4th Year Project

AI Based Electronic Component Identifier

Violet Concordia
B00125142

Department of Engineering
School of Informatics & Engineering
Technological University Dublin, Blanchardstown
Dublin 15.

4th Year Project
15th of May, 2023

Table of Contents

Table of Figures

AI based Electronic Component Identifier

1. Details

Author: B00125142 Violet Concordia

Supervisor: Benjamin Toland

Course Title: Bachelor of Engineering (Honours) in Computer Engineering in Mobile Systems

Course Year: 4th

Course Code: TU807

2. Declaration

The material contained in this assignment is the author's original work, except where work quoted is duly acknowledged in the text. No aspect of this assignment has been previously submitted in any other unit or course.

Signed: _________________________ **Date:** _________

3. Abstract

This report illustrates the development of a 4th year Computer Engineering Project, which is designed to tackle the issue of computer-based detection of objects present in any given image from a camera image/video feed. While object detection is a very vast topic, the project focuses specifically on electronic component detection and identification. It specialises in reading additional information from the electronic components, such as their rating values, or marking codes.

The focus audience of this project is engineers. It hopes to assist people in becoming engineers by providing a useful tool for identification as well as inciting curiosity to investigate both the electronic components and inner workings of the project. It also aims to be a valuable tool for existing engineers by assisting identification of electronic components in bulk.

The project utilises Artificial Intelligence to achieve its goals with high confidence values. To run AI a model is required. The model must be trained from a dataset. The dataset for the training of the model is gathered through the set rig, which consists of a camera stand with a surface that the items are placed on, and the camera attached to it with a ring light positioned directly above the placed items, for optimal and consistent viewing angles and visibility.

The project is controlled through a GUI application that provides feedback on the information gathered, and the tools set up to interact with the device. Currently the GUI application is deployed on a desktop machine, and the camera on a set rig.

The desktop receives the information from the camera through a USB connection. The purpose of the set rig is to both provide an efficient environment for data gathering purposes, and to constrain the amount of variation a tested-upon image has exposure to. This drastically reduces the angles, lighting conditions, distance, and background variations of the input images, which is highly beneficial for a higher confidence value of the object detections.

Thanks to the project being developed and running on C++, in addition to using the YOLO architecture – the project has the opportunity to expand into a high confidence, high efficiency project, constantly-improving through user-submitted dataset that is deployable on desktop computers, laptops, microprocessors, and mobile (Android/iOS) devices.

4. Acknowledgement

I am very grateful to Technological University Dublin for accepting me as a student, and providing me with the opportunity to take on this project.

I'd like to acknowledge and express my gratitude towards my project supervisor, Benjamin Toland, who has taken on me and my custom project and has provided excellent guidance and feedback throughout the development of this project.

I am also very grateful for the existence and availability of search engines, for obvious reasons.

I would also like to thank my wonderful friends, old and new, peers that I was able to meet thanks to my ability to attend our campus, and the quiet spaces provided for us to further our education with minimal interruptions.

I am also grateful for having been introduced to Qt Creator by my brother, Justas Bartnykas. Qt Creator has been my go-to IDE for development of C++ code for the past beyond 5 years now. I have also been provided access to a fairly expensive camera that he owned which allowed the project to begin sooner than it otherwise could have.

I'd like to offer my sincere apology and thanks to anyone else I may have missed that has contributed to this project in any way.

5. Introduction

5.1. *Acronyms*

IDE – Integrated Development Environment

A sophisticated text editor, designed specifically for code development in certain languages. Most IDEs today come with a high range of optional plugins that can be used to further increase production speed, and reduce redundant tasks via automation of said tasks.

YOLO – You Only Look Once [1]

An image detection architecture that the project is based on.

SIP – Single Inline Package

Several of same type of component, all packaged in a single line.

Example: an SIP resistor, which features multiple resistors, all connected to a single ground pin.

AI – Artificial Intelligence

A term often used in deep learning, which is the process of attempting to simulate intelligence that is similar to that of a human by the use of a computer, in order to tackle issues that a standard style of computer operation either fails to, or performs at incredibly slow rates.

CPU – Central Processing Unit

The component of a computer where the core computations are processed.

GPU – Graphics Processing Unit

An optional component of a computer that is dedicated and optimised in computing graphical tasks.

LDR – Light Dependant Resistor

A resistor that varies in resistance relatively to the amount of light the body of the component is exposed to.

LED – Light Emitting Diode

An electrical component that emits light when current is passed through the circuit.

AC – Alternating Current

Electrical current that oscillates.

DC – Direct Current

Electrical current that stays constant.

PCB – Printed Circuit Board

A silicon board that has parts of it etched away, with only conductive tracks remaining in specific positions that are pre-planned using a CAD software. PCBs are widely used to implement electronic circuits.

OCR – Optical Character Recognition

Software based reading alphabetical characters from an image that contains written text.

RGB – Red Green Blue

Commonly used to referred to a way of defining colors by their Red, Green and Blue properties.

HSV – Hue Saturation Value

Commonly used to referred to a way of defining colors by their Hue, Saturation and Value properties.

FPS – Frames Per Second

A measure of video refresh rate.

NMS – Non-maximum Suppression

A filtering technique used on predictions of object detectors.

Picks the smallest intersection of bounding boxes, according to their confidence values.

SSH – Secure Socket Shell

A network protocol that allows users (most often system administrators) to remotely and securely access a shell of a remote device, even on an unsecured network.

HTTP – Hypertext Transfer Protocol

An application layer internet protocol used to transmit media such as documents, images, etc...

HTTPS – Hypertext Transfer Protocol Secure

An extension of HTTP, providing security through encryption.

5.2. *The Problem*

Detection of Objects from An Image

When we look at an image we can immediately discern objects that are displayed without ever having to think about them. It is an effortless process.

We deduct from our past experiences to determine information almost immediately.

This ability of ours to recognise objects is thanks to millions of years of evolution in a world where not detecting a potential threat makes the difference between life and death. It is in our nature to detect objects within a moment's notice.

The problem arises when we want to implement a computer to process object detection for us.

A computer is built upon the most simple of arithmetic tasks, combined together from an incredible amount of transistors that perform these tasks, into massive systems – which operate entirely in digital binary.

Computers are designed to process computational tasks at utter precision and consistency.

A computer has no concept of learning from past experiences, nor any feelings that may affect its decisions. Given the same instructions it will produce the same results on its first day after manufacture, and the last day of its operation (provided the unit was not damaged in a way that it would yield unpredictable results).

When a computer is exposed to an image in form of a pixel grid, all it truly sees is numerical values that are assigned to each cell of the grid. Computers have no concept of color, let alone real life objects. The simple action of either moving or scaling an object completely changes the arrangement of the pixels that represent this object.

Examples

Figure 1: 220Ohm resistor: Body pixel close up

Figure 2: 220Ohm resistor: Single band pixel close up

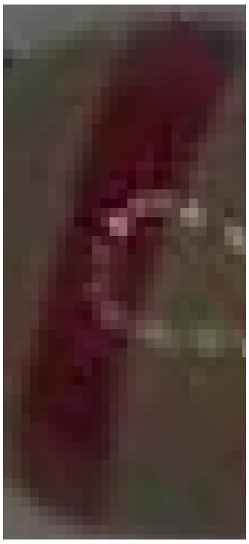


Figure 3: 220Ohm resistor: Single band pixel close up, with arbitrary rotation applied

The width and height of the image have both changed due to this rotation.

The order of the pixels has changed drastically.

To a person- this is no issue. It is obvious that this is still the same object. However, to a computer, the data has just shifted around entirely.

While this is a complex issue to tackle already, it only becomes more complex when we want to detect something generic, such as a tree.

A tree has many properties that may differ, such as: the presence, color, shape, and size of leaves/needles, the thickness, size, and color of trunk, branching styles, etc. while still being effortlessly identifiable as a tree by an intelligent creature.

Figure 4: Example of generic tree variations [2]

Importance of object detection

There are various cases in which automation of object detection is more beneficial as opposed to having a human constantly observing footage.

Security

In the field of security through digital cameras object detection is an incredibly useful tool for monitoring potential intruders inside a facility.

For facilities that utilise dozens, or even hundreds of cameras object detection is a very valuable tool that requires minimal human interaction while providing high level of certainty, and persistent monitoring.

Depending on the security requirements lower security facilities may not require hiring a person for constant monitoring of the security footage, and offers live notifications for any unexpected activity observed.

Production

Quality control

During production of anything from farm produce to electronic components, consistent detection and rejection of items with damage is essential.

With use of object detection flaws can be recognised incredibly efficiently, and this information may be passed onto the production line, identifying exactly which item was detected to have flaws, and be discarded automatically, without ever requiring any human interaction.

If the detection is sophisticated enough to be more reliable than a person, this opens up new opportunities for the improvement of speed and efficiency in the production, as a computer's computational power may be expanded unlike a person.

Analysis

Thorough inspection of potential objects on final products is a crucial part of many fields of production. Features that may have developed during the process of manufacture may be detected on the final products. This includes positive, negative, and purely analytical features.

Examples (this page)

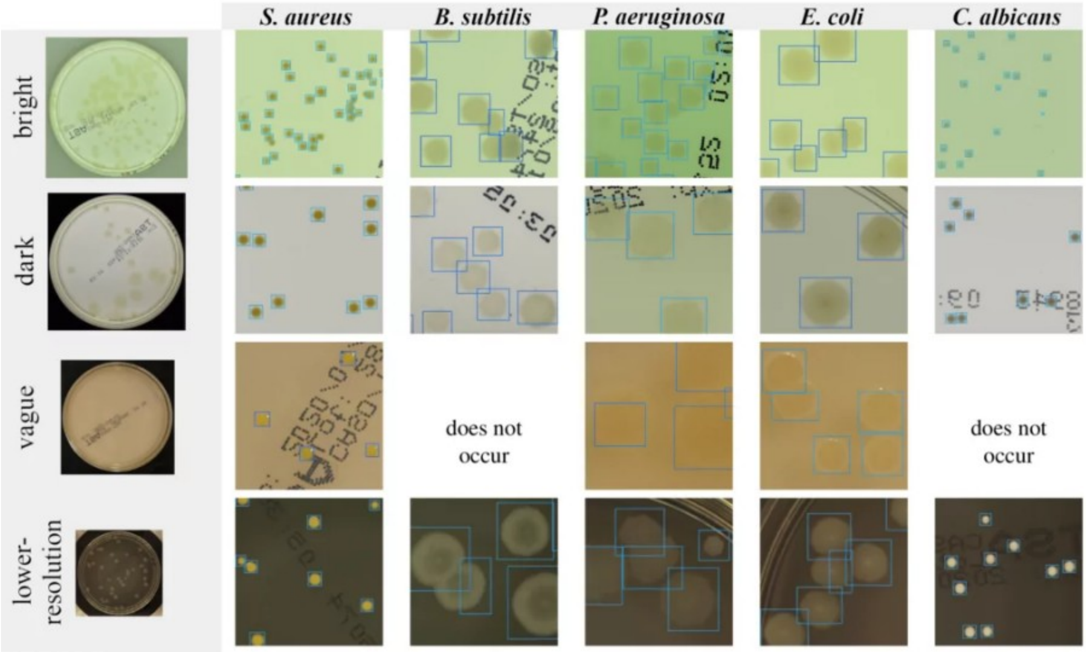


Figure 5: Analysis through object detection example: Petri dish [3]

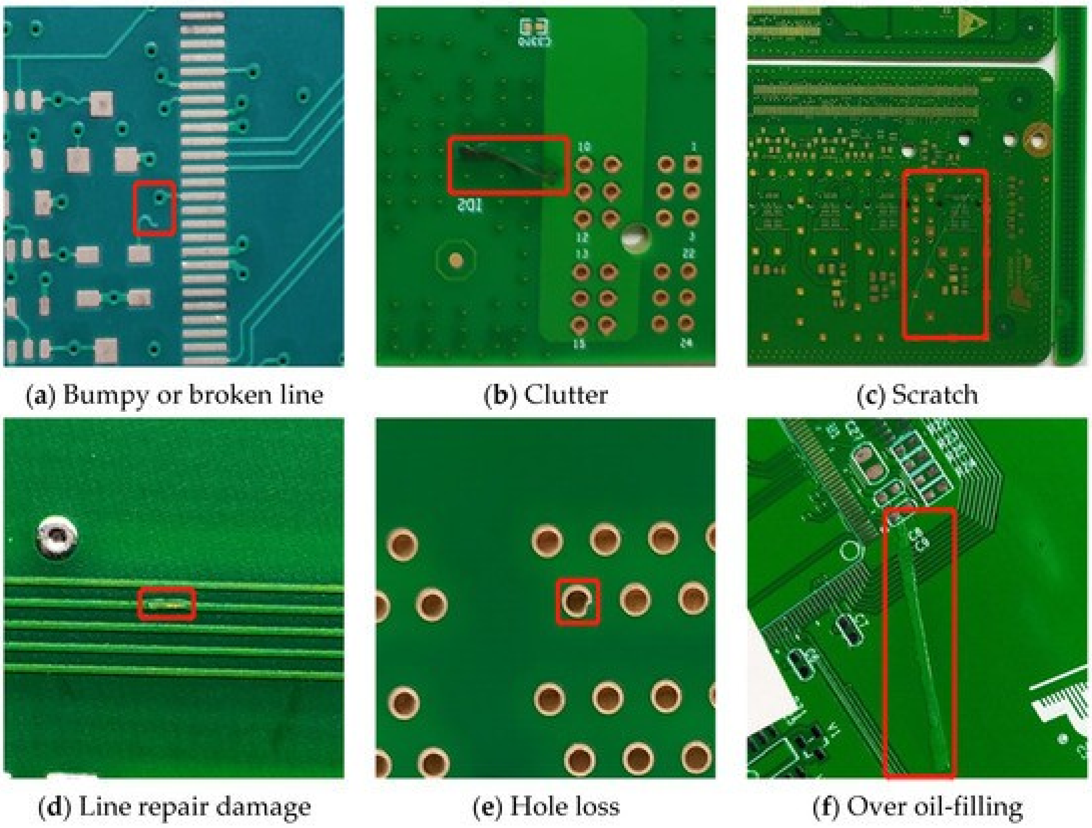


Figure 6: Analysis through object detection example: PCB manufacture [4]

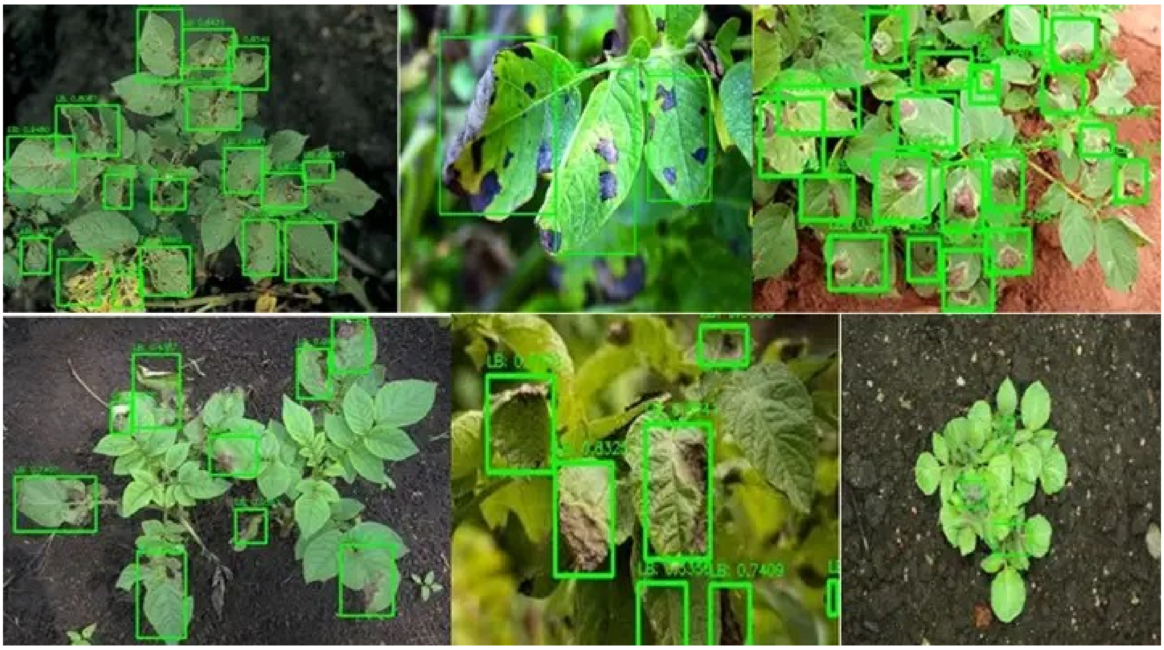


Figure 7: Analysis through object detection example: Crop disease [5]

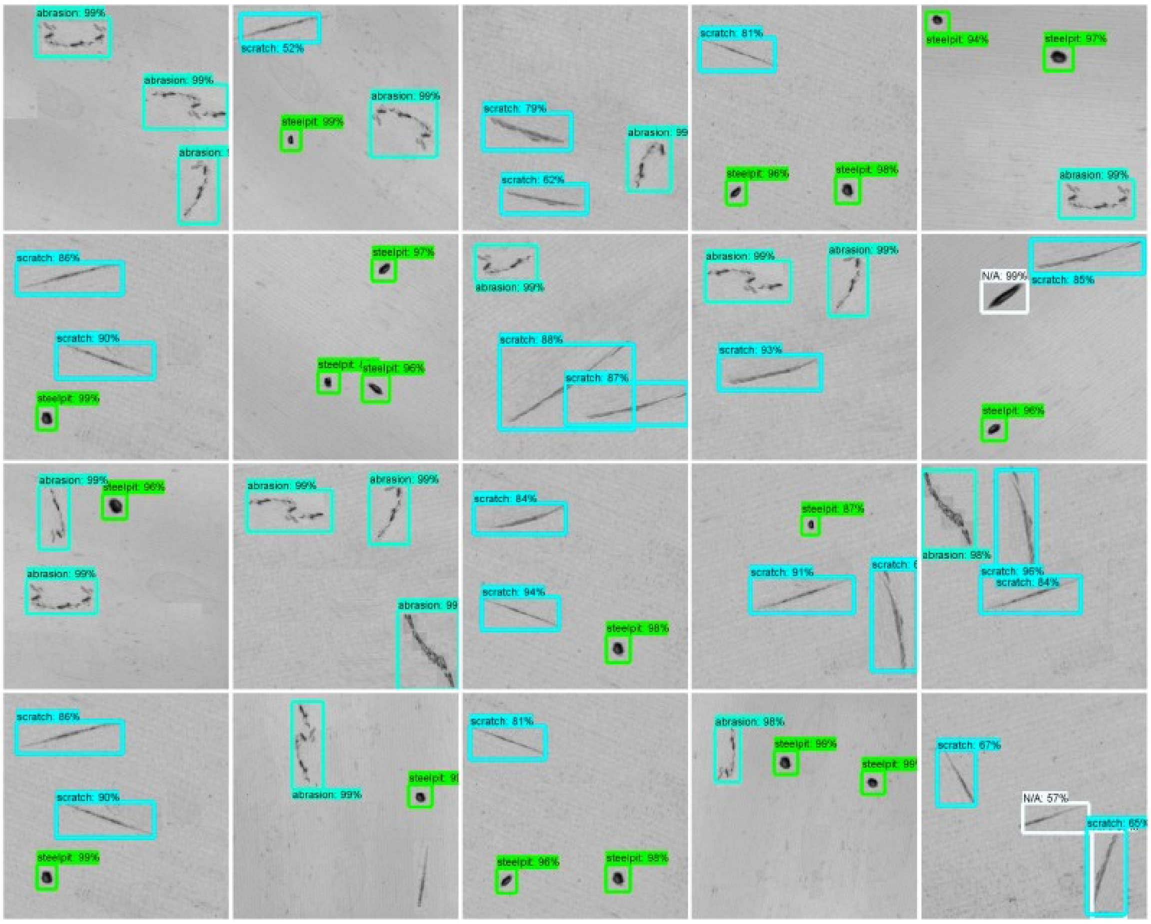


Figure 8: Analysis through object detection example: Surface Defection [6]

5.3. *Existing Solutions*

Algorithmic object detection

Algorithm based detection can be used to effectively identify very specific criteria, which can be expressed as an analytical value, or a trend.

Examples

- Specific color based properties.
- Specific shapes by following line trends.
- Must be coherent shapes, does not perform well with partial shapes.
- Specific patterns.

Note: This method will struggle heavily at detecting and identifying any generic object, for example: Trees, cats, dogs, people, cars, etc.

AI based Object Detection

Neural Networks

Description

Neural networks are an imitation of biological creatures intelligence, which is known as Artificial Intelligence. AI is designed to be ran on a computer.

Has the ability to be trained, which simulates past experience of biological intelligence.

A complex combination of usually many millions of digital neurons, with analog based values for each neuron, which result in a form of decision making based on a massive combination of criteria, rather than individual pixels.

These neurons work together to identify the incoming information and produce an output that resembles what the network has learned during the training of the model.

The networks are trained through deep learning.

Training using Deep Learning

Training is usually based off of a pre-trained model, that is trained on a big dataset. Most pre-trained models are trained on the COCO dataset, which is publicly available, and holds a vast amount of data. This kickstarts the model with data that it can repurpose to use as a base.

Initially, the model parameters are set to a random state.

The output that it produces when fed input data, is of course, also random.

Epochs are ran on the model to train the model.

Epoch

The process of tweaking the entire model based on the existing parameters, dataset, and the current output that it produces – by use of a loss function, randomness, and clever technique, in hopes of improving the detection ability of the data the model was trained upon.

The best performing model is kept between the newly trained model, and the previous best.

Many epochs are ran in order to polish the model as much as possible.

In the scope of this project, somewhere between 250 and 500 epochs will suffice, depending on the size of the dataset.

Inference

In object detection, inference is the utilisation of the model to process classifications of objects on the image.

Classification

The process of using the steps provided in the model to identify objects from an input image, through steps that depend on the architecture used.

The identified objects are marked with a bounding box, class they belong to, and confidence in the prediction value.

6. Description

6.1. *Summary*

AI based Electronic component identification, running on a custom-trained model that is trained during the development of this project.

6.2. *Features*

Detection of objects from an image via Inference

Detect and display boundaries for each identified class, and the confidence value of this detection from the input image using Inference.

Classes present in the dataset that the model was trained upon;

1. Resistor
2. Singular
3. SIP Resistor
4. Diode
5. Capacitor
6. AC
7. DC
8. LEDs
9. Integrated Circuits
10. LDR
11. PCB terminal

Live labelling

The ability to take a snapshot of the current frame, defining appropriate labels, and saving this labelled snapshot for future training. All from inside the GUI.

Alternatively, taking snapshots of the GUI and saving them for later labelling.

It should be noted that the labels must be adequately labelled.

If mass deployed with the ability to live label and submit the images to a server for further training, there must be a sophisticated method of ensuring that the submitted data is 100% accurate.

If an image is mis-labelled, or an object is missed – this would cause mis-training, or worse – de-training of the model, corrupting the dataset and setting the training **backwards** rather than forwards.

Potential solutions

Manual inspection of submitted images, for the purpose of quality control, which is crucial for the success of the model.

Submission of labelled images

Requires manual inspection.

Un-labelled images

Requires manual labelling.

Both of the solutions pose an issue of requiring manual labor from a person that is not the user, and is adequately knowledgable of which data is sufficient both in image quality, and labelling – which causes a linear demand of paid workers for the submission flow rate.

6.3. *Milestones*

Sorted from highest priority, to lowest.

Each of these steps should be polished before continuing to the next one, to provide a solid foundation for the next step to be based on.

Base camera rig

Setting up the camera on a rig.

Base GUI

GUI with essentials to interface the camera through USB, with

A live display from the camera on the rig.

Ability to take images by pressing a button.

Support for running Inference.

Initial dataset gathering

~100 images of a single class, taken from the rig for initial training and testing of the model.

Initial inference model training

For the purpose of testing inference on the rig.

The initial training should not take long at all, and does not require to be polished.

Training for ~100 epochs should be sufficient, with each epoch taking ~20 seconds on the machine available.

Inference running

Proof of concept. The results will not be perfect as the dataset is minimal, and only contains 1 class.

Further dataset gathering

At least 250 pictures of each class of every component that the project is designed to detect.

Further model training

This training will take considerably longer than the initial training. Around 2 minutes per epoch, and should be ran for at least 300 epochs.

The goal is to reach 0.8 from range of 0 to 1 confidence values.

Post-processing

Ability to gather further information from the detection bounding boxes provided by the inference.

Live training

Optional: Ability to label the images from the device, without requiring external software.

It may be advantageous given the time-frame of the project to instead gather data during a session and labelling it afterwards.

Existing labelling related software offers quality of life features, such as rough auto labelling of the images, which only requires the user to adjust the bounding boxes and confirm their validity, rather than having to define the boxes from start to finish.

Testing with video footage from a mobile device

After the previous steps are in good shape, investigation of moving the inference to a mobile device will begin.

If the confidence values are not up to standard, more data will be gathered from this and potentially other mobile devices, and further training will follow, until the results are adequate.

If adequate results are achieved before the deadline of this project, deployment to a mobile device will be started.

If the frame rates are not sufficient enough, the inference may be ran on still images to improve user experience.

7. Research

7.1. *Focus Audience*

While this project may be retrained and refocused to be utilised for many different fields – it is trained for electrical component identification, which is focused towards engineers.

This project focuses on both existing engineers, and ones that are interested in becoming engineers.

Having access to the provided by the project quick identification of components, count of each, and any potential additional information saves time spent manually analysing this information.

7.2. *Architectures*

7.2.1. R-CNN [7]

Description

R-CNN, which stands for Region Based Convolutional Neural Networks was released in 2013. It was developed by Ross Girshick. As other object detection architectures, R-CNN takes an input image, and outlines bounding boxes where it believes an item of a certain class is present.

Disadvantages

- Not real-time.
- On average, takes 47 seconds to process a single frame.
- According to the Git repository, the project was seemingly abandoned about 5 years ago.

Discussion

It should be noted that R-CNN has a successor called Fast R-CNN and Faster R-CNN.

However, even the fastest of the successors still barely manage 5 frames a second at best.

While 5 frames a second is an impressive and definitely useable result, there are alternative architectures that offer a significant improvement in inference time.

7.2.2. SSD

Description

SSD, which stands for Single Shot Detector was released in 2017 by Max deGroot and Ellis Brown.

Disadvantages

Similar to the previous entry on the list the project was seemingly abandoned about 4 years ago.

Discussion

Offers great frame-rates of an average of 45 frames per second when tested on a relatively old graphics card, namely the NVIDIA GTX 1060.

7.2.3. Ultralytics YOLO

Figure 9: Ultralytics YOLO

[1]

Discussion

YOLO, which stands for You Only Look Once is a popular image segmentation and object detection model that was originally developed by Joseph Redmon and Ali Farhadi. As the name suggests, YOLO focuses on detection of multiple classes in a single "look", which is a single analysis of the entire input image. An approach like this may seem too good to be true, and that it should come with significant cost to the speed and confidence of the model. But when the results are analysed, that could barely be further from the truth.

YOLO is an incredibly efficient and accurate architecture. When compared to many other architectures before YOLO, realistically no matter how quick the other architectures may be- it is an incredibly superior approach as other architectures would approach detection by reanalysing the entire image for every single class that the model was trained for– increasing the time taken per detection additively per class.

These days most sophisticated architectures approach object detection similarly to YOLO, but YOLO still boasts being a state-of-the-art architecture that continues to improve and grow to this day.

Because of how far ahead the YOLO architecture is when compared to most other architectures, it is utilised very commonly throughout object detection projects.

Brief History

Versions

- **YOLOv1**

The first version was released in 2015, and it very quickly became popular due to the significantly superior speed and accuracy when compared to other architectures.

- **YOLOv4**

Released in 2018, Introduction of Mosaic data augmentation, and a new and improved loss function – decreasing time taken to achieve better results for the trained model.

- **YOLOv5 [8]**

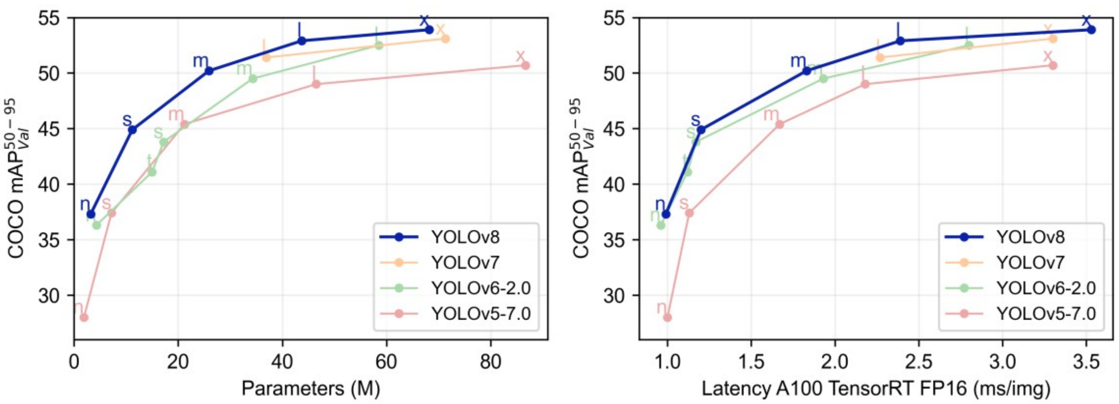
Released in 2020, Introducing support for Object Tracking – which allows following a moving object, and Panoptic Segmentation, which allows identification of overlapping objects, with accurate bounding boxes.

- **Ultralytics YOLOv8 [9]**

The latest version of YOLO as of today. YOLOv8 is a state-of-the-art model that builds upon the already very successful previous YOLO versions, introducing new performance and flexibility features.

Full support for previous YOLO versions, making it incredibly convenient for existing users of previous YOLO versions to take advantage of the new features.

Comparison



[9]

Figure 10: YOLO family comparison chart

In general, YOLOv8 is superior to all of its predecessors.

While YOLOv5 is mostly underperforming when compared to the next versions, it is important to note how incredibly minimal the delays are even on a version so outdated now.

7.2.3.1. Pre-Trained Models

YOLO offers pre-trained models that are used to start train custom models.

Model	size (pixels)	mAP^{val} 50-95	mAP^{val} 50	Speed CPU b1 (ms)	Speed V100 b1 (ms)	Speed V100 b32 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n	640	28.0	45.7	**45**	**6.3**	**0.6**	**1.9**	**4.5**
YOLOv5s	640	37.4	56.8	98	6.4	0.9	7.2	16.5
YOLOv5m	640	45.4	64.1	224	8.2	1.7	21.2	49.0
YOLOv5l	640	49.0	67.3	430	10.1	2.7	46.5	109.1
YOLOv5x	640	50.7	68.9	766	12.1	4.8	86.7	205.7
YOLOv5n6	1280	36.0	54.4	153	8.1	2.1	3.2	4.6
YOLOv5s6	1280	44.8	63.7	385	8.2	3.6	12.6	16.8
YOLOv5m6	1280	51.3	69.3	887	11.1	6.8	35.7	50.0
YOLOv5l6	1280	53.7	71.3	1784	15.8	10.5	76.8	111.4
YOLOv5x6 + TTA	1280 1536	55.0 **55.8**	72.7 **72.7**	3136 -	26.2 -	19.4 -	140.7 -	209.8 -

Figure 11: YOLOv5 pre-trained model choice [1]

Model	size (pixels)	mAP^{val} 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Figure 12: YOLOv8 pre-trained model choice [1] [1]

Analysis of the model properties

- **Size:** The pixel height and width the model operates up to.
- **mAP:** Single-model single-scale values while detecting on the COCO val2017 dataset.
- **Speed:** Averaged time taken using the Amazon EC2 P4d instance on the COCO dataset.
- **Params (In Millions):** The number of parameters that are tweaked per epoch while training, and processed during inference.
- **FLOPS:** Floating Point Operations Per Second. A measure based on Floating Point Operations that is relevant in the field of Deep Learning.

Observations

In comparison of YOLOv5 and YOLOv8 versions a clear advantage can be seen when taking into account the size of the model (param count), and the resulting mAP output, as well as the time taken.

Each model has its advantages and disadvantages, and should be picked depending on the project.

Diminishing results can be observed on the mAP values when compared to the time taken (Speed).

In some circumstances, maximum precision is essential, and is prioritised over the hardware requirements. This is when a higher model should be chosen.

In the scope of this project – several pre-trained models have been used, including both YOLOv5 and YOLOv8, for the purpose of comparison.

Internal steps of the You Only Look Once Inference

Figure 13: YOLO detection: Input Image [10]

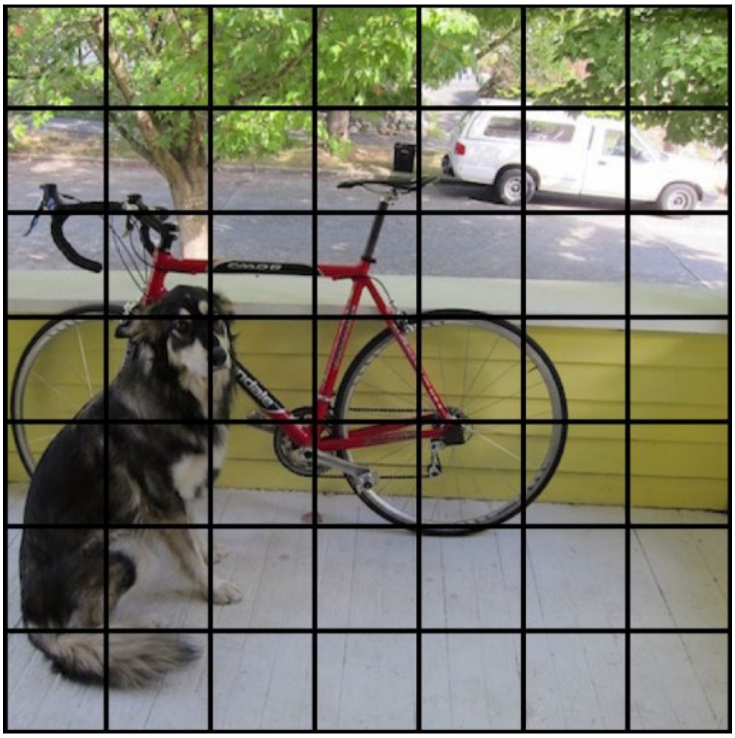


Figure 14: YOLO detection: Splitting into S by S grid, S being 7 in this figure. [10]

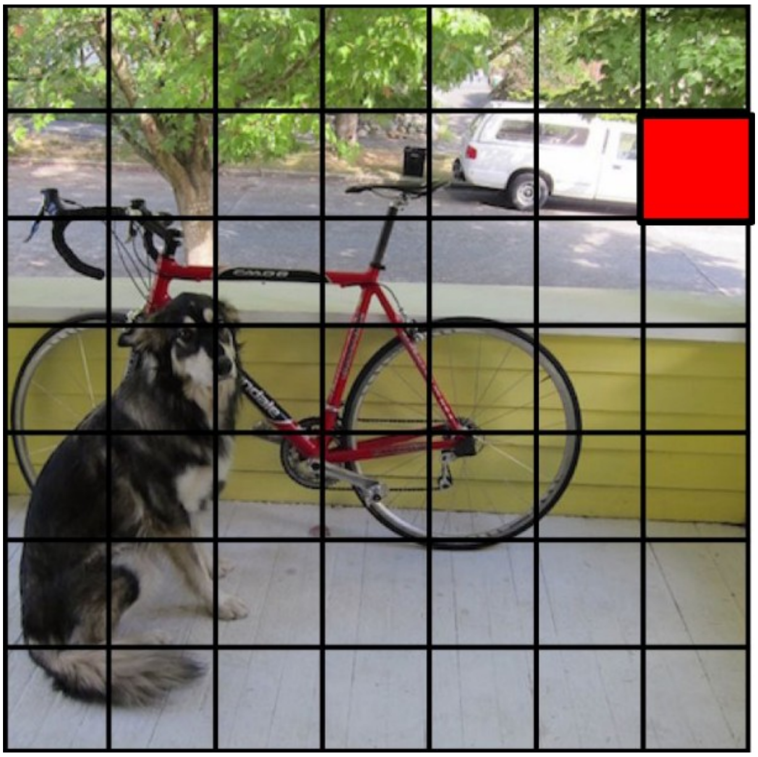


Figure 15: YOLO detection: Each cell predicts the bounding boxes and confidence values of each box. [10]

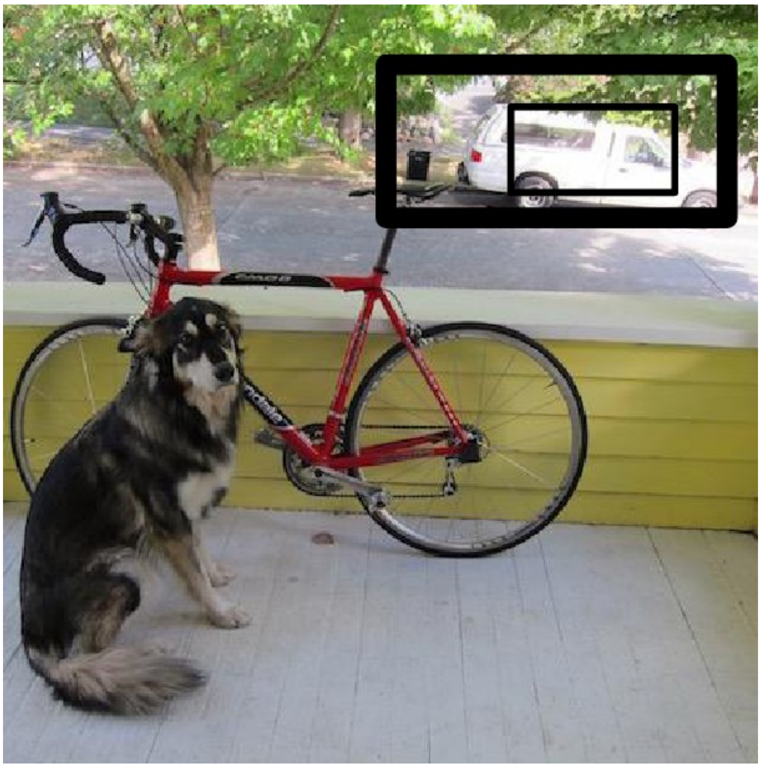


Figure 16: YOLO detection: Identified bounding box. [10]

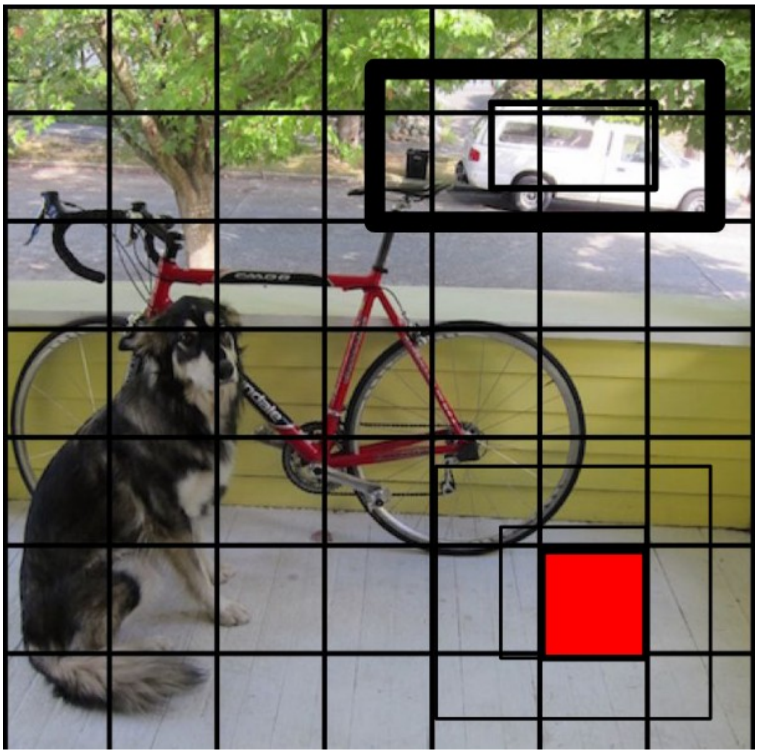


Figure 17: YOLO detection: Previous steps being repeated for each cell. [10]

Figure 18: YOLO detection: All identified bounding boxes. [10]

Figure 19: YOLO detection: All identified probabilities for each grid cell.

[10]

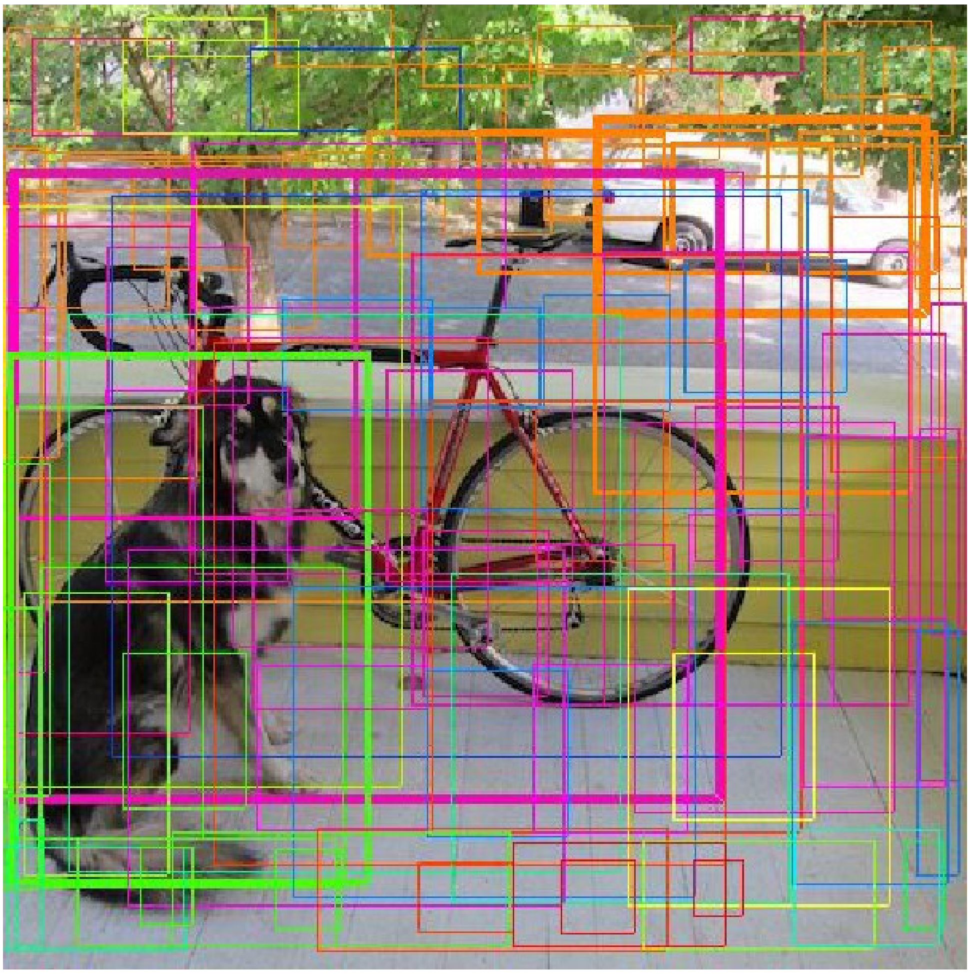


Figure 20: YOLO detection: Each bounding box is "shaded" in the case of this example with how much each one covers of which probability grid.

[10]

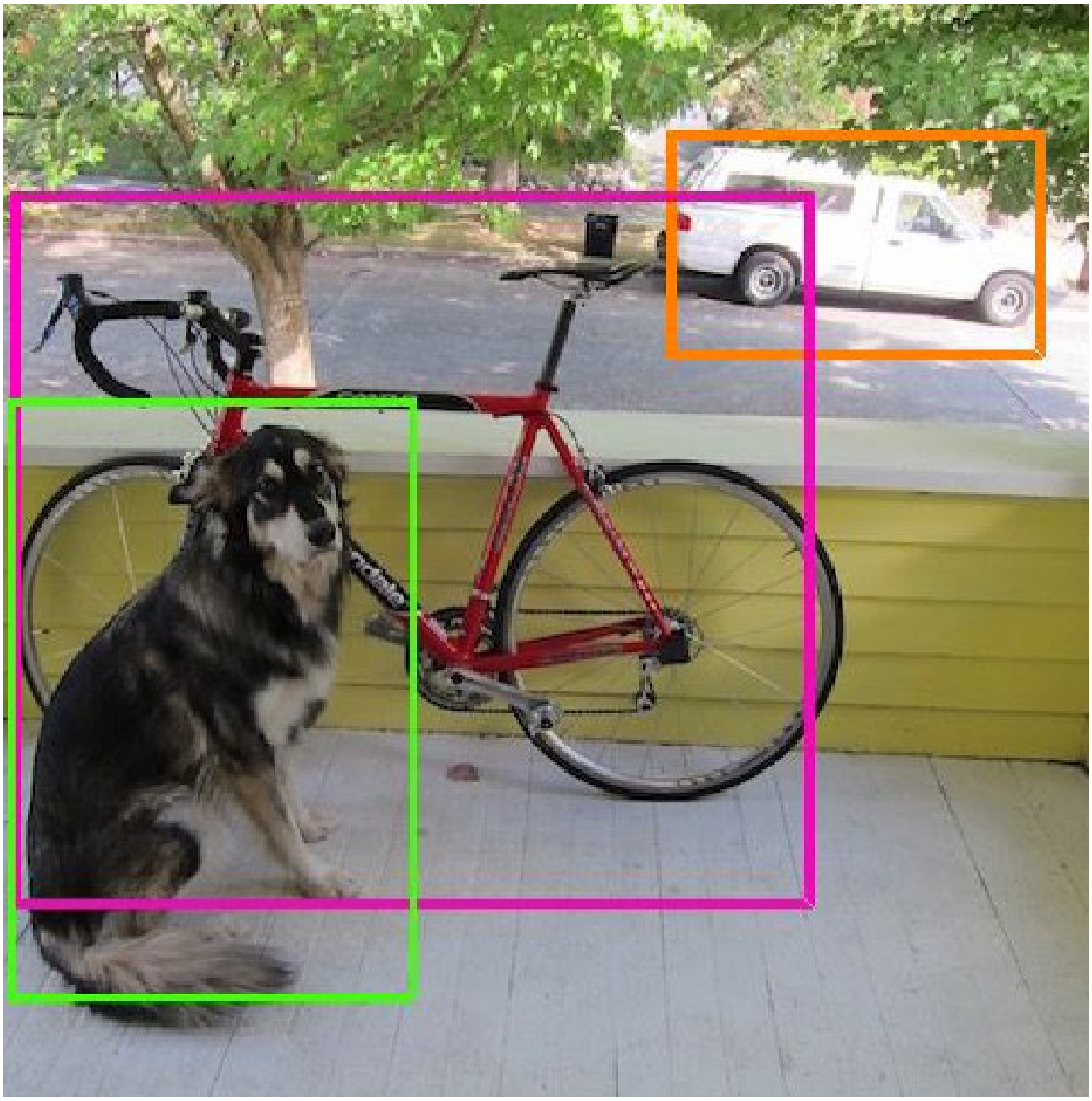


Figure 21: YOLO detection: The bounding boxes are reduced using NMS. This is the final step. [10]

This is the final output that YOLO provides: Bounding boxes with classification (In this example, the classification is marked by a color. The real output is a string.), and the confidence value (In this example, confidence is marked by the opacity of the boxes. The real output is a number ranging from 0 to 1).

7.3. *Rationale*

7.3.1. IDE

Qt Creator [11] was chosen for being cross-platform, and being a personal choice of IDE for development of C++ based GUI applications.

7.3.2. Version Keeping

Git [12] was chosen as the version control system because it is open source, fast, efficient, and widely supported by many applications, including the IDE.

7.3.3. Git GUI

GitKraken [13] is a powerful and intuitive client for Git that offers very informative visuals. While there are many alternatives, GitKraken was chosen both because of its specifications, and the fact that it is a personal preference.

7.3.4. Language

C++ [14] was chosen for being cross-platform, efficient, low level, and overall personal language of choice for GUI application development, with years of experience.

Benefits:

- **Efficiency:** Allows more potential for implementation on lower end systems,
- **Low level:** Allows lower access to USB devices and hardware, as well as hardware acceleration options,
- **Cross-Platform:** Widely accessible.

7.3.5. Architecture

YOLO [1] has been chosen as the architecture that this project utilises for the AI detection. It offers incredibly efficient, and accurate results – and is highly scalable when it comes to identified component count due to the You Only Look Once approach.

At the start of the project, there was already a high bias towards YOLO due to the highly positive past experience with YOLOv5 and all the incredible features that it offers.

Upon release of YOLOv8 and all the superior features and specifications that it provides on top of the previous versions – the YOLO family was an obvious choice in the architecture that will be used for the project.

7.3.6. Set Rig

Figure 22: The rig used for this project, with the camera attached.

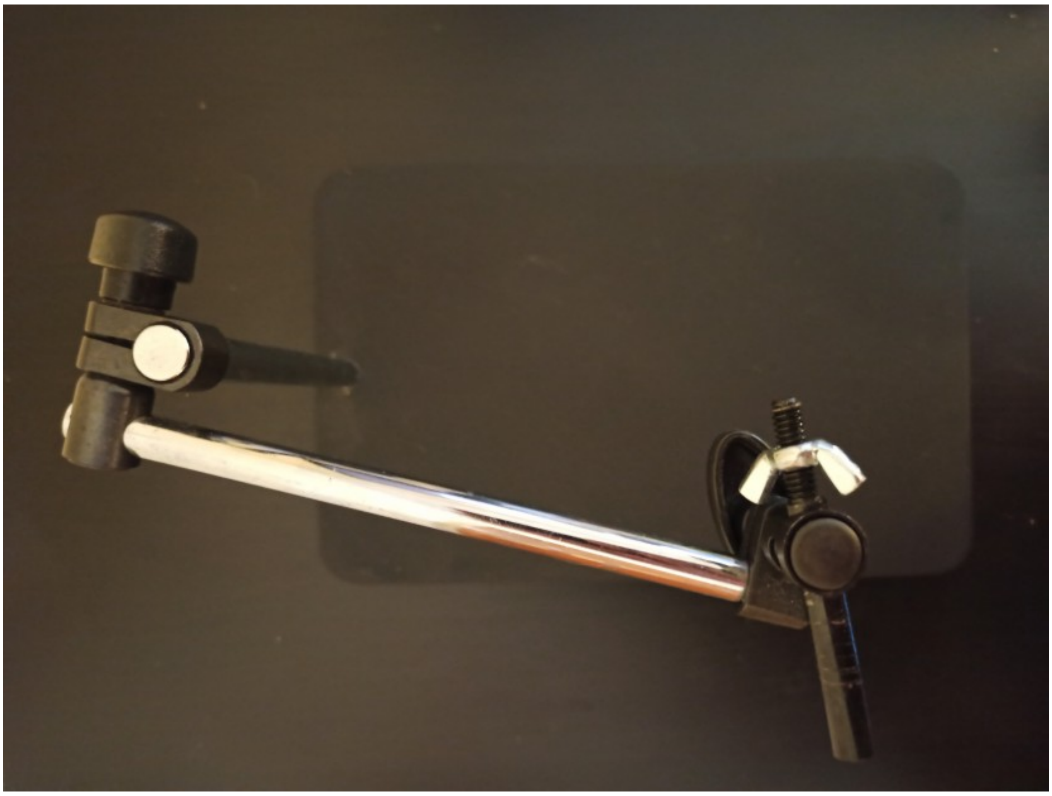


Figure 23: The rig used for this project.

Dimensions

Base Width: 280mm

Base Length: 180mm

Height: 175mm to 350mm (Adjustable arms)

Cost: ~40€ [15]

Choice Rationale

There is a high variety of camera rigs available for purchase and for the purposes of this project – as long as the rig has a surface and the camera has a convenient way to be mounted – there is not much of a difference on which rig is chosen, provided it is adequate in dimensions and is capable of steadily mounting the camera.

The rig is quite basic, but does the job well, and was already available at home. Vast amount of alternatives are available online

7.3.7. Camera

Figure 24: Camera, model OSY-C100-4-12. Camera of choice for the project. **[16]**

Model: OSY-C100-4-12 [16]

Specifications

Video

- **HDMI:** 4K@30fps, 1080p@60fps
- **USB:** 1080p@30fps

Image

- **Resolution:** 12 Megapixels
- **Pixel Size:** 1.55µm by 1.55µm
- **Dimensions:** 59*95*22.6mm
- **Cost:** ~250€

Chosen mode of operation: USB connection – 1080p@30fps.

This mode was chosen due to the high compatibility range of a USB connection, and both the resolution and frame-rate being more than sufficient for the purposes of the project.

Choice Rationale

While this camera is over-qualified for the project, as mentioned in the Acknowledgement section, this camera was available at the start of the project, and was chosen out of convenience of availability.

For the purposes of object detection at the scale of this project, 1080p at 30 frames per second is plenty of incoming data.

The quality of the output images has been tested to be sufficient.

7.3.8. Ring Light

Figure 25: Ring Light used for this project. [17]

Specifications

- AC power plug.

- Maximum diameter 65mm.
- Adjustable intensity via knob on the side.

Cost: ~15€ [17]

Choice Rationale

This ring right was chosen due to being far more than sufficient in supplying adequate lighting, and being reasonably cheap.

Training

- **Angle range**

Due to the angle and lighting both being known and mostly set thanks to using a set rig, the input dataset does not need to cover angles and lighting outside what the rig will expose it to during runtime.

The angle range is reduced only to looking from top to down, eliminating the rest of the angle range.

Only the components being detected need to be trained in all angles, as opposed to the camera gathering the dataset requiring to be positioned in different angles.

- **Lighting**

While the lighting will change depending on the room conditions, the ring light around the camera will provide significant consistency in lighting.

While this does not not eliminate the necessity to train against various lighting conditions, it does reduce their significance and increase certainty of the detection.

Having a top to down view also eliminates the majority of issues that come with glare from high luminosity bodies, such as clouds or the sun.

A set rig significantly limits the distance that the objects will be from the camera during runtime, allowing for further confidence in the predictions.

- **Static background**

Apart from dust or unexpected objects present on the rig's surface, which should be removed before usage – the background that the objects are in front of will stay mostly consistent.

This reduces the necessity to gather data of the same object under backgrounds that are not expected to be used during runtime.

The sum of all the points covered above results in a significant reduction in data required to train when compared to a setup without a set rig, for equivalent confidence values during runtime.

The reduction of data required to train makes it feasible to train relatively high quality models from data gathered and trained from home.

- **Running**

The set position of the camera, a significant reduction in distance between the objects, and significant consistency of the lighting provided by the ring light, and the static background – will boost the confidence of the inference considerably.

7.4. *Hardware*

7.4.1. Training

Rented Dedicated Server

Advantages

- **Speed**

Utilises multiple GPUs – Quicker epoch computations, resulting in quicker training.

- **Cloud based**

Allows for parallel computing, as opposed to using your personal computer at home, which will slow down any work required to be done on the computer.

- **Disadvantages**
    - Setup time – must setup the cloud environment to reliably receive and train the dataset. This task is relatively simple, but it should be setup well as a solid foundation for the cloud computation to be quick and easy to interact with and manage.
    - Cloud based – long upload and download times, because datasets tend to be considerably big in size. A smaller dataset of ~2000 images takes up ~3gb of space. This is not a significant amount of data for a local machine to transfer, but it is a considerable amount for uploading.
- **Cost**

The bigger/computationally more capable the server – the higher the rates become.

Personal Computer

Advantages

- **Setup time**

Local – Provided a local machine is already owned, it is immediately available.

Pictures are taken from the machine itself. No upload/download times.

- **Cost**

As opposed to a rented server – acquiring your own machine has the benefit of owning the machine, and being able to use it indefinitely (Or until it eventually breaks.)

While the initial cost of acquiring an adequate machine for deep learning is higher than renting a server for a few months, it is a worthwhile long-term investment into a machine that can be used for a variety of casual or intensive tasks for an engineer.

Disadvantages

- **Speed**

A local machine will likely contain one, maybe two GPUs.

When compared to a sophisticated server that runs many GPUs – a local machine will most likely process the training at a slower rate than a dedicated server would.

7.4.2. Inference

Discussion

After the training is done, which is usually over the span of 10's, and sometimes 100's of hours, depending on the size of the dataset and the epoch count – Running the trained model for inference only takes time in the range of milliseconds to process a single frame.

How long specifically is directly tied to the speed of hardware that the model is being ran on, and the size of the model.

Even with all the speed optimisations offered by the YOLO family, a lower end device such as a Raspberry Pi 4 may take 1-2 seconds to process a single 360p image.

It is important to pick appropriate hardware for one’s particular use cases.

7.4.2.1. Devices

Microprocessors

- **Raspberry Pi 4 [18]**

Specifications

CPU

  - **Core Count:** 4
  - **Maximum Frequency:** 1.5GHz

GPU

  - **Core Count:** 4
  - **Maximum Frequency:** 700MHz

- **Beaglebone [19]**

Specifications

CPU

  - **Core Count:** 1
  - **Maximum Frequency:** 1GHz

GPU

  - **Core Count:** 2
  - **Maximum Frequency:** 532MHz

- **Nvidia Jetson Nano [20]**

Specifications

CPU

  - **Core Count:** 4
  - **Maximum Frequency:** 1.479GHz

GPU

  - **Core Count:** 128
  - **Maximum Frequency:** 921MHz

Observations

This microprocessor is targeted towards quick graphical computations, which can instead be used for deep learning.

USB computation extensions

- **Intel Neural Compute Stick 2 [21]**

Ease of access

Due to the device being specialised for neural computations, it is not a common device by any means.

Combined with the price tag of ~100 eur, this device will likely only be owned by developers, as opposed to users.

As this device is unlikely to be owned by a user of the project, it would not be wise to require owning one to run our inference.

Discussion

The project will be able to support a compute stick as an alternative to a GPU.

Advantage

Offers computational power through a USB connection – can be used to run Inference on existing devices, such as a laptop.

Specifications

  - **Core Count (SHAVE [22]):** 16
  - **Processor Base Frequency:** 700MHz
  - **Memory:** 2GB

- **Android [23] Phone**

Specifications

Specifications depend on the specific device.

There are countless types of android devices on the market, all with varying specifications.

Camera

The vast majority of mobile phones on the market today have a built-in camera.

Ease of access

Widely and easily accessible.

Most people own a mobile device, and have it on them in most cases.

The app may be obtained from an App Store, that mobile devices have easy access to, as long as they have access to the internet.

- **Windows [24]/Linux [25]/Mac [26] Desktop/Laptop Machine**

Specifications

Specifications depend on the specific device.

The specs of a desktop/laptop machine will most likely beat the specs of both a phone, and a microprocessor.

Ease of Access

Desktops are widely accessible in environments where it would be relevant to use this project, such as the home of the user, or the campus a student is in.

The machine must have permissions for USB connections and running the application.

Cross-Platform

The application is designed through Qt Creator [11] and therefore is cross-platform.

Cloud Machine

Performance directly correlates to the cost.

Workload

The inference and post-processing workload would be processed solely on the cloud.

New Constraints

Because the processing would be done on the cloud, access to internet would required.

Having a live, lossless video feed would be essential for adequate results.

This would put a huge strain on the required processing power on the user's device.

A compromise could be to replace live video feed processing with image submissions instead.

However, even a mobile device would likely be able to complete processing a single frame in the duration it would take to send the data on to the cloud, the cloud to process it, and then receive the processed data back.

This approach in the end may not reduce the processing power significantly enough, and may actually introduce latency, depending on the connection quality.

Cost

Running a server costs a monthly fee that may be avoided by running the server from home, but doing so would hinder the expandability of the project, most likely resulting in dissatisfaction through latency for the users.

7.5. *Software*

7.5.1. Identification

Description

Post-processing of the components in the bounding boxes detected by inference, which may have additional information that can be identified by a variety of approaches.

Marking Codes

- **Resistors**

Color	Color	1st Band (x10Ω)	2nd Band (Ω)	3rd Band (Multiplier)	4th Band (± Tolerance)
Black		0	0	$x10^{0}$	20.00%
Brown		1	1	$x10^{1}$	1.00%
Red		2	2	$x10^{2}$	2.00%
Orange		3	3	$x10^{3}$	3.00%
Yellow		4	4	$x10^{4}$	100.00%
Green		5	5	$x10^{5}$	0.01%
Blue		6	6	$x10^{6}$	0.25%
Violet		7	7	$x10^{7}$	0.10%
Grey		8	8	$x10^{8}$	0.05%
White		9	9	$x10^{9}$	10.00%
Gold				$x10^{-1}$	5.00%
Silver				$x10^{-2}$	10.00%

Figure 26: Resistor Color Code Table

The most common type of resistor has 4 bands. 1 for tens, 1 for units, 1 for multiplier, and 1 for tolerance. However, these codes may be used on any higher number of bands.

For the sake of an example, let's take a resistor that has 6 bands: brown, green, red, orange, black, silver.

This resistor has 1 thousand, 5 hundreds, 2 tens, and 3 units, with a multiplier of 10^0, and 10% tolerance. This resistor is 1523Ohms, give or take 152.3Ohms.

A real life example

Figure 27: 220Ohm resistor

Color codes

Brown = 1

Red = 2

Gold = 5% tolerance

Calculations

$$Resistance\ (Ohm) = (10*(1st\ band) + (2nd\ band)) * 10^{(3th\ band)}$$

$$(10*(2) + (2)) * 10^{1}$$

$$220 Ohm$$

$$220 \pm (5\% * 220)$$

$$5\% * 220 = 11$$

$$209\ to\ 231 Ohm\ Resistor$$

Capacitors

	1st Number (x10pF)	2nd Number (pF)	3rd Number (Multiplier x10^n)	Capacitance (pF)
Examples	1	0	0	10
	3	3	1	330
	1	5	2	1500

Figure 28: Capacitor Code Table

Figure 29: 100nF Capacitor [27]

$$Capacitance\ (pF) = (10*(1st\ number) + (2nd\ number)) * 10^{(3th\ number)}$$

$$(10*(1) + (0)) * 10^{(4)}$$

$$100000pF$$

$$100nF$$

Integrated Circuits

Unfortunately for the purposes of automatic identification of Integrated Circuit markings, most IC manufacturers do not follow any global standard for marking their ICs.

Most manufacturers tend to have their own internal IC marking standards.

Due to this fact – only known markings can be used to identify components.

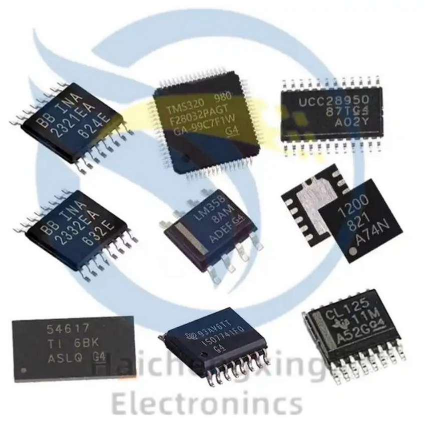


Figure 30: Variety of labelling on components that originate from different IC manufacturers. [28]

7.5.1.1. Examples

Input image

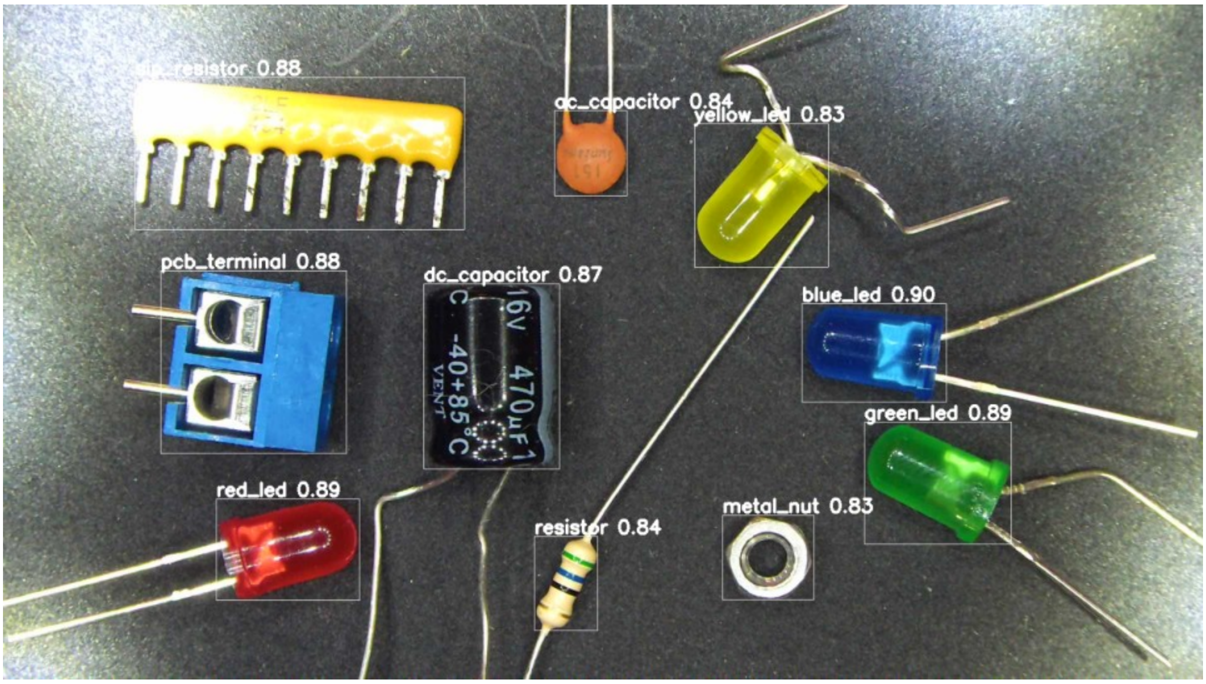


Figure 31: Example of 3rd trained model running Inference.

Note: *This is an old example to illustrate progression. The text and bounding boxes become significantly more distinguishable with progression.*

LED color

- **Inference method**

Different color LEDs may be trained as individual classes.

Has the disadvantage of requiring training for each individual LED separately, as opposed to one generic LED.

- **Algorithm method**

The most prominent color may be identified by sorting all the colors from the image into their hue values, and checking which hue is most active.

Has the advantage of working on any LED.

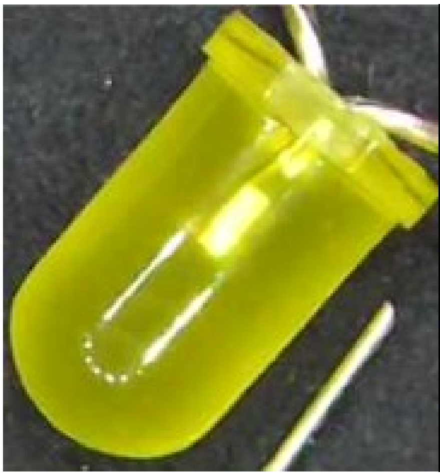


Figure 32: Color extraction example: Raw input image

Figure 33: Color extraction example: Heavy contrast applied

- **Contrast approach**

Has the disadvantage of potentially giving false information if the background is too vibrant.

Approaches After Filtering

Color Histogram

Figure 34: Color extraction example: Color Histogram (Example from Photoshop UI) [29]

This histogram was ran on the contrast applied version of the image.

Results show clearly prominent yellow, which is accurate.

HSV Analysis

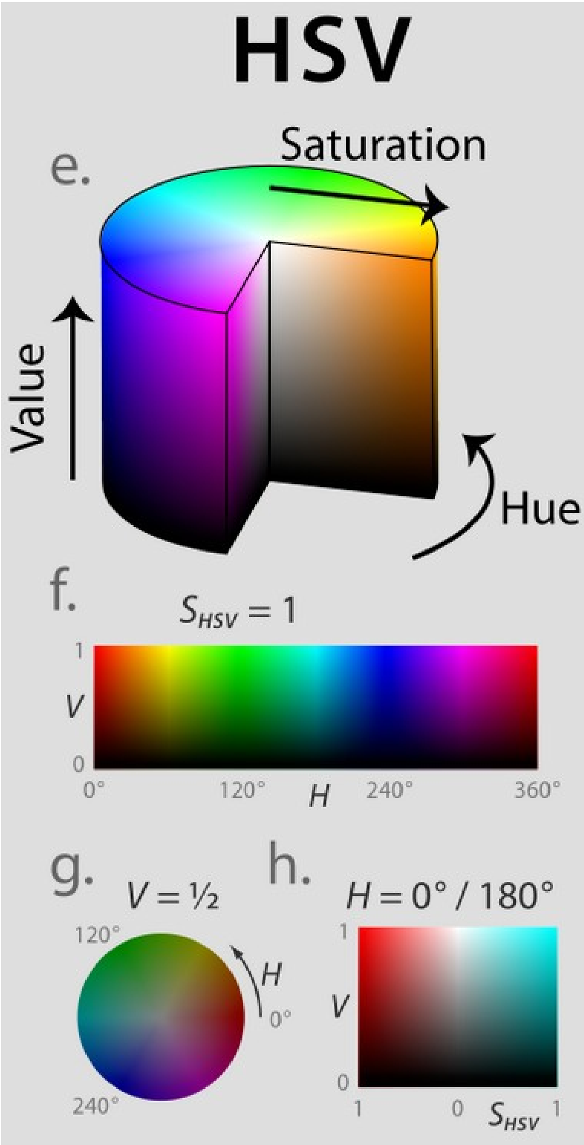


Figure 35: HSV chart overview [30]

HSV, or Hue Saturation Value, is an alternative way to represent colors.

It can be advantageous over RGB in situations such as this.

Taking the average of all the pixels hue values that have a value above a certain threshold. Around 0.7 on a range from 0 to 1 should be appropriate.

Note: This example would ignore colors that are darker than 0.7, on a range of 0 to 1.

Hue is in the range of 0 to 360 degrees.

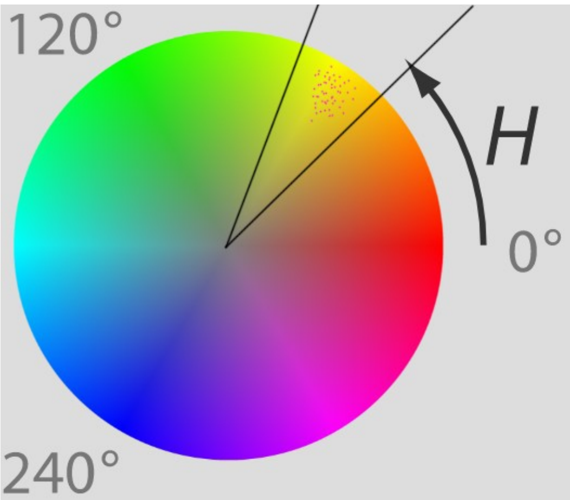


Figure 36: A circle hue chart with the data represented as pink dots. [30]

The pink dots represent the pixel values obtained from the previous step.

Taking the average of this data, the result would land in the degree value that can be easily determined as yellow, by separating the hue circle into sections of colors by degrees ranges.

Yellow is between 72° and 108° degrees on the hue circle, above ~0.5 in saturation, and above ~0.7 in value.

Resistor code value

The color codes can be identified by processing the image using filters and otherwise until only the prominent colors remain. Then, the positions of the color codes relative to the body of the resistor can be used to identify the specific positions and order of the color codes. These can be processed into the actual ohm value.

IC Components

- **Pin count**

The pin count can be identified by processing the image using filters until there is a clear contrast between the body of the chip, and the pins.

One approach that could help identify the number of pins would be drawing a line between two of the pins and seeing how many of the pins touch this line. Taking the line that touches the most pins would provide the pin count of this IC.

- **Information written on the component**

OCR, Optical Character Recognition may be used to extract the text based information. This would require clear visibility, proper alignment, and clear visibility of the text. This may be a challenge under various lighting conditions, as electronic components tend to be tiny, reflective, and the outside casing is prone to wear – scratching off the text.

8. Technology

8.1. *Hardware*

Rig

A mounted camera above a surface (part of the product).

Produces a controlled environment live feed for the application.

Model Training via Deep Learning

Machine used: Personal computer

Personal computer specifications:

CPU: AMD RyzenTM 7 5800X3D

- **Brand:** AMD
- **Name:** Ryzen 7 58700X3D
- **Core count:** 8
- **Thread count:** 16
- **Base clock frequency:** 3.4GHz
- **L3 Cache:** 96MB
- **Maximum operating temperature:** 90°C

System Memory: Corsair [31] Vengeance RGB PRO SL

- **Capacity:** 2×16GB
- **Type:** DDR4
- **Frequency:** 3.6GHz

GPU: GeForce RTX 3060 Ti

- **Memory Capacity:** 8192MB
- **Memory Type:** GDDR6X
- **Base clock frequency:** 1.41GHz
- **CUDA core count: 4864**

Technology Utilised: CUDA

Special cores that are designed for compute-intensive tasks.

These run parallel with the CPU, and may also run parallel with multiple GPUs.

They are perfect for deep learning, as deep learning is incredibly compute intensive, in a way that can be ran in parallel.

Deep learning training times are predictable, and stay mostly constant between epochs.

This means that there are no race conditions, and the more processing power available, the quicker the epoch will finish.

Deep learning computation with CPU Cores and GPU CUDA Cores running in parallel.

8.2. *Software*

8.2.1. Training

Labels

Labelling is an essential part of training for Object Detection.

Labels are bounding boxes that determine what object is present where in the input dataset, in order for the model to do its best to detect them.

Training versus Evaluation

The dataset is separated into two categories – one on which the model is being adjusted on, and another on which the model is being ran to determine the new confidence values achieved by the alterations.

Both datasets must be labelled for fully automated evaluation.

Augmentation

Description

Modification of the provided data to simulate differences in the environment that would occur in a real life scenario, and to provide a challenge in form of imperfections to both train against, and test against.

Augmentation examples

- **Rotation**

Introduction of random rotation, with respect to the label position. Specified in a 0 to 360° range.

Simulates real life scenario of a different angle the feed is provided in.

- **Blurring**

Introduction of random spots of blur. Specified in frequency and intensity.

Simulates real life scenario of change in focus, fog, and smudges on the camera lens.

- **Resizing**

Introduction of random scaling of the image, at a specified frequency and range of scaling.

Simulates real life scenario of distance. Up-close objects will cover a far bigger pixel area in an image than a far away one would, and should be trained against this to prevent detection of only certain distance away objects.

- **Joining up of multiple images to create new ones**

Cutting and joining of images into new images.

Creates additional images from the existing dataset that appear unique, making most of the existing dataset, with only a slight devalue in information stored in regards to training.

- **Addition of glare**

Introduction of random glares, of specified frequency and intensity.

Simulates bright objects interacting with the lens, ensuring the model does not get confused about glare in real life scenarios.

- **Addition of spots**

Introduction of random spots and smudges.

Simulates dust, dirt, and other particles that may be present in a real life scenario, ensuring the model does not get confused by a partial coverage of an object.

Visual examples

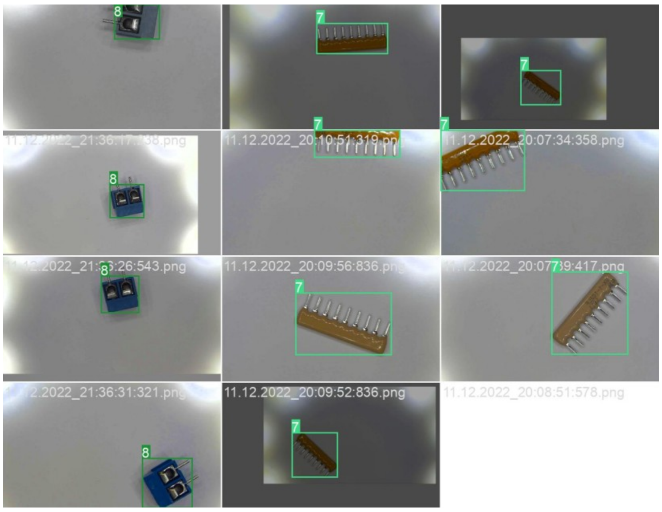


Figure 37: Mild augmentation example 1

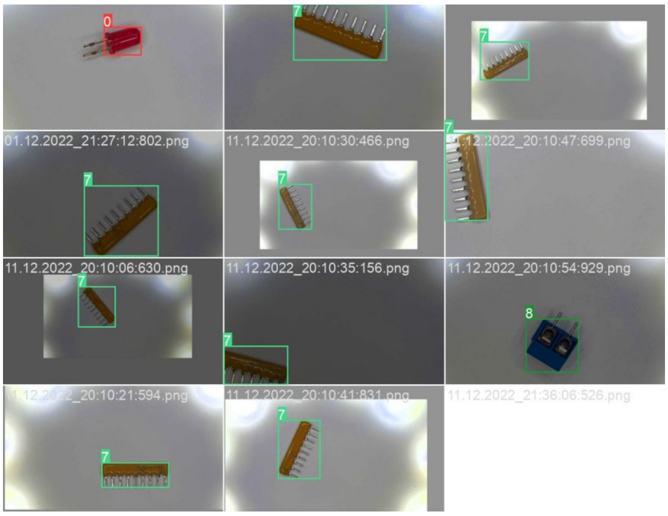


Figure 38: Mild augmentation example 2

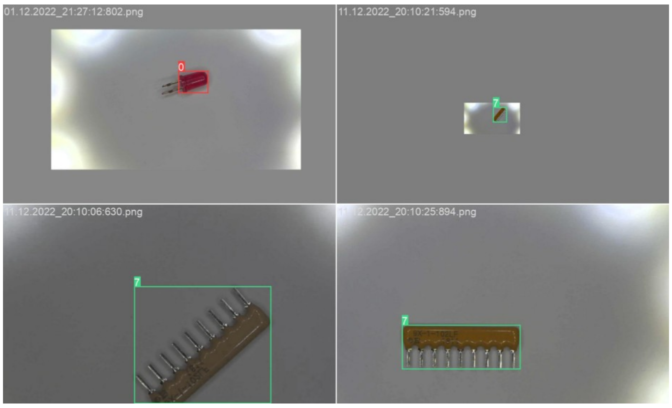


Figure 39: Mild augmentation example 3

8.2.2. Inference

Internal AI Object Detection Steps

1. **Classification**

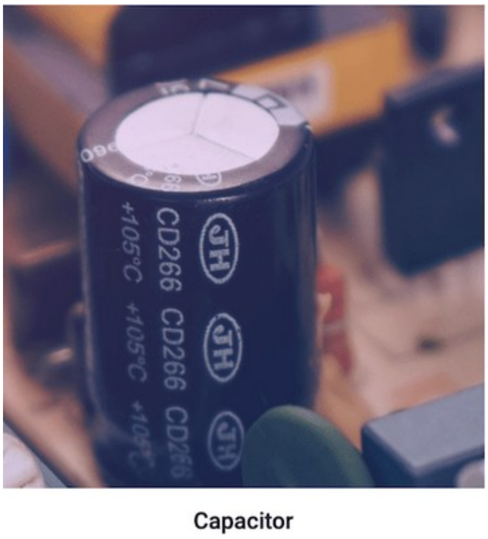


Figure 40: Object Detection: Classification [9]

The identification of a part of an image believed to contain an item of a class the model was trained to detect.

2. **Object Detection**

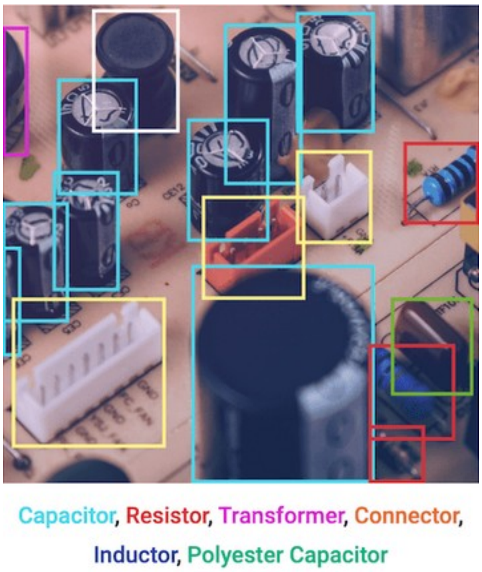


Figure 41: Object Detection: Item Boundary Identification [9]

The Bounding by a box of the classified segments of the image.

3. **Segmentation**

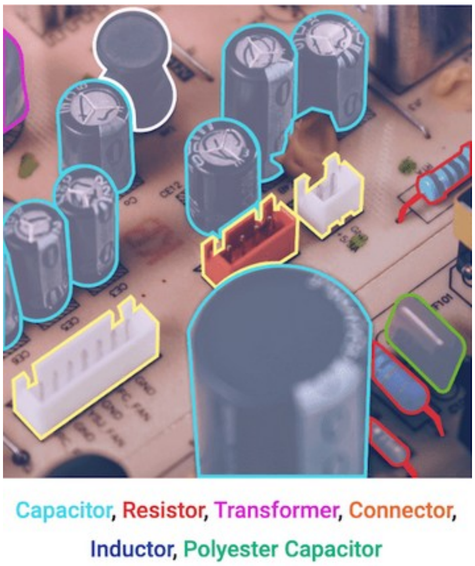


Figure 42: Object Detection: Segmentation [9]

Segmentation is the process of identifying the exact bounding box of the item detected.

8.2.3. Image Manipulation

OpenCV [32] has been chosen to manipulate image/video.

This library contains a vast variety of useful tools such as:

- Reading and writing if image and video,
- Resizing, rotation, and cropping of images,
- A plethora of filter/transformation helper methods.

It offers many different classes. Some of the more common ones that will be seen in this project include:

- **Mat**: a 2D matrix that in our cases represents an image. A Mat contains channels. For greyscale
- **Rect**: An object that keeps track of a certain position – a bounding box. It stores the top left point of the area, together with the width and height. In OpenCV, this class is primarily used to define areas.
- **Vec3b**: A 3 channel pixel value. Stores color information for each pixel inside a Mat.

OpenCV will be heavily used in both post-processing, and display of the frames.

8.2.4. Diagramming Application

EdrawMax [33] (by EdrawSoft [34]**)** was used for several diagrams throughout this documentation.

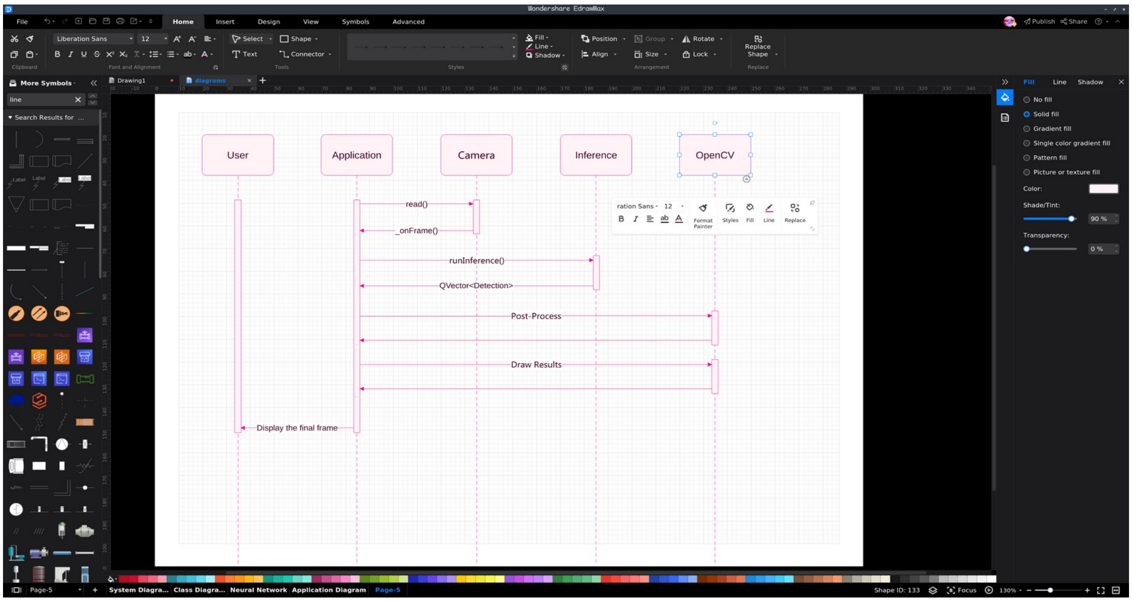


Figure 43: EdrawMax: Sequence diagram creation [33]

The software can be used to create a high variety of diagrams, such as:

- Mind Maps
- Charts
- UML...

EdrawMax also supports a wide variety of options when exporting the created media, including:

- PDF,
- HTML,
- SVG...

This software covers incredibly many fields, with a large collection of templates to choose from.

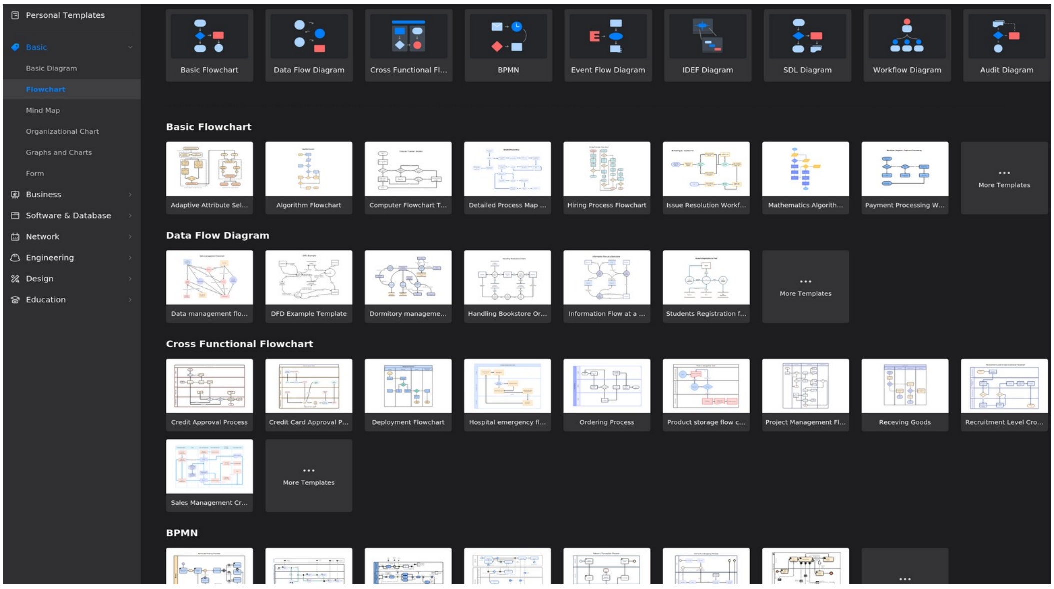


Figure 44: EdrawMax: Flowchart templates [33]

While it is an incredibly powerful diagramming software – it is also very generic. This can sometimes mean it is lacking some specialisation.

8.2.5. Mind Mapping Application

Even though EdrawMax is capable of creating basic mind maps, **EdrawMind [35] (By EdrawSoft [34])** is dedicated specifically for creation of mind maps, and was used to create both the initial project proposal, and this report mind maps. Similarly to EdrawMax, EdrawMind can also export to a variety of different formats. One of which is .odt, which is an OpenOffice document file. This report was first structured using EdrawMind, exported as .odt, and reformatted to the final version.

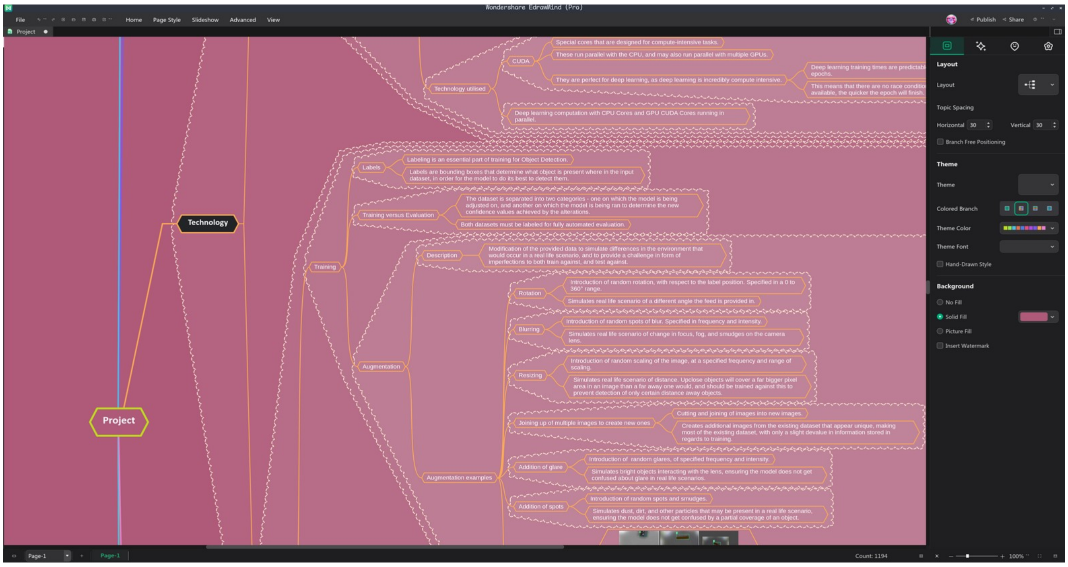


Figure 45: EdrawMind: Project mind map before export [35]

9. Setup

Note:

This setup has been carried out on the previously mentioned Lubuntu 20.04 machine. Steps may slightly vary on other Operating Systems. Throughout the project, the machines presented were not consistent. A personal laptop, and a personal desktop were used. A custom UI theme was used on these machines. Some of the steps may appear slightly different on your machine.

Performance-wise this is insignificant, as both machines are running the same version of Lubuntu 20.04, and both utilise a Nvidia GPU featuring CUDA cores.

Libraries

Working directory

Let `~/opencv/` be our working directory.

Required items

- **CMake**
  - **Description**

A FOSS (Free and open-source software) designed to automate building, testing, packaging, and installing software by using compiler-independent methods.

CMake does not build, rather it generates another system's build files.

The user is able to specify exactly which components they would like to be included, and their properties.

The generated build files are then used to build from.

  - **Version**

`3.16.3-1ubuntu1.20.04.1`

  - **Platform**

Cross-platform

- **Installation**

Linux

```
starlight@starlight:~$ sudo apt install cmake-gui
[sudo] password for starlight:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'cmake-qt-gui' instead of 'cmake-gui'
cmake-qt-gui is already the newest version (3.16.3-1ubuntu1.20.04.1).
```

Figure 46: Terminal command to install cmake-gui

- **OpenCV [32]**

From a terminal in ~/opencv/

git clone https://github.com/opencv/opencv.git

Clone the OpenCV source repository

cd ./opencv

Enter the repository directory

git checkout 4.7.0

Checkout this specific version

- **OpenCV Extra Modules [36]**

From a terminal in ~/opencv/

git clone https://github.com/opencv/opencv_contrib.git

Clone the OpenCV Extra Modules source repository

cd ./opencv_contrib

Enter the repository directory

git checkout 4.7.0

Checkout this specific version

- **CUDA [37]**

Download the appropriate version for your machine

Download Link

https://developer.nvidia.com/cuda-downloads?target_os=Linux&target_arch=x86_64&Distribution=Debian&target_version=11&target_type=deb_local

Extract into ~/opencv/[filename]

This command may be used:

tar -xvf ./cudnn-linux-x86_64-8.6.0.163_cuda11-archive.tar.xz

Note: v argument is optional. V stands for verbose, which indicates to print out every file being extracted.

We also need an empty ~/opencv/build directory for the output of CMake

Finally, the ~/opencv/ directory should have the following content

Figure 47: ~/opencv directory setup for CMake-GUI

OpenCV

Generation of build files using CMake GUI [38]

Open CMake and enter cmake-gui into the terminal.

Then we have to select the source and binaries paths:

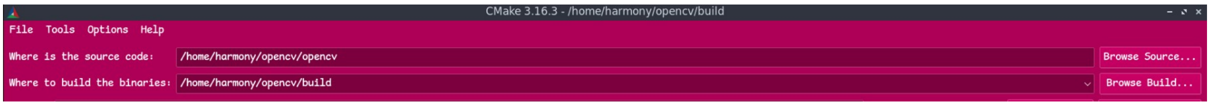


Figure 48: CMake GUI: Source and build binaries directories

Source path should be ~/opencv/opencv

Binaries path should be ~/opencv/build

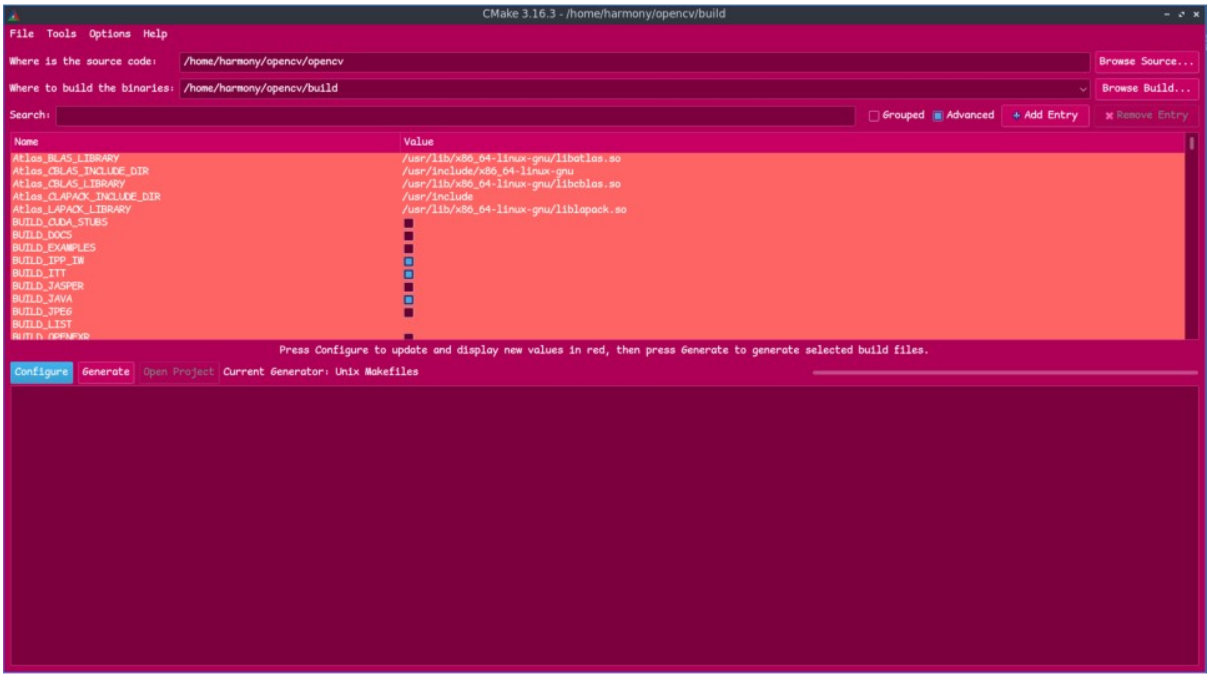


Figure 49: CMake GUI: initial configuration (Configure is highlighted in blue)

Press Configure.

The default options of Unix Makefiles and Use default native compilers will most likely suffice. Choose Finish.

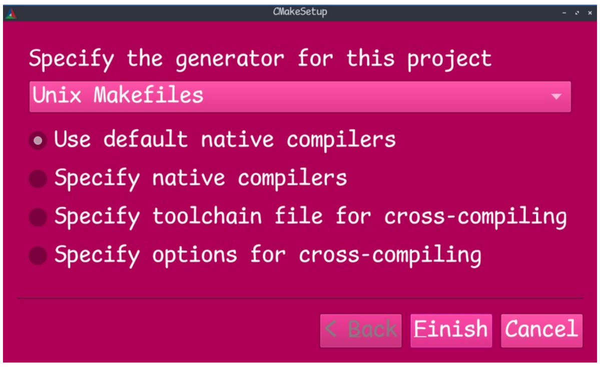


Figure 50: CMake GUI: Generator selection

Flags setup

OpenCV

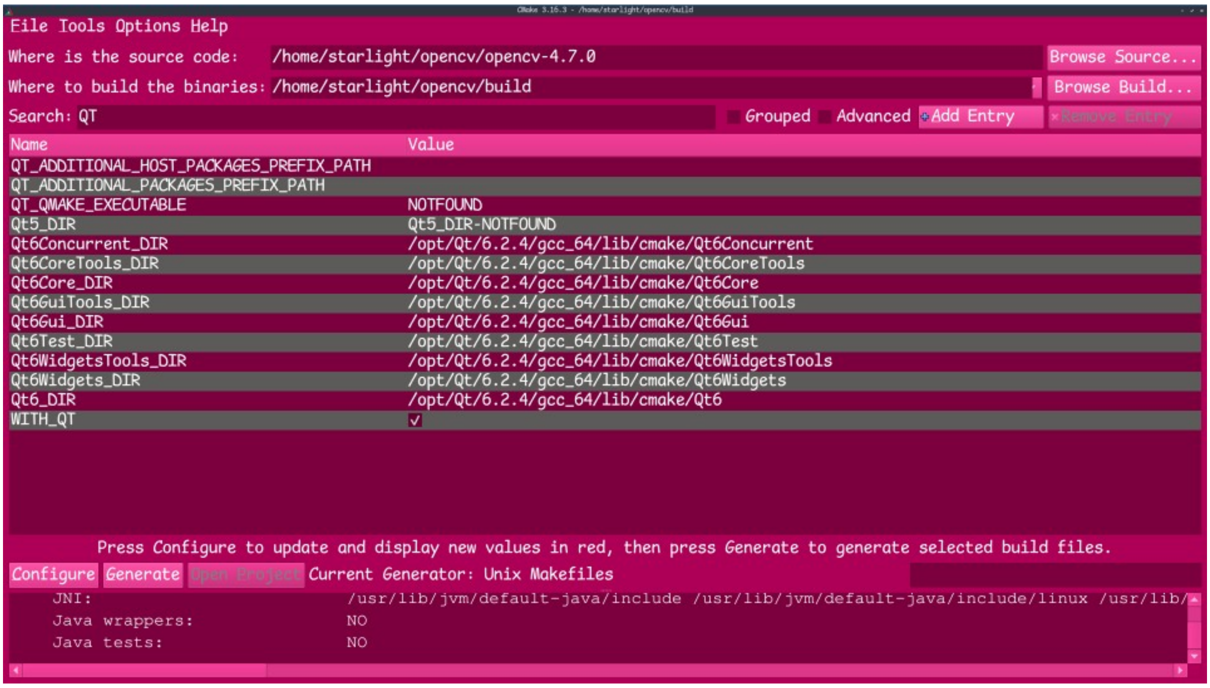


Figure 51: CMake GUI: Qt flags

Flags required for Neural Networks

- **WITH_ONNX:** ✅
- **WITH_INF_ENGINE:** ✅
- **WITH_NGRAPH:** ✅
- **WITH_CUDA:** ✅
- **CUDA_TOOLKIT_ROOT_DIR:** /usr/local/cuda
- **CUDA_TOOLKIT_INCLUDE:** /usr/local/cuda/include
- **CUDA_VERSION:** Make sure expected version is found. 11.4 was chosen for this project.
- **WITH_CUDNN:** ✅
- **CUDNN_LIBRARY:** ~/opencv/cudnn-linux-x86_64-8.6.0.163_cuda11-archive/lib/libcudnn.so
- **CUDNN_INCLUDE_DIR:** ~/opencv/cudnn-linux-x86_64-8.6.0.163_cuda11-archive/include
- **OPENCV_DNN_CUDA:** ✅

Flags required for OpenCV

- **OPENCV_EXTRA_MODULES_PATH:** ~/opencv/opencv_contrib-4.x/modules
- **BUILD_opencv_dnn:** ✅

Flags required for Qt Creator

- **WITH_QT:** ✅
- **Qt6_DIR:** /opt/Qt/6.2.4/gcc_64/lib/cmake/Qt6Qt6*Missing directories (follow pattern of others)
- **WITH_GTK:** ✅
- **WITH_GTK_2_X:** ✅

Extra required flags

- **WITH_FFMPEG:** ✅
- **WITH_PNG:** ✅
- **OPENCV_ENABLE_NONFREE:** ✅
- **CUDA_FAST_MATH:** ✅
- **ENABLE_FAST_MATH:** ✅

If any of the options were not available, simply press configure again and they should appear.

The reason for this is that some flags cause other flags to appear.

For example: WITH_CUDNN is required for CUDNN related settings.

Repeat from the start until all the flags are as specified.

Press **Generate**.

The ~/opencv/build directory will be populated with all the necessary build files.

Build

Navigate to ~/opencv/build from your terminal

Execute the command: make -j $(nproc)

nproc: Returns how many cores the machine has.

The command then becomes make -j 8, if the machine had 8 cores.

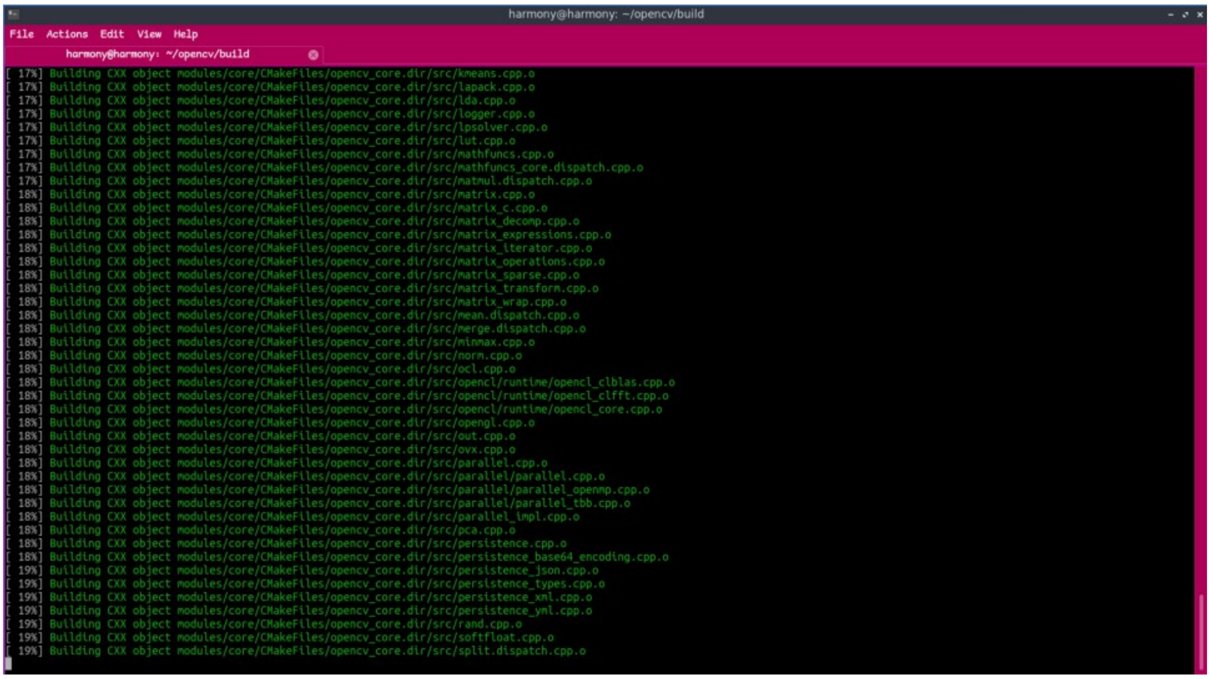


Figure 52: Building using the generated by CMake files, with the make command

This process took somewhere around **4 hours**.

The duration of this process is heavily dependent on the specs of the machine.

Installation

From ~/opencv/build, execute the command

sudo make install

```
[ 95%] Built target opencv_perf_optflow
[ 95%] Built target opencv_test_optflow
[ 95%] Built target opencv_stitching
[ 95%] Built target opencv_perf_stitching
[ 95%] Built target opencv_test_stitching
[ 97%] Built target opencv_tracking
[ 97%] Built target opencv_perf_tracking
[ 97%] Built target opencv_test_tracking
[ 97%] Built target opencv_cudaoptflow
[ 97%] Built target opencv_perf_cudaoptflow
[ 97%] Built target opencv_test_cudaoptflow
[ 98%] Built target opencv_stereo
[ 98%] Built target opencv_perf_stereo
[ 98%] Built target opencv_test_stereo
[ 99%] Built target opencv_superres
[ 99%] Built target opencv_perf_superres
[ 99%] Built target opencv_test_superres
[ 99%] Built target opencv_videostab
[ 99%] Built target opencv_test_videostab
[ 99%] Built target opencv_annotation
[ 99%] Built target opencv_visualisation
[ 99%] Built target opencv_interactive-calibration
[ 99%] Built target opencv_version
[100%] Built target opencv_model_diagnostics
Install the project...
-- Install configuration: "Release"
-- Up-to-date: /usr/local/share/licenses/opencv4/ippicv-readme.htm
-- Up-to-date: /usr/local/share/licenses/opencv4/ippicv-EULA.txt
-- Up-to-date: /usr/local/share/licenses/opencv4/ippicv-third-party-programs.txt
-- Up-to-date: /usr/local/share/licenses/opencv4/ippiw-support.txt
-- Up-to-date: /usr/local/share/licenses/opencv4/ippiw-third-party-programs.txt
-- Up-to-date: /usr/local/share/licenses/opencv4/ippiw-EULA.txt
-- Up-to-date: /usr/local/share/licenses/opencv4/opencl-headers-LICENSE.txt
-- Up-to-date: /usr/local/share/licenses/opencv4/ade-LICENSE
-- Up-to-date: /usr/local/include/opencv4/opencv2/cvconfig.h
-- Up-to-date: /usr/local/include/opencv4/opencv2/opencv_modules.hpp
-- Up-to-date: /usr/local/lib/cmake/opencv4/OpenCVModules.cmake
-- Installing: /usr/local/lib/cmake/opencv4/OpenCVModules-release.cmake
-- Up-to-date: /usr/local/lib/cmake/opencv4/OpenCVConfig-version.cmake
-- Up-to-date: /usr/local/lib/cmake/opencv4/OpenCVConfig.cmake
-- Up-to-date: /usr/local/bin/setup_vars_opencv4.sh
-- Up-to-date: /usr/local/share/opencv4/valgrind.supp
-- Up-to-date: /usr/local/share/opencv4/valgrind_3rdparty.supp
-- Up-to-date: /usr/local/share/licenses/opencv4/libjpeg-turbo-README.md
-- Up-to-date: /usr/local/share/licenses/opencv4/libjpeg-turbo-LICENSE.md
```

Figure 53: result of sudo make install command

Finally, OpenCV is now fully installed, and ready to be used.

```
-- Installing: /usr/local/bin/opencv_annotation
-- Set runtime path of "/usr/local/bin/opencv_annotation" to "/usr/local/lib:/usr/local/cuda/lib64:/home/harmony/opencv/cudnn-linux-x86_64-8.6.0.163_cuda11-archive/lib:/home/harmony/
t/6.2.3/gcc_64/lib"
-- Installing: /usr/local/bin/opencv_visualisation
-- Set runtime path of "/usr/local/bin/opencv_visualisation" to "/usr/local/lib:/usr/local/cuda/lib64:/home/harmony/opencv/cudnn-linux-x86_64-8.6.0.163_cuda11-archive/lib:/home/harmo
y/Qt/6.2.3/gcc_64/lib"
-- Installing: /usr/local/bin/opencv_interactive-calibration
-- Set runtime path of "/usr/local/bin/opencv_interactive-calibration" to "/usr/local/lib:/usr/local/cuda/lib64:/home/harmony/opencv/cudnn-linux-x86_64-8.6.0.163_cuda11-archive/lib:/
ome/harmony/Qt/6.2.3/gcc_64/lib"
-- Installing: /usr/local/bin/opencv_version
-- Set runtime path of "/usr/local/bin/opencv_version" to "/usr/local/lib:/usr/local/cuda/lib64:/home/harmony/opencv/cudnn-linux-x86_64-8.6.0.163_cuda11-archive/lib"
-- Installing: /usr/local/bin/opencv_model_diagnostics
-- Set runtime path of "/usr/local/bin/opencv_model_diagnostics" to "/usr/local/lib:/usr/local/cuda/lib64:/home/harmony/opencv/cudnn-linux-x86_64-8.6.0.163_cuda11-archive/lib"
harmony@harmony:~/opencv/build$
```

Figure 54: OpenCV libraries are now ready to use.

Qt Creator [11]

Installation

Version: 6.2.4

Link: https://www.qt.io/download

The Open Source version was used for this project.

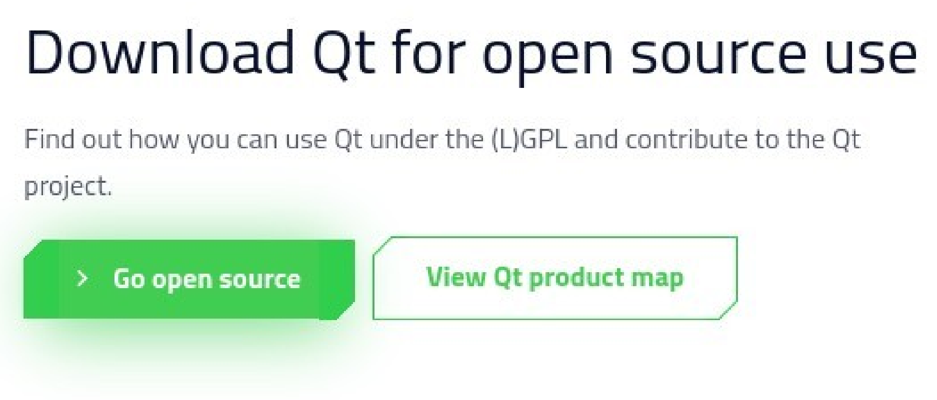


Figure 55: Qt Creator website: Open Source

We want the Qt Online Installer.

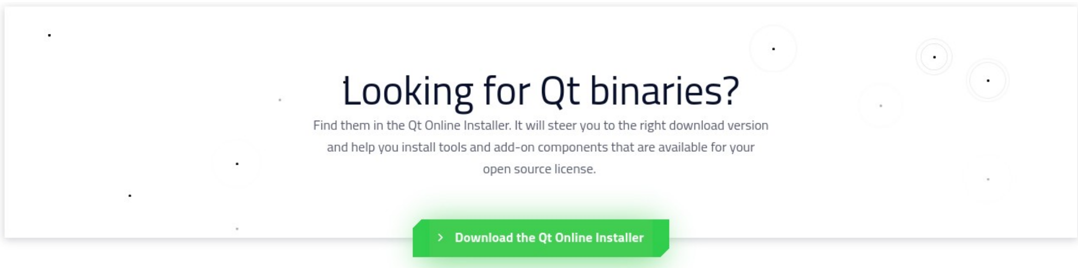


Figure 56: Qt Creator website: Online Installer

The system is running Linux – we need the matching version of the installer.

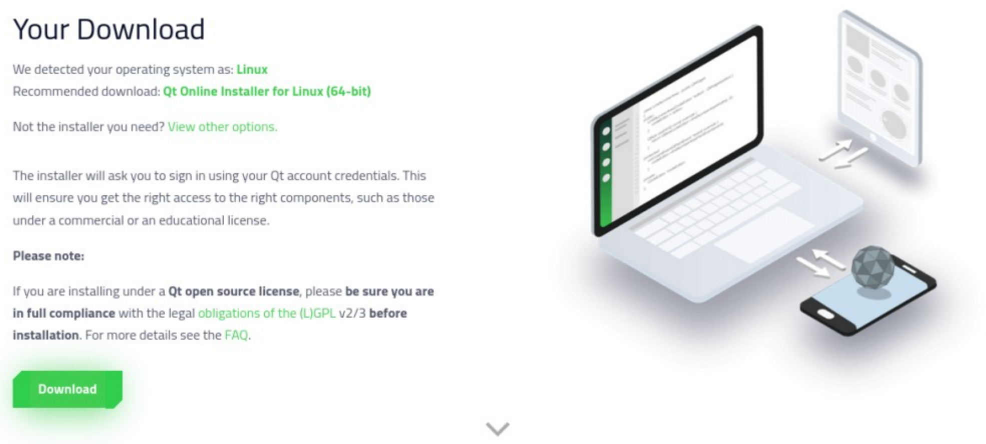


Figure 57: Qt Creator website: Online Installer for Linux 64 bit download page

Figure 58: Qt Creator installer: Version selection

This project was developed in the Qt Creator version 6.2.4.

Project

Note: The design of the project is up to interpretation, and only key parts will be covered briefly.

This project is open source, and the reader may clone the repository to run it from this point on.

This can be achieved by navigating to an appropriate directory to hold the repository, and entering the command:

`git clone https://github.com/Harmonised7/ai_identifier.git`

If you are using SSH instead of HTTPS, the command is instead:

```
git clone git@github.com:Harmonised7/ai_identifier.git
```

The rest of this section will no longer be a tutorial, but an overview of the project internals.

Project Creation

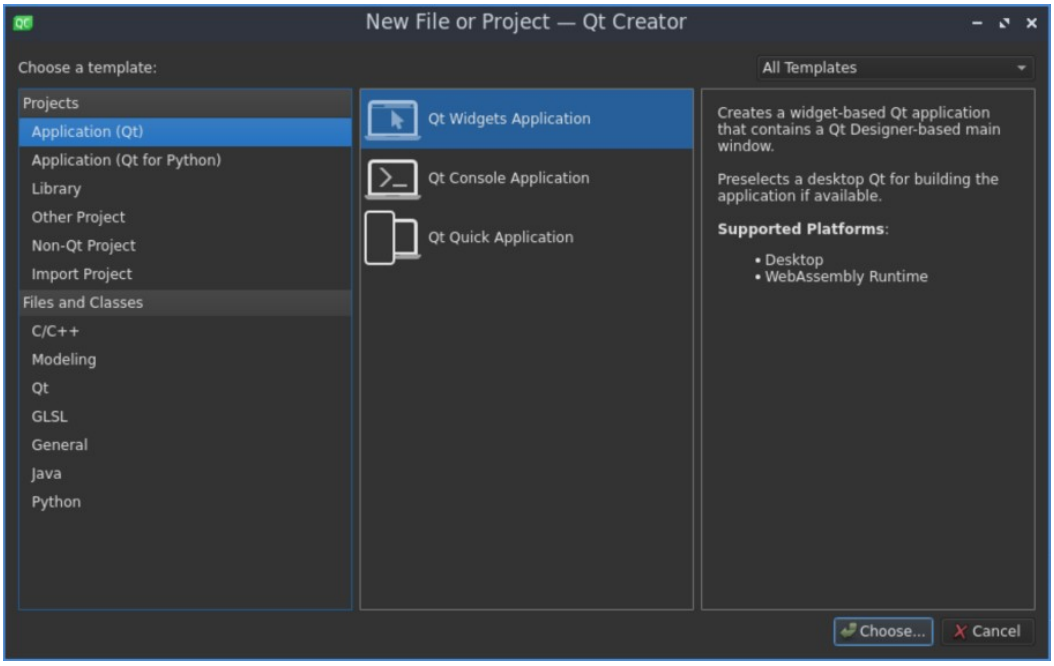


Figure 59: Qt Creator: Type of project selection

Qt Widgets Application is the type of application we are using.

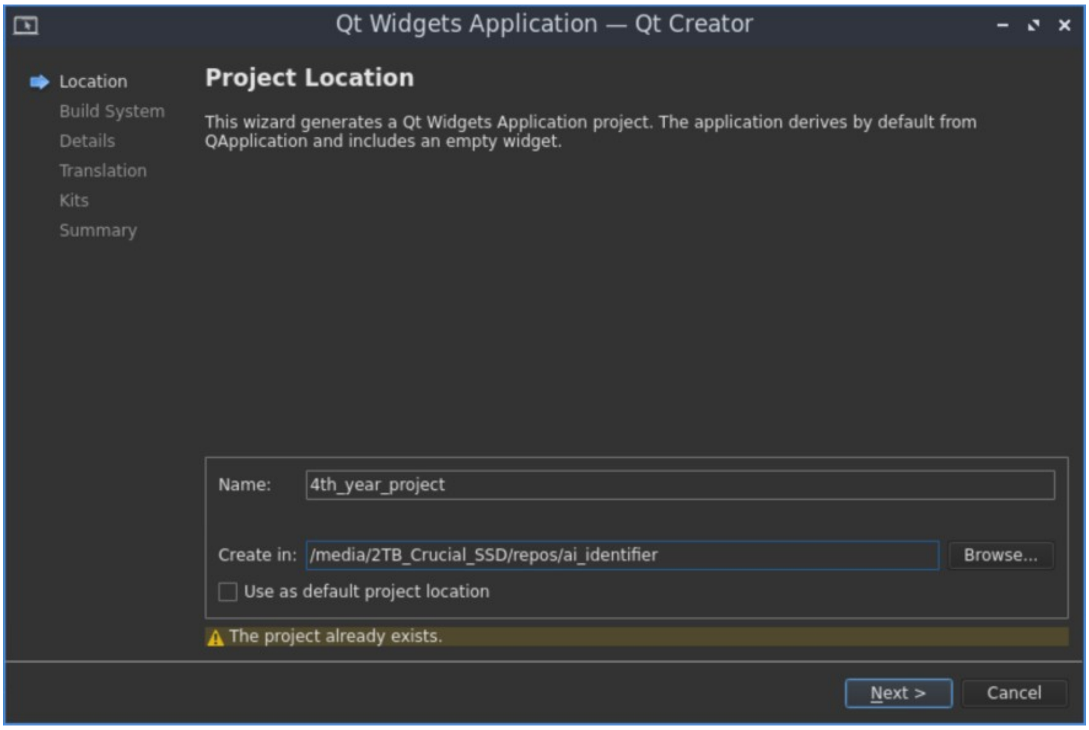


Figure 60: Qt Creator: Project location selection

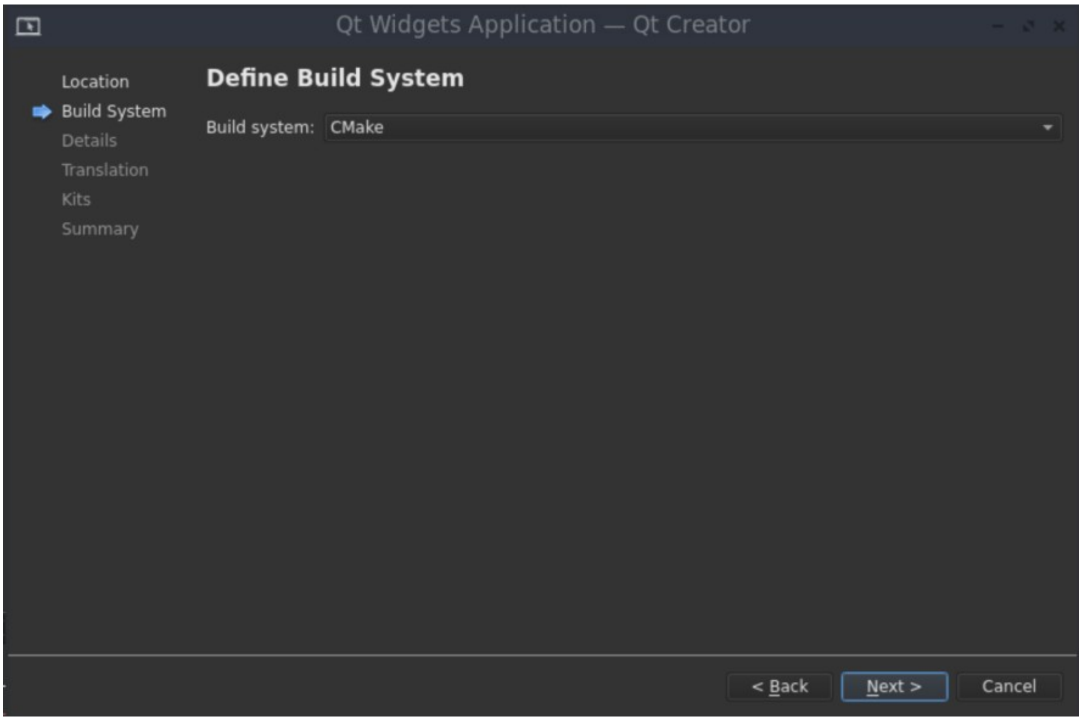


Figure 61: Qt Creator: Build system selection

Build System: CMake

We are using CMake to generate build files once again, this time – the terminal version.

Build Kits

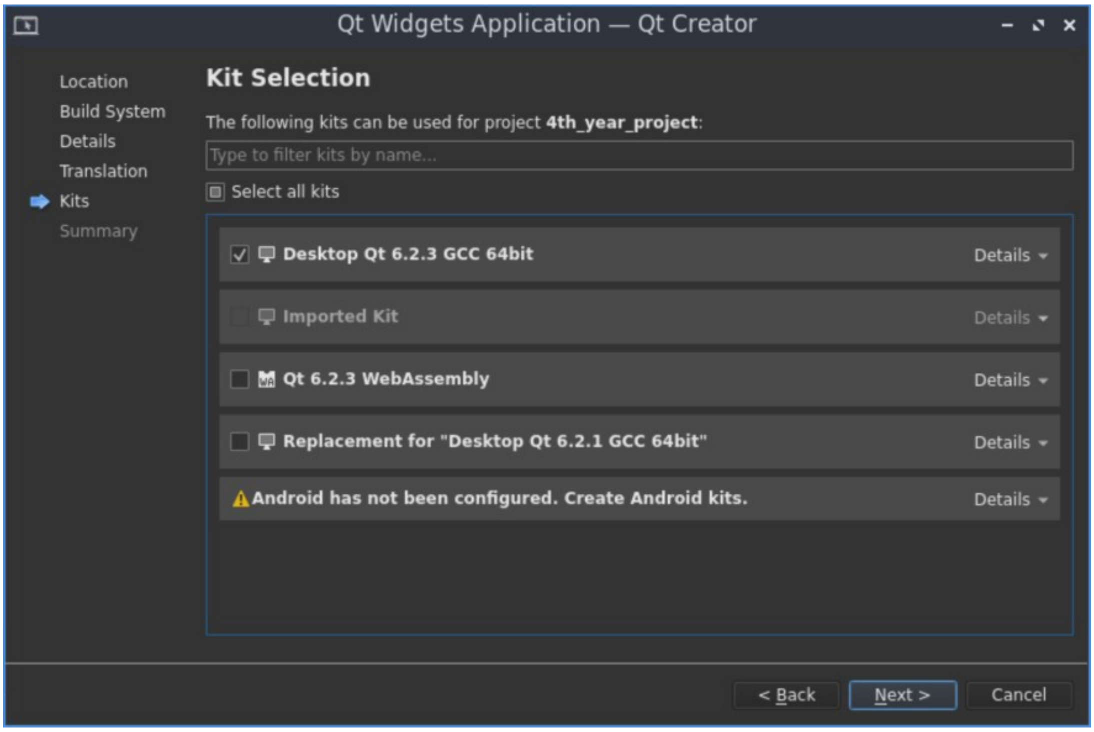


Figure 62: Qt Creator: Build kits selection

For the dev environment and basic setup, we want to use the Desktop kit.

In the future, an Android kit may be necessary to deploy the project on Android devices.

New build kits may be added at any time.

Brief Qt Creator UI design environment introduction

To begin UI editing, first we need to navigate to the .ui extension file. On a new project, this file is called mainwindow.ui.

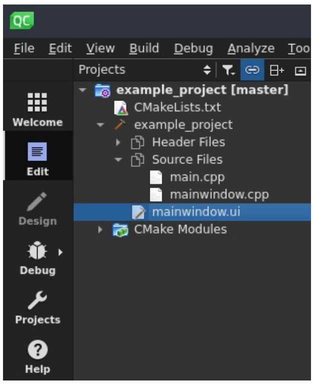


Figure 63: Qt Creator: mainwindow.ui

Opening this file, we are greeted with the interactive UI editor.

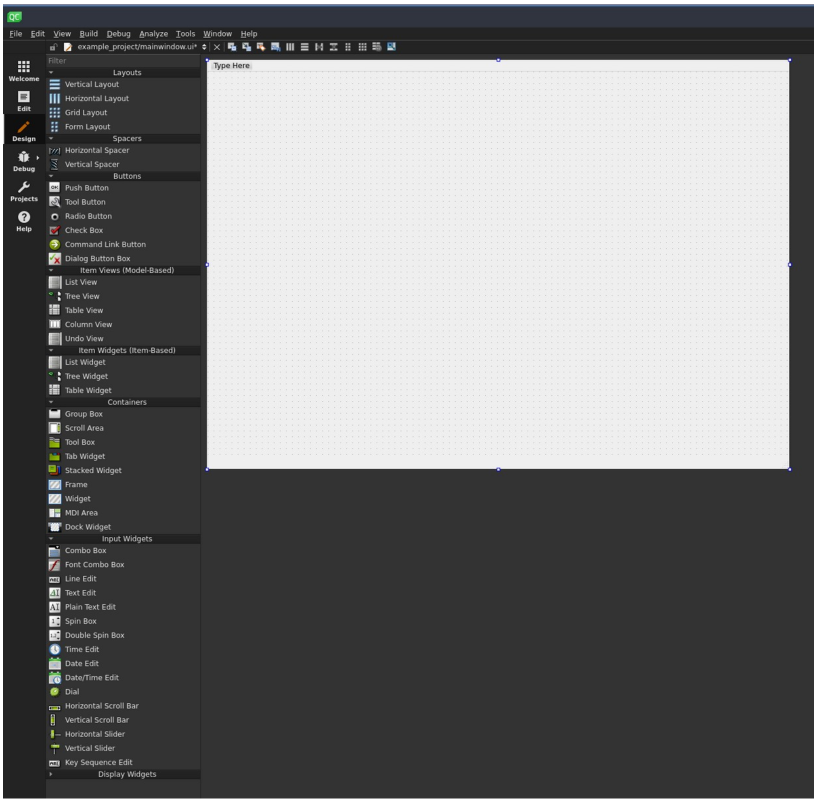


Figure 64: Qt Creator: UI editor

Qt Creator UI editor allows for easy drag and drop of interactive components such as buttons, sliders, layouts, text boxes, etc...

With this basic UI setup example of a Button and a read only QTextEdit box, layed out using Vertical and Horizontal layout.

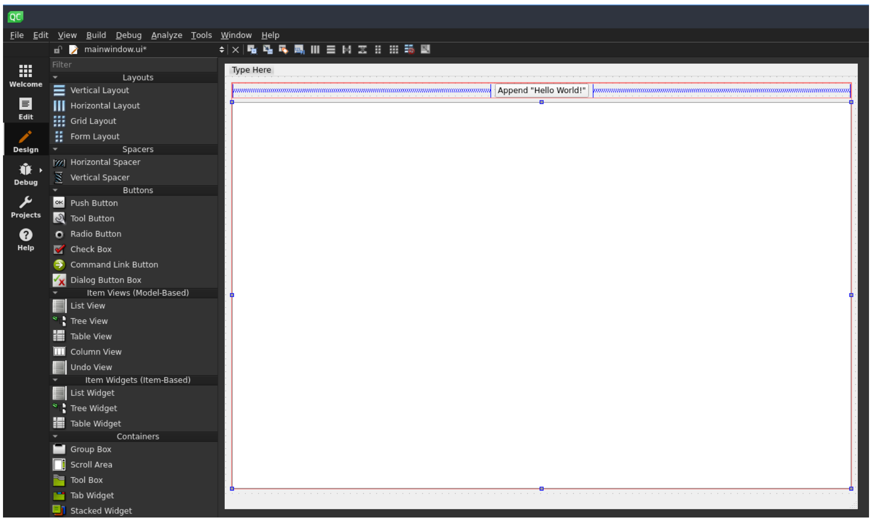


Figure 65: Qt Creator: Basic UI setup example

To edit a UI element, at the bottom right of the interface, we can find the properties panel of the selected element. This is how the button text has been changed to Append "Hello World!"

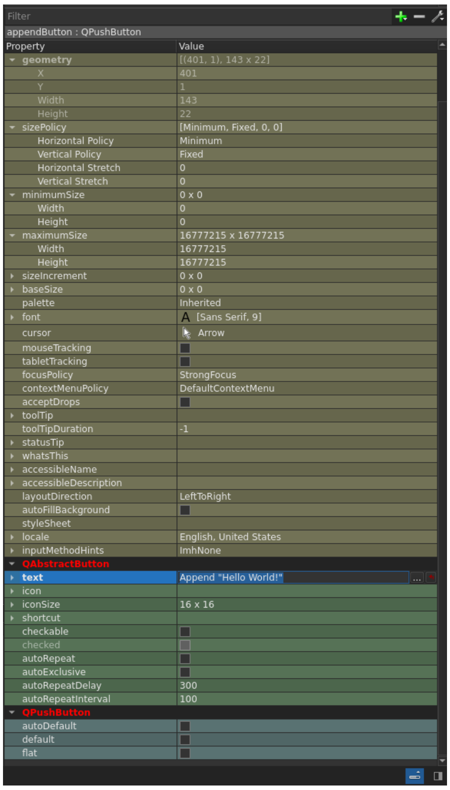


Figure 66: Qt Creator: UI element properties panel

Now we want to setup the button to be clickable. This can be achieved by right clicking the GUI element we want to become interactable, and then choosing Go to slot...

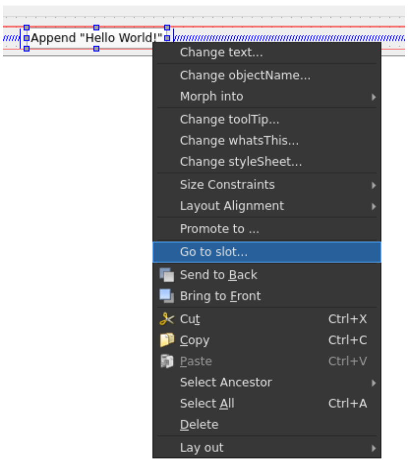


Figure 67: Qt Creator: UI element right click panel

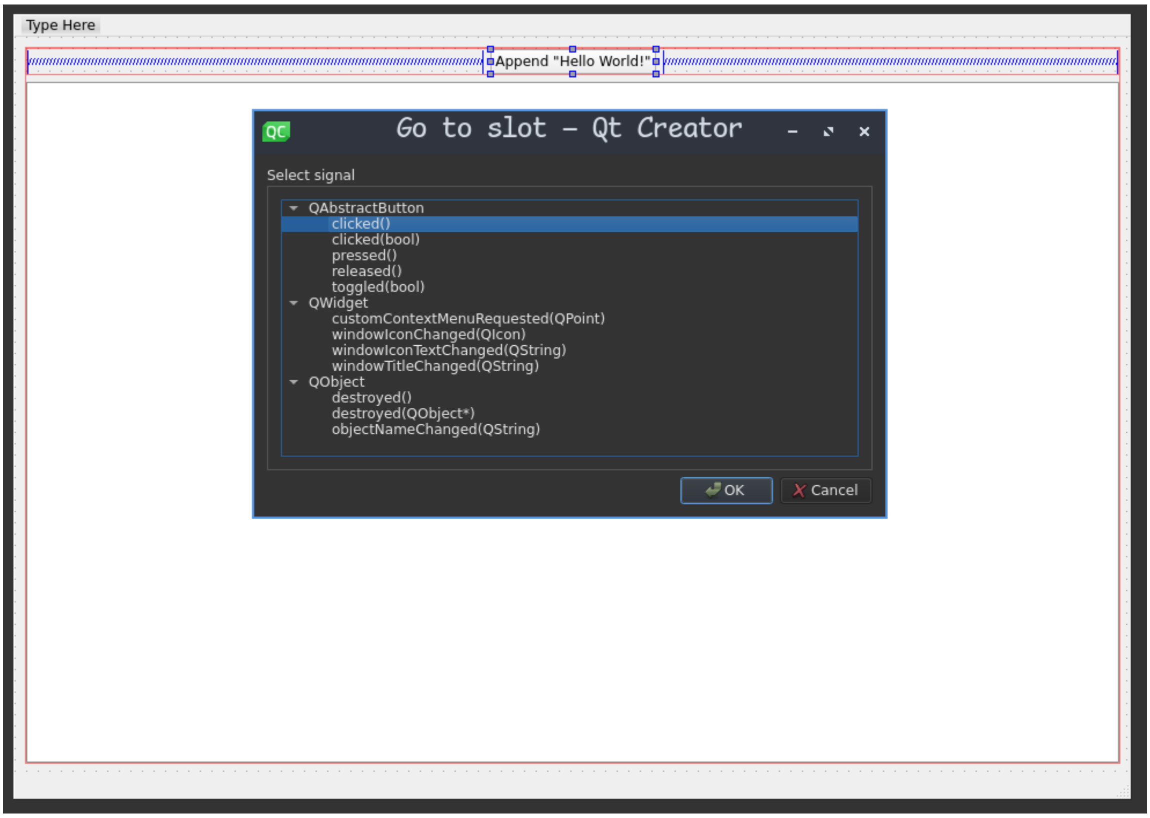


Figure 68: Qt Creator: Go to slot interface

In this interface, we can choose a certain event to go to the declaration of the slot (or create if it does not exist already)

Because this was the first time accessing this slot, Qt Creator filled in some gaps for us. The most relevant ones to us are the changes in mainwindow.h, and mainwindow.cpp.

Slots and Signals are essentially Qt Creator equivalent of Events and Listeners. In our case, the button listens to the event sent by the part of the application that monitors mouse clicks, and redirects it to our slot.

Each of the GUI elements may have slots unique to its class. In the figure above, you can see QObject superclass has its own slots to override, as does QWidget, and QAbstractButton. A Slider for example, would have its own unique slots to hook into.

Overall, slots are a very powerful and intuitive way to keep an application efficient, and easy to implement. The functional example provided above took a total of less than 5 minutes to setup.

In mainwindow.h, a slot was created for the button being clicked.

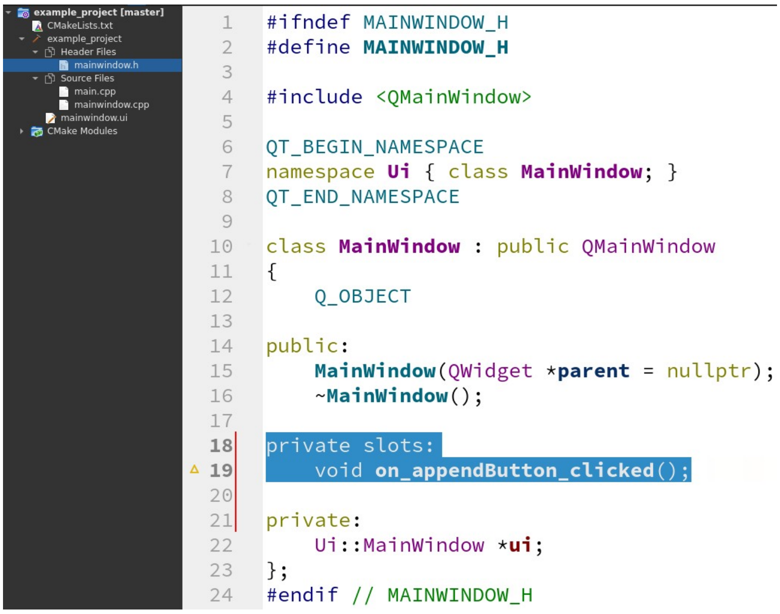


Figure 69: Qt Creator: mainwindow.h generated slot

In the header file of our MainWindow class, the slot declaration that was automatically generated by the Go to slot... interface.

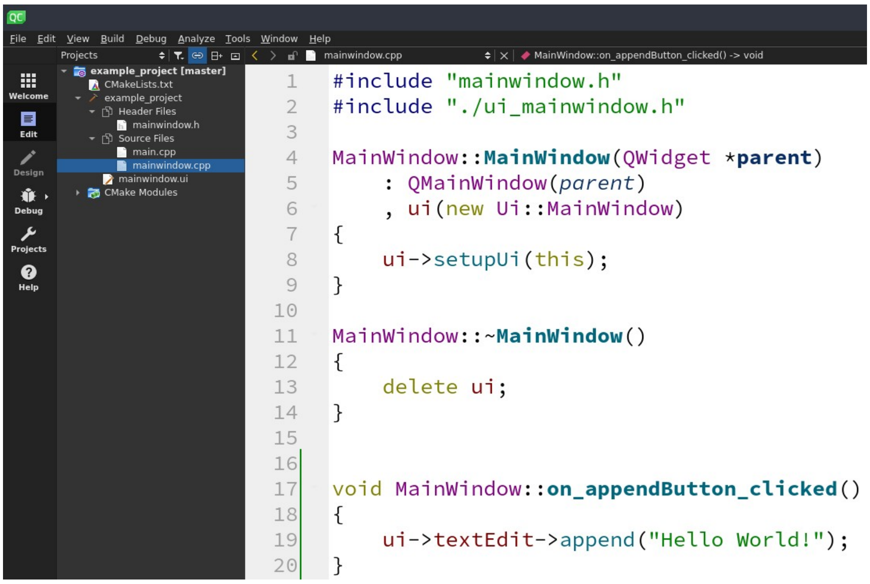


Figure 70: Qt Creator: mainwindow.cpp generated slot method

And also the method implementation was generated in the source file of the MainWindow class. This is the method that gets called when the button recognises that it was clicked.

Extra code was added as an example that references the TextEdit box, which is the read only text box in our UI, and appends “Hello World” to it.

Running the example Application

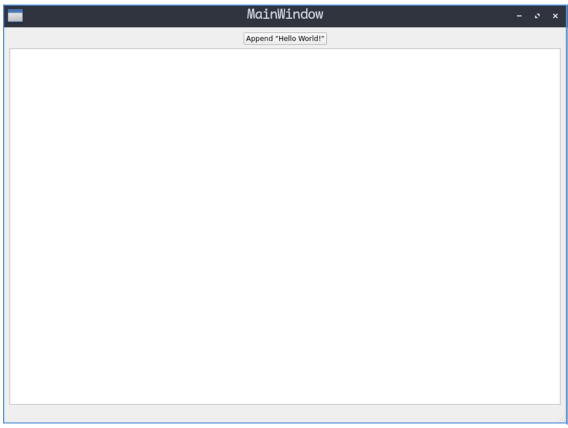


Figure 71: Qt Creator: Example application: Before clicking the append button

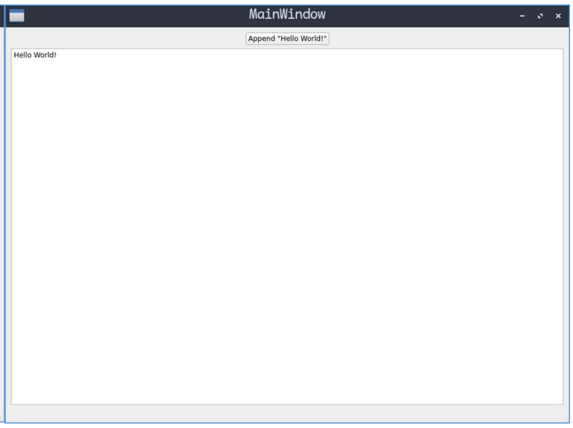


Figure 72: Qt Creator: Example application: After clicking the append button

The above figures illustrate the difference before and after the Append button.

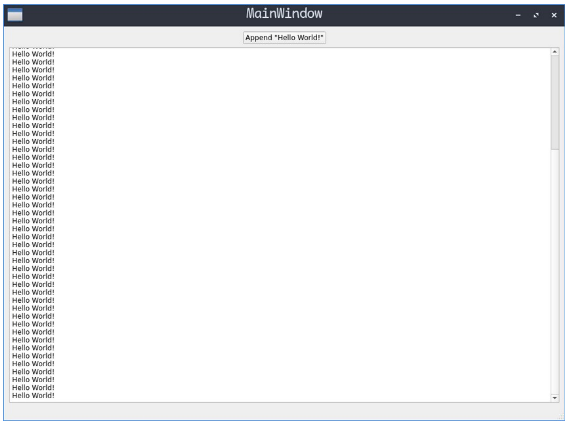


Figure 73: Qt Creator: Example application: After clicking the append button many times

Of course the button may now be clicked indefinitely, or at least until the application runs out of resources.

As seen in the above figure, when the text edit box overflows – a scroll bar becomes available. The conditions of its appearance are fully customisable in the properties panel, as are most of the properties of any of the GUI elements.

Setup

```
# CUDA_START
set(CUDA_TOOLKIT_ROOT_DIR "/usr/local/cuda-11.4")
find_package(CUDA 11.4 REQUIRED)

set(CMAKE_CUDA_STANDARD 11)
set(CMAKE_CUDA_STANDARD_REQUIRED ON)
# CUDA_END

# OPENCV_START
find_package(OpenCV REQUIRED)
include_directories(${OpenCV_INCLUDE_DIRS})
include_directories(4th_year_project PRIVATE "/usr/local/include/opencv4")
target_link_libraries(4th_year_project PRIVATE ${OpenCV_LIBS})
# OPENCV_END
```

Figure 74: Qt Creator: CMakeLists.txt (project file) configuration – including the required libraries

Because we used cmake as our build system, the project file is CMakeLists.txt, located in the project directory. This file is already populated with build instructions for a base project upon first open.

Additional build specifications were added to include CUDA and OpenCV libraries. The CUDA version matches the one compiled during OpenCV installation.

Operation Sequence Diagram

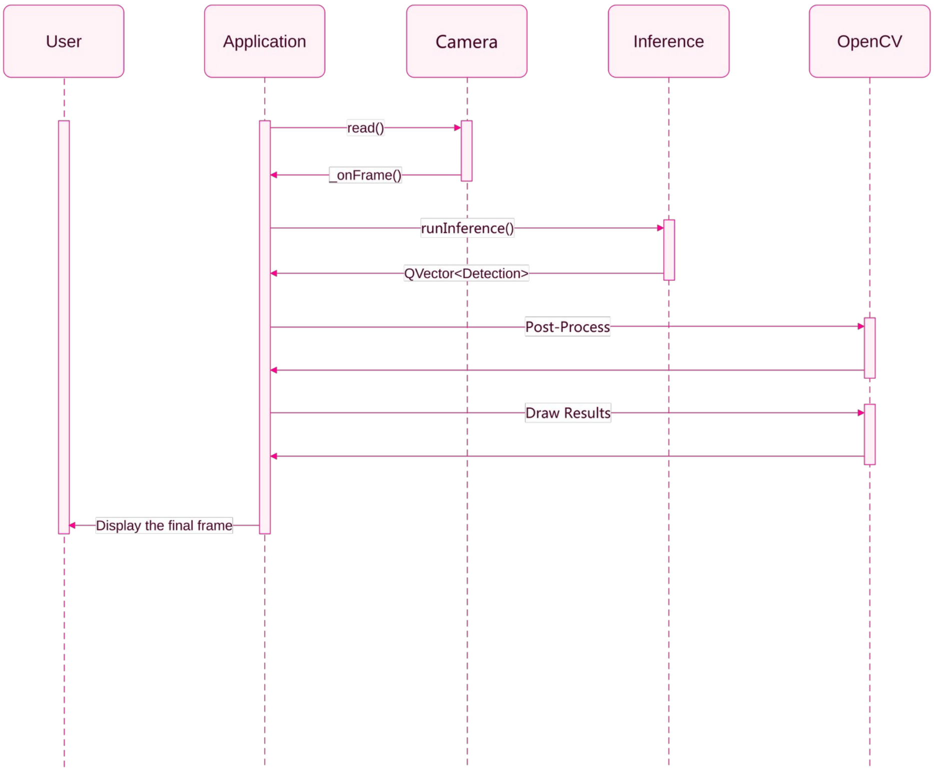


Figure 75: Application Sequence Diagram [33]

Inference is responsible for all the detections, while OpenCV is responsible for all the drawing. Finally, the user receives the processed frame and information.

Systems

- **Threads**

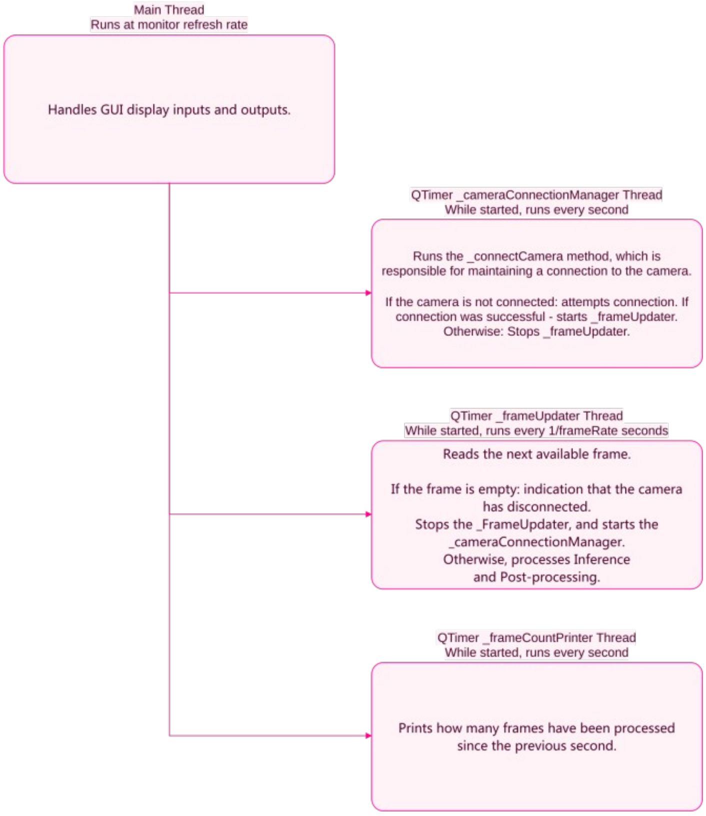


Figure 76: Qt Creator: Simplified overview of the utilised threads.

Note that the frame updating thread utilises both CPU and GPU threads while running inference and post-processing.

- **Camera**

The connection is maintained completely automatically.

The application starts with the connected flag being false. This flag determines if it should try connecting to the camera.

While this flag is false a QTimer will run, which is a separate thread that executes every x amount of time.

This timer is responsible for attempting to connect to the camera until it is connected.

Once the connection is established the connected flag is set to true, and the timer ends.

```
INFO: Attempting connection...
ERROR! Camera not found! Please connect the camera to continue.
INFO: Attempting connection...
INFO: Attempting connection...
INFO: Attempting connection...
INFO: Attempting connection...
INFO: Attempting connection...
INFO: Attempting connection...
INFO: Attempting connection...
INFO: Attempting connection...
INFO: Attempting connection...
INFO: Camera connected
9.2fps
15.2fps
```

Figure 77: Console output from before plugging in the camera through USB, to after: Connection to the camera being established.

To replace the connection timer a frame retrieval timer starts in its place.

- **Frame retrieval**

If we are unable to communicate to the camera this indicates it was disconnected, and the connected flag is set to false again.

A frame is requested from the camera every specified amount of time, depending on the frame rate.

Each frame is sent off to processing while another one is being retrieved asynchronously, thanks to QTimer threads.

- **UI**

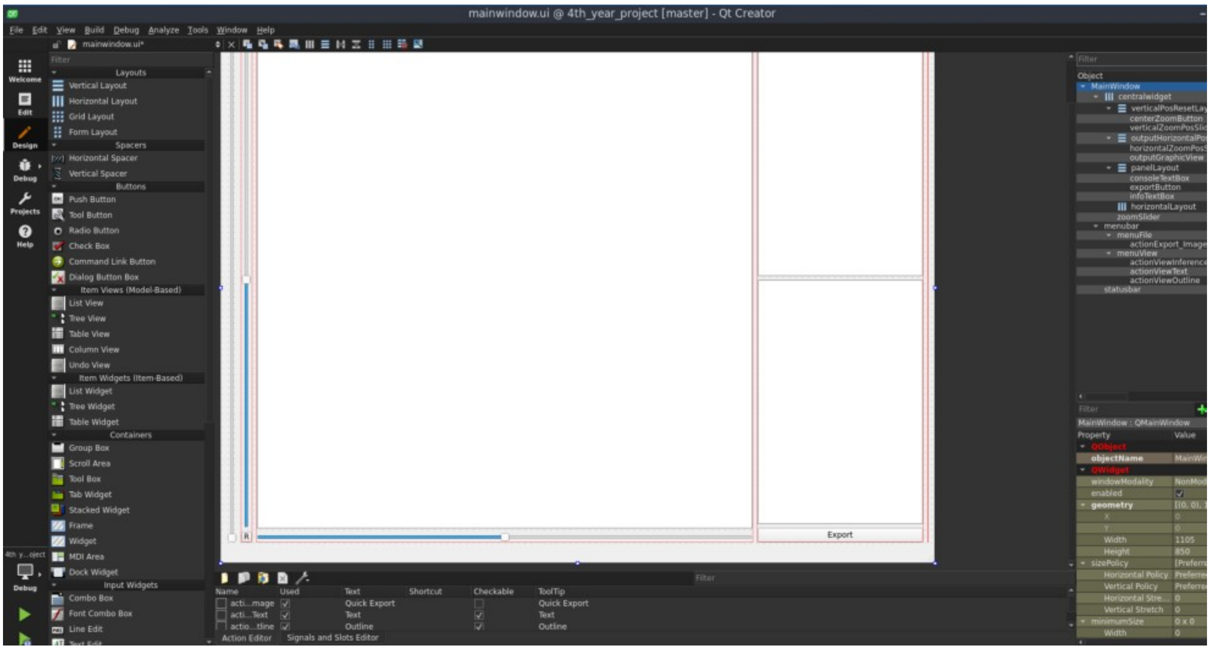


Figure 78: Qt Creator: UI design

Elements

- Live camera feed

- Zoom control

For ease of focusing the camera and inspection of components up-close.

Comes with a reset position to middle button.

Used in precise inspection of the components and camera focus adjustment.

Examples of zoom controls

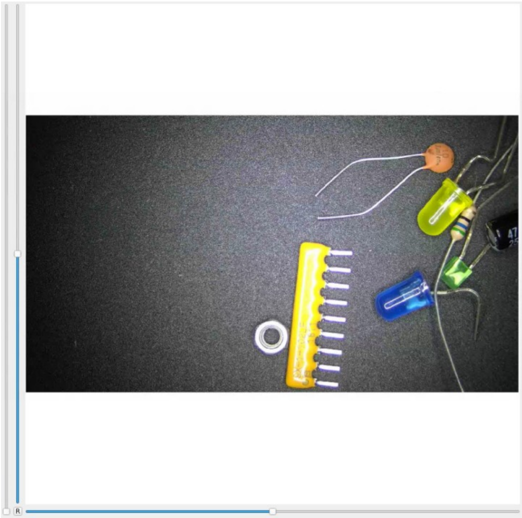


Figure 79: Qt Creator: UI: No zoom

Figure 80: Qt Creator: UI: Zoom example 1: SIP resistor

Figure 81: Qt Creator: UI: Zoom example 2: AC capacitor

Figure 82: Qt Creator: UI: Zoom example 3: LEDs

- **Status bar for configuring what gets ran/displayed**

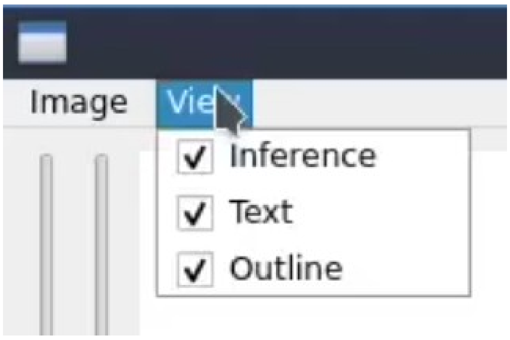


Figure 83: QT Creator: UI: Status bar

  - Inference detections
  - Text display
  - Outline of found components

Export button

Exports the entire frame, both before and after inference

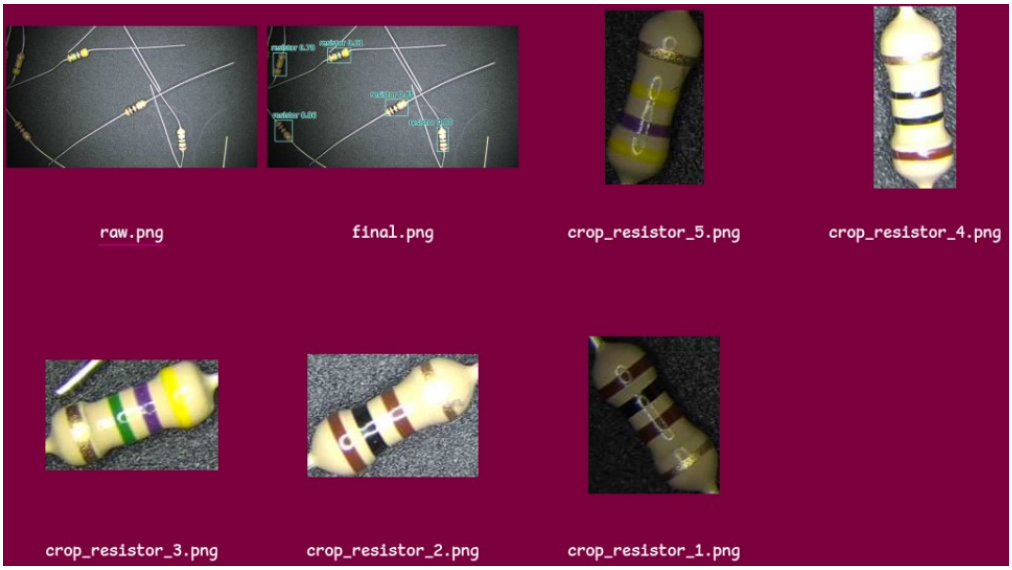


Figure 84: Export button output example

Each detection is exported separately.

This feature will become very useful in Post-processing algorithm development further ahead in this document.

Console and Inference/Post-processing information output boxes

Provide general information about the detections.

Inference

Confidence Threshold: Confidence requirement, under which the detection is rejected.

Score Threshold: Score requirement, under which the detection is rejected.

NMS Threshold: NMS requirement, under which the detection is rejected.

Detection Struct – detection information container

```
struct Detection
{
    int classId{0};
    QString className{};
    QString extra{};
    float confidence{0.0};
    cv::Scalar color{};
    cv::Rect box{};
    cv::Mat mat{};
};
```

Figure 85: Code structure: Detection struct

classId: Internal logic, does not concern the user – used to determine the class name.

ClassName: Name of the class (Resistor/Capacitor/etc...)

extra: Post processing information, appended to the name during display

confidence: How confident the inference is that this detection is indeed this class, and right here. Ranges from 0 to 1, 0% to 100%.

color: Display color of this detection.

box: The location and size of the detection, represented as a box.

mat: A cropped out image from the original, from the position and size of the box.

Post-processing

Saturation adjustment

Utilises Histogram equalisation provided by OpenCV libraries to roughly adjust every detection to a specific saturation value, in order to make post-processing more consistent.

Bulk examples

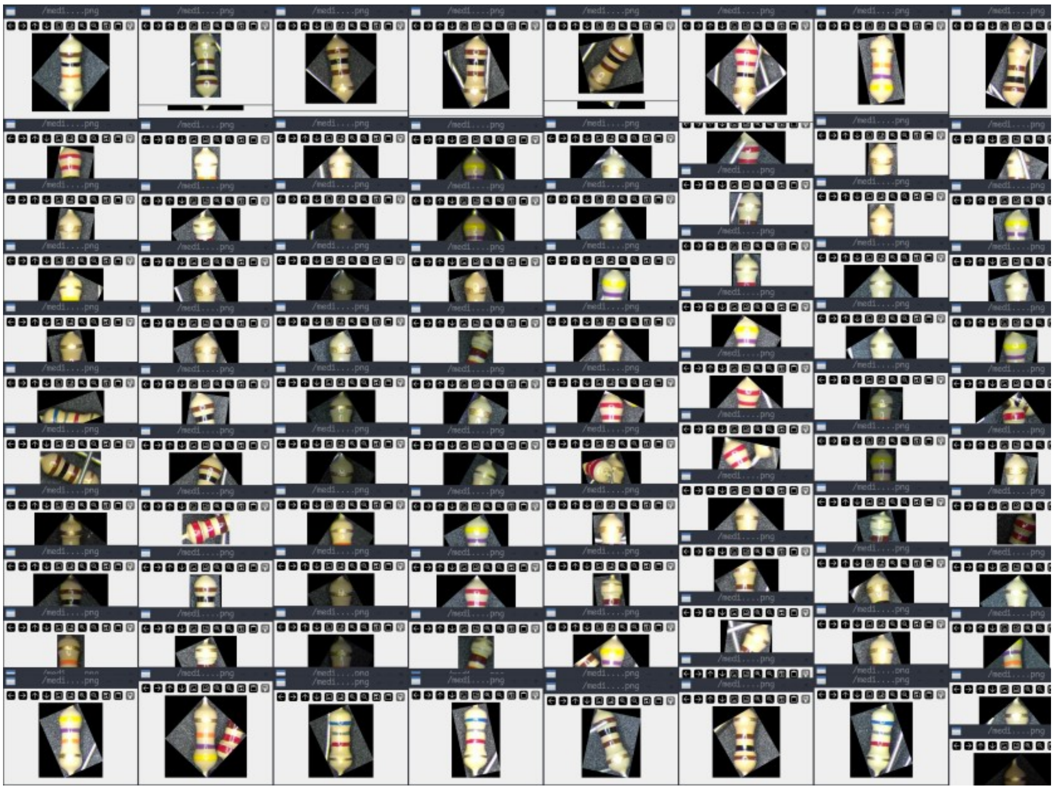


Figure 86: Post-Processing: Saturation adjustment: Before

Figure 87: Post-Processing: Saturation adjustment: After

The brightness of the images now matches the others to a considerable degree.

This provides us with a more consistent workspace for the algorithm to work from.

Orientation

Draws contours around a greyscale version of the component in order to determine orientation.

The images are automatically rotated to face forward.

The extra space that is caused by rotation is filled with pure black pixels, indicating no information.

Resistor Decoding Algorithm

The algorithm was originally designed on a separate work environment, with sliders to control every key parameter.

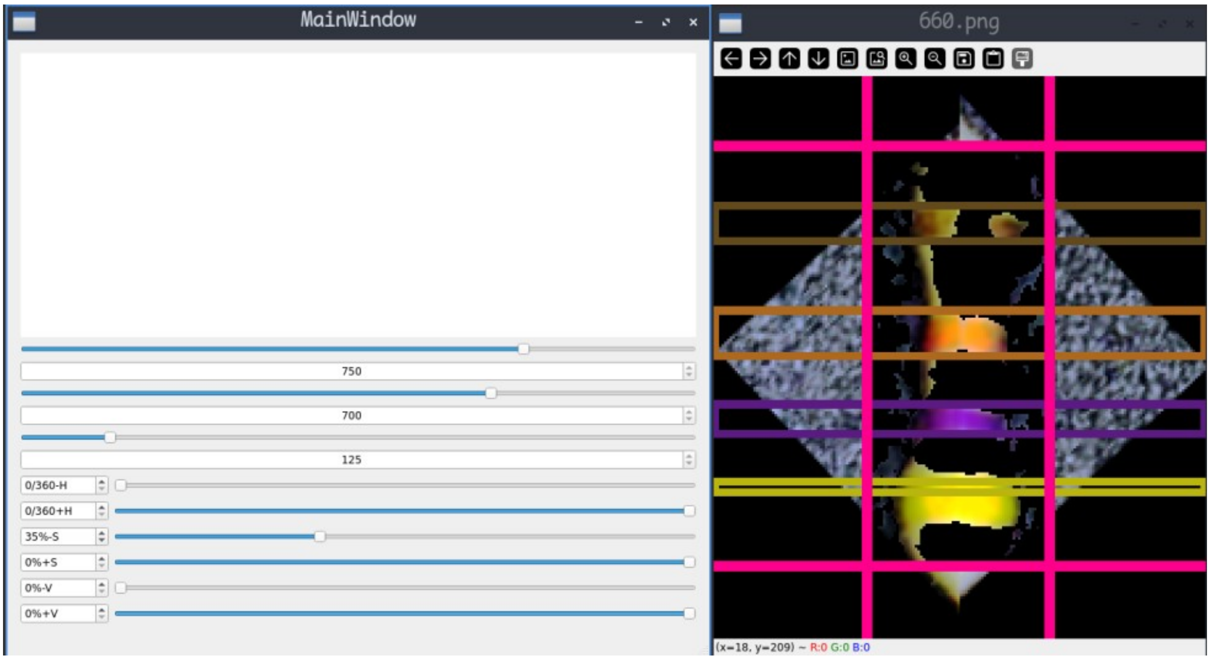


Figure 88: Side project: Post-Processing resistors captured from the rig itself.

Dataset

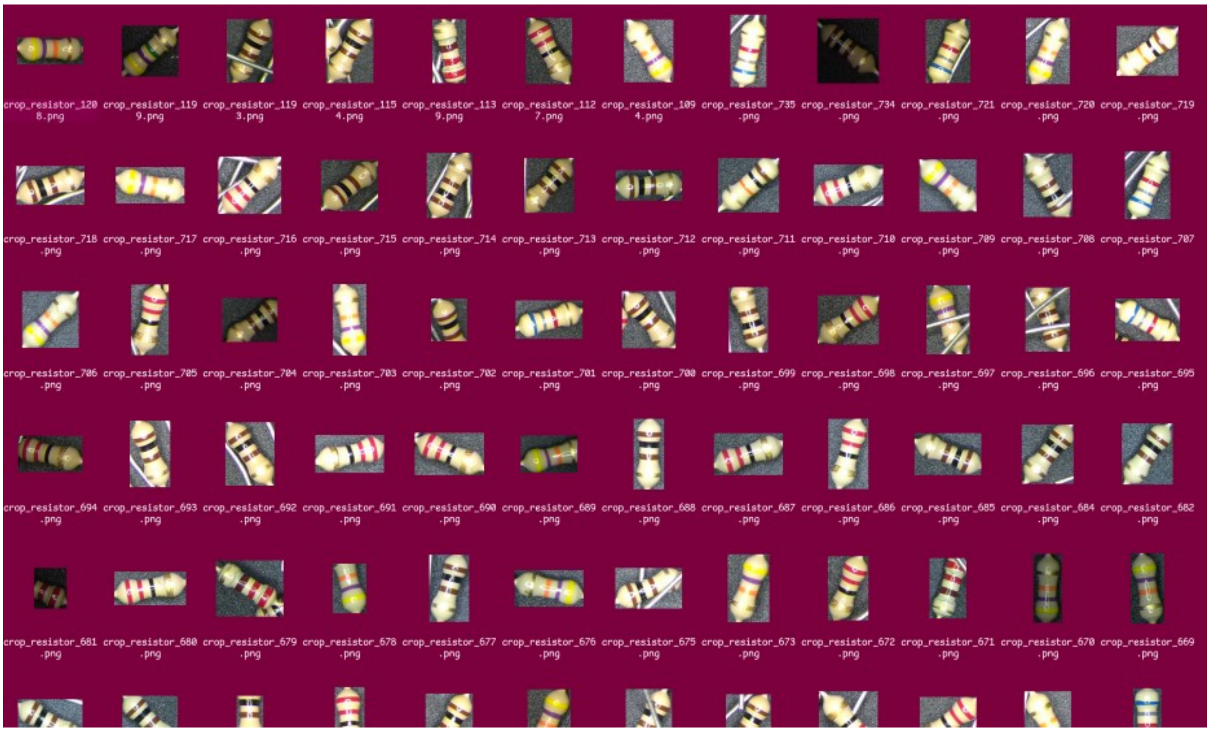


Figure 89: Gathered resistor dataset example

Over 1500 pictures of resistors were gathered from the rig itself for the algorithm to be evaluated against.

```
static QList<QColor> getOhmBands(Mat inputMat, const qreal horizontalBoundPos = 0.7,
                                 const qreal verticalBoundPos = 0.75,
                                 const qreal middleCropBoundPos = 0.125,
                                 const qreal minHue = 0, const qreal maxHue = 1,
                                 const qreal minSat = 0.35, const qreal maxSat = 1,
                                 const qreal minVal = 0, const qreal maxVal = 1,
                                 const bool display = false, QString displayName = "")
```

Figure 90: Post-Processing: The resistor decoding algorithm method declaration

The algorithm was designed on a separate project, and the values that were found to do the job best on the entire 1500 resistor dataset were left as defaults.

Arguments

The cropped out, prepared, and rotated input image of a resistor.

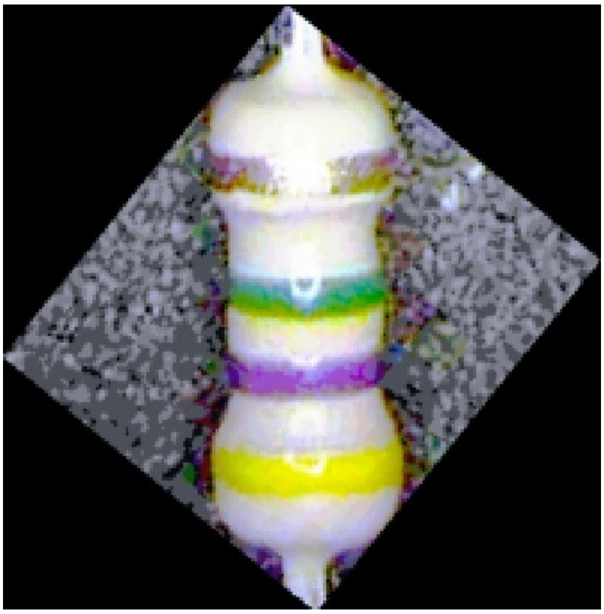


Figure 91: Post-Processing: Resistor algorithm: Input Mat example

qreal horizontalBoundPos: Ranges 0 to 1, Controls the horizontal magenta bounds position

qreal verticalBoundPos: Ranges 0 to 1, Controls the vertical magenta bounds position

qreal middleCropBoundPos: Ranges 0 to 1, Controls how much of the vertical middle is cropped out. (Will be covered in further detail under the Glare section.)

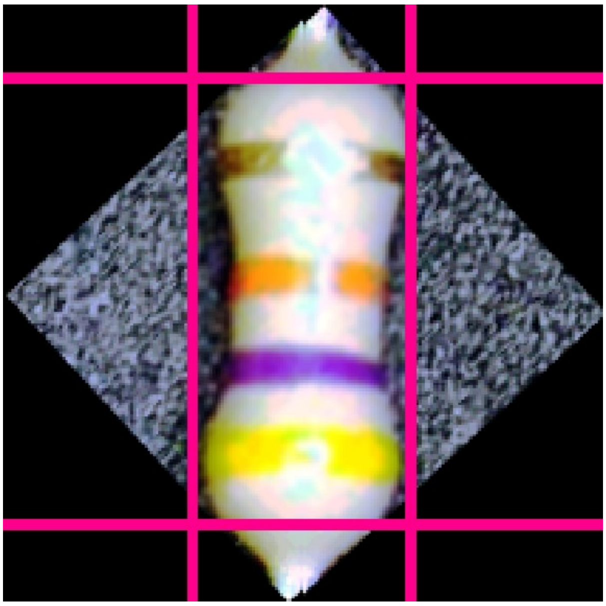


Figure 92: Post-Processing: Resistor decoding: vertical/horizontal bounds

HSV Filter

Minimum/Maximum

Hue: Ranges from 0 to 1

Saturation: Ranges from 0 to 1

Value: Ranges from 0 to 1

Filters pixels inclusively/exclusively (depends if min or max is the higher value), replacing them with 0, 0, 0 RGB (black)

bool display

Debug feature; visualises the internal workings of the algorithm. Used to provide the media in this report.

QString display

The name of the debug display, if it is enabled.

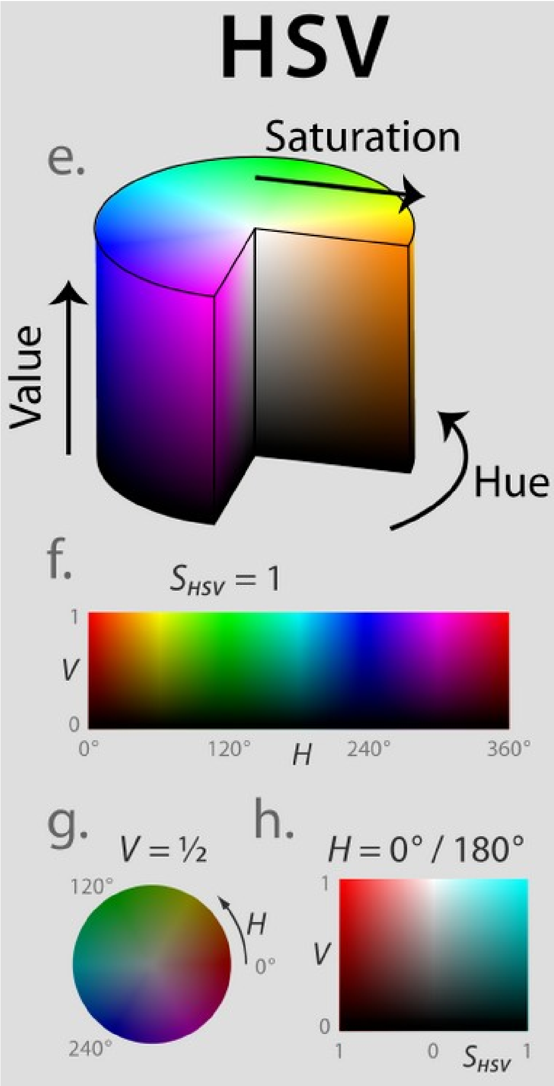


Figure 93: HSV diagram

[30]

Steps

1. Further preparation processing
2. Slight blurring to average out the bands colors

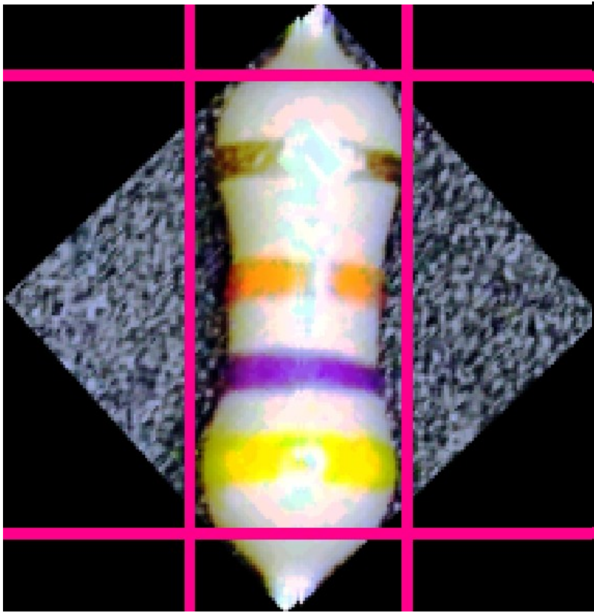


Figure 94: Post-Processing: Resistor: Before blur

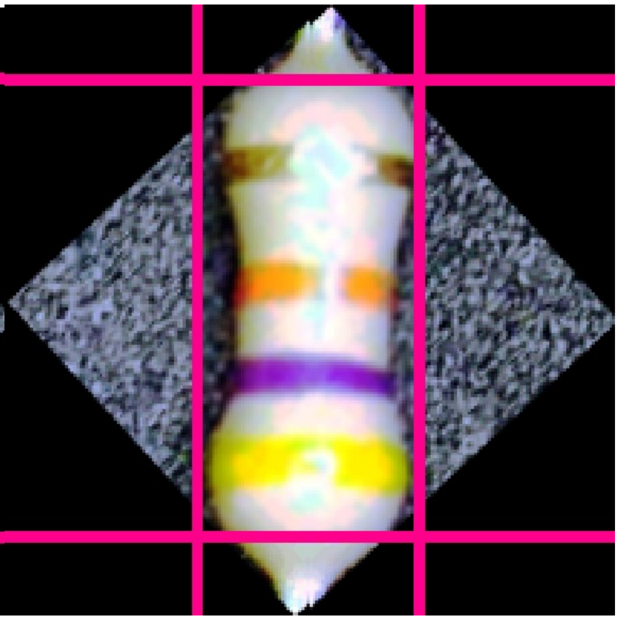


Figure 95: Post-Processing: Resistor: After blur

3. Resistor body and background filtering, using the HSV filter arguments parameters.

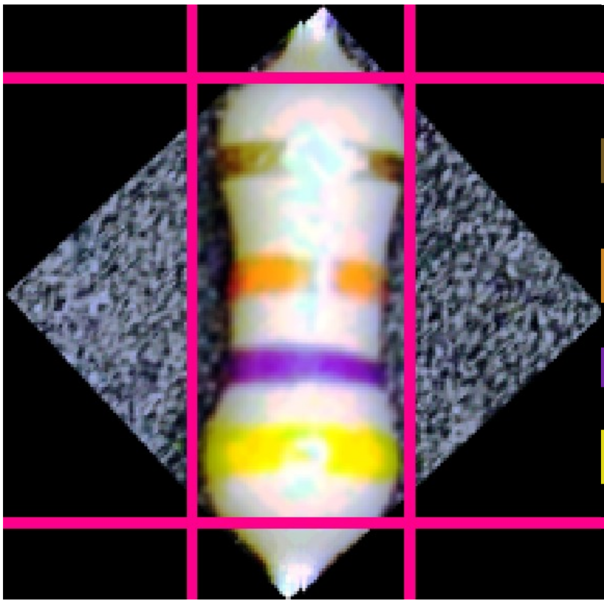


Figure 96: Post-Processing: Resistor: Before HSV filter

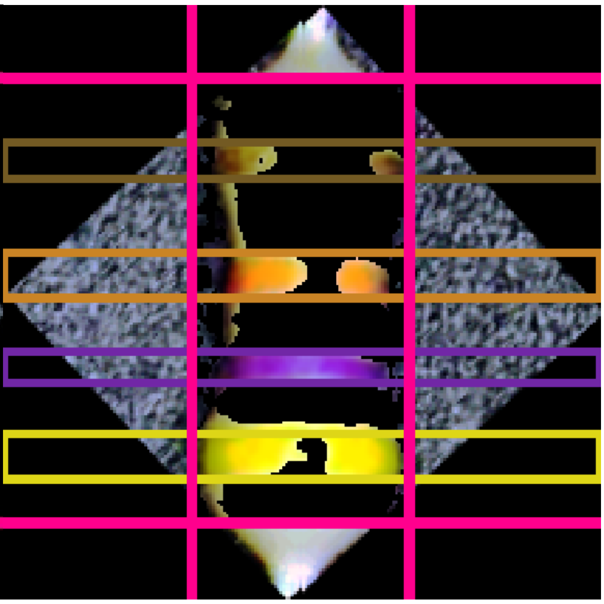


Figure 97: Post-Processing: Resistor: After HSV filter

Glare-destroyed information filtering

Over-exaggerated vertical crop from the middle centre, for demonstration purposes

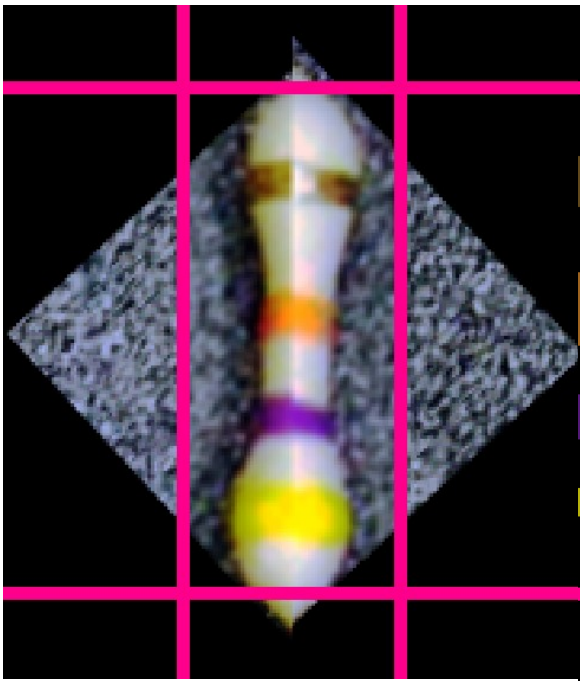


Figure 98: Post-Processing: Resistor: Cropping the middle vertical section (before HSV filter)

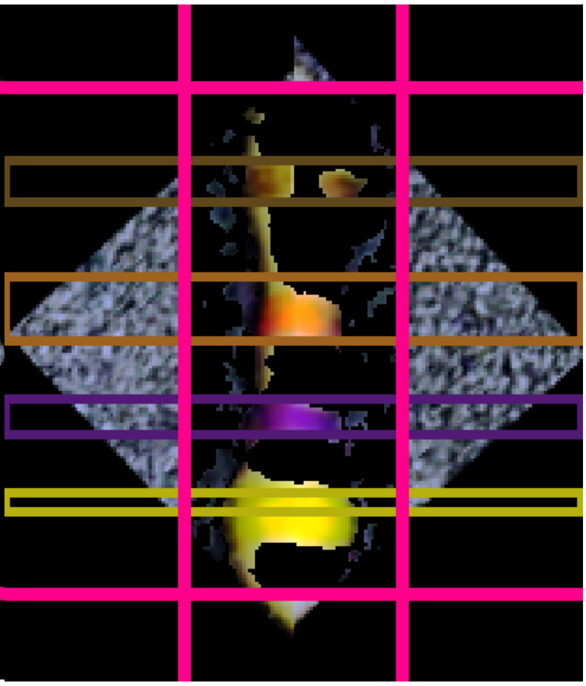


Figure 99: Post-Processing: Resistor: Cropping the middle vertical section (after HSV filter)

Since the information is destroyed by glare there is no reason to process the white strip in the middle.

Cropping out from the vertical middle leaves us with more information in the line horizontally.

The Band Extraction Algorithm

Steps

1. Average of every row's hue and count of non black (empty) pixels are obtained.
2. For every row top to bottom, ratio of non-zero pixels is compared to the width of the row being processed.

- If the count is above a certain threshold, the algorithm assumes this is a color band, and enters band mode.
- Otherwise, it continues to the next row, until the condition is met, or no more rows remain.

3. Once in band mode, the algorithm starts to keep track of the average hue of the band.

- This is what determines the final color of the band.

4. Once the algorithm detects an insufficient ratio of color in the row, band mode ends.

- If the tracked band height was more than a certain small threshold (Aims to prevent noise counting as bands), it pushes the average hue onto the Bands List, and advances a certain step size forward since there will be no band immediately after a band.

5. The steps loop, until no more rows remain.

Color determination

The final band colors identifier names are determined from a combination of the Hue, Saturation, and Value of the colors.

For example, red/brown can only be within the ~330-360 and ~0-30 range.

This can be further processed by analysing the saturation and value of the color to determine if this was a red color or brown color.

Optimisation (Boundaries)

Thanks to the set shape of a resistor, the algorithm is able to narrow down on which area the bands are present in.

This means there is no need to process outside of the predicted magenta bounds.

This reduces the processing power required to an average of 1/4th (difference between the entire image area, and the insides of the magenta bounds.)

Example

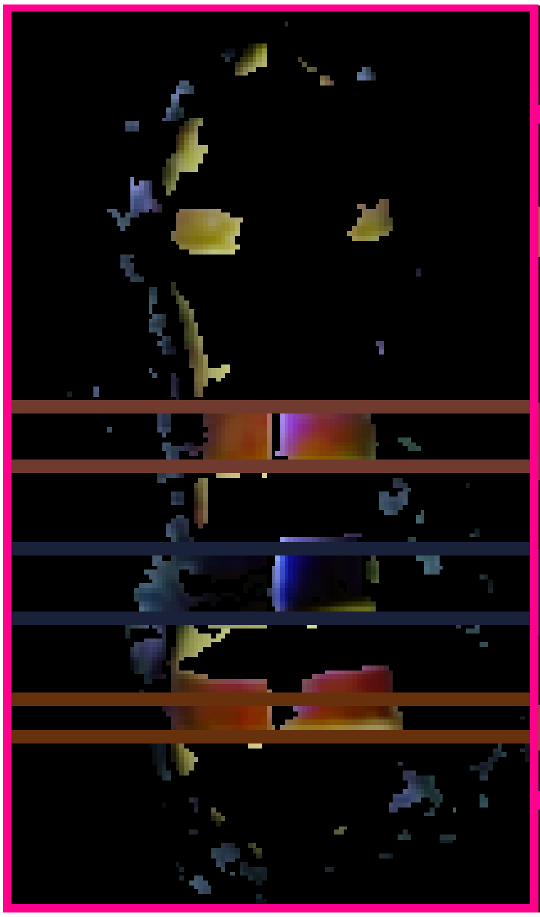


Figure 100: Post-Processing: Resistor: Boundaries of interest: Before boundary application

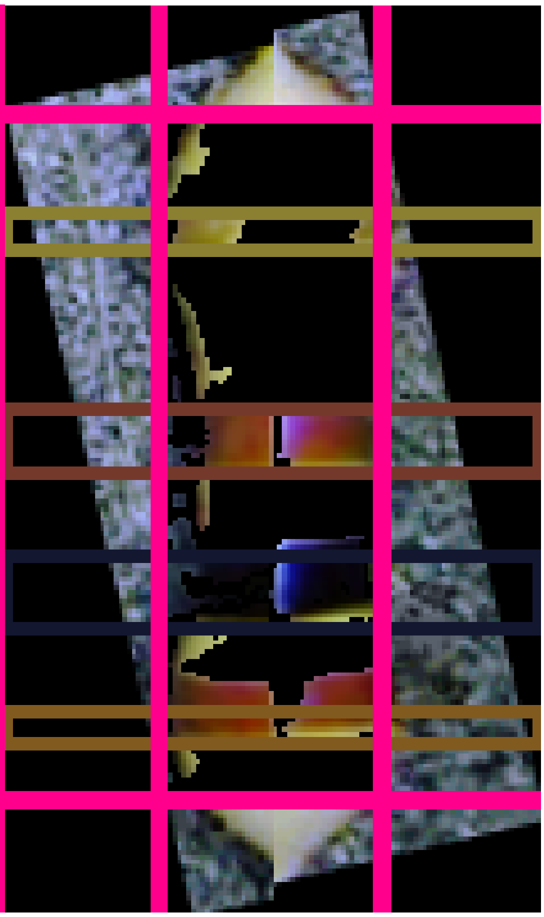


Figure 101: Post-Processing: Resistor: Boundaries of interest: After boundary application

Notice the lack of processing outside of the pink bounds.

Note: The algorithm is calibrated to the optimised version. It expects a specific ratio between color and blank pixels in the entire row that is being tested.

This is the reason the gold band was not found in the unoptimised version.

Early Bulk Testing

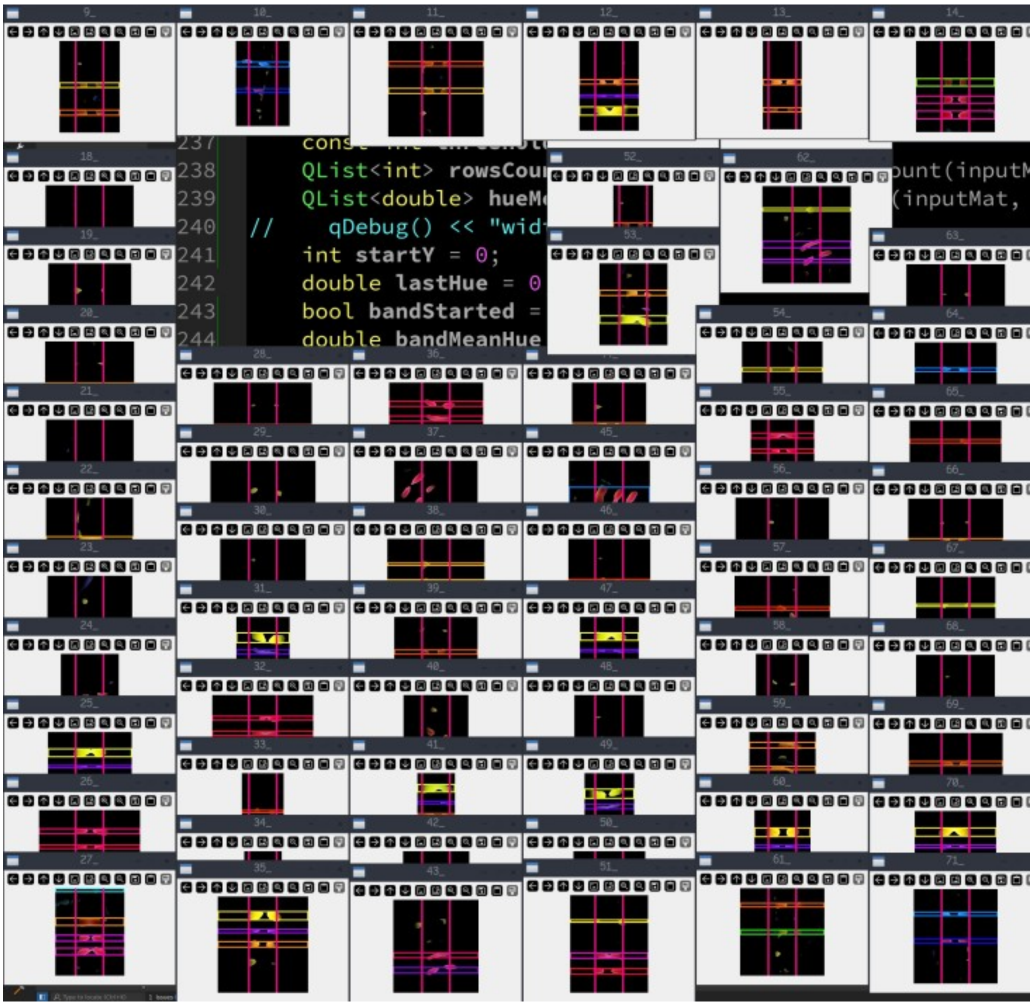


Figure 102: Post-Processing: Resistor: Bulk testing and calibration example 1

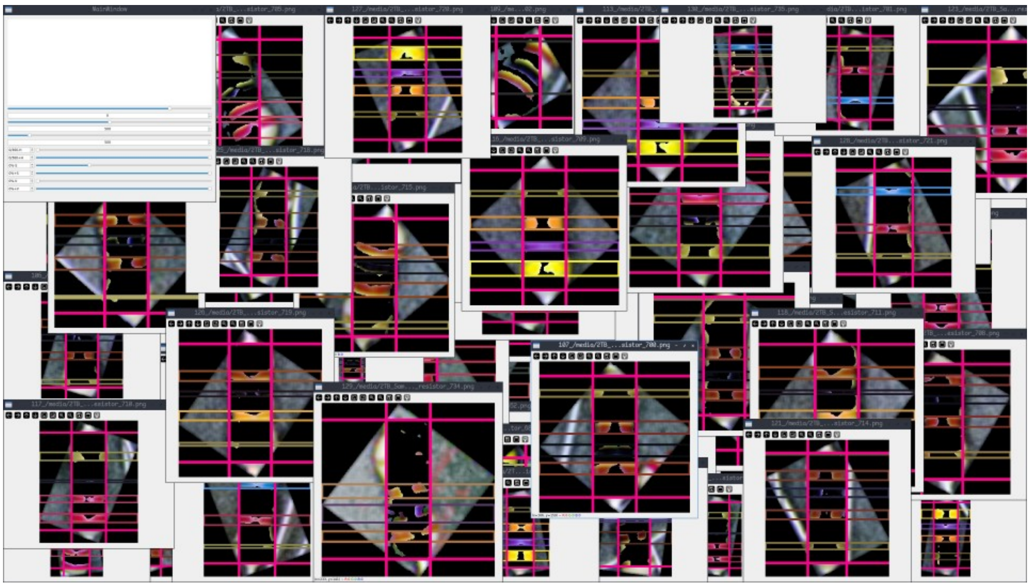


Figure 103: Post-Processing: Resistor: Bulk testing and calibration example 2

Example Results

- **Bulk Calibration**

Determined resistor values are displayed as the last numbers in the window name. The algorithm achieved 100% success rate for this particular random batch test.

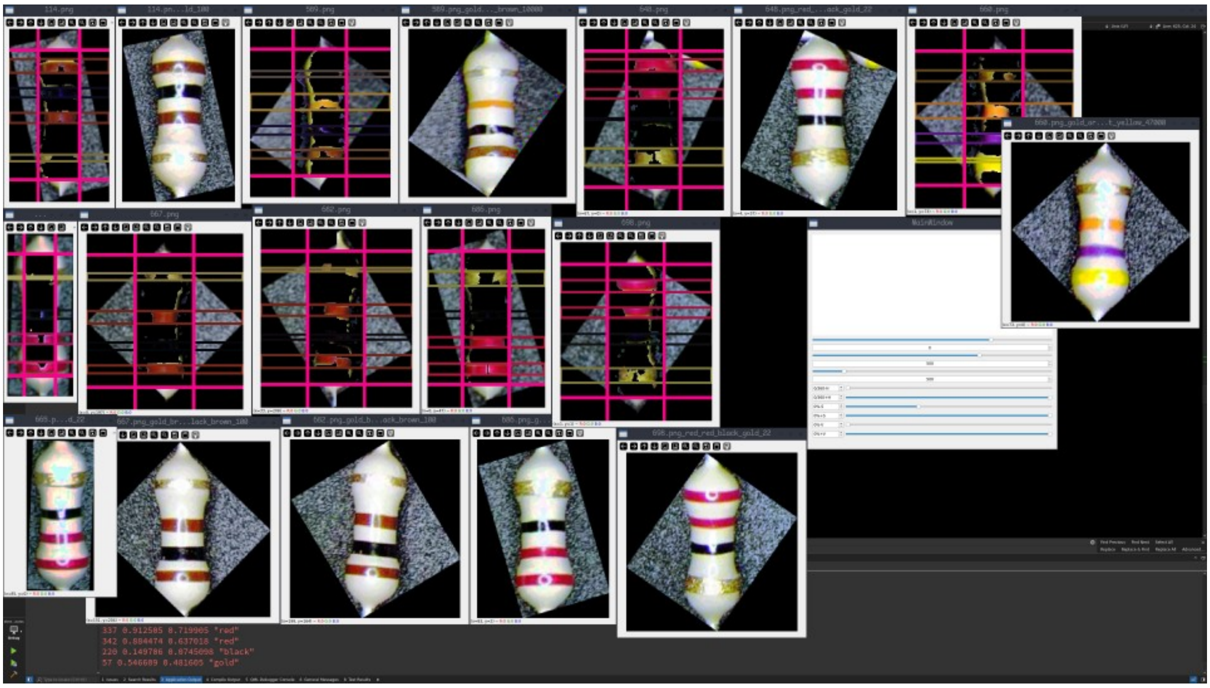


Figure 104: Post-Processing: Resistor: Testing and calibration, found values displayed at the end of the window name.

- **Found Optimal Calibration**

70% horizontal cut off (vertical magenta bounds)

25% vertical cut off (horizontal magenta bounds)

12.5% cut out from the vertical middle

Full Hue range

35% to 100% Saturation range

Full Value range

- Final result of the algorithm, identifying 4 bands with colors: Yellow, Purple, Green, and Gold.

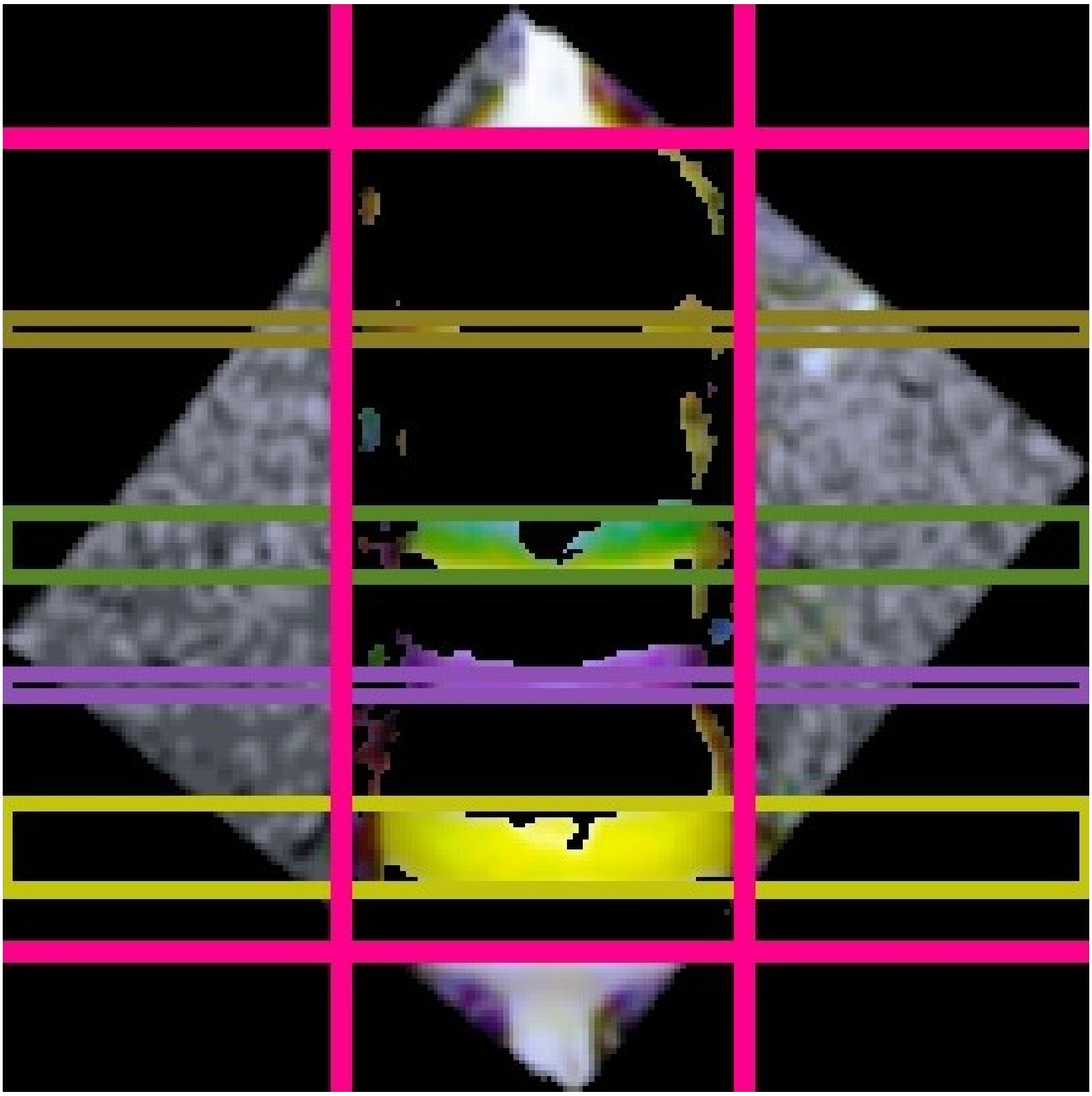


Figure 105: Post-Processing: Resistor: Successful implementation on a static image

Optical Character Recognition

Even in the most ideal of conditions – OCR has more often than not failed to provide adequate results, while costing a significant amount of processing power, with drops in frames down to 25%, or below.

Some of the components may have their surface worn, while others are very reflective and cause significant glare.

OCR has proved itself to be rather unreliable. Furthermore – the testing was done while facing the component upright.

In a real life scenario, the component will face any random direction, so it must first be rotated in the exact direction to face upwards to be readable.

While this is certainly achievable, it is a further hit to the processing power, because each rotation of 0, 90, 180, and 270 degrees would require a test – increasing the total required processing power to over 4 times of what simply running OCR on a prepared image would cost.

It has been concluded that OCR is simply not something that is feasible to be utilised during runtime for this project.

Perhaps with further development in OCR efficiency, the project may expand to support OCR in the future.

Future deployment

Qt creator has a Maintenance tool, that allows to modify which modules are installed, after the initial installation.

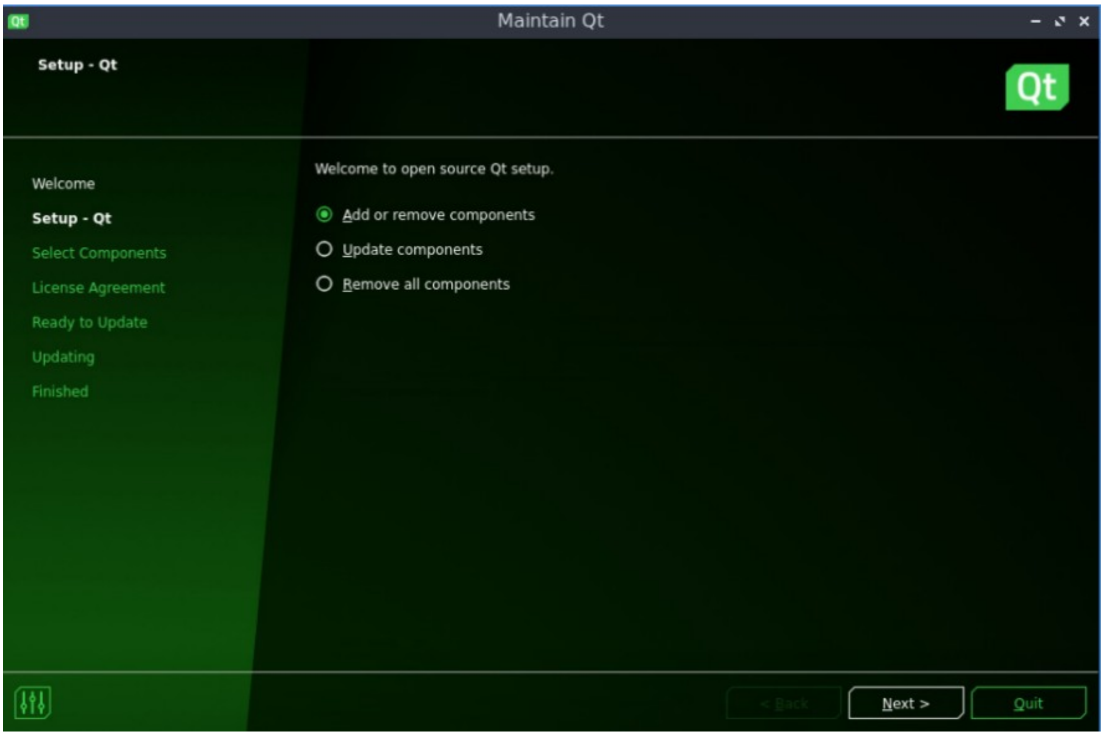


Figure 106: Qt Creator: Maintenance Tool: Add/Remove/Update choice

This tool is very convenient when requiring a new module, for example: Ninja, for Android development.

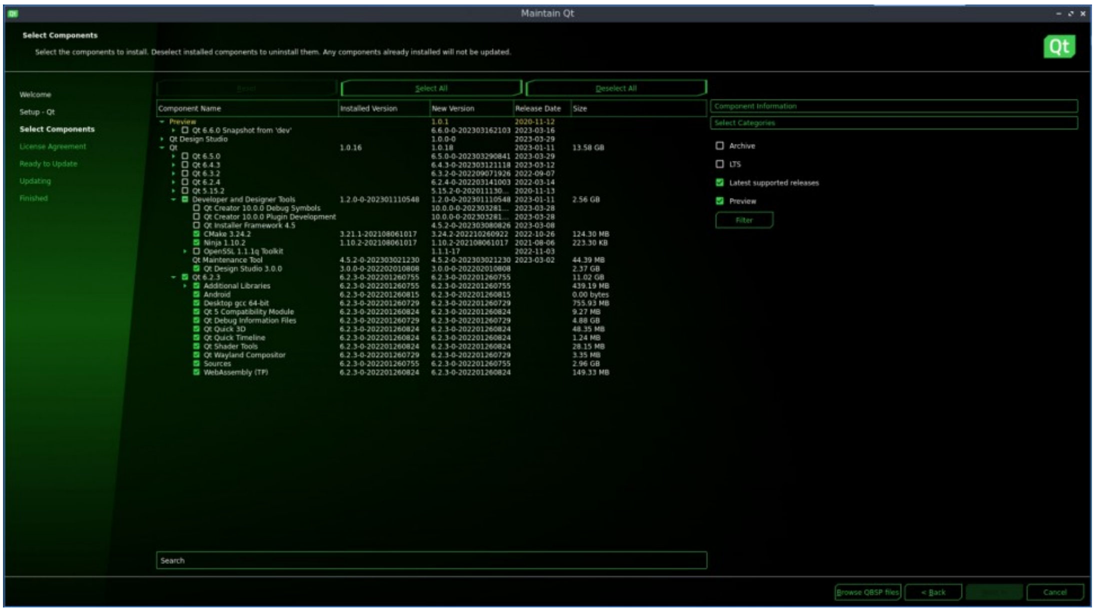


Figure 107: Qt Creator: Maintenance Tool: Component selection

10. Timeline

Name	Begin date	End date	Duration	Duration (Weeks, rounded up)
Camera rig	9/20/22	10/4/22	15	3
Delivery times (Rig, Camera)	9/20/22	10/3/22	14	2
Physical rig setup with camera	10/4/22	10/4/22	1	1
GUI	9/1/22	10/19/22	49	7
Basic GUI (Console, smart camera connectivity, display, interface)	9/1/22	10/19/22	49	7
Camera feed to GUI interfacing (Includes resolution/framerate/frame handling, QoL methods)	10/5/22	10/19/22	15	3
Research	8/1/22	1/19/23	172	25
Optimal for inference camera research	9/1/22	9/12/22	12	2
AI object detection research and picking the most optimal architecture for the project (YoloV5 was picked)	8/1/22	9/19/22	50	8
Optimal camera rig research	9/13/22	9/19/22	7	1
Devices to potentially deploy on research	12/6/22	1/4/23	30	5
Color code identification from images research	1/6/23	1/19/23	14	2
Post processing	1/20/23	3/1/23	41	6
Optical Character Recognition (OCR)	1/20/23	2/18/23	30	5
Color code reading	1/20/23	3/1/23	41	6
Object detection implementation	9/1/22	2/14/23	167	24
First, basic dataset gathering (manual for early, asynchronous Inference testing) and model training	10/5/22	10/5/22	1	1
Bare minimum YoloV5 training and Inference implementation	9/1/22	10/30/22	60	9
Further data gathering and training of the model on various components	10/31/22	2/14/23	107	16
Deployment on a more mobile device	3/3/23	4/18/23	47	7

Figure 108: Project timeline table

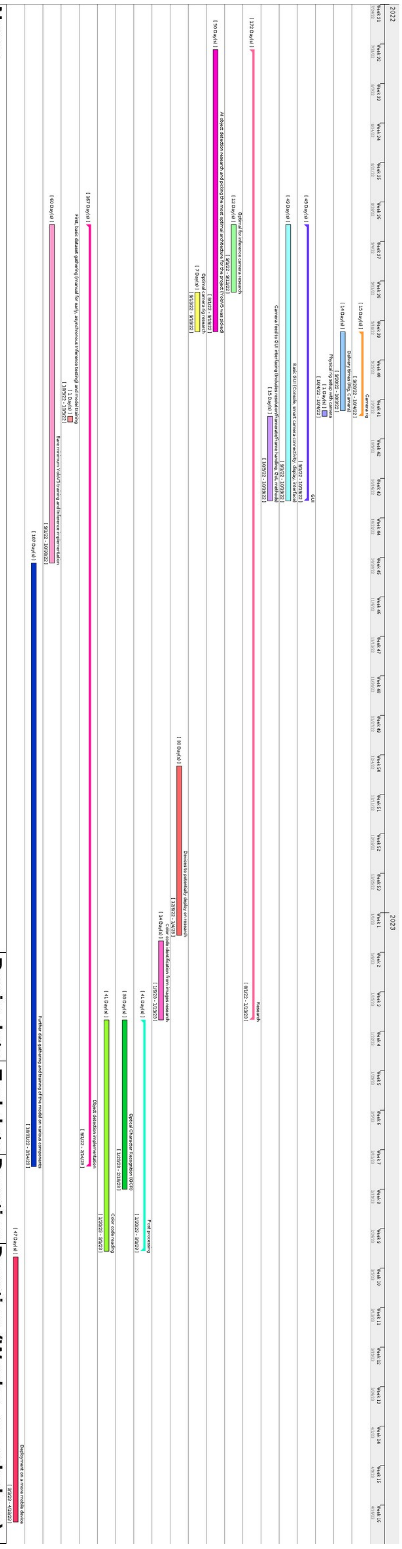


Figure 109: Project timeline [39]

10.1.1. UI improvements

Contrast text

The text creation was upgraded from plain color text, to a bright outline with a darker color fill.

This resulted in ease of readability of the text under all background conditions.

Examples

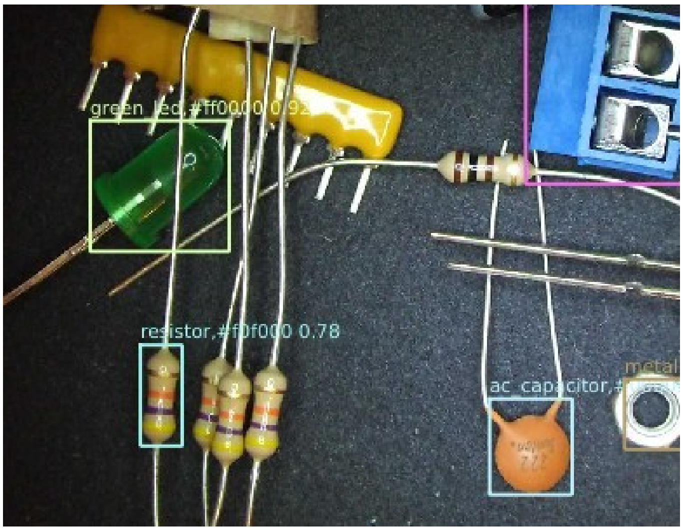


Figure 110: Plain text example

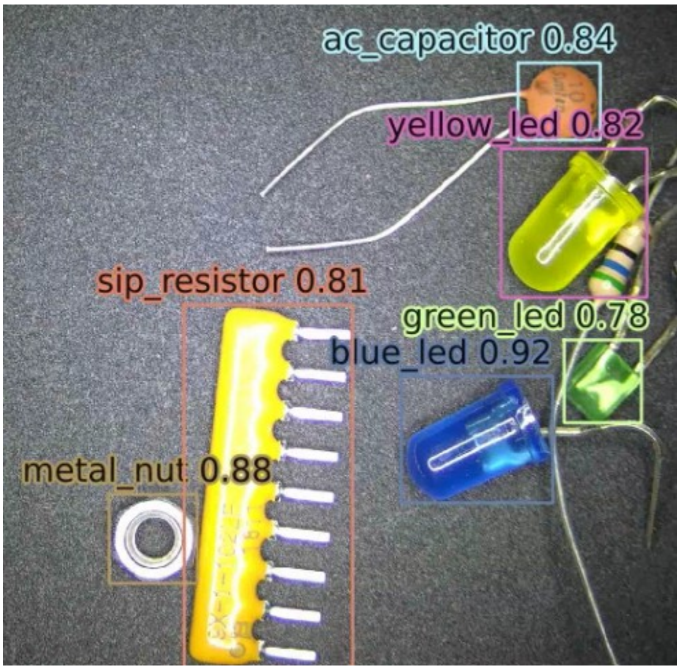


Figure 111: Contrast text example

Horizontal Text Adjustment

The text position adjustment was designed to always fit text on the screen without it leaving the screen.

To achieve this, the horizontal width of the screen and the box, position of the box on the screen, and the display text width had to be considered.

It is designed for the text to start directly from the start of the component when the component is all the way on the left, and to end at the end of the component when the component is all the way to the right, with a smooth transition in-between.

Examples

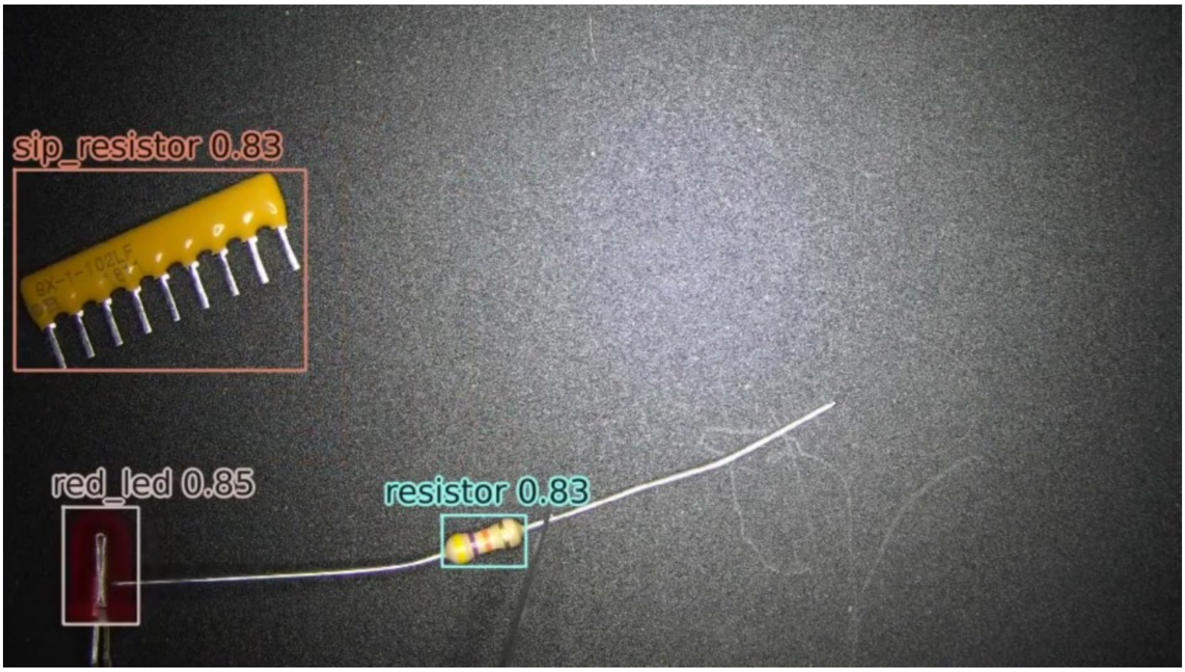


Figure 112: Horizontal text adjustment: left side

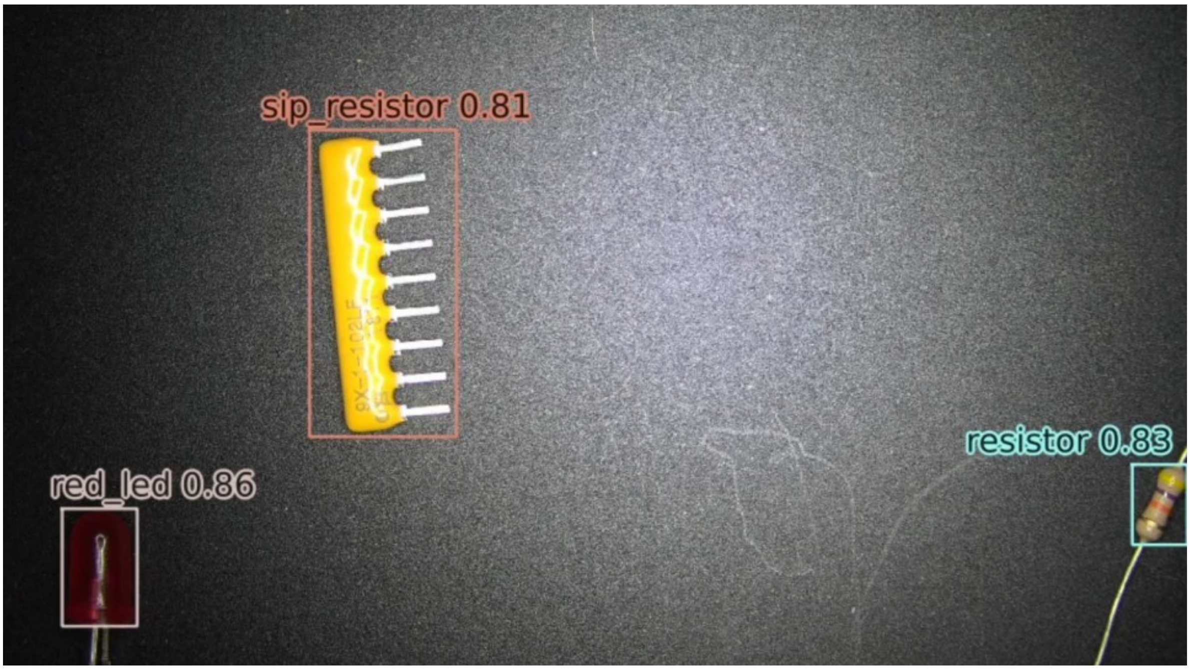


Figure 113: Horizontal text adjustment: middle

Figure 114: Horizontal text adjustment: right side

Post Processing

Figure 115: Post processing identifying resistor colors

Figure 116: Post processing identifying ohm values

10.2. *Progress*

10.2.1. Issues encountered

Augmentation Rotation Issue

Description

A glitch in augmentation provided by YOLOv5 [8], where rotation during augmentation has shifted the bounding boxes of the components, causing inaccurate feedback to the model, and preventing it from training appropriately.

Example

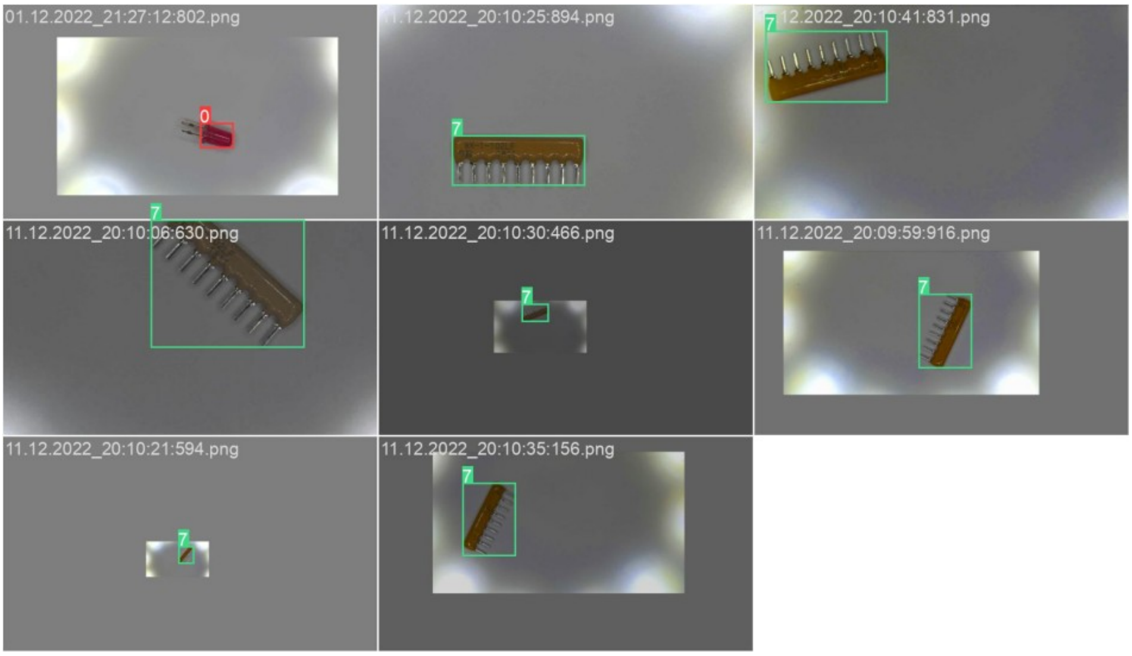


Figure 117: Rotation augmentation issue: Expected bounding boxes after rotation augmentation

Note the snug fit of the bounding box around the edges of the component.

That is desirable, as it provides accurate information on what the model should be looking for.

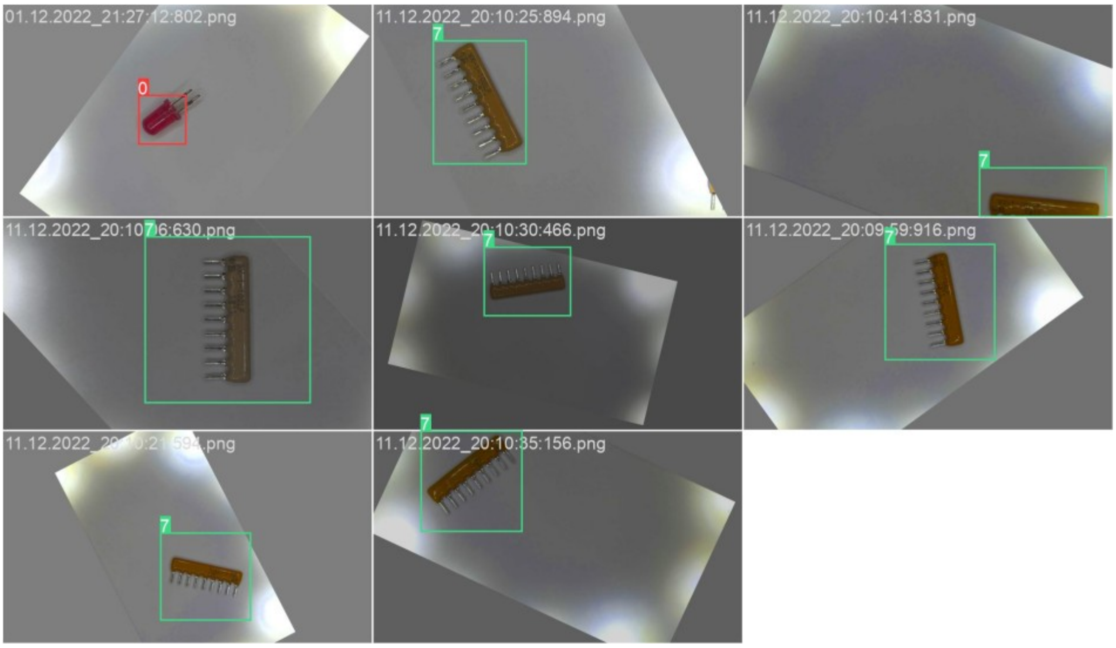


Figure 118: Rotation augmentation issue: Actual bounding boxes after augmentation

Note the unnecessarily expanded bounding boxes.

This will train the model in undesirable ways, detecting parts it should not.

This issue has been submitted to the YoloV5 GitHub [40] repository issues page: https://github.com/ultralytics/yolov5/issues/10639

Information gathered from replies as of today's date

This issue has been reported to be part of YOLOv7 [41] augmentation also, and is currently on the list of issues that need to be resolved.

Training

1st Batch, Test Run

Architecture: YOLOv5

Pre-trained model used: yolov5s

Image Count

- **Training:** 100
- **Evaluation:** 20

Class: Resistor only

Augmentation: Default

Epoch count: 400

Average time per epoch: 34 seconds

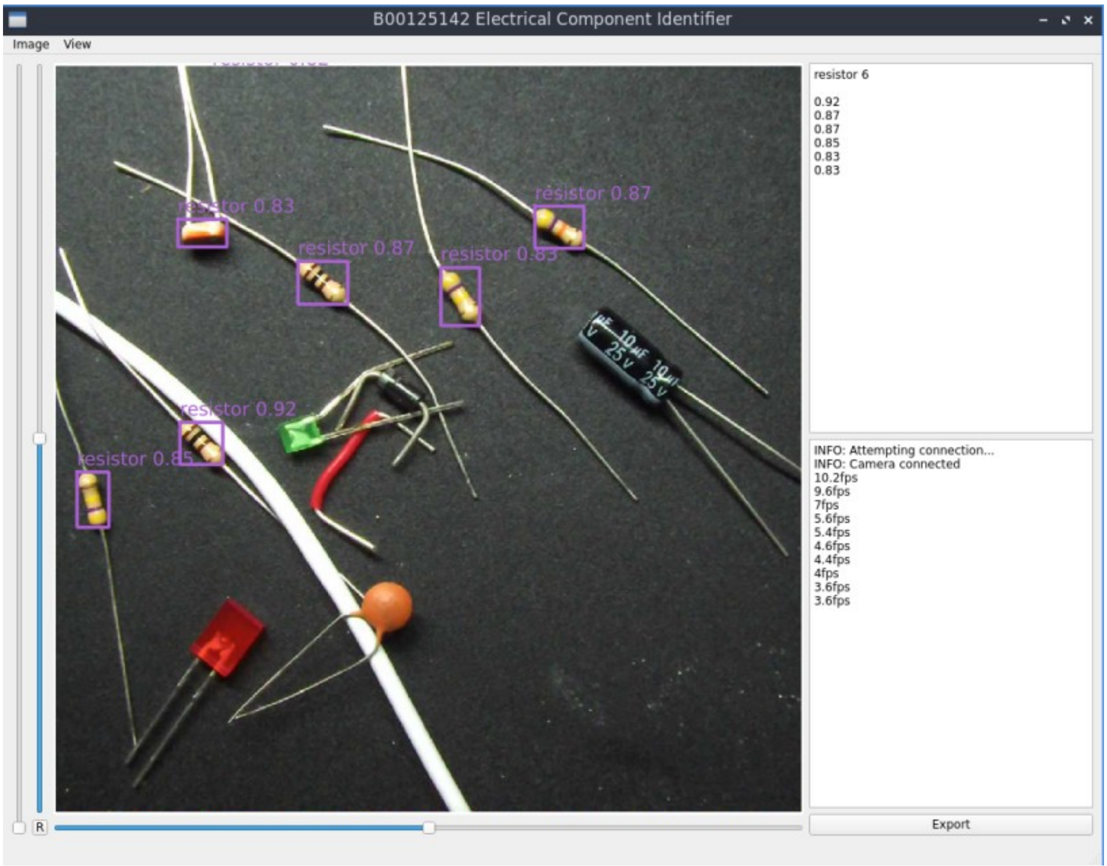


Figure 119: 1st batch model inference example

Observations

Surprisingly good results for a model trained from 120 images, with confidence values above 0.8 and sometimes over 0.9!

An LDR has been detected as a resistor.

This is due to the fact that it is in some sense similar to a resistor, especially when only trained on a very low end of 3 digits amount of images.

Further, it was only trained on resistors. The model has never seen an LDR to be trained against it.

2nd batch

Architecture: YOLOv5

Pre-trained model used: yolov5m

Image Count

- **Training:** 1800
- **Evaluation:** 540

Classes

1. red_led
2. green_led
3. blue_led
4. yellow_led
5. ac_capacitor
6. dc_capacitor
7. resistor
8. sip_resistor
9. pcb_terminal

Augmentation: Default

Epoch count: 250

Average time per epoch: 2 minutes

Observations

Rather poor results. Confidence values usually below 0.7, and struggled to classify accurately.

3rd Batch

Architecture: YOLOv5

Pre-trained model used: yolov5m

Image Count

- **Training:** 2393
- **Evaluation:** 724

Classes

1. red_led
2. green_led
3. blue_led
4. yellow_led
5. ac_capacitor
6. dc_capacitor
7. resistor
8. sip_resistor
9. pcb_terminal
10. metal_nut

Augmentation: Default

Epoch count: 300

Average time per epoch: 2 minutes and 20 seconds

Observations

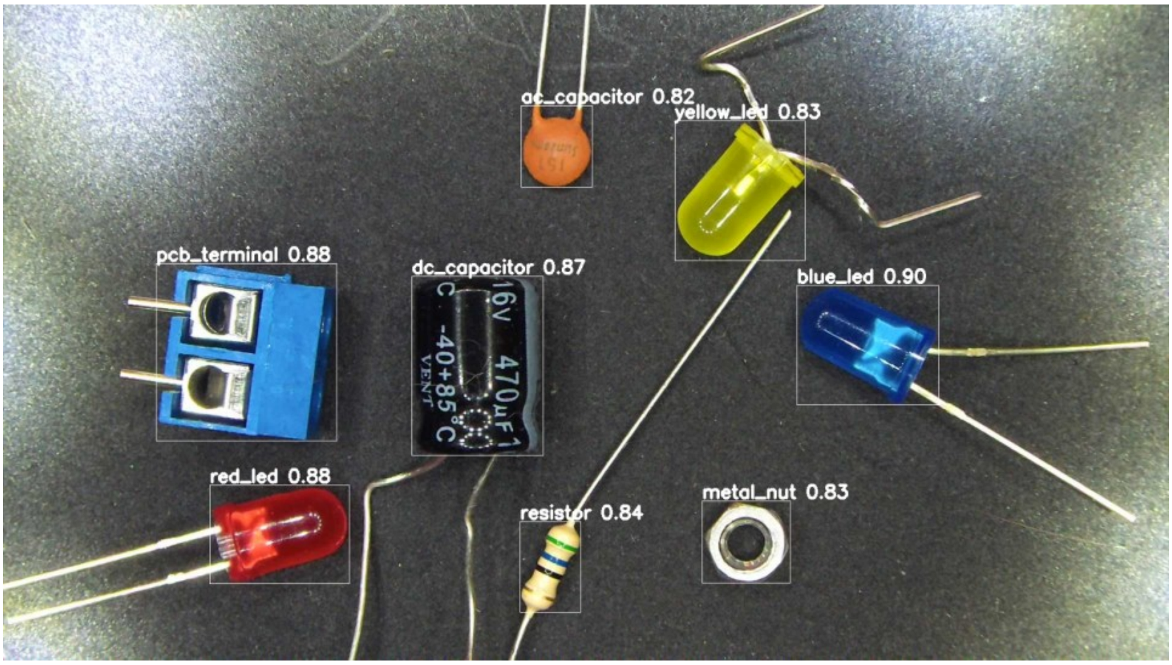


Figure 120: 3rd batch model inference example

A ring light has been introduced for both training and inference running purposes.

Great results with confidence values consistently above 0.8, classifying all classes accurately!

The new-found boost in performance is limited to the environment of a set rig.

11. Discussion

Great results have been achieved in object detection.

The model has been trained on on over 3000 images that were taken and labelled over the course of several months.

At this rate, a significant increase in the dataset would be required for adequate results outside of a set rig, which would require a significant amount of additional time investment that is beyond the time constraints for this project. This, of course, is not a viable option in the provided time constraint, but should be considered were the project to be continued in the future.

A further boost in performance was instead achieved by transitioning to a bigger model, which came with a cost to the performance, dropping the frame-rate from an average of 28 frames per second, to 12 frames per second. This compromise of inference confidence over computational performance was deemed preferable.

The training process for the model was well-planned and executed, resulting in a well-tuned model that is capable of meeting the criteria for a set-rig operation of the project.

Overall, YOLOv8 at the time of writing was not at a state in which it would be preferable to utilise it in favor of YOLOv5. YOLOv8 is still in development and while the planned features will most certainly be incredible – YOLOv5 was chosen due to reliability and availability of the existing features.

The post-processing algorithms were carried out as initially conceptualised, and turned out to be a great success!

However, post-processing came at the cost of a considerable amount of processing power but has yielded very impressive results that otherwise could've only been hoped for.

The original project concept at the time of the project proposition encountered a slight shift in focus.

The project was intended to focus on a field that is more familiar, which was more on the theme of a high precision tool, which requires a set rig to push beyond the limitations to produce high quality results.

At some point, the focus shifted towards a more flexible but less precise concept with the consideration of more generic use through mobile deployment.

As mobile deployment of object detection was not explored beforehand nor planned for, this has put additional strain on the timeline.

Unfortunately, it has been concluded that even if the dataset had been expanded a few times, mobile deployment with a live video feed would simply not be feasible due to the hardware limitations, such as a considerably weaker graphics card, or the fact that it would lack CUDA cores, which accelerate inference by a considerable amount – reducing the performance drastically. Additionally, the lack of a set rig environment that the project has been designed for would most likely require an even bigger model to be trained and utilised, in order to provide sufficient results in the vastly more varied environment that a mobile device camera could be exposed to.

An option that was heavily considered was to move the processing part onto the cloud to compensate for the lack of computational power, however – the project's core design was to provide live video feed processing as an assistant tool the user may use for live reference as they move the components around. Reduction from live video on a screen to manually taking pictures and waiting for the results to come back would've interfered with the core idea to a point where it would have become a different project, regardless – the research towards this potential goal was very insightful, and has inspired several future projects.

With more resources and advances in mobile technology – this project has the potential to one day make it into the mobile device market.

12. Conclusion

In conclusion– The initial goals of the project were fulfilled with success, beyond the expected results for the time-frame that was available.

while the project did not meet all the new goals, it did explore them thoroughly, providing various insight into those topics.

Throughout the project, many challenges were faced – including picking and training the right AI model for the job, performance and accuracy optimisations, and technical issues. However, these issues were handled with proper care, and were resolved relatively swiftly and thoroughly.

The project was an overall great success, both as a learning experience, and as a utility tool it has set out to be. It was finished on a great note– in which it serves its intended function adequately, and has room to be expanded by further training of the model and expansion of supported platforms.

13. References

[1] G. Jocher, A. Chaurasia, and J. Qiu, "YOLO by Ultralytics." Jan. 2023. Accessed: Jan. 29, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[2] "Pine Tree Facts, Types, Identification, Diseases, Pictures," *Coniferous Forest*. https://www.coniferousforest.com/plants-trees/pine (accessed Jan. 29, 2023).

[3] "Microorganism recognition and counting on Petri plates," *NeuroSYS: AI & Custom Software Development*, Oct. 13, 2021. https://neurosys.com/blog/recognition-and-counting-of-microorganisms (accessed Jan. 27, 2023).

[4] X. Liao, S. Lv, D. Li, Y. Luo, Z. Zhu, and C. Jiang, "YOLOv4-MN3 for PCB Surface Defect Detection," *Appl. Sci.*, vol. 11, no. 24, Art. no. 24, Jan. 2021, doi: 10.3390/app112411701.

[5] B. T. Marvel, "CROP DISEASE DETECTION USING IMAGE PROCESSING TECHNIQUE AND CNN NETWORKS : Part2," *Medium*, May 01, 2022. https://medium.com/@b.thusharmarvel97/crop-disease-detection-using-image-processing-technique-and-cnn-networks-part2-b3f8588f77e5 (accessed Jan. 27, 2023).

[6] Q. Jiang, D. Tan, Y. Li, S. Ji, C. Cai, and Q. Zheng, "Object Detection and Classification of Metal Polishing Shaft Surface Defects Based on Convolutional Neural Network Deep Learning," *Appl. Sci.*, vol. 10, no. 1, Art. no. 1, Jan. 2020, doi: 10.3390/app10010087.

[7] R. Girshick, "rbgirshick/rcnn." Jan. 29, 2023. Accessed: Jan. 29, 2023. [Online]. Available: https://github.com/rbgirshick/rcnn

[8] "ultralytics/yolov5." Ultralytics, Nov. 06, 2022. Accessed: Nov. 06, 2022. [Online]. Available: https://github.com/ultralytics/yolov5

[9] "Detection - Ultralytics YOLOv8 Docs." https://docs.ultralytics.com/tasks/detection/ (accessed Jan. 24, 2023).

[10] "7 Object Detection with R-CNN, SSD, and YOLO · Deep Learning for Vision Systems." https://livebook.manning.com/book/deep-learning-for-vision-systems/chapter-7/v-6/ (accessed Nov. 07, 2022).

[11] "Embedded Software Development Tools | Cross Platform IDE | Qt Creator." https://www.qt.io/product/development-tools (accessed May 03, 2022).

[12] "Git." https://git-scm.com/ (accessed May 09, 2023).

[13] "GitKraken Legendary Git Tools | GitKraken." https://www.gitkraken.com (accessed May 07, 2023).

[14] "cplusplus.com - The C++ Resources Network." https://www.cplusplus.com/ (accessed May 03, 2022).

[15] "46.12US $ 25% OFF|50mm Adjustable Focusing Holder Boom Arm Big Size Table Stand 180x 300x 200x 500 C Mount Lens Bracket For Lab Microscope Camera - Microscope Parts & Accessories - AliExpress," *aliexpress.com*. //www.aliexpress.com/item/32841619593.html?src=ibdm_d03p0558e02r02&sk=&aff_platform=&aff_trace_key=&af=&cv=&cn=&dp= (accessed Feb. 02, 2023).

[16] "244.77US $ |4k 30fps Hdmi Camera 1080p 60fps 1080i Live Webcam Usb Camera Recording 4k@30fps Industry C/cs-mount Camera With 4-12mm Lens - Industrial Camera - AliExpress," *aliexpress.com*. //www.aliexpress.com/item/4001178568776.html?src=ibdm_d03p0558e02r02&sk=&aff_platform=&aff_trace_key=&af=&cv=&cn=&dp= (accessed Feb. 02, 2023).

[17] "14.99US $ 24% OFF|Usb 144 Led Ring Light Illuminator 64 Lamp Industry Monocular Binocular Trinocular Stereo Video Microscope Lens Camera Magnifier - Microscopes - AliExpress," *aliexpress.com*. //www.aliexpress.com/item/1005003442168493.html?src=ibdm_d03p0558e02r02&sk=_mLKffyW&aff_platform=default&aff_trace_key=2d319059a2f6

424782f0b6314c8312fc-1673102021784-03474-_mLKffyW&af=&cv=&cn=&dp= (accessed Feb. 02, 2023).
[18] "Buy a Raspberry Pi 4 Model B – Raspberry Pi." https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ (accessed Jan. 24, 2023).
[19] "BeagleBoard.org - bone." https://beagleboard.org/bone (accessed Jan. 24, 2023).
[20] "Jetson Nano Developer Kit," *NVIDIA Developer*, Mar. 06, 2019. https://developer.nvidia.com/embedded/jetson-nano-developer-kit (accessed Jan. 24, 2023).
[21] "Intel® Neural Compute Stick 2," *Intel*. https://www.intel.com/content/www/us/en/developer/articles/tool/neural-compute-stick.html (accessed Jan. 29, 2023).
[22] "SHAVE v2.0 - Microarchitectures - Intel Movidius - WikiChip." https://en.wikichip.org/wiki/movidius/microarchitectures/shave_v2.0 (accessed May 09, 2023).
[23] "Android," *Android*. https://www.android.com/intl/en_ie/ (accessed Jan. 29, 2023).
[24] "Experience the Power of Windows 11 OS, Computers & Apps | Microsoft." https://www.microsoft.com/en-ie/windows (accessed Jan. 29, 2023).
[25] "Linux.org," *Linux.org*, Jan. 26, 2023. https://www.linux.org/ (accessed Jan. 29, 2023).
[26] "Mac," *Apple (Ireland)*. https://www.apple.com/ie/mac/ (accessed Jan. 29, 2023).
[27] "C104-25 Disc Capacitor, 0.1uf 25v (104z)." https://www.rfparts.com/c104-25.html (accessed Jan. 29, 2023).
[28] "New And Original Tps61080drcr Ic Integrated Circuit Electronic Components Marking Code Bcn Tps61080drc Vson-10 - Buy Tps61080drcr,Bcn,Tps61080drcr Original Product on Alibaba.com." https://www.alibaba.com/product-detail/New-and-Original-TPS61080DRCR-IC-Integrated_1600584880532.html (accessed Jan. 24, 2023).
[29] "Official Adobe Photoshop | Photo and design software." https://www.adobe.com/ie/products/photoshop.html (accessed Jan. 29, 2023).
[30] "HSL and HSV," *Wikipedia*. Jan. 04, 2023. Accessed: Jan. 24, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=HSL_and_HSV&oldid=1131514567
[31] "PC Components | Gaming Gear | CORSAIR." https://www.corsair.com/ww/en/ (accessed Jan. 29, 2023).
[32] "Home," *OpenCV*. https://opencv.org/ (accessed May 09, 2023).
[33] "EdrawMax Online: All-in-One Diagram Maker & Software." https://www.edrawmax.com/ (accessed May 03, 2022).
[34] "Easy-to-use Software, Easily Get Inspirations," *Edrawsoft*. https://www.edrawsoft.com/ (accessed May 09, 2023).
[35] "[OFFICIAL] EdrawMind: Free Collaborative Mind Mapping Software." https://www.edrawmind.com/ (accessed Apr. 30, 2023).
[36] "opencv/opencv_contrib." OpenCV, May 09, 2023. Accessed: May 09, 2023. [Online]. Available: https://github.com/opencv/opencv_contrib
[37] "CUDA GPUs - Compute Capability," *NVIDIA Developer*, Jun. 04, 2012. https://developer.nvidia.com/cuda-gpus (accessed May 09, 2023).
[38] "cmake-gui(1) — CMake 3.26.3 Documentation." https://cmake.org/cmake/help/latest/manual/cmake-gui.1.html (accessed May 12, 2023).
[39] B. S. s.r.o, "GanttProject: free project management tool for Windows, macOS and Linux," *GanttProject*. https://www.ganttproject.biz (accessed May 03, 2022).
[40] "Build software better, together," *GitHub*. https://github.com (accessed May 03, 2022).
[41] K.-Y. Wong, "Official YOLOv7." Jan. 29, 2023. Accessed: Jan. 29, 2023. [Online]. Available: https://github.com/WongKinYiu/yolov7
[42] "Small Business for Teams for Free." https://www.microsoft.com/en-ie/microsoft-teams/free (accessed May 09, 2023).

14. Appendix

14.1. *Project Planning*

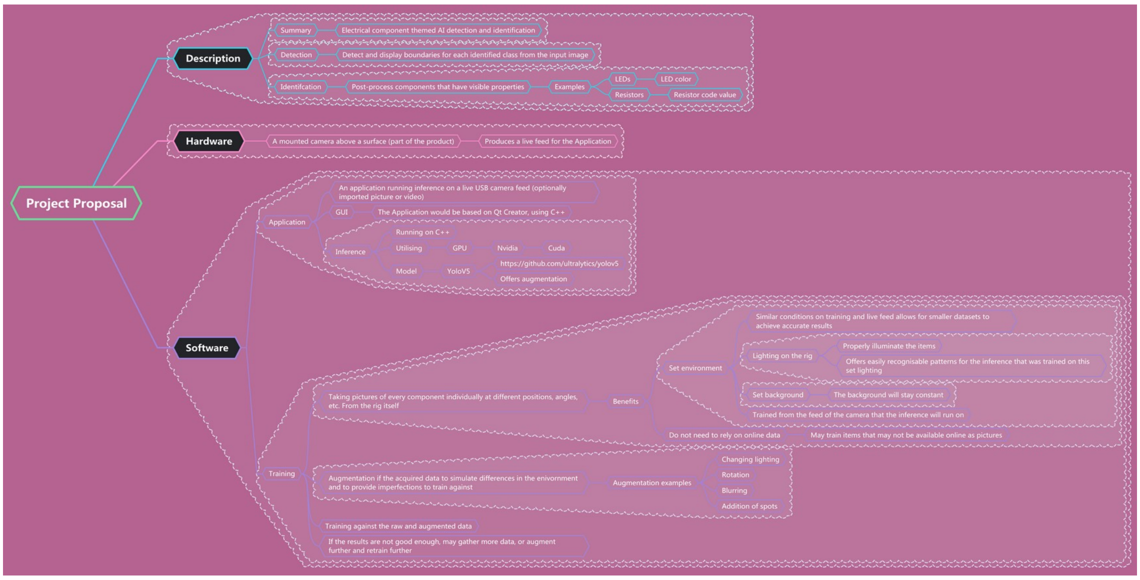


Figure 121: The original mind map of the project proposal using EdrawMind **[35]**

Project Proposal

The original plans and milestones of the project set out were proven to be well thought-through, providing the project with a solid foundation for the next milestones to be based upon.

Additional goals and approaches were discussed in great detail with the project supervisor and were researched thoroughly. Mobile deployment was looked into heavily, but unfortunately, the level of accuracy and processing power required for mobile deployment, without even considering the intensity of post-processing – is simply too much for a mobile device. However, even though the research has put a strain on the project deadlines – it was a valuable resource for this project, and those to come in the future.

14.2. *Salient Extracts from Project Diary*

Most of the meetings were held either during the project class time, or after class.

In-person meeting notes

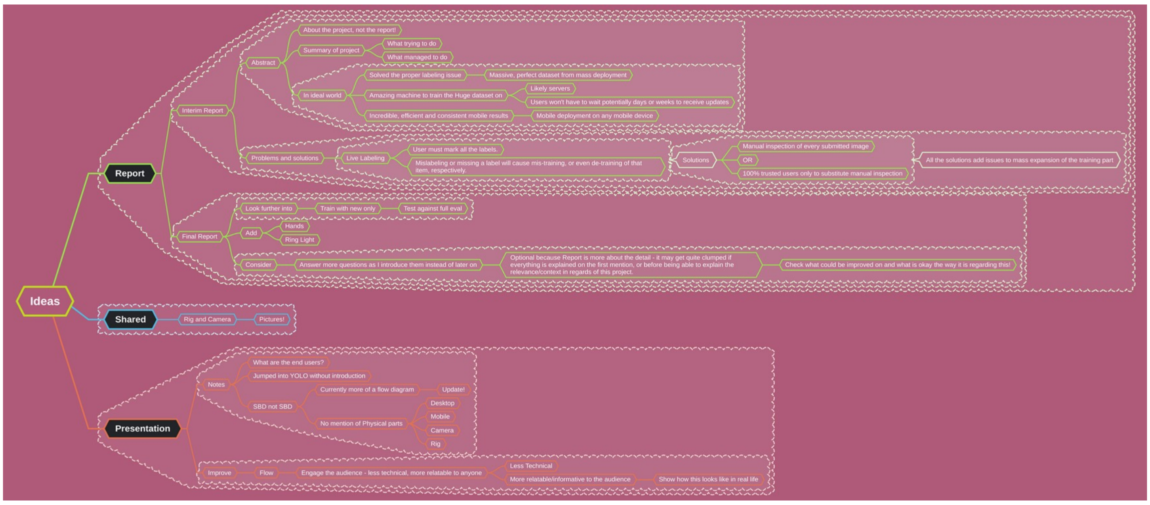


Figure 122: Raw notes taken during one of the meetings with the project supervisor

[35]

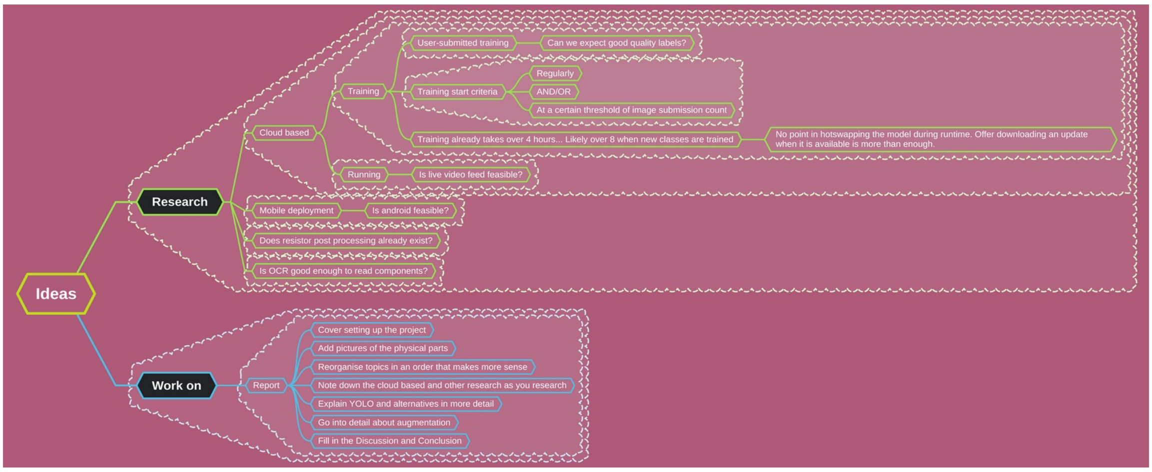


Figure 123: Raw notes taken during another one of the meetings with the project supervisor

[35]

A few of the meetings were also held online.

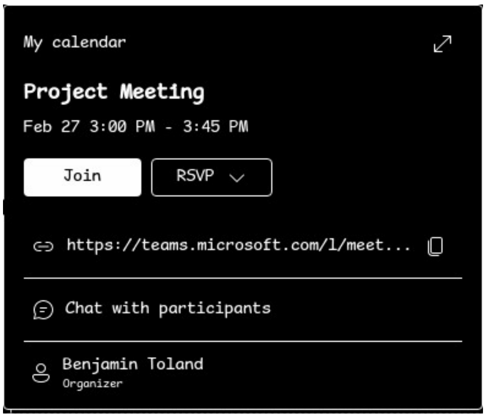


Figure 124: Online Meeting 1 [42]

This meeting discussed:

- the post processing algorithms.
- Potential of processing on the cloud.
- Overall recap of the progress.

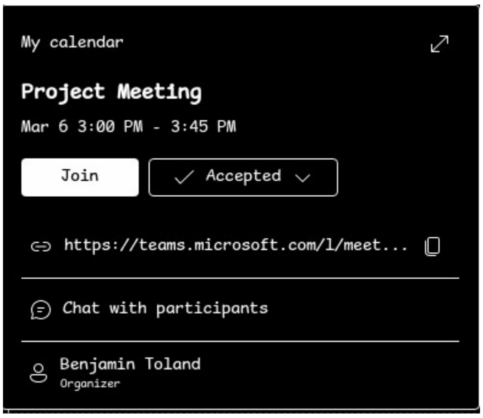


Figure 125: Online Meeting 2 [42]

This meeting discussed:

- The state and performance of the resistor post-processing.
- Testing the post-processing algorithm performance in bulk.
- Expectations for the remaining weeks.

14.3. *List of Software Code*

Static constants

```
static inline const double BEST_RED = 255, BEST_GREEN = 0, BEST_BLUE = 141;
static inline const QColor BEST_COLOR = QColor(BEST_RED, BEST_GREEN, BEST_BLUE);
static inline const Scalar BEST_SCALAR = Scalar(BEST_COLOR.blue(),  BEST_COLOR.green(), BEST_COLOR.red());
static inline const QColor WHITE_COLOR = QColor(255, 255, 255);
static inline const Scalar WHITE_SCALAR = Scalar(WHITE_COLOR.blue(),  WHITE_COLOR.green(), WHITE_COLOR.red());
static inline const QColor BLACK_COLOR = QColor(0, 0, 0);
static inline const Scalar BLACK_SCALAR = Scalar(BLACK_COLOR.blue(),  BLACK_COLOR.green(), BLACK_COLOR.red());

static inline const QString BLACK = "black";
static inline const QString BROWN = "brown";
static inline const QString RED = "red";
static inline const QString ORANGE = "orange";
static inline const QString YELLOW = "yellow";
static inline const QString GREEN = "green";
static inline const QString BLUE = "blue";
static inline const QString VIOLET = "violet";
static inline const QString GREY = "grey";
static inline const QString WHITE = "white";
static inline const QString GOLD = "gold";

static inline const QString RESISTOR = "resistor";
static inline const QString DC_CAPACITOR = "dc_capacitor";

static QMap<QString, int> makeResistorColorCodesMap()
{
    QMap<QString, int> map;
    map.insert(BLACK, 0);
    map.insert(BROWN, 1);
    map.insert(RED, 2);
    map.insert(ORANGE, 3);
    map.insert(YELLOW, 4);
    map.insert(GREEN, 5);
    map.insert(BLUE, 6);
    map.insert(VIOLET, 7);
    map.insert(GREY, 8);
    map.insert(WHITE, 9);
    map.insert(GOLD, 5);
    return map;
}
```

Note: “best” color is absolutely subjective. It is a play on the fact that this is one of my favorite colors.

Getting a component's orientation by creating a bounding box

```
static double getOrientation(const std::vector<Point>& pts, Mat& image, const double drawScale = 0)
{
    //Construction of a buffer for the PCA analysis
    const int imageSize = static_cast<int>(pts.size());
    Mat dataPoints(imageSize, 2, CV_64F);
    for (int i = 0; i < dataPoints.rows; i++) {
        dataPoints.at<double>(i, 0) = pts[i].x;
        dataPoints.at<double>(i, 1) = pts[i].y;
    }

    //PCA analysis (EigenValues, EigenVectors, etc!)
    const PCA pca(dataPoints, {}, PCA::DATA_AS_ROW);

    //Center of the bounding box (mean|average)
    Point center(pca.mean.at<double>(0, 0), pca.mean.at<double>(0, 1));

    //Orientation in radians
    double angleRads = atan2(pca.eigenvectors.at<double>(0, 1), pca.eigenvectors.at<double>(0, 0));

    if(drawScale > 0)
    {
        //Drawing the principal components
        circle(image, center, 3, Scalar(0, 255, 255), 2*drawScale);
        Point2d p1(center.x + 0.2*drawScale * pca.eigenvectors.at<double>(0, 0) * pca.eigenvalues.at<double>(0, 0),
                   center.y + 0.2*drawScale * pca.eigenvectors.at<double>(0, 1) * pca.eigenvalues.at<double>(0, 0));
        Point2d p2(center.x - 0.2*drawScale * pca.eigenvectors.at<double>(1, 0) * pca.eigenvalues.at<double>(1, 0),
                   center.y - 0.2*drawScale * pca.eigenvectors.at<double>(1, 1) * pca.eigenvalues.at<double>(1, 0));
        line(image, center, p1, Scalar(255, 0, 255), 2*drawScale);
        line(image, center, p2, Scalar(255, 255, 0), 1*drawScale);
    }

    return angleRads;
}
```

Determines the orientation of an item in the image by first drawing contours around it, and then investigating the trend.

HSV filter

```
static void hsvFilter(Mat inputMat,
                      const double &minHueF = 0, const double &maxHueF = 1,
                      const double minSatF = 0, const double maxSatF = 1,
                      const double minValF = 0, const double maxValF = 1,
                      const QColor elseColor = Util::BLACK_COLOR, const bool type = false,
                      const int inX1 = -1, const int inY1 = -1, const int inX2 = -1, const int inY2 = -1)
{
    //Only apply filter to area optimisation
    //Determine minimum and maximum x and y bounds
    const int x1 = inX1 >= 0 ? max(0, inX1) : 0;
    const int x2 = inX2 >= 0 ? min(inputMat.cols-1, inX2) : inputMat.cols-1;

    const int y1 = inY1 >= 0 ? max(0, inY1) : 0;
    const int y2 = inY2 >= 0 ? min(inputMat.rows-1, inY2) : inputMat.rows-1;

    for(int x = x1; x < x2; ++x)
    {
        for(int y = y1; y < y2; ++y)
        {
            Vec3b &bgr = inputMat.at<Vec3b>(Point(x, y));
            //QColor is in rgb format, while OpenCV Mat is in bgr.
            const QColor color = QColor(bgr[2], bgr[1], bgr[0]);
            const double hue = Util::clamp(color.hueF(), 0, 1);
            const double sat = color.saturationF();
            const double val = color.valueF();

            const bool correctHue = filter(hue, minHueF, maxHueF);
            const bool correctSat = filter(sat, minSatF, maxSatF);
            const bool correctVal = filter(val, minValF, maxValF);

            if(type == (correctHue && correctSat && correctVal))
            {
                bgr[0] = elseColor.blue();
                bgr[1] = elseColor.green();
                bgr[2] = elseColor.red();
            }
        }
    }
}
```

Filters out pixels that do not meet the Hue Saturation Value filter, replacing them with the elseColor.

Filter logic

```
static bool filter(const qreal &input, const qreal &minBound, const qreal &maxBound)
{
    if(maxBound < minBound) //Outside range filter mode
        return input < maxBound || input > minBound;
    else //Inside range filter mode
        return minBound <= input && input <= maxBound;
}
```

Determines if a value is within or outside the specified bounds. Supports two modes, one for being inside the range, the other for being outside the range.

Uniform saturation to a specific value

```
static void adjustSaturationTo(cv::Mat &image, const double &alpha, const double &beta)
{
    cv::Mat result;

    //Split the image into its color channels
    std::vector<cv::Mat> channels;
    cv::split(image, channels);

    //Apply brightness adjustment to each color channel
    for (int i = 0; i < channels.size(); ++i)
    {
        channels[i].convertTo(channels[i], -1, alpha, beta);
        cv::equalizeHist(channels[i], channels[i]);
    }

    //Merge the color channels back into a single image
    cv::merge(channels, result);

    //Set the result as the new image
    image = result;
}
```

Adjusts the saturation of any image to a certain uniform value, darkening bright images, and brightening dark images.

Reversing a QList of generic type T

```
template <typename T>
static QList<T> reverse(const QList<T> &listIn)
{
    //Declare the output list, with the input list size
    QList<T> listOut;
    listOut.reserve(listIn.size());
    //Copy the listIn into listOut in reverse
    std::reverse_copy(listIn.begin(), listIn.end(), std::back_inserter(listOut));
    return listOut;
}
```

Creating a reversed order QList<?>

Resistor band color identification

```
static QString identifyBandColor(QColor bandColor)
{
    const double value = bandColor.valueF();
    if(value < 0.25)
        return BLACK;

    const int hue = bandColor.hue();
    const double saturation = bandColor.saturationF();

    if(hue > 10)
    {
        if(hue < 30)
        {
            if(saturation < 0.3)
                return GOLD;
            else if(value < 0.6)
                return BROWN;
            else
                return ORANGE;
        }
        else if(hue < 50)
        {
            if(saturation < 0.5 || value < 0.6)
                return GOLD;
            else
                return ORANGE;
        }
        else if(hue <= 85)
        {
            if(saturation < 0.75 || value < 0.5)
                return GOLD;
            else
                return YELLOW;
        }
        else if(hue <= 140)
            return GREEN;
    }

    if(hue > 320 || hue <= 10)
    {
        if(hue > 320)
            return RED;
        else
        {
            if(saturation > 0.666 && value > 0.666)
                return RED;
            return BROWN;
        }
    }

    if(hue > 180 && hue < 290)
    {
        if(hue <= 245)
            return BLUE;
        else
            return VIOLET;
    }
    return "unknown";
}
```

Determines the name of the color, judging by the Hue, Saturation and Value properties.

Decoding bands

```
static QList<QString> identifyBandColors(QList<QColor> bands)
{

    QList<QString> stringCodedBands;
    if(bands.length() == 0)
        return stringCodedBands;
    for(const QColor color : bands)
    {
        stringCodedBands.push_back(identifyBandColor(color));
    }
    //We're looking at the resistor backwards
    if(stringCodedBands[0] == GOLD || stringCodedBands[0] == WHITE)
        stringCodedBands = reverse(stringCodedBands);
    return stringCodedBands;
}

static int decodeBand(QColor bandColor)
{
    return decodeBand(identifyBandColor(bandColor));
}

static int decodeBand(QString bandColor)
{
    //Converts color to the number it represents
    return RESISTOR_COLOR_CODES[bandColor];
}

static qreal decodeResistorBands(QList<QColor> bands)
{
    return decodeResistorBands(identifyBandColors(bands));
}

static qreal decodeResistorBands(QList<QString> bands)
{
    int ohms = 0;
    const int length = bands.length();
    //No bands is unreadable. Return 0.
    if(length == 0)
        return ohms;
    //We're looking at the resistor backwards. Reverse the bands.
    if(bands[0] == GOLD || bands[0] == WHITE)
        bands = reverse(bands);

    //If tolerance band is present, do not process it.
    const bool toleranceBandPresent = bands[length-1] == GOLD || bands[length-1] == WHITE;
    const int valueBandsLength = toleranceBandPresent ? length - 1 : length;

    //For all the bands that are not the tolerance band
    for(int i = 0; i < valueBandsLength; ++i)
    {
        const QString bandColor = bands[i];
        //The power indicates if the value is units, tens, hundreds, etc...
        //Works for both 3 and 4 band (and more) band resistors
        const int power = valueBandsLength-2-i;
        const int bandValue = decodeBand(bandColor);
        //Determines if value band, or magnitude band
        if(power >= 0)
            ohms += pow(10, power) * bandValue; //example for red red brown, or 2 2 1: 20 + 2
        else
            ohms *= pow(10, bandValue); //example continuation: 22 * 10^1 = 220Ohm
    }

    return ohms;
}
```

Decodes the Ohm value from a List of either String (color name) or QColor (color) objects.

Determining if a pixel essentially has zero information (is entirely black)

```
static bool isBlack(const Vec3b &pixel)
{
    return pixel[0] == 0 && pixel[1] == 0 && pixel[2] == 0;
}
```

Figure 126: Code: isBlack method

Determines if the pixel is completely black by comparing each of the channels to 0.

Counting non-zero|black pixels

```
static QList<int> getRowNonBlackCount(Mat inputMat)
{
    return getRowNonBlackCount(inputMat, 0, inputMat.cols);
}

static QList<int> getRowNonBlackCount(Mat inputMat, int x1, int x2)
{
    QList<int> rowsCount;
    for(int y = 0; y < inputMat.rows; ++y)
    {
        int count = 0;
        //Bounds optimisation
        for(int x = x1; x < x2; ++x)
        {
            const Vec3b &bgr = inputMat.at<Vec3b>(Point(x, y));
            //Count every pixel that is not 0x000000 (pure black)
            if(!isBlack(bgr))
                ++count;
        }
        rowsCount.push_back(count);
    }
    return rowsCount;
}
```

Figure 127: Code: getRowNonBlackCount method

Counts the number of pixels that are not black in a row, for every row. Optionally supports custom horizontal bounds, outside of which pixels are ignored from the count.

Getting the mean hue of non-zero|black pixels in every row

```
static QList<double> getRowMeanHue(Mat inputMat)
{
    return getRowMeanHue(inputMat, 0, inputMat.cols);
}

static QList<double> getRowMeanHue(Mat inputMat, int x1, int x2)
{
    QList<double> rowsCount;
    for(int y = 0; y < inputMat.rows; ++y)
    {
        double totalHue = 0;
        int count = 0;
        //Bounds optimisation
        for(int x = x1; x < x2; ++x)
        {
            const Vec3b &bgr = inputMat.at<Vec3b>(Point(x, y));
            const QColor color = QColor(bgr[2], bgr[1], bgr[0]);
            const double hueF = color.hueF();
            if(hueF < 0)
                continue;
            if(isBlack(bgr))
                continue;
            //Sum hue if not black
            totalHue += hueF;
            ++count;
        }
        //Push the mean value of this row
        rowsCount.push_back(count > 0 ? totalHue/(double) count : 0);
    }
    return rowsCount;
}
```

Figure 128: Code: getRowMeanHue method

Gets the mean hue of all the rows of an image, skipping black pixels. Optionally supports custom horizontal bounds, outside of which pixels are ignored from the count.

Getting the mean color of every row

```
static QList<QColor> getRowMeanColor(Mat inputMat, int x1, int x2)
{
    QList<QColor> rowsCount;
    for(int y = 0; y < inputMat.rows; ++y)
    {
        double r = 0;
        double g = 0;
        double b = 0;
        int count = 0;
        //Bounds optimisation
        for(int x = x1; x < x2; ++x)
        {
            const Vec3b &bgr = inputMat.at<Vec3b>(Point(x, y));
            const QColor color = QColor(bgr[2], bgr[1], bgr[0]);
            const double hueF = color.hueF();
            if(hueF < 0)
                continue;
            if(isBlack(bgr))
                continue;
            r += color.redF();
            g += color.greenF();
            b += color.blueF();
            ++count;
        }
        rowsCount.push_back(count > 0 ?
                                QColor::fromRgbF(r/(double) count,
                                                 g/(double) count,
                                                 b/(double) count) :
                                Util::BLACK_COLOR);
    }
    return rowsCount;
}
```

Figure 129: Code: getRowMeanColor method

Retrieves the mean color of every row of an image. Optionally supports custom horizontal bounds, outside of which pixels are ignored from the count.

Averaging colors|blurring

```
static Mat blur(Mat inputMat, const int blurAmount = 1)
{
    return blur(inputMat, blurAmount, 0, 0, inputMat.cols, inputMat.rows);
}

static Mat blur(Mat inputMat, const int blurAmount, const int x1, const int y1, const int x2, const int y2)
{
    const int rows = inputMat.rows;
    const int cols = inputMat.cols;
    Mat outputMat = inputMat.clone();

    //Bounds optimisation
    for(int row = y1; row < y2; ++row)
    {
        for(int col = x1; col < x2; ++col)
        {
            //Ignore if the pixel is black (optimisation, unlikely to contain information)
            if(isBlack(inputMat.at<Vec3b>(Point(col, row))))
                continue;
            const int y1 = max(0, row-blurAmount);
            const int y2 = min(rows, row+blurAmount);

            const int x1 = max(0, col-blurAmount);
            const int x2 = min(cols, col+blurAmount);
            double total = ((y2 - y1) + 1) * ((x2 - x1) + 1);
            int totalR = 0;
            int totalG = 0;
            int totalB = 0;

            //Get the average of the surrounding colors in blurAmount range from this pixel
            for(int y = y1; y <= y2; ++y)
            {
                for(int x = x1; x <= x2; ++x)
                {
                    const Vec3b &bgr = inputMat.at<Vec3b>(Point(x, y));
                    totalR += bgr[2];
                    totalG += bgr[1];
                    totalB += bgr[0];
                }
            }
            Vec3b &bgr = outputMat.at<Vec3b>(Point(col, row));
            bgr[2] = totalR / total;
            bgr[1] = totalG / total;
            bgr[0] = totalB / total;
        }
    }
    return outputMat;
}
```

Figure 130: Code: blur method

Averages pixel values according to the neighboring values, in a specified range. Optionally supports custom horizontal and vertical bounds, outside of which pixels are ignored from the count.

More versatile versions of opencv::imshow

```
static void betterShow(const Mat inputMat, const QString name, const double scale = 1)
{
    betterShow(inputMat, name.toStdString(), scale);
}

static void betterShow(const Mat inputMat, const std::string name, const double scale = 1)
{
    namedWindow(name, cv::WINDOW_NORMAL);
    resizeWindow(name, inputMat.cols*scale, inputMat.rows*scale);
    imshow(name, inputMat);
}
```

Figure 131: Code: betterShow method

Mat preparation for post-processing

```
static Mat prepareMatForProcessing(Mat inputMat, const qreal degsOffset = 0, const bool display = false)
{
    //Adjust the brightness to a uniform value.
    Util::adjustSaturationTo(inputMat, 0.1, 100);
    const Size imageSize = inputMat.size();
    const double imageWidth = inputMat.cols;
    const double midX = imageWidth/2.0;

    const double imageHeight = inputMat.rows;
    const double midY = imageHeight/2.0;

    QPointF midPos = QPointF(midX, midY);
    const QPointF origin = QPointF(0, 0);

    Mat grey;
    cvtColor(inputMat, grey, cv::COLOR_BGR2GRAY);

    Mat bw;
    threshold(grey, bw, 50, 255, cv::THRESH_BINARY | cv::THRESH_OTSU);

    //Identify the contours of the component
    vector<vector<Point>> contours;
    findContours(bw, contours, cv::RETR_LIST, cv::CHAIN_APPROX_NONE);

    double highestArea = 0;
    double orientation = 0;
    for (int i = 0; i < contours.size(); i++)
    {
        vector<Point> contour = contours[i];
        double area = cv::contourArea(contour);
        //Skip pieces that are too small (noise prevention)
        if (area < 2000/* || area > 10000*/)
            continue;
        if(display)
        {
            drawContours(inputMat, contours, i, cv::Scalar(255, 0, 255), 2);
        }
        //Find angle of rotation.
        const double countourOrientation = Util::getOrientation(contour, inputMat, 0);
        if(highestArea < area)
        {
            highestArea = area;
            orientation = countourOrientation;
        }
    }

    //Rotate the component, expanding the Mat to fit the entire before-rotation image (fill gaps with black).
    const cv::Point2f centerPoint(midX, midY);
    //Consider the degrees offset argument.
    const double finalDegs = Util::toDegs(orientation) + degsOffset;
    //Create the bounding box from a RotatedRectangle.
    const Rect2f boundingBox = cv::RotatedRect(cv::Point2f(), imageSize, finalDegs).boundingRect2f();
    //Create and populate the rotation Mat.
    cv::Mat rotMat = cv::getRotationMatrix2D(centerPoint, finalDegs, 1.0);
    rotMat.at<double>(0,2) += boundingBox.width/2.0 - imageWidth/2.0;
    rotMat.at<double>(1,2) += boundingBox.height/2.0 - imageHeight/2.0;
    //Rotate the Mat using he rotation Mat.
    warpAffine(inputMat, inputMat, rotMat, boundingBox.size());

    return inputMat;
}
```

Figure 132: Code: prepareMatForProcessing method

Adjusts the colors and orientation of the component, making the input images more uniform for post-processing.

Band extraction from opencv::Mat

```
static QList<QColor> getOhmBands(Mat inputMat, const qreal horizontalBoundPos = 0.7,
                                 const qreal verticalBoundPos = 0.75,
                                 const qreal middleCropBoundPos = 0.125,
                                 const qreal minHue = 0, const qreal maxHue = 1,
                                 const qreal minSat = 0.35, const qreal maxSat = 1,
                                 const qreal minVal = 0, const qreal maxVal = 1,
                                 const bool display = false, QString displayName = "")
{
    //Crop out the glare region
    const int cropFromMid = inputMat.cols * (cap(middleCropBoundPos, 0, 1)/2.) - 1;
    const Rect leftRect(0, 0, (inputMat.cols/2) - cropFromMid, inputMat.rows);
    const Rect rightRect(inputMat.cols/2 + cropFromMid, 0, (inputMat.cols/2) - cropFromMid, inputMat.rows);
    hconcat(Mat(inputMat, leftRect), Mat(inputMat, rightRect), inputMat);
    QList<QColor> ohmBands;

    const double imageWidth = inputMat.cols;
    const double imageHeight = inputMat.rows;
    const double lineWidth = 20;

    //Vertical Sampling
    const double rectHeight = imageHeight*cap(verticalBoundPos, 0, 1);
    const double x1 = cap(horizontalBoundPos, 0, 1) * ((imageWidth-lineWidth)/2.);
    const double x2 = imageWidth - x1;
    const int rectWidth = x2 - x1;

    const double y1 = (imageHeight - rectHeight) / 2.;
    const double y2 = y1 + rectHeight;

    //Blur a bit to reduce noise
    inputMat = Util::blur(inputMat, 1, x1, y1, x2, y2);
    //Apply the HSV filter
    Util::hsvFilter(inputMat, minHue, maxHue, minSat, maxSat, minVal, maxVal, Util::BLACK_COLOR, false, x1, y1, x2, y2);

    const int rowCountThreshold = rectWidth*0.25;
    //Gather rows data
    QList<int> rowsCount = Util::getRowNonBlackCount(inputMat, x1, x2);
    QList<QColor> colorMeans = Util::getRowMeanColor(inputMat, x1, x2);

    //Initialise bands scan
    int startY = 0;
    bool bandStarted = false;
    double bandMeanHue;

    //Scasn all the rows for bands
    for(int y = y1; y < y2; ++y)
    {
        //Decision if currently inside a band, or not.
        const QColor rowMeanColor = colorMeans[y];
        const double rowMeanHue = rowMeanColor.hueF();
        const int rowCount = rowsCount[y];
        //if current row is identified as a band
        if(rowCount > rowCountThreshold)
        {
            if(!bandStarted)
            {
                bandMeanHue = 0;
                startY = y;
            }
            bandMeanHue += rowMeanHue;
            bandStarted = true;
            continue;
        }
        //If a band was not started, scan the next row.
        if(!bandStarted)
            continue;
        //A band has ended. startY is the start of this band.
        bandStarted = false;

        const int height = y - startY;
        const int midY = (y + startY)/2;
        bandMeanHue /= height;

        //If band is tall enough (noise prevention)
        if(height >= 3)
        {
            const QColor midColor = colorMeans[midY];
            //Display information if display is enabled
            if(display)
            {
                const Scalar scalar = Util::toScalar(midColor);
                rectangle(inputMat, Rect(0, startY, imageWidth, height), scalar, 2);
            }
            ohmBands.push_back(midColor);
            //Skip ahead because bands have gaps.
            y += 16;
        }
    }
    //Display information if display is enabled
    if(display)
    {
        rectangle(inputMat, Rect(x1-1, 0, 2, imageHeight), Util::BEST_SCALAR, 2);
        rectangle(inputMat, Rect(x2-1, 0, 2, imageHeight), Util::BEST_SCALAR, 2);

        rectangle(inputMat, Rect(0, y1-1, imageWidth, 2), Util::BEST_SCALAR, 2);
        rectangle(inputMat, Rect(0, y2-1, imageWidth, 2), Util::BEST_SCALAR, 2);

        Util::betterShow(inputMat, displayName, 3);
    }
    return ohmBands;
}
```

Figure 133: Code: getOhmBands method

Retrieves a list of QColor that represents resistor bands found in the image. Optionally draws the output for debugging/demonstration purposes.

Generic map of type QMap<?, int> increment method

```
template<typename KEY>
static int increment(QMap<KEY, int> *map, const KEY key)
{
    const int value = map->contains(key) ?
                      map->value(key) + 1 :
                      1;
    map->insert(key, value);
    return map->value(key);
}
```

Figure 134: Code: increment method

Increments values of any keyed, int value QMap. Used in counting classes.

Exporting a QGraphicsScene to file

```
static void exportSceneToFile(QGraphicsScene &scene, QString name = getFormattedDate(), QString extension = "png")
{
    if(name.length() == 0)
        name = Util::getFormattedDate();

    //Export the scene as an image
    QImage image(scene.width(), scene.height(), QImage::Format_ARGB32);
    image.fill(Qt::transparent);
    QPainter painter(&image);
    scene.render(&painter);

    //Save image
    image.save(name + "." + extension);
}
```

Figure 135: Code: exportSceneToFile method

Exports a QGraphicsScene to a file. Optionally, name and extension can be specified. Default values are timestamp for the name, and .png for the extension.

Safer and more convenient opencv::Rect “constructor”

```
static Rect boundedRect(const qreal &x, const qreal &y, const qreal &width, const qreal &height,
                        const qreal minX = 0, const qreal minY = 0, const qreal maxX = CAMERA_WIDTH,
                        const qreal maxY = CAMERA_HEIGHT)
{
    return Rect
        (
            max(x, minX),
            max(y, minY),
            min(maxX - x, width),
            min(maxY - y, height)
        );
}
```

Figure 136: Code: boundedRect method

Creates a new opencv::Rect, with default values that are the size of the frames being retrieved from the camera.

Angle manipulation helper methods

```
static double toDegs(const double &rads)
{
    return qRadiansToDegrees(rads);
}

static double toRads(const double &degs)
{
    return qDegreesToRadians(degs);
}

static double getAngleDegs(const double &p1x, const double &p1y, const double &p2x, const double &p2y)
{
    return toDegs(getAngleRads(p1x, p1y, p2x, p2y));
}

static double getAngleRads(const double &p1x, const double &p1y, const double &p2x, const double &p2y)
{
    return -atan2(p2y * p1x - p2x * p1y, p1x * p2x + p1y * p2y);
}

static QPointF offsetVec2InRadius(double x, double y, const double &radius, const double &degs)
{
    const double rads = toRads(degs);
    return QPointF(x + radius * qSin(rads), y + radius * qCos(rads));
}

static QPointF offsetVec2InRadius(QPointF &originPos, const double &radius, const double &degs)
{
    return offsetVec2InRadius(originPos.x(), originPos.y(), radius, degs);
}
```

Figure 137: Code: angle manipulation methods

Conversion between Radians and Degrees, angle determining, offsetting Points from an origin, in a certain direction.

Random number generator helper methods

```
static int genRand(const int &min, const int &max = 0)
{
    if(min == max)
        return min;
    else if(max > min)
        return QRandomGenerator::global()->bounded(min, max + 1);
    else
        return QRandomGenerator::global()->bounded(max, min + 1);
}

static double genRandDouble(const double &min, const double &max = 0)
{
    if(min == max)
        return min;
    else if(max > min)
        return min + QRandomGenerator::global()->bounded(max - min);
    else
        return max + QRandomGenerator::global()->bounded(min - max);
}

static QPoint genRandQPoint(const cv::Rect &area)
{
    return genRandQPoint(QPoint(area.tl().x, area.tl().y), QPoint(area.br().x, area.br().y));
}

static Point genRandPoint(const cv::Rect &area)
{
    return genRandPoint(Point(area.tl().x, area.tl().y), Point(area.br().x, area.br().y));
}

static QPoint genRandQPoint(const cv::Mat &mat)
{
    return genRandQPoint(QPoint(0, 0), QPoint(mat.cols, mat.rows));
}

static Point genRandPoint(const cv::Mat &mat)
{
    return genRandPoint(Point(0, 0), Point(mat.cols, mat.rows));
}

static QPoint genRandQPoint(const QPoint &p1, const QPoint &p2)
{
    return QPoint(p1.x() + genRand(p2.x() - p1.x()), p1.y() + genRand(p2.y() - p1.y()));
}

static Point genRandPoint(const Point &p1, const Point &p2)
{
    return Point(p1.x + genRand(p2.x - p1.x), p1.y + genRand(p2.y - p1.y));
}

static QPoint genRandQPointOffset(const QPoint &p, const int &offset)
{
    return QPoint(p.x() + genRand(-offset, offset), p.y() + genRand(-offset, offset));
}

static Point genRandPointOffset(const Point &p, const int &offset)
{
    return Point(p.x + genRand(-offset, offset), p.y + genRand(-offset, offset));
}
```

Figure 138: Code: Random Number Generation method

Generation of random integers, doubles QPoints and Points, in specific ranges/areas.

Mapping and Distance related helper methods

```
static double map(const double &x, const double &in_min, const double &in_max, const double &out_min, const double &out_max)
{
    return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}

static double mapCapped(const double &x, const double &in_min, const double &in_max, const double &out_min, const double &out_max)
{
    return cap((x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min, in_min, in_max);
}

static double cap(const double &x, const double &min_bound, const double &max_bound)
{
    if(max_bound > min_bound)
        return max(min_bound, min(max_bound, x));
    else
        return max(max_bound, min(min_bound, x));
}

static double getDistance(Point &p1, Point &p2)
{
    return qSqrt(qPow(p2.x - p1.x, 2) + qPow(p2.y - p1.y, 2));
}

static double getDistance(QPoint &p1, QPoint &p2)
{
    return qSqrt(qPow(p2.x() - p1.x(), 2) + qPow(p2.y() - p1.y(), 2));
}
```

Figure 139: Code: Mapping and Distance methods

Mapping a value from one range to another range, capping a value between bounds, and getting distance between two points methods.

Conversion helper methods

```
static QPixmap matToPixmap(const Mat &openCVMat)
{
    return QPixmap::fromImage(QImage(static_cast<const unsigned char*>(openCVMat.data), openCVMat.cols, openCVMat.rows, static_cast<int>(openCVMat.step), QImage::Format_RGB888).rgbSwapped());
}

static Mat pixmapToMat(const QPixmap &QtPixmap)
{
    QImage _tempImage = pixmapToQImage(QtPixmap);

    Mat _tempMat = Mat(_tempImage.height(), _tempImage.width(), CV_8UC4, const_cast<uchar*>(_tempImage.bits()), static_cast<size_t>(_tempImage.bytesPerLine()));
    cvtColor(_tempMat, _tempMat, COLOR_BGRA2BGR);

    return _tempMat;
}

static QImage pixmapToQImage(const QPixmap &QtPixmap)
{
    return QtPixmap.toImage();
}
```

Zoom: 100%

Figure 140: Code: QPixMap and QImage conversion methods

Converts in-between QPixMap and Qimage, for various purposes.

DetectionGraphic paint method

```
void DetectionGraphic::paint(QPainter *painter, const QStyleOptionGraphicsItem *option, QWidget *widget)
{
    QRectF bound = boundingRect();
    QPainterPath path;
    //Pen (outline) uses bounding box color
    QPen pen = QPen(QColor(_detection.color[0], _detection.color[1], _detection.color[2]));
    //Brush (fill) uses 1/8th of bounding box color
    QBrush brush = QBrush(QColor(_detection.color[0]/8, _detection.color[1]/8, _detection.color[2]/8));
    pen.setWidth(5);
    painter->setPen(pen);
    painter->setBrush(brush);

    //Draw the box
    painter->drawPolyline(bound);

    //Text
    QFont font = painter->font();
    font.setPointSizeF(_fontSize);
    painter->setFont(font);
    QString text = _detection.className + " " + QString::number(_detection.confidence).mid(0, 4) + " " + _detection.extra;

    pen.setWidth(10);
    const qreal halfBoundWidth = bound.width()/2.0;
    //result of mapping 0 to CAMERA_WIDTH range, to 0 to 1 range.
    const qreal horizontalPos = (bound.left()+halfBoundWidth) / CAMERA_WIDTH;
    const qreal estimatedTextWidth = _fontSize*text.length()*0.7;
    //Text starts from left bound of box,
    //If the text length is more than the width of the bounding box
    //Offset the starting position until the text ends at the end of
    //the bounding box scaled by horizontalPos
    const qreal leftBound = bound.left() - max(0.0, estimatedTextWidth - bound.width())*horizontalPos;
    //Convert text to path
    path.addText(QPointF(leftBound, bound.top() - font.pointSizeF()/2.0), font, text);
    //Outline the text path
    painter->strokePath(path, pen);
    //Fill the text inside
    painter->fillPath(path, brush);
}
```

Figure 141: Code: DetectionGraphic paint override method

The code responsible for drawing the text of the Detections on the Inference output view.

Inference setup

```
void Inference::loadClassesFromFile()
{
    std::ifstream inputFile(classesPath.toStdString());
    if (inputFile.is_open())
    {
        std::string classLine;
        while (std::getline(inputFile, classLine))
            classes.push_back(QString::fromStdString(classLine));
        inputFile.close();
    }
}

void Inference::loadOnnxNetwork()
{
    net = cv::dnn::readNetFromONNX(modelPath.toStdString());
    if (cudaEnabled)
    {
        std::cout << "\nRunning on CUDA" << std::endl;
        net.setPreferableBackend(cv::dnn::DNN_BACKEND_CUDA);
        net.setPreferableTarget(cv::dnn::DNN_TARGET_CUDA);
    }
    else
    {
        std::cout << "\nRunning on CPU" << std::endl;
        net.setPreferableBackend(cv::dnn::DNN_BACKEND_OPENCV);
        net.setPreferableTarget(cv::dnn::DNN_TARGET_CPU);
    }
}
```

Figure 142: Code: Inference classes loading and ONNX network initialisation

Reads in the Inference classes, and starts the ONNX Network, running either on CUDA, or CPU cores.

Inference running: YoloV8

```
QVector<Detection> Inference::runInference(const cv::Mat &input)
{
    cv::Mat modelInput = input;
    cv::Mat blob;

    cv::dnn::blobFromImage(modelInput, blob, 1.0/255.0, modelShape, cv::Scalar(), true, false);
    net.setInput(blob);

    std::vector<cv::Mat> outputs;
    net.forward(outputs, net.getUnconnectedOutLayersNames());

    int rows = outputs[0].size[1];
    int dimensions = outputs[0].size[2];

    bool yolov8 = false;
    //YOLOv5 has an output of shape (batchSize, 25200, 85) (Num classes + box[x,y,w,h] + confidence[c])
    //YOLOv8 has an output of shape (batchSize, 84,  8400) (Num classes + box[x,y,w,h])
    if (dimensions > rows) //Check if the shape[2] is more than shape[1] (yolov8)
    {
        yolov8 = true;
        rows = outputs[0].size[2];
        dimensions = outputs[0].size[1];

        outputs[0] = outputs[0].reshape(1, dimensions);
        cv::transpose(outputs[0], outputs[0]);
    }
    float *data = (float *)outputs[0].data;

    float x_factor = modelInput.cols / modelShape.width;
    float y_factor = modelInput.rows / modelShape.height;

    std::vector<int> class_ids;
    std::vector<float> confidences;
    std::vector<cv::Rect> boxes;

    for (int i = 0; i < rows; ++i)
    {
        if (yolov8)
        {
            float *classes_scores = data+4;

            cv::Mat scores(1, classes.size(), CV_32FC1, classes_scores);
            cv::Point class_id;
            double maxClassScore;

            minMaxLoc(scores, 0, &maxClassScore, 0, &class_id);

            if (maxClassScore > modelScoreThreshold)
            {
                confidences.push_back(maxClassScore);
                class_ids.push_back(class_id.x);

                float x = data[0];
                float y = data[1];
                float w = data[2];
                float h = data[3];

                int left = int((x - 0.5 * w) * x_factor);
                int top = int((y - 0.5 * h) * y_factor);

                int width = int(w * x_factor);
                int height = int(h * y_factor);

                boxes.push_back(Util::boundedRect(left, top, width, height));
            }
        }
        else //yolov5
```

Figure 143: Code: Running YoloV8 Inference

Inference running: YoloV5

```
    else //yolov5
    {
        float confidence = data[4];

        if (confidence >= modelConfidenseThreshold)
        {
            float *classes_scores = data+5;

            cv::Mat scores(1, classes.size(), CV_32FC1, classes_scores);
            cv::Point class_id;
            double max_class_score;

            minMaxLoc(scores, 0, &max_class_score, 0, &class_id);

            if (max_class_score > modelScoreThreshold)
            {
                confidences.push_back(confidence);
                class_ids.push_back(class_id.x);

                float x = data[0];
                float y = data[1];
                float w = data[2];
                float h = data[3];

                int left = int((x - 0.5 * w) * x_factor);
                int top = int((y - 0.5 * h) * y_factor);

                int width = int(w * x_factor);
                int height = int(h * y_factor);

                boxes.push_back(Util::boundedRect(left, top, width, height));
            }
        }
    }

    data += dimensions;
}
```

Figure 144: Code: Running YoloV5 Inference

Both YoloV5 and YoloV8 results are achieved through roughly the same approach, with minor variations between the generations.

Processing Inference results into QList of Detection structs

```
std::vector<int> nms_result;
cv::dnn::NMSBoxes(boxes, confidences, modelScoreThreshold, modelNMSThreshold, nms_result);

QList<Detection> detections{};
for (unsigned long i = 0; i < nms_result.size(); ++i)
{
    int idx = nms_result[i];

    Detection result;
    result.classId = class_ids[idx];
    result.className = classes[result.classId];
    result.confidence = confidences[idx];

    //Determine class color by using the id as a RNG seed.
    std::mt19937 gen(result.classId);
    std::uniform_int_distribution<int> dis(100, 255);
    result.color = cv::Scalar(dis(gen),
                              dis(gen),
                              dis(gen));
    result.box = boxes[idx];
    result.mat = Mat(input, result.box);

    //Post-processing
    if(result.className == Util::RESISTOR)
    {
        Mat preparedMat = Util::prepareMatForProcessing(result.mat, 90);
        const QList<QColor> bandColors = Util::getOhmBands(preparedMat);
        const QList<QString> bands = Util::identifyBandColors(bandColors);
        const qreal ohmValue = Util::decodeResistorBands(bands);
        if(ohmValue > 0)
            result.extra += QString::number(ohmValue) + " Ohms";
    }
    else if(result.className == Util::DC_CAPACITOR)
    {
        Mat prepared = Util::prepareMatForProcessing(result.mat);
        result.extra = Util::performOCR(prepared);
        Util::betterShow(prepared, (QString) "capacitor");
    }
    detections.push_back(result);
}

return detections;
```

Figure 145: Code: Gathering Inference results into Detection structs list

Conversion of Inference results format to QList<Detection> format, which the application uses, and post-processing of Resistor color bands, and OCR progress for Capacitor text reading.

App initialisation

```
MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    _connectCamera();
    _frameUpdater.setInterval(_frameIntervalMs);
    qDebug() << _frameIntervalMs;
    #ifdef FRAME_PRINT
    _frameCountPrinter.setInterval(FRAME_PRINT_EVERY_MS);
    QObject::connect(&_frameCountPrinter, &QTimer::timeout, this, &MainWindow::_printFramerate);
    #endif
    #ifdef CONNECTION_MANAGER
    _cameraConnectionManager.setInterval(1000);

    _outputGraphicsView = QSharedPointer<QGraphicsView>(ui->outputGraphicView);
    _outputGraphicsView->minimumSize().setWidth(CAMERA_WIDTH);
    _outputGraphicsView->minimumSize().setHeight(CAMERA_HEIGHT);
    _outputGraphicsView->size().setWidth(CAMERA_WIDTH);
    _outputGraphicsView->size().setHeight(CAMERA_HEIGHT);
    _outputGraphicsView->setFrameShape(QFrame::NoFrame);
    _outputGraphicsView->setRenderHints(QPainter::Antialiasing | QPainter::LosslessImageRendering | QPainter::TextAntialiasing);

    if(!_vCap.isOpened())
    {
        print("ERROR! Camera not found! Please connect the camera to continue.");
        _frameUpdater.stop();

        #ifdef FRAME_PRINT
        _frameCountPrinter.stop();
        #endif

        _cameraConnectionManager.start();

    }
    QObject::connect(&_cameraConnectionManager, &QTimer::timeout, this, &MainWindow::_connectCamera);
    #endif

    QObject::connect(&_frameUpdater, &QTimer::timeout, this, &MainWindow::_onFrame);
}
```

Figure 146: Code: Initialising the Application

When the application launches, it gets initialised.

- The connection/frame retrieval (if camera connected successfully)/frames-per-second print related threads start running.
- The display window size is adjusted to match the camera.

Frame Processing

```
void MainWindow::_onFrame()
{
    //Wait for a new frame from camera and store it into 'frame'
    _vCap.read(_frameMat);
    //Check if we succeeded
    if (_frameMat.empty())
    {
        print("ERROR! Camera disconnected");
        _frameUpdater.stop();
        destroyWindow(_cameraWindowName);

        #ifdef CONNECTION_MANAGER
        _cameraConnectionManager.start();
        _vCap.release();
        #endif
        return;
    }

    _detections.clear();
    _detectionsByClass.clear();
    _frameScene.clear();

    //FPS Count
    ++_frameCountTotal;
    ++_frameCount;

    #ifdef FRAME_PRINT
    if(_frameCountTotal % 10 == 0)
        qDebug() << "INFO: Retrieving frame..." << _frameCountTotal;
    #endif

    //Clear info box
    ui->infoTextBox->clear();

    if(_inferenceEnabled)
    {
        //Run Inference
        _runInference(_frameMat);
    }

    const int matSize = _frameMat.cols;

    //Crop to specified zoom/pos values
    const double scale = 1 - (ui->zoomSlider->value()/SLIDER_RANGE);
    const double horizontalWidthPos = ui->horizontalZoomPosSlider->value()/SLIDER_RANGE;
    const double verticalWidthPos = 1.0-(ui->verticalZoomPosSlider->value()/SLIDER_RANGE);
    _frameMat = Util::cropShrink(_frameMat, scale, horizontalWidthPos, verticalWidthPos);

    //Convert to Pixmap
    _framePixmap = Util::matToPixmap(_frameMat);
    _frameScene.addPixmap(_framePixmap);

    //Add detections to the Scene
    const int detectionCount = _detections.length();
    for(int i = 0; i < detectionCount; ++i)
    {
        _frameScene.addItem(new DetectionGraphic(_detections[i]));
    }

    //Set output
    _outputGraphicsView->fitInView(0, 0, matSize, matSize, Qt::KeepAspectRatio);
    _outputGraphicsView->setScene(&_frameScene);
    _outputGraphicsView->update();
    _outputGraphicsView->show();
}
```

Figure 147: Code: Processing the frames

If a frame was not retrievable, assumes the camera got disconnected, and starts the camera connection thread. Otherwise, runs Inference on the retrieved frame, and displays the results.

Camera initialisation and connection

```
void MainWindow::_connectCamera()
{
    #ifdef CONNECTION_PRINT
    print("INFO: Attempting connection... "/* + QString((_vCap.isOpened() ? "open" : "closed"))*/);
    #endif
    if(!_vCap.isOpened())
    {
        _vCap.open(_cameraPath, _apiRef);
        if(_vCap.isOpened())
        {
            _initCamera();
            print("INFO: Camera connected");
            _frameUpdater.start();

            #ifdef FRAME_PRINT
            _frameCountPrinter.start();
            _frameCount = 0;
            #endif

            #ifdef CONNECTION_MANAGER
            _cameraConnectionManager.stop();
            #endif
        }
        else
            _frameUpdater.stop();
    }
}


void MainWindow::_initCamera()
{
    _vCap.set(cv::CAP_PROP_FRAME_HEIGHT, CAMERA_HEIGHT);
    _vCap.set(cv::CAP_PROP_FRAME_WIDTH, CAMERA_WIDTH);
    _vCap.set(cv::CAP_PROP_FPS, 24);
}
```

Figure 148: Code: Initialising and connecting the Camera

If the camera is not connected, attempts to connect the camera. If connection is successful, stops the camera connection manager thread, and starts the frame retrieval thread.

Running inference and organising the Detections

```
void MainWindow::_runInference(const Mat &inputMat)
{
    _detectionCountMap->clear();
    for(Detection detection : _inf.runInference(inputMat))
    {
        _detections.push_back(detection);
    }

    const int detectionsCount = _detections.size();
    _appendInfoBox("Detection count: " + QString::number(detectionsCount) + "\n");

    for(Detection detection : _detections)
    {
        //Categorise each detection by class name
        if(!_detectionsByClass.contains(detection.className))
            _detectionsByClass.insert(detection.className, {});
        _detectionsByClass[detection.className].push_back(detection);
    }

    for(const QString &key : _detectionsByClass.keys())
    {
        _appendInfoBox(key + ": " + QString::number(_detectionsByClass.value(key).length()));
    }

    _appendInfoBox();

    for(const auto detection : _detections)
    {
        _appendInfoBox(QString::number(detection.confidence).mid(0, 4) + " " + detection.className + " " + detection.extra);
    }
}
```

Figure 149: Code: Running inference and organising the Detections

Runs Inference and organises any potential Detections by their class id, appending the count information to the top display box.

Exporting the current frame with Inference results

```
void MainWindow::exportImage(const bool &verbose)
{
    const QString date = Util::getFormattedDate();
    const QString dirPath = "./export/" + date;
    QDir().mkpath(dirPath);
    Util::exportSceneToFile(_frameScene, dirPath + "/" + "final");
    cv::imwrite(QString(dirPath + "/" + "raw.png").toStdString(), _frameMat);
    if(verbose)
        print(dirPath + " exported");

    for(auto className : _detectionsByClass.keys())
    {
        const QList<Detection> detections = _detectionsByClass[className];
        const int length = detections.length();
        for(int i = 0; i < length; ++i)
        {
            Detection detection = detections[i];
            cv::imwrite(QString(dirPath + "/" + "crop_" + className + "_" + QString::number(i+1) + ".png").toStdString(), detection.mat);
        }
        print("    " + QString::number(length) + "x" + className);
    }
}
```

Figure 150: Code: Exporting the current frame with Inference data

Exports the raw and processed images, as well as all the detected by inference items separately, into the /exports/[date]/ directory.