve to find an optimal number of iterations and the band filter making use of my implementation of K-Means.

Despite these being the main places that the complexity can be observed, many of the other algorithms within my program still solve difficult problems such as thinking of the resistor locator algorithm.

File Structure

Overall Project Structure

To make it easier to navigate around my code, I have broken the system down into different categories of file that I put into different folders. The overall structure of the python project is shown below:



data: location of data files used in the detection system

detection: main resistor detector source code area

testing: test harness sources and test data

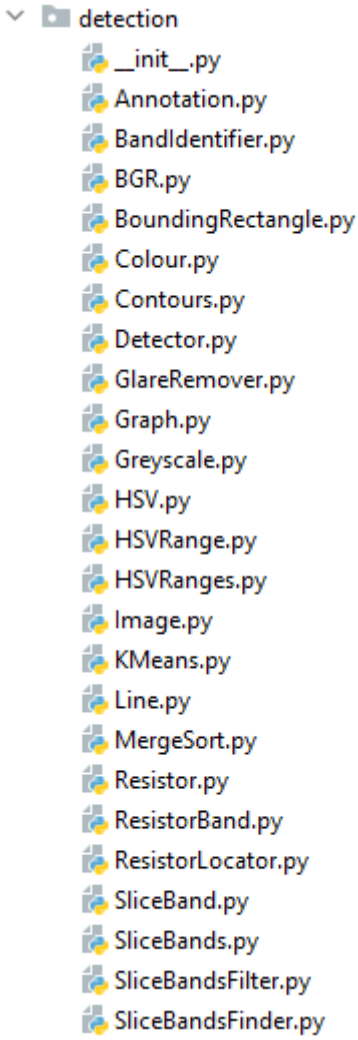
venv: standard python library folder

webserver: Webserver source code and resources (HTML /CSS)

Figure 53 Overall folder structure

Detection Directory

The files within the detection directory can be seen below.



- Contains the resistor detector image processing classes
- "Main" class is Detector.py
- Detection related classes
  - Detector
  - Colour
  - Resistor
  - ResistorBand
  - ResistorLocator
  - SliceBand
  - SliceBands
  - SliceBandFilter
  - SliceBandFinder
- Image manipulation classes
  - Image
  - BGR
  - HSV
  - Greyscale
- Utililty / Algorithm classes
  - KMeans
  - MergeSort
  - Line
  - Graph (used only to present data during development)

Figure 54 Detection Directory

Testing Directory

The file structure within the testing directory can be seen below.

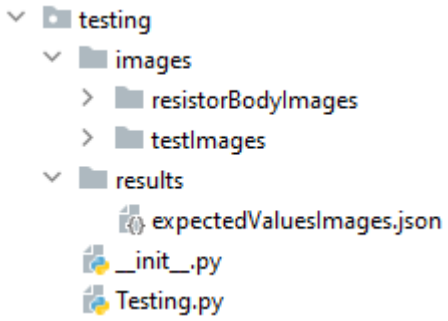
	**testing:** top level testing folder **images:** top level images subfolder **resistorBodyImages:** set of cropped resistor images for development testing **testImages:** set of resistor images for automated testing **results:** folder with expected results data

Figure 55 - Testing directory

Within the testing directory, I have an images folder. Within this folder, there is a folder of test images for the automated testing as well as well as cropped images with just the resistor body (these were used for debugging/development).

The results folder holds a json file that contains the expected results for each image within the testImages folder.

The Testing.py file contains the code to run the automated/module tests.

Webserver Directory

The file structure for the webserver directory can be seen below.

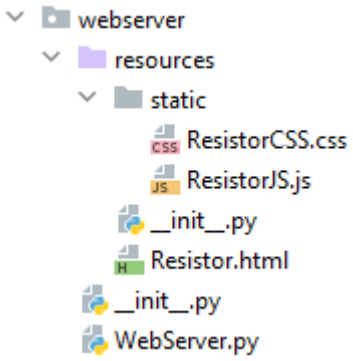
	**webserver:** top level folder with main webserver **resources:** subfolder with webserver resources **static:** standard subfolder name for static webserver resources

Figure 56 - Webserver directory

The WebServer.py file contains the code for the Flask webserver. The webpage hosted is the Resistor.html page within the resources folder.

The webpage requires the resources (CSS and js file) within the static folder within the resources directory to function correctly.

Data Folder

The structure for the data folder can be seen below.

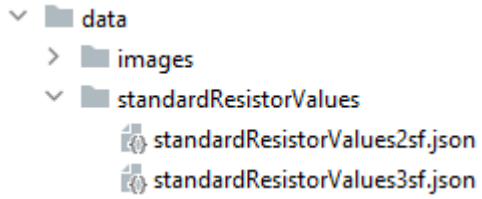
	**data:** top level folder for data **images:** folder for storing images uploaded for processing **standardResistorValues:** configuration files containing standard resistor values

Figure 57 - Data folder

The images folder is for saving the images input by the user for use in the resistor value detector.

The standardResistorValues folder contains 2 json files that contain the standard resistor values for different types of resistors.

Merge Sort

Purpose

Throughout the program I have needed to sort data within lists which is why it was suitable to write a merge sort algorithm. As mentioned in the research section, merge sort was a suitable choice over other sorts due to its efficiency – its worst time complexity is n*log(n) iterations.

Key Parts of Algorithm

As outlined in the research section, the merge sort algorithm sorts the data in a list through a technique called divide and conquer. After splitting the list until it is just singular items, you can sort and merge pairs of elements. After obtaining all the new equal sized merged lists, you can repeat this sorting and merging process until you end up with the full sorted list.

I have written a recursive method of carrying out this process.

Splitting the Data into Singular Elements

As mentioned above, you must firstly split the data up until it is 1 in length. I have achieved this by using a recursive call that returns the data value when it is 1 in length.

The sort function that can be seen in the code below is responsible for carrying out this splitting process, it takes data (in the form of a list) in as a parameter.

```
# The entry point for the sort
def sort(self, data, reverse=False):

    if len(data) <= 1:
        return data

    middle_index = len(data) // 2

    left = data[:middle_index]
    right = data[middle_index:]

    left = self.sort(left, reverse)
    right = self.sort(right, reverse)
```

Firstly, I check whether the length of the input list is already 1. If the length already is 1, the data is technically already sorted, and the list is returned. If not, the input data is split in half after finding the middle index. This gives me the left and right sides of the list.

After obtaining the left and right sides of the list, I call the sort function again for both of them. Until the length of the data is 1 (the base case), nothing be returned.

When the length of the data is 1, it means that the original list has been successfully split up.

Merging and Sorting the Left and Right Lists

When values for the left and right list have a returned value, the 2 lists can be sorted and then merged.

Below is a diagram for this process of sorting and merging the lists.

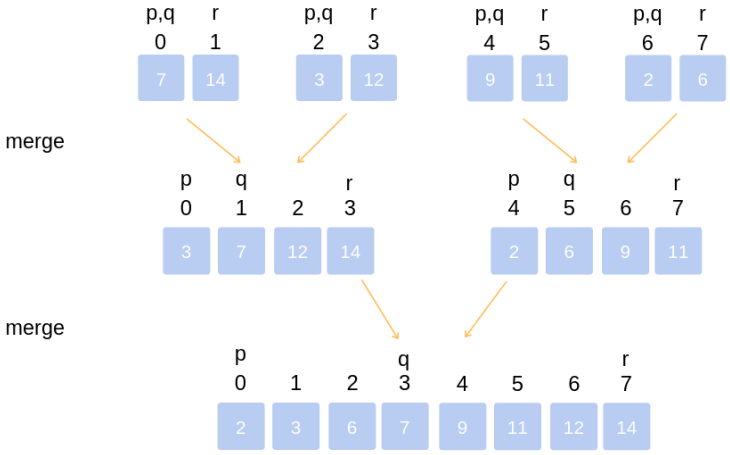


Figure 58 - Sorting and merging the data (Merge Sort Algorithm, 2021)

To sort the items, I make a comparison between the items of data in the 2 different lists. My code implentation of this can be seen below, with the sort_and_merge_lists function. This function takes in 2 lists and whether or not the sort should be reversed.

```
# Responsible for sorting and merging the lists
def sort_and_merge_lists(self, left, right, reverse):

    left_index = 0
    right_index = 0

    sorted_and_merged_list = []

    while left_index < len(left) and right_index < len(right):

        if reverse:
            if left[left_index] > right[right_index]:
                sorted_and_merged_list.append(left[left_index])
                left_index += 1

            else:
                sorted_and_merged_list.append(right[right_index])
                right_index += 1

        else:
            if left[left_index] < right[right_index]:
                sorted_and_merged_list.append(left[left_index])
                left_index += 1

            else:
                sorted_and_merged_list.append(right[right_index])
                right_index += 1

    sorted_and_merged_list.extend(left[left_index:])
    sorted_and_merged_list.extend(right[right_index:])

    return sorted_and_merged_list
```

Firstly, I have defined left_index and right_index as 0. These indexes will allow me to access the correct data items for the comparisons between items in the left and right lists.

I use a while loop to iterate through the data and order the items into a newly defined list (sorted_and_merged_list). This while loop runs if the left and right index values are both less than the length of their corresponding lists.

The comparisons are done inside of the while loop. Before carrying out a comparison between items, I check if the sort should be reversed. If the sort is reversed, the comparison I make is whether or not the item in the left list at the left index is bigger than the item in the right list at the right index rather than comparing whether it is smaller (when the sort is not reversed).

After the comparison takes place, I append the larger item (if the sort is reversed) or the smaller item (if the sort is not reversed) to sorted_and_merged_list. I then add 1 to the index that corresponds to the side the item that was added to the list is in.

When the while loop has stopped, the sorted_list variable contains a list containing the sorted, merged data and sorted_and_merged_list is returned.

Recursion Calls

Due to the way that the sort function was called in the first place, sorted_and_merged_list will return as the result of the left or right variable multiple times depending on the amount of times it was called (the length of the initial list).

```
left = self.sort(left, reverse)
right = self.sort(right, reverse)

sorted_and_merged_list = self.sort_and_merge_lists(left, right, reverse)

return sorted_and_merged_list
```

This means that the returned results of the sort function after the initial splitting are the left and right lists which were merged and sorted (sorted_list). The sort_and_merge function is then called repeatedly with these returned results for the left and right lists. When the returned list is the same length as the original list, there are no more recursive calls to return to and the value returns to the initial place at which it was called.

Resistor Locator Algorithm

Purpose

The resistor locator algorithm is responsible for extracting just the resistor body from an image. This is crucial within my program as it ensures that the other parts of my application are working upon the correct parts of the image. This algorithm was decided upon after much experimentation and observation that can see in the evaluation section of this document.

Key Parts of Algorithm

The algorithm that I have developed to locate the resistor within an image goes through several stages before locating the resistor.

Background Identification

The Initial Image

The data that is passed into the resistor locator is an instance of the Image class that contains a matrix that represents the resized version of the input image. The contents of this image are the resistor body and its wires on a white background. The purpose of resizing the original image is to reduce the amount of time that it takes to process the image and locate the resistor body.

Figure 59 - Initial image of resistor

Identifying the Background

Firstly, due to the nature of the image, you can invertedly monochrome the image with specific thresholds to distinguish between the resistor body and its wires from the background. As my program requires users to put the resistors on a white piece of paper before they can be scanned, a well-defined monochrome threshold can differentiate a resistor body and its wires from the background.

Before converting the image to a monochrome, I have reinitialized the image matrix with the Greyscale class as you can only monochrome a greyscale image. The code below shows the thresholds I have used for the monochrome as well as the function that carries out the monochrome.

```
greyscale_image = Greyscale(self.image.image, 'BGR')

monochrome_image = greyscale_image.monochrome(inverted=True, block_size=151, C=21)
```

```
# Converts an image to monochrome.
def monochrome(self, inverted=False, block_size=51, C=21):
    try:
        thresholding = cv2.THRESH_BINARY if not inverted else cv2.THRESH_BINARY_INV

        self.image = cv2.adaptiveThreshold(self.image, 255, cv2.ADAPTIVE_THRESH_MEAN_C, thresholding,
block_size, C)

        return self

    except:
        raise Exception("Error converting image to monochrome")
```

After this inverted monochrome image has been obtained, I can find the contours of the image. This is done using the find_contours function.

This is done because it is likely that the monochrome image has set both the glare on the resistor as well as the background to 0. Obtaining the contours of the image can then be used to fill in the resistor. Below is the output of the monochrome function and you can see this exact issue, with the glare on the resistor being set to a value of 0.

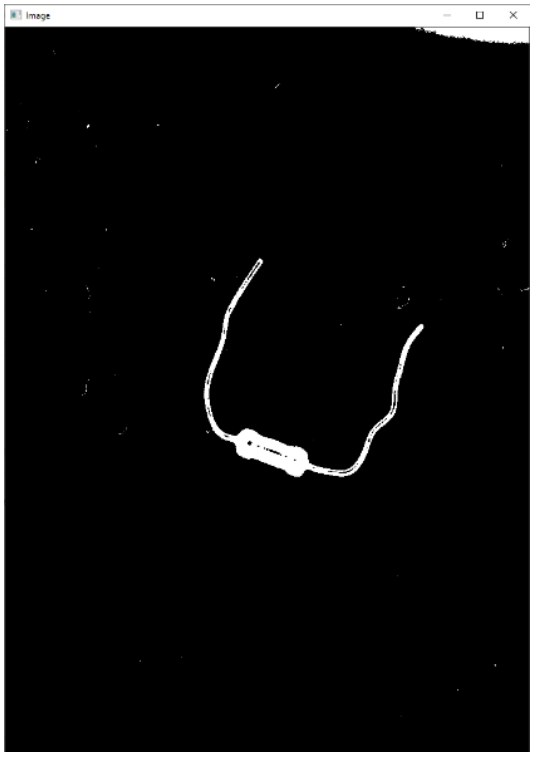


Figure 60 - Monochrome image

Drawing these contours onto the image with the cv2.FILLED mode addresses this issue. This cv2.FILLED function allows me to fill the area bounded by the contours if thickness < 0. The image below is the result of drawing the contours onto the image with the cv2.FILLED mode and the resistor has been filled in with white after using this function. This is important as the next stage erodes the image from where values are 0.

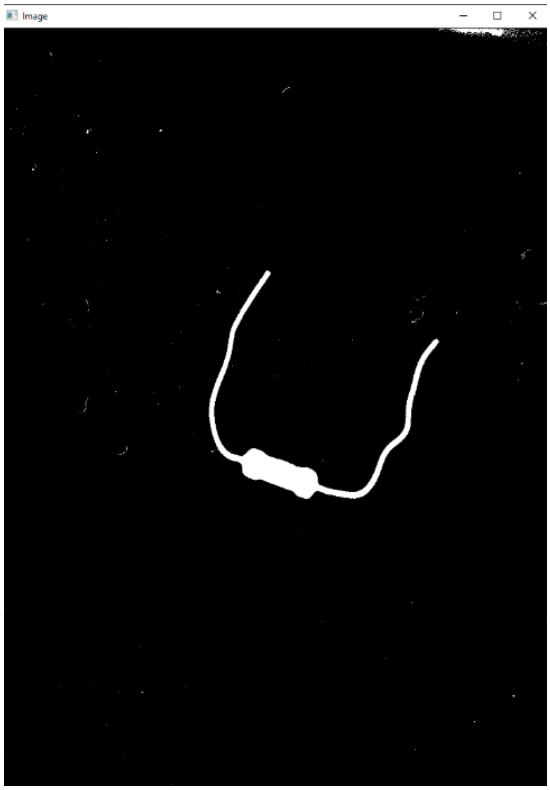


Figure 61 - Monochrome image with contours drawn

Locating the Resistor Body

Within this monochrome image, I can now locate the image as at this stage, the resistor and its wires are clearly distinguished from the resistor background.

To locate the resistor, I can erode the image. The basic concept of this image erosion is like that of soil erosion. Soil erosion erodes away the edges of the soil, similarly to how image erosion will erode the edges of the lighter regions of the image. This erosion process erodes speckles of noise in the background as well as the wires of the resistor to leave just the resistor body.

How the erosion is applied depends on what the matrix representing the structuring element (kernel) is set to. The structuring element I have opted to use is a 3x3 kernel that is full of 1s with the centre as the anchor point.

Eroding the Wires

Every time erosion is applied to the image, the resistor reduces in area. The resulting image, after 1 erode iteration can be seen below.

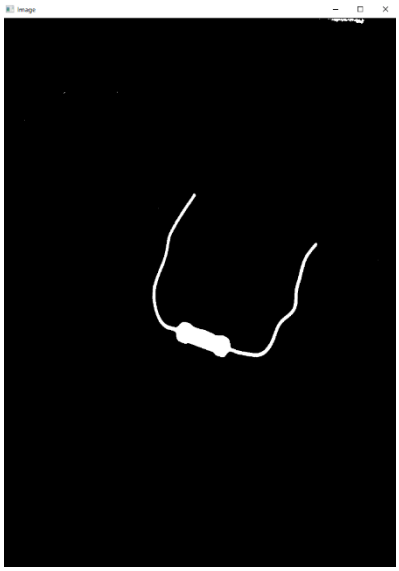


Figure 62 - Monochrome image after 1 erode iteration

As you can see, the noise in the background have been eroded away and the resistor wires have got skinnier.

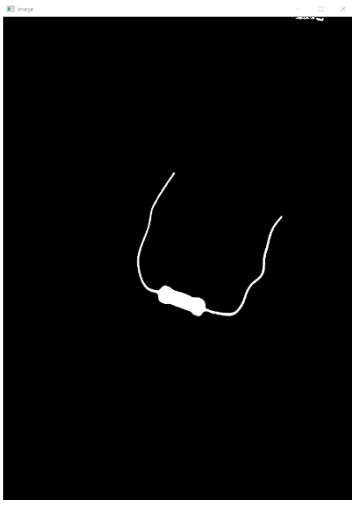


Figure 63 - Monochrome image after 2 erode iterations

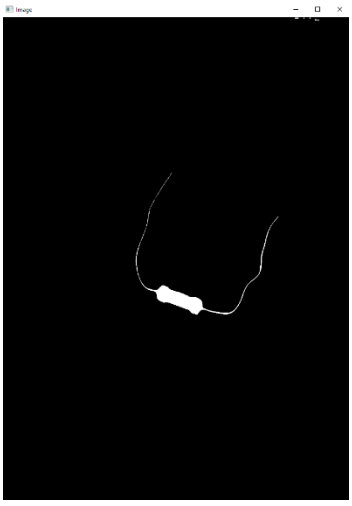


Figure 64 - Monochrome image after 4 erode iterations

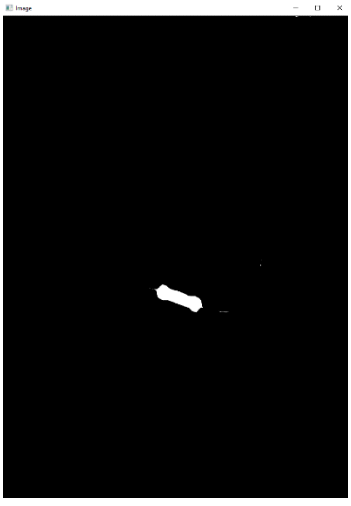


Figure 65 - Monochrome image after 6 erode iterations

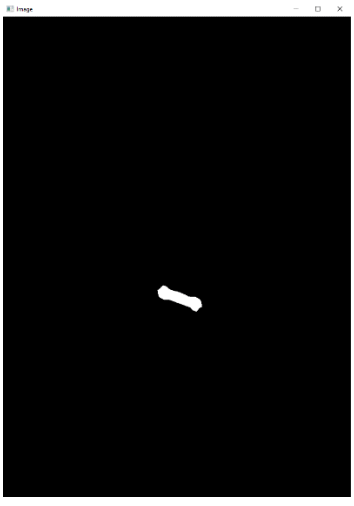


Figure 66 - Monochrome image after 8 erode iterations

To locate just the resistor body, I need to find the number of erosions needed to erode the wires. I can assume that the contour that produces the largest area within the monochrome image is likely the resistor so for each new image produced after an erosion, I have found the contour with the largest area, squared its area, and appended it to a list. The point near large decreases of the area

followed by diminishing returns is likely to be close to the number of iterations needed to erode the resistor wires.

Below is a graph of the squared contour areas against the iteration number that I have acquired for the resistor that can be seen below.

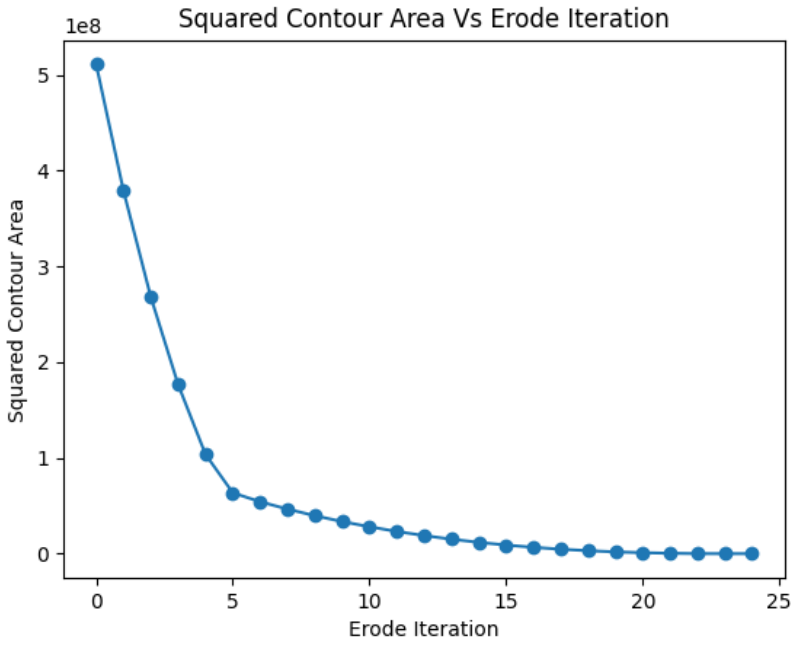


Figure 67 - Graph of squared contour area against erode iteration

From this graph, we can see when the when the number of erode iterations is 5, we start to see diminishing returns by eroding the image more. When the number of iterations is 5, the resultant image seen below is produced.

Figure 68 - Monochrome image after 5 erode iterations

The code responsible for obtaining the values of the largest square contour areas can be seen below. It takes in the Greyscale (Image) class instance that contains the monochrome image as well as the initial contours of the image as its parameters.

```
# Finding the amount of erosion needed to remove the resistor wires.
def find_erode_iterations(self, image, contours):
    try:
        image = Greyscale(image.image)

        biggest_initial_contour = Contours(contours).find_biggest()

        biggest_initial_contour_area = cv2.contourArea(biggest_initial_contour)

        squared_contour_areas = [biggest_initial_contour_area ** 2]

        empty_image = False

        while not empty_image:

            eroded_image = image.erode(1)

            if eroded_image.count_non_zero_pixels() != 0:

                contours, _ = eroded_image.find_contours()
                biggest_contour = Contours(contours).find_biggest()

                contour_area = cv2.contourArea(biggest_contour)
                squared_contour_areas.append(contour_area ** 2)

            else:
                empty_image = True
```

Firstly, I find the area of the biggest contour in the initial input monochrome image before any erosion and append its area to the squared_contour_areas list.

I have then used a while loop that only runs again if the image while it is not empty, this is to avoid errors of eroding an empty image.

For each iteration of the while loop, I test whether the image is empty by checking if the number of non-zero pixels is greater than 0.

If the image is not empty, I find the contours for the image and identify the biggest one. I then append the squared area of this biggest area to a list called squared_contour_areas.

Using the Elbow Method to Find Optimal Erode Iterations

A reason for using the Elbow Method is that my program more versatile when locating the image as the amount of erode iterations changes depending on the nature of the input image.

I can clearly see that the optimal number of erode iterations must be near the point 5, and I know that I can identify the point 5 easily at it is the Knee of the graph.

I have developed a method of identifying the knee of the graph using coordinate geometry. This is all done in the Line class.

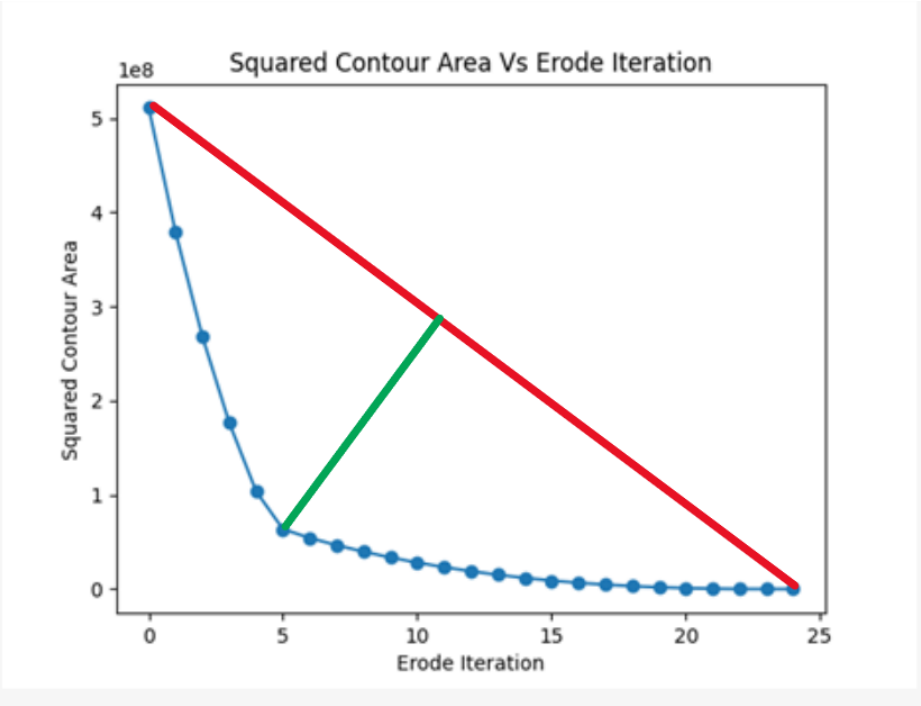


Figure 69 - Annotated graph

red = spine line

green = largest perpendicular distance

As seen in the image above, you can see that the knee can be identified by calculating the point that produces the largest perpendicular distance from the spine line.

The code below is responsible for finding this largest perpendicular distance (the knee), it takes in the list of the erode iteration with its respective squared largest contour areas as the parameter.

```
# Finds the knee of data.
def find_knee(self, points):
    try:
        point_1 = points[0]
        point_2 = points[len(points) - 1]

        spine = self.from_points(point_1, point_2)

        intersection_distances = []

        for current_point in points:
            current_line = self.from_gradient(spine.normal(), current_point)

            intersection = current_line.find_intersection(spine)

            distance = current_line.length(intersection, current_point)

            intersection_distances.append(distance)

        knee = intersection_distances.index(max(intersection_distances))

        return knee

    except Exception as error:
        print(f'Error finding knee: {error}')
```

Firstly, I have extracted the first and last points called point_1 and point_2 from the list of points.

I have then obtained the line that represents the spine this is done using the from_points function that allows me to create a line from 2 points.

The code for the from_points function can be seen below.

```
# Make a line from 2 points.
def from_points(self, point_1, point_2):
    try:
        line = Line()

        line.gradient = self.find_gradient(point_1, point_2)
        line.constant = point_1[1] - line.gradient * point_1[0]

        return line

    except:
        raise Exception(f"Could not create line from points: {point_1} and {point_2}")
```

This function initializes a new instance of the line class and assigns this new instance a gradient and a constant.

The constant of the line is found using the equation:

$$c = y - mx$$

This can be seen in the find_constant function below where:

y = point_1[1]

m = self.gradient

x = point_1[0]

```
# Find the constant value of the line.
def find_constant(self, point):
    if self.gradient is not None:
        return point[1] - self.gradient * point[0]

    else:
        return None
```

The gradient of the line is found using the equation:

$$\frac{difference\ in\ y}{difference\ in\ x}$$

The code implementation for the find_gradient function can be seen below where:

difference in x = point_2[0] - point_1[0]

difference in y = point_2[1] - point_1[1]

```
# Find the gradient of a line from 2 points.
def find_gradient(self, point_1, point_2):
    try:
        dx = (point_2[0] - point_1[0])
        dy = (point_2[1] - point_1[1])

        gradient = dy / dx

        return gradient

    except ZeroDivisionError:
        print('Line is vertical.')
        return None
```

As we now have an instance of the Line class for the spine line, the gradient of the perpendicular line (normal) can be found using the equation:

$$-\frac{1}{m}$$

The value for the gradient can be returned if the line has a gradient within my code which can be seen below where:

m = self.gradient

```
# Returns the normal gradient.
def normal(self):
    try:
        return -1 / self.gradient

    except:
        print('Line has no gradient.')
```

After calculating this normal gradient, it is possible to find line that passes through the data point and that is perpendicular to the spine line.

I have then iterated through all the points and created a new instances of the Line class for each point. This is done using the from_gradient function from this gradient and a point and can be seen below.

```
# Make a line from a point and a gradient.
def from_gradient(self, gradient, point):
    try:
        line = Line()

        line.gradient = gradient
        line.constant = line.find_constant(point_1)

        return line

    except:
        raise Exception(f"Could not create line from gradient: {gradient} and point: {point}")
```

Now that the spine line's perpendicular lines that pass through the data points have been created, it is possible to find the intersections of the lines using some simple algebra that can be seen below.

$$y = m_1x + c_1 \quad y = m_2x + c_2$$
$$m_1x + c_1 = m_2x + c_2$$
$$m_1x - m_2x = c_2 - c_1$$
$$x(m_1 - m_2) = c_2 - c_1$$
$$x = \frac{(c_2 - c_1)}{(m_1 - m_2)}$$

After obtaining this x coordinate, you can solve for y to get y using either line with the equation:

$$y = mx + c$$

My code implementation to find the x and y coordinates of the intersection of 2 lines can be seen below where:

c_2 = line_2.constant

c_1 = self.constant

m_2 = line_2.gradient

Figure 70 - Algebra to find x intersection of 2 lines

m_1 = self.gradient

```
# Finds the intersection of 2 lines.
def find_intersection(self, line_2):
    try:
        x = (line_2.constant - self.constant) / (self.gradient - line_2.gradient)
        y = self.solve_for_y(x)

        intersection = [x, y]

        return intersection

    except:
        raise Exception('Error finding intersection.')
```

```
# Solve the line for the y value from an x value.
def solve_for_y(self, x):
    if self.gradient is not None and self.constant is not None:
        return float(self.gradient) * x + float(self.constant)

    else:
        print('Can not solve on a vertical or horizontal line.')
```

After obtaining the coordinate of the intersection, I can calculate the length of a line between 2 points using the Euclidean distance formula. My implementation for this can be seen below:

```
# Finds the length of 2 lines between 2 points.
def length(self, point_1, point_2):
    try:
        dx_squared = (point_1[0] - point_2[0]) ** 2
        dy_squared = (point_1[1] - point_2[1]) ** 2

        distance = np.sqrt(dx_squared + dy_squared)

        return distance

    except:
        raise Exception('Error finding line length.')
```

I have calculated the distance between every point and its corresponding intersection and appended them to a list called intersection distances.

I can then find the index of the maximum value of this list to obtain the number of iterations that is at the knee of the graph.

Extracting the Resistor Body from the Image

Finding the Resistor Body Contour

After obtaining the value of the knee of the graph, I created a “safe knee” which is the value of the knee + 5. The reason for this safe knee is to fully ensure that only the resistor body is left over after the erode iterations.

I then proceed to erode the original image for erode iterations defined by the safe knee. After performing this number of erode iterations on the resistor image seen above, the that can be seen below is produced.

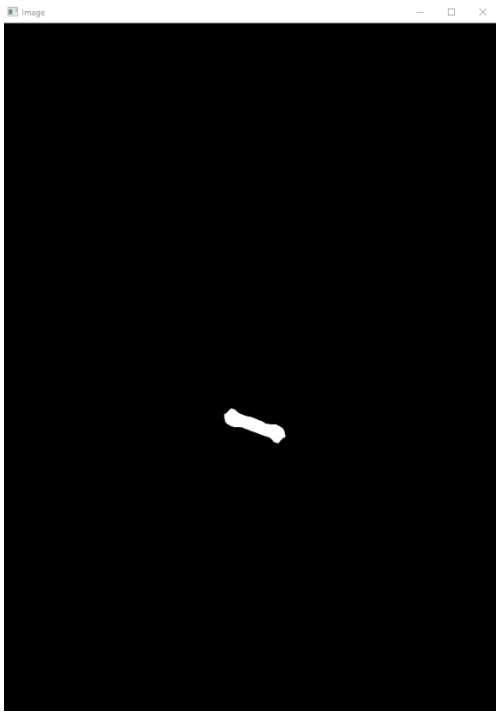


Figure 71 - Monochrome image after "safe knee" iterations

The code below is responsible for carrying out this purpose.

```
erode_iterations = self.find_erode_iterations(filled_image.clone(), contours)

eroded_image = filled_image.erode(erode_iterations)
```

After this image is acquired, I can then find the contours of the image where the biggest remaining contour will be the body of the resistor. I find this biggest contour with the code below.

```
# The biggest contour should only be the resistor body
contours, _ = eroded_image.find_contours()

resistor_body_contour = Contours(contours).find_biggest()

return resistor_body_contour
```

Extracting the Resistor Body Image from the Original Image

After obtaining the contour for the resistor body, I extract an image of just the resistor body.

I have done this with the code that can be seen below.

```
# Extracts the resistor from a rectangle
def extract_resistor(self, resistor_body_contour):
    try:
        resistor_rectangle = cv2.minAreaRect(resistor_body_contour)

        box_points = cv2.boxPoints(resistor_rectangle)
        box_points = np.int0(box_points)

        width = int(resistor_rectangle[1][0])
        height = int(resistor_rectangle[1][1])

        source_points = box_points.astype('float32')

        destination_points = np.array([[0, height - 1],
                                       [0, 0],
                                       [width - 1, 0],
                                       [width - 1, height - 1]], dtype='float32')

        matrix = cv2.getPerspectiveTransform(source_points, destination_points)

        self.image = self.image.warp_perspective(matrix, width, height)

        if self.image.width() < self.image.height():
            self.image = self.image.rotate_90_clockwise()

        return self.image

    except:
        raise Exception("Error trying to extract resistor.")
```

Firstly, I use the cv2 minAreaRect function to obtain the coordinates and angle of the minimum area rectangle that can fit into the contour.

I then rotate this minimum area rectangle until it has an angle of 0 degrees using source and destination points. As the angle of 0 degrees can allow the resistor to be vertical or horizontal, I ensure the resistor is oriented horizontally by comparing the height to the width. For the image to be horizontal, the width must be bigger than the height, if this is not the case, I rotate the image 90 degrees clockwise.

K-Means

Purpose

As described in the research section of this document, K-means is an algorithm that is used to split data into a k number of clusters by calculating the optimal centroids for the data.

Within my program, I have used K-Means to identify glare and locate slice bands (both of these implementations are mentioned later in this section).

Key Parts of Algorithm

Calculating the Euclidean Distance

I have calculated the Euclidean distance between 2 3d coordinates.

```
# Finds the distance between 2 points.
def find_distance(self, point_1, point_2):
    try:
        dx_squared = (point_1[0] - point_2[0]) ** 2
        dy_squared = (point_1[1] - point_2[1]) ** 2
        dz_squared = (point_1[2] - point_2[2]) ** 2

        distance = np.sqrt(dx_squared + dy_squared + dz_squared)

        return distance

    except:
        print(f"Error trying to find distance between {point_1} and {point_2}")
```

The parameters of this function are 2 points, labelled point_1 and point_2. These points are 3d coordinates which are represented in the form of a 1d list that is 3 in length.

Linking back to the equation for the 3d Euclidean distance:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Within the code:

dx_squared = $(x_2 - x_1)^2$

dy_squared = $(y_2 - y_1)^2$

dz_squared = $(z_2 - z_1)^2$

To get the Euclidean distance, I have then added dx_squared, dy_squared and dz_squared together and square rooted their sum.

Initializing the Centroids (K-means++)

The code below is responsible for finding the seeds for the fit function to use. As highlighted in the research section, using K-means++ to do this means that the algorithm will return better centroids for the data than if the initial centroids were just randomly selected.

The initialize_centroids function that can be seen below takes in the data as the parameter.

The first centroid is a point that is randomly selected from within the data.

```
# Initialises the seeds to be used for fit().
def initialize_centroids(self, data):
    number_of_items = len(data)

    centroids = {1: (data[rd.randint(0, number_of_items - 1)])}

    for centroid_number in range(1, self.number_of_centroids):
        minimum_intra_cluster_distances = self.find_minimum_intra_cluster_distances(data, centroids)

        next_centroid = self.find_next_centroid(data, minimum_intra_cluster_distances)

        centroids[centroid_number + 1] = next_centroid

    return centroids
```

Next, find the minimum intra cluster distance for each item in the data out of all the cluster centroids. This is done using the find_minimum_intra_cluster_distances and the find_intra_cluster_distances_for_centroids functions.

Below, the code for the find_intra_cluster_distances_for_centroids and find_intra_cluster_distances_for_centroids can be found.

```
# Finds the minimum intra cluster distances for each item.
def find_minimum_intra_cluster_distances(self, data, centroids):
    intra_cluster_distances_for_centroids = self.find_intra_cluster_distances_for_centroids(data, centroids)

    minimum_intra_cluster_distances = []

    for item_index in range(len(data)):
        distances_from_centroids = []

        for _, intra_cluster_distances in intra_cluster_distances_for_centroids.items():
            distances_from_centroids.append(intra_cluster_distances[item_index])

        minimum_intra_cluster_distances.append(min(distances_from_centroids))

    return minimum_intra_cluster_distances
```

```
# Finding the intra-clusters distances for centroids.
def find_intra_cluster_distances_for_centroids(self, data, centroids):
    intra_cluster_distances_for_centroids = {}

    for centroid_number, centroid in centroids.items():

        intra_cluster_distances = []

        for item in data:
            intra_cluster_distance = self.find_distance(centroid, item)
            intra_cluster_distances.append(intra_cluster_distance)

        intra_cluster_distances_for_centroids[centroid_number] = intra_cluster_distances

    return intra_cluster_distances_for_centroids
```

The find_intra_cluster_distances_for_centroids function calculates the intra cluster distance to every centroid for every item of data and return a dictionary.

The result of the find_intra_cluster_distances_for_centroids function can then be used to find the minimum intra cluster distances. I have appended all the intra cluster distances for an item to one list and found the smallest value of that list. This smallest value is then appended to the list of minimum intra cluster distances.

Below is the code for the find_next_centroid function. It takes in the data and the centroid distances as parameters.

```
def find_next_centroid(self, data, minimum_centroid_distances):
    minimum_centroid_distances = np.array(minimum_centroid_distances)

    index_of_highest_distance = np.argmax(minimum_centroid_distances)

    next_centroid = data[index_of_highest_distance]

    return next_centroid
```

The value that produced the largest intra cluster distance within the list of minimum intra cluster distances is likely to be the next centroid. Using the np.argmax function, I have obtained the index of this item and retrieve the data value at this index.

The data value at that index is then set to be the next centroid, and the algorithm keeps iterating through until all the centroids have been defined.

Identifying Cluster Centroids

The code below is responsible for identifying cluster centroids for the input data. This is done by:

- Labelling every item in the data is labelled with its closest centroid.
- Grouping the data by these labels.

```
# K-means algorithm to find the labels and centroids based on the centroids.
def fit(self, data, centroids, iteration=0, max_iterations=15):
    try:
        self.centroids = {}
        self.labels = []

       intra_cluster_distances_for_centroids = self.find_intra_cluster_distances_for_centroids(data, centroids)

        self.find_labels(data, intra_cluster_distances_for_centroids)

        data_grouped_by_label = self.group_data_by_label(data)

        self.move_centroids(data_grouped_by_label)
        iteration += 1
        centroids_moved = self.check_if_centroids_moved(centroids)

        if iteration == 1 or centroids_moved:
            if iteration <= max_iterations:
                return self.fit(data, self.centroids, iteration)
            else:
                return self

        else:
            return self

    except:
        print("Error with KMeans fit, value of K is too large")
```

Moving the centroids

Before the centroids can be moved, the data must firstly be labelled and grouped.

To label the data, calculate the distance between every item in the data and the initial centroids. The centroid that results in the minimum distance is the label that the item will receive.

The function that calculates the distance between every item in the data and every initial centroid is the find_intra_cluster_distances_for_centroids function. It takes in the data and the initial centroids as parameters and returns a dictionary where the keys are the centroid number, and the value is intra cluster distance for every item to that centroid.

```
# Finding the intra-clusters distances for centroids.
def find_intra_cluster_distances_for_centroids(self, data, centroids):
```

```
    intra_cluster_distances_for_centroids = {}

    for centroid_number, centroid in centroids.items():

        intra_cluster_distances = []

        for item in data:
            intra_cluster_distance = self.find_distance(centroid, item)
            intra_cluster_distances.append(intra_cluster_distance)

        intra_cluster_distances_for_centroids[centroid_number] = intra_cluster_distances

    return intra_cluster_distances_for_centroids
```

The function that then finds the labels is the find_labels function. It takes the result of the find_intra_cluster_distances_for_centroids function as a parameter.

```
# Labelling items with the number of their closest centroid.
def find_labels(self, data, intra_cluster_distances_for_centroids):
    self.labels = []

    for item_index in range(len(data)):
        distances_from_centroids = []

        for _, intra_cluster_distances in intra_cluster_distances_for_centroids.items():
            distances_from_centroids.append(intra_cluster_distances[item_index])

        self.labels.append(np.argmin(distances_from_centroids) + 1)
```

The code above determines the closest centroid by identifying the cluster centroid that has the smallest distance out all the intra cluster distances for that item.

After the labels for the data have been identified, the data can then be grouped by their labels.

The function that groups the data is the group_data_by_label function.

```
# Grouping the data by the labels.
def group_data_by_label(self, data):
    data_grouped_by_label = {}

    for centroid_number in range(1, self.number_of_centroids + 1):
        data_grouped_by_label[centroid_number] = []

    for centroid_number in range(1, self.number_of_centroids + 1):
        for index, label in enumerate(self.labels):
            if label == centroid_number:
                data_grouped_by_label[centroid_number].append(data[index])

    return data_grouped_by_label
```

This function returns a dictionary that assigns each item to their corresponding label. The grouped values for each centroid number share a common closest centroid.

The centroids can finally be moved after the data has been grouped by their labels.

The move_centroids function is responsible for doing this and takes in the result of the group_data_by_label function.

```
# Moving centroids by calculating the mean of the data grouped by centroid.
def move_centroids(self, data_grouped_by_label):
    for centroid_number in range(1, len(data_grouped_by_label) + 1):
        x_total = 0
        y_total = 0
        z_total = 0

        for item in data_grouped_by_label[centroid_number]:
            x_total += item[0]
            y_total += item[1]
            z_total += item[2]
```

```
        number_of_items = len(data_grouped_by_label[centroid_number])

        x_mean = x_total / number_of_items
        y_mean = y_total / number_of_items
        z_mean = z_total / number_of_items

        self.centroids[centroid_number] = [x_mean, y_mean, z_mean]
```

As the data is now grouped by their common closest centroids, finding the mean of the x, y, and z values individually will give the x, y, and z coordinates of the new centroid. Update the corresponding self.centroid value with the new values of the centroid.

Recursion and Conditions

Using the function once on the data may not provide the best cluster centroids. This is because the shift in the centroids could result in the items in the data being assigned different centroids if the algorithm was running again with the new centroids as the initial centroids. To achieve the best centroids possible, I have written this function recursively which means it calls itself until certain conditions are met. When these conditions are met, I know that the centroids acquired by my program are good enough to be considered optimal for the data and I can stop running the algorithm.

The conditions for the recursion to stop is that either if the centroids have not moved (and the program is on an iteration higher than 1) or the defined maximum number of allowed iterations is reached.

Below is the part of the code that is checks for these conditions. On line 7, is the recursive call where the new centroids (stored in self.centroids) are used as the centroid argument for the recursive call. This recursive call only runs if the iteration number is 1 or if the centroids have moved. The code responsible for checking for these conditions can be seen below.

```
iteration += 1

centroids_moved = self.check_if_centroids_moved(centroids)

if iteration == 1 or centroids_moved:
    if iteration <= max_iterations:
        return self.fit(data, self.centroids, iteration, max_iterations)
    else:
        return self

else:
    return self
```

The code below is responsible for checking if the centroids have moved.

```
def check_if_centroids_moved(self, centroids):
    inter_cluster_distances = []

    for centroid_number in range(1, self.number_of_centroids + 1):
        inter_cluster_distance = np.subtract(self.centroids[centroid_number], centroids[centroid_number])

        inter_cluster_distance_sum = np.sum(inter_cluster_distance)

        inter_cluster_distances.append(abs(inter_cluster_distance_sum))

    inter_cluster_distances_sum = sum(inter_cluster_distances)

    if inter_cluster_distances_sum != 0:
        return True
```

```
        else:
            return False
```

This check_if_centroids_moved function takes the initial centroids as a parameter. These initial centroids are then compared to the newly calculated centroids which are stored in the self.centroids variable.

To determine whether the centroids has moved, firstly, calculate the and take the absolute value of the inter cluster distance between the initial centroids and their corresponding new centroid values by subtracting them from one another. If the sum of absolute values of the inter cluster distances is 0, then the centroids have not moved as there is no difference between the initial and new centroid values and vice versa.

Obtaining the Optimal Number of Centroids (Dunn Index)

I have used the Dunn Index to identify the optimal number of centroids that are to be used. This is important within my program as this is used to determine the number of bands the resistor has if the user does not choose to specify that themself.

As outlined in my research section, the Dunn Index is given by the formula:

$$Dunn\ Index = \frac{minimum\ inter\ cluster\ distance}{maximum\ intra\ cluster\ distance}$$

The code below carries out this calculation on multiple values of k and identifies the optimal value.

```
# Finds the optimal number of clusters using Dunn-index.
def find_optimal_number_of_clusters(self, data):
    dunn_indexes = []

    for centroid_amount in range(3, 7):
        self.centroids = {}
        self.number_of_centroids = centroid_amount
        self.fit(data, self.initialize_centroids(data))

        data_grouped_by_label = self.group_data_by_label(data)

        maximum_intra_cluster_distance = self.find_max_intra_cluster_distance(data_grouped_by_label)

        minimum_inter_cluster_distance = self.find_min_inter_cluster_distance()

        dunn_index = minimum_inter_cluster_distance / maximum_intra_cluster_distance

        dunn_indexes.append(dunn_index)

    optimal_number_of_clusters = np.argmax(dunn_indexes) + 3

    return optimal_number_of_clusters
```

By running the K-Means fit function on the data for various values for K and rating the resultant centroids, I can identify the optimal number of centroids. As resistors can only have 3 to 6 bands, I have chosen to run the algorithm for values of k in the range 3 to 6 (inclusive).

At the start of each iteration, I have reset the self values as I am utilizing the same class instance when running the fit function. After resetting these values, I have called the fit function with the data and the seeds (obtained using the initialize_centroids function described above).

After running the fit function, I can then obtain the 2 values needed to calculate the Dunn Index – the minimum inter cluster distance and the maximum intra cluster distance.

Finding the maximum intra cluster distance

The maximum intra cluster distance for a value of k is obtained with the find_max_intra_cluster_distance function that can be seen below. The function takes in the data grouped by label as a parameter.

```
# Finds the maximum intra-clusters distance.
def find_max_intra_cluster_distance(self, data_grouped_by_label):
    maximum_intra_cluster_distance = -np.inf

    for centroid_number, data in data_grouped_by_label.items():
        for item in data:
            intra_cluster_distance = self.find_distance(item, self.centroids[centroid_number])

            maximum_intra_cluster_distance = max(maximum_intra_cluster_distance, intra_cluster_distance)

    if maximum_intra_cluster_distance == 0:
        return 1
    else:
        return maximum_intra_cluster_distance
```

Initially, I have set the maximum intra cluster distance to negative infinity so that it is guaranteed that it will get overwritten by the next value. I have then found the intra cluster distance from each point its closest centroid and then used the max function to compare and determine the maximum intra cluster distance (the higher intra cluster distance overwrites the old one provided it is higher).

The maximum intra cluster distance cannot return 0 as it is the denominator of the Dunn Index equation, and you cannot divide by 0. To avoid an error, I have chosen to set this value to 1.

Finding the minimum inter cluster distance

The minimum inter cluster distance for a value of k is obtained using the find_min_inter_cluster_distance function that can be seen below.

```
# Finds the minimum inter-clusters distance.
def find_min_inter_cluster_distance(self):
    minimum_inter_cluster_distance = np.inf

    for _, centroid_1 in self.centroids.items():
        for _, centroid_2 in self.centroids.items():

            if centroid_1 != centroid_2 or len(self.centroids) == 1:
                inter_cluster_distance = self.find_distance(centroid_1, centroid_2)

                minimum_inter_cluster_distance = min(minimum_inter_cluster_distance, inter_cluster_distance)

    return minimum_inter_cluster_distance
```

Initially I have set the minimum inter cluster distance to infinity – a value that will get overwritten by any comparison.

Then, to obtain the minimum inter cluster distance, I have found the distance between every combination of centroids provided that the centroid is not comparing to itself and used the min function to compare the minimum inter cluster distance to the current inter cluster distance and make the result of the minimum inter cluster distance the smaller distance.

Calculating the Dunn Index and Finding the Optimal Value

```
    dunn_index = minimum_inter_cluster_distance / maximum_intra_cluster_distance

    dunn_indexes.append(dunn_index)

optimal_number_of_clusters = np.argmax(dunn_indexes) + 1

return optimal_number_of_clusters
```

After obtaining these values, I have calculated the Dunn Index for that value of k and appended it to a list of the Dunn Indexes. After the program has iterated through range of values of k, I have used the np.argmax function to return the index of the highest Dunn Index and then added 3 to obtain the optimal number of clusters.

Tuning the Optimal Value

After returning this estimated optimal value, I have decided to add multiple extra erode iterations to fully ensure that the wires will be fully eroded. As eroding the image extra times makes little difference, this does not cause any issues and ensures that the resistor is located correctly.

Glare Removal

I have removed the glare using three steps. Each of these steps are explained below.

Blurring the Image

Firstly, before I can identify glare within the image, I have very strongly horizontally blurred the resistor.

By doing this, I have exaggerated the rows of the image at which the glare is worst.

This is done using the code seen below.

```
original_image = self.image.clone()

self.image = BGR(self.image.image).blur(round(self.image.width() * 10), 1)
```

Firstly, I have cloned the input image as I am going to modify the image itself. After cloning the image, I have blurred the image on the horizontal axis for an amount that is 10 times the image width and on the vertical axis for none.

The modification that this makes to the image can be seen below.

Figure 72 - Image before strong horizontal blur

Figure 73 - Image after strong horizontal blur

As you can see, the areas of the image which had glare within them are now obvious and can be identified using K-Means.

Glare Identification

Conversion to HSV

Before I have identified the glare, I have converted the image to contain HSV values rather than BGR ones. This is because when identifying the glare, I will be interested in looking at how the V (intensity) of the varies across the image.

K-Means to Find Colour Clusters

As seen in the image above, the strongly horizontally blurred image has less colour components than the original image. Therefore, I have used K-Means to identify the 5 clusters in which the image colours lie.

I have visualized these 5 colours and their proportions for the horizontally blurred image in the image below.

Figure 74 - 5 colour clusters for horizontally blurred image

As you can see, the first, second clusters are much lighter than the other ones and if you refer to the original image above, these clusters refer to areas of glare.

I have identified these lighter clusters by taking a mean of the V values of the returned cluster centroids and declared the cluster centroids with V values higher than this mean to be glare centroids.

The code implementation carrying out this purpose can be seen below.

```
# Find the colour clusters.
def find_colour_clusters(self):
    try:
        image_data = self.image.image.reshape(self.image.height() * self.image.width(), 3)

        # Find and display most dominant colors
        clusters = KMeans(n_clusters=5).fit(image_data)

        return clusters

    except Exception as error:
        raise Exception(f'Error finding colour clusters, {error}')
```

```
# Identifies the glare clusters based on V values.
def identify_glare_clusters(self, clusters):
    try:
        hsv_colours = []

        for colour in clusters.cluster_centers_:
            hsv_colours.append(colour)

        v_values = [hsv_colour[2] for hsv_colour in hsv_colours]

        if v_values:
            mean_v_value = np.mean(v_values)
        else:
            mean_v_value = [0]

        glare_clusters = []
```

```
        for v_value in v_values:
            if v_value > mean_v_value:
                glare_index = v_values.index(v_value)
                glare_clusters.append(glare_index)

        return glare_clusters

    except Exception as error:
        raise Exception(f'Error identifying glare clusters, {error}')
```

First, within the find_colour_clusters function, I reshape the data to be in a supported format for the scikit K-Means function. The reason for using the scikit K-Means function over my own is due to the amount of data within an image matrix. Even though my K-Means implementation would support the image matrix values (as I calculate the 3d Euclidean distance which correspond to the H, S and V values), my implementation is far too inefficient to be considered acceptable for use in the final program. If given more time, I would optimize my K-Means algorithm and implement it over the scikit one.

After resizing the image matrix, I call the fit function with K defined as 5. This returns an object that contains the centroids, labels, and histogram.

Within the identify_glare_clusters function, I firstly have extracted the V values from the HSV values of the centroids within the clusters object. After obtaining these values, I have calculated the mean V value for the cluster centroids, and I have then iterated through the initial cluster centroids declaring any centroid with a V value above this mean to be classified as a glare centroid. I have appended these glare centroids to a list and returned them.

Removing the Glare

After the glare centroids have been identified, I can iterate through the image and remove pixels (by assigning them to 0) with labels (assigned closest centroids) that are classified as glare centroids.

The remove_clusters function is the code implementation of this. It takes in the clusters object and the glare_clusters as parameters.

```
# Change the pixels to 0 if its closest centroid identified as a glare clusters.
def remove_clusters(self, clusters, glare_clusters):
    try:
        cluster_labels = clusters.labels_

        cluster_labels = cluster_labels.reshape(self.image.height(), self.image.width())

        for row in range(cluster_labels.shape[0]):
            for column in range(cluster_labels.shape[1]):
                if cluster_labels[row][column] in glare_clusters:
                    self.image.image[row][column] = 0

        return self

    except Exception as error:
        raise Exception(f'Error removing clusters from image, {error}')
```

After acquiring the labels from within the clusters object, I resize the labels to match the dimensions of the image. I then iterate through the labels and check if the label for a pixel matches a glare cluster label, if it does, I set the value of that pixel to 0.

Doing this on the image above produces the result seen below.

Figure 75 - Glare region identified with 0 values on blurred image

After the location of the glare within the resistor image has been identified, I obtain a mask for the remaining image by masking every single pixel with a value higher than 0 (since the black region is composed 0 values). Doing this for the image above gives the result that can be seen below.

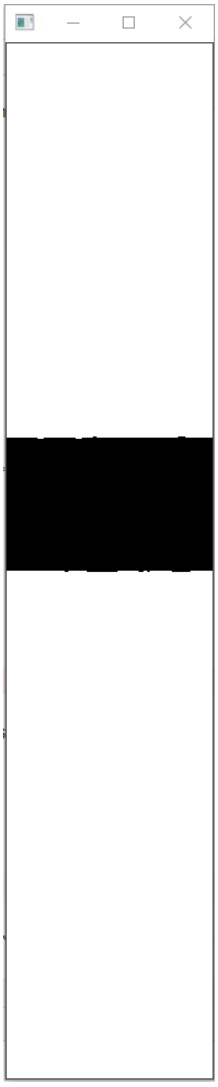


Figure 76 - Glare mask image

After acquiring this mask, I can see I am left with 2 white areas. This is done using the mask function that can be seen below.

```
# Masks the image for non black values.
def mask(self):
    try:
        hsv_image = HSV(self.image.image, 'BGR')

        colour_mask = hsv_image.mask(HSVRange([0, 255], [0, 255], [1, 255]))

        greyscale_mask_image = Greyscale(colour_mask, 'HSV')

        return greyscale_mask_image

    except Exception as error:
        raise Exception(f'Error masking image, {error}')
```

Firstly, I convert the image from a BGR image to an HSV image, then I proceed to mask pixels that fall into the HSV range [0, 255], [0, 255], [1, 255] (non 0 pixels). As my mask function returns an matrix rather than an image, I have initialized an instance of the Greyscale class with that matrix.

I have then chosen to identify the largest white area as combining both may cause problems with the continuation of the bands. The code for carrying this out can be seen below.

```
greyscale_mask_image = image.mask(glare_mask.image)

contours, _ = Greyscale(greyscale_mask_image.image, 'BGR').find_contours()

largest_contour = Contours(contours).find_biggest()
```

The variable greyscale_mask_image is the result of the mask function seen above and after finding its contours, I identify the largest contour.

After identifying this largest contour, I find the bounding rectangle of it and cut out the region defined by this bounding rectangle from the original image. This is done using the code below.

```
bounding_rectangle = BoundingRectangle(largest_contour)

no_glare_image = image.region(bounding_rectangle.x, bounding_rectangle.y,
bounding_rectangle.width, bounding_rectangle.height)

return no_glare_image
```

I have used my BoundingRectangle class to obtain the bounding rectangle of the contours and have used my region function within the image class to return the no glare image. This returned no glare image can be seen in the image below.

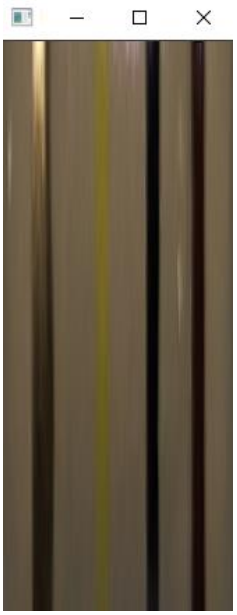


Figure 77 - No glare image

Identifying and Locating the Bands

Below I will explain my method for identifying a band's colour and position.

HSV Masking

Within my program, I have used HSV masking to identify and locate the bands. By masking an HSV range from the image, I can obtain a mask for a colour.

Increasing the Amount of Data to Work With

I have done 2 things to increase the amount of data I can extract from each image. Both can be seen in the code below.

```
self.image = self.image.resize(self.image.width(), self.image.height() * 20)

self.image = GlareRemover(self.image).main()

self.image = self.image.resize(self.image.width(), self.image.height() * 5)

blurred_image = BGR(self.image.image).blur(1, round(self.image.height() * 0.5))

image_slices = blurred_image.create_slices(round(blurred_image.height() * 0.05))
```

Before the masking process, I have stretched the image vertically to make the image 20 times its original height.

After stretching the image, I call the glare remover class's main function (which is described above) to remove the obvious glare from the image and obtain an image with no glare.

On obtaining this no glare image, I once again stretch the image vertically to make it 5 times its original height as the glare remover will have greatly decreased the height of the image.

I found that applying a vertical blur on the image greatly improved the results achieved by the masking as the effects of imperfections in the images are greatly reduced. This is shown within the images below.

Figure 78 - Top of image

Figure 79 - Top of image with vertical blur

As you can see, the noise in the resistor body background colour is reduced greatly and the consistency of the bands is greatly increased, with their colours being more solid and uniform.

I have then decided to split up the blurred stretched image into what I refer to as “image slices”. For the input image, I split the image into 20 equal sized slices (regions), as masking these image slices rather than the full image gives me more data to work with, greatly improving the accuracy of my program. I have done this using my create_slices function, with the image slice height argument being set as the original image height multiplied by 0.05 (5%). The code that creates these image slices can be seen below. This function is a part of the Image class.

```
# Returns the slices of an image.
def create_slices(self, slice_height):
    try:
        height = self.height()

        slice_amount = height // slice_height

        image_slices = []

        for slice_number in range(slice_amount):
            x = 0
            y = slice_number * slice_height

            image_slice = self.clone().region(x, y, self.width(), slice_height)

            image_slices.append(image_slice)

        return image_slices

    except:
        raise Exception("Error slicing image")
```

I have calculated the number of slices by dividing the original image height by the slice height parameter that was passed in and have iterated through the image for this defined number of times, obtaining a region of the image until the number of image slices is equal to the slice amount.

Finding Slice Bands

After obtaining these image slices, I have masked all the possible resistor colours using the HSV ranges I have defined for each colour. For each image slice, I find the slice bands. For each image slice, my code to find the slice bands can be seen below.

```
# Finds the bands.
def find_bands(self, image_slice):
    try:
        for colour in Colour:

            greyscale_mask_image = self.band_mask(colour, image_slice)

            non_zero_pixels = greyscale_mask_image.count_non_zero_pixels()

            band_contours = None

            if non_zero_pixels != 0:
                band_contours, _ = greyscale_mask_image.find_contours()

            if band_contours is not None:

                for contour in band_contours:
                    bounding_rectangle = BoundingRectangle(contour)

                    edge_band = False

                    if colour == Colour.WHITE or colour == Colour.GREY:
                        edge_band = self.check_if_edge_band(bounding_rectangle)
```

```
                    background_masked = self.check_if_background_masked(bounding_rectangle, image_slice)

                    if not edge_band and not background_masked:
                        slice_band = SliceBand(colour, bounding_rectangle)

                        self.slice_bands.add_band(slice_band)

        return self

    except Exception as error:
        raise Exception(f'Error trying to find bands for image slice, {error}')
```

I iterate through the colours within my enumeration Colour class that contains all the possible resistor colours.

For each colour, I have masked the HSV ranges. Before being able to do this I lookup the HSV ranges for a colour, these are stored within the HSVRanges class in the form of a dictionary and for_colour function does the lookup for the ranges of a colour. This can be seen in the code blow.

```
hsv_ranges = HSVRanges().for_colour(colour)
```

After acquiring these ranges, I convert the image to an HSV format and mask the HSV ranges. To mask the ranges, I use the cv2 inRange function to return the mask for the specified range. This can be seen in the code below in the mask function.

```
hsv_image = HSV(image_slice.image, 'BGR')

colour_mask = hsv_image.mask(hsv_ranges)

greyscale_mask_image = Greyscale(colour_mask, 'HSV')

return greyscale_mask_image
```

```
# Masking HSV ranges from an image.
def mask(self, hsv_ranges):
    try:
        h, s, v = self.split()

        if hsv_ranges.h_range[0] > hsv_ranges.h_range[1]:
            h_range_1 = cv2.inRange(h, hsv_ranges.h_range[0], 180)
            h_range_2 = cv2.inRange(h, 0, hsv_ranges.h_range[1])

            h_range = cv2.bitwise_or(h_range_1, h_range_2)

        else:
            h_range = cv2.inRange(h, hsv_ranges.h_range[0], hsv_ranges.h_range[1])

        s_range = cv2.inRange(s, hsv_ranges.s_range[0], hsv_ranges.s_range[1])
        v_range = cv2.inRange(v, hsv_ranges.v_range[0], hsv_ranges.v_range[1])

        # Narrowing down the image to only show the HSV values of the desired range
        hs_mask = np.bitwise_and(h_range, s_range)
        hsv_mask = np.bitwise_and(hs_mask, v_range)

        return hsv_mask

    except:
        print(f'Error masking HSV ranges {hsv_ranges}')
```

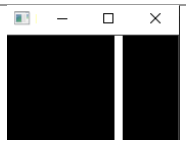
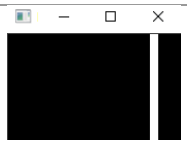
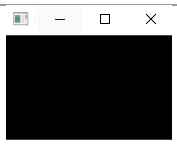
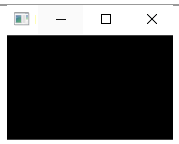
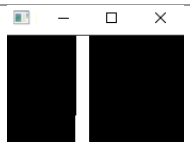
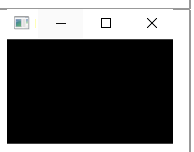
In addition to obtaining the range, I have also used bitwise “and” and “or” operations to combine the masks in the ways I desire.

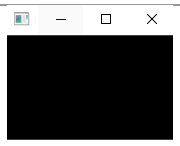
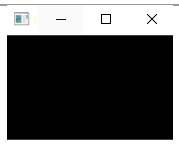
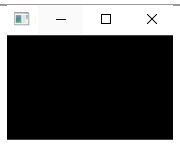
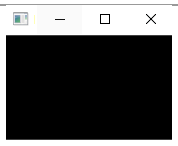
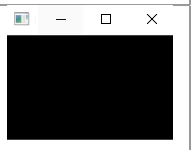
To obtain the H mask, as the HSV H ranges can wrap around 180 (since cv2 uses 180 degrees as the highest angle), I firstly check if the starting point of the h range is higher than the ending point. If the starting point of the H range is higher than the ending point, I mask 2 ranges and use a bitwise or

operation to then combine these masks to end up with 1 mask. Otherwise, I just mask the H range as specified.

After obtaining this H range, I mask the S and V ranges. Once these are masked, I use the bitwise and operation to combine these masks together. This means only areas that the 3 masks overlap in will be a part of the mask.

Below, I have stated the colour, the HSV ranges that are masked for that colour and the resultant image mask for a singular image slice.

Colour	Black	Brown	Red	Orange	Yellow	Green
H, S and V Range	[0, 255], [0, 255], [0, 30]	[170, 10], [100, 190], [20, 75]	[175, 5], [100, 255], [50, 150]	[5, 15], [100, 255], [70, 130]	[20, 30], [100, 255], [80, 130]	[40, 75], [100, 255], [40, 90]
Image						

Colour	Blue	Violet	Grey	White	Gold	Silver
H, S and V Range	[100, 115], [100, 255], [40, 80]	[135, 170], [70, 255], [35, 85]	[0, 255], [0, 20], [50, 75]	[0, 20], [10, 40], [110, 255]	[10, 25], [70, 150], [30, 70]	[0, 1], [0, 1], [80, 130]
Image						

As you can see from the data above, this specific image slice has matched the colours on the resistor well. The main problem within this specific image slice is that gold has identified extra bands outside of the real gold band. This is caused by its overlapping HSV ranges.

As seen in the code above within the find_bands function, for all the colours. After obtaining the mask image for a colour, if there are more than 0 nonzero pixels, I find the contours and then I initialize a bounding rectangle. I then check the bounding rectangle meets certain criteria before I append it to the list of slice bands. These criteria can be seen below.

- For all colours, The resistor body background must not have been masked. I check for this condition by seeing if the bounding rectangle width is greater than the width of half the slice image.

- If the band is grey or white, the bounding rectangle must not be on the absolute edges of the image as if it is, it likely means that the background paper has been masked (due to the resistor locator not eroding the image enough).

If both conditions are met, I find the contours and then the bounding rectangle for the image mask and initialize an instance of the SliceBand class. I then add these instances of the SliceBand class to the list attribute inside of the SliceBands class (the slice bands).

Filtering the Identified Bands

After updating the SliceBands class's list attribute with all the slice bands, I can start to filter out some of the slice bands.

Removing Short Slice Bands

The first filter that I apply to the slice bands is removing the short bands. This is because short bands are likely to be noise because of overlapping mask ranges than actual bands. If the bounding rectangle for the band is less than a quarter of the mean slice bands height, I remove the band from the list of slice bands. This is done using the code below, it is found within the SliceBands class.

```
height = self.get_mean_height()

minimum_height = max(1, height / 4)

short_bands = []

for slice_band in self.slice_bands:
    if slice_band.bounding_rectangle.height <= minimum_height:
        short_bands.append(slice_band)

for slice_band in short_bands:
    self.remove_band(slice_band)
```

Firstly, I have calculated the mean height of the slice bands. After obtaining the mean height, I have found the higher value between 1 and the mean height divided by 4 to use as the minimum height threshold.

After finding this minimum height threshold, I have iterated through the slice bands and appended any slice band that has a bounding rectangle height lower than this minimum height threshold to a list called short bands. I have then proceeded to remove every short band from the list of slice bands.

Removing Narrow Slice Bands

The next filter that I apply to the slice bands is to remove the slice bands that are particularly narrow. I have removed slice bands that are less than or equal to 2 as they are too thin to be a real band. This is done using the code below.

```
minimum width = 2

narrow_bands = []

for slice_band in self.slice_bands:
    if slice_band.bounding_rectangle.width <= minimum_width:
        narrow_bands.append(slice_band)

for slice_band in narrow_bands:
    self.remove_band(slice_band)
```

Using K-Means to Identify the Locations of the Bands

After removing obvious anomalous slice bands by removing particularly thin or short slice bands, I am left with a list of slice bands much more likely to be genuine. I have used K-Means to identify where the clusters of these slice bands are by running the K-Means fit function on the x centres of the slice bands (the x centre of a slice band is the x coordinate of the bounding rectangle + half the width of the bounding rectangle). My code implementation of this can be seen below.

```
x_centres = self.slice_bands.get_x_centres()
clusters = self.find_x_cluster(x_centres, number_of_bands)
```

Firstly, I call the get_x_centres function to obtain a list of the x centres for all the slice bands. After obtaining this list, I call the find_x_cluster function with the arguments of the x_centres list and the number of bands (if defined on the webpage). The code within the find_x_cluster function can be seen below.

```
# Using K-means to find where there are clusters of X (locating the band).
def find_x_cluster(self, x_centres, number_of_bands):
    try:
        x_y_z_list = np.array([[x, 0, 0] for x in x_centres])

        k_means = KMeans()

        if number_of_bands is None:
            number_of_bands = k_means.find_optimal_number_of_clusters(x_y_z_list)

        k_means.number_of_centroids = number_of_bands
        seeds = k_means.initialize_centroids(x_y_z_list)
        clusters = k_means.fit(x_y_z_list, seeds)

        return clusters

    except Exception as error:
        raise Exception(f'Error while trying to find x cluster, {error}')
```

Firstly, I format the input x_centres list to be a numpy array that contains 3d coordinates as this is what my K-Means algorithm works with. Within a list, I create lists that are 3 in length with 0 values for the y and z coordinates for every x centre.

If the number_of_bands parameter is a none type, I use my K-Means find_optimal_number_of_clusters function to try to work out the number of bands that the input resistor contains (this algorithm has been explained earlier in the document under the K-Means implementation section), else I just set the number of centroids to the value specified within the number_of_bands parameter.

After defining the number of centroids to be used for the fit function, I run the fit function with seeds (initial centroid values) that have been determined using the K-Means++ algorithm. This returns the K-Means object, which encapsulates the centroids dictionary as well as a list of the labels for each data item.

Sorting Centroids and Removing False Centroids

The centroids returned by the fit function may be unordered and therefore I must order them before using them so that they correspond to the order of bands on the resistor. I have sorted these centroids by appending the x values of the centroids to a list and have then used my merge sort implementation to sort them. This is done using the code below.

```
centroids = [centroid[0] for centroid in clusters.centroids.values()]

sorted_centroids = MergeSort().sort(centroids)
```

After sorting the centroids, I can attempt to identify centroids that are likely to be “false centroids”. A “false centroid” is a centroid that is too close to another centroid that was created due to the value for the number of centroids in the fit function being too high. I have identified these false centroids using the code below.

```
mean_difference = self.find_mean_difference(sorted_centroids)

previous_centroid = None
false_centroids = []

for centroid in sorted_centroids[:]:
    if previous_centroid:
        difference = centroid - previous_centroid

        if difference < (mean_difference * 0.2):
            false_centroids.append(centroid)

    previous_centroid = centroid

for centroid in false_centroids:
    sorted_centroids.remove(centroid)
```

Firstly, I find the mean difference between the sorted centroids and then I iterate through the centroids in the sorted centroids. If the difference between the previous centroid and the current centroid is less than the mean difference multiplied by 0.2, I append it to the false_centroids list.

I then iterate through the false centroids list to remove the values within this list from the sorted centroids.

Selecting the Possible Bands

After obtaining the sorted, corrected centroids, I can select slice bands if their x centres (slice band bounding rectangle x coordinate + half the bounding rectangle width) lie within a certain deviation of a centroid. I have done this using the identify_possible_bands function that can be seen below and is called using the sorted_centroids and deviation where sorted_centroids is a list of the sorted centroids and the deviation is the leniency of the range at which a slice band’s x centre can lie in to be classified as a possible band.

```
# Identifying the slice bands that match up with the x centroids.
def identify_possible_bands(self, sorted_centroids, deviation):
    try:
        possible_bands = {}

        for possible_band in range(len(sorted_centroids)):
            possible_bands[possible_band] = []

        mean_centroid_difference = self.find_mean_difference(sorted_centroids)

        for band_number, centroid in enumerate(sorted_centroids):
            for slice_band in self.slice_bands.list():

                centroid_band_difference = abs(slice_band.x_centre() - centroid)

                if centroid_band_difference < mean_centroid_difference * deviation:
                    possible_bands[band_number].append(slice_band)

        return possible_bands
```

```
    except Exception as error:
        print(f'identify_possible_bands(), {error}')
```

Firstly, I create a dictionary with empty lists for the number of sorted centroids in the sorted_centroids list.

I have then calculated the mean difference of the sorted_centroids parameter. When this mean difference has been calculated, for every centroid, I iterate through all the slice bands. If the absolute value of the difference between the slice band's x centre and the centroid is less than the mean centroid difference multiplied by the mean difference, the band is within the range to be classified to be classified as a slice band and it is appended to the corresponding band list in the possible band dictionary.

However, the deviation may be too small to pick up many close bands. That is why my get_possible_bands function that can be seen below adjusts the deviation argument depending on the length of the shortest list within the dictionary.

```
# Keep trying to find binds with a certain deviation until a sufficient amount has been found.
def get_possible_bands(self, sorted_centroids):
    try:
        possible_bands = self.identify_possible_bands(sorted_centroids, 0.2)

        if len(min(possible_bands.values(), key=len)) <= 5:

            deviation = 0.21

            while len(min(possible_bands.values(), key=len)) <= 3 and deviation <= 0.3:
                possible_bands = self.identify_possible_bands(sorted_centroids, deviation)

                deviation += 0.01

        return possible_bands

    except Exception as error:
        print(f'possible_bands(), {error}')
        raise Exception(f'Error trying to find possible bands, {error}')
```

After the possible bands have been returned, I check if the shortest list within the dictionary has less than or equal to 5 possible slice bands. If it does not, I adjust the deviation by 0.01 which increases it to 0.21 from 0.2. I have then used a while loop to keep calling the identify possible bands function with higher and higher values for the deviation (incrementing it by 0.01 on each iteration) until either the length of the shortest list in the possible bands dictionary is more than 3 or the deviation is more than 0.3.

At this point, I obtain a dictionary with keys being the band position and the values being the lists of possible slice bands.

I then add these possible bands into the SliceBands slice_band_groups attribute as these SliceBand objects belong in the SliceBands class. I return the instance of the class I create after adding the possible bands in.

```
slice_band_groups = SliceBands().load_slice_band_groups(possible_bands)

return slice_band_groups
```

Identifying the Bands

After obtaining the possible bands, I need to choose which colour best suits each band. I have chosen the bands based on several criteria.

Weighting the Bands

I have chosen to weight the identified colours within the possible slice bands. Due to overlapping colour ranges, I know that some colours must take priority over others. For example, black and white must be weighted lower than other colours as their HSV ranges overlap with many other colours. I have made a dictionary to hold the colours and its corresponding weight which can be seen below, these values were achieved after much testing on automated image tests and real data.

```
self.colour_weights = {
    Colour.UNKNOWN: 0,
    Colour.BLACK: 0.5,
    Colour.BROWN: 1,
    Colour.RED: 1,
    Colour.ORANGE: 0.75,
    Colour.YELLOW: 1,
    Colour.GREEN: 1,
    Colour.BLUE: 1,
    Colour.VIOLET: 1,
    Colour.GREY: 1,
    Colour.WHITE: 0.5,
    Colour.GOLD: 1,
    Colour.SILVER: 0.75,
}
```

Finding the Colour Counts

Before I can apply the weighting, I must count the number of times each colour appears in the possible bands for a band. I have done this using my find_band_colour_counts function that can be seen below.

```
# Finding the band colour counts for each band.
def find_band_colour_counts(self):
    try:
        slice_band_groups = self.possible_bands.groups()

        band_colour_counts = []

        for index, _ in enumerate(slice_band_groups):
            band_colour_counts.append({})

        for index, slice_band_group in enumerate(slice_band_groups):

            colour_counts = band_colour_counts[index]

            if slice_band_group:

                for slice_band in slice_band_group:
                    colour = slice_band.colour

                    count = colour_counts.get(colour, 0)

                    colour_counts[colour] = count + 1
            else:
                colour_counts[Colour.UNKNOWN] = 1

        return band_colour_counts

    except Exception as error:
        raise Exception(f'Error counting number of possible bands, {error}')
```

Firstly, I retrieve slice_band_groups from the possible_bands (which is a SliceBands object). After retrieving this 2d list, I create a list containing empty dictionaries for the length of the slice band groups.

Once this list of dictionaries has been created, I iterate through each slice band group within slice_band_groups. For each slice band group, I iterate through every slice band in the slice band group, and I set/increase the count of each colour in the colour counts dictionary.

After the loop finishes iterating, I will have a list of dictionaries that contains the colours and their corresponding counts.

Applying Colour Weights

The function that is responsible for applying the colour weights is the apply_colour_weights function and it can be seen below.

```
# Applying the colour weights to their corresponding band colour counts.
def apply_colour_weights(self, band_colour_counts):
    try:
        for index, colour_counts in enumerate(band_colour_counts):

            for colour, count in colour_counts.items():

                weight = self.colour_weights[colour]

                if colour is Colour.GOLD or colour is Colour.SILVER:
                    if index == 0 or index == len(band_colour_counts) - 1:
                        weight *= 2

                    else:
                        weight /= 2

                colour_counts[colour] = weight * count

        return band_colour_counts

    except Exception as error:
        raise Exception(f'Error applying colour weights, {error}')
```

After finding the colour counts, I can apply the colour weights. I iterate through the list of dictionaries (band_colour_counts), and for each dictionary (colour_counts), I iterate through the colours and counts of the dictionary. For each colour in the dictionary, I look up the weight for the colour from the colour_weights dictionary that can be seen earlier in the document.

As gold bands and silver bands can only be found on the multiplier (which is very rare) or the tolerance band (which is much more common), I have decided to multiply the weight of gold and silver bands if they are at either end of the resistor (increased weight is applied at both ends as the image could be in the wrong orientation) and divide their weights by 2 if they are not.

After I have obtained the weight for the colour, I multiply the count for the colour in the dictionary by the weighting.

Majority Voting the Bands

After I have obtained the weighted counts for the band colours for a band, for each band, I can say that the colour with the highest count is most likely to be the resistor colour. This is done using the find_most_frequent_bands function that can be seen below.

```
# Majority voting the resistor bands.
def find_most_frequent_bands(self, band_colour_counts):
    try:
        resistor_bands = []

        for colour_counts in band_colour_counts:
            colour = max(colour_counts, key=colour_counts.get)

            resistor_bands.append(ResistorBand(colour))

        return resistor_bands

    except Exception as error:
        raise Exception(f'Error finding most frequent bands, {error}')
```

I iterate through the band_colour_counts list of dictionaries and for each band, for the most frequent colour, I create an instance of the ResistorBand class that I then append to the list of resistor bands.

Processing Resistor Bands

Checking if Resistor Bands are Valid

As resistors have standard values for the digit bands (digit bands are band 1, 2 or band 1, 2, 3), by looking at the first couple bands of the resistor, I can determine whether the detected resistor values are valid.

I have found these standard resistor values on the website Electronic Notes (Standard Resistor Values: E3, E6, E12, E24, E48 & E96, 2021) and have translated all these numerical to their corresponding colours using a conversion program I made which I have stored in external files (standardResistorValues2sf.json and standardResistorValues3sf.json).

Each of these json files contains a list of the possible colours that the first 2 or 3 bands can be. The code that carries out this check can be seen below.

```
# Checks if the resistor digit band bands are in the standard values database.
def check_valid(self, digit_band_colours):
    try:

        if len(digit_band_colours) >= 2:
            number_of_bands = self.get_number_of_bands()

            if number_of_bands < 5:
                number_of_digit_bands = 2

            else:
                number_of_digit_bands = 3

            data_file = f'standardResistorValues{number_of_digit_bands}sf.json'

            with open(f'{os.getcwd()}\\data\\standardResistorValues\\{data_file}') as valid_band_colours_list:

                valid_band_colours_list = json.load(valid_band_colours_list)

                if digit_band_colours in valid_band_colours_list:
                    return True

                else:
                    return False

        else:
            return False
```

```
    except Exception as error:
        print(f'check_valid(), {error}')
```

As the digit band values must be a minimum of 2 in length, firstly I check that the number of bands in the digit_band_colours parameter is more than or equal to 2.

If there are 2 or more digit band colours in digit_band_colours, I get the total of number of bands from the self bands value. If the total number of bands is 3 or 4, I use the 2sf standard resistor values file else I use the 3sf standard resistor values file.

After identifying which file needs to be used, I can then load the json file using the json import within python. This returns the data within the json as a 2d array, I have called this variable the valid_band_colours_list. I can then check if the digit_band_colours parameter values are a member of the valid_band_colours list. If they are, I return true (the resistor is valid), else I return false (the resistor is invalid).

Correcting the Orientation of the Resistor

As the input resistor image, may be in the wrong orientation, I check whether the detected values are in the correct order by using my check_valid function that can be seen above. In some cases, the orientation can easily be corrected by being flipped as the as resistors cannot have certain band colours such as gold (and silver) at the start of the resistor (gold is a very common tolerance band). In other cases, where both digit band colour combinations are possible (do not contain silver or gold), one orientation will be valid while the other will not. I find the optimal orientation of the resistor bands using the code that can be seen below.

```
self.valid = self.check_valid(self.get_digit_band_colours(self.colours()))

if self.valid is True:
    return self

else:
    self.bands = self.bands[::-1]
    self.valid = self.check_valid(self.get_digit_band_colours(self.colours()))

    return self
```

For each call of the check_valid function. The argument used for this function are the first 3 bands (acquired from the get_digit_band_colours function) from the formatted resistor band colours (from the colours function).

Firstly, using the original detected bands, I call the check_valid function. If check_valid returns true, I return the resistor object, else I flip the resistor bands and check if the resistor is valid again. If the detected values are accurate, one of these orientations will have produced an output that is valid, as one of the orientations of the resistor must have been produced digit band colours that are a part of the standard resistor values.

Webserver

App Routes

My webserver has 3 app routes:

- /ui – a get method that is responsible for returning the webpage and its resources (html page, css, and javscript files).
- /api/detect – a post api call that is responsible for running the resistor detector after the image has been input on the webpage and the scan button has been pressed. The code for this app route can be seen below.

```
# Calls the detector program.
@app.route('/api/detect', methods=['POST'])
def detect():

    file = request.files['file']

    if request.values:
        resistor_type = int(request.values['type'])

    else:
        resistor_type = None

    location = f'{os.getcwd()}\\data\\images\\{file.filename}'

    try:

        with open(location, 'wb') as target:
            file.save(target)

        resistor, resistor_image = Detector().detect(location, resistor_type)

        resistor_image_byte_stream = resistor_image.byte_stream()

        resistor_image_byte_stream = resistor_image_byte_stream.decode('utf-8')

        resistor_colours = resistor.colours()

        if resistor_type is None:
            resistor_type = resistor.get_number_of_bands()

        return jsonify(colours=resistor_colours, type=resistor_type, image=resistor_image_byte_stream,
                       valid=resistor.valid, error='')

    except Exception as error:
        print(error)

        colours = [None, None, None, None, None, None]

        if resistor_type is None:
            resistor_type = 6

        return jsonify(colours=colours, type=resistor_type, image=None, valid=False, error=str(error))
```

When the detect app route is called, I receive a request. From this request, I extract the image file and the specified resistor type (if specified with lock button).
I then save the file to .../data/images directory and call the detect function.
This detect function returns the resistor located within the image as well as the as a resistor object that contains the resistor values and validity.

Before responding to the initial request, I convert the resistor image to a byte stream and decode it and I format the resistor bands to be in the supported format of the webpage. After converting these values, I respond to the request with the resistor colours, the resistor type, the resistor image, the resistor validity, and an empty error (as no error has occurred).
If there is an error at any point, I will return a list of none bands, the type as 6, no image, the resistors invalidity as well as a the error.

- /api/validate – a post api call which is responsible for checking if the input digit bands on the website are valid. This api call occurs every time the resistor band colour buttons on the webpage are updated. The code for this validate app route can be seen below.

```
@app.route('/api/validate', methods=['POST'])
def validate():
    resistor_bands_colours = request.values["resistor_bands"]

    resistor_bands_colours = json.loads(resistor_bands_colours)

    resistor = Resistor(resistor_bands_colours)

    digit_band_colours = resistor.get_digit_band_colours(resistor_bands_colours)

    valid = resistor.check_valid(digit_band_colours)

    return jsonify(valid=valid)
```

When the validate app route is called, I receive a request. From this request I extract the resistor_band_colours from the request's values and format it (using the json loads function). After I have obtained the resistor bands colours, I initialize an instance of the Resistor class with these resistor bands, and have then called the get_digit_band_colours and check_valid functions that have been described earlier in the implementation section.
I respond to the request with the result of the check_valid function call.

Testing

Ongoing Testing

Module Tests

Within my testing file, I have developed module tests that allowed me to test specific parts of my program. I have done this by developing automatic tests that input the data needed to run certain parts of the program. For example, to ensure the band location/identification was working properly, I have input pre-cropped images of the resistor body (so that I do not need to use the resistor locator to run the file) directly into the main function of the band finding class by creating an entry point within that file and have verified manually if there is or isn't an improvement.

This allowed me to develop modules of my programs separately from each other, which meant that I could develop the band locator (which is something I had already established how to do to some degree in the prototype), before I had developed the resistor locator (which is something I had to spend much time after thinking of an algorithm for).

Full Automated testing

During the development of my program, I have used the full automated testing that I developed allowed me to test my full program once I had developed all the key components. This meant it was very quick to verify whether a change to my algorithms would increase or decrease the accuracy of my program as I could obtain an accuracy for each time the program was ran by comparing detected outcomes to expected outcomes.

Final Testing

To verify that my program met all the objectives I performed two types of testing:

- Automated Testing
- Manual Testing

The automated testing was aimed at verifying the resistor location and band detection worked for as many types of resistor image as possible

The manual testing was aimed at verifying the overall program function when used through the GUI worked properly. This reflects the real-world usage of the program.

All the tests for my program are based on the initial objectives. I have linked each objective number to a test number, as well as what the point of the test is.

Full Automated Testing

I have incorporated automated testing for testing my program. Due to the nature of how many tests need to be conducted to ensure the reliability of my program, I have decided that automated testing is the best way to verify that the resistor band colour detection is up moderately accurate.

To carry out this automated testing, I have created a directory that contains resistor images containing all the possible colours for resistors. I have gone through and renamed all these images with the outcome I am expecting.

After doing this I can then print out and compare the expected detected resistor band colours (the colours in the filename) against the resistor band colours that my program has detected. I can also calculate a final accuracy score at the end that shows the percentage of correct colours that have been detected.

If this final accuracy score is above 80%, I will consider my program to be reliable to a reasonable degree as, in practice, inaccurate readings could also be easily amended after resistor values are output as per the design of my program.

The results of the automated testing are presented in my test video where I show the results of running the automated test program and the results it obtained.

Manual Testing

In the design section I defined a set of tests that I would run to meet the objectives.

Most of the tests are manually run using the GUI application.

For the manual testing I used the following setup:

- The resistor locator application webserver running on my home computer
- The resistor locator client running on a device (iPhone / Android phone) for performing the manual testing with.

The manual testing consisted of running through the steps for defined in the tests design

The results of the manual testing are presented in the test video which shows a screen capture recording of me running through the test steps and the results are summarized in the test table.

Test Results

The results of both the manual and automated tests a are presented in my test video.

A test table is provided that explains where each test appears in the test video, and which objective it is linked to.

Test Video

The test video contains a recording of all the tests (both automated and manual) that were performed.

The link for my test video can be seen below. All timestamps in the test table below are references to this video.

Video link:

https://www.youtube.com/watch?v=5jGOKuXCPMk

Test Table

The test table contains the following information

- Test Number.
- Number of objective that the test is linked to.
- Description of the test steps.
- Type (Image from camera, image from file, erroneous image).
- The expected outcome of the test.
- The timestamp of the test.
- The actual outcome of the test as a Pass or Fail.

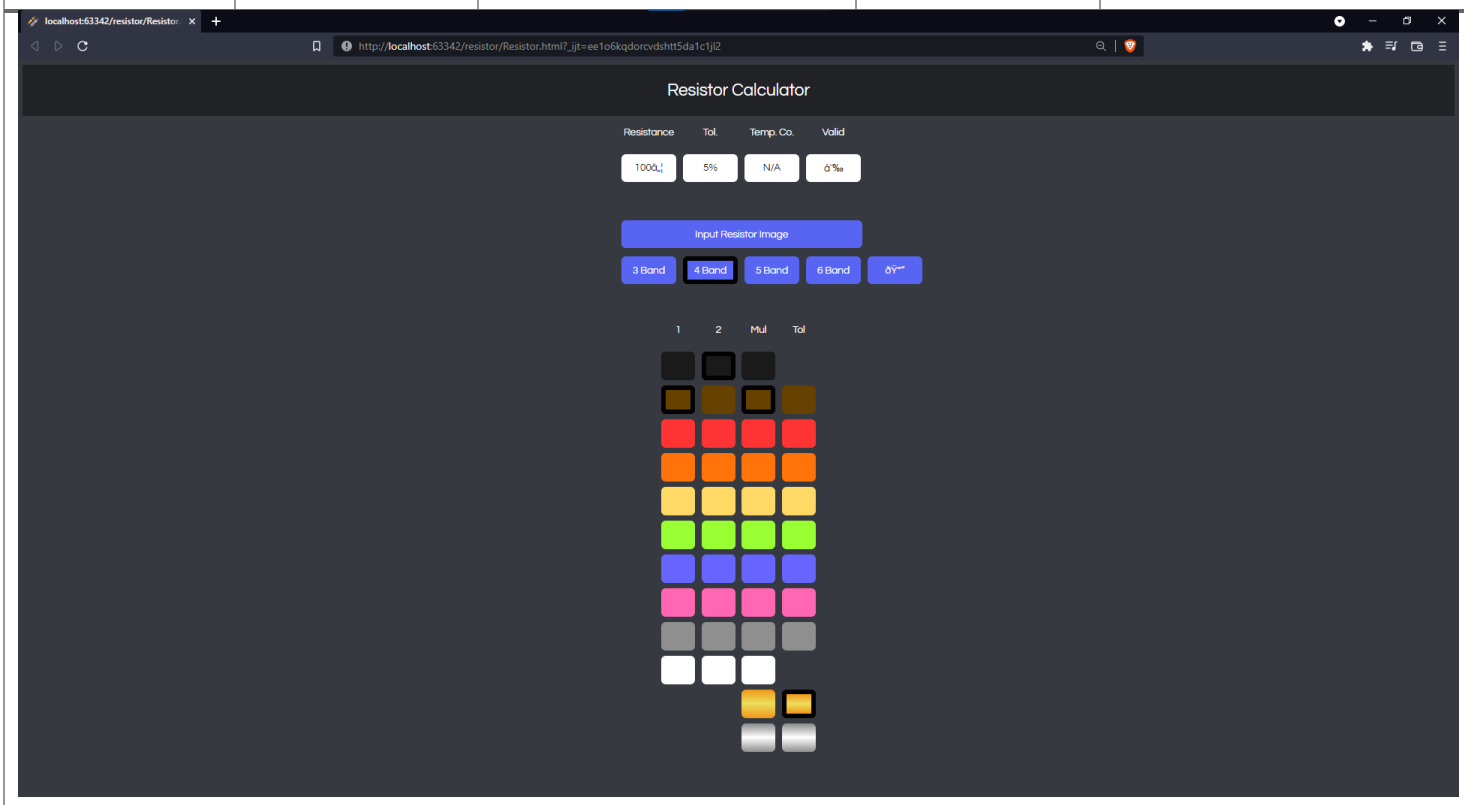
The test table for my application can be seen below.

Test Number	Objective	Description	Data / Type	Expected Outcome
1	1	Input images of resistors by selecting pre-existing files and scan.	Normal, from pre-existing files.	Webpage updates with correct resistor value displayed.
Actual outcome: 2:07 – 3:43, Pass				
2	1	Input images of resistors via camera input and scan.	Normal, picture taken on the spot.	Webpage updates with correct resistor value displayed.
Actual outcome: 0:09 – 2:07, Pass				

Test Number	Objective	Description	Data / Type	Expected Outcome
3	1	Input image of blank page and scan.	Erroneous, no resistor in image.	Webpage updates with error saying no resistor could be in the image.
Actual outcome: 5:50 – 6:24, Pass				
4	2	Input images of resistors by selecting pre-existing files and scan.	Normal, from pre-existing files.	Webpage updates with correct resistor band colour buttons selected.
Actual outcome: 2:07 – 3:43, Pass				
5	2	Input images of resistors via camera input and scan.	Normal, picture taken on the spot.	Webpage updates with correct resistor band colour buttons selected.
Actual outcome: 0:09 – 2:07, Pass				
6	2	Input image of blank page and scan.	Erroneous, no resistor in image.	Webpage updates with error saying no resistor could be located within the image.
Actual outcome: 5:50 – 6:24, Pass				
7	3	Input image of resistor via camera roll, scan and then change the selected resistor band colour buttons.	Normal, from pre-existing file.	After scanning, webpage updates with returned resistor values from initial image. After correcting bands, observe that the resistor value and validity status are updated.
Actual outcome: 6:24 – 7:14, Pass				
8	3	Input image of resistor via camera roll, scan and then change the selected resistor band colour buttons.	Erroneous, returned resistor value is not fully accurate.	After scanning, webpage updates with returned resistor values from initial image. After correcting bands, observe that the resistor

Test Number	Objective	Description	Data / Type	Expected Outcome
				value and validity status are updated.
Actual outcome: 6:24 – 7:14 (for camera input but is the same process), Pass				
9	4	Select various valid and invalid combinations of resistor values using the resistor band colour buttons.	Normal.	Resistor value and validity status are updated when the resistor band colour values change.
Actual outcome: 4:15 – 5:48				
10	5	Input images of resistors by selecting pre-existing files and scan.	Normal, form pre-existing files.	The resistor values and resistor band colour buttons are both output after scanning the image.
Actual outcome: 2:07 – 3:43, Pass				
11	6	Input images of resistors by selecting pre-existing files and scan.	Normal, form pre-existing files.	Output resistor value matches the resistor when it is in its correct orientation with its validity being verified.
Actual outcome: 2:07 – 3:43, Pass				
12	6	Input images of resistors via camera input and scan.	Normal, picture taken on the spot.	Output resistor value matches the resistor when it is in its correct orientation with its validity being verified.
Actual outcome: 0:09 – 2:07, Pass				
13	6	Input pre-existing image of problematic resistor and scan.	Erroneous, returned resistor value is not fully accurate.	Incorrect output resistor value has its validity being declared as wrong.
Actual outcome: 6:24 – 7:14, Pass				

Test Number	Objective	Description	Data / Type	Expected Outcome
14	6	Input image of blank page and scan.	Erroneous, no resistor in the image.	Output values are reset to their default, all resistor band colour buttons are deselected and the validity status is reset to invalid.
Actual outcome: 5:50 – 6:24, Pass				
15	7	Select various valid and invalid combinations of resistor values using the resistor band colour buttons.	Normal.	Output resistor values are displayed with their correct unit.
Actual outcome: 4:15 – 5:48, Pass				
16	8	Open the HTML page without running the webserver and select various resistor values using the resistor band colour buttons.	Normal.	Output resistor values are recalculated when the resistor band colour buttons are changed.
Actual outcome: Pass, as outputs show correct values and band buttons can be selected as usual (however some symbols show differently, and validity checker and resistor image input options are not available).				

Test Number	Objective	Description	Data / Type	Expected Outcome
				
17	9	Input pre-existing resistor images that have the resistors in different locations and scan.	Normal, from pre-existing files.	Output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
Actual outcome: 8:39 – 9:07, Pass (different input mode, but result would be the same)				
18	9	Input images of resistors via camera input with the resistors in different locations and scan.	Normal, picture taken on the spot.	Output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
Actual outcome: 8:39 – 9:07, Pass				

Test Number	Objective	Description	Data / Type	Expected Outcome
19	10	Input images of resistors via camera input with the resistor being different orientations and scan.	Normal, picture taken on the spot.	Output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
Actual outcome: 7:43 – 8:39, Pass				
20	11	Input pre-existing resistor images that have different amounts of bands and scan.	Normal, from pre-existing files.	Output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
Actual outcome: 7:30 – 7:36 (5 band) 0:09 – 2:07 (4 band) Pass				
21	11	Input images of resistors that have different amounts of bands via the camera input.	Normal, picture taken on the spot.	Output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
Actual outcome: 2:56 – 3:04 (5 band) 3:04 – 3:41 (4 band) Pass				
22	12	Access application on desktop computer, input pre-existing	Normal, from pre-existing file.	Webpage displays correctly, output resistor values are correct, verified as valid and the

Test Number	Objective	Description	Data / Type	Expected Outcome
		resistor image and scan.		resistor band colours selected correspond to the resistor in its correct orientation.
Actual outcome: 2:13 – 2:51, Pass				
23	12	Access application on phone, input pre-existing resistor image and scan.	Normal, from pre-existing file.	Webpage displays correctly, output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
Actual outcome: 2:07 – 3:43, Pass				
24	12	Access application on phone, use camera input to input resistor image and scan.	Normal, picture taken on the spot.	Webpage displays correctly, output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
Actual outcome: 0:09 – 2:07, Pass				
25	13	Access application on device running Windows, input pre-existing resistor image and scan.	Normal, from pre-existing file.	Webpage displays correctly, output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
Actual outcome: 2:13 – 2:51, Pass				

Test Number	Objective	Description	Data / Type	Expected Outcome
26	13	Access application on device running Android, input pre-existing resistor image and scan.	Normal, from pre-existing file.	Webpage displays correctly, output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
Actual outcome: 3:23 – 3:45, Pass				
27	13	Access application on device running Android, use camera input to input resistor image and scan.	Normal, picture taken on the spot.	Webpage displays correctly, output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
Actual outcome: 1:44 – 2:05, Pass				
28	13	Access application on device running iOS, input pre-existing resistor image and scan.	Normal, from pre-existing file.	Webpage displays correctly, output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.
Actual outcome: 2:51 – 3:23, Pass				
29	13	Access application on device running iOS, use camera input to input resistor image and scan.	Normal, picture taken on the spot.	Webpage displays correctly, output resistor values are correct, verified as valid and the resistor band colours selected correspond to the resistor in its correct orientation.

Test Number	Objective	Description	Data / Type	Expected Outcome
Actual outcome: 0:09 – 1:44, Pass				
30	14.1	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure black resistor bands are correctly detected over 80% of the time.
Actual outcome: 9:10 – 10:57, Pass				
31	14.2	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure brown resistor bands are correctly detected over 80% of the time.
Actual outcome: 9:10 – 10:57, Pass				
32	14.3	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure red resistor bands are correctly detected over 80% of the time.
Actual outcome: 9:10 – 10:57, Pass				
33	14.4	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure orange resistor bands are

Test Number	Objective	Description	Data / Type	Expected Outcome
				correctly detected over 80% of the time.
Actual outcome: 9:10 – 10:57, Pass				
34	14.5	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure yellow resistor bands are correctly detected over 80% of the time.
Actual outcome: 9:10 – 10:57, Pass				
35	14.6	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure green resistor bands are correctly detected over 80% of the time.
Actual outcome: 9:10 – 10:57, Pass				
36	14.7	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure blue resistor bands are correctly detected over 80% of the time.
Actual outcome: 9:10 – 10:57, Pass				
37	14.8	Run automated testing from within the program on a collection of pre-existing images that contains resistors that	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure violet

Test Number	Objective	Description	Data / Type	Expected Outcome
		have all the possible resistor band colours.		resistor bands are correctly detected over 80% of the time.
Actual outcome: 9:10 – 10:57, Pass				
38	14.9	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure grey resistor bands are correctly detected over 80% of the time.
Actual outcome: 9:10 – 10:57, Pass				
39	14.10	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure white resistor bands are correctly detected over 80% of the time.
Actual outcome: 9:10 – 10:57, Pass				
40	14.11	Run automated testing from within the program on a collection of pre-existing images that contains resistors that have all the possible resistor band colours.	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band values. Ensure gold resistor bands are correctly detected over 80% of the time.
Actual outcome: 9:10 – 10:57, Pass				
41	14.12	Run automated testing from within the program on a collection of pre-existing images that	Normal, from pre-existing files.	Application console prints out expected resistor values next to detected resistor band

Test Number	Objective	Description	Data / Type	Expected Outcome
		contains resistors that have all the possible resistor band colours.		values. Ensure silver resistor bands are correctly detected over 80% of the time.
Actual outcome: No data.				

Evaluations

Overall, the project ended up being much larger and took much longer than anticipated. However, despite this, I have finally managed to create a system that meets all the objectives I have set for it.

Meeting Objectives

The test plan I produced in the design section was specifically written to test each of the objectives I had defined as user stories. Each test that passes verifies that I had met the objective associated with the test.

Although the test results prove that all the objectives have been met, below you can see a summary of the objectives and how they were met.

Objective	Description	Test Result	Notes
1	As a user I want to be able to take a photo of a resistor and have the resistor value calculated for me.	Pass	Most of the manual tests demonstrate this capability.
2	As a user I want the bands colors that have been automatically detected from the resistor image to be displayed.	Pass	Demonstrated this in tests where I provided image from either phone camera or from file.
3	As a user I want to be able to override the band colors that were detected so that I can make sure the resistor calculation is accurate.	Pass	Successfully demonstrated the system can work in this “correction” mode and can be used to correct the bands that were detected.
4	As a user I want to be able to enter the resistor band colors manually (without needing an image) and have the value calculated for me.	Pass	Successfully demonstrated the system can work in this “manual” mode and can be used to enter the bands that with no image being required.
5	As a user I want the calculated resistor value to be displayed next to the resistor bands.	Pass	Demonstrated the GUI showing resistor band values when used in the different modes (normal / correction and manual).

Objective	Description	Test Result	Notes
6	As a user I want the system to tell me when the set of bands does not correspond to a standard resistor value.	Pass	Provided two types of feedback to the user. Show when a scanned image produced an invalid set of bands. Show when there is an error processing the resistor image with an error banner.
7	As a user I want the displayed resistor value details to include the important resistor details. 1. Resistance in Ohms 2. Tolerance in percent 3. Temperature constant in ppm/K	Pass	These values are populated correctly when the bands are detected or filled manually and show the units for each field are shown when a value is present.
8	As a user I would like to be able to use the system in an offline mode.	Pass	Successfully demonstrated running the program on the desktop as a local server, where manual input works correctly, and the values are updated.
9	As a user I would like the system to work when the resistor is located anywhere in the image.	Pass	Took pictures with the resistor in a variety of locations and orientations in the image and it is successfully detected.
10	As a user I would like the system to work when the resistor is smaller or larger in the image.	Pass	The resistors that I used within the test video varied greatly in size, with some being double the size of others.
11	As a user I want the system to be able to calculate the resistance for resistors with a different number of bands.	Pass	I successfully demonstrated value detection on 4 and 5 band resistors. The K-Means K identification algorithm I produced gave the ability to make this work.

Objective	Description	Test Result	Notes
12	As a user I would like to be able detect resistors using different devices.	Pass	I was able to test on Windows PC / iPhone and Android tablet.
13	As a user I would like to be able detect resistors on devices with different operating systems.	Pass	I was able successfully to test on Windows / iOS and Android phones.
14	As a user, I would like the application to work with all resistor colours. 13.Black 14.Brown 15.Red 16.Orange 17.Yellow 18.Green 19.Blue 20.Violet 21.Grey 22.White 23. Gold 24. Silver	Pass*	*Silver was untested I was unable to test silver band detection because none of the resistors I had access to had silver bands. Because I included the ability to detect silver in the band detector, I am hopeful it would work in the same way as the gold band detection as it shares many similarities with the gold band.

The Client's Thoughts

Client Feedback

After I finished developing my application, I showed it to my client in a meeting and asked for their thoughts and feedback. I have documented their feedback below.

- "The ability to correct the orientation of the resistor values is an excellent feature that I did not even consider as an issue."
- "The ability for a user override is a great and essential feature and is a great and well thought out way to address the problem of incorrect values."
- "The interface you have developed looks very aesthetically pleasing, clear and precise. Its features are obvious, and it looks easy to use."
- "The application is very slick, well put together, quick, and responsive."
- "I am very impressed with the accuracy of the program, and it has surpassed my expectations."

- “I am very pleased that the application works so well and will set it up to run in the department to ultimately sort out the resistors within the box of resistors. This application will greatly speed up time taken to find resistor values and will allow anyone to help me to sort the unsorted resistors.”
- “The lock button idea is a great idea as it is very easy for the user to identify the number of bands a resistor has.”

I also asked the client a couple questions.

Question	Is my application better than the existing resistor scanner applications I showed you during our first meeting?
Answer	Yes, its accuracy and ease of use is much better. I am very pleased with the system you have created.

Question	Did you see any issues or improvements for my application?
Answer	The functionality of the program is great, and I am very pleased that you have created a working solution. I do not see any issues; however, I think an improvement that could be made is implementing the second version of the wireframe UI with the live camera input feed.

Conclusion

I am very happy with the feedback that my client has provided and have included it in its entirety above. The client’s feedback suggests he is pleased with the system I have created and that it has surpassed his expectations in many areas – such as the accuracy and the ability to correct the resistor’s orientation.

This is because the objectives that I have set out for my system have been tailored around the client’s demands as well as key points for the functionality of the program.

However, due to the amount of time I had, I could not implement the second version of the wireframe UI that I designed that includes the live video feed to input the resistors. The client has mentioned this as a possible improvement, and in future I may work to implement this functionality within my program.

Overall, I am also very pleased that my application will see real-world use in aiding the sorting of resistors at Reading School.

Thoughts During Development

Implementing the GUI

Developing the UI was smooth sailing. By positioning the elements of the website within invisible tables, the UI I produced looks almost identical to that in the design.

Below are both the images of my GUI concept vs the GUI I have implemented within the project.

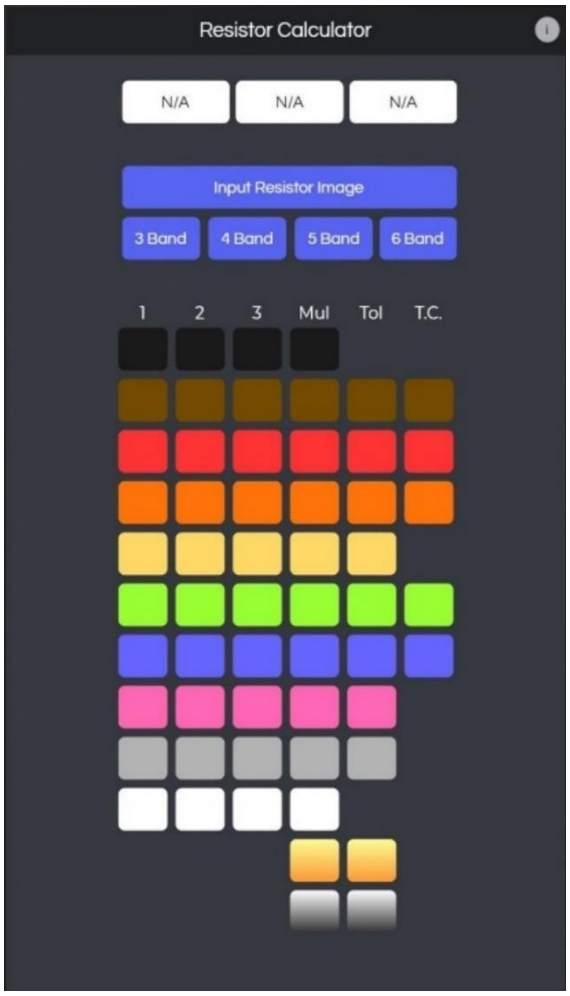


Figure 80 - Design concept GUI

Figure 81 - Real GUI

I have made a couple adjustments to the real GUI compared to the design concept I made. These differences include:

- A dedicated display box to show the resistors validity.
- A lock button to override the number of bands chosen by the program.
- No help button, as I did not have time to create a video on how to use the application. If I had more time, I would have implemented this help button.

Apart from these adjustments, I would say that I have successfully created a GUI true to the initial design. I thoroughly enjoyed turning my initial design concept into a real webpage using HTML, CSS, and JavaScript.

Resistor Body Location

Initial Approaches

Hough-lines and K-Means

My first approach to locating the resistor within an image was by applying canny edge detection on the image, finding an image's Hough-lines, and then using K-means on the resulting image.

In theory, identifying the location at which there are clusters of Hough-lines would be where the image was, however, in practice, this was not the case, and the resistor was not located to a high enough accuracy to be considered acceptable.

Haar Classifier

As mentioned in the prototype in the research section of this document, I did not have much success with using a Haar Classifier (machine learning) to locate the resistor within the image as I did not have enough data to train it. Due to my very limited success with this, I decided it was best to develop my own algorithm to locate the resistor body within the image.

Final Implementation

After much experimentation and thought, I realized that I could use the fact that the resistor wires were much thinner than the resistor body to isolate the resistor.

I ended up using the cv2 erode operation to remove the wires from the image in the end, the number of erosions it takes to erode the wires away will be much less than the amount to erode the resistor body.

However, as I wanted to create a robust resistor body locator, I used coordinate geometry to help me identify the number of erode iterations needed to remove the wires from the image rather than just hard coding a value that works for my dataset.

Thoughts

This algorithm was very hard to think of as there is no easy Google search to find how to tackle this problem. I had to spend much time just experimenting with different cv2 operations and think about whether I could put them to use when locating the resistor.

The biggest issue with the resistor location were the wires on the resistor being bent in different directions meant I could not just assume that the resistor was within the middle of wherever the image is not white.

I have ensured that my resistor body locator algorithm works on resistors of all sizes and wire lengths and shapes as it is important, I create a robust application as I do not know the exact use case for the user, and I want to make the application as easy and simple to use as possible.

Band Location and Colour Identification

Issues

Developing the band locator was much harder than expected. The biggest issues I encountered along the way were:

- The glare on the resistor body causing edge detection on the bands to fail.
- The gold bands blending into the background of resistor.
- The speckle inside the gold band making causing unwanted noise in outputs.
- The fact that the resistor image was not smooth (had much random noise).
- Not having enough data to make good estimates of the band colours.
- Different light conditions make the colour identification inaccurate.

Addressing the Issues

However, I managed to find ways around these issues within my final implementation.

I addressed the issue of glare on the resistor by using my own implementation of K-Means to identify glare pixels and I extracted a section of the image with no glare. Despite developing my own implementation of K-Means, I have used the Sci-Kit library for this purpose due to how much faster it runs. If had more time to develop my application, I would focus on optimizing my K-Means implementation so that I can use it for the glare identification.

I addressed the issue of gold bands blending into the background the resistor by widening my HSV ranges to be more lenient. Doing this meant more noise/incorrect bands were picked up, but filtering algorithms, weighting and majority voting allowed me to get around this issue.

I addressed random noise and the speckle within the gold band by doing a light vertical blur on the resistor before masking the HSV ranges, this blur is much lighter than the blur I initially used to remove glare.

I addressed the issue of not having enough data by slicing the image of the resistor body into 20 smaller slices which I mask the bands for. This gives me much data to work with for later stages.

Final Implementation

My final implementation of the band location and colour identification is an extension to what can be seen within the prototype stage.

I discovered and expanded the HSV ranges by using the tool I implemented within my program that allowed me to click on any pixel of the image and obtain the HSV value for that pixel. I have done this on several sets of resistor images to ensure I have a wide variety of data and I have added some leeway to accommodate for slightly differing conditions.

After, adding this lenience to the HSV ranges, I found that extra noise/incorrect areas were picked up. To counteract this, instead of fine-tuning the HSV ranges to 1 set of resistor images, I chose to come up with algorithms that remove bands that do not fulfil certain conditions.

Band Filtering

The concept of band filtering was a biproduct of the other algorithms within the program. After tweaking the HSV ranges to be more robust and correct, I had to find a way to distinguish real bands from random masked “noise” that came because of the more lenient HSV ranges.

Issues

While trying to filter the bands, the issues I encountered were:

- Outliers skewing the data.
- Knowing what conditions that I could set to identify a band as an acceptable mask of the colour.

Addressing the Issues

To address the issue of outliers skewing the data, I developed algorithms to remove obvious outliers, such as those which are too narrow or too short to possibly be a good mask for a band.

I eventually realized that the after masking the resistor bands, I will find clusters of bands around certain areas within the image (the actual band locations). If I could identify these areas of the image, I could then have data that allowed me to choose whether a band is a possible contender for being the actual resistor band. After researching algorithms to find clusters, I found K-Means. I could use K-Means to identify the x locations at which the clusters of bands occurred, all I had to do was provide my K-Means the correct value for the number of clusters. I tried using the knee method to identify this optimal number of clusters, however, I found that calculating the Dunn Index for a range of values of K provided better results.

Resistor Band Identification

Issues

My initial idea to identify the resistor band was to majority vote it in from a list of possible bands for the band. However, I quickly found out that my lenient black and white HSV colour ranges caused these colours to be the most frequent colour for many of the bands.

Another issue I encountered was that I would have bands at positions that they could not possibly be at – as it would make the resistor invalid.

Another issue I encountered was the orientation of the resistor. The user’s input image may not match the correct orientation of the resistor for its correct resistor values.

Addressing the Issues

To solve the issue of lenient HSV ranges, I implemented a weighting system into my program. This weighting system meant that I could prioritize certain colours over others. By giving colours with overlapping/problematic HSV ranges weights that corresponded to the priority that they took over each other, I could still majority vote the bands in, however with this new weighted value as the frequency of the colour. After experimenting with the colour weights, I managed to greatly decrease the number of times problematic colours would overwrite each other.

To solve the issue of bands being in impossible positions, I made my program check the position of gold and silver bands. As gold and silver bands cannot possibly be in the for most of the bands in the middle of the resistor, I added an algorithm that increases or decreases the weight of silver and gold depending on its position within the resistor.

To address the issue of the resistor orientation, I researched resistors. After researching resistors, I found that they have standard digit values. After finding this out, I implemented an algorithm that flipped the resistor until it is in an orientation that satisfies a standard digit value. This greatly increased the accuracy of the output of my program.

Webserver

Having written Flask based applications before, writing the webserver for my program did not prove that difficult. After refreshing my knowledge on the Python Flask library and XML HTTP requests, I could write the webserver prototype for sending an image. After creating the prototype, I improved the file structure by splitting up the HTML and JavaScript, as well as adding a CSS file, so I could style the page and navigate between my code more easily.

Critical Path Diagrams

The critical path diagrams I made at the start of the project have proven to be a useful tool to track my progress through the development stages of my program. I have stuck to this for the most part, and it has helped keep me on track and knowing what to do, however for some parts when I have encountered an issue I did not fore-see, I have had to stray away to rethink a solution or implement a fix.

Final Thoughts

I would never have predicted that types of difficult issues I had to solve to get my program to work until I started actual development and prototyping my project. Some of the issues that I have mentioned above took a very long time to come up with a solution to and required much thought and research, however I am very glad that I spent the effort, ensuring I was able to produce a working system.

Despite this, and although it has taken much longer than expected, I have thoroughly enjoyed working on my project and I believe that I have successfully produced a system that is a solution for the problem of detecting a resistor's values from an image. I am proud of my project as:

- My system is fully working and makes use of many advanced image processing and non-image processing algorithms to achieve its final goal.
- The GUI I have created is better than the concept I designed before developing the implementation.
- There is sufficient error handling to inform the user of what went wrong if an image does not work.

Appendix

All the code for my application can be found below. Because my program is across multiple files, I have pasted the code from each file into separate tables.

Webserver

Python Code

```
import json
import os.path
from os.path import join, dirname, realpath

from flask import Flask, request, render_template, jsonify

from detection.Detector import Detector
from detection.Resistor import Resistor
from detection.ResistorBand import ResistorBand

UPLOADS_PATH = join(dirname(realpath(__file__)))
app = Flask(__name__, template_folder='resources', static_folder='resources/static/')


# Returns the UI of the page.
@app.route('/ui', methods=['GET'])
def ui():
    return render_template('Resistor.html')


# Calls the detector program.
@app.route('/api/detect', methods=['POST'])
def detect():

    file = request.files['file']

    if request.values:
        resistor_type = int(request.values['type'])

    else:
        resistor_type = None

    location = f'{os.getcwd()}\\data\\images\\{file.filename}'

    try:

        with open(location, 'wb') as target:
            file.save(target)

        resistor, resistor_image = Detector().detect(location, resistor_type)

        resistor_image_byte_stream = resistor_image.byte_stream()

        resistor_image_byte_stream = resistor_image_byte_stream.decode('utf-8')

        resistor_colours = resistor.colours()

        if resistor_type is None:
            resistor_type = resistor.get_number_of_bands()

        return jsonify(colours=resistor_colours, type=resistor_type, image=resistor_image_byte_stream,
                       valid=resistor.valid, error='')

    except Exception as error:
```

```
        print(error)

        colours = [None, None, None, None, None, None]

        if resistor_type is None:
            resistor_type = 6

        return jsonify(colours=colours, type=resistor_type, image=None, valid=False, error=str(error))


@app.route('/api/validate', methods=['POST'])
def validate():
    resistor_bands_colours = request.values["resistor_bands"]

    resistor_bands_colours = json.loads(resistor_bands_colours)

    resistor = Resistor(resistor_bands_colours)

    digit_band_colours = resistor.get_digit_band_colours(resistor_bands_colours)

    valid = resistor.check_valid(digit_band_colours)

    return jsonify(valid=valid)


if __name__ == '__main__':
    app.run()
```

HTML Page

```
<!DOCTYPE html>
<html lang="en">
<meta name="viewport" content="width=device-width, maximum-scale=1.0"/>
<link rel="preconnect" href="https://fonts.googleapis.com">
<link rel="preconnect" href="https://fonts.gstatic.com" crossorigin>
<link href="https://fonts.googleapis.com/css2?family=Questrial&display=swap" rel="stylesheet">
<link rel="stylesheet" type="text/css" href="static/ResistorCSS.css">
<link rel="icon"
      type="image/png"
      href="https://images-na.ssl-images-amazon.com/images/I/716cFc4OXFL.png">
<meta name="viewport" content="width=device-width, initial-scale=0.5, maximum-scale=1" />

<body>
    <div id="banner">

        <div id="banner-content">Resistor Calculator</div>

        <div id="error-banner" class="hide_banner">

            <div id="error-banner-content">Error.</div>

            <button id="close_banner" class="banner-button" onclick="close_banner()">X</button>

        </div>

    </div>


    <div id="main-content">

        <table class="space_bottom">

            <tr>

                <td><p style="color:white">Resistance</p></td>

                <td><p style="color:white">Tol.</p></td>

                <td><p style="color:white">Temp. Co.</p></td>

                <td><p style="color:white">Valid</p></td>
```

```
            </tr>

            <tr>

                <td><button id="resistance"
                            class="output_window add_curve">N/A</button></td>

                <td><button id="tolerance"
                            class="output_window add_curve">N/A</button></td>

                <td><button id="temperature_constant"
                            class="output_window add_curve">N/A</button></td>

                <td><button id="valid"
                            class="output_window add_curve">N/A</button></td>


            </tr>

        </table>

        <table class="space_top">

            <tr>

                <td><input type="button"
                           id="select_button"
                           value="Input Resistor Image"
                           class="input_button input_image_button add_curve"
                           onclick="clickButton('select_input')"/></td>

                <td><input type="file"
                           id="select_input"
                           style="display:NONE;"
                           accept="image/*"
                           onchange="drawFileAndShowScanButton()"/></td>

            </tr>

            <tr>
                <td><input type="button"
                           id="scan_button"
                           value="Scan Resistor"
                           class="input_button input_image_button add_curve"
                           style="display:NONE;"
                           onclick="uploadFile()"/></td>
            </tr>

        </table>

        <table class="space_bottom">

            <tr>
                <td><input type="button"
                           id="3_band"
                           value="3 Band"
                           class="input_button band_amount_button add_curve"
                           onclick="resistorType(3)"/></td>

                <td><input type="button"
                           id="4_band"
                           value="4 Band"
                           class="input_button band_amount_button add_curve"
                           onclick="resistorType(4)"/></td>

                <td><input type="button"
                           id="5_band"
                           value="5 Band"
                           class="input_button band_amount_button add_curve"
                           onclick="resistorType(5)"/></td>

                <td><input type="button"
                           id="6_band"
                           value="6 Band"
```

```
                        class="input_button band_amount_button add_curve add_selected"
                        onclick="resistorType(6)"/></td>

                <td><input type="button"
                           class="input_button band_amount_button add_curve"
                           id="resistor_type_lock"
                           onclick="lockResistorType()"
                           value="🔒"/></td>

            </tr>

        </table>

        <table>

            <tr>

                <td>

                    <canvas id="canvas_id" class="hide_canvas"></canvas>

                </td>

            </tr>

        </table>

        <table>

            <tr>

                <td id="band_1_header"><p style="color:white">1</p></td>

                <td id="band_2_header"><p style="color:white">2</p></td>

                <td id="band_3_header"><p style="color:white">3</p></td>

                <td id="band_4_header"><p style="color:white">Mul</p></td>

                <td id="band_5_header"><p style="color:white">Tol</p></td>

                <td id="band_6_header"><p style="color:white">T.C.</p></td>

            </tr>

            <tr>
                <td id="band_1">

                    <input type="button"
                           id="band_button_1_BLACK"
                           class="band_button add_curve band_1 BLACK"
                           onclick="bandButtonPress(1, 'BLACK')"/>

                    <input type="button"
                           id="band_button_1_BROWN"
                           class="band_button add_curve band_1 BROWN"
                           onclick="bandButtonPress(1, 'BROWN')"/>

                    <input type="button"
                           id="band_button_1_RED"
                           class="band_button add_curve band_1 RED"
                           onclick="bandButtonPress(1, 'RED')"/>

                    <input type="button"
                           id="band_button_1_ORANGE"
                           class="band_button add_curve band_1 ORANGE"
                           onclick="bandButtonPress(1, 'ORANGE')"/>

                    <input type="button"
                           id="band_button_1_YELLOW"
                           class="band_button add_curve band_1 YELLOW"
                           onclick="bandButtonPress(1, 'YELLOW')"/>

                    <input type="button"
```

```
                        id="band_button_1_GREEN"
                        class="band_button add_curve band_1 GREEN"
                        onclick="bandButtonPress(1, 'GREEN')"/>

                <input type="button"
                        id="band_button_1_BLUE"
                        class="band_button add_curve band_1 BLUE"
                        onclick="bandButtonPress(1, 'BLUE')"/>

                <input type="button"
                        id="band_button_1_VIOLET"
                        class="band_button add_curve band_1 VIOLET"
                        onclick="bandButtonPress(1, 'VIOLET')"/>

                <input type="button"
                        id="band_button_1_GREY"
                        class="band_button add_curve band_1 GREY"
                        onclick="bandButtonPress(1, 'GREY')"/>

                <input type="button"
                        id="band_button_1_WHITE"
                        class="band_button add_curve band_1 WHITE"
                        onclick="bandButtonPress(1, 'WHITE')"/>

            </td>

            <td id="band_2">

                <input type="button"
                        id="band_button_2_BLACK"
                        class="band_button add_curve band_2 BLACK"
                        onclick="bandButtonPress(2, 'BLACK')"/>

                <input type="button"
                        id="band_button_2_BROWN"
                        class="band_button add_curve band_2 BROWN"
                        onclick="bandButtonPress(2, 'BROWN')"/>

                <input type="button"
                        id="band_button_2_RED"
                        class="band_button add_curve band_2 RED"
                        onclick="bandButtonPress(2, 'RED')"/>

                <input type="button"
                        id="band_button_2_ORANGE"
                        class="band_button add_curve band_2 ORANGE"
                        onclick="bandButtonPress(2, 'ORANGE')"/>

                <input type="button"
                        id="band_button_2_YELLOW"
                        class="band_button add_curve band_2 YELLOW"
                        onclick="bandButtonPress(2, 'YELLOW')"/>

                <input type="button"
                        id="band_button_2_GREEN"
                        class="band_button add_curve band_2 GREEN"
                        onclick="bandButtonPress(2, 'GREEN')"/>

                <input type="button"
                        id="band_button_2_BLUE"
                        class="band_button add_curve band_2 BLUE"
                        onclick="bandButtonPress(2, 'BLUE')"/>

                <input type="button"
                        id="band_button_2_VIOLET"
                        class="band_button add_curve band_2 VIOLET"
                        onclick="bandButtonPress(2, 'VIOLET')"/>

                <input type="button"
                        id="band_button_2_GREY"
                        class="band_button add_curve band_2 GREY"
                        onclick="bandButtonPress(2, 'GREY')"/>

                <input type="button"
```

```
                    id="band_button_2_WHITE"
                    class="band_button add_curve band_2
                    WHITE" onclick="bandButtonPress(2, 'WHITE')"/>

            </td>

            <td id="band_3">

                <input type="button"
                    id="band_button_3_BLACK"
                    class="band_button add_curve band_3 BLACK"
                    onclick="bandButtonPress(3, 'BLACK')"/>

                <input type="button"
                    id="band_button_3_BROWN"
                    class="band_button add_curve band_3 BROWN"
                    onclick="bandButtonPress(3, 'BROWN')"/>

                <input type="button"
                    id="band_button_3_RED"
                    class="band_button add_curve band_3 RED"
                    onclick="bandButtonPress(3, 'RED')"/>

                <input type="button"
                    id="band_button_3_ORANGE"
                    class="band_button add_curve band_3 ORANGE"
                    onclick="bandButtonPress(3, 'ORANGE')"/>

                <input type="button"
                    id="band_button_3_YELLOW"
                    class="band_button add_curve band_3 YELLOW"
                    onclick="bandButtonPress(3, 'YELLOW')"/>

                <input type="button"
                    id="band_button_3_GREEN"
                    class="band_button add_curve band_3 GREEN"
                    onclick="bandButtonPress(3, 'GREEN')"/>

                <input type="button"
                    id="band_button_3_BLUE"
                    class="band_button add_curve band_3 BLUE"
                    onclick="bandButtonPress(3, 'BLUE')"/>

                <input type="button"
                    id="band_button_3_VIOLET"
                    class="band_button add_curve band_3 VIOLET"
                    onclick="bandButtonPress(3, 'VIOLET')"/>

                <input type="button"
                    id="band_button_3_GREY"
                    class="band_button add_curve band_3 GREY"
                    onclick="bandButtonPress(3, 'GREY')"/>

                <input type="button"
                    id="band_button_3_WHITE"
                    class="band_button add_curve band_3 WHITE"
                    onclick="bandButtonPress(3, 'WHITE')"/>

            </td>

            <td id="band_4">

                <input type="button"
                    id="band_button_4_BLACK"
                    class="band_button add_curve band_4 BLACK"
                    onclick="bandButtonPress(4, 'BLACK')"/>

                <input type="button"
                    id="band_button_4_BROWN"
                    class="band_button add_curve band_4 BROWN"
                    onclick="bandButtonPress(4, 'BROWN')"/>

                <input type="button"
                    id="band_button_4_RED"
```

```
                        class="band_button add_curve band_4 RED"
                        onclick="bandButtonPress(4, 'RED')"/>

                <input type="button"
                        id="band_button_4_ORANGE"
                        class="band_button add_curve band_4 ORANGE"
                        onclick="bandButtonPress(4, 'ORANGE')"/>

                <input type="button"
                        id="band_button_4_YELLOW"
                        class="band_button add_curve band_4 YELLOW"
                        onclick="bandButtonPress(4, 'YELLOW')"/>

                <input type="button"
                        id="band_button_4_GREEN"
                        class="band_button add_curve band_4 GREEN"
                        onclick="bandButtonPress(4, 'GREEN')"/>

                <input type="button"
                        id="band_button_4_BLUE"
                        class="band_button add_curve band_4 BLUE"
                        onclick="bandButtonPress(4, 'BLUE')"/>

                <input type="button"
                        id="band_button_4_VIOLET"
                        class="band_button add_curve band_4 VIOLET"
                        onclick="bandButtonPress(4, 'VIOLET')"/>

                <input type="button"
                        id="band_button_4_GREY"
                        class="band_button add_curve band_4 GREY"
                        onclick="bandButtonPress(4, 'GREY')"/>

                <input type="button"
                        id="band_button_4_WHITE"
                        class="band_button add_curve band_4 WHITE"
                        onclick="bandButtonPress(4, 'WHITE')"/>

                <input type="button"
                        id="band_button_4_GOLD"
                        class="band_button add_curve band_4 GOLD"
                        onclick="bandButtonPress(4, 'GOLD')"/>

                <input type="button"
                        id="band_button_4_SILVER"
                        class="band_button add_curve band_4 SILVER"
                        onclick="bandButtonPress(4, 'SILVER')"/>

            </td>

            <td id="band_5">

                <button class="band_button add_curve band_5 NONE"></button>

                <input type="button"
                        id="band_button_5_BROWN"
                        class="band_button add_curve band_5 BROWN"
                        onclick="bandButtonPress(5, 'BROWN')"/>

                <input type="button"
                        id="band_button_5_RED"
                        class="band_button add_curve band_5 RED"
                        onclick="bandButtonPress(5, 'RED')"/>

                <input type="button"
                        id="band_button_5_ORANGE"
                        class="band_button add_curve band_5 ORANGE"
                        onclick="bandButtonPress(5, 'ORANGE')"/>

                <input type="button"
                        id="band_button_5_YELLOW"
                        class="band_button add_curve band_5 YELLOW"
                        onclick="bandButtonPress(5, 'YELLOW')"/>
```

```
                    <input type="button"
                           id="band_button_5_GREEN"
                           class="band_button add_curve band_5 GREEN"
                           onclick="bandButtonPress(5, 'GREEN')"/>

                    <input type="button"
                           id="band_button_5_BLUE"
                           class="band_button add_curve band_5 BLUE"
                           onclick="bandButtonPress(5, 'BLUE')"/>

                    <input type="button"
                           id="band_button_5_VIOLET"
                           class="band_button add_curve band_5 VIOLET"
                           onclick="bandButtonPress(5, 'VIOLET')"/>

                    <input type="button"
                           id="band_button_5_GREY"
                           class="band_button add_curve band_5 GREY"
                           onclick="bandButtonPress(5, 'GREY')"/>

                    <button class="band_button add_curve band_5 NONE"></button>

                    <input type="button"
                           id="band_button_5_GOLD"
                           class="band_button add_curve band_5 GOLD"
                           onclick="bandButtonPress(5, 'GOLD')"/>

                    <input type="button"
                           id="band_button_5_SILVER"
                           class="band_button add_curve band_5 SILVER"
                           onclick="bandButtonPress(5, 'SILVER')"/>

                </td>

                <td id="band_6">

                    <button class="band_button add_curve band_6 NONE"></button>

                    <input type="button"
                           id="band_button_6_BROWN"
                           class="band_button add_curve band_6 BROWN"
                           onclick="bandButtonPress(6, 'BROWN')"/>

                    <input type="button"
                           id="band_button_6_RED"
                           class="band_button add_curve band_6 RED"
                           onclick="bandButtonPress(6, 'RED')"/>

                    <input type="button"
                           id="band_button_6_ORANGE"
                           class="band_button add_curve band_6 ORANGE"
                           onclick="bandButtonPress(6, 'ORANGE')"/>

                    <input type="button"
                           id="band_button_6_YELLOW"
                           class="band_button add_curve band_6 YELLOW"
                           onclick="bandButtonPress(6, 'YELLOW')"/>

                    <button class="band_button add_curve band_6 NONE"></button>

                    <input type="button"
                           id="band_button_6_BLUE"
                           class="band_button add_curve band_6 BLUE"
                           onclick="bandButtonPress(6, 'BLUE')"/>

                    <input type="button"
                           id="band_button_6_VIOLET"
                           class="band_button add_curve band_6 VIOLET"
                           onclick="bandButtonPress(6, 'VIOLET')"/>

                </td>

            </tr>
```

```
        </table>

    </div>

</body>

<script src="static/ResistorJS.js"></script>
```

CSS

```
body {
    background-color: #36393F;
}

td {
    text-align: center;
    padding: 0
}

table {
    margin-left:auto;
    margin-right:auto;
    border: 0;
}

.input_button {
    color: #ffffff;
    background-color: #5865F2;
    border: solid 0;
    text-align: center;
    font-family: 'Questrial', sans-serif;
    font-size: 18px;
    margin: 5px 5px;
    cursor: pointer;
    position: relative;
}

.band_button {
    color: #000000;
    border: solid 0;
    text-decoration: NONE;
    margin: 10px 5px;
    cursor: pointer;
    display: block;
    width: 60px;
    height: 50px;
    position: relative;
}


.band_1 {
    top: -60px;
}

.band_2 {
    top: -60px;
}

.band_3 {
    top: -60px;
}

.band_4 {
}

.band_5 {
}

.band_6 {
    top: -120px;
}
```

```
.BLACK {
    background-color: #1a1a1a;
}

.BROWN {
    background-color: #664100;
}

.RED {
    background-color: #ff3333;
}

.ORANGE {
    background-color: #ff7308;
}

.YELLOW {
    background-color: #ffd966;
}

.GREEN {
    background-color: #99ff33;
}

.BLUE {
    background-color: #6666ff;
}

.VIOLET {
    background-color: #ff66b3;
}

.GREY {
    background-color: #8f8f8f;
}

.DARKGREY {
    background-color: #202225;
}

.WHITE {
    background-color: #ffffff;
}

.GOLD {
     background-image: linear-gradient(to top, #F09819 0%, #EDDE5D  51%, #f09819 100%)
}

.SILVER {
    background-image: linear-gradient(to top, #8f8f8f 0%, #ffffff 51%, #8a8a8a 100%)
}

.NONE {
    background-color: Transparent;
    background-repeat:no-repeat;
    border: NONE;
    cursor: default;
    overflow: hidden;
    outline: NONE;
}

.add_margin{
    margin: 5px 3px;
}

.add_curve{
    border-radius: 8px;
}

.add_shadow{
    box-shadow: 0 5px 3px #aaaaaa;
}

.add_selected {
```

```
    border: solid 8px #000000;
}

.band_amount_button {
    width: 98px;
    height: 50px;
}

.input_image_button {
    width: 428px;
    height: 50px;
}

.output_window {
    background-color: #ffffff;
    border: none;
    color: black;
    width: 98px;
    height: 50px;
    text-align: center;
    display: inline-block;
    margin: 10px 5px;
    font-family: 'Questrial', sans-serif;
    font-size: 18px;
    cursor: default;
}

.space_top {
    padding-top: 25px
}

.space_bottom {
    padding-bottom: 25px;
}

.placeholder {
    font-style: italic;
    color: #8a8a8a;
}

.hide_canvas {
    width: 0;
    height: 0;
}


#banner {
    position: relative;
    background-color: #202225;
    width: 100%;
    background-size: 100%;
 }

#banner-content {
    max-width: 430px;
    margin: 0 auto;
    padding: 30px;
    color: white;
    text-align: center;
    font-family: 'Questrial', sans-serif;
    font-size: 30px;
 }

.hide_banner {
    display: none;
}

#error-banner {
    position: relative;
    background-color: #ff0000;
    width: 100%;
    background-size: 100%;
 }
```

```
#error-banner-content {
    max-width: 430px;
    margin: 0 auto;
    padding: 5px;
    color: white;
    text-align: center;
    font-family: 'Questrial', sans-serif;
    font-size: 15px;
 }

#main-content {
    max-width: 430px;
    margin: 0 auto;
    font-family: 'Questrial', sans-serif;
    font-size: 18px;
    padding: 0;
}
```

JavaScript

```
// defining commonly used elements
const select_input = document.getElementById('select_input');
const canvas = document.getElementById('canvas_id');
// picture canvas
var context = canvas.getContext('2d');

//defining variables
var image = new Image();

// defining resistor variables
var resistor_bands = [];
var resistor_type = 6;

// defining button tracking variables
var button_presses = [];
var band_presses = [];

var current_type_pressed = 6;

// locked status
var resistor_type_lock = false

var valid = false
var error_message = ''

// simulate button click
function clickButton(element_id) {
    document.getElementById(element_id).click();
}

function drawFileAndShowScanButton() {
    let file = select_input.files[0];
    drawFile(file);
    document.getElementById('scan_button').style.display = 'inline'
}

function uploadFile() {
    // getting uploaded file and its location
    let file = select_input.files[0];
    uploadAndResponse(file, resistor_type_lock);
}

// https://stackoverflow.com/a/40831598
function base64ToBlob(base_64_data, content_type) {
    let slice_size = 1024;
    let byte_characters = atob(base_64_data);
    let bytes_length = byte_characters.length;
    let slices_count = Math.ceil(bytes_length / slice_size);
    let byte_arrays = new Array(slices_count);

    for (let slice_index = 0; slice_index < slices_count; ++slice_index) {
        let begin = slice_index * slice_size;
```

```
        let end = Math.min(begin + slice_size, bytes_length);
        let bytes = new Array(end - begin);

        for (let offset = begin, i = 0 ; offset < end; ++i, ++offset) {
            bytes[i] = byte_characters[offset].charCodeAt(0);
        }
        byte_arrays[slice_index] = new Uint8Array(bytes);
    }
    return new Blob(byte_arrays, { type: content_type });
}

function validateResistor() {
    let form = new FormData();
    let xhr = new XMLHttpRequest();

    try {
        // posting the values to flask
        form.append('resistor_bands', JSON.stringify(resistor_bands))
        xhr.open('post', '/api/validate', true);
        xhr.send(form);


        // responses to the ajax post
        xhr.onreadystatechange = function () {
            if (xhr.readyState === XMLHttpRequest.DONE) {

                // change back to global if it doesn't work
                let response = JSON.parse(xhr.responseText)
                valid = response['valid'];
                outputValid()
            }
        }
    }

    catch {
        let error = 'Cannot validate result: cannot connect to webserver.'
        console.log(error)
        error_message = error
    }
}

// outputting errors to a banner for the user to see
function outputError() {
    var error_banner = document.getElementById('error-banner');
    var error_banner_content = document.getElementById('error-banner-content');

    if (error_message !== '') {
        if (error_banner.classList.contains('hide_banner')) {
            error_banner.classList.remove('hide_banner')
        }

        error_banner_content.innerText = error_message
    }

    else {

        if (!(error_banner.classList.contains('hide_banner'))) {
            error_banner.classList.add('hide_banner')

        }
    }
}

// closing the banner if the banner close button is clicked
function close_banner() {
    var error_banner = document.getElementById('error-banner');

    if (!(error_banner.classList.contains('hide_banner'))) {
            error_banner.classList.add('hide_banner')
    }
}

function uploadAndResponse(file, resistor_type_lock) {
```

```
    // creating variables for ajax
    let form = new FormData();
    let xhr = new XMLHttpRequest();

    if (resistor_type_lock === true) {
        form.append('type', resistor_type);
    }

    // posting the values to flask
    form.append('file', file);
    xhr.open('post', '/api/detect', true);
    xhr.send(form);

    // responses to the ajax post
    xhr.onreadystatechange = function () {
        if (xhr.readyState === XMLHttpRequest.DONE) {

            // change back to global if it doesn't work
            let resistor = JSON.parse(xhr.responseText)

            let resistor_bands = resistor['colours'];
            let number_of_bands = resistor['type'];

            let resistor_image_byte_stream = resistor['image'];
            valid = resistor['valid'];
            error_message = resistor['error'];

            outputValid()
            outputError()

            console.log('COLOURS: ' + resistor_bands)
            console.log('BANDS: ' + number_of_bands)

            // calculating values for resistor
            resistorType(number_of_bands)
            outputResistorValues(processResistor(resistor_bands[0], resistor_bands[1], resistor_bands[2],
resistor_bands[3], resistor_bands[4], resistor_bands[5]))

            if (resistor_image_byte_stream !== null) {
                // displaying resistor image
                let resistor_image_blob = base64ToBlob(resistor_image_byte_stream, 'image/jpeg')

                drawFile(resistor_image_blob)
            }

            // selecting bands
            let index = 0;

            for (; index < resistor_bands.length; index++) {
                selectBandButton(index + 1, resistor_bands[index])
            }
        }
    }

}

function calculateHeight() {
    let ratio = image.width / image.height

    let scale = 420 / ratio

    return Math.round(scale)
}

function drawFile(file) {
    let reader = new FileReader();

    canvas.classList.remove('hide_canvas')

    reader.onload = function (e) {
        let dataURL = e.target.result

    image.onload = function() {
        canvas.width = 420
```

```
        canvas.height = calculateHeight()

        context.drawImage(image, 0, 0, image.width, image.height, 0, 0,  canvas.width,  canvas.height);
        }
        image.src = dataURL;
    }

    reader.readAsDataURL(file);
  }


function stopSelected(element_id) {
    try {
        document.getElementById(element_id).classList.remove('add_selected');
    }
    catch {
        console.log('Error deselecting button (maybe button does not exist).')
    }
}


function addSelected(element_id) {
    try {
        document.getElementById(element_id).classList.add('add_selected');
    }
    catch {
        console.log('Specified element id does not exist.')
    }
}

function checkBandPressDupes(band) {
    if (band_presses.includes(band)) {

        let index = band_presses.indexOf(band);
        let dupe_flash_element = button_presses[index];

        delete button_presses[index];
        delete band_presses[index];

        stopSelected(dupe_flash_element);
    }
}


function findElementId(band, colour) {
    band.toString()

    let element_target = ('_' + band + '_' + colour)

    return 'band_button' + element_target;
}


function selectBandButton(band, colour) {
    if (band !== 'NONE') {
        let element_id = findElementId(band, colour);

        checkBandPressDupes(band);

        button_presses.push(element_id);
        band_presses.push(band);
        addSelected(element_id)
    }
}

function bandButtonPress(band, colour) {
    selectBandButton(band, colour)

    if (band === 1) {
        resistor_bands[0] = colour
    }

    if (band === 2) {
        resistor_bands[1] = colour
```

```
    }

    if (band === 3) {
        resistor_bands[2] = colour
    }

    if (band === 4) {
        resistor_bands[3] = colour
    }

    if (band === 5) {
        resistor_bands[4] = colour

    }

    if (band === 6) {
        resistor_bands[5] = colour
    }

    validateResistor()
    outputValid()

    outputResistorValues(processResistor(resistor_bands[0], resistor_bands[1], resistor_bands[2],
resistor_bands[3], resistor_bands[4], resistor_bands[5]))
}

function getDigits(band_1, band_2, band_3) {
    let digits = {
        'BLACK':'0',
        'BROWN':'1',
        'RED':'2',
        'ORANGE':'3',
        'YELLOW':'4',
        'GREEN':'5',
        'BLUE':'6',
        'VIOLET':'7',
        'GREY':'8',
        'WHITE':'9'
    }

    let digit_1 = digits[band_1]
    let digit_2 = digits[band_2]
    let digit_3 = digits[band_3]

    if (typeof digit_1 === 'undefined') {
        digit_1 = ''
    }

    if (typeof digit_2 === 'undefined') {
        digit_2 = ''
    }

    if (typeof digit_3 === 'undefined') {
        digit_3 = ''
    }

    return digit_1 + digit_2 + digit_3
}

function getMultiplier(colour) {
    let multipliers = {
        'BLACK':1,
        'BROWN':10,
        'RED':100,
        'ORANGE':1000,
        'YELLOW':10000,
        'GREEN':100000,
        'BLUE':1000000,
        'VIOLET':10000000,
        'GREY':100000000,
        'WHITE':1000000000,
        'GOLD':0.1,
        'SILVER':0.01
    }
```

```
    let multiplier = multipliers[colour]

    if (typeof multiplier === 'undefined') {
        return 1
    }
    return multiplier
}


function getTolerance(colour) {
    let tolerances = {
        'BLACK':0,
        'BROWN':1,
        'RED':2,
        'ORANGE':3,
        'YELLOW':4,
        'GREEN':0.5,
        'BLUE':0.25,
        'VIOLET':0.1,
        'GREY':0.05,
        'GOLD':5,
        'SILVER':10
    }

    if (typeof tolerances[colour] === 'undefined') {
        return undefined
    }
    return tolerances[colour]
}

function getTemperatureConstant(colour) {
    let temperature_constants = {
        'BLACK':0,
        'BROWN':100,
        'RED':50,
        'ORANGE':15,
        'YELLOW':25,
        'BLUE':10,
        'VIOLET':5
    }

    if (typeof temperature_constants[colour] === 'undefined') {
        return undefined
    }

    return temperature_constants[colour]
}

function shortenResistance(resistance) {
    let suffix_magnitude = Math.floor(Math.floor(Math.log10(resistance), 1000) / 3);
    resistance = +resistance.toFixed(5);

    console.log(resistance)

    let large_suffixes = {
        0: '',
        1: 'K',
        2: 'M',
        3: 'G',
        4: 'T',
        5: 'P',
    }

    if (resistance > 1) {
        let suffix = large_suffixes[suffix_magnitude];
        let mantissa = resistance / 10 ** (suffix_magnitude * 3);
        return mantissa.toString() + suffix;
    }

    else {
        return resistance
    }
```

```
}

function processResistor(band_1, band_2, band_3, band_4, band_5, band_6) {
    resistor_bands = [band_1, band_2, band_3, band_4, band_5, band_6]

    let digits = getDigits(band_1, band_2, band_3);
    let multiplier = getMultiplier(band_4);
    let tolerance = getTolerance(band_5);
    let temperature_constant = getTemperatureConstant(band_6);

    let resistance = digits * multiplier;

    resistance = shortenResistance(resistance)

    return {'resistance':resistance, 'tolerance':tolerance, 'temperature_constant':temperature_constant}
}


function removeTableColumns(band_amount) {
    deselectColumns()

    let inactive_bands = {
        3:['band_3', 'band_5', 'band_6'],
        4:['band_3', 'band_6'],
        5:['band_6']
    }

    let index = 0;

    let remove_columns = inactive_bands[band_amount];

    for (; index < remove_columns.length; index++) {

        document.getElementById(remove_columns[index]).style.display = 'none';
        document.getElementById(remove_columns[index] + '_header').style.display = 'none';
    }
}


function addTableColumns(band_amount) {
    let columns_for_band = {
        3:['band_1', 'band_2', 'band_4'],
        4:['band_1', 'band_2', 'band_4', 'band_5'],
        5:['band_1', 'band_2', 'band_3', 'band_4', 'band_5'],
        6:['band_1', 'band_2', 'band_3', 'band_4', 'band_5', 'band_6']
    }

    let index = 0;

    let add_columns = columns_for_band[band_amount];

    for (; index < add_columns.length; index++) {

        document.getElementById(add_columns[index]).style.display = '';
        document.getElementById(add_columns[index] + '_header').style.display = '';
    }
}


function resistorType(band_amount) {
    resistorTypeButtonPress(band_amount)

    if (band_amount < resistor_type) {
        removeTableColumns(band_amount);
    }

    else {
        addTableColumns(band_amount);
    }
    resistor_type = band_amount;
}

function deselectColumns() {
    let index = 0;
```

```
    for (; index < button_presses.length; index++) {

        if (typeof button_presses[index] !== 'undefined') {
            let element_id = button_presses[index];

            try {
                document.getElementById(element_id).classList.remove('add_selected');
            }
            catch {
                console.log('Deselected a button that does not exist.')
            }
        }
    }

    resistor_bands = [];
    button_presses = [];
    band_presses = [];

    // recalculating the resistance after a column is deselected
    outputResistorValues(processResistor(resistor_bands[0], resistor_bands[1], resistor_bands[2],
resistor_bands[3], resistor_bands[4], resistor_bands[5]))
}

function outputValid() {
    if (valid === true) {
        document.getElementById('valid').innerText = '✔';
    }
    else {
        document.getElementById('valid').innerText = '✕';
    }
}

function outputResistorValues(resistor_values) {
    // converting string to int
    let resistance = resistor_values['resistance']

    // outputting the resistance by updating HTML for specific elements
    document.getElementById('resistance').innerText = resistor_values['resistance'] + 'Ω';
    document.getElementById('tolerance').innerText = resistor_values['tolerance'] + '%';
    document.getElementById('temperature_constant').innerText = resistor_values['temperature_constant'] +
'ppm/K';

    // hiding the elements if no values are present
    if (resistance === 0) {
        document.getElementById('resistance').innerText = 'N/A';
    }

    if (typeof resistor_values['tolerance'] === 'undefined') {
        document.getElementById('tolerance').innerText = 'N/A';
    }

    if (typeof resistor_values['temperature_constant'] === 'undefined') {
        document.getElementById('temperature_constant').innerText = 'N/A';
    }
}

function lockResistorType() {
    let resistor_type_lock_element = document.getElementById('resistor_type_lock');

    if (resistor_type_lock_element.value === '🔓') {
        resistor_type_lock_element.value = '🔒';

        resistor_type_lock = true;

        resistor_type_lock_element.classList.add('add_selected')
    }

    else {
        resistor_type_lock_element.value = '🔓';

        resistor_type_lock = false;
```

```
        resistor_type_lock_element.classList.remove('add_selected')
    }

}

function resistorTypeButtonPress(type) {

    let new_type_pressed_element_id = type + '_band';
    let current_type_pressed_element_id = current_type_pressed.toString() + '_band';

    stopSelected(current_type_pressed_element_id);
    addSelected(new_type_pressed_element_id);

    current_type_pressed = type;

}

// defining commonly used elements
const select_input = document.getElementById('select_input');
const canvas = document.getElementById('canvas_id');
// picture canvas
var context = canvas.getContext('2d');

//defining variables
var image = new Image();

// defining resistor variables
var resistor_bands = [];
var resistor_type = 6;

// defining button tracking variables
var button_presses = [];
var band_presses = [];

var current_type_pressed = 6;

// locked status
var resistor_type_lock = false

var valid = false
var error_message = ''

// simulate button click
function clickButton(element_id) {
    document.getElementById(element_id).click();
}

function drawFileAndShowScanButton() {
    let file = select_input.files[0];
    drawFile(file);
    document.getElementById('scan_button').style.display = 'inline'
}

function uploadFile() {
    // getting uploaded file and its location
    let file = select_input.files[0];
    uploadAndResponse(file, resistor_type_lock);
}

// https://stackoverflow.com/a/40831598
function base64ToBlob(base_64_data, content_type) {
    let slice_size = 1024;
    let byte_characters = atob(base_64_data);
    let bytes_length = byte_characters.length;
    let slices_count = Math.ceil(bytes_length / slice_size);
    let byte_arrays = new Array(slices_count);

    for (let slice_index = 0; slice_index < slices_count; ++slice_index) {
        let begin = slice_index * slice_size;
        let end = Math.min(begin + slice_size, bytes_length);
        let bytes = new Array(end - begin);

        for (let offset = begin, i = 0 ; offset < end; ++i, ++offset) {
            bytes[i] = byte_characters[offset].charCodeAt(0);
```

```
        }
        byte_arrays[slice_index] = new Uint8Array(bytes);
    }
    return new Blob(byte_arrays, { type: content_type });
}

function validateResistor() {
    let form = new FormData();
    let xhr = new XMLHttpRequest();

    try {
        // posting the values to flask
        form.append('resistor_bands', JSON.stringify(resistor_bands))
        xhr.open('post', '/api/validate', true);
        xhr.send(form);


        // responses to the ajax post
        xhr.onreadystatechange = function () {
            if (xhr.readyState === XMLHttpRequest.DONE) {

                // change back to global if it doesn't work
                let response = JSON.parse(xhr.responseText)
                valid = response['valid'];
                outputValid()
            }
        }
    }

    catch {
        let error = 'Cannot validate result: cannot connect to webserver.'
        console.log(error)
        error_message = error
    }
}

// outputting errors to a banner for the user to see
function outputError() {
    var error_banner = document.getElementById('error-banner');
    var error_banner_content = document.getElementById('error-banner-content');

    if (error_message !== '') {
        if (error_banner.classList.contains('hide_banner')) {
            error_banner.classList.remove('hide_banner')
        }

        error_banner_content.innerText = error_message
    }

    else {

        if (!(error_banner.classList.contains('hide_banner'))) {
            error_banner.classList.add('hide_banner')

        }
    }
}

// closing the banner if the banner close button is clicked
function close_banner() {
    var error_banner = document.getElementById('error-banner');

    if (!(error_banner.classList.contains('hide_banner'))) {
            error_banner.classList.add('hide_banner')
    }
}

function uploadAndResponse(file, resistor_type_lock) {

    // creating variables for ajax
    let form = new FormData();
    let xhr = new XMLHttpRequest();

    if (resistor_type_lock === true) {
```

```
        form.append('type', resistor_type);
    }

    // posting the values to flask
    form.append('file', file);
    xhr.open('post', '/api/detect', true);
    xhr.send(form);

    // responses to the ajax post
    xhr.onreadystatechange = function () {
        if (xhr.readyState === XMLHttpRequest.DONE) {

            // change back to global if it doesn't work
            let resistor = JSON.parse(xhr.responseText)

            let resistor_bands = resistor['colours'];
            let number_of_bands = resistor['type'];

            let resistor_image_byte_stream = resistor['image'];
            valid = resistor['valid'];
            error_message = resistor['error'];

            outputValid()
            outputError()

            console.log('COLOURS: ' + resistor_bands)
            console.log('BANDS: ' + number_of_bands)

            // calculating values for resistor
            resistorType(number_of_bands)
            outputResistorValues(processResistor(resistor_bands[0], resistor_bands[1], resistor_bands[2],
resistor_bands[3], resistor_bands[4], resistor_bands[5]))

            if (resistor_image_byte_stream !== null) {
                // displaying resistor image
                let resistor_image_blob = base64ToBlob(resistor_image_byte_stream, 'image/jpeg')

                drawFile(resistor_image_blob)
            }

            // selecting bands
            let index = 0;

            for (; index < resistor_bands.length; index++) {
                selectBandButton(index + 1, resistor_bands[index])
            }
        }
    }

}

function calculateHeight() {
    let ratio = image.width / image.height

    let scale = 420 / ratio

    return Math.round(scale)
}

function drawFile(file) {
    let reader = new FileReader();

    canvas.classList.remove('hide_canvas')

    reader.onload = function (e) {
        let dataURL = e.target.result

    image.onload = function() {
        canvas.width = 420
        canvas.height = calculateHeight()

        context.drawImage(image, 0, 0, image.width, image.height, 0, 0,  canvas.width,  canvas.height);
        }
        image.src = dataURL;
```

```
    }

    reader.readAsDataURL(file);
  }


function stopSelected(element_id) {
    try {
        document.getElementById(element_id).classList.remove('add_selected');
    }
    catch {
        console.log('Error deselecting button (maybe button does not exist).')
    }
}


function addSelected(element_id) {
    try {
        document.getElementById(element_id).classList.add('add_selected');
    }
    catch {
        console.log('Specified element id does not exist.')
    }
}

function checkBandPressDupes(band) {
    if (band_presses.includes(band)) {

        let index = band_presses.indexOf(band);
        let dupe_flash_element = button_presses[index];

        delete button_presses[index];
        delete band_presses[index];

        stopSelected(dupe_flash_element);
    }
}


function findElementId(band, colour) {
    band.toString()

    let element_target = ('_' + band + '_' + colour)

    return 'band_button' + element_target;
}


function selectBandButton(band, colour) {
    if (band !== 'NONE') {
        let element_id = findElementId(band, colour);

        checkBandPressDupes(band);

        button_presses.push(element_id);
        band_presses.push(band);
        addSelected(element_id)
    }
}

function bandButtonPress(band, colour) {
    selectBandButton(band, colour)

    if (band === 1) {
        resistor_bands[0] = colour
    }

    if (band === 2) {
        resistor_bands[1] = colour
    }

    if (band === 3) {
        resistor_bands[2] = colour
    }
```

```
    if (band === 4) {
        resistor_bands[3] = colour
    }

    if (band === 5) {
        resistor_bands[4] = colour

    }

    if (band === 6) {
        resistor_bands[5] = colour
    }

    validateResistor()
    outputValid()

    outputResistorValues(processResistor(resistor_bands[0], resistor_bands[1], resistor_bands[2],
resistor_bands[3], resistor_bands[4], resistor_bands[5]))
}

function getDigits(band_1, band_2, band_3) {
    let digits = {
        'BLACK':'0',
        'BROWN':'1',
        'RED':'2',
        'ORANGE':'3',
        'YELLOW':'4',
        'GREEN':'5',
        'BLUE':'6',
        'VIOLET':'7',
        'GREY':'8',
        'WHITE':'9'
    }

    let digit_1 = digits[band_1]
    let digit_2 = digits[band_2]
    let digit_3 = digits[band_3]

    if (typeof digit_1 === 'undefined') {
        digit_1 = ''
    }

    if (typeof digit_2 === 'undefined') {
        digit_2 = ''
    }

    if (typeof digit_3 === 'undefined') {
        digit_3 = ''
    }

    return digit_1 + digit_2 + digit_3
}

function getMultiplier(colour) {
    let multipliers = {
        'BLACK':1,
        'BROWN':10,
        'RED':100,
        'ORANGE':1000,
        'YELLOW':10000,
        'GREEN':100000,
        'BLUE':1000000,
        'VIOLET':10000000,
        'GREY':100000000,
        'WHITE':1000000000,
        'GOLD':0.1,
        'SILVER':0.01
    }

    let multiplier = multipliers[colour]

    if (typeof multiplier === 'undefined') {
        return 1
```

```
    }
    return multiplier
}


function getTolerance(colour) {
    let tolerances = {
        'BLACK':0,
        'BROWN':1,
        'RED':2,
        'ORANGE':3,
        'YELLOW':4,
        'GREEN':0.5,
        'BLUE':0.25,
        'VIOLET':0.1,
        'GREY':0.05,
        'GOLD':5,
        'SILVER':10
    }

    if (typeof tolerances[colour] === 'undefined') {
        return undefined
    }
    return tolerances[colour]
}

function getTemperatureConstant(colour) {
    let temperature_constants = {
        'BLACK':0,
        'BROWN':100,
        'RED':50,
        'ORANGE':15,
        'YELLOW':25,
        'BLUE':10,
        'VIOLET':5
    }

    if (typeof temperature_constants[colour] === 'undefined') {
        return undefined
    }

    return temperature_constants[colour]
}

function shortenResistance(resistance) {
    let suffix_magnitude = Math.floor(Math.floor(Math.log10(resistance), 1000) / 3);
    resistance = +resistance.toFixed(5);

    console.log(resistance)

    let large_suffixes = {
        0: '',
        1: 'K',
        2: 'M',
        3: 'G',
        4: 'T',
        5: 'P',
    }

    if (resistance > 1) {
        let suffix = large_suffixes[suffix_magnitude];
        let mantissa = resistance / 10 ** (suffix_magnitude * 3);
        return mantissa.toString() + suffix;
    }

    else {
        return resistance
    }

}

function processResistor(band_1, band_2, band_3, band_4, band_5, band_6) {
    resistor_bands = [band_1, band_2, band_3, band_4, band_5, band_6]
```

```
    let digits = getDigits(band_1, band_2, band_3);
    let multiplier = getMultiplier(band_4);
    let tolerance = getTolerance(band_5);
    let temperature_constant = getTemperatureConstant(band_6);

    let resistance = digits * multiplier;

    resistance = shortenResistance(resistance)

    return {'resistance':resistance, 'tolerance':tolerance, 'temperature_constant':temperature_constant}
}


function removeTableColumns(band_amount) {
    deselectColumns()

    let inactive_bands = {
        3:['band_3', 'band_5', 'band_6'],
        4:['band_3', 'band_6'],
        5:['band_6']
    }

    let index = 0;

    let remove_columns = inactive_bands[band_amount];

    for (; index < remove_columns.length; index++) {

        document.getElementById(remove_columns[index]).style.display = 'none';
        document.getElementById(remove_columns[index] + '_header').style.display = 'none';
    }
}


function addTableColumns(band_amount) {
    let columns_for_band = {
        3:['band_1', 'band_2', 'band_4'],
        4:['band_1', 'band_2', 'band_4', 'band_5'],
        5:['band_1', 'band_2', 'band_3', 'band_4', 'band_5'],
        6:['band_1', 'band_2', 'band_3', 'band_4', 'band_5', 'band_6']
    }

    let index = 0;

    let add_columns = columns_for_band[band_amount];

    for (; index < add_columns.length; index++) {

        document.getElementById(add_columns[index]).style.display = '';
        document.getElementById(add_columns[index] + '_header').style.display = '';
    }
}


function resistorType(band_amount) {
    resistorTypeButtonPress(band_amount)

    if (band_amount < resistor_type) {
        removeTableColumns(band_amount);
    }

    else {
        addTableColumns(band_amount);
    }
    resistor_type = band_amount;
}

function deselectColumns() {
    let index = 0;

    for (; index < button_presses.length; index++) {

        if (typeof button_presses[index] !== 'undefined') {
            let element_id = button_presses[index];
```

```
            try {
                document.getElementById(element_id).classList.remove('add_selected');
            }
            catch {
                console.log('Deselected a button that does not exist.')
            }
        }
    }

    resistor_bands = [];
    button_presses = [];
    band_presses = [];

    // recalculating the resistance after a column is deselected
    outputResistorValues(processResistor(resistor_bands[0], resistor_bands[1], resistor_bands[2],
resistor_bands[3], resistor_bands[4], resistor_bands[5]))
}

function outputValid() {
    if (valid === true) {
        document.getElementById('valid').innerText = '✔';
    }
    else {
        document.getElementById('valid').innerText = '✕';
    }
}

function outputResistorValues(resistor_values) {
    // converting string to int
    let resistance = resistor_values['resistance']

    // outputting the resistance by updating HTML for specific elements
    document.getElementById('resistance').innerText = resistor_values['resistance'] + 'Ω';
    document.getElementById('tolerance').innerText = resistor_values['tolerance'] + '%';
    document.getElementById('temperature_constant').innerText = resistor_values['temperature_constant'] +
'ppm/K';

    // hiding the elements if no values are present
    if (resistance === 0) {
        document.getElementById('resistance').innerText = 'N/A';
    }

    if (typeof resistor_values['tolerance'] === 'undefined') {
        document.getElementById('tolerance').innerText = 'N/A';
    }

    if (typeof resistor_values['temperature_constant'] === 'undefined') {
        document.getElementById('temperature_constant').innerText = 'N/A';
    }
}

function lockResistorType() {
    let resistor_type_lock_element = document.getElementById('resistor_type_lock');

    if (resistor_type_lock_element.value === '🔓') {
        resistor_type_lock_element.value = '🔒';

        resistor_type_lock = true;

        resistor_type_lock_element.classList.add('add_selected')
    }

    else {
        resistor_type_lock_element.value = '🔓';

        resistor_type_lock = false;

        resistor_type_lock_element.classList.remove('add_selected')
    }

}
```

```
function resistorTypeButtonPress(type) {

    let new_type_pressed_element_id = type + '_band';
    let current_type_pressed_element_id = current_type_pressed.toString() + '_band';

    stopSelected(current_type_pressed_element_id);
    addSelected(new_type_pressed_element_id);

    current_type_pressed = type;

}
```

Annotation

```
import cv2

from detection.Image import Image


# Class initiates with an image and inherits from Image. Is responsible for any annotations on the image.
class Annotation(Image):
    def __init__(self, image):
        super().__init__(image)

    # Draws a circle on a certain radius at a certain point on an image.
    def draw_circle(self, x, y, radius, thickness=2):
        self.image = cv2.circle(self.image, (x, y), radius=radius, color=(255, 255, 255), thickness=thickness)

        return self

    # Draws a rectangle of a certain width and height at a certain point on an image.
    def draw_rectangle(self, x, y, width, height, thickness=2):
        self.image = cv2.rectangle(self.image, (x, y), (x + width, y + height), (0, 255, 0), thickness)

        return self

    # Draws the contours on an image.
    def draw_contours(self, contours=None):
        for contour in contours:
            self.image = cv2.drawContours(self.image, [contour], 0, (255, 255, 255), cv2.FILLED)

        return self

    # Draws hough lines onto an image.
    def draw_hough_lines(self, lines):
        for line in lines:
            for x1, y1, x2, y2 in line:
                cv2.line(self.image, (x1, y1), (x2, y2), (255, 0, 0), 2)

        return self
```

BandIdentifier

```
from detection.Colour import Colour
from detection.ResistorBand import ResistorBand


# Initiates with the possible bands and the full list of slice bands. Finds the resistor bands from the
initiated data.
class BandIdentifier:
    def __init__(self, possible_bands):
        self.possible_bands = possible_bands
        self.colour_weights = {
            Colour.UNKNOWN: 0,
            Colour.BLACK: 0.5,
```

```
            Colour.BROWN: 1,
            Colour.RED: 1,
            Colour.ORANGE: 0.75,
            Colour.YELLOW: 1,
            Colour.GREEN: 1,
            Colour.BLUE: 1,
            Colour.VIOLET: 1,
            Colour.GREY: 1,
            Colour.WHITE: 0.5,
            Colour.GOLD: 1,
            Colour.SILVER: 0.75,
        }

    # Finding the band colour counts for each band.
    def find_band_colour_counts(self):
        try:
            slice_band_groups = self.possible_bands.groups()

            band_colour_counts = []

            for index, _ in enumerate(slice_band_groups):
                band_colour_counts.append({})

            for index, slice_band_group in enumerate(slice_band_groups):

                colour_counts = band_colour_counts[index]

                if slice_band_group:

                    for slice_band in slice_band_group:
                        colour = slice_band.colour

                        count = colour_counts.get(colour, 0)

                        colour_counts[colour] = count + 1
                else:
                    colour_counts[Colour.UNKNOWN] = 1

            return band_colour_counts

        except Exception as error:
            raise Exception(f'Error counting number of possible bands, {error}')

    # Applying the colour weights to their corresponding band colour counts.
    def apply_colour_weights(self, band_colour_counts):
        try:
            for index, colour_counts in enumerate(band_colour_counts):

                for colour, count in colour_counts.items():

                    weight = self.colour_weights[colour]

                    if colour is Colour.GOLD or colour is Colour.SILVER:
                        if index == 0 or index == len(band_colour_counts) - 1:
                            weight *= 2

                        else:
                            weight /= 2

                    colour_counts[colour] = weight * count

            return band_colour_counts

        except Exception as error:
            raise Exception(f'Error applying colour weights, {error}')

    # Majority voting the resistor bands.
    def find_most_frequent_bands(self, band_colour_counts):
        try:
            resistor_bands = []

            for colour_counts in band_colour_counts:
                colour = max(colour_counts, key=colour_counts.get)
```

```
                resistor_bands.append(ResistorBand(colour))

            return resistor_bands

        except Exception as error:
            raise Exception(f'Error finding most frequent bands, {error}')

    # Finding the resistor bands with the most frequent bands.
    def find_resistor_bands(self):
        try:
            band_colour_counts = self.find_band_colour_counts()

            band_colour_counts = self.apply_colour_weights(band_colour_counts)

            resistor_bands = self.find_most_frequent_bands(band_colour_counts)

            return resistor_bands

        except Exception as error:
            raise Exception(f'Error identifying resistor bands: {error}')
```

BGR

```
import cv2

from detection.Image import Image


# Initiates with an image and inherits from Image. Is responsible for performing functions that work on BGR
images.
class BGR(Image):
    def __init__(self, image, type='BGR'):
        bgr = self.bgr_conversion(image, type)
        super().__init__(bgr)

    # Blurs the image based on the input height and width.
    def blur(self, width=1, height=-1):
        height = self.height() if height < 0 else height

        self.image = cv2.blur(self.image, (width, height))

        return self

    # Applies a bilateral filter.
    def bilateral_filter(self):
        self.image = cv2.bilateralFilter(self.image, 5, 100, 75)

        return self

    # Conversion between types.
    def bgr_conversion(self, image, type):
        if type == 'BGR':
            return image

        if type == 'GREYSCALE':
            return cv2.cvtColor(image, cv2.COLOR_GRAY2BGR)

        if type == 'HSV':
            return cv2.cvtColor(image, cv2.COLOR_HSV2BGR)
```

BoundingRectangle

```
import cv2


# Initiates with a contour, creates a bounding rectangles and encapsulates the attributes.
class BoundingRectangle:
```

```
    def __init__(self, contour):
        self.x, self.y, self.width, self.height = cv2.boundingRect(contour)
```

Colour

```
from enum import Enum


# Enum class for colours.
class Colour(Enum):
    UNKNOWN = -1,
    BLACK = 0,
    BROWN = 1,
    RED = 2,
    ORANGE = 3,
    YELLOW = 4,
    BLUE = 5,
    GREEN = 6,
    VIOLET = 7,
    GREY = 8,
    WHITE = 9,
    GOLD = 10,
    SILVER = 11

    def __str__(self):
        return self.name
```

Contours

```
import cv2

from detection.MergeSort import MergeSort


# Initiates with contours and is responsible for carrying out operations on contours.
class Contours:
    def __init__(self, contours):
        self.contours = contours

    # Finds the biggest contour from a list of contours.
    def find_biggest(self):
        try:
            sorted_contours = self.sort()

            biggest_contour = sorted_contours[len(self.contours) - 1]

            return biggest_contour

        except Exception as error:
            raise Exception(f'Error finding biggest contour, {error}')

    # Sorts contours based on their areas.
    def sort(self):
        try:
            contour_areas = []
            area_for_contour = {}

            for contour in self.contours:
                contour area = cv2.contourArea(contour)
                contour_areas.append(contour_area)
                area_for_contour[contour_area] = contour

            contour_areas = [cv2.contourArea(contour) for contour in self.contours]

            sorted_contour_areas = MergeSort().sort(contour_areas)

            sorted_contours = []
```

```
            for sorted_contour_area in sorted_contour_areas:
                sorted_contours.append(area_for_contour[sorted_contour_area])

            return sorted_contours

        except Exception as error:
            raise Exception(f'Error sorting contours, {error}')
```

Detector

```
from detection.BandIdentifier import BandIdentifier
from detection.Image import Image
from detection.Resistor import Resistor
from detection.ResistorLocator import ResistorLocator
from detection.SliceBandsFilter import SliceBandsFilter
from detection.SliceBandsFinder import SliceBandsFinder


# The main class of the program, responsible for managing the operation of the other classes.
class Detector:
    def __init__(self):
        self.image = None
        self.resistor = None

    # Resizes the image to a more processable size.
    def resize_image(self):
        if self.image.width() > 1280:
            ratio = self.image.width() / self.image.height()

            scale = 1280 / ratio

            self.image.resize(1280, round(scale))

    # Detects the resistor bands from an image.
    def detect(self, location, maximum_band_count):
        try:
            self.image = Image().load(location)

            self.resize_image()

            resistor_image = ResistorLocator(self.image).locate()

            slice_bands = SliceBandsFinder(resistor_image.clone()).find_all_bands()

            slice_band_filter = SliceBandsFilter(slice_bands)

            possible_bands = slice_band_filter.find_possible_bands(maximum_band_count)

            if 3 <= len(possible_bands.groups()) <= 6:

                resistor_bands = BandIdentifier(possible_bands).find_resistor_bands()

                self.resistor = Resistor(resistor_bands).main()

                return self.resistor, resistor_image

            else:
                raise Exception('Not enough bands were located')

        except Exception as error:
            print(error)
            raise Exception(f'Error detecting resistor values, {error}')
```

GlareRemover

```
import cv2
import numpy as np
from sklearn.cluster import KMeans

from detection.BGR import BGR
from detection.BoundingRectangle import BoundingRectangle
from detection.Contours import Contours
from detection.Greyscale import Greyscale
from detection.HSV import HSV
from detection.HSVRange import HSVRange


# Initiates with an image and is responsible for removing the glare within an image.
class GlareRemover:
    def __init__(self, image):
        self.image = image

    # Shows the colour clusters (for development only).
    def show_colour_clusters(self, colours_information):
        try:
            rectangle = np.zeros((50, 300, 3), dtype=np.uint8)

            start = 0

            for colour in colours_information:
                # print(colour, f'{round((percent * 100), 5)} %')
                end = start + (colour[0] * 300)

                cv2.rectangle(rectangle, (int(start), 0), (int(end), 50), colour[1].astype('uint8').tolist(),
-1)

                start = end

            rectangle_image = HSV(rectangle)
            #rectangle_image.show()

        except Exception as error:
            print(f'show_colour_clusters(), {error}')

    # Identifies the glare clusters based on V values.
    def identify_glare_clusters(self, clusters):
        try:
            hsv_colours = []

            for colour in clusters.cluster_centers_:
                hsv_colours.append(colour)

            v_values = [hsv_colour[2] for hsv_colour in hsv_colours]

            if v_values:
                mean_v_value = np.mean(v_values)
            else:
                mean_v_value = v_values[0]

            glare_clusters = []

            for v_value in v_values:
                if v_value > mean_v_value:
                    glare_index = v_values.index(v_value)
                    glare_clusters.append(glare_index)

            return glare_clusters

        except Exception as error:
            raise Exception(f'Error identifying glare clusters, {error}')

    # Change the pixels to 0 if its closest centroid identified as a glare clusters.
    def remove_clusters(self, clusters, glare_clusters):
        try:
```

```
            cluster_labels = clusters.labels_

            cluster_labels = cluster_labels.reshape(self.image.height(), self.image.width())

            for row in range(cluster_labels.shape[0]):
                for column in range(cluster_labels.shape[1]):
                    if cluster_labels[row][column] in glare_clusters:
                        self.image.image[row][column] = 0

            #self.image.show()

            return self

        except Exception as error:
            raise Exception(f'Error removing clusters from image, {error}')

    # Find the colour clusters.
    def find_colour_clusters(self):
        try:
            image_data = self.image.image.reshape(self.image.height() * self.image.width(), 3)

            # Find and display most dominant colors
            clusters = KMeans(n_clusters=5).fit(image_data)

            centroids = clusters.cluster_centers_

            labels = np.arange(0, len(np.unique(clusters.labels_)) + 1)

            hist, _ = np.histogram(clusters.labels_, bins=labels)
            hist = hist.astype('float')
            hist /= hist.sum()

            colours = sorted([(percent, colour) for percent, colour in zip(hist, centroids)])

            self.show_colour_clusters(colours)

            return clusters

        except Exception as error:
            raise Exception(f'Error finding colour clusters, {error}')

    # Masks the image for non black values.
    def mask(self):
        try:
            hsv_image = HSV(self.image.image, 'BGR')

            colour_mask = hsv_image.mask(HSVRange([0, 255], [0, 255], [1, 255]))

            greyscale_mask_image = Greyscale(colour_mask, 'HSV')

            return greyscale_mask_image

        except Exception as error:
            raise Exception(f'Error masking image, {error}')

    # Removes the glare from the image by taking the biggest region.
    def remove_glare_from_image(self, glare_mask, image):
        try:
            greyscale_mask_image  = image.mask(glare_mask.image)

            contours, _ = Greyscale(greyscale_mask_image .image, 'BGR').find_contours()

            largest_contour = Contours(contours).find_biggest()

            bounding_rectangle = BoundingRectangle(largest_contour)

            no_glare_image = image.region(bounding_rectangle.x, bounding_rectangle.y,
bounding_rectangle.width,
                                          bounding_rectangle.height)

            return no_glare_image

        except Exception as error:
            raise Exception(f'Error removing glare from image, {error}')
```

```
    # Runs the functions within the glare remover.
    def main(self):
        try:
            original_image = self.image.clone()

            self.image = BGR(self.image.image).blur(round(self.image.width() * 10), 1)

            self.image = HSV(self.image.image)

            clusters = self.find_colour_clusters()

            glare_clusters = self.identify_glare_clusters(clusters)

            self.remove_clusters(clusters, glare_clusters)

            glare_mask = self.mask()

            no_glare_image = self.remove_glare_from_image(glare_mask, original_image)

            return no_glare_image

        except Exception as error:
            raise Exception(f'Error trying to remove glare from image, {error}')
```

Greyscale

```
import cv2

from detection.Image import Image

'''
Initiates with an image and current image type (for conversion) and inherits from Image. Responsible for
carrying out
operations that work on Greyscale images.
'''


class Greyscale(Image):
    def __init__(self, image=None, type='GREYSCALE'):
        greyscale = self.greyscale_conversion(image, type)
        super().__init__(greyscale)

    # Finds contours of an image.
    def find_contours(self):
        try:
            contours, hierarchy = cv2.findContours(self.image, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)

            return contours, hierarchy

        except:
            raise Exception("Could not find contours")

    # Returns the number of non zero pixels.
    def count_non_zero_pixels(self):
        try:
            return cv2.countNonZero(self.image)

        except:
            raise Exception("Error counting non zero pixels")

    # Converts an image to monochrome.
    def monochrome(self, inverted=False, block_size=51, C=21):
        try:
            thresholding = cv2.THRESH_BINARY if not inverted else cv2.THRESH_BINARY_INV

            self.image = cv2.adaptiveThreshold(self.image, 255, cv2.ADAPTIVE_THRESH_MEAN_C, thresholding,
block_size, C)

            return self
```

```
        except:
            raise Exception("Error converting image to monochrome")

    # Conversion between types.
    def greyscale_conversion(self, image, type):
        try:
            if type == 'BGR':
                return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

            if type == 'HSV':
                return image

            if type == 'HSV':
                return cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

            if type == 'GREYSCALE':
                return image

        except Exception as error:
            raise Exception(f"Error converting image from greyscale {error}")
```

HSV

```
import cv2
import numpy as np

from detection.Image import Image

'''
Initiates with an image and current image type (for conversion) and inherits from Image. Responsible for
carrying out
operations that work on HSV images.
'''


class HSV(Image):
    def __init__(self, image, type='HSV'):
        hsv = self.HSV_conversion(image, type)
        super().__init__(hsv)

    # Masking HSV ranges from an image.
    def mask(self, hsv_ranges):
        try:
            h, s, v = self.split()

            if hsv_ranges.h_range[0] > hsv_ranges.h_range[1]:
                h_range_1 = cv2.inRange(h, hsv_ranges.h_range[0], 180)
                h_range_2 = cv2.inRange(h, 0, hsv_ranges.h_range[1])

                h_range = cv2.bitwise_or(h_range_1, h_range_2)

            else:
                h_range = cv2.inRange(h, hsv_ranges.h_range[0], hsv_ranges.h_range[1])

            s_range = cv2.inRange(s, hsv_ranges.s_range[0], hsv_ranges.s_range[1])
            v_range = cv2.inRange(v, hsv_ranges.v_range[0], hsv_ranges.v_range[1])

            # Narrowing down the image to only show the HSV values of the desired range
            hs_mask = np.bitwise_and(h_range, s_range)
            hsv_mask = np.bitwise_and(hs_mask, v_range)

            return hsv_mask

        except:
            print(f'Error masking HSV ranges {hsv_ranges}')

    # Splitting and HSV image into its h, s and v components.
    def split(self):
        try:
            h, s, v = cv2.split(self.image)
```

```
            return h, s, v

        except Exception as error:
            raise Exception(f'Error splitting image into h, s, v components, {error}')

    # Conversion between types.
    def HSV_conversion(self, image, type):
        try:
            if type == 'BGR':
                return cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

            if type == 'HSV':
                return image

        except Exception as error:
            raise Exception(f'Error converting image from greyscale, {error}')
```

HSVRange

```
# An encapsulation class that initiates with the h,s and v ranges.
class HSVRange:
    def __init__(self, h_range, s_range, v_range):
        self.h_range = h_range
        self.s_range = s_range
        self.v_range = v_range
```

HSVRanges

```
from detection.Colour import Colour
from detection.HSVRange import HSVRange


# Class that contains all the hsv ranges for every resistor colour.
class HSVRanges:
    def __init__(self):
        self.hsv_ranges = {
            Colour.UNKNOWN: HSVRange([255, 255], [255, 255], [255, 255]),
            Colour.BLACK: HSVRange([0, 255], [0, 255], [0, 30]),
            Colour.BROWN: HSVRange([170, 10], [100, 190], [20, 75]),
            Colour.RED: HSVRange([175, 5], [100, 255], [50, 150]),
            Colour.ORANGE: HSVRange([5, 15], [100, 255], [70, 130]),
            Colour.YELLOW: HSVRange([20, 30], [100, 255], [80, 130]),
            Colour.GREEN: HSVRange([40, 75], [100, 255], [40, 90]),
            Colour.BLUE: HSVRange([100, 115], [100, 255], [40, 80]),
            Colour.VIOLET: HSVRange([135, 170], [70, 255], [35, 85]),
            Colour.GREY: HSVRange([0, 255], [0, 20], [50, 75]),
            Colour.WHITE: HSVRange([0, 20], [10, 40], [110, 255]),
            Colour.GOLD: HSVRange([10, 25], [70, 150], [30, 70]),
            Colour.SILVER: HSVRange([0, 1], [0, 1], [80, 130]),
        }

    def for_colour(self, colour):
        return self.hsv_ranges[colour]
```

Image

```
import base64
import os
import uuid

import cv2

colour_list = []
```

```
# Initiates with an image. Responsible for carrying out the generic image operations that work on all image
types.
class Image:
    def __init__(self, image=None):
        self.image = image

    # Returns a copy of the image.
    def clone(self):
        return Image(self.image)

    # Loads an image at a specified location.
    def load(self, location):
        try:
            self.image = cv2.imread(location)

            return self

        except:
            raise Exception("Error loading image")

    # Returns the width of the image.
    def width(self):
        try:
            return self.image.shape[1]

        except:
            print("Error getting width for image")

    # Returns the height of the image.
    def height(self):
        try:
            return self.image.shape[0]

        except:
            print("Error getting height for image")

    # Resizes the image according to the specified width and height.
    def resize(self, width, height):
        try:
            self.image = cv2.resize(self.image, (width, height))

            return self

        except:
            print("Error resizing image")

    # Returns a region of an image based on an offset and a size.
    def region(self, x, y, width, height):
        try:
            self.image = self.image[y:y + height, x:x + width]

            return self

        except Exception as error:
            print(f"Error getting region of image {error}")

    # Shows an image and returns colour values for clicked pixels.
    def show(self):
        try:
            cv2.destroyWindow('Image')

        except:
            print('Image window not open')

        cv2.imshow('Image', self.image)
        cv2.setMouseCallback('Image', self.mouse)

        return cv2.waitKey(0) & 0xff

    # Handles mouse events.
    def mouse(self, event, x, y, flags, parameters):
        try:
            if event == cv2.EVENT_LBUTTONDOWN:
```

```
                hsv_image = cv2.cvtColor(self.image, cv2.COLOR_BGR2HSV)

                colour = hsv_image[y][x]

                colour_list.append(list(colour))
                print(colour_list)

        except:
            print('Clicked outside image')

    # Saves an image with a random name.
    def save(self):
        try:
            filename = str(uuid.uuid1().int)

            absolute_file_path = f'{os.path.abspath(os.curdir)}\\resistorImages\\{filename}.jpg'

            if not cv2.imwrite(f'{absolute_file_path}', self.image):
                raise Exception('Could not write image.')

            return absolute_file_path

        except:
            raise Exception("Error saving image")

    # Warps the perspective of the image based on a matrix, width and height.
    def warp_perspective(self, matrix, width, height):
        try:
            self.image = cv2.warpPerspective(self.image, matrix, (width, height))

            return self

        except:
            print("Error trying to perform warp perspective on image")

    # Erodes the image for a certain amount of iterations.
    def erode(self, iterations):
        try:
            element = cv2.getStructuringElement(cv2.MORPH_ERODE, (3, 3), (1, 1))
            self.image = cv2.erode(self.image, element, iterations=iterations)

            return self

        except:
            raise Exception(f"Error eroding image with {iterations} iterations")

    # Returns the slices of an image.
    def create_slices(self, slice_height):
        try:
            height = self.height()

            slice_amount = height // slice_height

            image_slices = []

            for slice_number in range(slice_amount):
                x = 0
                y = slice_number * slice_height

                image_slice = self.clone().region(x, y, self.width(), slice_height)

                image_slices.append(image_slice)

            return image_slices

        except:
            raise Exception("Error slicing image")

    # Masks an image based on an input mask.
    def mask(self, mask):
        try:
            self.image = cv2.bitwise_and(self.image, self.image, mask=mask)

            return self
```

```
        except:
            print("Error masking image with mask")

    # Rotates the image 90 degrees clockwise.
    def rotate_90_clockwise(self):
        try:
            self.image = cv2.rotate(self.image, cv2.ROTATE_90_CLOCKWISE)

            return self

        except:
            print("Error rotating image")

    # Returns the byte-stream for an image.
    def byte_stream(self):
        try:
            encoded_image, buffer = cv2.imencode('.jpg', self.image)
            byte_stream_image = base64.b64encode(buffer)

            return byte_stream_image

        except:
            print("Error converting image to byte stream")
```

KMeans

```
import random as rd

import numpy as np


# Initiates with the number of centroids and contains the various parts of the K-Means algorithm.
class KMeans:
    def __init__(self, number_of_centroids=4):
        self.number_of_centroids = number_of_centroids
        self.centroids = {}
        self.labels = []

    # Finds the distance between 2 points.
    def find_distance(self, point_1, point_2):
        try:
            dx_squared = (point_1[0] - point_2[0]) ** 2
            dy_squared = (point_1[1] - point_2[1]) ** 2
            dz_squared = (point_1[2] - point_2[2]) ** 2

            distance = np.sqrt(dx_squared + dy_squared + dz_squared)

            return distance

        except Exception as error:
            print(f'find_distance, {error}')

    # Finds the distance between items and centroids.
    def find_distance_to_centroids(self, data):
        item_to_centroids_distances = {}

        for centroid_number in range(1, self.number_of_centroids + 1):
            item_to_centroids_distances[centroid_number] = []

        for centroid_number in range(1, self.number_of_centroids + 1):
            for index in range(len(data)):
                item_to_centroid_distance = self.find_distance(data[index], self.centroids[centroid_number])

                item_to_centroids_distances[centroid_number].append(item_to_centroid_distance)

        return item_to_centroids_distances

    # Finds the minimum inter-clusters distance.
    def find_min_inter_cluster_distance(self):
        minimum_inter_cluster_distance = np.inf
```

```
        for _, centroid_1 in self.centroids.items():
            for _, centroid_2 in self.centroids.items():

                if centroid_1 != centroid_2 or len(self.centroids) == 1:
                    inter_cluster_distance = self.find_distance(centroid_1, centroid_2)

                    minimum_inter_cluster_distance = min(minimum_inter_cluster_distance,
inter_cluster_distance)

        return minimum_inter_cluster_distance

    # Finds the maximum intra-clusters distance.
    def find_max_intra_cluster_distance(self, data_grouped_by_label):
        maximum_intra_cluster_distance = -np.inf

        for centroid_number, data in data_grouped_by_label.items():
            for item in data:
                intra_cluster_distance = self.find_distance(item, self.centroids[centroid_number])

                maximum_intra_cluster_distance = max(maximum_intra_cluster_distance, intra_cluster_distance)

        if maximum_intra_cluster_distance == 0:
            return 1
        else:
            return maximum_intra_cluster_distance

    # Finds the optimal number of clusters using Dunn-index.
    def find_optimal_number_of_clusters(self, data):
        dunn_indexes = []

        for centroid_amount in range(3, 7):
            self.centroids = {}
            self.number_of_centroids = centroid_amount
            self.fit(data, self.initialize_centroids(data))

            data_grouped_by_label = self.group_data_by_label(data)

            maximum_intra_cluster_distance = self.find_max_intra_cluster_distance(data_grouped_by_label)

            minimum_inter_cluster_distance = self.find_min_inter_cluster_distance()

            dunn_index = minimum_inter_cluster_distance / maximum_intra_cluster_distance

            dunn_indexes.append(dunn_index)

        optimal_number_of_clusters = np.argmax(dunn_indexes) + 3

        return optimal_number_of_clusters

    def find_next_centroid(self, data, minimum_centroid_distances):
        minimum_centroid_distances = np.array(minimum_centroid_distances)

        index_of_highest_distance = np.argmax(minimum_centroid_distances)

        next_centroid = data[index_of_highest_distance]

        return next_centroid

    # Initialises the seeds to be used for fit().
    def initialize_centroids(self, data):
        number_of_items = data.shape[0]

        centroids = {1: (data[rd.randint(0, number_of_items - 1)])}

        for centroid_number in range(1, self.number_of_centroids):

            minimum_centroid_distances = []

            for item in data:
                minimum_item_centroid_distance = np.inf

                # Find the distance.
                for _, centroid in centroids.items():
```

```
                    item_centroid_distance = self.find_distance(item, centroid)
                    minimum_item_centroid_distance = min(minimum_item_centroid_distance,
item_centroid_distance)

                minimum_centroid_distances.append(minimum_item_centroid_distance)

            # Item that produces the largest distance is the centroid.
            next_centroid = self.find_next_centroid(data, minimum_centroid_distances)

            centroids[centroid_number + 1] = next_centroid

        return centroids

    # Labelling items with the number of their closest centroid.
    def find_labels(self, data, intra_cluster_distances_for_centroids):
        self.labels = []

        for item_index in range(len(data)):
            distances_from_centroids = []

            for _, intra_cluster_distances in intra_cluster_distances_for_centroids.items():
                distances_from_centroids.append(intra_cluster_distances[item_index])

            self.labels.append(np.argmin(distances_from_centroids) + 1)

    # Grouping the data by the labels.
    def group_data_by_label(self, data):
        data_grouped_by_label = {}

        for centroid_number in range(1, self.number_of_centroids + 1):
            data_grouped_by_label[centroid_number] = []

        for centroid_number in range(1, self.number_of_centroids + 1):
            for index, label in enumerate(self.labels):
                if label == centroid_number:
                    data_grouped_by_label[centroid_number].append(data[index])

        return data_grouped_by_label

    # Moving centroids by calculating the mean of the data grouped by centroid.
    def move_centroids(self, data_grouped_by_label):
        for centroid_number in range(1, len(data_grouped_by_label) + 1):
            x_total = 0
            y_total = 0
            z_total = 0

            for item in data_grouped_by_label[centroid_number]:
                x_total += item[0]
                y_total += item[1]
                z_total += item[2]

            number_of_items = len(data_grouped_by_label[centroid_number])

            x_mean = x_total / number_of_items
            y_mean = y_total / number_of_items
            z_mean = z_total / number_of_items

            self.centroids[centroid_number] = [x_mean, y_mean, z_mean]

    # Finding the intra-clusters distances for centroids.
    def find_intra_cluster_distances_for_centroids(self, data, centroids):
        intra_cluster_distances_for_centroids = {}

        for centroid_number, centroid in centroids.items():

            intra_cluster_distances = []

            for item in data:
                intra_cluster_distance = self.find_distance(centroid, item)
                intra_cluster_distances.append(intra_cluster_distance)

            intra_cluster_distances_for_centroids[centroid_number] = intra_cluster_distances

        return intra_cluster_distances_for_centroids
```

```
    # Checking the centroids have moved by comparing the moved centroids to the initially input centroids.
    def check_if_centroids_moved(self, centroids):
        inter_cluster_distances = []

        for centroid_number in range(1, self.number_of_centroids + 1):
            inter_cluster_distance = np.subtract(self.centroids[centroid_number], centroids[centroid_number])

            inter_cluster_distance_sum = np.sum(inter_cluster_distance)

            inter_cluster_distances.append(abs(inter_cluster_distance_sum))

        inter_cluster_distances_sum = sum(inter_cluster_distances)

        if inter_cluster_distances_sum != 0:
            return True

        else:
            return False

    # K-means algorithm to find the labels and centroids based on the centroids.
    def fit(self, data, centroids, iteration=0, max_iterations=15):
        try:
            self.centroids = {}
            self.labels = []

            intra_cluster_distances_for_centroids = self.find_intra_cluster_distances_for_centroids(data,
centroids)

            self.find_labels(data, intra_cluster_distances_for_centroids)

            data_grouped_by_label = self.group_data_by_label(data)

            self.move_centroids(data_grouped_by_label)
            iteration += 1

            centroids_moved = self.check_if_centroids_moved(centroids)

            if iteration == 1 or centroids_moved:
                if iteration <= max_iterations:
                    return self.fit(data, self.centroids, iteration, max_iterations)
                else:
                    return self

            else:
                return self

        except Exception as error:
            print(f'fit(), {error}')
```

Line

```
import numpy as np

'''
Class responsible for line operations. There are multiple ways to initialise lines and multiple operations
that can be
performed.
'''


class Line:
    def __init__(self):
        self.gradient = None
        self.constant = None

    # Make a line from 2 points.
    def from_points(self, point_1, point_2):
        try:
            line = Line()
```

```
            line.gradient = self.find_gradient(point_1, point_2)
            line.constant = point_1[1] - line.gradient * point_1[0]

            return line

        except:
            raise Exception(f"Could not create line from points: {point_1} and {point_2}")

    # Make a line from a point and a gradient.
    def from_gradient(self, gradient, point):
        try:
            line = Line()

            line.gradient = gradient
            line.constant = point[1] - line.gradient * point[0]

            return line

        except:
            raise Exception(f"Could not create line from gradient: {gradient} and point: {point}")

    # Find the gradient of a line from 2 points.
    def find_gradient(self, point_1, point_2):
        try:
            dx = (point_2[0] - point_1[0])
            dy = (point_2[1] - point_1[1])

            gradient = dy / dx

            return gradient

        except ZeroDivisionError:
            print('Line is vertical.')
            return None

    # Find the constant value of the line.
    def find_constant(self, point):
        if self.gradient is not None:
            return point[1] - self.gradient * point[0]

        else:
            return None

    # Solve the line for the x value from a y value.
    def solve_for_x(self, y):
        if self.gradient is not None and self.constant is not None:
            return (y - self.constant) / self.gradient

        else:
            print('Can not solve on a vertical or horizontal line.')

    # Solve the line for the y value from an x value.
    def solve_for_y(self, x):
        if self.gradient is not None and self.constant is not None:
            return float(self.gradient) * x + float(self.constant)

        else:
            print('Can not solve on a vertical or horizontal line.')

    # Returns the normal gradient.
    def normal(self):
        try:
            return -1 / self.gradient

        except:
            print('Line has no gradient.')

    # Finds the intersection of 2 lines.
    def find_intersection(self, line_2):
        try:
            x = (line_2.constant - self.constant) / (self.gradient - line_2.gradient)
            y = self.solve_for_y(x)

            intersection = [x, y]
```

```
            return intersection

        except:
            raise Exception('Error finding intersection.')

    # Finds the length of 2 lines between 2 points.
    def length(self, point_1, point_2):
        try:
            dx_squared = (point_1[0] - point_2[0]) ** 2
            dy_squared = (point_1[1] - point_2[1]) ** 2

            distance = np.sqrt(dx_squared + dy_squared)

            return distance

        except:
            raise Exception('Error finding line length.')

    # Finds the knee of data.
    def find_knee(self, points):
        try:
            point_1 = points[0]
            point_2 = points[len(points) - 1]

            spine = self.from_points(point_1, point_2)

            intersection_distances = []

            for current_point in points:
                current_line = self.from_gradient(spine.normal(), current_point)

                intersection = current_line.find_intersection(spine)

                distance = current_line.length(intersection, current_point)

                intersection_distances.append(distance)

            knee = intersection_distances.index(max(intersection_distances))

            return knee

        except Exception as error:
            print(f'Error finding knee: {error}')
```

MergeSort

```
# The merge sort algorithm.
class MergeSort:
    def __init__(self):
        pass

    # Responsible for sorting and merging the lists
    def sort_and_merge_lists(self, left, right, reverse):

        left_index = 0
        right_index = 0

        sorted_and_merged_list = []

        while left_index < len(left) and right_index < len(right):

            if reverse:
                if left[left_index] > right[right_index]:
                    sorted_and_merged_list.append(left[left_index])
                    left_index += 1

                else:
                    sorted_and_merged_list.append(right[right_index])
                    right_index += 1
```

```
            else:
                if left[left_index] < right[right_index]:
                    sorted_and_merged_list.append(left[left_index])
                    left_index += 1

                else:
                    sorted_and_merged_list.append(right[right_index])
                    right_index += 1

        sorted_and_merged_list.extend(left[left_index:])
        sorted_and_merged_list.extend(right[right_index:])

        return sorted_and_merged_list

    # The entry point for the sort
    def sort(self, data, reverse=False):

        if len(data) <= 1:
            return data

        middle_index = len(data) // 2

        left = data[:middle_index]
        right = data[middle_index:]

        left = self.sort(left, reverse)
        right = self.sort(right, reverse)

        sorted_and_merged_list = self.sort_and_merge_lists(left, right, reverse)

        return sorted_and_merged_list
```

Resistor

```
import json
import os


# The resistor class initiates with resistor bands and contains various operations that work on the resistor
bands.
class Resistor:
    def __init__(self, bands=None):
        self.bands = bands
        self.valid = False

    # Returns the resistor type.
    def get_number_of_bands(self):

        if self.bands:

            number_of_bands = 0

            for band in self.bands:
                if band is not None:
                    number_of_bands += 1

            return number_of_bands

        else:
            return 6

    # Returns the formatted bands for the resistor.
    def colours(self):
        type = self.get_number_of_bands()

        if type == 3:
            colours = [self.bands[0].colour.name, self.bands[1].colour.name, None, self.bands[2].colour.name,
None,
                       None]

        elif type == 4:
```

```
            colours = [self.bands[0].colour.name, self.bands[1].colour.name, None, self.bands[2].colour.name,
                       self.bands[3].colour.name, None]

        elif type == 5:
            colours = [self.bands[0].colour.name, self.bands[1].colour.name, self.bands[2].colour.name,
                       self.bands[3].colour.name,
                       self.bands[4].colour.name, None]

        elif type == 6:
            colours = [self.bands[0].colour.name, self.bands[1].colour.name, self.bands[2].colour.name,
                       self.bands[3].colour.name,
                       self.bands[4].colour.name, self.bands[5].colour.name]
        else:
            colours = [None, None, None, None, None, None]

        return colours

    # Gets the digit band bands.
    def get_digit_band_colours(self, bands):
        try:
            digit_band_colours = []

            if bands:
                for index in range(3):
                    if bands[index] is not None:
                        digit_band_colours.append(bands[index])

                return digit_band_colours

        except Exception as error:
            raise Exception(f'Error getting digit band colours, {error}')

    # Checks if the resistor digit band bands are in the standard values database.
    def check_valid(self, digit_band_colours):
        try:

            if len(digit_band_colours) >= 2:
                number_of_bands = self.get_number_of_bands()

                if number_of_bands < 5:
                    number_of_digit_bands = 2

                else:
                    number_of_digit_bands = 3

                data_file = f'standardResistorValues{number_of_digit_bands}sf.json'

                with open(f'{os.getcwd()}\\data\\standardResistorValues\\{data_file}') as
valid_band_colours_list:

                    valid_band_colours_list = json.load(valid_band_colours_list)

                    if digit_band_colours in valid_band_colours_list:
                        return True

                    else:
                        return False

            else:
                return False

        except Exception as error:
            print(f'check_valid(), {error}')

    # Finds the most probable correct version of a resistor.
    def main(self):
        try:
            self.valid = self.check_valid(self.get_digit_band_colours(self.colours()))

            if self.valid is True:
                return self

            else:
                self.bands = self.bands[::-1]
```

```
                self.valid = self.check_valid(self.get_digit_band_colours(self.colours()))

                return self

        except Exception as error:
            raise Exception(f'Error while trying to validate resistor, {error}')
```

ResistorBand

```
# Encapsulation class for the colour of a resistor band.
class ResistorBand:
    def __init__(self, colour):
        self.colour = colour
```

ResistorLocator

```
import cv2
import numpy as np

from detection.Annotation import Annotation
from detection.Contours import Contours
from detection.Greyscale import Greyscale
from detection.Line import Line


# Initiates with an image and contains various operations that aid with the location of a resistor.
class ResistorLocator:
    def __init__(self, image):
        self.image = image

    # Finding the amount of erosion needed to remove the resistor wires.
    def find_erode_iterations(self, image, contours):
        try:
            image = Greyscale(image.image)

            biggest_initial_contour = Contours(contours).find_biggest()

            biggest_initial_contour_area = cv2.contourArea(biggest_initial_contour)

            squared_contour_areas = [biggest_initial_contour_area ** 2]

            empty_image = False

            while not empty_image:

                eroded_image = image.erode(1)

                if eroded_image.count_non_zero_pixels() != 0:

                    contours, _ = eroded_image.find_contours()
                    biggest_contour = Contours(contours).find_biggest()

                    contour_area = cv2.contourArea(biggest_contour)
                    squared_contour_areas.append(contour_area ** 2)

                else:
                    empty_image = True

            points = []

            for x_index in range(len(squared_contour_areas)):
                points.append([x_index, squared_contour_areas[x_index]])

            knee = Line().find_knee(points)

            safe_knee = knee + 5

            return safe_knee
```

```
        except Exception as error:
            raise Exception(f'Error trying to find erode iterations, {error}')

    # Finding the contour of the resistor body.
    def find_resistor_contour(self):
        try:
            greyscale_image = Greyscale(self.image.image, 'BGR')

            monochrome_image = greyscale_image.monochrome(inverted=True, block_size=151, C=21)

            contours, _ = monochrome_image.find_contours()

            filled_image = Annotation(monochrome_image.image).draw_contours(contours)

            filled_image = Greyscale(filled_image.image)

            erode_iterations = self.find_erode_iterations(filled_image.clone(), contours)

            eroded_image = filled_image.erode(erode_iterations)

            contours, _ = eroded_image.find_contours()

            resistor_body_contour = Contours(contours).find_biggest()

            return resistor_body_contour

        except Exception as error:
            raise Exception(f'Error trying to find resistor contour, {error}')

    # Extracts the resistor from a rectangle
    def extract_resistor(self, resistor_body_contour):
        try:
            resistor_rectangle = cv2.minAreaRect(resistor_body_contour)

            box = cv2.boxPoints(resistor_rectangle)
            box = np.int0(box)

            # Get width and height of the detected resistor_rectangle.
            width = int(resistor_rectangle[1][0])
            height = int(resistor_rectangle[1][1])

            source_points = box.astype('float32')

            # Coordinate of the points in box points after the resistor_rectangle has been straightened.
            destination_points = np.array([[0, height - 1],
                                           [0, 0],
                                           [width - 1, 0],
                                           [width - 1, height - 1]], dtype='float32')

            # The perspective transformation.
            matrix = cv2.getPerspectiveTransform(source_points, destination_points)

            self.image = self.image.warp_perspective(matrix, width, height)

            if self.image.width() < self.image.height():
                self.image = self.image.rotate_90_clockwise()

            return self.image

        except Exception as error:
            raise Exception(f'Error trying to extract resistor, {error}')

    # Extracts the image of the resistor body from a resistor body contour.
    def locate(self):
        try:
            resistor_body_contour = self.find_resistor_contour()

            resistor_image = self.extract_resistor(resistor_body_contour)

            return resistor_image

        except Exception as error:
            raise Exception(f'Error locating resistor in image, {error}')
```

SliceBand

```
from detection.ResistorBand import ResistorBand


# A class that encapsulates slice band details and carries out slice band oper
class SliceBand(ResistorBand):
    def __init__(self, colour, bounding_rectangle):
        super().__init__(colour)
        self.bounding_rectangle = bounding_rectangle

    # The string function for formatting the class output for development.
    def __str__(self):
        return f"{self.colour.name}, ({self.bounding_rectangle.x}, {self.bounding_rectangle.y}, " \
               f"{self.bounding_rectangle.width}, {self.bounding_rectangle.height})"

    # Returns the lower x coordinate for a slice band.
    def x_low(self):
        return self.bounding rectangle.x

    # Returns the upper x coordinate for a slice band.
    def x_high(self):
        return self.bounding_rectangle.x + self.bounding_rectangle.width

    # Returns the centre x coordinate of a slice band
    def x_centre(self):
        return self.bounding_rectangle.x + self.bounding_rectangle.width / 2

    # Checks if a slice band overlaps with another slice band.
    def has_overlap(self, other):
        if other.x_low() - 1 <= self.x_low() <= other.x_high() + 1:
            return True

        if other.x_low() - 1 <= self.x_high() <= other.x_high() + 1:
            return True

        if self.x_low() - 1 <= other.x_low() <= self.x_high() + 1:
            return True

        if self.x_low() - 1 <= other.x_high() <= self.x_high() + 1:
            return True

        return False

    # Checks the size of the overlap between 2 slice bands
    def overlaps_size(self, other):
        low = max(self.x_low(), other.x_low())

        high = min(self.x_high(), other.x_high())

        result = high - low

        if result < 0:
            return 0

        return result

    # Returns the slice band with a bigger width out of 2 slice bands
    def biggest_width(self, other):
        return max(self.bounding_rectangle.width, other.bounding_rectangle.width)

    # Calculates the overlap amount between 2 slice bands
    def overlap_amount(self, other):
        overlap = self.overlaps_size(other)

        width = self.biggest_width(other)

        return overlap / width

    # Count the number of overlaps with other slice bands.
    def count_overlaps(self, slice_bands):
```

```
        result = 0

        for slice_band in slice_bands:
            if self.overlap_amount(slice_band) > 0:
                result = result + 1

        return result

    # Checks if a slice band has any overlaps with other slice bands.
    def has_any_overlaps(self, slice_bands):
        for slice_band in slice_bands:

            if self is slice_band:
                continue

            if self.has_overlap(slice_band):
                return True

        return False
```

SliceBands

```
import numpy as np


# Class for slice bands operations
class SliceBands:
    def __init__(self):
        self.slice_bands = []
        self.slice_band_groups = []

    # Outputting the slice bands nicely when printed.
    def __str__(self):
        text = ''

        for index, overlap_group in enumerate(self.groups()):
            text += f'Group {index}\n'

            for slice_band in overlap_group:
                text += '    ' + str(slice_band) + '\n'

        return text

    # Loading the slice band groups from a dictionary of slice band groups.
    def load_slice_band_groups(self, slice_band_dictionary):
        try:
            for slice_band_group in slice_band_dictionary.values():
                self.slice_band_groups.append(slice_band_group)

            return self

        except Exception as error:
            print(f'load_slice_band_groups(), {error}')

    # Returns the slice bands from the self values
    def list(self):
        return self.slice_bands

    # Returns the slice band groups from the self values.
    def groups(self):
        return self.slice_band_groups

    # Returns the amount of  slice band groups
    def get_band_group_count(self):
        return len(self.groups())

    # Adds a slice band to the slice bands and re-sorts the slice bands depending on their x centres.
    def add_band(self, slice_band):
        try:
            self.slice_bands.append(slice_band)
            self.slice_bands = sorted(self.slice_bands, key=lambda band: band.x_centre())
```

```
            self.slice_band_groups = self.insert_into_slice_band_group(self.slice_band_groups, slice_band)

            return self

        except Exception as error:
            print(f'add_band(), {error}')

    # Adds multiple slice bands to the slice bands.
    def add_bands(self, slice_bands):
        for slice_band in slice_bands:
            self.add_band(slice_band)

        return self

    # Removes a band from the slice bands.
    def remove_band(self, slice_band):
        try:
            self.slice_bands.remove(slice_band)

            for slice_band_group in self.slice_band_groups:

                if slice_band in slice_band_group:
                    slice_band_group.remove(slice_band)

            return self

        except Exception as error:
            print(f'remove_band(), {error}')

    # Returns the x centres for all the slice bands
    def get_x_centres(self):
        return [slice_band.x_centre() for slice_band in self.slice_bands]

    # Returns the widths for all the slice bands
    def get_widths(self):
        return [slice_band.bounding_rectangle.width for slice_band in self.slice_bands]

    # Returns the mean width of the slice bands
    def get_mean_width(self):
        return np.mean(self.get_widths())

    # Removes bands which are less than or equal to 2 pixels in width from the slice bands.
    def remove_narrow_bands(self):
        minimum_width = 2

        narrow_bands = []

        for slice_band in self.slice_bands:
            if slice_band.bounding_rectangle.width <= minimum_width:
                narrow_bands.append(slice_band)

        for slice_band in narrow_bands:
            self.remove_band(slice_band)

        return self

    # Returns the heights for all the slice bands
    def get_heights(self):
        return [slice_band.bounding_rectangle.height for slice_band in self.slice_bands]

    # Returns the mean height for the slice bands
    def get_mean_height(self):
        return np.mean(self.get_heights())

    # Removes bands whose heights are less than the mean height / 4 (or 1 pixel).
    def remove_short_bands(self):
        try:
            height = self.get_mean_height()

            minimum_height = max(1, height / 4)

            short_bands = []
```

```
            for slice_band in self.slice_bands:
                if slice_band.bounding_rectangle.height <= minimum_height:
                    short_bands.append(slice_band)

            for slice_band in short_bands:
                self.remove_band(slice_band)

            return self

        except Exception as error:
            print(f'remove_short_bands(), {error}')

    # Checks if input slice bands overlaps with slice bands.
    def has_any_overlaps(self, slice_bands):
        for slice_band in slice_bands:

            if self is slice_band:
                continue

            if slice_band.has_overlap(slice_band):
                return True

        return False

    # Inserts slice band into slice band groups.
    @staticmethod
    def insert_into_slice_band_group(slice_band_groups, slice_band):

        for slice_band_group in slice_band_groups:

            if slice_band.has_any_overlaps(slice_band_group):
                slice_band_group.append(slice_band)
                return slice_band_groups

        slice_band_group = [slice_band]
        slice_band_groups.append(slice_band_group)

        return slice_band_groups
```

SliceBandsFilter

```
import numpy as np

from detection.KMeans import KMeans
from detection.MergeSort import MergeSort
from detection.SliceBands import SliceBands


# Initiates with slice bands and carries out operations to select only the most likely slice bands.
class SliceBandsFilter:
    def __init__(self, slice_bands=None):
        self.slice_bands = slice_bands

    # Using K-means to find where there are clusters of X (locating the band).
    def find_x_cluster(self, x_centres, number_of_bands):
        try:
            x_y_z_list = np.array([[x, 0, 0] for x in x_centres])

            k_means = KMeans()

            if number_of_bands is None:
                number_of_bands = k_means.find_optimal_number_of_clusters(x_y_z_list)

            k_means.number_of_centroids = number_of_bands
            seeds = k_means.initialize_centroids(x_y_z_list)
            clusters = k_means.fit(x_y_z_list, seeds)

            return clusters

        except Exception as error:
            raise Exception(f'Error while trying to find x cluster, {error}')
```

```
    # Finds the mean difference for a list.
    def find_mean_difference(self, list):
        try:
            differences = np.diff(list)
            mean_difference = np.mean(differences)

            return mean_difference

        except Exception as error:
            raise Exception(f'Error while trying to find mean difference, {error}')

    # Identifying the slice bands that match up with the x centroids.
    def identify_possible_bands(self, sorted_centroids, deviation):
        try:
            possible_bands = {}

            for possible_band in range(len(sorted_centroids)):
                possible_bands[possible_band] = []

            mean_centroid_difference = self.find_mean_difference(sorted_centroids)

            for band_number, centroid in enumerate(sorted_centroids):
                for slice_band in self.slice_bands.list():

                    centroid_band_difference = abs(slice_band.x_centre() - centroid)

                    if centroid_band_difference < mean_centroid_difference * deviation:
                        possible_bands[band_number].append(slice_band)

            return possible_bands

        except Exception as error:
            print(f'identify_possible_bands(), {error}')

    # Removing centroids that are close to each other.
    def remove_false_centroids(self, clusters):
        try:
            centroids = [centroid[0] for centroid in clusters.centroids.values()]

            sorted_centroids = MergeSort().sort(centroids)

            mean_difference = self.find_mean_difference(sorted_centroids)

            previous_centroid = None
            false_centroids = []

            for centroid in sorted_centroids[:]:
                if previous_centroid:
                    difference = centroid - previous_centroid

                    if difference < (mean_difference * 0.2):
                        false_centroids.append(centroid)

                previous_centroid = centroid

            for centroid in false_centroids:
                sorted_centroids.remove(centroid)

            return sorted_centroids

        except Exception as error:
            print(f'remove_false_centroids(), {error}')

    # Keep trying to find binds with a certain deviation until a sufficient amount has been found.
    def get_possible_bands(self, sorted_centroids):
        try:
            possible_bands = self.identify_possible_bands(sorted_centroids, 0.2)

            if len(min(possible_bands.values(), key=len)) <= 5:

                deviation = 0.21

                while len(min(possible_bands.values(), key=len)) <= 3 and deviation <= 0.3:
```

```
                    possible_bands = self.identify_possible_bands(sorted_centroids, deviation)

                    deviation += 0.01

            return possible_bands

        except Exception as error:
            raise Exception(f'Error trying to find possible bands, {error}')

    # Remove obvious outlier bands that do not meet certain criteria.
    def remove_short_bands(self):
        try:
            self.slice_bands.remove_short_bands()

        except Exception as error:
            print(f'remove_short_bands(), {error}')

    # Remove obvious outlier bands that do not meet certain criteria.
    def remove_narrow_bands(self):
        try:
            self.slice_bands.remove_narrow_bands()

        except Exception as error:
            print(f'remove_narrow_bands(), {error}')

    # Identifying the possible bands out of the slice bands by only keeping bands that meet certain criteria.
    def find_possible_bands(self, number_of_bands):
        try:
            self.remove_short_bands()
            self.remove_narrow_bands()

            x_centres = self.slice_bands.get_x_centres()
            clusters = self.find_x_cluster(x_centres, number_of_bands)

            sorted_centroids = self.remove_false_centroids(clusters)
            possible_bands = self.get_possible_bands(sorted_centroids)

            slice_band_groups = SliceBands().load_slice_band_groups(possible_bands)

            return slice_band_groups

        except Exception as error:
            raise Exception(f'Error finding possible bands, {error}')
```

SliceBandsFinder

```
import numpy as np

from detection.BGR import BGR
from detection.BoundingRectangle import BoundingRectangle
from detection.Colour import Colour
from detection.GlareRemover import GlareRemover
from detection.Greyscale import Greyscale
from detection.HSV import HSV
from detection.HSVRanges import HSVRanges
from detection.SliceBand import SliceBand
from detection.SliceBands import SliceBands


# Initiates with an image, slices the image and is responsible for finding the slice bands of each slice.
class SliceBandsFinder:
    def __init__(self, image):
        self.image = image
        self.slice_bands = SliceBands()

    # Check if the band is right at the edge of the image.
    def check_if_edge_band(self, bounding_rectangle):
```

```
        try:
            image_width = self.image.width()

            if bounding_rectangle.x == 0 or bounding_rectangle.x == image_width:
                return True

            else:
                return False

        except Exception as error:
            raise Exception(f'Error checking edge band, bounding rectangle is None, {error}')

    # Returns a mask for the specified colour on an image slice.
    def band_mask(self, colour, image_slice):
        try:
            hsv_ranges = HSVRanges().for_colour(colour)

            hsv_image = HSV(image_slice.image, 'BGR')

            colour_mask = hsv_image.mask(hsv_ranges)

            greyscale_mask_image = Greyscale(colour_mask, 'HSV')

            return greyscale_mask_image

        except Exception as error:
            raise Exception(f'Error masking band, {error}')

    # Checks if the mask has masked the background rather than a mask.
    def check_if_background_masked(self, bounding_rectangle, image_slice):
        if bounding_rectangle.width > (image_slice.width() * 0.5):
            return True

        else:
            return False

    # Finds the bands.
    def find_bands(self, image_slice):
        try:
            for colour in Colour:

                greyscale_mask_image = self.band_mask(colour, image_slice)

                non_zero_pixels = greyscale_mask_image.count_non_zero_pixels()

                band_contours = None

                if non_zero_pixels != 0:
                    band_contours, _ = greyscale_mask_image.find_contours()

                if band_contours is not None:

                    for contour in band_contours:
                        bounding_rectangle = BoundingRectangle(contour)

                        edge_band = False

                        if colour == Colour.WHITE or colour == Colour.GREY:
                            edge_band = self.check_if_edge_band(bounding_rectangle)

                        background_masked = self.check_if_background_masked(bounding_rectangle, image_slice)

                        if not edge_band and not background_masked:
                            slice_band = SliceBand(colour, bounding_rectangle)

                            self.slice_bands.add_band(slice_band)

            return self

        except Exception as error:
            raise Exception(f'Error trying to find bands for image slice, {error}')

    # Calculating the variance of a list of values
    def calculate_variance(self, values):
```

```
        try:
            number_of_values = len(values)

            values_sum = np.sum(values)
            mean = values_sum / number_of_values

            squared_value_mean_differences = [(value - mean) ** 2 for value in values]

            sum_value_mean_differences = sum(squared_value_mean_differences)

            variance = sum_value_mean_differences / number_of_values

            return variance

        except Exception as error:
            raise Exception(f'Error to calculate variance: {error}')

    # Finds all the bands for all slices.
    def find_all_bands(self):
        try:
            self.image = self.image.resize(self.image.width(), self.image.height() * 20)

            self.image = GlareRemover(self.image).main()

            self.image = self.image.resize(self.image.width(), self.image.height() * 5)

            blurred_image = BGR(self.image.image).blur(1, round(self.image.height() * 0.5))

            image_slices = blurred_image.create_slices(round(blurred_image.height() * 0.05))

            for index in range(len(image_slices)):
                self.find_bands(image_slices[index])

            return self.slice_bands

        except Exception as error:
            raise Exception(f'Error finding slice bands, {error}')
```

References

A better and faster way to remove glare. (2021, 8 12). Retrieved from Hackaday: https://hackaday.io/project/11943-open-indirect-ophthalmoscope/log/43906-a-better-and-faster-way-to-remove-glare

A guide to the Modern Minimal UI style. (2021, 8 3). Retrieved from uxdesign: https://uxdesign.cc/a-guide-to-the-modern-minimal-ui-style-531ac1e9fbfe

A Short Guide to Writing Software Requirements. (21, 6 22). Retrieved from pjsrivastava: https://www.pjsrivastava.com/a-short-guide-to-writing-software-requirements

Android Software Development. (2021, 1 7). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/Android_software_development

Android User Interface. (2021, 7 1). Retrieved from Concept Draw: https://www.conceptdraw.com/solution-park/software-android-user-interface

Beginners' guide to wireframing UIs: benefits & best practices. (2, 8 2021). Retrieved from justinmind: https://www.justinmind.com/blog/why-wireframing-your-interface/

Brand. (2021, 8 3). Retrieved from Discord: https://discord.com/branding

Braza, J. (2021, 5 4). *What is a Resistor?* Retrieved from Circuit Basics: https://www.circuitbasics.com/what-is-a-resistor/

C++ Quick Guide. (2021, 7 1). Retrieved from sceweb: https://sceweb.sce.uhcl.edu/helm/WEBPAGE-Cpp/my_files/Quick%20Guide/cpp_overview.html#:~:text=C%2B%2B%20is%20a%20statically%20typed,and%20low%2Dlevel%20language%20features.

Cascade Classifier. (21, 8 22). Retrieved from opencv: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html#:~:text=It%20is%20a%20machine%20learning,detect%20objects%20in%20other%20images.&text=Initially%2C%20the%20algorithm%20needs%20a,faces)%20to%20train%20the%20classifier.

Cascade classifier example. (2021, 8 21).

Data Flow Diagram Symbols. (2021, 11 27).

Discord Colors. (2021, 8 3). Retrieved from Brand Palettes: https://brandpalettes.com/discord-colors/#:~:text=The%20colors%20for%20the%20Discord,black%20and%20Not%20quite%20black.

Encyclopedia of Clinical Neuropsychology. (2021, 6 30). Retrieved from Edge Detection: https://link.springer.com/referenceworkentry/10.1007%2F978-3-319-56782-2_1360-2

Flask. (2021, 8 1). Retrieved from pypi: https://pypi.org/project/Flask/

Google Play Store, Resistance Calculator. (2021, 4 22). Retrieved from Google Play Store: https://play.google.com/store/apps/details?id=com.lpellis.rcalcfree&hl=en_GB&gl=US

Google Play Store, Resistor color code calculator. (2021, 4 22). Retrieved from Google Play Store: https://play.google.com/store/apps/details?id=com.jedemm.resistorcalculator&hl=en_GB&gl=US

Google Play Store, Resistor Scanner (beta). (2021, 4 22). Retrieved from Google Play Store: https://play.google.com/store/apps/details?id=noldyLabs.resistorscanner&hl=en_GB&gl=US

Google Play Store, Resistor Scanner. (2021, 3 31). Retrieved from Google Play Store: https://play.google.com/store/apps/details?id=com.mhdev.resistorscanner&showAllReviews=true

How do I see depth? (2021, 6 30). Retrieved from scecinfo: http://scecinfo.usc.edu/geowall/stereohow.html

How the brain processes images. (2021, 6 29). Retrieved from Scientific American: https://blogs.scientificamerican.com/mind-guest-blog/how-the-brain-processes-images/

Java Summary. (2021, 7 1). Retrieved from Land Of Code: http://www.landofcode.com/java-tutorials/java-summary.php

KNN Classification. (2021, 7 20).

Levels in Data Flow Diagrams (DFD). (2021, 11 27).

Machine Learning Basics with the K-Nearest Neighbors Algorithm. (2021, 8 22). Retrieved from towards data science: https://towardsdatascience.com/machine-learning-basics-with-the-k-nearest-neighbors-algorithm-6a6e71d01761

Merge Sort Algorithm. (2021, 11 17).

opencv-python. (2021, 8 1). Retrieved from pypi: https://pypi.org/project/opencv-python/

ResearchGate. (2021, 11 29).

Smith, S. (2021, 7 1). *Android GUI Programming*. Retrieved from Small Business: https://smallbusiness.chron.com/android-gui-programming-33253.html

Standard Resistor Values: E3, E6, E12, E24, E48 & E96. (2021, 11 24).

What Are Wireframes? (2021, 8 2). Retrieved from balsamiq: https://balsamiq.com/learn/articles/what-are-wireframes/

Your brain is lying to you - colour is all in your head. (2021, 6 30). Retrieved from The Nature of Things: https://www.cbc.ca/natureofthings/features/your-brain-is-lying-to-you-colour-is-all-in-your-head-and-other-colourful-f