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
29th of January, 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 [2]

A filtering technique used on predictions of object detectors.

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

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

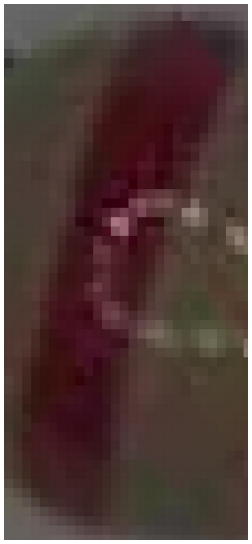

Figure 3: 220Ohm resistor: Single band pixel close up, with arbitrary rotation applied

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 [34]

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)

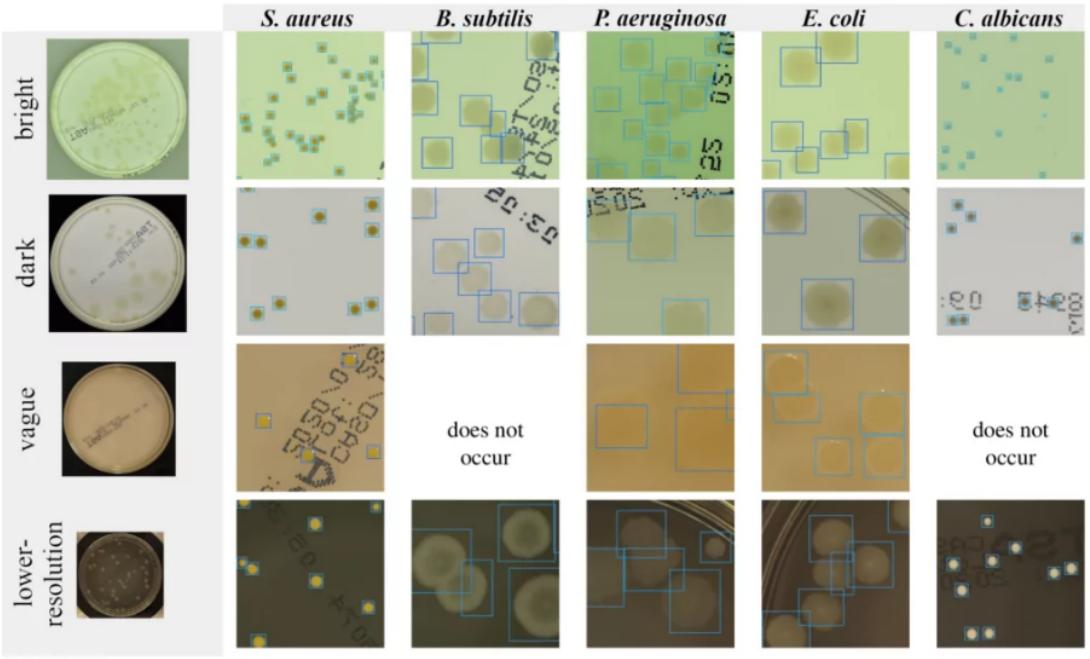

Figure 5: Analysis through object detection example: Petri dish [37]

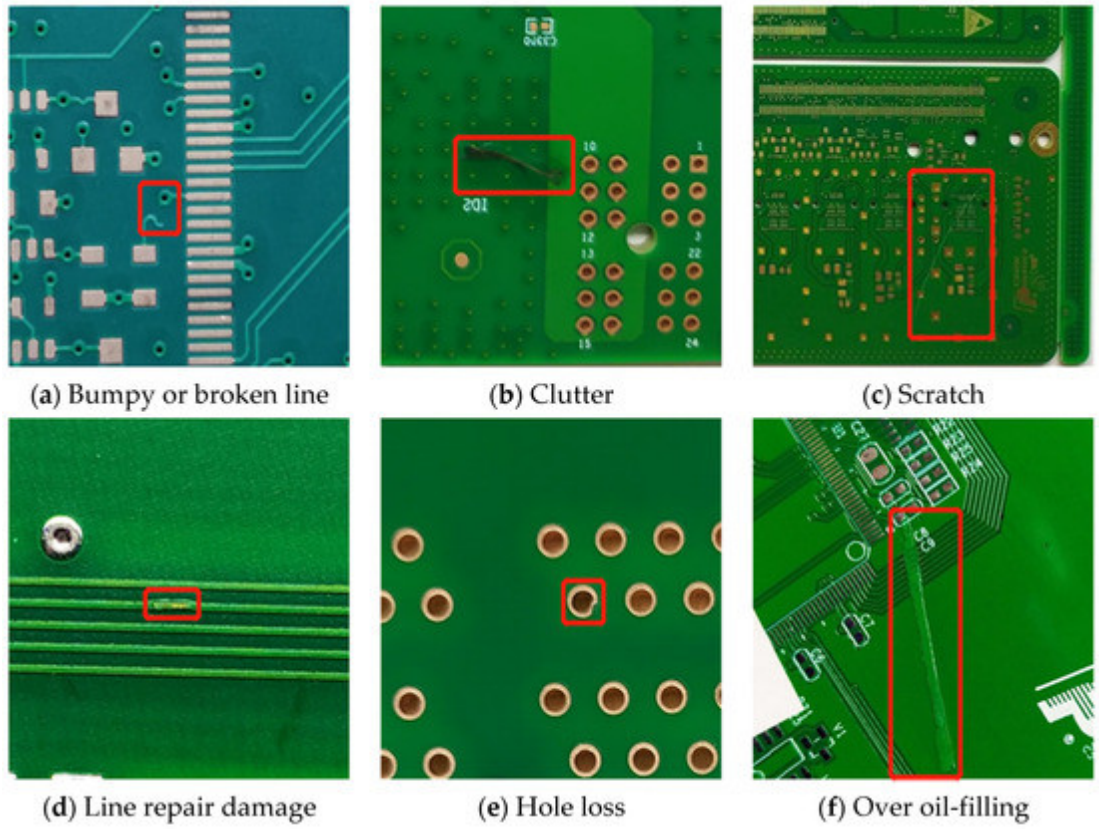

Figure 6: Analysis through object detection example: PCB manufacture [35]

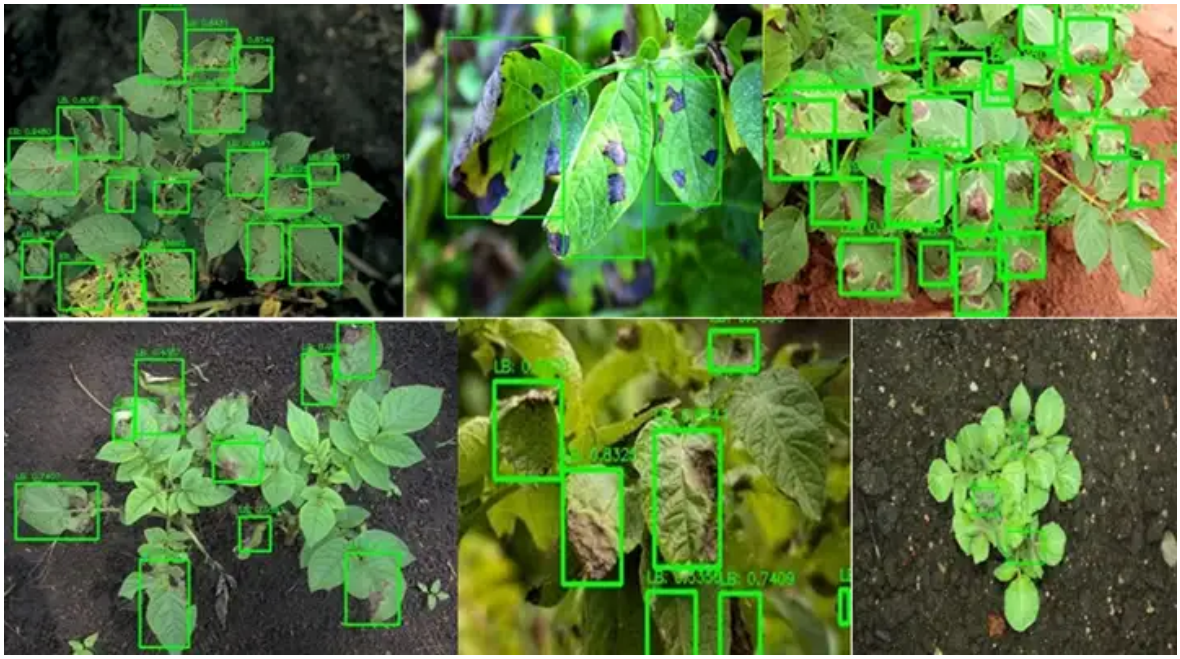

Figure 7: Analysis through object detection example: Crop disease [30]

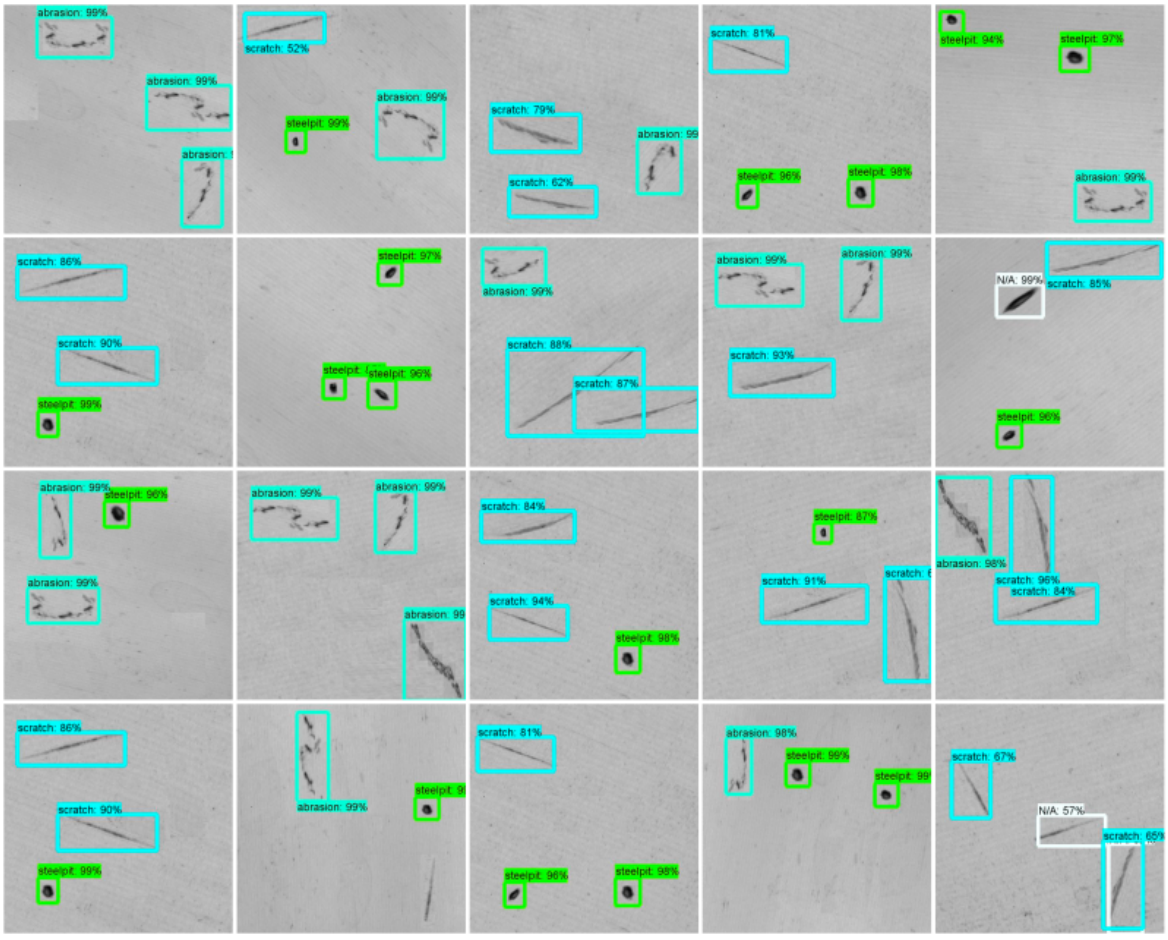

Figure 8: Analysis through object detection example: Surface Defection [36]

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 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.

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 will be 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 timeframe 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 [3]

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 [4]

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 [1]

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**

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**

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

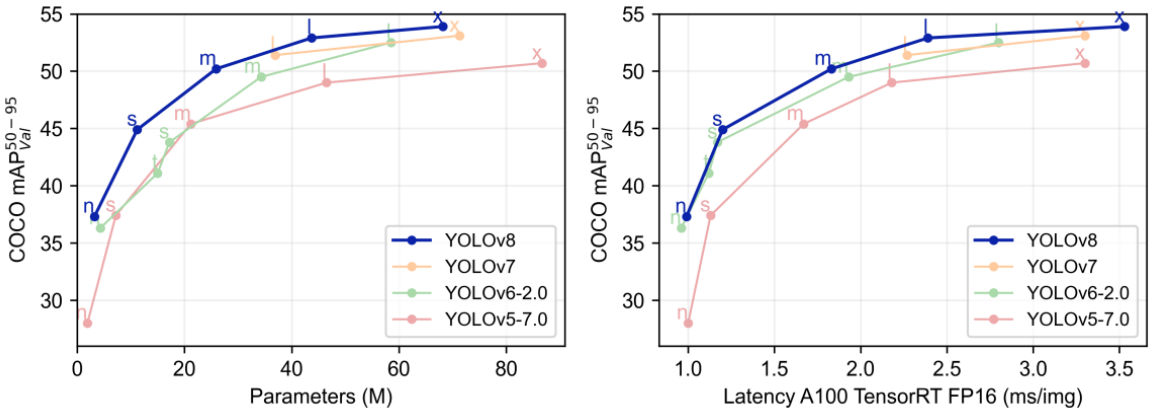

Figure 10: YOLO family comparison chart [1]

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 [23]

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]

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 [5]

Figure 13: YOLO detection: Input Image [5]

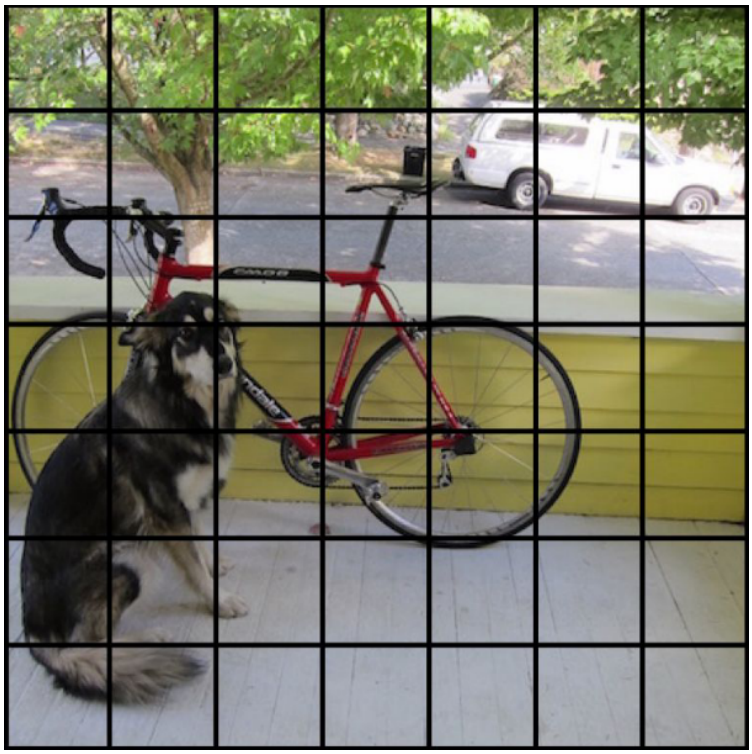

Figure 14: YOLO detection: Splitting into S by S grid, S being 7 in this figure. [5]

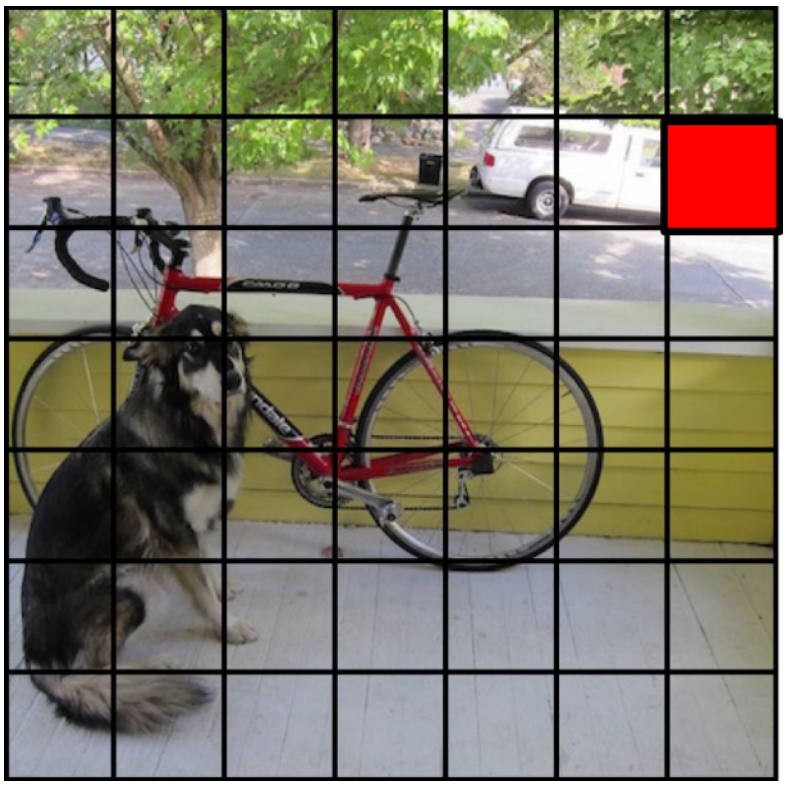

Figure 15: YOLO detection: Each cell predicts the bounding boxes and confidence values of each box. [5]

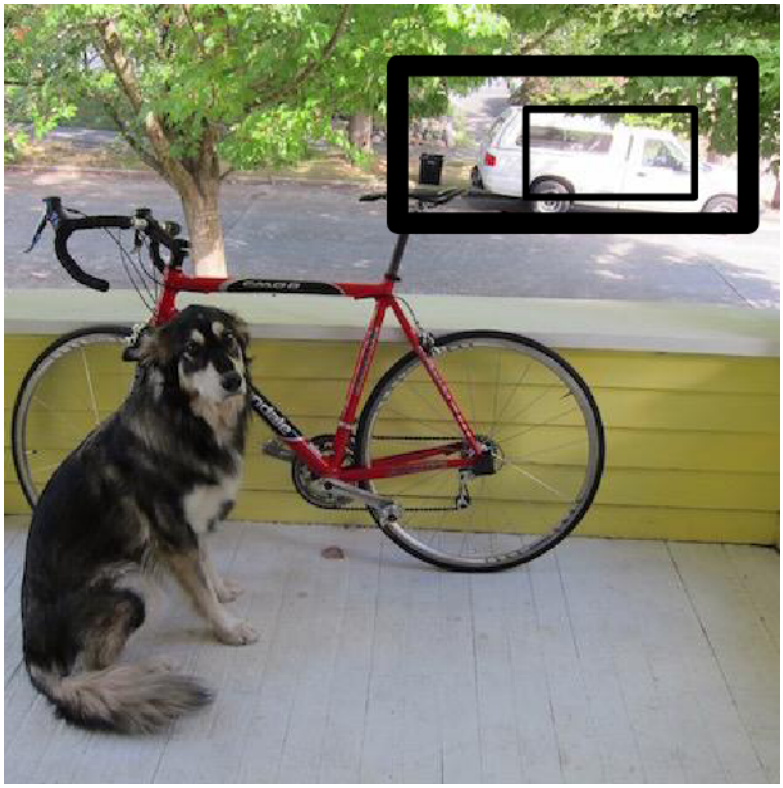

Figure 16: YOLO detection: Identified bounding box. [5]

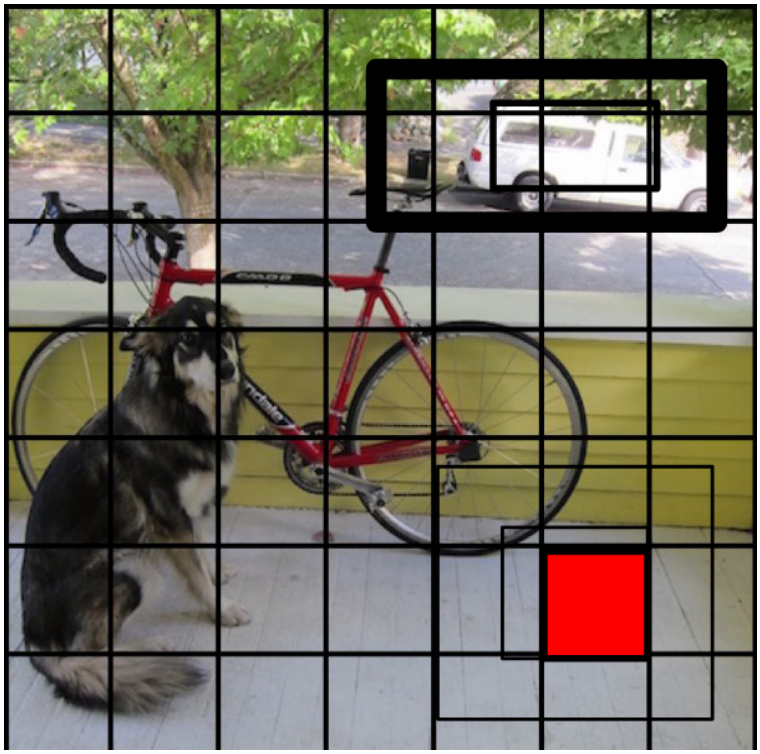

Figure 17: YOLO detection: Previous steps being repeated for each cell. [5]

Figure 18: YOLO detection: All identified bounding boxes. [5]

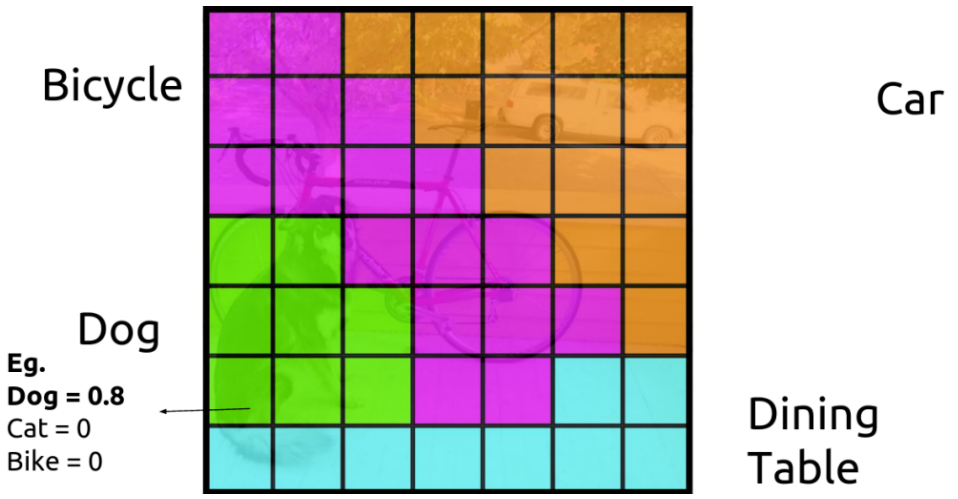

Figure 19: YOLO detection: All identified probabilities for each grid cell. [5]

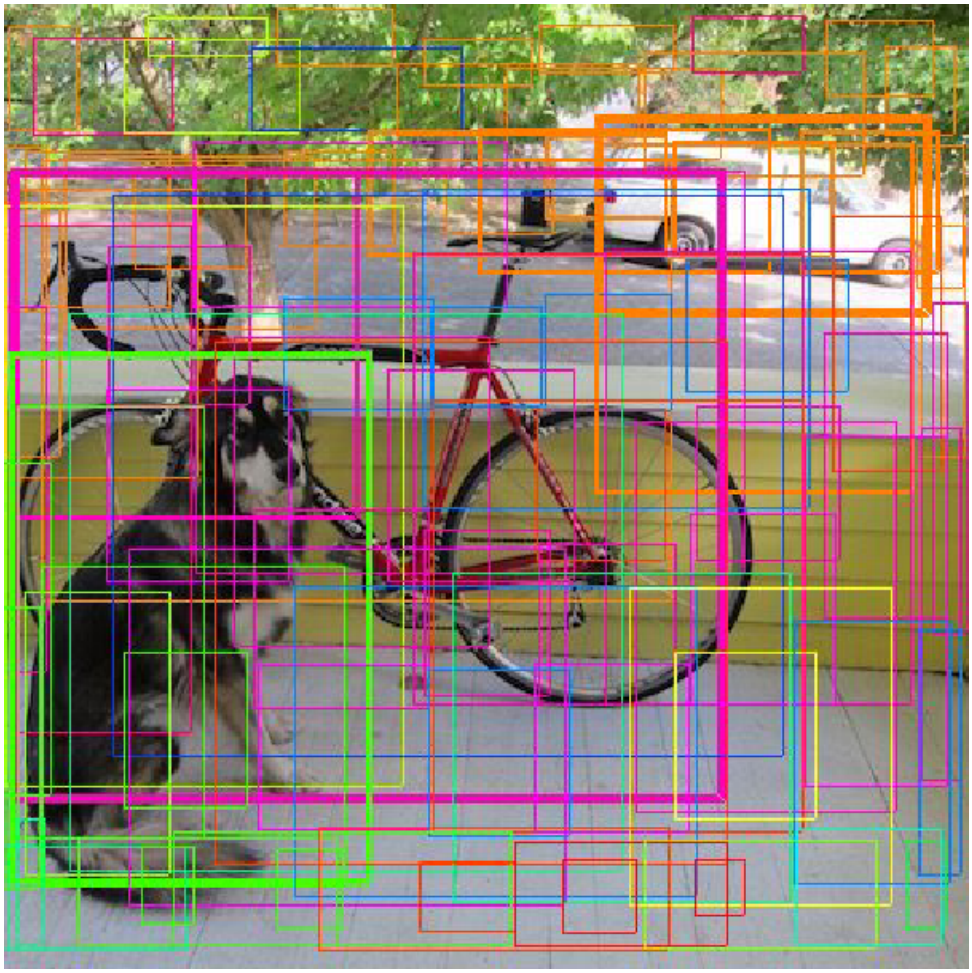

Figure 20: YOLO detection: Each bounding box is "shaded" in the case of this example with how much each one covers of which probability grid. [5]

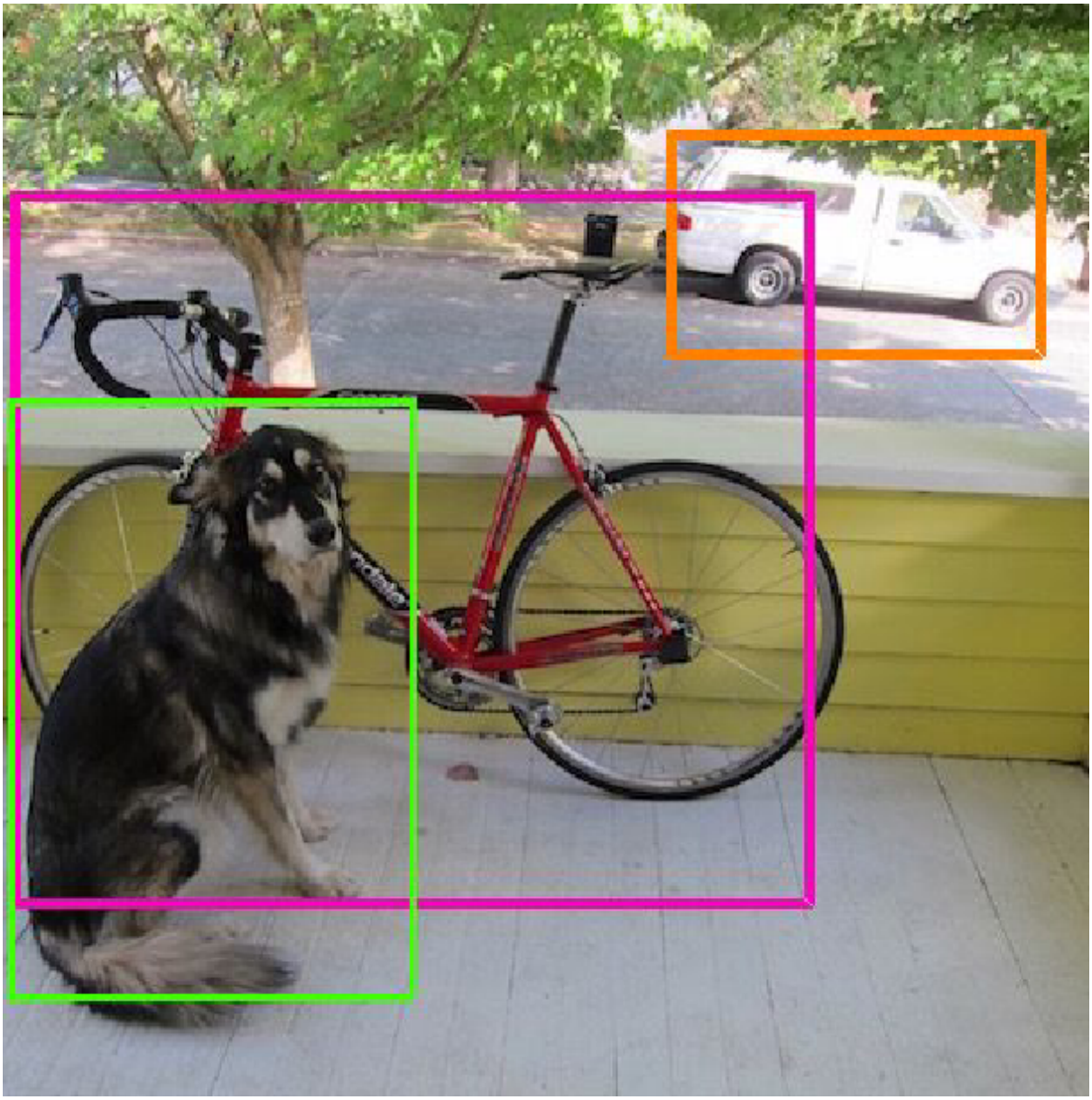

Figure 21: YOLO detection: The bounding boxes are reduced using NMS. This is the final step. [5]

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

7.3.2. Language

C++ [7] 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.

7.3.3. Architecture

YOLO has been chosen as the architecture that this project utilises for the AI detection.

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.

7.3.4. Set Rig

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

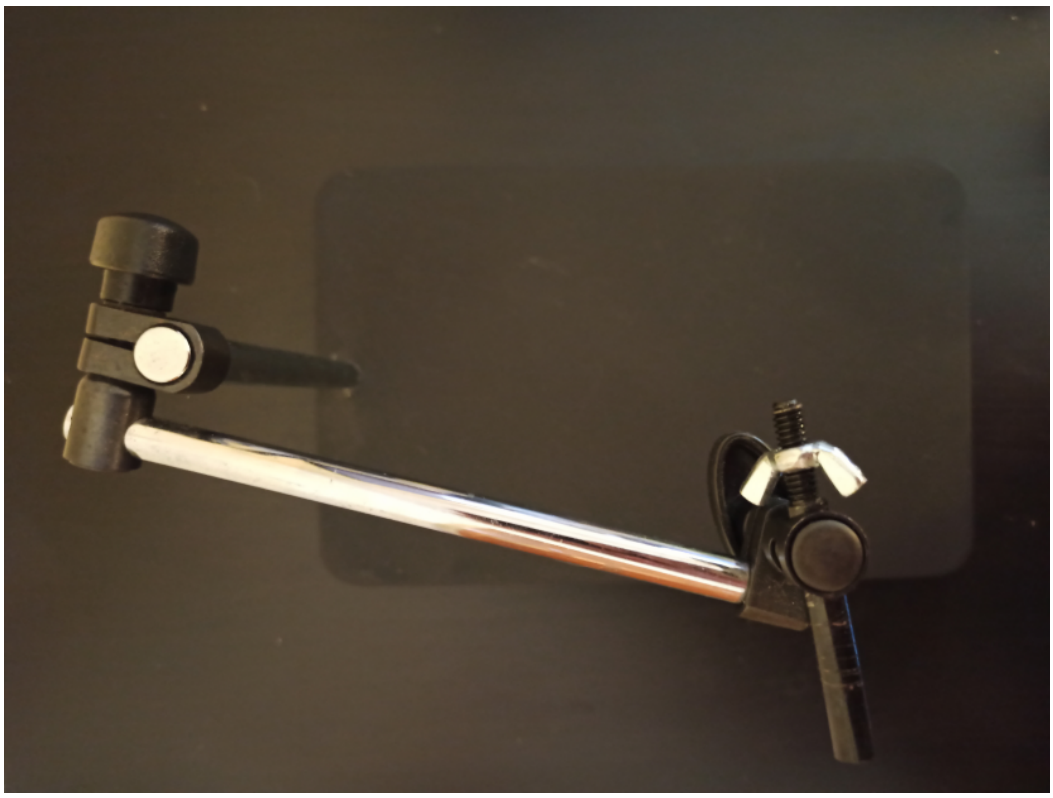

Figure 23: The rig used for this project.

Dimensions

Base Width: 280mm

Base Length: 180mm

Height: 175mm to 350mm (Adjustable arms)

Cost: ~40€

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.

7.3.5. Camera

Figure 24: Camera, model OSY-C100-4-12. Camera of choice for the project. [38]

Model

OSY-C100-4-12 [8]

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.

7.3.6. Ring Light

Figure 25: Ring Light used for this project. [33]

Specifications

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

Cost: ~15€

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
    - Cloud based – upload and download times.
    - 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 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.

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 [9]**

Specifications

CPU

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

GPU

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

- **Beaglebone [10]**

Specifications

CPU

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

GPU

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

- **Nvidia Jetson Nano [11]**

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 [12]**

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):** 16
- **Processor Base Frequency:** 700MHz
- **Memory:** 2GB

- **Android Phone [13]**

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 [14]/Linux [15]/Mac [16] 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 [6] 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

Figure 27: 220Ohm resistor

Color codes

Brown = 1

Red = 2

Gold = 5% tolerance

Calculations

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

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

$$220Ohm$$

$$220 \pm (5\%^{*}220)$$

$$5\%^{*}220 = 11$$

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

Capacitors

	1st Number (x10pF)	2nd Number (pF)	3rd Number (Multiplier x10n)	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.

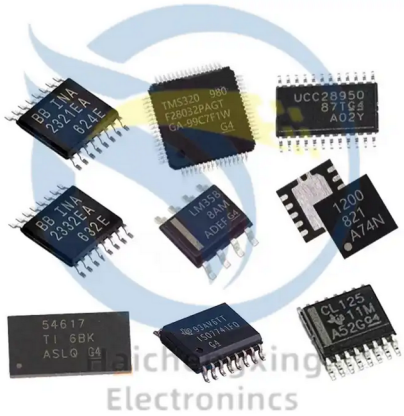

Figure 30: Variety of labelling on components that originate from different IC manufacturers. [29]

7.5.1.1. Examples

Input image

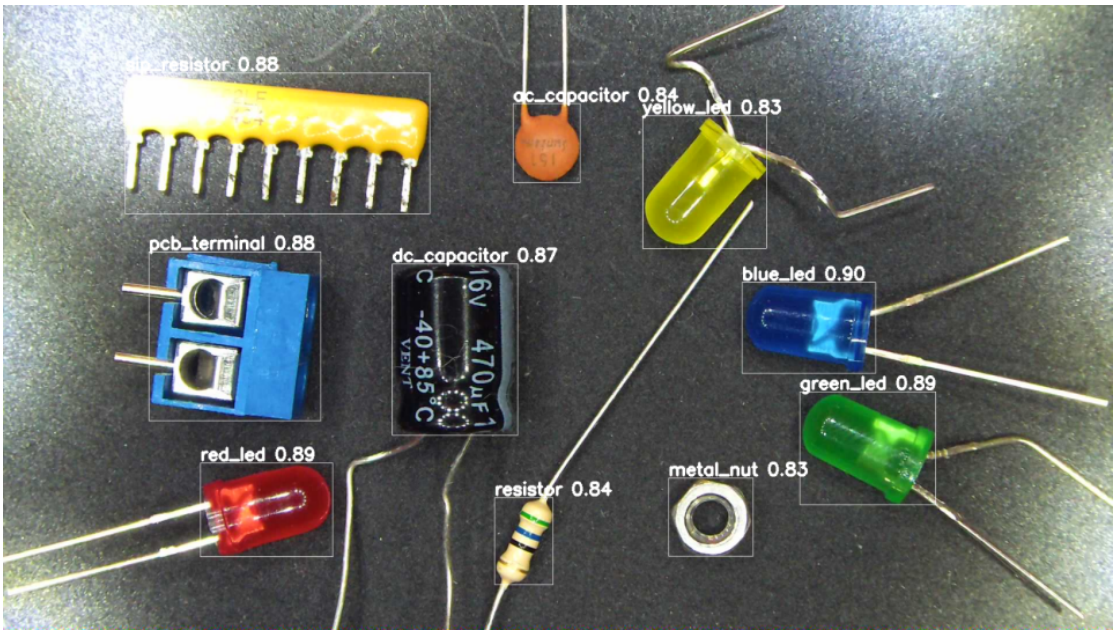

Figure 31: Example of 3rd trained model running Inference.

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.

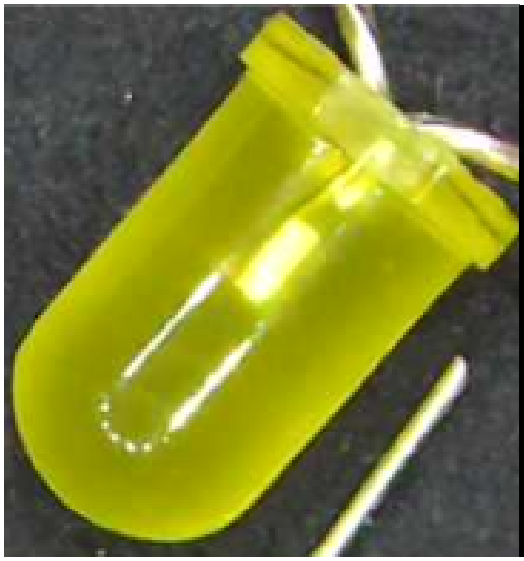

Figure 32: Color extraction example: Raw input image

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 [32])

Results show clearly prominent yellow which is accurate.

HSV Analysis

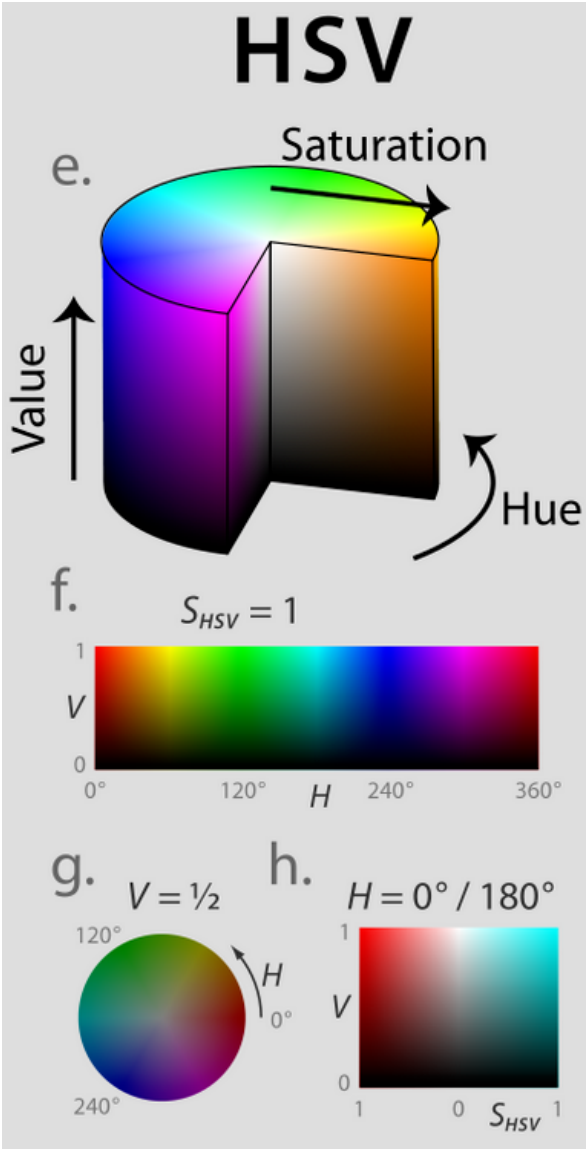

Figure 35: HSV chart overview [28]

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.

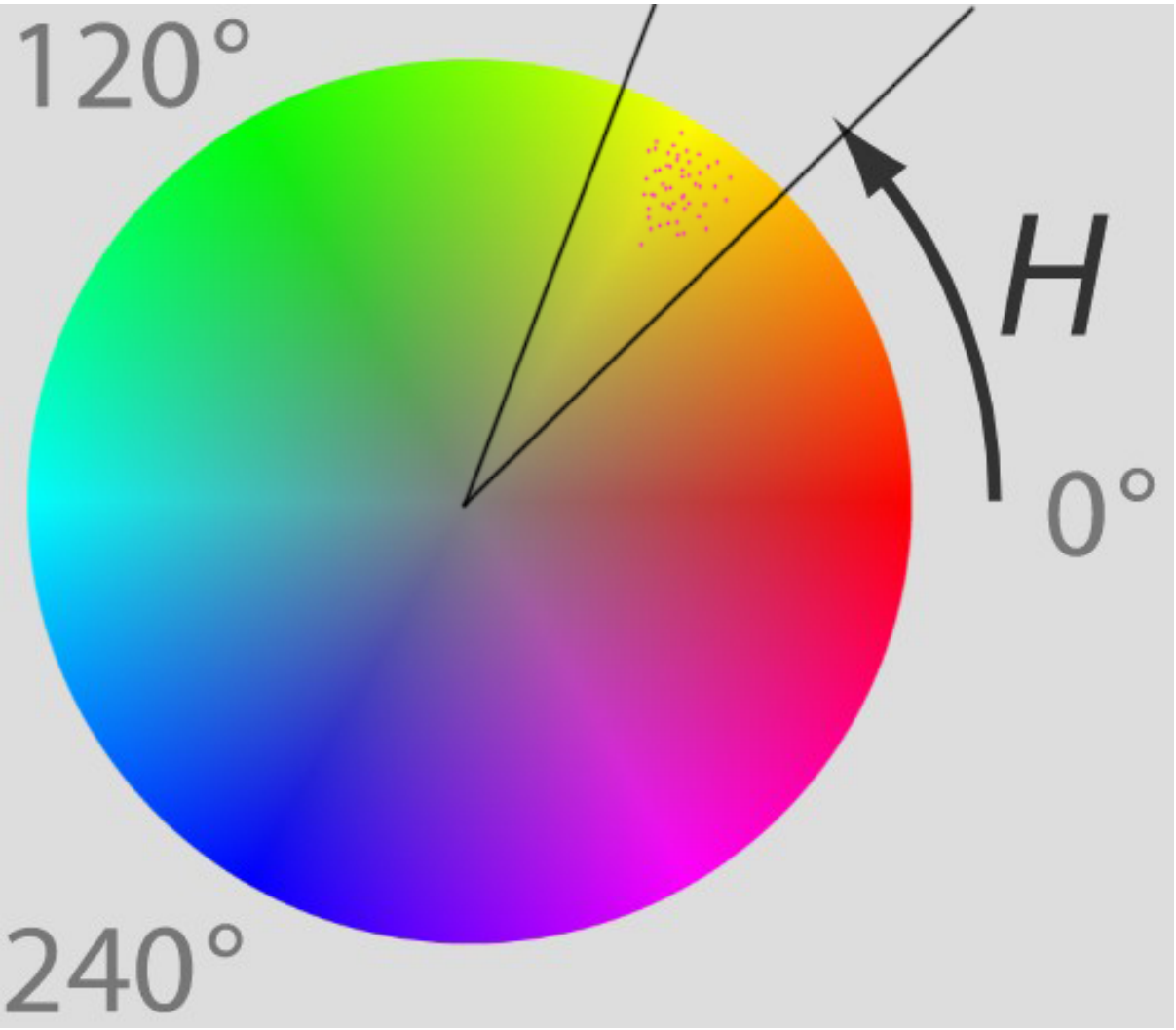

Figure 36: A circle hue chart with the data represented as pink dots. [28]

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.

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 may be used to extract the text based information.

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

CPU: AMD RyzenTM 7 5800X3D [17]

Brand: AMD [18]

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 Vengeance [19] RGB PRO SL [20]

- **Capacity:** 2x16GB
- **Type:** DDR4
- **Frequency:** 3.6GHz

GPU: GeForce RTX 3060 Ti [21]

- **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. Upclose 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

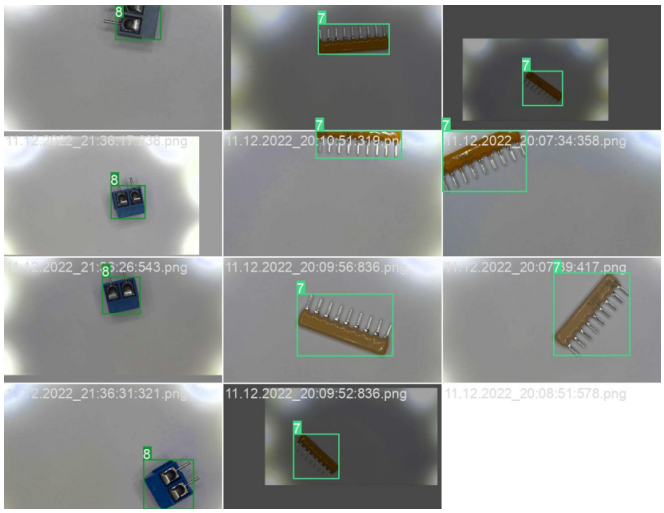

Figure 37: Mild augmentation example 1

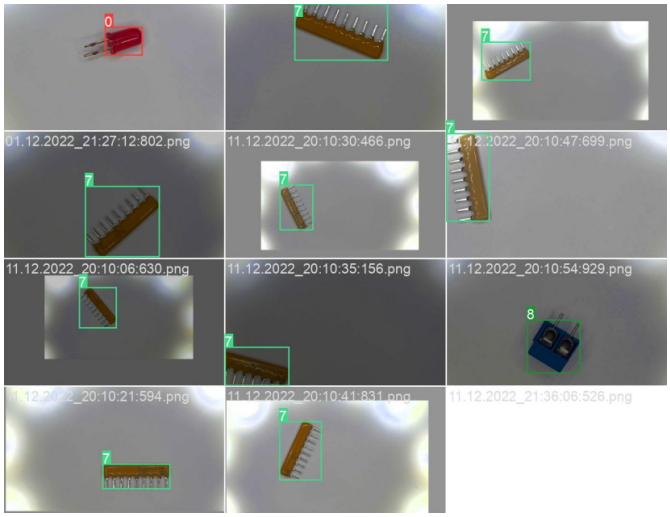

Figure 38: Mild augmentation example 2

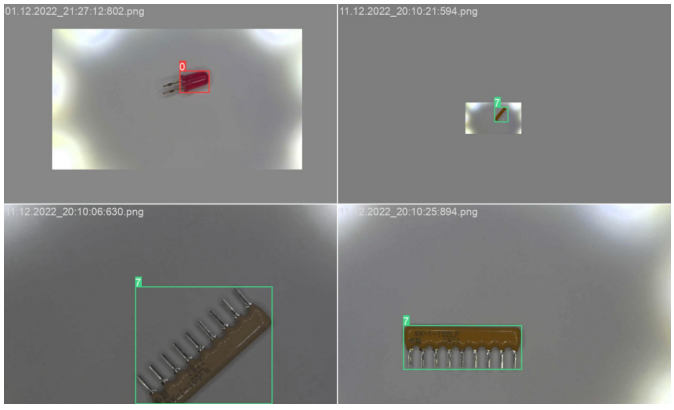

Figure 39: Mild augmentation example 3

8.2.2. Inference

Internal AI Object Detection Steps

1. **Classification**

Figure 40: Object Detection: Classification [31]

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**

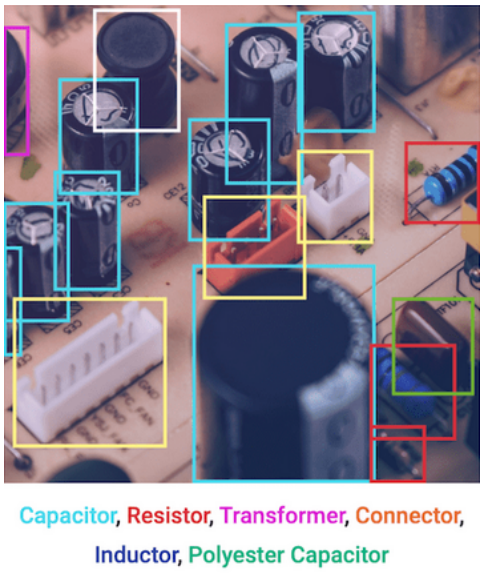

Figure 41: Object Detection: Item Boundary Identification [31]

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

3. Segmentation

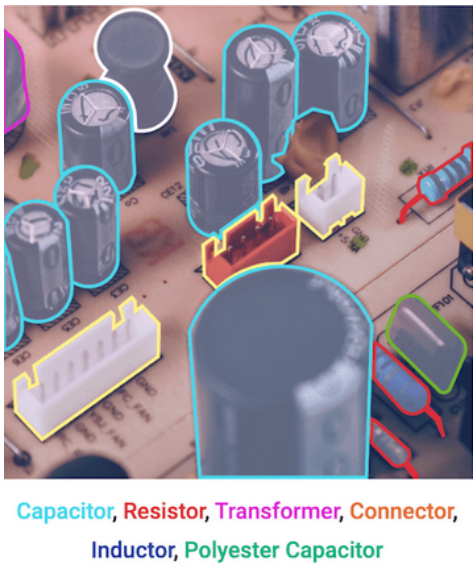

Figure 42: Object Detection: Segmentation [31]

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

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 an Nvidia GPU featuring CUDA cores.

Libraries

Working directory

Let ~/opencv/ be our cmake 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 43: Terminal command to install cmake-gui

- **OpenCV**

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**

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**

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 44: ~/opencv directory setup for CMake-GUI

OpenCV

Generation of build files using CMake GUI

Open CMake and enter cmake-gui into the terminal.

Then we have to select the source and binaries paths:

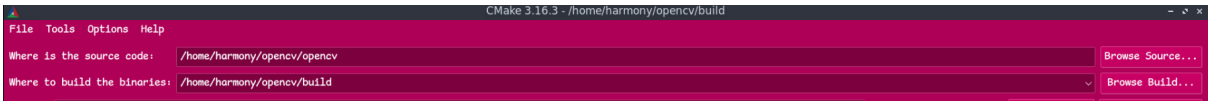

Figure 45: CMake GUI: Source and build binaries directories

Source path should be ~/opencv/opencv

Binaries path should be ~/opencv/build

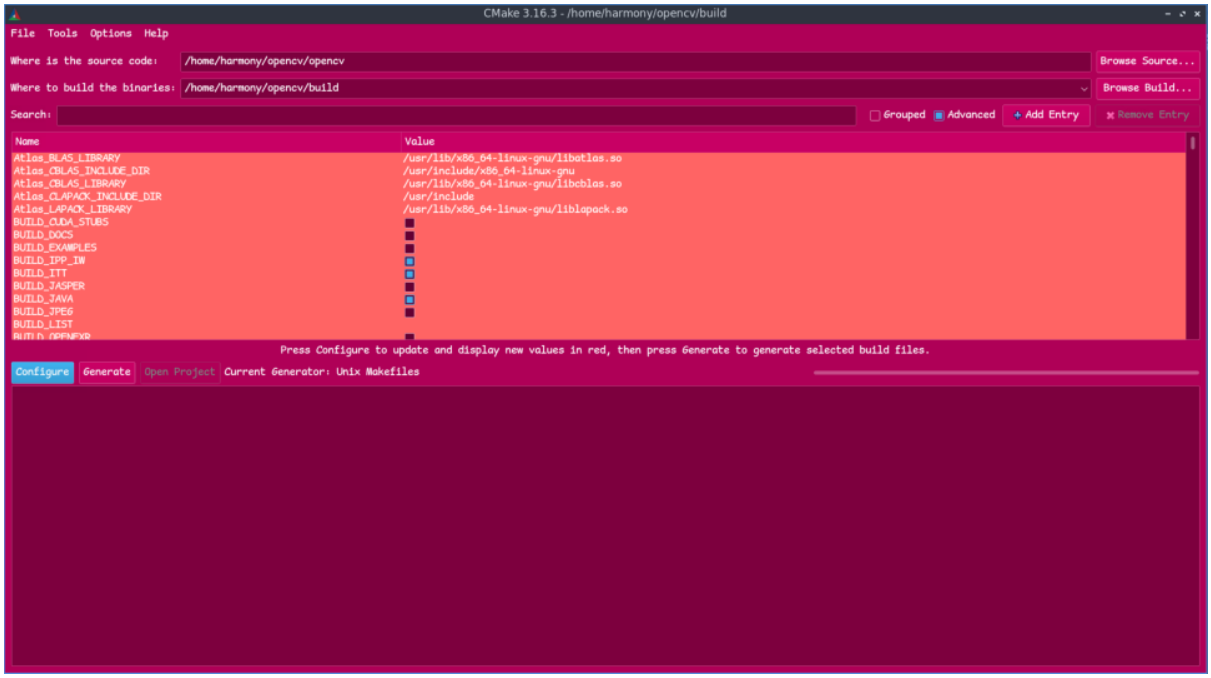

Figure 46: CMake GUI: initial configuration (Configure is highlighted in blue)

Press Configure.

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

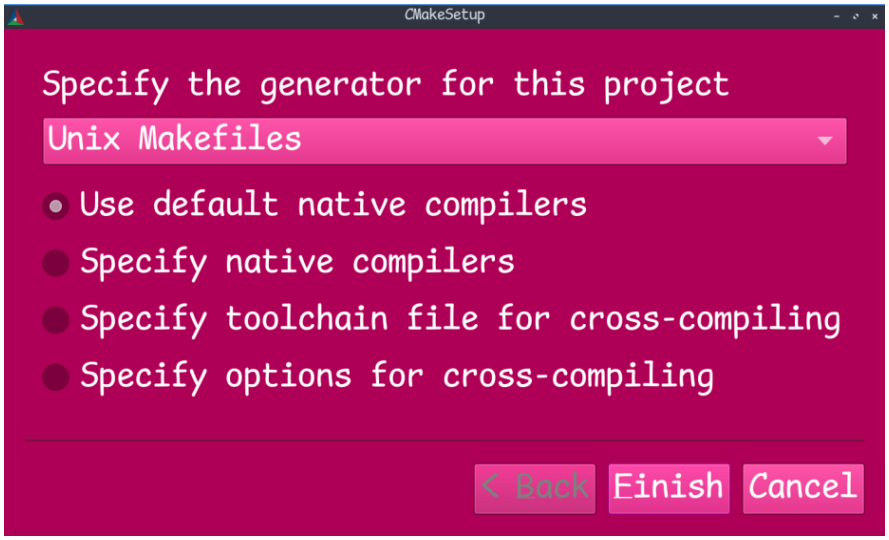

Figure 47: CMake GUI: Generator selection

Flags setup

OpenCV

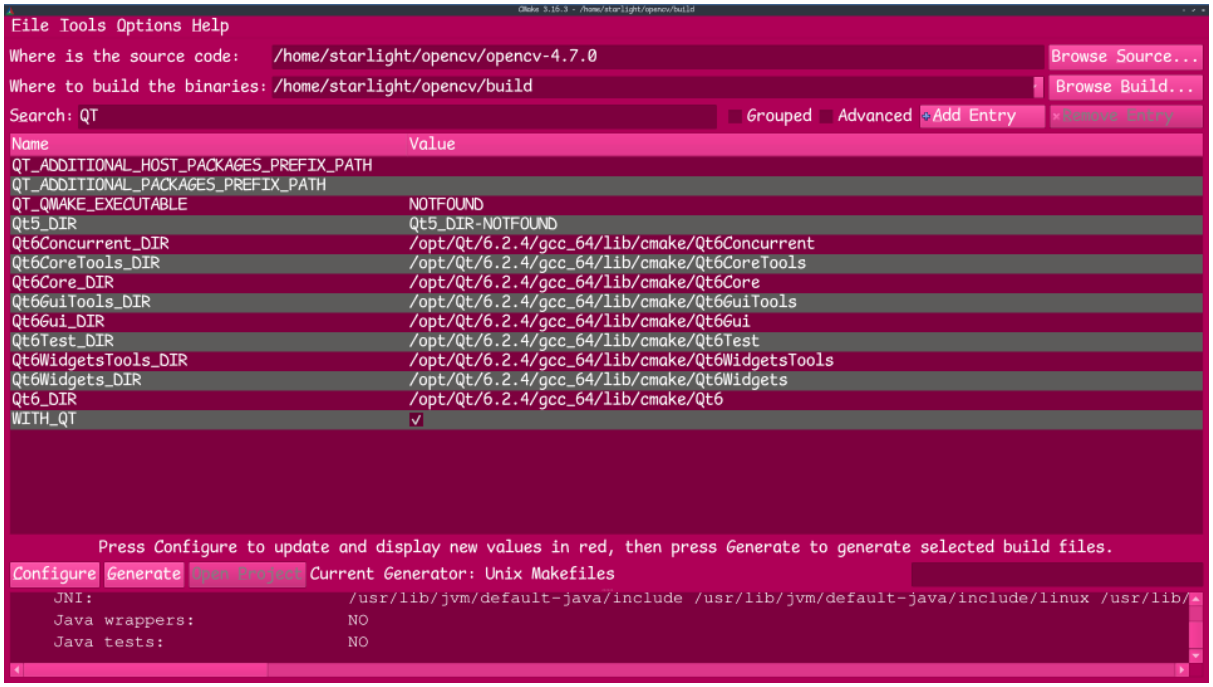

Figure 48: CMake GUI: Qt flags

Flags required for Neural Networks

- **WITH_ONNX:** ✅
- **WITH_INF_ENGINE:** ✅
- **WITH_NGRAPH:** ✅
- **CUDAWITH_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.

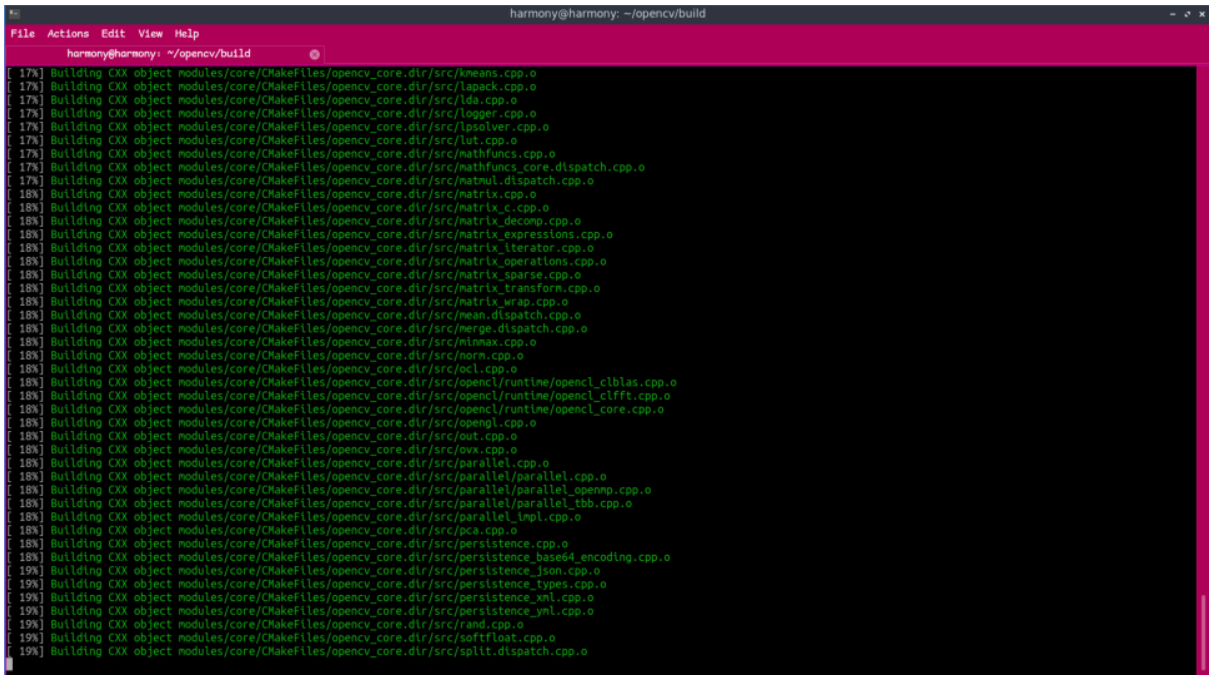

Figure 49: Building using the generated by CMake files, with the make command

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 50: result of sudo make install command

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

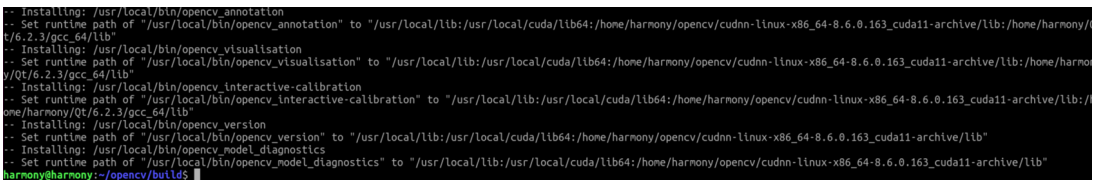

Figure 51: OpenCV libraries are now ready to use.

Qt Creator

Installation

Version: 6.2.4

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

The Open Source version was used for this project.

Figure 52: Qt Creator website: Open Source

We want the Qt Online Installer.

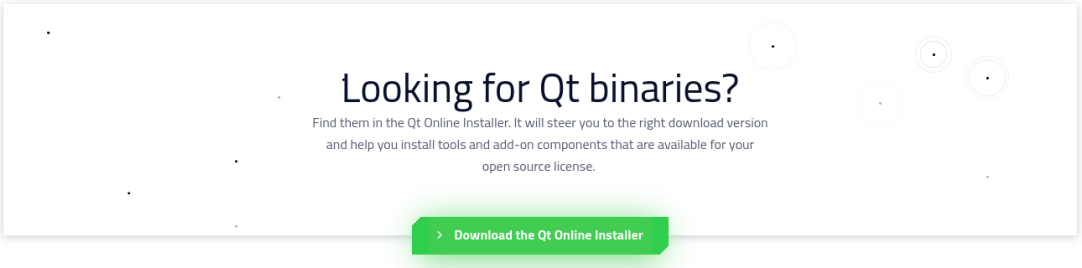

Figure 53: Qt Creator website: Online Installer

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

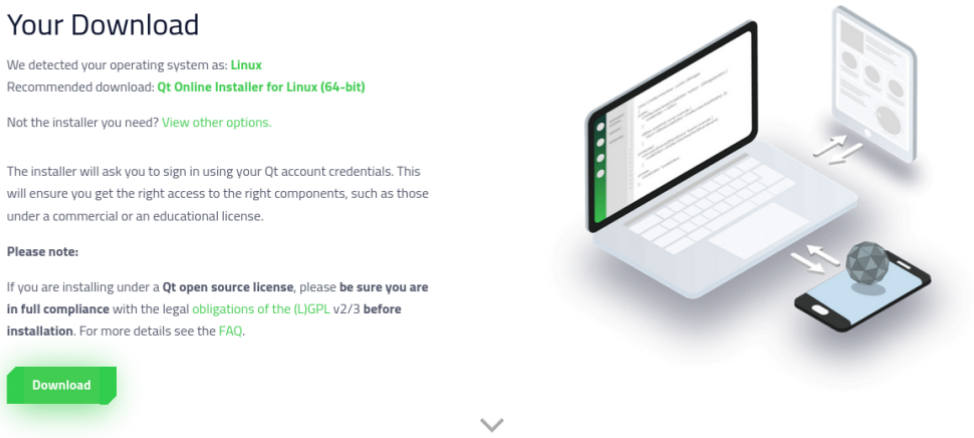

Figure 54: Qt Creator website: Online Installer for Linux 64 bit download page

Figure 55: 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

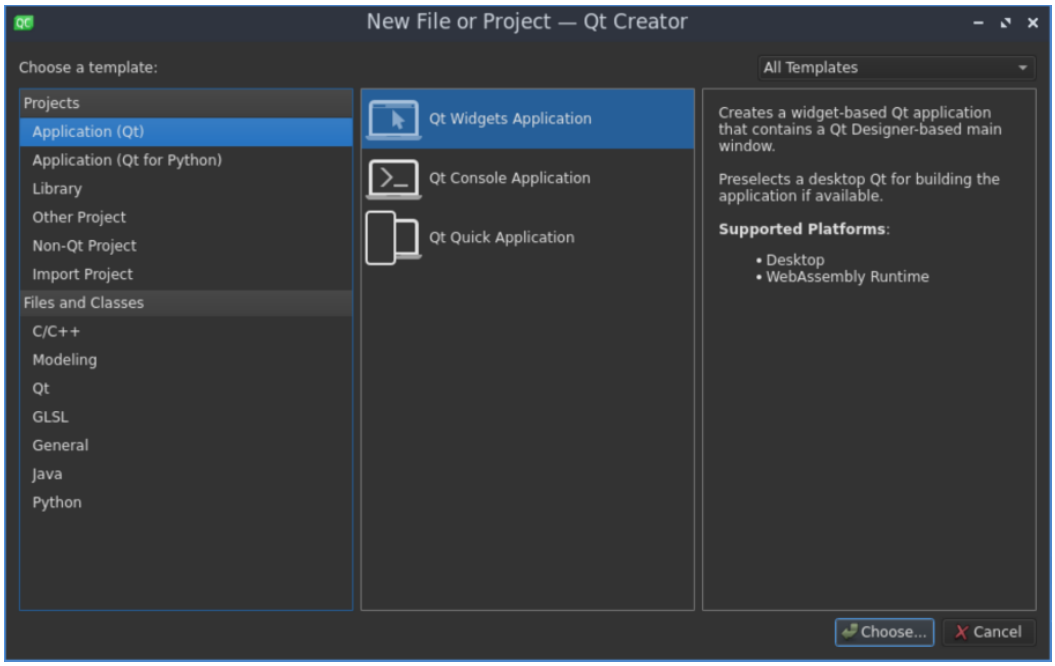

Figure 56: Qt Creator: Type of project selection

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

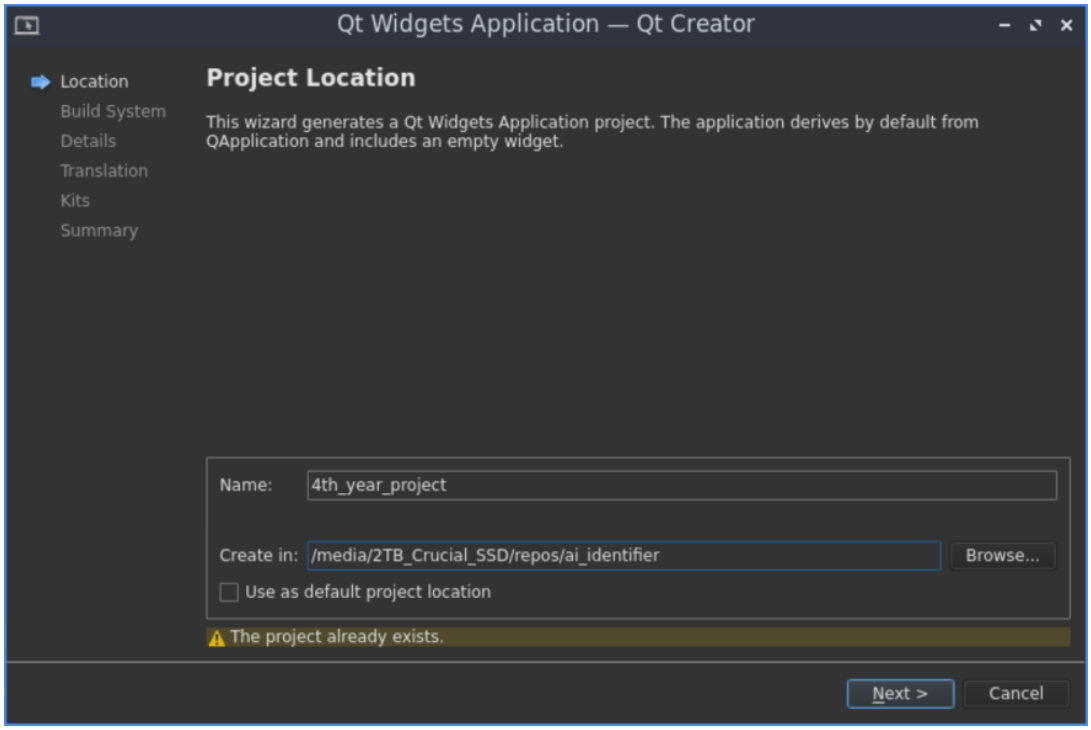

Figure 57: Qt Creator: Project location selection

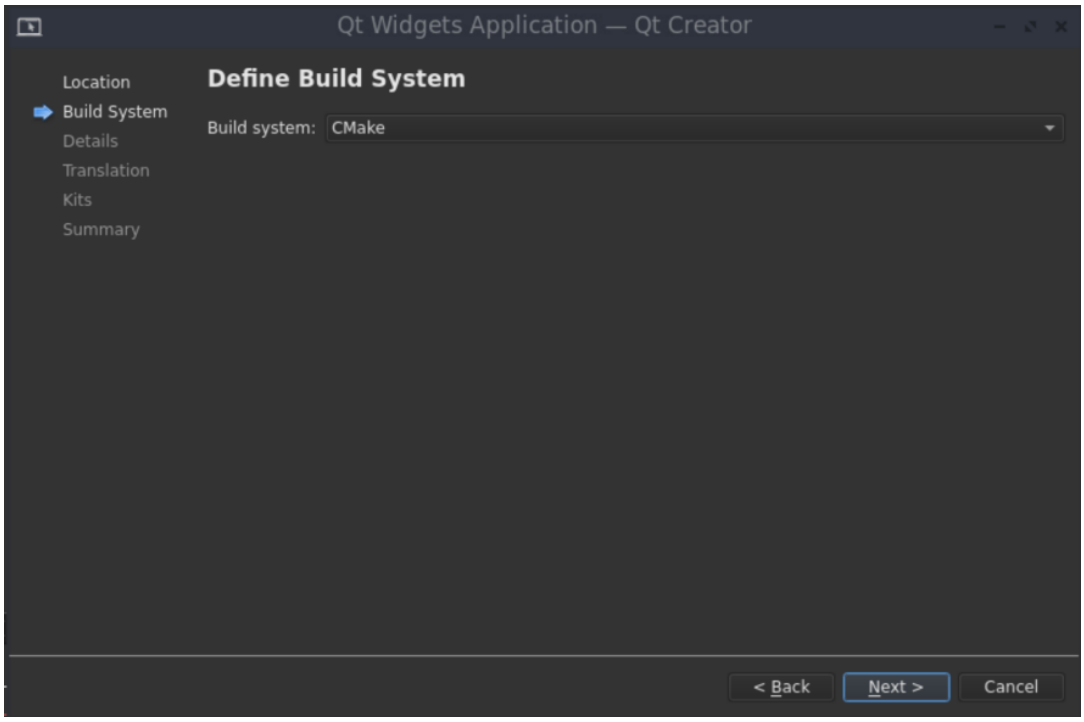

Figure 58: Qt Creator: Build system selection

Build System: **CMake**

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

Build Kits

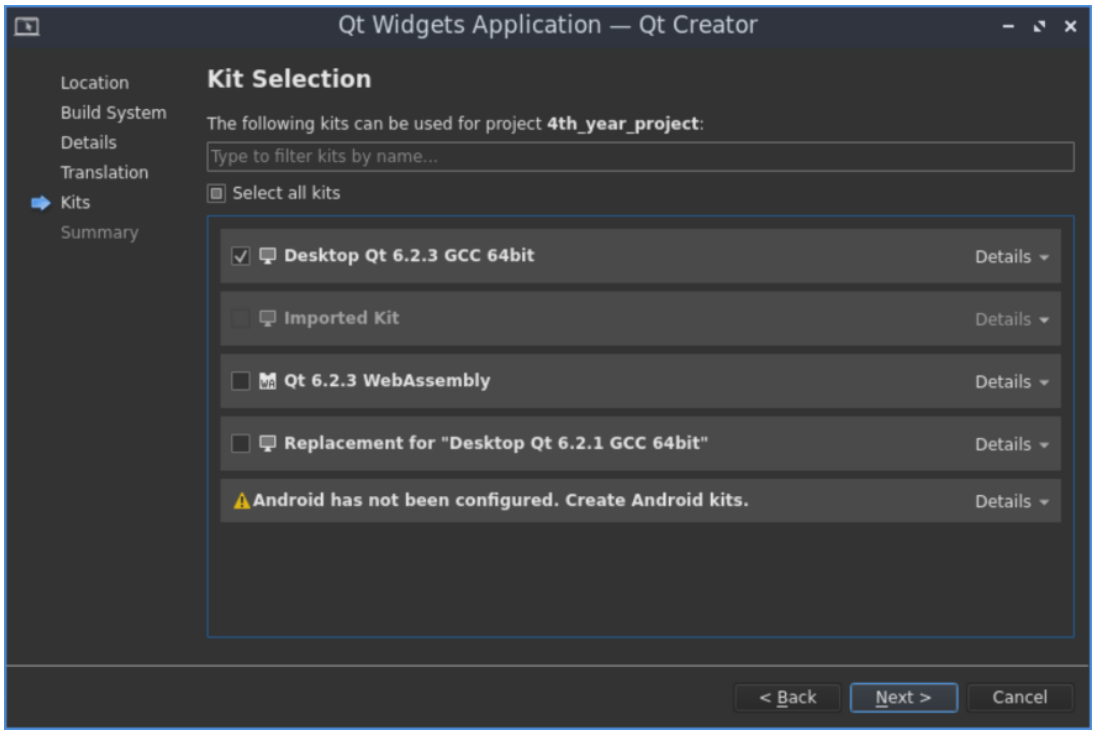

Figure 59: Qt Creator: Build kits selection

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.

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

The **CUDA** version matches the one compiled during **OpenCV** installation.

Systems

- **Threads**

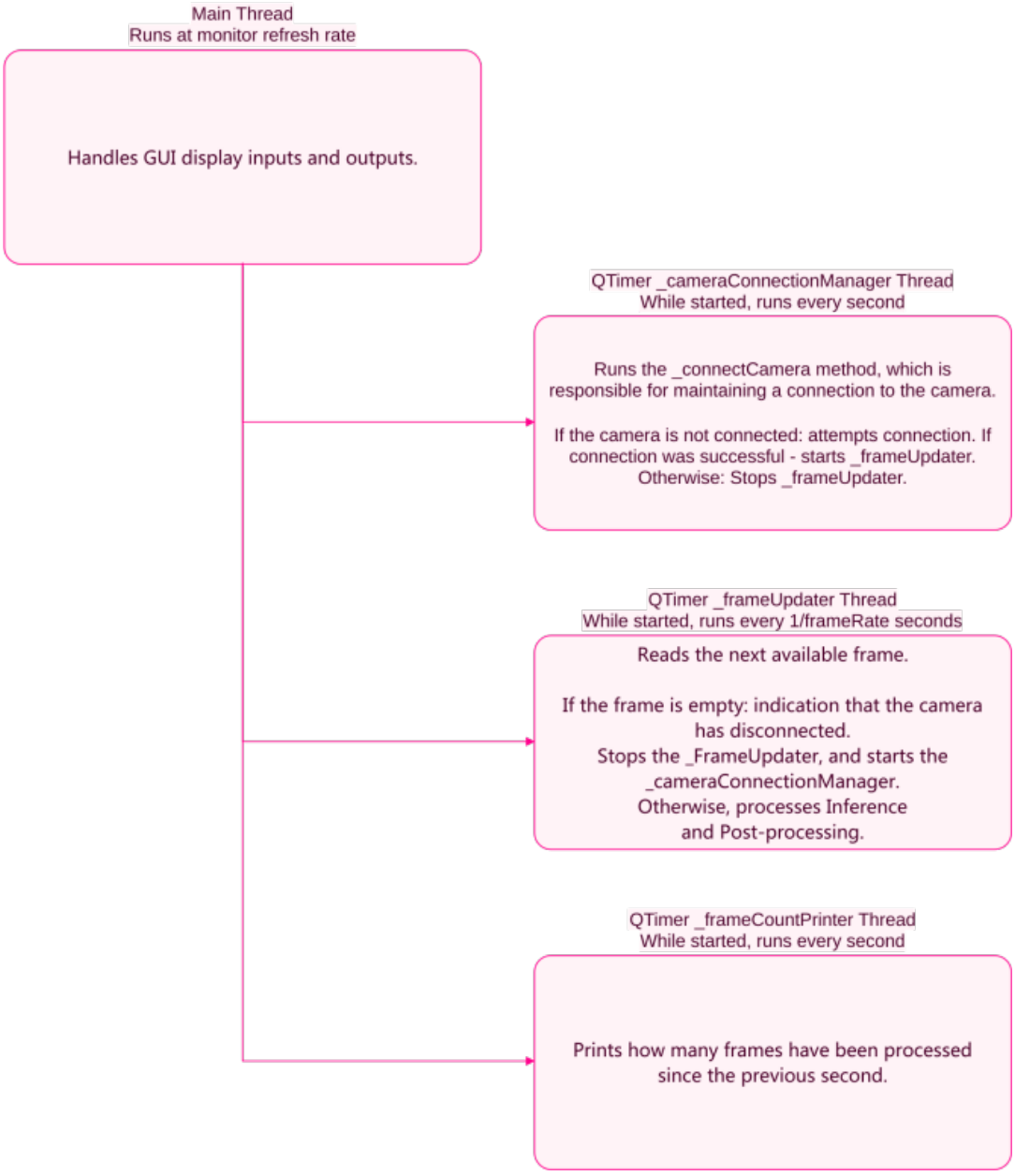

Figure 61: Qt Creator: Simplified overview of the utilised threads.

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 62: 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**

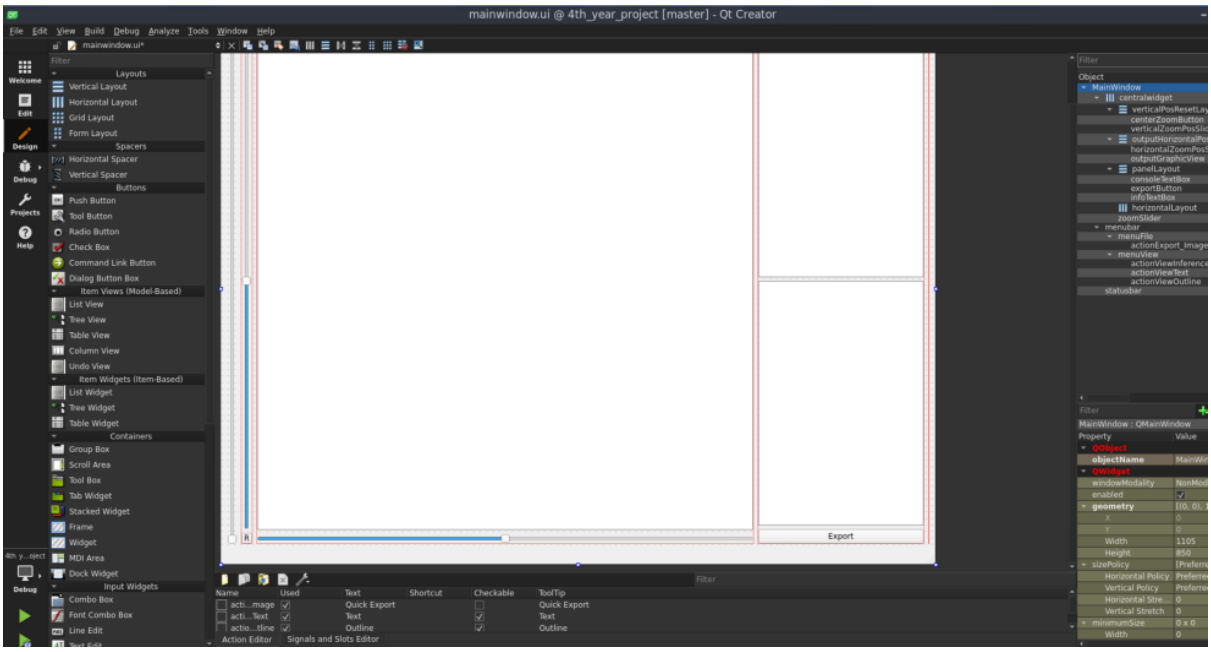

Figure 63: Qt Creator: UI design

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

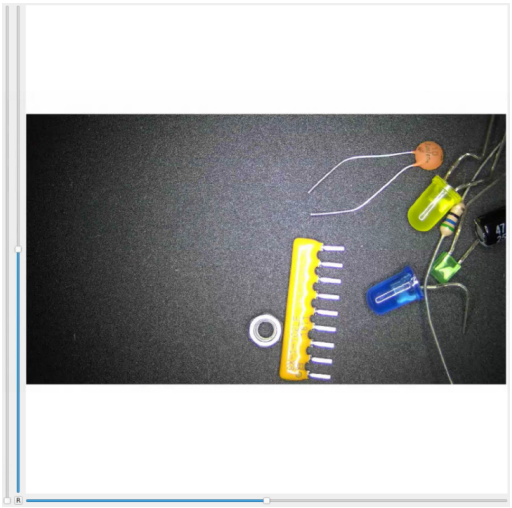

Figure 64: Qt Creator: UI: No zoom

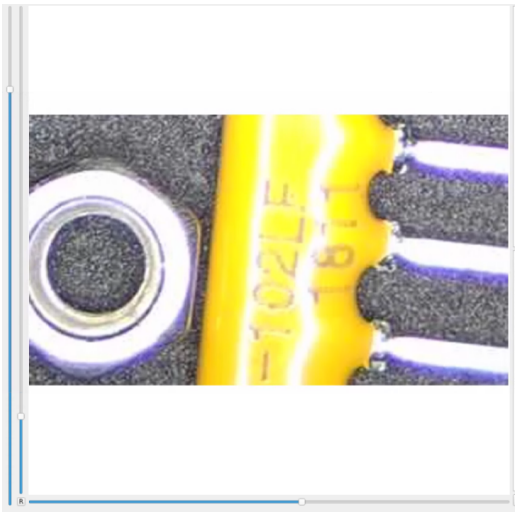

Figure 65: Qt Creator: UI: Zoom example 1: SIP resistor

Figure 66: Qt Creator: UI: Zoom example 2: AC capacitor

Figure 67: Qt Creator: UI: Zoom example 3: LEDs

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

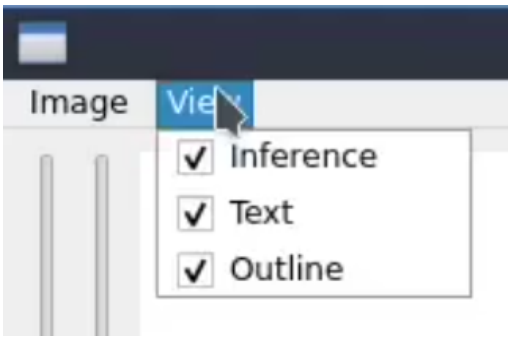

Figure 68: QT Creator: UI: Status bar

  - Inference detections
  - Text display
  - Outline of found components

Export button

Exports the entire frame, both before and after inference

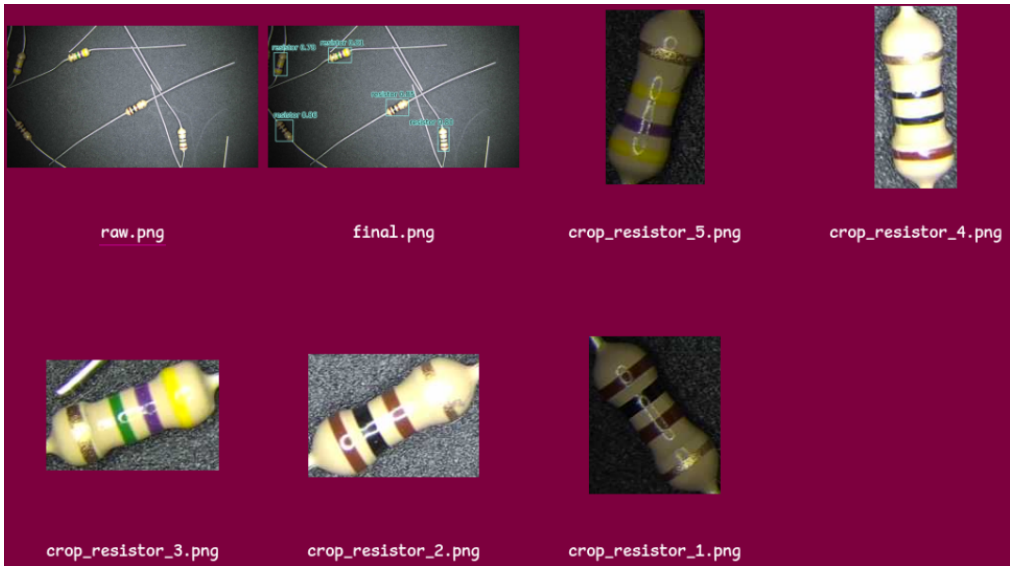

Figure 69: Export button output example

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 70: 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

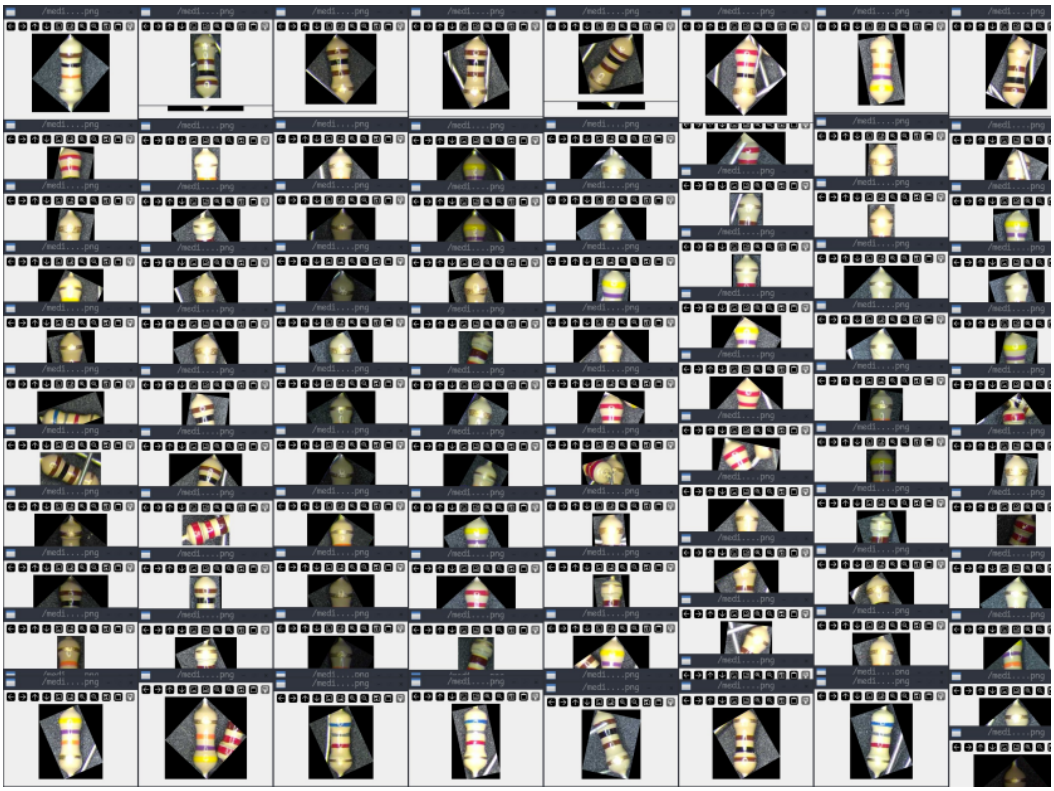

Figure 71: Post-Processing: Saturation adjustment: Before

Figure 72: 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 countours 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.

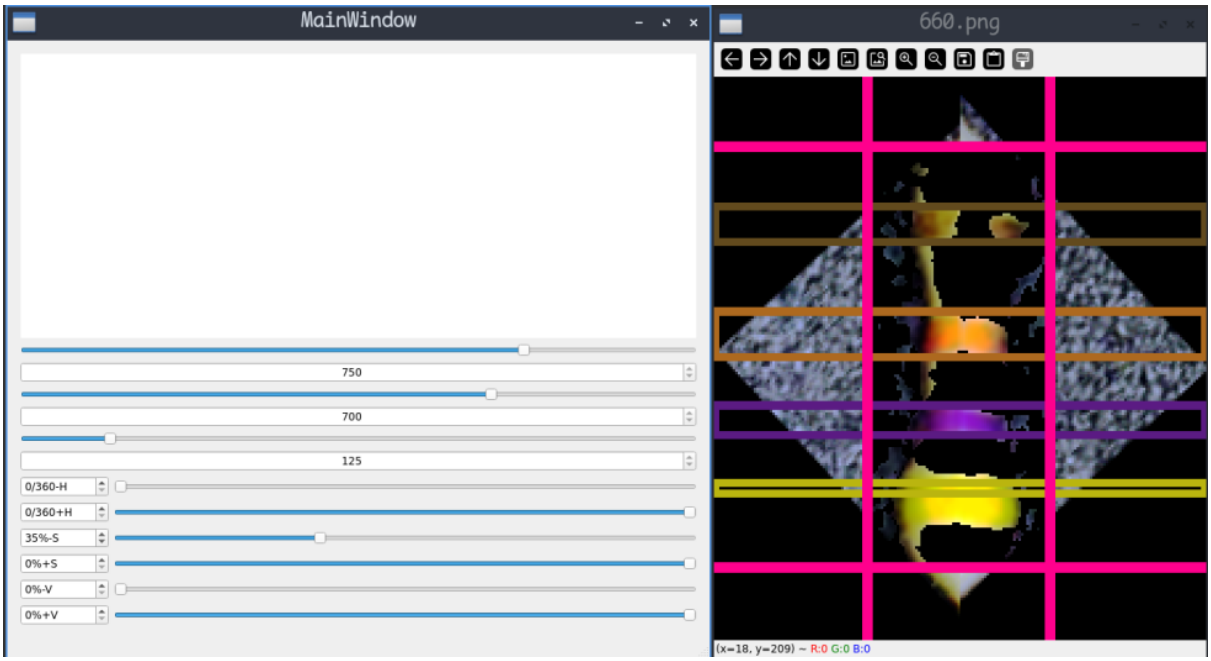

Figure 73: Side project: Post-Processing resistors captured from the rig itself.

Dataset

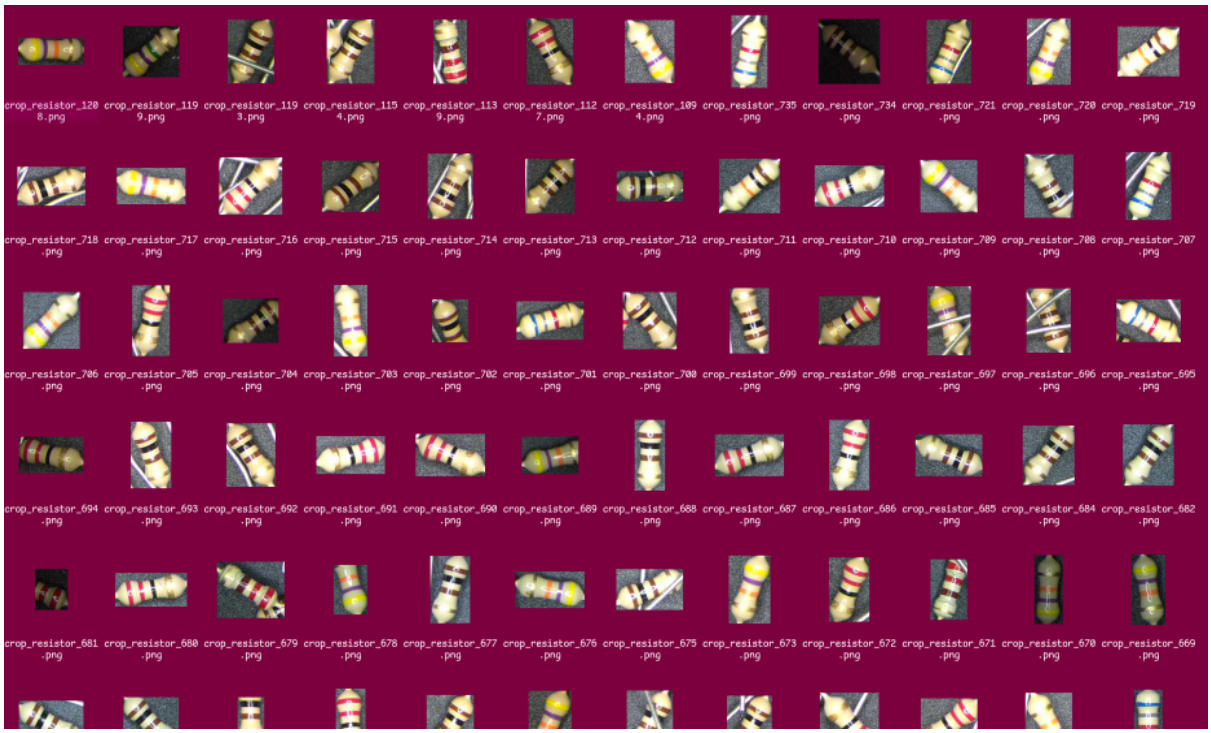

Figure 74: Gathered resistor dataset example

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 75: 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

input

Mat

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

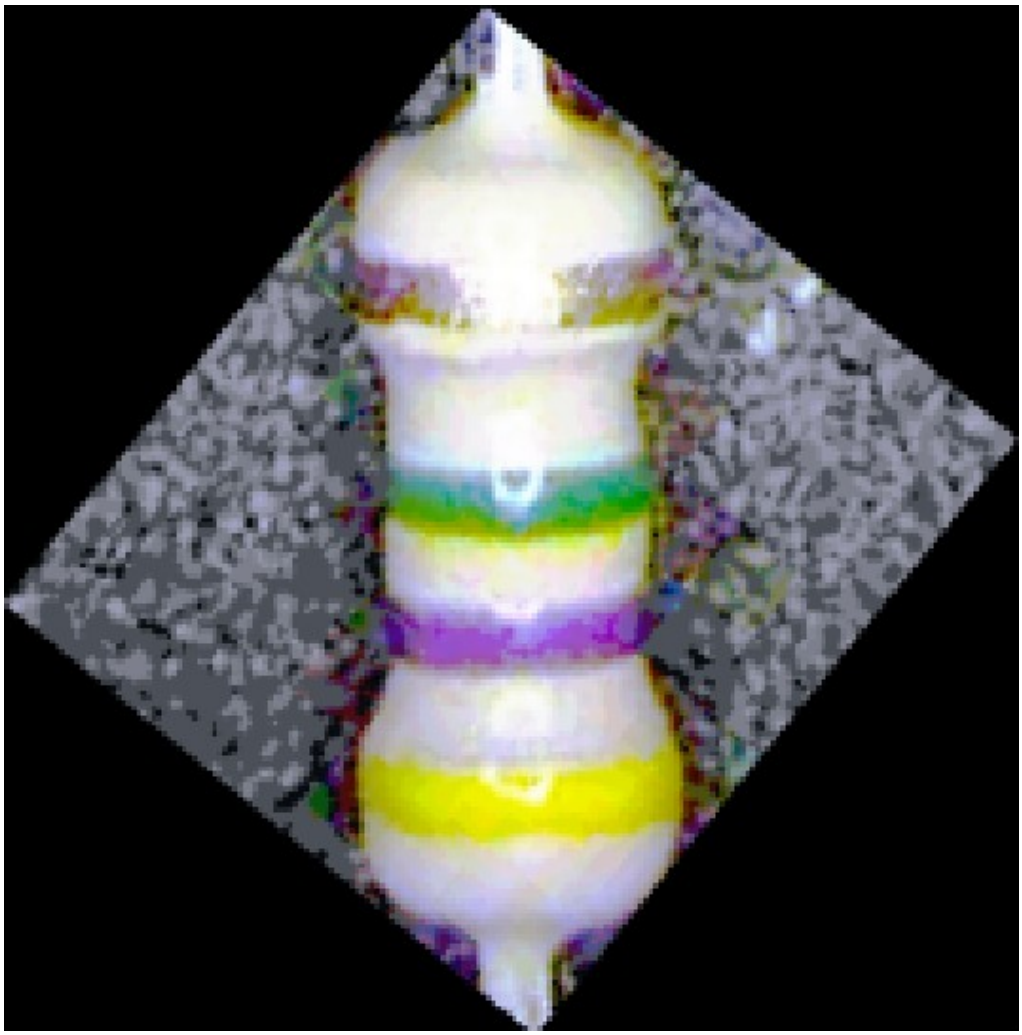

Figure 76: Post-Processing: Resistor algorithm: Input Mat example

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.)

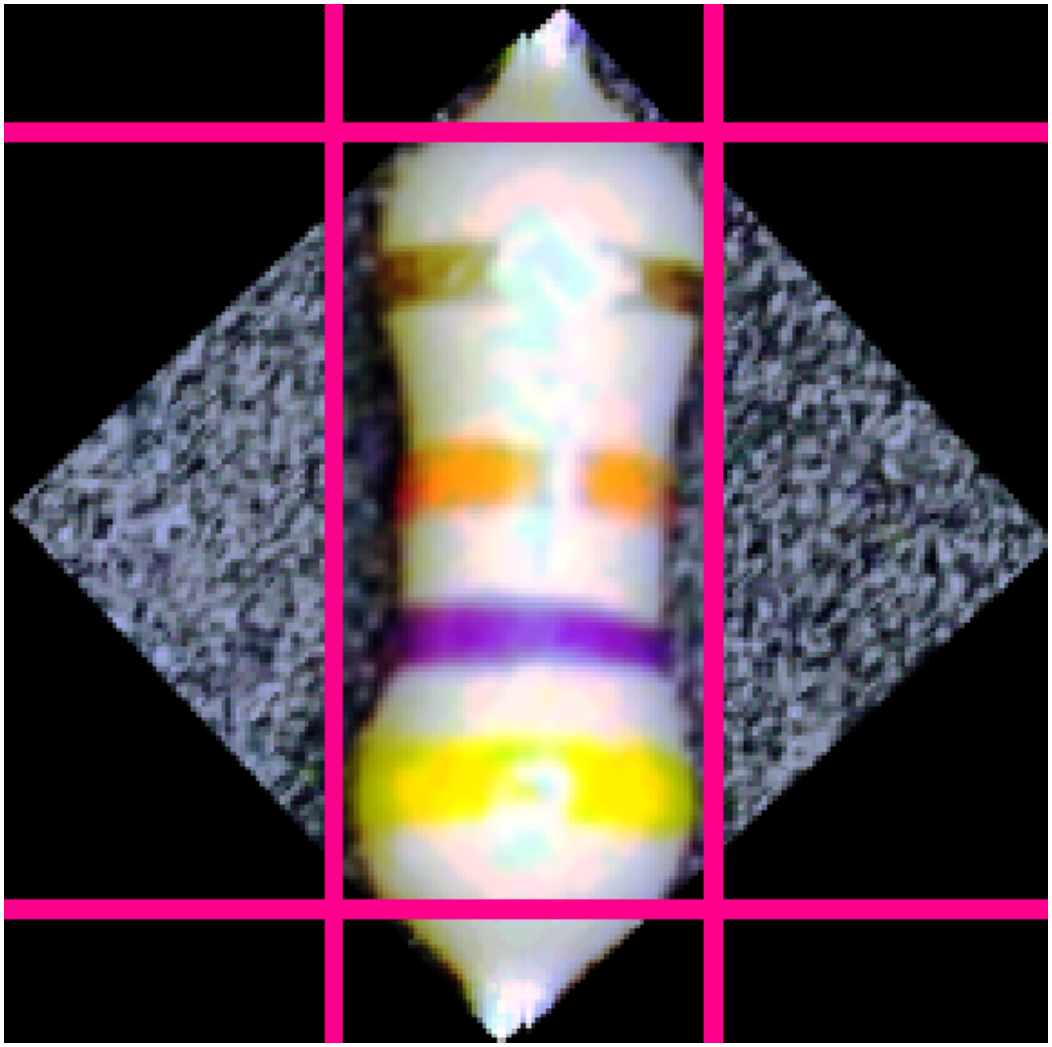

Figure 77: Post-Processing: Resistor decoding: vertical/horizontal bounds

HSV Filter

Minimum/Maximum

Hue: Ranges 0 to 1

Saturation: Ranges 0 to 1

Value: Ranges 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.

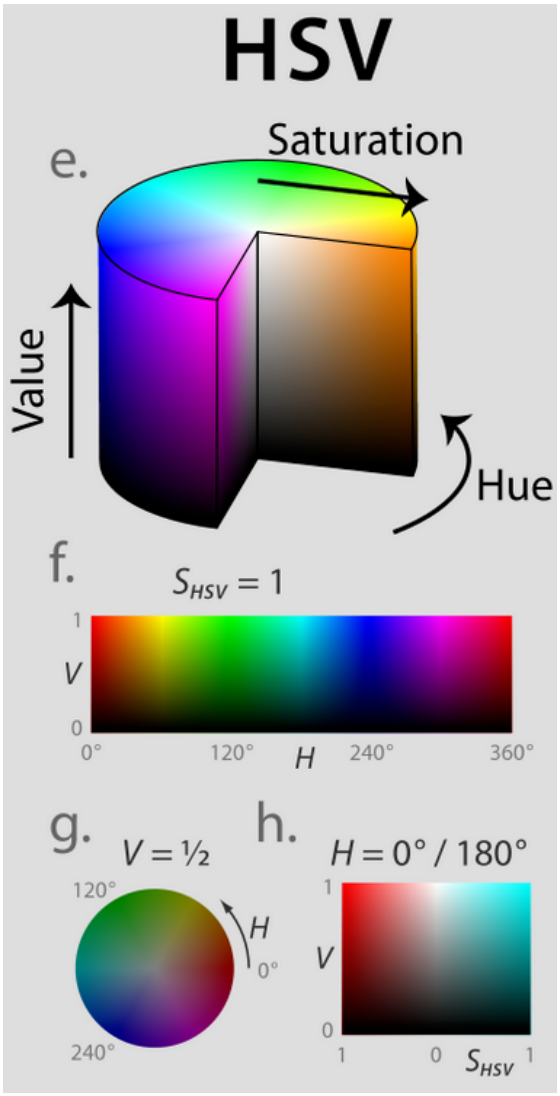

Figure 78: HSV diagram [28]

Steps

Further preparation processing

Slight blurring to average out the bands colors

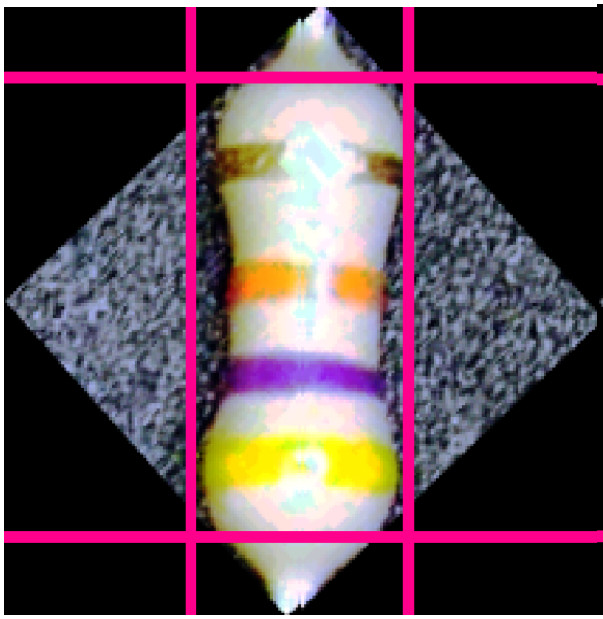

Figure 79: Post-Processing: Resistor: Before blur

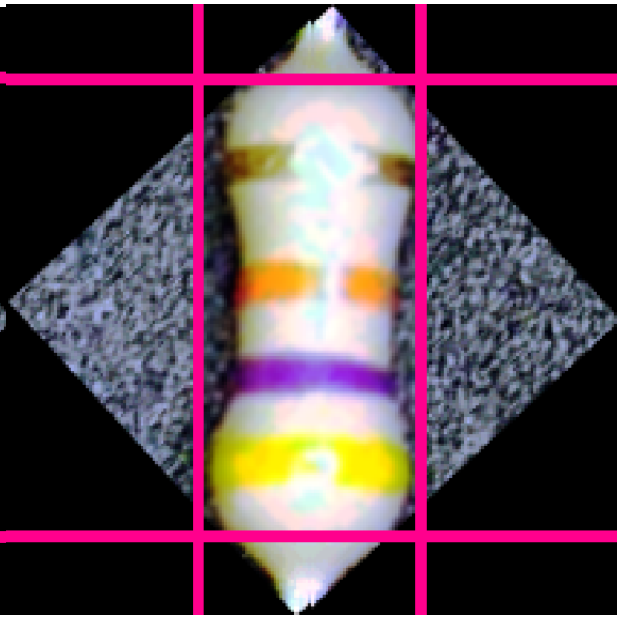

Figure 80: Post-Processing: Resistor: After blur

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

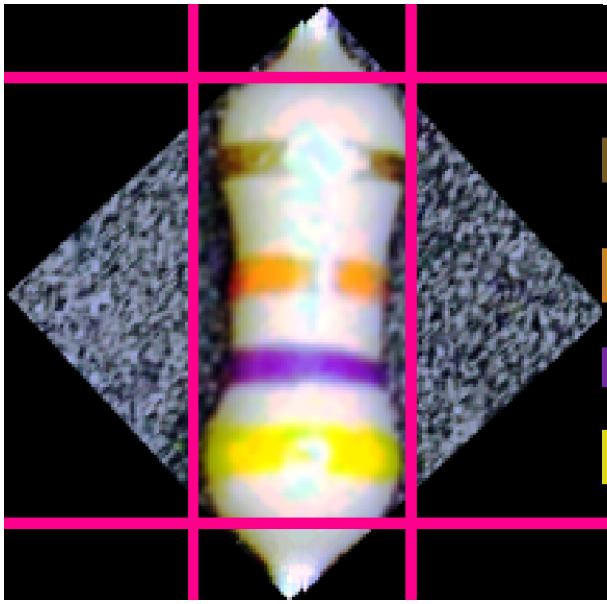

Figure 81: Post-Processing: Resistor: Before HSV filter

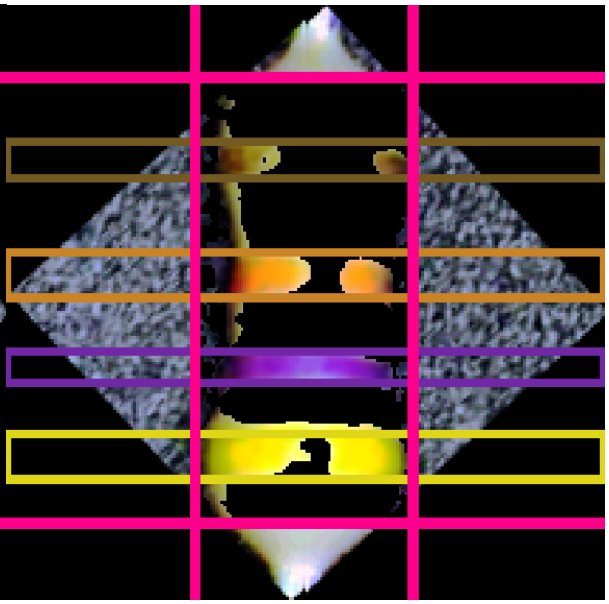

Figure 82: Post-Processing: Resistor: After HSV filter

Glare-destroyed information filtering

Over-exaggerated vertical crop from the middle middle, for demonstrational purposes

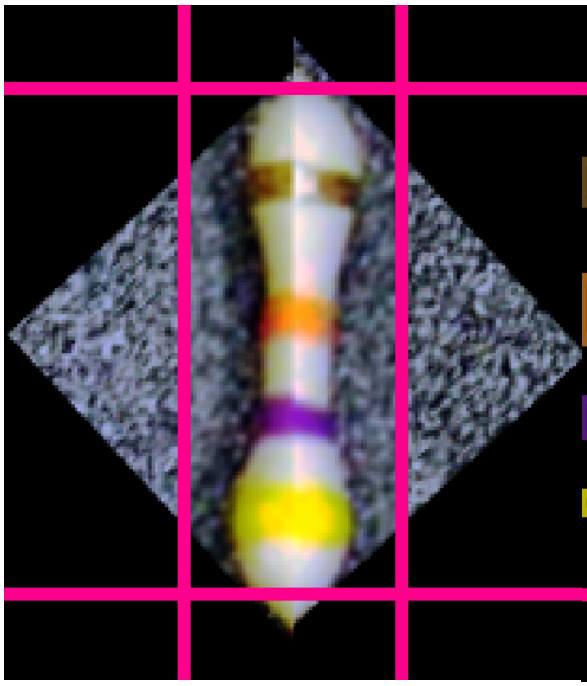

Figure 83: Post-Processing: Resistor: Cropping the middle vertical section (before HSV filter)

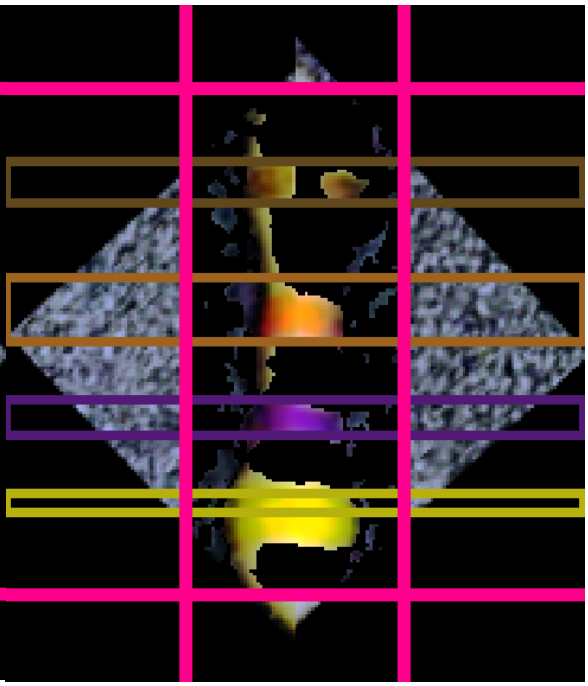

Figure 84: Post-Processing: Resistor: Cropping the middle vertical section (after HSV filter)

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

Initialisation

Average of every row's hue and count of non black (empty) pixels are obtained.

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.

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.

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, because there will be no band immediately after a band.

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 ~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, 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

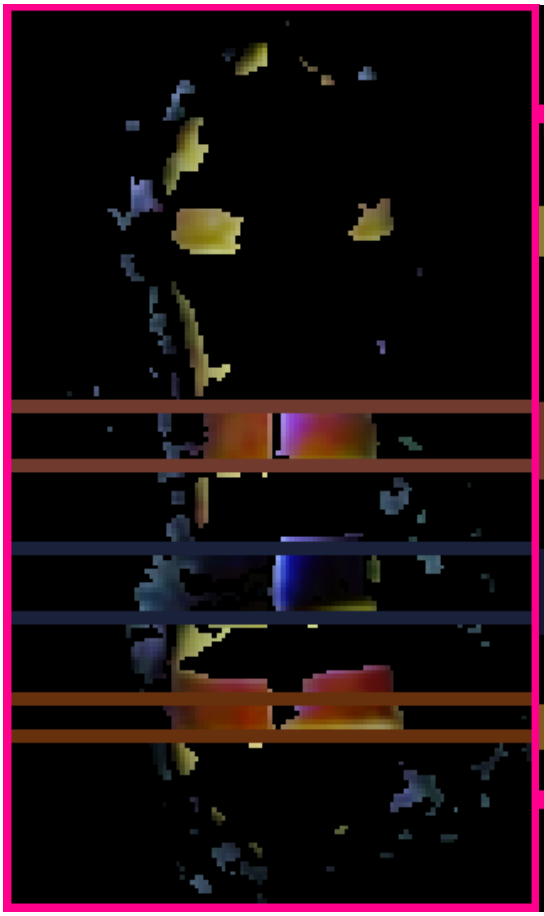

Figure 85: Post-Processing: Resistor: Boundaries of interest: Before boundary application

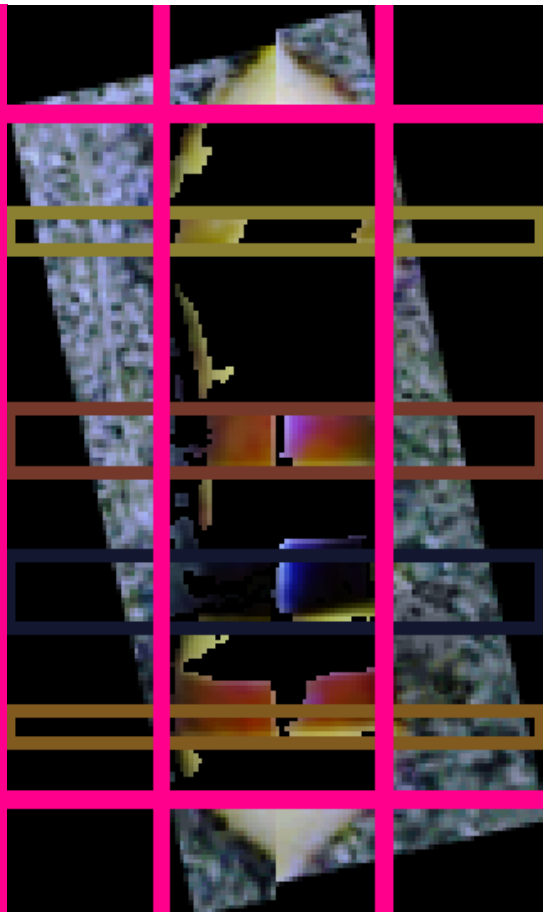

Figure 86: Post-Processing: Resistor: Boundaries of interest: After boundary application

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

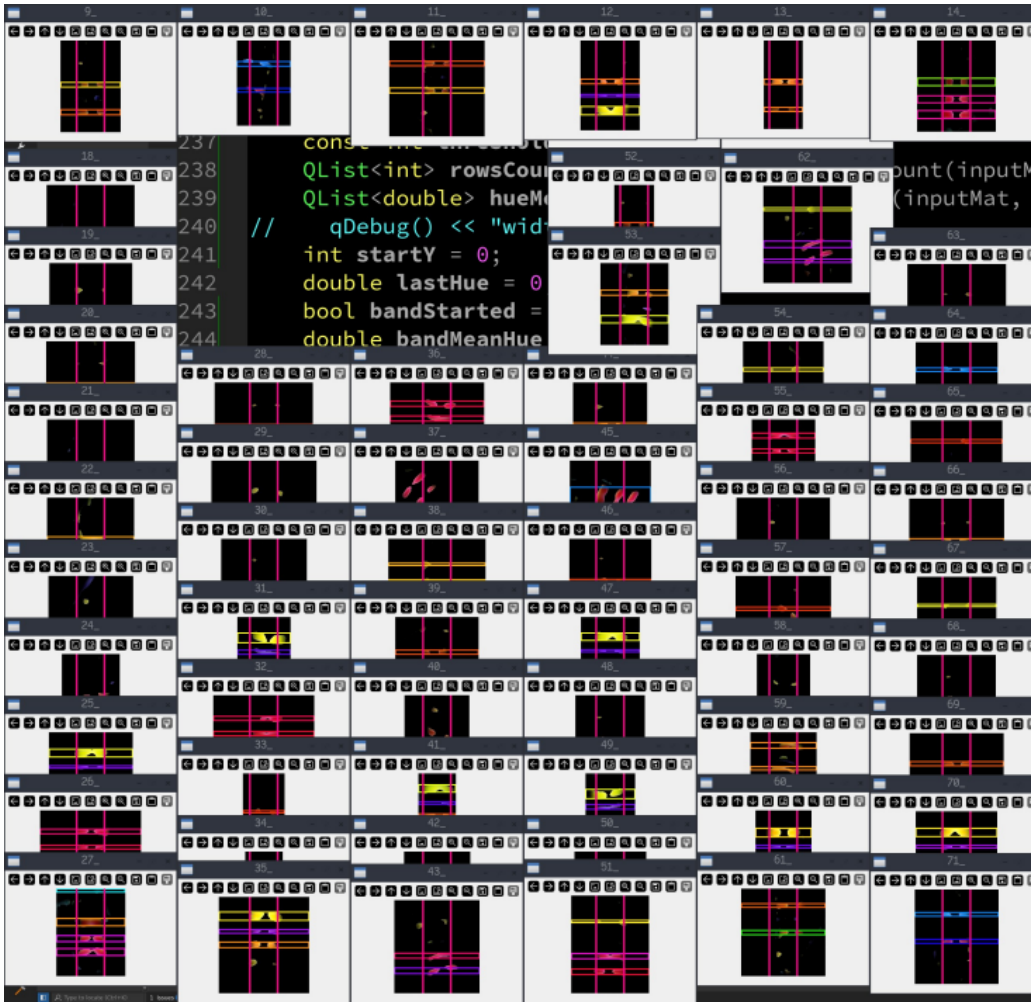

Figure 87: Post-Processing: Resistor: Bulk testing and calibration example 1

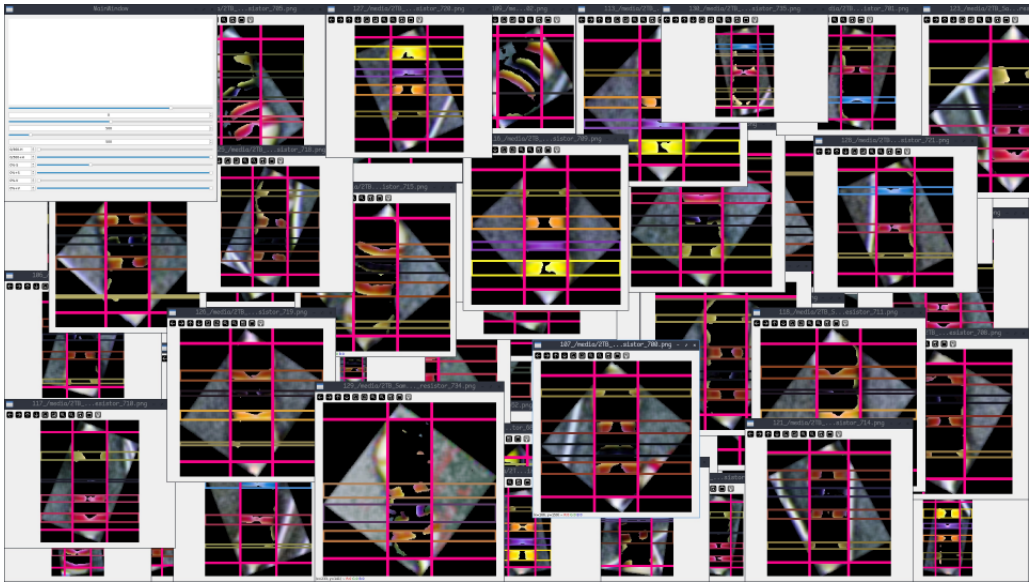

Figure 88: Post-Processing: Resistor: Bulk testing and calibration example 2

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.

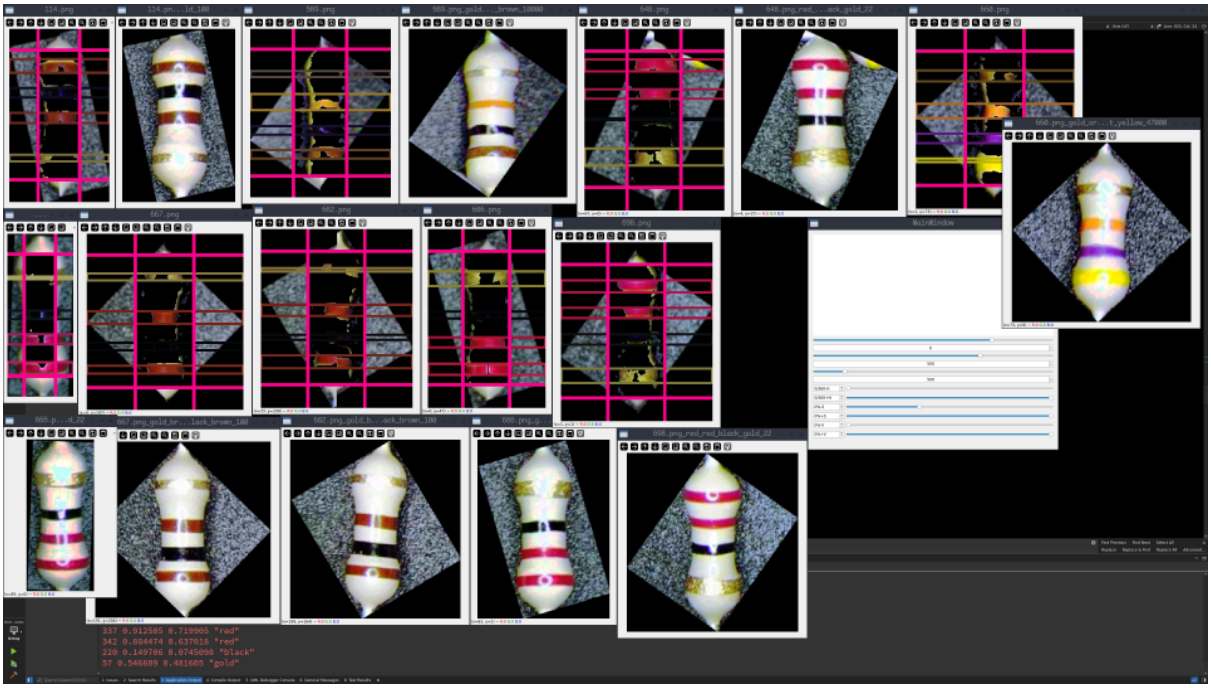

Figure 89: Post-Processing: Resistor: Testing and calibration, found values displayed at the end of the window name.

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.

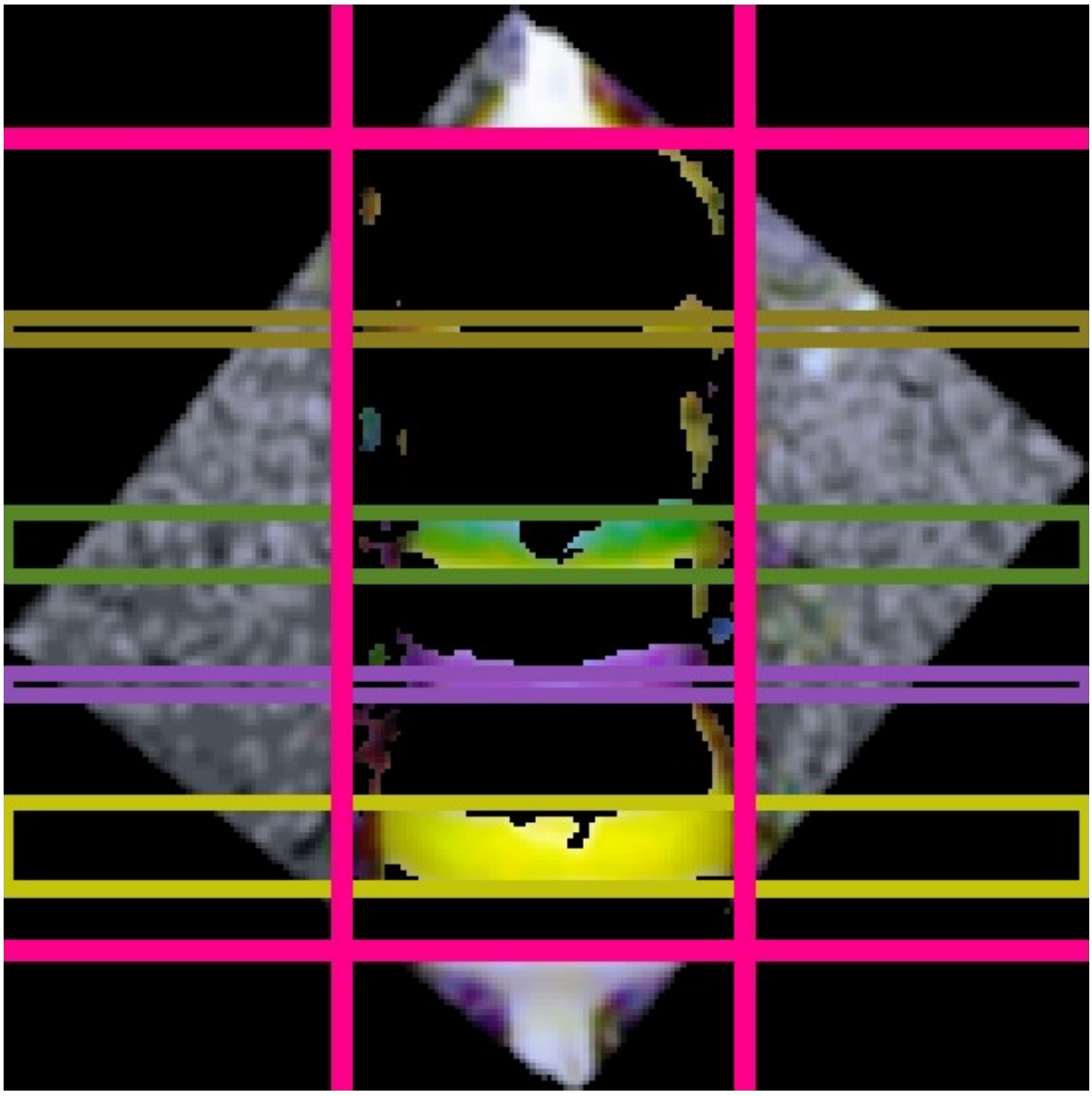

Figure 90: Post-Processing: Resistor: Successful implementation on a static image

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.

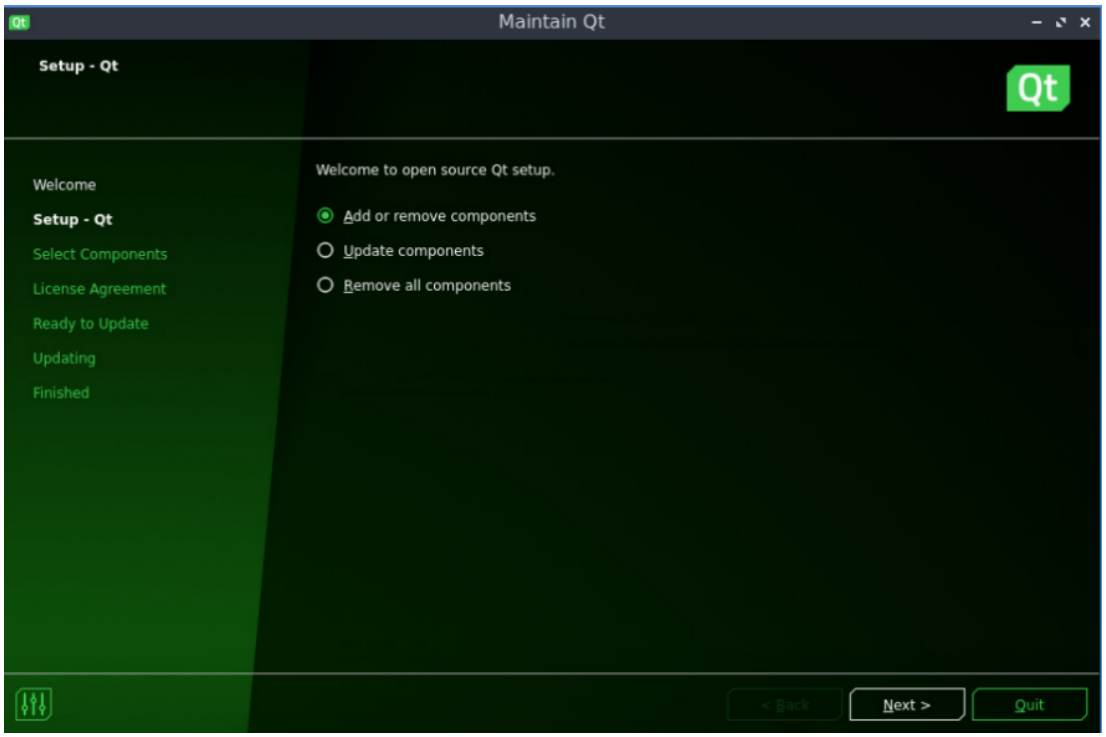

Figure 91: Qt Creator: Maintenance Tool: Add/Remove/Update choice

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

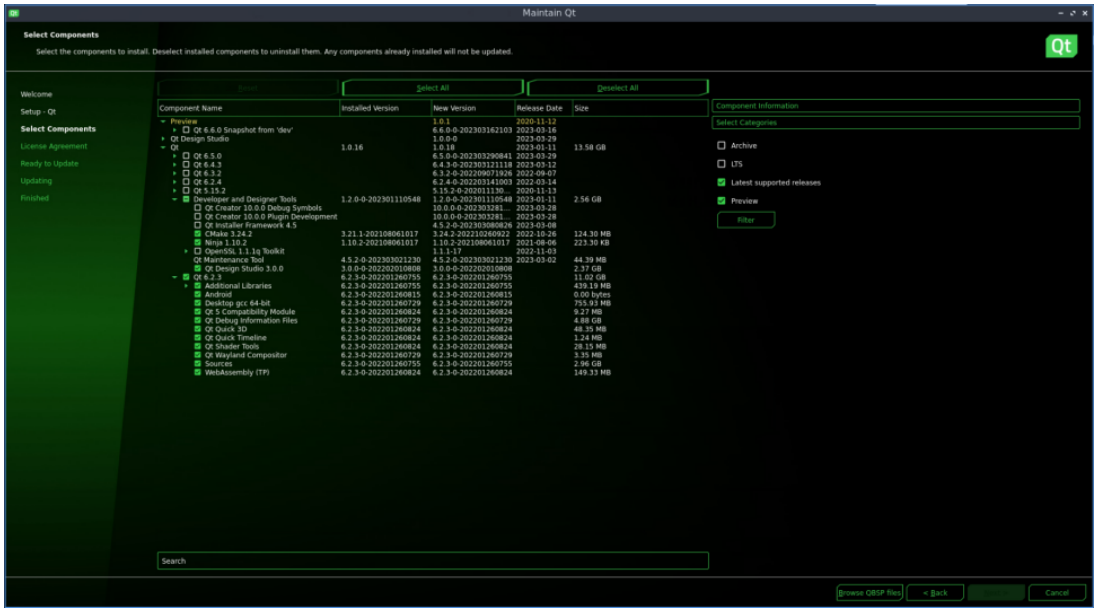

Figure 92: Qt Creator: Maintenance Tool: Component selection

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 93: Project timeline table

Figure 94: Project timeline

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

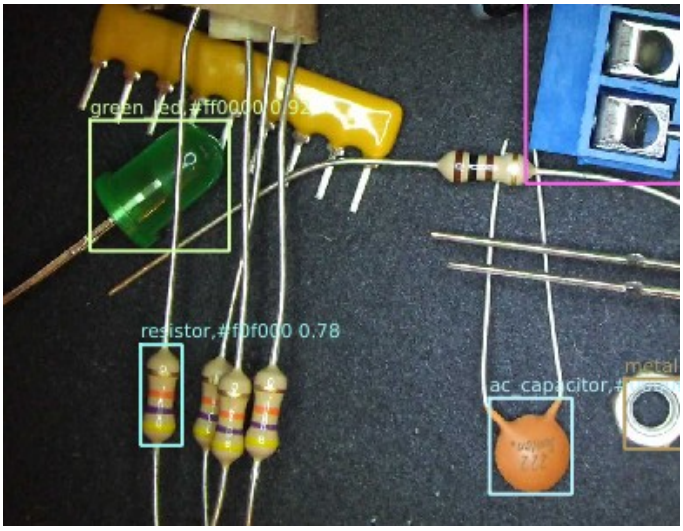

Figure 95: Plain text example

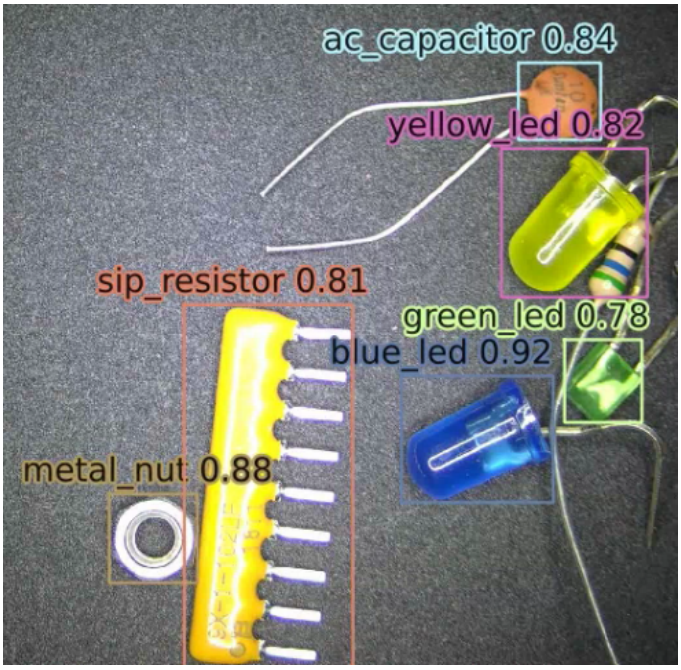

Figure 96: Contrast text example

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 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

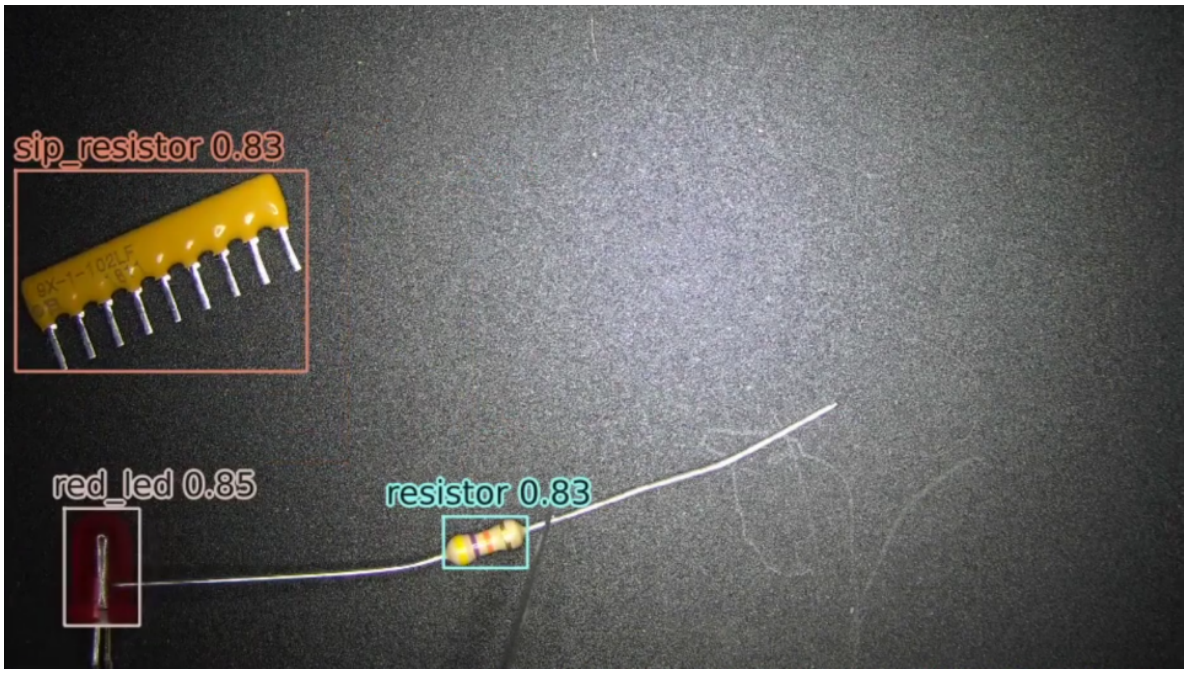

Figure 97: Horizontal text adjustment: left side

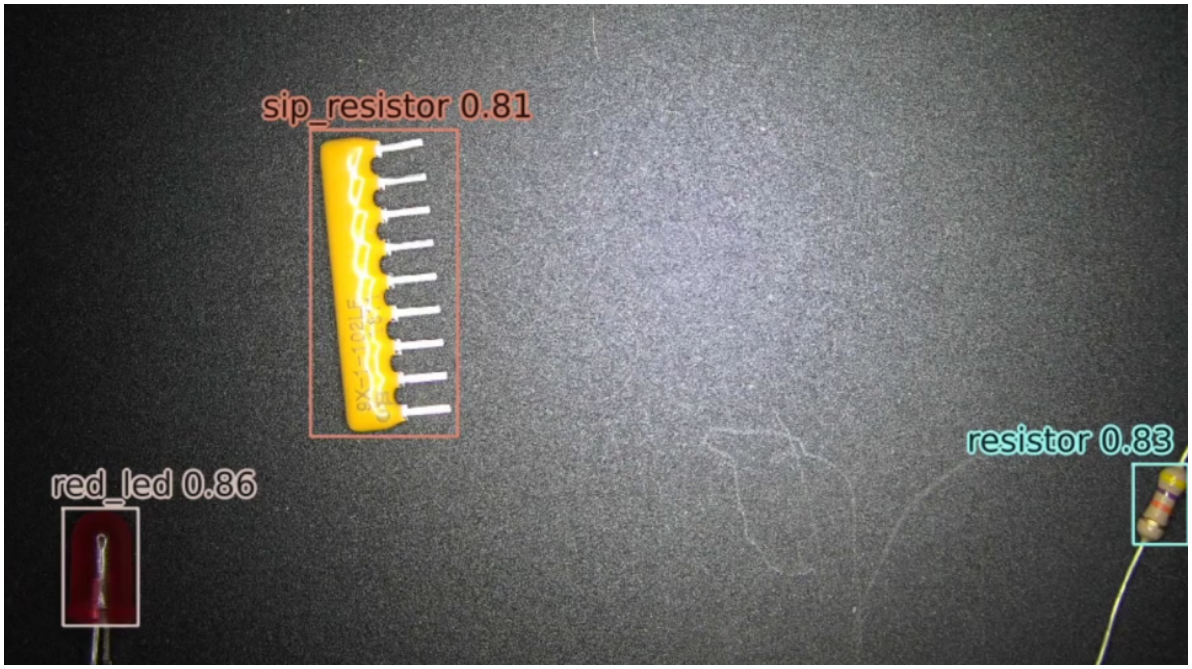

Figure 98: Horizontal text adjustment: middle

Figure 99: Horizontal text adjustment: right side

Post Processing

Figure 100: Post processing identifying resistor colors

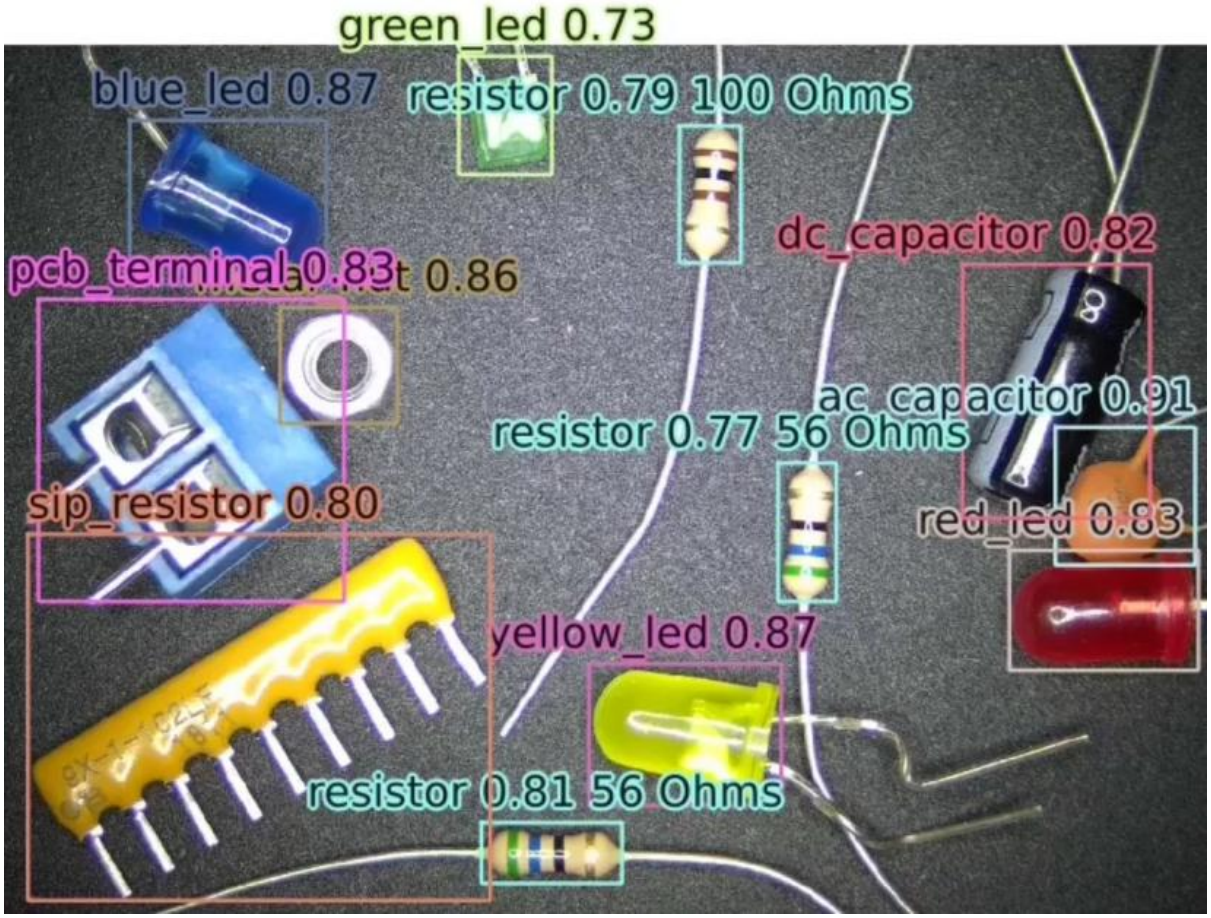

Figure 101: Post processing identifying ohm values

10.2. *Progress*

10.2.1. Issues encountered

Augmentation rotation issue

Description

A glitch in augmentation provided by YOLOv5 [23], where rotation during augmentation has shifted the bounding boxes of the components, causing inaccurate feedback to the model, preventing it from training appropriately.

Example

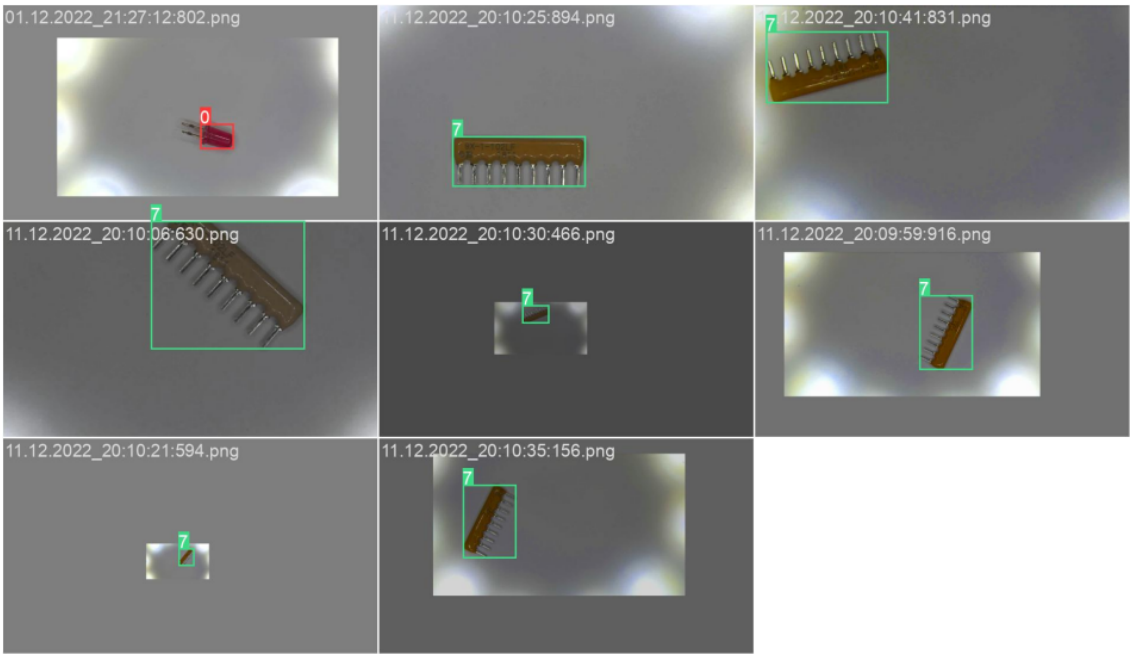

Figure 102: Rotation augmentation issue: Expected bounding boxes after rotation augmentation

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.

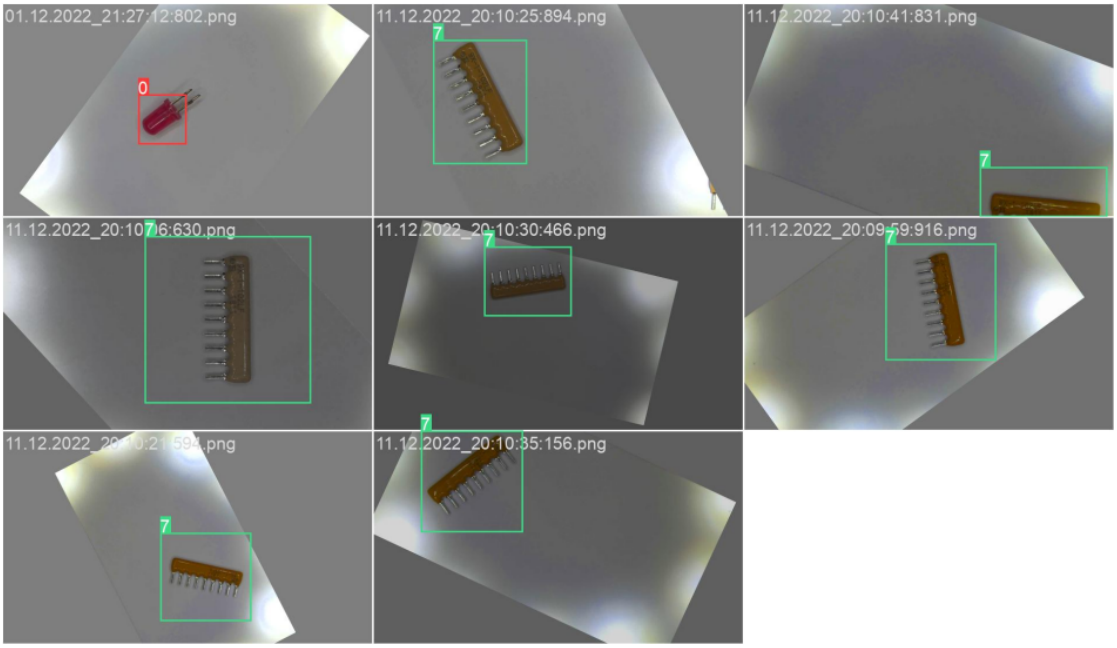

Figure 103: Rotation augmentation issue: Actual bounding boxes after augmentation

Note the unnecessarily expanded bounding boxes.

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

Submitted GitHub [24] issue

Link [25]

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 [26] augmentation also.

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

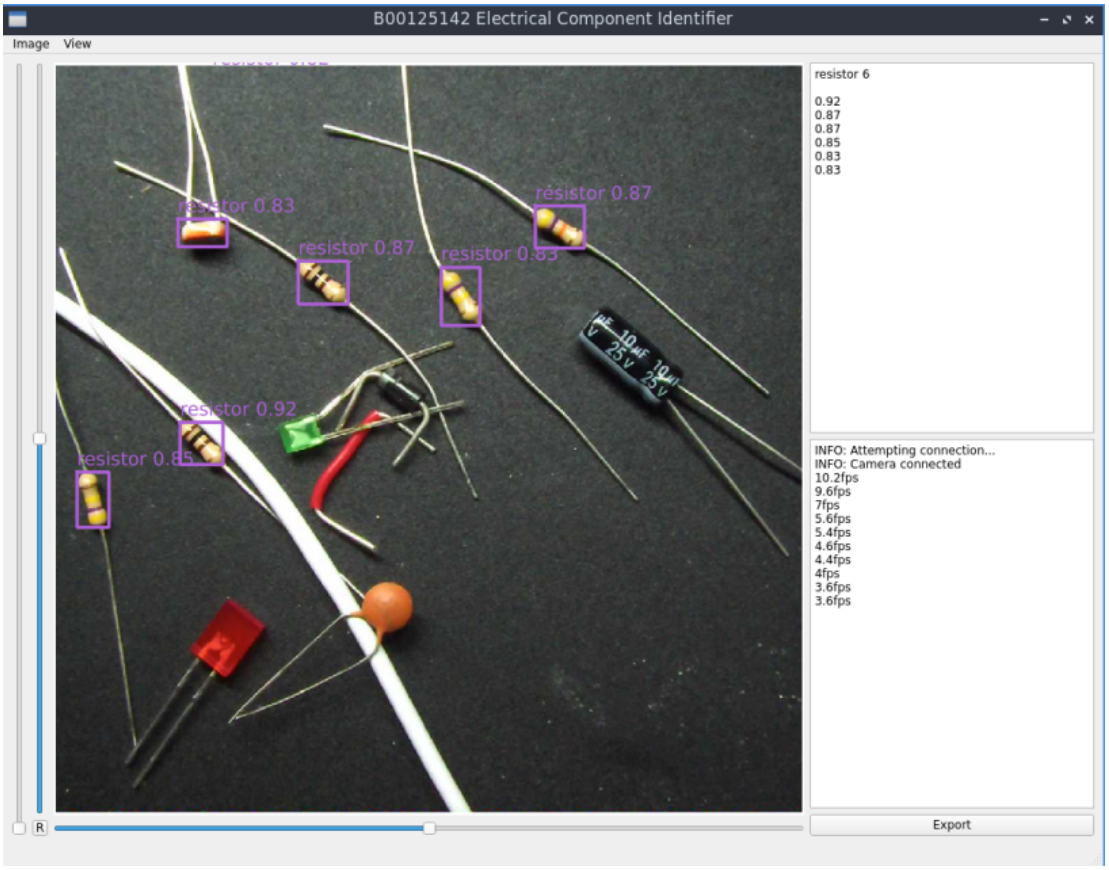

Figure 104: 1st batch model inference example

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 because 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, 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

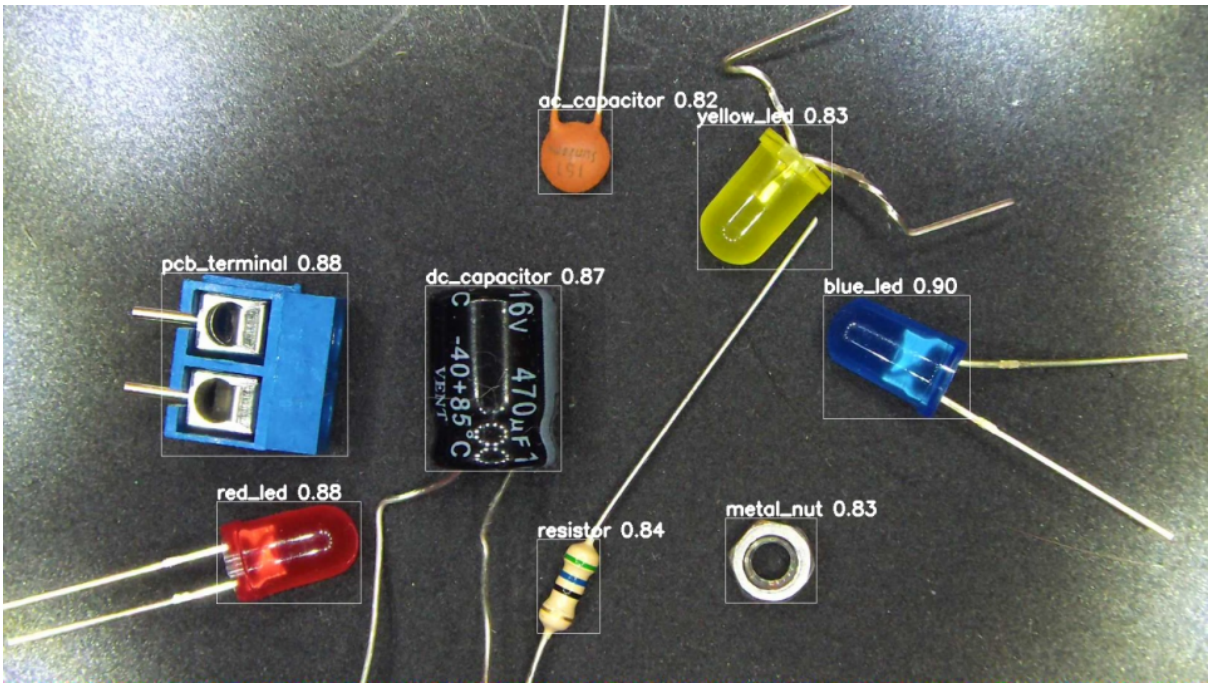

Figure 105: 3rd batch model inference example

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

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 train 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 outside of the time constraints for this project.

A further boost in performance was instead achieved by transitioning to a bigger model.

This of course came at a cost performance, dropping from camera side hardware limitation, to the laptop that the project was developed on limitation.

This is of course is not a viable option in the provided time constraint, but should be considered if the project were to be continued in the future.

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 a cost of a considerable amount of processing power – has yielded very impressive results that could've only been hoped for.

The original project concept at the time of project proposition had 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 has 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 has been expanded a few times – mobile deployment with a live video feed would simply not be feasible due to the lack of CUDA core availability on mobile devices, and the lack of a set rig environment that the project has been designed for.

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 be a different project, but it was very insightful to research the options 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 – while the project did not meet all the new goals – it did explore them in detail.

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

The project was an overall great success both as a learning experience, and as a utility tool it has set out to be.

The project was finished on a great state – 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.

[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] "Non Maximum Suppression: Theory and Implementation in PyTorch," Jun. 02, 2021. https://learnopencv.com/non-maximum-suppression-theory-and-implementation-in-pytorch/ (accessed Jan. 29, 2023).
[3] R. Girshick, "rbgirshick/rcnn." Jan. 29, 2023. Accessed: Jan. 29, 2023. [Online]. Available: https://github.com/rbgirshick/rcnn
[4] M. deGroot, "SSD: Single Shot MultiBox Object Detector, in PyTorch." Jan. 24, 2023. Accessed: Jan. 24, 2023. [Online]. Available: https://github.com/amdegroot/ssd.pytorch
[5] "YOLO detection documentation." https://web.cs.ucdavis.edu/~yjlee/teaching/ecs289g-winter2018/YOLO.pdf (accessed Jan. 29, 2023).
[6] "Embedded Software Development Tools | Cross Platform IDE | Qt Creator." https://www.qt.io/product/development-tools (accessed May 03, 2022).
[7] "C++," *Wikipedia*. Jan. 28, 2023. Accessed: Feb. 02, 2023. [Online]. Available: https://en.wikipedia.org/w/index.php?title=C%2B%2B&oldid=1135982045
[8] "ON AIR OSY C100 4 12 | WEB Cameras | Micro POV & Webcam | Cameras ... - New." https://www.adcom.it/en/cameras-and-drones/micro-pov-webcam/web-cameras/on-air-osy-c100-4-12/p_n_34_470_3457_55486 (accessed Apr. 23, 2023).
[9] "Buy a Raspberry Pi 4 Model B – Raspberry Pi." https://www.raspberrypi.com/products/raspberry-pi-4-model-b/ (accessed Jan. 24, 2023).
[10] "BeagleBoard.org - bone." https://beagleboard.org/bone (accessed Jan. 24, 2023).
[11] "Jetson Nano Developer Kit," *NVIDIA Developer*, Mar. 06, 2019. https://developer.nvidia.com/embedded/jetson-nano-developer-kit (accessed Jan. 24, 2023).
[12] "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).
[13] "Android," *Android*. https://www.android.com/intl/en_ie/ (accessed Jan. 29, 2023).
[14] "Experience the Power of Windows 11 OS, Computers & Apps | Microsoft." https://www.microsoft.com/en-ie/windows (accessed Jan. 29, 2023).
[15] "Linux.org," *Linux.org*, Jan. 26, 2023. https://www.linux.org/ (accessed Jan. 29, 2023).
[16] "Mac," *Apple (Ireland)*. https://www.apple.com/ie/mac/ (accessed Jan. 29, 2023).
[17] "AMD Ryzen™ 7 5800X3D Gaming Processor." https://www.amd.com/en/products/cpu/amd-ryzen-7-5800x3d (accessed Jan. 29, 2023).
[18] "Welcome to AMD." https://www.amd.com/en/home (accessed Jan. 29, 2023).
[19] "PC Components | Gaming Gear | CORSAIR." https://www.corsair.com/ww/en/ (accessed Jan. 29, 2023).
[20] "VENGEANCE RGB PRO SL 16GB (2x8GB) DDR4 DRAM 3200MHz C16 Memory Kit – White." https://www.corsair.com/ww/en/Categories/Products/Memory/Vengeance-RGB-PRO-SL-White/p/CMH16GX4M2E3200C16W (accessed Jan. 29, 2023).
[21] "NVIDIA GeForce RTX 3060 Family," *NVIDIA*. https://www.nvidia.com/en-gb/geforce/graphics-cards/30-series/rtx-3060-3060ti/ (accessed Jan. 29, 2023).
[22] "World Leader in AI Computing," *NVIDIA*. https://www.nvidia.com/en-gb/ (accessed Jan. 29, 2023).
[23] "ultralytics/yolov5." Ultralytics, Nov. 06, 2022. Accessed: Nov. 06, 2022. [Online]. Available: https://github.com/ultralytics/yolov5
[24] "Build software better, together," *GitHub*. https://github.com (accessed May 03, 2022).
[25] "Degrees rotation augmentation messes up the size of the labels · Issue #10639 · ultralytics/yolov5," *GitHub*. https://github.com/ultralytics/yolov5/issues/10639 (accessed Jan. 29, 2023).

[26] K.-Y. Wong, "Official YOLOv7." Jan. 29, 2023. Accessed: Jan. 29, 2023. [Online]. Available: https://github.com/WongKinYiu/yolov7
[27] "C104-25 Disc Capacitor, 0.1uf 25v (104z)." https://www.rfparts.com/c104-25.html (accessed Jan. 29, 2023).
[28] "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
[29] "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).
[30] 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).
[31] "Detection - Ultralytics YOLOv8 Docs." https://docs.ultralytics.com/tasks/detection/ (accessed Jan. 24, 2023).
[32] "Official Adobe Photoshop | Photo and design software." https://www.adobe.com/ie/products/photoshop.html (accessed Jan. 29, 2023).
[33] "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=2d319059a2f6424782f0b6314c8312fc-1673102021784-03474-_mLKffyW&af=&cv=&cn=&dp= (accessed Feb. 02, 2023).
[34] "Pine Tree Facts, Types, Identification, Diseases, Pictures," *Coniferous Forest*. https://www.coniferousforest.com/plants-trees/pine (accessed Jan. 29, 2023).
[35] 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.
[36] 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.
[37] "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).
[38] "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).