IMPERIAL

A Vision-Assisted Mechatronic Component Sorter

MEng Final Year Report

Author
Shaheen Amin
CID: 01866464

Supervised by
Dr. E. Stott
Dr. T. Constandinou

A thesis submitted in fulfilment of requirements for the degree of
Master of Electronic and Information Engineering

Department of Electrical and Electronic Engineering
Imperial College London
2024

PLAGIARISM DECLARATION

I affirm that I have submitted, or will submit, an electronic copy of my final year project report to the provided EEE link.
I affirm that I have submitted, or will submit, an identical electronic copy of my final year project to the provided Blackboard module for Plagiarism checking.
I affirm that I have provided explicit references for all the material in my Final Report that is not authored by me but is represented as my work.
I have used GPTv4, as an aid in the preparation of my report. GPTv4 was used for LaTeX formatting and spellchecking, however, all technical content is original.

ABSTRACT

To address the issue of electronic waste and cluttered workspaces in the Level 1 Labs, this project aims to develop an automated system for identifying and sorting electronic components. Utilising Computer Vision techniques, the system will classify electrical components such as resistors, capacitors, LEDs and inductors, and sort them into designated bins. The project consists of three core components: the mechanical design of the system, the integration of electronics and software, and the development of a Computer Vision system. This project aims to efficiently resolve these issues.

ACKNOWLEDGEMENTS

I would like to thank my parents, who have done nothing but support me throughout my academic journey. Their sacrifices have enabled me and my siblings to pursue our dreams, and I am forever grateful for their unwavering support.
I would also like to thank my supervisor, Dr. Ed Stott, for his guidance and expertise when it came to the technical aspects of the project. His insights and feedback have been invaluable in shaping the project into what it is today.
Finally, I would like to thank my wonderful friends and classmates, who have all been a source of inspiration and motivation throughout my time at Imperial. Their love and support has truly been a light in the dark, and I am eternally grateful for their presence in my life.

Contents

1 PROJECT SPECIFICATION

Contents

The motivation for this project is three-fold; to solve the problem of electronic waste and cluttered workspaces in the Level 1 Electrical Engineering Labs; to alleviate the time-consuming process of sorting electronic components from the Labs' technicians; and to bring the Lab closer to meeting the requirements for the LEAF (Laboratory Efficiency Assessment Framework) certification [1] that the Lab is currently working towards. The LEAF certification is a framework that aims to improve the efficiency of laboratories by reducing waste, energy consumption, and costs. This project aligns with the LEAF certification's goals by reducing the amount of electronic waste produced by the Lab.

- Resistors
- Capacitors
- Ceramic Capacitors
- Inductors
- Diodes
- MOSFETs
- Transistors
- LEDs
- Wires
- Integrated Circuits

Examples of components that are commonly found in the Level 1 labs are shown in the above table.

1.1 Project Goals

The project aims to achieve these goals by employing state-of-the-art computer vision techniques to classify various electronic components, and then sort them into designated bins. The project must be able to meet the following requirements:

Vision System: The system must be able to accurately classify the different electrical components used in the Level 1 labs. It should operate in real-time and be able to classify components as they move along the conveyor belt with reasonable accuracy. It is essential that the means of classification is not too computationally expensive, as the system must be able to classify components in real-time as they move along the conveyor belt.

Mechanical System: The system must be able to sort the classified components into designated bins. The system will make use of a mechanical design to physically move the components into the correct bins, and its design must be easily producible and cost-effective. The mechanical design should be easy to disassemble and reassemble for maintenance purposes, and therefore it must also be robust and reliable. Standard parts should be used in the design where possible to reduce costs and increase the ease of maintenance.

Electronics: The electronics of the system should be able to control the mechanical system and the vision system, and must be safe for use in a laboratory environment. The electronics must be able to control the stepper motors and sensors used in the mechanical system, and the camera used in the vision system. Careful consideration must be given to the power requirements of the system, and the electronics must be able to handle the power requirements of the mechanical system and the vision system, to ensure that the system operates correctly and safely.

User Interface: The system must have a user interface from which to observe and control the system's state. The user interface should be easy to use and intuitive, and should provide feedback to the user

about the system's state. The user interface should also provide the user with the ability to control the system, for example, to start and stop the system.

Concurrency and Error Handling: The system must be able to handle multiple components on the conveyor belt at once, and must be able to handle errors that may occur during operation. Due to the nature of the system, having multiple subsystems with cross dependencies, it is essential that the system can handle errors gracefully and recover from them without causing damage to the system or the components being sorted. It is essential that the system software employs multiprocessing or threading to handle the concurrent operation of the different subsystems to mitigate latency to the user.

This report seeks to document the development of the project and provide justification for the design decisions made throughout the project. The report will also evaluate the project's performance in terms of accuracy, efficiency, and usability, and suggest areas for future work.

Discussion of existing solutions and relevant literature is explored in Section 2, the design and system architecture of the project is discussed in Section 3, the implementation of the project is discussed in Section 4, and the evaluation of the project is discussed in Section 6, with tests in Section 5. The report concludes with a discussion of the project's limitations and suggestions for future work in Section 7. The repository of this project, including all design files and code, can be found in the Appendix 9.1.

2 BACKGROUND

Contents

This chapter will delve into the background of the project, exploring the existing literature on computer vision systems for component identification, real-time computer vision architectures, and the mechanical design of existing sorting machines. This research will inform the design of the project and its various systems. Research into these topics will inform the decisions made in the design of the project and its various systems.

2.1 Computer Vision Techniques

A range of computer vision techniques have been explored in the literature, ranging from PCA (Principle Component Analysis) by Dhenge et al. [2] to more modern computer vision techniques like CNNs as used by Xu et al. [3], Chand and Lal [4], and very recently, Vision Transformers (ViTs) by Dosovitskiy et al. [5]. Given the rate of advancement of computer vision techniques, the techniques used in the literature reflect the state of the art at the time of writing, and so are not necessarily the most viable current techniques for the project, but they inspire novel approaches to the problem of component identification.

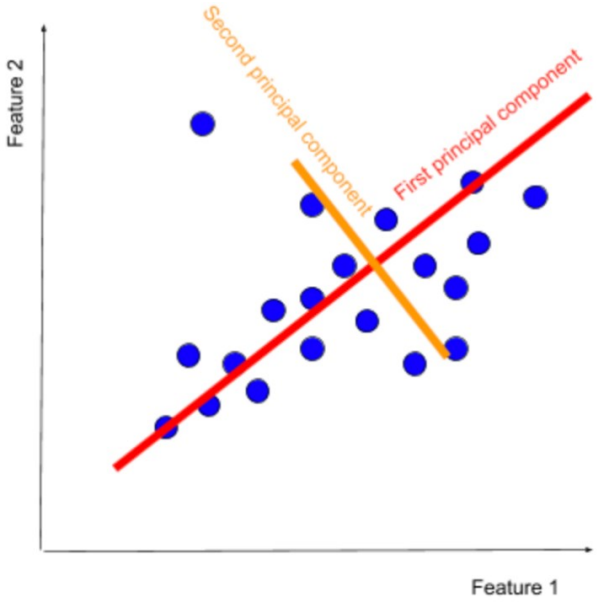


Figure 1: Resistor Test Set [6]

PCA with ANN (Artificial Neural Networks) as used by Dhenge et al. [2] is a relatively simple but outdated statistical technique that was successful in identifying nuts and bolts. Principal Component Analysis (PCA) is a widely used unsupervised machine learning algorithm primarily for dimensionality reduction, data compression, and noise reduction. PCA works by identifying the principal components, or orthogonal vectors, that best represent the data, in order to reduce the data's dimensionality, as shown in Figure 2. The dataset is then projected onto these principal components, which can then be used for training a classifier. The paper uses PCA for feature extraction, and then uses an ANN (Artificial Neural Network) to classify the components, achieving an accuracy of 95%. This is a very promising result, as it shows that PCA is a viable approach to component identification. However, it is important to note that PCA is a relatively outdated technique, and may not be the most viable approach to the problem of component identification, as more advanced techniques like CNNs are known to be much more effective at feature extraction.

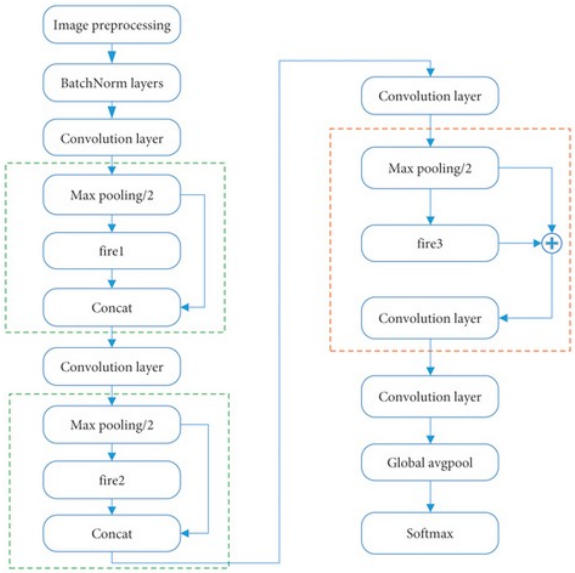


Figure 2: Faster SqueezeNet CNN architecture [3]

Although a valid approach by Dhenge et al. [2], CNNs (Convolutional Neural Networks) are known to be much more effective at feature extraction, and do not "have the problem of low recall and accuracy", as mentioned by Xu et al. [3], that PCA may suffer from. PCA excels at identifying components that are very distinct, like nuts and bolts, but may struggle with components that are more similar, like different types of capacitors, for example. Convolutional Neural Networks (CNNs) make use of convolutional layers to extract features from the image, which are then used to classify the image, often layering multiple convolutional filters to extract more complex features. They are an ever-evolving field, with new architectures being developed frequently, meaning that they are always at the cutting edge of computer vision research. To support the claim of CNNs effectiveness at feature extraction over PCA, Xu et al. [3] use the Faster SqueezeNet CNN architecture to identify 22 different subcategories of electronic components, specifically resistors, capacitors, and inductors, which is directly applicable to the project. They achieve a TPR (True Positive Rate) of 99.999% with only a 2.67ms average inference time on a GTX 1050 2GB GPU (released in 2016), which would translate to 374 FPS (frames per second); a very impressive result. This work helps to support the claim that a CNN is a more viable approach to the problem of component identification, and so this will inform the design of the project's computer vision system.

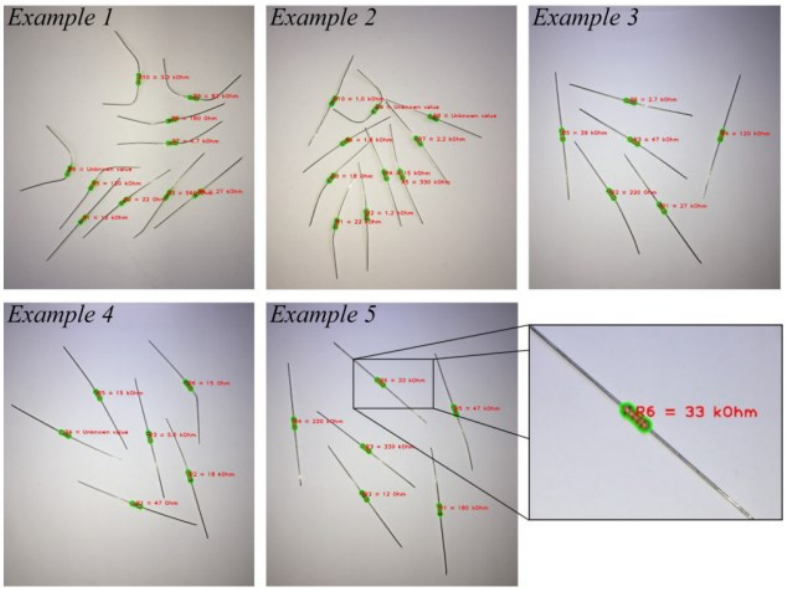


Figure 3: Resistor Test Set Muminovic and Sokic [7]

The paper by Muminovic and Sokic [7] discusses the use of an SVM (Support Vector Machine) to characterise resistors, by identifying the resistor's centroid (centre of mass), determining the resistor's orientation, and then analysing the bands to determine the resistor's value. This is directly applicable to the project, as it is a viable approach to identifying resistors, which are very distinct from other components, given the presence of colour bands. The paper's novel approach to identifying the resistor values allows it to achieve high accuracy of 86%. However, this is calculated on a per-resistor basis — an incorrect band does not mean the entire resistor is incorrect, which is not the case in the project's vision system, where the entire component must be identified correctly otherwise it will be sorted incorrectly. The images are also taken from a distance as shown in Figure 3, which may make the bands appear smaller than they would in the project's vision system, which may affect the accuracy of the approach.

The inference time seems to be incredibly fast, only being 0.7ms, although it is important to note this was run on an Intel® Core™ i5-6200U CPU, rather than an edge device like the Raspberry Pi 4, which may have a slower inference time. In any case, it shows that the SVM is a viable approach to consider for the project.

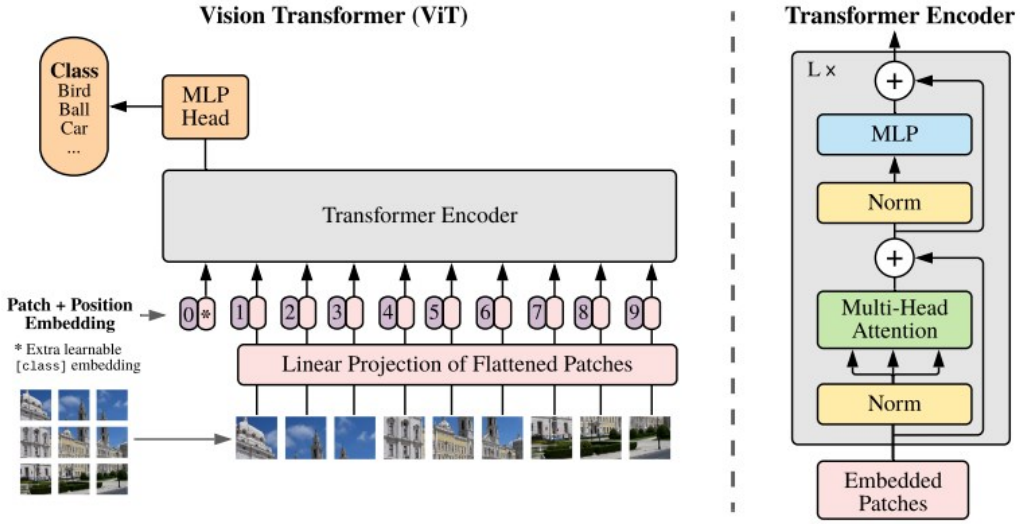


Figure 4: Vision Transformer [5]

Vision Transformers, as discussed by Dosovitskiy et al. [5], are a novel approach to computer vision that has shown to be very effective at image classification tasks, achieving an accuracy of 88.5% on the ImageNet dataset, which is a very challenging dataset for image classification. As described by the authors, they make use of self-attention mechanisms, as shown in Figure 4, to allow the model to learn long-range dependencies in the image, which is a key advantage of the Vision Transformer over CNNs. This is a very promising result, as it shows that Vision Transformers are a viable approach to computer vision tasks, and may be a viable approach to the problem of component identification. However, it is important to note that Vision Transformers are a very new architecture, and so may not be as well-documented as CNNs, which may make it more difficult to implement in the project. Due to their novelty, they also tend to be more computationally expensive that other architectures as explained by the authors Li et al. [8], who compare and optimise the Vision Transformer architecture to make it more in-line with the efficiency of CNNs. This is a key consideration for the project, as the system must be able to run in real-time on the Raspberry Pi 4, which may not be possible with the traditional Vision Transformer architecture.

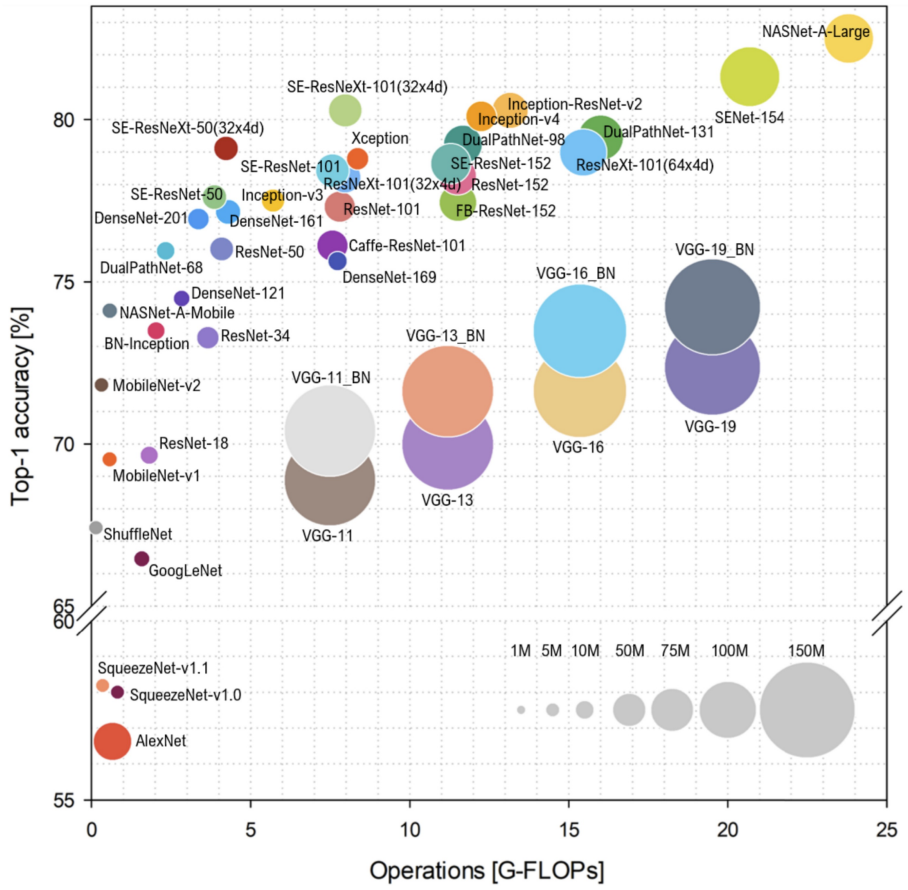


Figure 5: CNN architecture comparisons [9]

Although all the approaches outlined above have potential for the project, the most viable approach is a CNN, as it is a well-established technique for computer vision tasks, and is known to be very effective at feature extraction. The paper by Ghimire et al. [9] explores the efficiency of different CNN architectures, as shown in Figure 5, and details different methods to optimise the architectures for real-time inferencing. The sheer number of different CNN architectures shows the complexity of the field, and the difficulty in choosing the most appropriate architecture for the project, but also demonstrates their flexibility and ability to be tailored to the specific requirements of the project. The paper also explores the use of different optimisation techniques to reduce the computational complexity of the models, which is a key requirement of the project, as the system must be able to run in real-time on the Raspberry Pi 4.

2.2 Convolutional Neural Networks Architectures

This section will explore the different CNN architectures that are commonly used in computer vision tasks, and their effectiveness at identifying objects in real-time. A requirement for the chosen architecture is that it must be able to run in real-time on the Raspberry Pi 4, which has a 1.8GHz quad-core 64-bit ARM Cortex-A72 CPU and up to 8GB of RAM [10], however the Pi must not be overwhelmed by the computational complexity of the model, as other subsystems in the project must also run concurrently. This requires the Computer Vision system to be both computationally efficient and accurate, which is a difficult balance to strike.

For the explicit purpose of identifying electronic components, the paper by Chand and Lal [4] compares a range of different object detection architectures, including YOLOv3, YOLOv4 and Faster SqueezeNet, with YOLOv4 achieving a mAP (mean Average Precision) of 98.6%. The mean average precision is defined by the following formula:

$$\text{mAP} = \frac{1}{n} \sum_{i=1}^{n} \text{AP}_i$$

Where n is the number of classes, and $\texttt{AP}_i$ is the average precision for class i. The average precision is defined as the area under the precision-recall curve, and is an objective measure of the model's performance. For mAP^{50}, the IoU (Intersection over Union) threshold is defined as being 50%, meaning that as long as there is at least 50% overlap between the predicted bounding box and the ground truth bounding box, the prediction is considered correct. In the formula below, A is the ground truth bounding box, and B is the predicted bounding box, with $\text{A} \cap \text{B}$ being the intersection of the two bounding boxes, and $\text{A} \cup \text{B}$ being the union of the two bounding boxes.

$$\text{IoU} = \frac{\text{A} \cap \text{B}}{\text{A} \cup \text{B}}$$

YOLO (You Only Look Once) is a real-time object detection architecture, that is used very commonly in computer vision applications and is very effective at identifying objects in real-time Terven and Cordova-Esparza [11]. It makes use of a single CNN that takes in an image and outputs a list of bounding boxes and class probabilities for each bounding box. This is in contrast to other object detection architectures, such as R-CNN, which uses a CNN to propose regions of interest and then uses a second CNN to classify the regions of interest. Other papers like Guo et al. [12] also comment on YOLOv4's effectiveness at identifying electronic components, achieving 93.94% mAP on a dataset of 20 different components, and the paper also comments on YOLOv4's ability to run in real-time, achieving 67 FPS, albeit on a powerful NVIDIA TITAN Xp GPU. For the Raspberry Pi 3B, the paper by Sismananda et al. [13] has shown to run YOLOv3 at a very low 1 FPS, with an IoU (Intersection over Union) accuracy of 86.7%, which is relatively low.

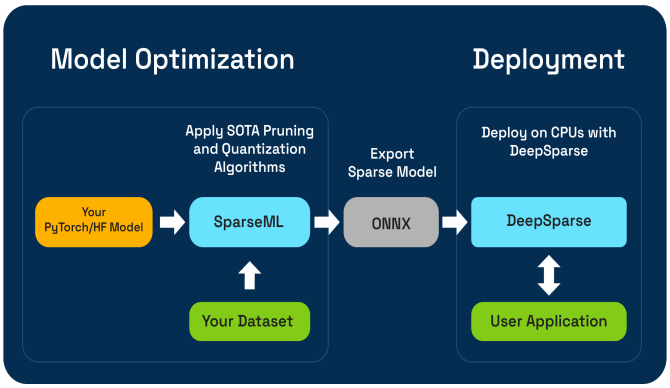


Figure 6: SparseML Pipeline [14]

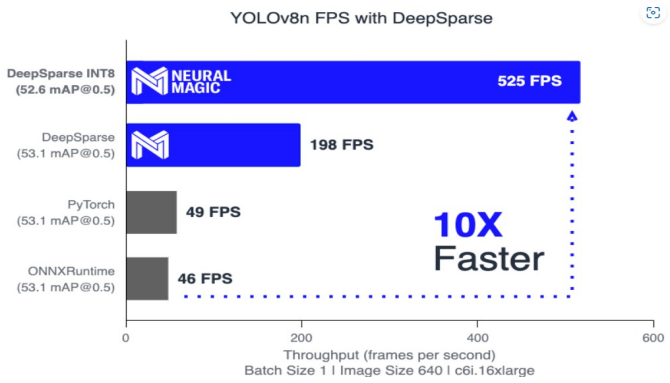


Figure 7: DeepSparse Performance [15]

However, efforts made by Neural Magic [15] in the optimisation of YOLOv5 and YOLOv8 show performance improvements of up to 10x on CPUs, through their open-source optimisation toolkit SparseML [14] and CPU inferencing runtime, DeepSparse [16] as shown in Figure 6 and Figure 7. This is an incredibly promising result, as it shows that it is possible to run YOLOv8 in real-time on a CPU, which may help to achieve real-time inferencing on the Raspberry Pi 4, which is a key requirement of the project.

SparseML is a framework that helps automate the optimisation of deep learning models by employing sparsification techniques (the removal of unimportant weights from the model) and quantisation techniques (the reduction of the precision of the weights in the model) to reduce its computational complexity, and therefore its inference time. SparseML makes use of "recipes" to define the optimisation process, which can be customised to the specific requirements of the model, and can be used to optimise a range of different deep learning models, including YOLOv8. Pre-sparsified models exist on their "Model Zoo", which can be used to train models from scratch, or to fine-tune existing models.

DeepSparse is a CPU inferencing runtime that is designed to take sparsified models and run them on CPUs, with the aim of achieving real-time inferencing on CPUs. it is not neccessary to use DeepSparse with SparseML, as SparseML can be used to optimise models for any inferencing runtime, and DeepSparse can be used to run any model as long as it is in the correct format, typically the ONNX (Open Neural Network Exchange) format. This flexibility is very useful, as it allows for the use of the most appropriate inferencing runtime for the project.

2.3 Training Methods

When training a neural network, it is important to have training data. For a computer vision system, this means having a dataset of images that are representative of the conditions that the system will be used in and are well-labelled with bounding boxes and class labels. It is important to have a large dataset to ensure that the model is robust and generalises well, which is time-consuming to create.

To solve this problem, a paper by Yang et al. [17] proposes various training techniques, the most promising of which is semi-supervised learning. In this approach, the model is trained on a small labelled dataset, and then used to label a large unlabelled dataset, which is reviewed and corrected by a human.

This process is repeated until the model is sufficiently accurate, and then the model is trained on the large labelled dataset. This approach is very promising, as it allows for the training of a robust model with a large dataset, without the time-consuming process of manually labelling the entire dataset.

2.4 Mechanical Design

As the system is a mechatronic system, the mechanical design is a key component of the project. The mechanical design of the system must be able to transport components to the computer vision system,

and then sort the components into designated bins. The mechanical design must be robust, reliable, and efficient and able to handle a range of different components. The mechanical design must also be able to interface with the other systems in the project, and so must be designed with this in mind. It is important to review existing sorting machines to inform the design of the project.

2.4.1 Transport Mechanisms

The paper by Dhenge et al. [2] depicts a conveyor belt system that transports nuts and bolts to a computer vision system for identification, which aligns with the goals of the project. The hardware prototype is shown in Figure 8, and a down-facing webcam is used to capture images of the components as they pass by. The webcam uses Principle Component Analysis (PCA) to identify the components, which will be discussed in the next section.

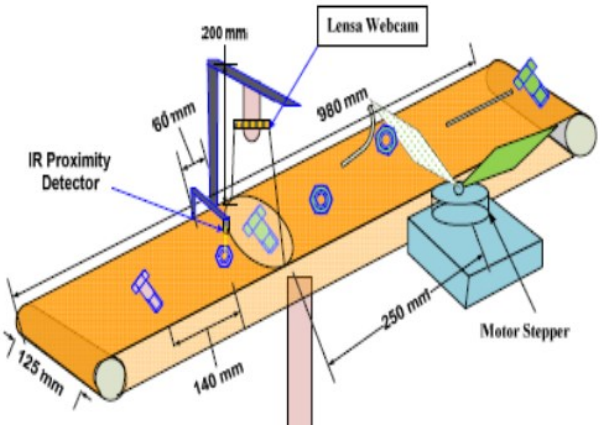


Figure 8: Nut and bolt sorter [2]

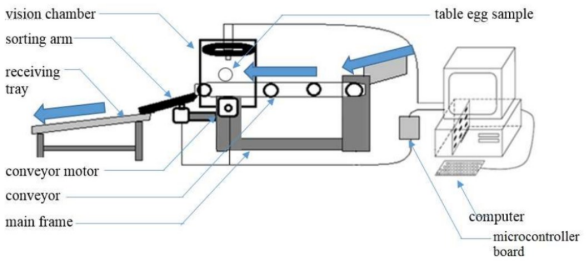


Figure 9: Egg Sorter [18]

The components then separate into two chutes, one for nuts and one for bolts, using a stepper motor, which may be useful for the future design of the sorting system. Additionally, the approach taken by the paper Quilloy et al. [18] to sort Philippine table eggs also features a similar conveyor belt system, a down-facing camera and an arm to sort the eggs into different categories as shown in Figure 9. The approaches taken here are similar to industrial sorting systems seen in videos while researching for this project. This approach seems to answer three major design questions for the project; how to transport the components to the computer vision system; how to transport the components from the computer vision System to the sorting system; and the placement of the camera.

Other approaches include vibratory feeders [19], which use precise vibrations to orientate and transport components, and pneumatic systems from Abe et al. [20] (also suggested by project Supervisor Dr. Stott when dicussing transporting mechanisms in project meetings), use air pressure in tubes to transport components. These approaches require more complex hardware, and lack the easily achievable precision of the conveyor belt system, for example in the case of the pneumatic system, a complex network of tubes and valves and an air compressor is required, which is not practical for the project. Robotic arms are commonly used in the industry for sorting, however, this is not viable as this would either require an expensive robotic arm, or a complex system of motors and actuators to move the components.

From the approaches above, it seems that a conveyor belt with a down-facing camera is the most viable approach to transporting the components.

2.4.2 Bowl Feeders

While researching for this project, and especially in videos [21], it seems most industrial sorting machines use a Vibratory Bowl Feeder (VBF) to help feed components into the sorting system.

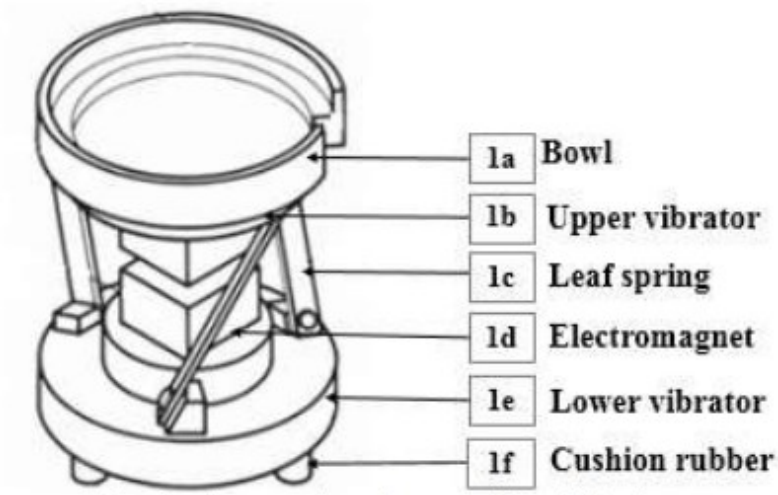


Fig. 1. Automatic vibratory bowl feeder.

Figure 10: VBF [22]

As shown in Figure 10, the VBF consists of a bowl that vibrates coupled with a spring and electromagnet. The paper by Nam et al. [22] explores the optimal design of a VBF for USB keycaps, by attempting to identify the ideal parameters for the structure of the bowl, sorting track, mounting adapter, and suspension system. The paper also uses modal analysis to determine the natural frequencies of the system and uses this to avoid resonant conditions that might cause inefficient or erratic operation.

This paper is useful as it provides a comprehensive overview of the design of a VBF, and provides a good starting point for the design of the VBF. In the future, the project may make use of one for fully autonomous sorting. The paper also provides a good overview of the design considerations for the VBF, and so can be used as a reference during the design process.

Additionally, Reinhart and Loy [23] delve into a mathematical model of a VBF, optimising more on the overall performance of the VBF rather than efficiency, and Silversides et al. [24] provides a good overview of the forces involved in the operation of a VBF, strengthening the basis for its design and viability. The paper by Zhengyang Zhang [25] outlines a sorting system for vials and does not make use of a VBF, instead opting for a turntable design that mechanically orientates the vials. It primarily operates by using a design that is specific to the geometry of the vials, and so does not apply to this project, however, it does provide a good insight into the design of a sorting system.

An alternative to the VBF could be a robotic arm; fine control over movement would be necessary as the components are small and may be tangled, however, as mentioned before would require its own set of sensors and camera systems. As the components can be tangled, there are not many viable alternatives to the VBF or a robotic arm, so between these two solutions, it seems that the VBF is the most viable approach if the system is to be fully autonomous, as the vibrations of the VBF would help to untangle the components.

2.5 Key Takeaways

After reviewing the literature, the following key takeaways were made:

The most viable approach to the problem of component identification is to use a CNN, specifically the YOLOv8 architecture, as they are very effective at classification tasks, which translates to being able to identify electronic components. With the ability to optimise the model using SparseML and DeepSparse, the model may be able to run on the Raspberry Pi 4 in real-time, which is a key requirement of the project.

The most viable approach to the problem of component transportation seems to be a conveyor belt system with a down-facing camera, and so the design of the mechatronics and the computer vision system will be based on the research outlined above.

A VBF is the most viable approach to the feeding mechanism of the sorting system, and so the design of the VBF (if employed in the project) will be based on the research outlined above.

Additionally, the YOLOv8 architecture will be employed, making use of the SparseML and DeepSparse optimisation tools with potential of using semi-supervised learning techniques to train the model.

3 DESIGN

Contents

This chapter will discuss the design and system architecture of the project. The project is divided into three main components: the mechanical design, the integration of electronics and software, and the computer vision system. The mechanical design will be discussed in Subsection 3.3, the computer vision system in Subsection 3.4, and the electronics and software integration in Subsection 3.5.

3.1 System Architecture

The project's design is underpinned by a modular design philosophy; each subsystem is designed to be as independent as possible, allowing for easier debugging and maintenance, and exposes only high level interfaces to other subsystems. This not only simplifies the development process but also allows for easier integration of new features in the future.

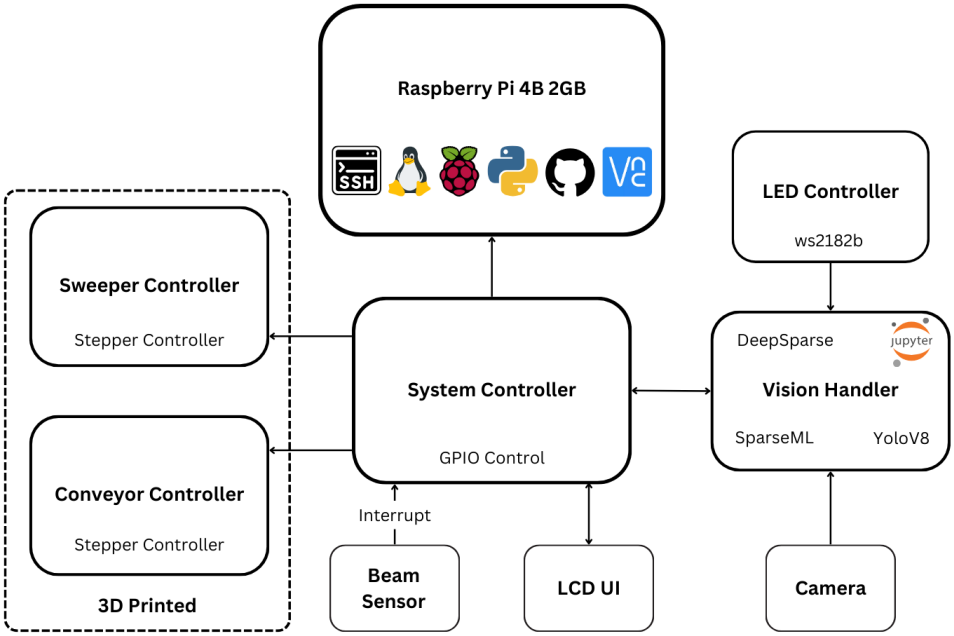


Figure 11: System Architecture Diagram

A System Controller oversees the entire system, interacting with all other components and controlling all communication. It is responsible for interacting with the Vision Handler, and passing commands to the Conveyor Controller and Sweeper Controller. The Vision Handler is responsible for processing images from the camera and performing inference for classification. The Conveyor Controller is responsible for controlling the conveyor belt, while the Sweeper Controller is responsible for controlling the sweeper. Both of these controllers also control a TMC2209 stepper motor driver, which in turn controls the stepper motors that drive the conveyor belt and the sweeper.

There is two-way communication between the LCD UI and System Controller as the user may input commands to the system, and the System Controller may send status updates to the LCD UI.

3.2 Hardware

When selecting hardware for the project, it is vital to consider the requirements of the system and to evaluate the trade-offs between alternatives. This section will detail the choice of hardware and provide justification for the selection.

3.2.1 Raspberry Pi 4 Model B

The Raspberry Pi 4 Model B [10] was chosen as the main component of the system and is the central hub that all other components are connected to. This model was chosen as it is regarded as a reliable SoC (System on Chip) and is widely used in the industry. It has a large amount of software and driver support, ensuring confidence in finding solutions to problems encountered over the course of the project. Additionally, it has GPIO pins that can be used to control the other components in the system, such as stepper motors and LEDs. It also has a dedicated CSI port which allows for a camera to be connected directly to the SoC, which is necessary for the computer vision system.

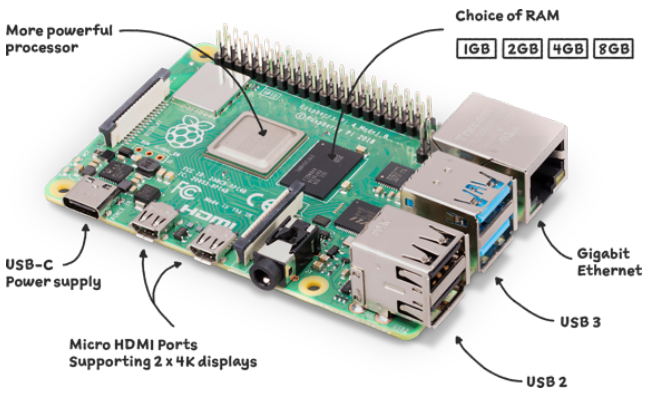


Figure 12: Raspberry Pi 4 Model B 2GB [10]

It also features Wi-Fi support, allowing for SSH and remote development software, as well as an HDMI port for the display. Originally, the 4 GB model was to be used however an order issue resulted in receiving the 2 GB model. This was found to be not an issue as the 2 GB model still performed well, but there was now a heavier need for efficient code to ensure the system runs smoothly.

An alternative to the Pi family of SoCs could be an Arduino-compatible microcontroller, but while they are capable of controlling the components of the system, and potentially drive an LCD, they lack the resources to run the Computer Vision system and stream video from the camera. Due to this, they cannot be used as a replacement for the Pi. It was considered to use an Arduino-compatible microcontroller to delegate control of certain components to overload the Pi, but this was found to be unnecessary as the Pi was able to handle the load of the system.

A viable alternative to the Pi could be a Nvidia Jetson Nano [26], as it is designed for Computer Vision applications and has a dedicated GPU. The dedicated GPU would reduce the likelihood of the Computer Vision system being a bottleneck, and it has a CSI port for the camera. In terms of CPU performance, the Nano is weaker than the Pi, having a quad-core ARM Cortex-A57, whereas the Pi has a quad-core ARM Cortex-A72 [10]. However, the worse CPU performance is offset by the dedicated GPU, and the Nano has 4 GB of RAM, making the Nano a very attractive alternative to the Pi. The Nano's drawback is that it is significantly more expensive than the Pi (up to 3x from retailers), and given the ability to optimise the computer vision system as discussed in Section 2 (Background), the Pi is still a viable choice for the computer vision system.

To address the lack of a dedicated GPU for inference on the Pi, an article from Medium [27] benchmarked the Pi's performance on various computer vision tasks with various accelerators like the Intel Neural Compute Stick 2 and the Google Coral USB Accelerator against other SoCs like the Jetson Nano and the Coral Dev Board. It was found that the Coral Dev Board was the best performing, however the Pi using optimisation libraries like TensorFlow Lite showed promising results, which helps to strengthen the argument for using the Pi for the computer vision system, especially with the SparseML and DeepSparse libraries that were discussed in Section 2 (Background).

The benchmarks can be found in Appendix Figure 84.

3.2.2 DFRobot 7″ Touchscreen Display

The DFRobot 7″ Touchscreen Display was chosen for the LCD panel as it is a relatively cost-effective display that is compatible with the Raspberry Pi. It has touchscreen support and features Raspberry Pi 4 mounting holes on its back, as well as HDMI adapter boards to connect to the Pi. This means that a physical mount for the Pi does not need to be designed, and only a mount for the display is required. The display is also powered by the Pi, so no additional power supply is required.

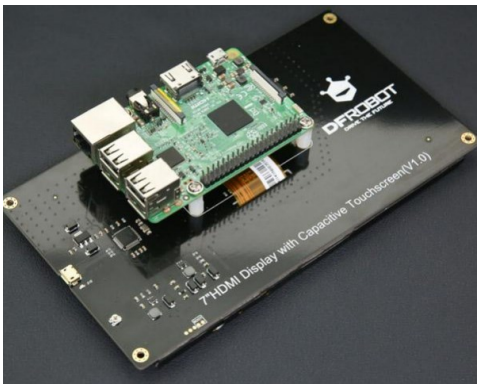


Figure 13: DFRobot 7″ Touchscreen Display [28]

Other alternatives to the DFRobot 7″ display were various 10″ displays however many required proprietary drivers and cables, and were more expensive. The inclusion of mounting holes on the back of the DFRobot display was also a major factor in its selection, which seems to be uncommon in other displays.

3.2.3 Okdo Adjustable Focus OV5647 Camera

For the Computer Vision system, a camera is required to capture images of the components. The Okdo Adjustable Focus OV5647 Camera was chosen as it is CSI compatible, meaning it can be connected directly to the Raspberry Pi, and has a manual focus ring, allowing it to be used as a macro camera to capture images of small components placed directly above it. The manual focus ring allows the focus of the camera to be specifically tuned to the design of the system, allowing for the best possible image quality. It also has a 5MP sensor [29], which is sufficient for the Computer Vision system, as high-resolution images would be preprocessed and reduced, only adding to the amount of processing required.

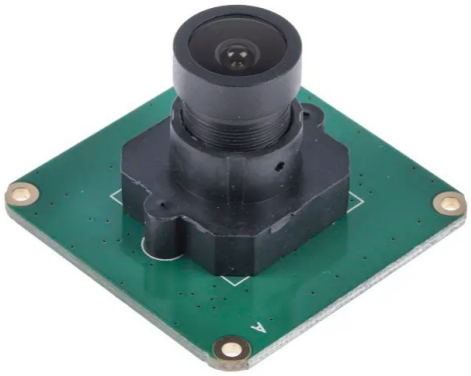


Figure 14: Okdo Adjustable Focus OV5647 Camera [30]

Other cameras were considered, such as the Raspberry Pi Camera Module V2 [31], however, it lacks a manual focus ring, which may result in a Minimum Object Distance (the closest distance at which an object can be captured) that is too far for the design of the system. Another camera, the Raspberry Pi High Quality Camera [32] also has the option of replacing the lens with a dedicated macro lens, however these are incredibly expensive.

3.2.4 WS2812B LEDs

To illuminate the components on the conveyor belt, a solution was required that could provide bright, controllable light. The WS2812B LEDs [33] were chosen as they are individually addressable, meaning

that each LED can be controlled independently, allowing for a wide range of colours and patterns to be displayed. They are also relatively cheap and easy to source, and are compatible with the Raspberry Pi and have a large amount of support due to their popularity. They also can run off 5V, meaning it can share the input power rail with the Raspberry Pi and are also easy to control, requiring only a single GPIO pin to control many LEDs as they can work with either SPI or PWM signals, depending on the chosen library.

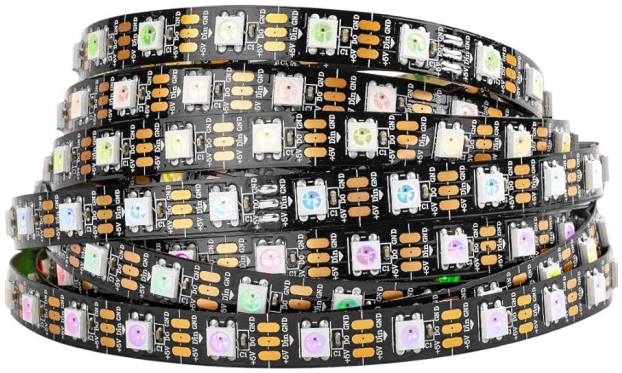


Figure 15: WS2812B LEDs [33]

Figure 16: 12V COB Ring Light

Originally, a 12V COB ring light was considered, as it could produce a uniform light source, however the specific ring was not dimmable and resulted in strange flickering when attempting to do so with a MOSFET. It also did not produce a clean white light, but a more blue light. The WS2812B LEDs were chosen as they could produce any light colour, and could be cut and placed in any configuration, allowing for a more customisable design.

3.2.5 Stepper Motors

Stepper motors possess the ability to move in precise increments, making them ideal for the Sweeper System which requires precise control. The Conveyor System is not as demanding, but stepper motors were chosen for consistency across the system. The NEMA17 series of stepper motors were chosen [34] as they are widely used and have a large amount of support and documentation, also typically used in FDM printers (fused deposition modelling, or more commonly, 3D printers). With the ability to rotate at 1200RPM/900RPM at 12V, and a holding torque 48/63Ncm (for single/double stack motors), they are more than capable of driving the conveyor belt and sweeper arm.

Figure 17: NEMA17 Stepper Motor [34]

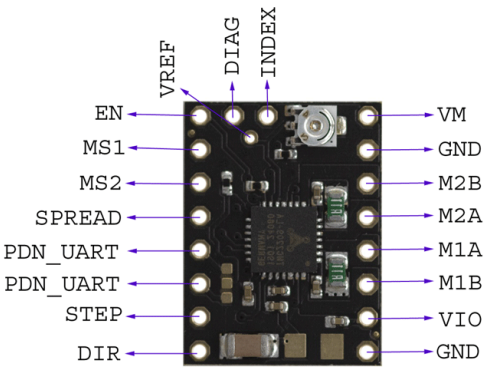


Figure 18: TMC2209 Stepper Motor Driver [35]

For driving the motors, TMC2209s were selected due to their silent operation, and microstepping features, although not required for the system, a method of allowing the motors to achieve higher precision

and more silent operation. Due to their popularity in FDM printing and the ease of use, they have a large amount of support and documentation. They are also capable of driving the NEMA17 series of stepper motors, and are compatible with the Raspberry Pi. Originally, the DRV8825 was considered [36], however they added significant noise to the system and were unpleasant to use.

3.2.6 Break Beam Sensor

Due to the complexity of the system, it is vital to ensure that the system is as efficient as possible to ensure real-time performance. The design choice was made to use an IR Beam Break Sensor [37] to detect the presence of objects on the conveyor belt to reduce the computational overhead of constant inference on the Vision Handler. The System Controller connects directly to the IR Beam Break Sensor, which is used to detect the presence of objects on the conveyor belt, making use of interrupts to detect changes in the sensor's state. This allows for a fast response from the system without much computational overhead. The sensor itself does not change lighting conditions as it operates with infrared light, which is not visible to the human eye. The sensor is also easy to install and maintain, as it only requires a power supply and a digital input pin to operate.

Figure 19: IR Beam Break Sensor [37]

An alternative to the IR Beam Break Sensor would be to use software approaches on the Vision Handler to detect the presence of objects on the conveyor belt, for example using OpenCV [38] to detect large changes between successive frames. This could be implemented in a number of ways such as background subtraction or histogram comparison. However, this would also require the Vision Handler to constantly process frames, which would increase the computational overhead of the system. The IR Beam Break Sensor is a more efficient solution as it allows the system to only process images when an object is detected on the conveyor belt.

An approach to dynamically adjust the frame rate of the camera could be considered, increasing frame rates when there are no objects on the conveyor belt in anticipation of objects being placed on the conveyor belt, and reducing frame rates when objects are detected to reduce computational overhead. This could potentially impose resource contention between the Vision Handler and the System Controller while it is controlling the Sweeper and Conveyor Belt. The framerate also needs to take into consideration the conveyor speed to not miss objects, requiring both more complexity and computational resources.

An alternative to the IR Beam Break Sensor would be to use a proximity sensor to detect the presence of objects on the conveyor belt, however this requires constant polling which would be less efficient than using an interrupt-based approach. Additionally, the IR Beam Break Sensor is a more reliable solution as it is less prone to false positives than a software-based approach. The IR Beam Break Sensor is a

more robust solution as it is less prone to interference from external factors such as lighting conditions or shadows.

For these reasons, the IR Beam Break Sensor was chosen as the best solution for detecting the presence of objects on the conveyor belt.

3.2.7 Power Supply Unit

To power the system, two parameters were taken into account when choosing a PSU; the power rating and voltage.

I(mA)	Power (W)	Current Voltage (V)
118	15.2	241.6

Figure 20: Power consumption of the system

Figure 21: 24V DC 6.25A Power Supply Unit [39]

Firstly, the power supply must be able to drive all components in the system with overhead in case of spikes in power consumption. The system consists of a Raspberry Pi 4 Model B, a DFRobot 7" Touchscreen Display, two NEMA17 stepper motors and TMC2209 stepper motor drivers, an IR Beam Break Sensor, an Okdo Adjustable Focus OV5647 Camera, and 16 WS2812B LEDs.

In the early stages of the project, with only the Raspberry Pi, the DFRobot 7" Touchscreen Display, WS2812B and Okdo Camera, the power consumption was measured to be under 20W, recorded using a smart plug with a power meter as shown in Figure 20. It was thought that the addition of the NEMA17 motors and TMC2209 stepper motors would contribute 50W (running at 2A 12V) to the power consumption, making for a power consumption of under 70W. A 30W overhead would have been added to the power supply, resulting in a 100W power supply. However, a 150W power supply was used instead due to an overestimation on the power consumption of the system before finalising the design. In hindsight, the 150W power supply was excessive, and a 100W power supply would have been sufficient.

Additionally, the voltage of the power supply must be greater than or equal to the highest voltage of any component in the system to enable the use of step down convertors to power the components. The highest voltage of any component in the system is 12V and the lowest is 5V. Initially, 24V components were a possibility due to the variety of LEDs and stepper motors, so a 24V power supply was chosen.

This resulted in selecting a 24V DC 6.25A Power Supply Unit.

3.2.8 Step Down Convertors

To power the components of the system, step-down convertors were used to convert the 24V output of the PSU to the required voltage of the components. The components that required a 12V supply were the NEMA17 stepper motors and TMC2209 stepper motor drivers, and the components that required a 5V supply were the Raspberry Pi, DFRobot 7" Touchscreen Display, Okdo Adjustable Focus OV5647 Camera, and WS2812B LEDs.

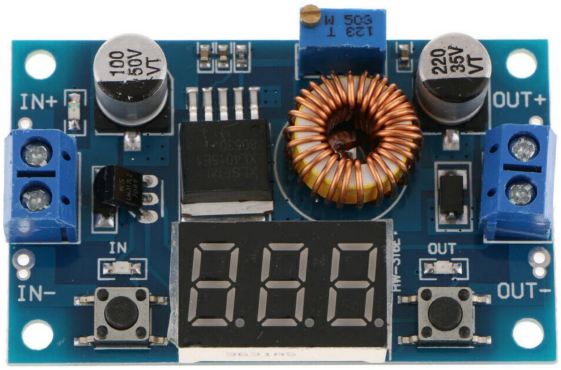


Figure 22: XL4015 High Power Step Down Convertor [40]

The DFRobot and Camera is powered directly from the Pi, however it is necessary that the WS2812B LEDs are powered from the PSU to prevent the Pi from being overloaded. Each LED can consume up to 60mA at full brightness, and with 16 LEDs, the total current draw could be up to 960mA, which may be too much for the Pi to handle. Instead, the LEDs and Pi are connected to a USB hub that is powered by a step-down convertor at 5V, which is connected to the PSU. Likewise, the NEMA17 motor and TMC2209 stepper motor driver are powered by a step-down convertor at 12V.

The XL4015 High Power Step Down Convertor was chosen as it is capable of handling the current draw of the components, and is adjustable, allowing for fine-tuning of the output voltage. It is able to draw up to 75W, which is more than enough for the two 5V and 12V systems.

3.2.9 2020 Aluminium Extrusion

To provide a sturdy frame for the system, 2020 aluminium extrusion was chosen as it is lightweight, strong, and cost-effective. It is also easy to work with, as it can be easily cut to size and assembled using standard M3 screws and T-nuts and FDM printed parts. The 2020 aluminium extrusion is used to provide a frame for the conveyor system.

Figure 23: 2020 aluminium extrusion [41]

Other options like 2040 or 2060 extrusion were considered, however, they were found to be too large and heavy for the system, and would have added unnecessary weight and cost. 2020 extrusion was found to be the best compromise between strength, weight, and cost. Cylinder extrusion like rods would not be suitable as they would not provide a flat surface for the components to be mounted on, and would not provide a frame for the system.

3.2.10 GT2 Timing Belt

To drive the sweeper arm, a timing belt is needed to transfer the rotational motion of the stepper motor to the linear motion of the sweeper arm. The GT2 Timing Belt was chosen as it is widely used in 3D printers and is compatible with the NEMA17 stepper motor. It is also easy to source and is cost-effective. The belt is also easy to cut to size and is easy to attach to the stepper motor and the sweeper arm.

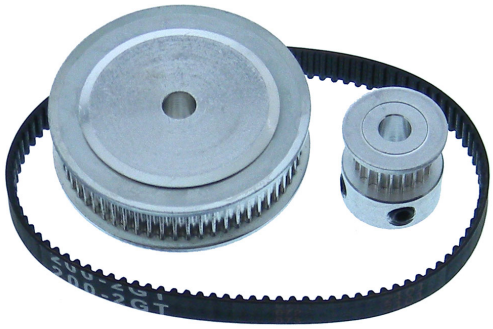


Figure 24: GT2 timing belt and pulley [42]

The reduction gear shown in Figure 24 (used to increase torque at the cost of speed) was not used as the NEMA17 stepper motor was found to be powerful enough to drive the sweeper arm without the need for a reduction gear.

3.3 Mechanical Design and Electronics

As the system has a major physical component, the mechanical design is a crucial aspect of the project. The mechanical design is responsible for the physical structure of the system, including the conveyor system, the sweeper, bins, the housing for the LCD and PSU (Power Supply Unit), and the camera mount. It provides the foundation for the system to operate and interact with the environment, and as such it is important to ensure that the mechanical design is robust and reliable.

The system also has a major electronics component due to the requirement of moving parts, namely the conveyor and sweeper systems. Care must be taken to ensure that the power delivery to these systems is stable and reliable, and connections are secure. Careful consideration must also be taken to ensure electrical safety, preventing damage to the system and harm to the user.

The system has been designed in FreeCAD [43], a free and open-source parametric 3D CAD software. Aligning with the modular design approach of the project, the parametric nature of FreeCAD allows for easy modification of the design by modifying numerical parameters, which is useful in designs that require frequent changes. It was decided to use FDM printing to produce the components of the system using PLA, a biodegradable thermoplastic derived from renewable resources such as corn starch or sugarcane [44]. PLA is easy to print, and does not require ventilation or post-processing, unlike other materials like ABS. The known brittleness and low heat resistance of PLA are not a concern for this project, as the system is not exposed to high temperatures or direct mechanical stress. An alternative to PLA would be PETG, which has higher heat resistance and is less brittle than PLA, however, it is more difficult to print and is more prone to warping - it is also less environmentally friendly than PLA and less cost-effective.

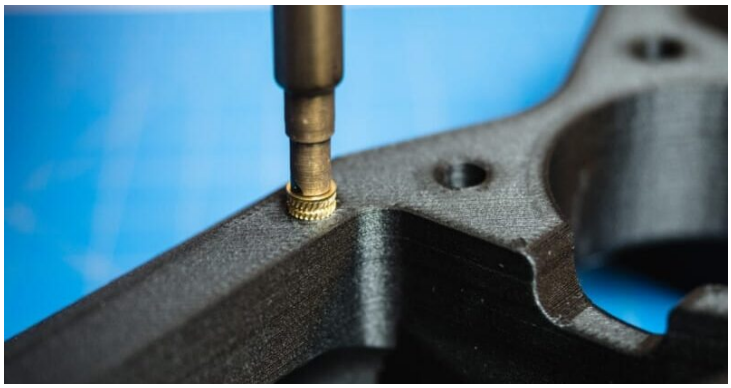


Figure 25: Heat Set Inserts [45]

All components are designed to be easily assembled and disassembled, allowing for easy maintenance and repair. They are easily assembled using standard M3 screws and heat-set brass inserts, which are easy to source and replace. The inserts are melted into the plastic using a soldering iron, providing a strong and reliable connection. The knurls on the inserts provide a strong grip on the plastic, preventing them from rotating when screws are inserted. The use of heat-set inserts also prevents the plastic from being stripped when screws are inserted, which can happen with repeated use of screws in the same hole.

3.3.1 Conveyor

As discussed in the Background (Section 2), a conveyor belt system with a face down camera seems to be the most promising approach. For the design of the conveyor system, it is imperative that the following requirements are met:

- The conveyor belt must be able to move objects from the input to the output.
- The conveyor belt must be able to move objects at a consistent speed.
- The conveyor belt must be able to move objects in a controlled manner.
- The belt must be detachable for maintenance and repair.
- The conveyor must include a tensioning mechanism to ensure the belt is taut.
- The conveyor must include a mount for the break beam sensor.
- The design of the conveyor must mount the stepper motor that drives it.
- The idler that allows the belt to move must be mounted and is free to rotate to reduce friction and ensure smooth operation.
- The actively driven roller must be easily coupled and decoupled from the stepper motor.

The belt will be made of thick paper as it is cost-effective, easy to replace, and is reasonably rough and can withstand tension forces. It can also be in different colours which may help with contrast in the images for the Vision Handler. Other alternatives to paper would be rubber or silicone, however, these are more expensive and difficult to source and cut to size. Alternatively an FDM printed belt could be used, but this would require more maintenance and may lead to louder operation.

The foundation of the conveyor system will be 2020 aluminium extrusion, which is lightweight, strong, and cost-effective. Custom-designed FDM printed PLA parts will be used to fulfil the requirements of the conveyor system. To facilitate smooth rotation on the idlers and driven roller, ball bearings will be used. A NEMA17 stepper motor will be used to drive the driven roller, as it is powerful enough to drive the belt and is easy to source; torque is not a concern as the belt is lightweight and the speed of the NEMA17 series of stepper motors is sufficient to not require a reduction gear. The stepper motor will be controlled by a TMC2209 stepper motor driver, a silent, and efficient stepper motor driver that is easy to use and is compatible with the NEMA17 stepper motor.

NEMA17 stepper motors typically require 12V, meaning a separate means of delivering power to the motors from the Pi is required.

3.3.2 Sweeper

To sort components travelling along the belt, many approaches were considered.

One such approach would be compressed air to blow components off the conveyor belt into bins as seen in commercial systems. However, this approach is not feasible as it would require a large and potentially loud air compressor to generate the compressed air, as well as a system of nozzles and valves. Alternatively a single nozzle and valve with a moving arm could be used, however this poses the same issues.

Another approach could be to use a series of actuated arms to push components off the conveyor belt into bins. The arms would be designed in such a way where they have two states; either open or closed, making them operate more like a gate. This approach would require a series of actuators and a complex control system to ensure that these gates are in the correct position at the correct time. Each gate would need a motor which would quickly become expensive and difficult to route with the limited GPIO pins on the Raspberry Pi.

An improvement to this design would be to instead carefully design the gates such that the end position of their rotation is either in the open or closed state. An arm on a moving track with a flexible rod could then be used to rotate the arms as it travelled past, effectively allowing the system to carefully control which bin the component goes into. This would require a single motor and a carefully designed arm, which would be more cost-effective and easier to control. This is an approach that is discussed in Section 4, but was not chosen for the final design due to the next approach.

The approach that was taken was to simply attach a static arm to a moving carriage that ran alongside the conveyor belt. The arm is angled, allowing the movement of the conveyor belt to push components off the belt and into the bins. This approach does not require intricately designed gates or complex control systems, and is simple and cost-effective. It requires the use of a NEMA17 stepper motor and an endstop for precise control of the arm's position, and a TMC2209 stepper motor driver to control the stepper motor.

Again, the issue of power delivery arises, as the NEMA17 stepper motor requires 12V.

3.4 Computer Vision System

To identify and classify components on the conveyor belt, a Computer Vision system is required. As discussed in the Background (Section 2), the approach taken will be to use the YOLOv8 object detection model, due to its notable inference speed, performance, and ability to optimise to deploy on the Raspberry Pi.

3.4.1 Image Processing

YOLOv8's training framework [46] allows image augmentation to be applied to the dataset in-flight, which can be used to artificially increase the size of the dataset and improve the model's performance. This can be used to apply transformations to the images such as rotation, scaling, and flipping, as well as more complex transformations such as perspective warping. This can be used to artificially increase the size of the dataset and improve the model's performance.

It is important to select data augmentation schemes that would result in the images being in the same domain as the images that the model will be inferring on. For example, grayscale images should not be used as the model will be inferring on colour images. Scaling augmentations should be used sparingly

as to not remove the component from the image, while rotation augmentations should be used to ensure that the model is invariant to the orientation of the component.

For the task of component identification, the following augmentations were selected:

Augmentation	Value	Reasoning
HSV_H	0.05	This determines the range of hue values that the image can be augmented by. A 5% change in hue can represent lighting conditions changing, which is important for the model to be invariant to. Changing it too much will result in the training data to not be in the same domain as the inference data.
HSV_S	0.3	This determines the range of saturation values that the image can be augmented by. Justification is the same as above.
HSV_V	0.2	This determines the range of value (degree of whiteness) values that the image can be augmented by. Justification is the same as above.
Degrees	180	This determines the range of rotation values that the image can be augmented by. Allowing any degree of rotation is important as the components may be placed on the conveyor belt at any orientation.
Translate	0.1	This determines the range of translation values that the image can be augmented by. Allowing a 10% translation is important as the components may not be placed in the centre of the image, however too much translation may result in the component being cut off.
Scale	0.8	This determines the minimum scaling factor that the image can be augmented by. This provides robustness to the specific mounting height of the camera.
Shear	10.0	This determines the range of shear values that the image can be augmented by. Shearing is important as the components may not be placed in the centre of the image.
Perspective	0.0	This determines the range of perspective values that the image can be augmented by. Perspective warping is not required as the camera is always placed directly above the conveyor belt.
Flipud	0.5	This determines the probability that the image will be flipped up-down. Flipping the image up-down is important as the components may be placed on the conveyor belt in any orientation.
Fliplr	0.5	This determines the probability that the image will be flipped left-right. Justification is the same as above.
Mosaic	0.5	This determines the probability that the image will be augmented by a mosaic. According to YOLO [46], this greatly improves model performance.

Table 1: Data Augmentation Parameters

3.4.2 Real-time Object Detection

To perform real-time object detection, it is necessary to deploy a model with low inference latency, especially for the Raspberry Pi where computational resources are scarce. As discussed in the Background (Section 2), the YOLOv8 object detection model was initially chosen due to the ability to be optimised by optimisation libraries DeepSparse [16] and SparseML [14] to run on the Raspberry Pi. Pretrained 'sparsified' models are available for the YOLOv8 model, which have been pruned and quantised to reduce the number of parameters and the amount of computation required to run the model, preventing the need to 'sparsify' the model manually. In the event that the model could not be successfully deployed using these libraries, the model could have been run normally at a reduced frame rate, or could have been hosted on the cloud and accessed via an API — though this may introduce latency and would require an internet connection. However, during more research, the standard YOLOv8 model was not found to be suitable for the task.

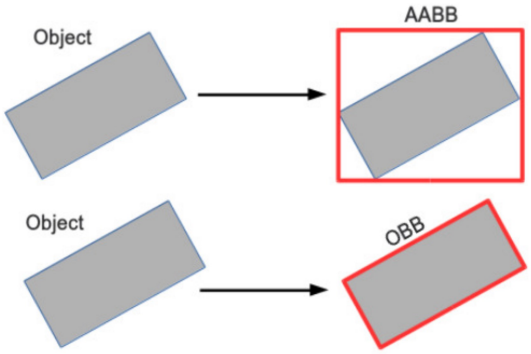


Figure 26: OBB vs AABB Bounding Boxes [47]

Common Computer Vision architectures typically draw AABBs (Axis Aligned Bounding Boxes) when detecting the object in the camera frame, as shown in Figure 26. However, for the specific task of component identification it is more useful to draw OBBs (Oriented Bounding Boxes) which aligns with the orientation of the component, allowing the component to be cropped out and further inspected, for example in the case of value identification. There is a variant of the YOLO architecture that supports OBBs, called YOLOv8-obb [46], which will be used for this task.

Model	mAP^{50}	Latency (CPU ONNX) (ms)	Params (M)	FLOPs (B)
YOLOv8n-obb	78.0	204.77	3.1	23.3
YOLOv8s-obb	79.5	424.88	11.4	76.3
YOLOv8m-obb	80.5	763.48	26.4	208.6
YOLOv8l-obb	80.7	1278.42	44.5	433.8
YOLOv8x-obb	81.36	1759.10	69.5	676.7

Table 2: YOLOv8-obb Model Performance [46]

The OBB models come pretrained on the DOTOv1 dataset [48], which contains 15 classes of objects, including components. Pretrained models are useful as they have already learnt useful features from a large dataset, and can be fine-tuned on a smaller dataset to improve performance [49]. The performance of the different sized YOLOv8-obb models is shown in Table 2. The most crucial parameters to take into consideration is the mAP (mean average precision, explained in Subsection 2.2) and the inference latency.

From Table 2, the YOLOv8x-obb model has the best performance, with a mAP^{50} of 81.36 and a latency of 1759.10ms. The YOLOv8n-obb model has the worst performance, with a mAP^{50} of 78.0 and a latency of 204.77ms. However, the Pi is likely significantly slower than the CPU used in the benchmarks, so the actual inference latency will be higher, making the 0.5 FPS of the YOLOv8x-obb model infeasible, but the 5 FPS of the YOLOv8n-obb model more promising, a 9x improvement in speed for only a 3.1% decrease in mAP^{50}.

As mentioned in Subsubsection 3.2.6, the system will use a break beam sensor to detect the presence of objects on the conveyor belt, reducing the potential of the Vision Handler to be a bottleneck due to constant inference. With the promising performance increase of using the DeepSparse [16] and SparseML [14] libraries to optimise the model for the Raspberry Pi, the YOLOv8n-obb model will be used for the project.

3.5 Software

The software of the system is responsible for controlling the hardware components, processing images from the camera, and interfacing with the user. The software is divided into several subsystems,

each responsible for a different aspect of the system. The software is designed to be modular, with each subsystem exposing a high-level interface to other subsystems, allowing for easy integration and maintenance.

Each component has the capability to run independently to facilitate debugging and maintenance, and communicates with the System Controller to receive commands. The System Controller is responsible for managing the communication between the components, and is the central hub of the system.

The software is written in Python [50], due to its ease of use and large amount of support and libraries available, and will be run entirely on the Raspberry Pi. Git [51] is used for version control to ensure that changes to the software can be tracked and reverted if necessary, Pylint [52] is used for code linting to ensure that the code is clean and readable, RealVNC [53] is used for remote access to the Raspberry Pi, and Visual Studio Code [54] is used as the IDE for development. As mentioned in Subsection 3.3, FreeCAD [43] has been used for the mechanical design of the system.

As outlined in Section 1, the system will need to employ multiprocessing or threading to ensure that the system can run concurrently without issues. The software will be designed to be thread-safe to ensure it can run in real-time with no obvious latency to the user. The software will also be designed to be fault-tolerant, with error handling and logging to ensure that the system can recover from errors and continue to operate.

3.5.1 LCD UI

The LCD UI is responsible for displaying the status of the system, such as the current state of the system, the status of the conveyor belt and sweeper, and the classification of components. It also allows the user to interact with the system, such as starting and stopping the system, and manually controlling the conveyor belt and sweeper. The LCD UI communicates with the System Controller to receive status updates and send commands.

The LCD UI is written with Pygame [55] and Pygame GUI [56], a Python library for creating graphical user interfaces. Pygame GUI builds upon the Pygame library, providing UI elements so that they do not need to be created from scratch. Pygame also provides camera support, allowing the Vision Handler to receive frames for inference or displaying the feed on the LCD UI. It also allows for precise control over what is being drawn and the event loop, allowing for a responsive UI.

3.5.2 Data Annotation Tool

To train the YOLOv8-obb model, a dataset of images of components is required. The dataset must be annotated with the bounding boxes of the components in the images, which can be done using a tool such as Roboflow [57]. However, the components will only be visible from the Raspberry Pi, so it can become tedious to take images from the Pi, transfer them to a computer, annotate them, and repeat for every distinct value within the component and then for every type of component.

Instead, the decision was made to design a custom dataset labelling tool that could directly capture images from the Raspberry Pi, annotate them, and save them locally for training. The dataset would be correctly structured for training with YOLO, facilitating the training process. The tool is responsible for capturing images from the camera, displaying them to the user, allowing the user to draw bounding boxes around the components, and saving the images and annotations to disk.

This tool was written using CustomTkinter [58], a modern fork of the Tkinter library that provides a more modern look and feel, and is more customisable. CustomTkinter provides a high-level interface for creating GUIs, allowing for easy creation of UI elements and event handling. Draw loops are not needed as the UI is event-driven, allowing for easy development.

4 IMPLEMENTATION

Contents

This chapter discusses the implementation of the design that was discussed in Section 3, and the problems and changes that were encountered during the implementation process.

4.1 Mechanical Design

As discussed in Section 3, the system is designed in FreeCAD [43], a parametric 3D modelling software. The below figures show the system in its entirety, with the front and back views of the system.

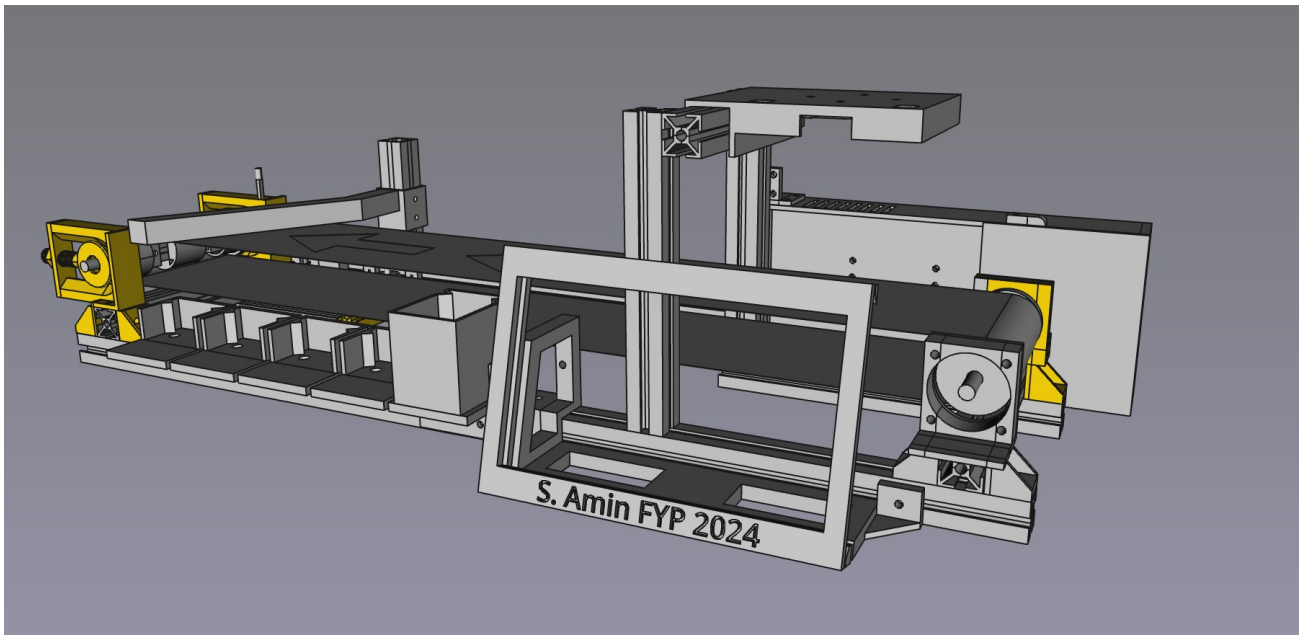


Figure 27: Front View of the Mechanical Design

The conveyor belt with the direction of travel is indicated with arrows. The camera is mounted on the top of the system, with LED lights inside the camera mount. The LCD mount for the DFRobot 7" LCD is clearly visible at the front, with a NEMA17 stepper motor mount positioned next to it for the driven conveyor rollers. The sorting bins are clearly present in the front of the system for easy user access.

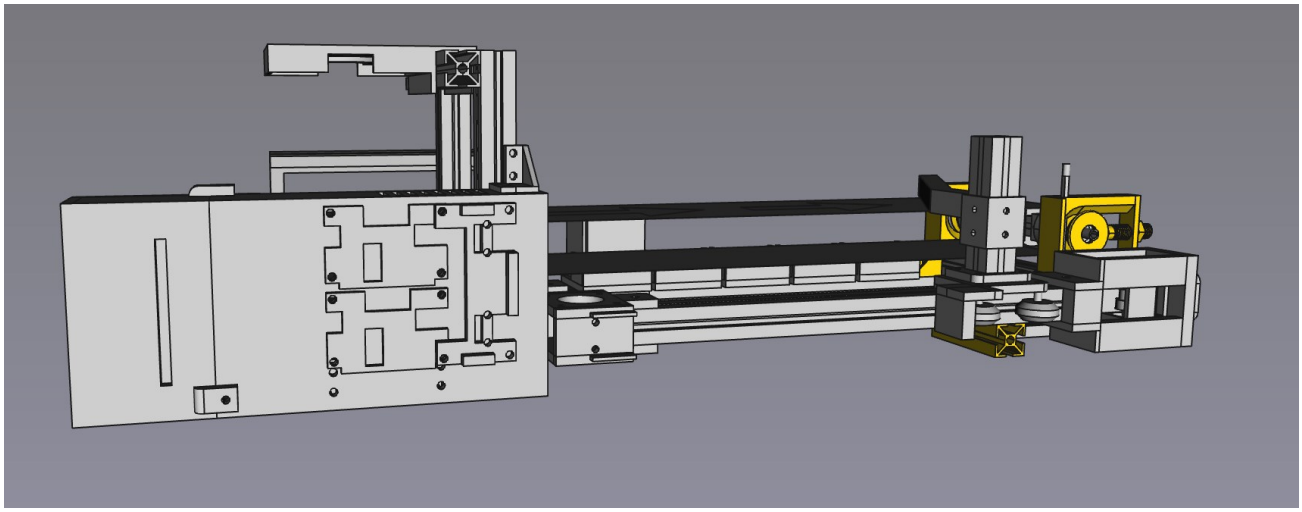


Figure 28: Back View of the Mechanical Design

From the back view of the system, the PSU enclosure is clearly visible with mounting plates for the step-down converters. A belt-driven arm is visible on the back, making use of another NEMA17 motor to drive it and a belt tensioner to ensure the belt is taut. The belt-driven arm is used to push components off the conveyor belt into the sorting bins. The tensioners for the conveyor belt itself are also visible in the back view on the right side.

4.1.1 FDM Printer Settings

As the system is using FDM printed parts, it is necessary to ensure that the parts are printed with the correct settings to give them the desired strength and durability. The parts are printed with a heavily modified Voxelab Aquila C2 FDM printer, with the following settings:

Setting	Value	Justification
Layer Height	Dynamic (0.16-0.48 mm)	A dynamic layer height to balance quality and printing speed. The slicer will automatically adjust the layer height based on the required resolution of the current layer.
Wall Width	1.8 mm	A 0.6 mm nozzle was used to increase printer speed, so the wall width is a multiple of the nozzle diameter. Three walls were chosen as a rule of thumb.
Infill Density	16%	The percentage of the part that is filled with extruded plastic. Higher infills are unnecessary as the parts are not load-bearing, and the air gaps for lower infills allow some flexibility in the parts.
Supports	0% Infill Tree Supports	Supports are required for 3D printed parts that have overhangs. Tree supports are used as they are easier to remove and use less material. The supports work well without infill due to the larger nozzle size.
Base Printing Speed	72 mm/s	The speed at which the printer extrudes plastic. This was calculated from determining the printer's flow rate (maximum extrusion per second), the extrusion width and layer height. The speed was then set to 80% of the maximum speed to ensure quality. The flow rate was determined to be $26mm^3/s$, the layer height at its largest was 0.48mm, and the extrusion width was 0.6mm, from the above settings. Using the formula: $FlowRate = LayerHeight \times ExtrusionWidth \times Speed$ the maximum speed was determined to be 90mm/s, reduced to 80% for quality, reaching 72mm/s.

Table 3: FDM Printer Settings

4.1.2 Power Supply Unit Enclosure

Due to the potential of high-current carrying wires, the PSU has been enclosed in an FDM printed case, which prevents accidental contact with the PSU's terminals.

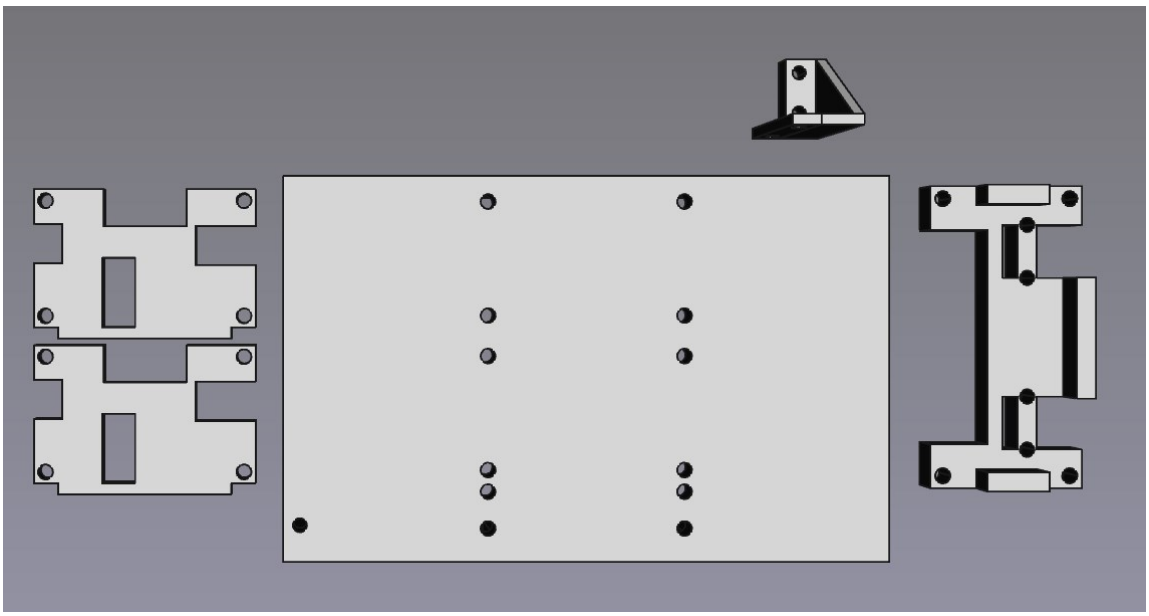


Figure 29: PSU Enclosure

The PSU enclosure contains several components;

PSU Container: The PSU container is designed to house the 24V 6.25A power supply unit, and provides mounting holes for the other components. It attaches to the PSU using M3 screws.

Step-Down Converter Mounts: The step-down converters are mounted on the PSU container using M3 screws. The step-down converters are used to convert the 24V output from the PSU to 12V and 5V for the system, shown by the two mounts at the top of the PSU container.

USB Breakout Mount: For the 12V and 5V power outputs, a USB breakout board is used to provide power to the components, allowing standard cables to be used for power distribution. The USB breakout board is mounted on the right side of the PSU container, connecting to the step-down converters.

PSU Switch Housing: For safety reasons, the connections between the switch and the PSU are enclosed in a housing to prevent accidental contact with the high voltage connections. There is a slot in the Switch Housing for the power cable to pass through to the step-down converters, and a switch snap-fit mount to hold the switch in place on the side.

PSU Mount: The PSU mount is used to mount the PSU container to the 2020 extrusion frame of the system. The mount is screwed into the top of the PSU container and the 2020 extrusion frame, using M3 screws and T-nuts.

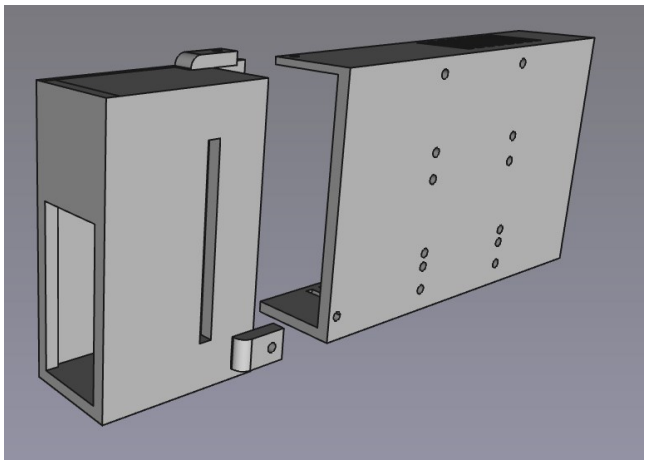


Figure 30: PSU Switch Housing

Figure 31: IEC C14 Power Socket [59]

A standard IEC C14 power connector and an RS Pro Snap-In Fused Rocker Switch [59] are used to connect the PSU to the mains power. As the power supply is a 6.25A power supply, a 6A fuse was chosen for the switch, which is the closest available fuse rating to the power supply's current rating. This ensures that the fuse will trigger before the power supply is damaged, should there be a short circuit in the system. For wiring the power supply and connecting the step-down converters, 18AWG wire was chosen which is rated for 17A [60], which is sufficient for the system's power consumption. The wire is securely crimped to the power supply using a crimping tool.

Test	Result	Req	Description
Earth Continuity	0.06Ω	≤ 0.1Ω	Verifies the earth wire is connected.
Earth Leakage	0.2mA	≤ 0.75mA	Ensure the touchable metal parts cannot cause harm.
Insulation Resistance	320MΩ	≥ 1MΩ	Ensure wires are sufficiently insulated.

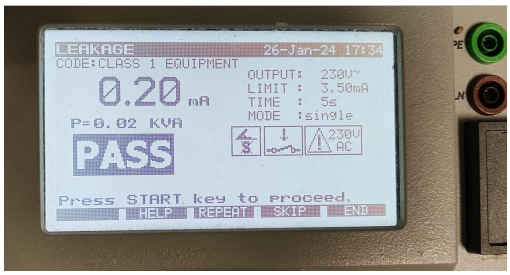


Table 4: PAT Testing Results and PAT Machine

Due to the concern of electrical safety, the power supply has been PAT tested by technicians in the Level 1 Labs and has been deemed safe for use. PAT (Portable Appliance Testing) is a process by which electrical appliances are routinely checked for safety [61, 62]. The results of the PAT test are shown in Table 4.

4.1.3 LCD and Camera Mounts

For the DFRobot 7″ LCD, a mount was designed to hold the LCD in place on the front of the system. The LCD is affixed at a 15° angle to the front of the system, allowing for easy viewing by the user.

The mount is designed to be attached to the 2020 extrusion frame of the system, using M3 screws and T-nuts.

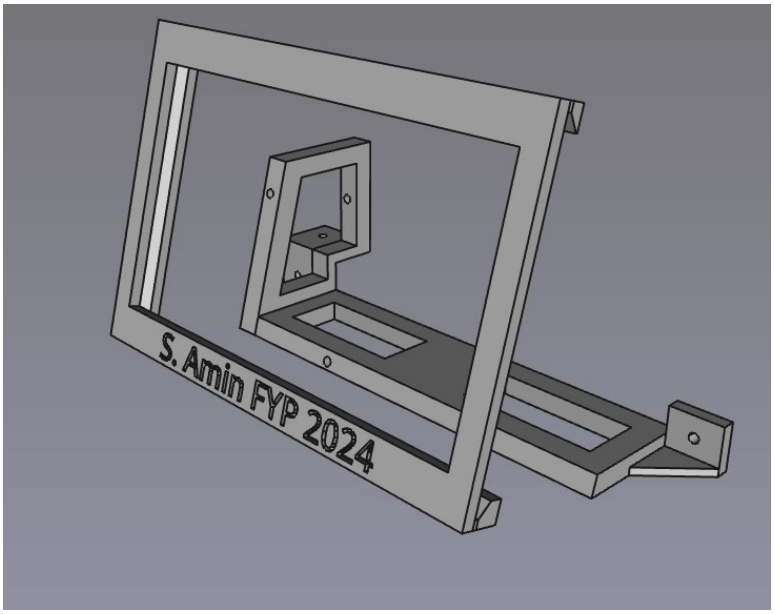


Figure 32: LCD Mount

The design consists of two parts;

LCD Cover: The LCD cover is designed to hold the LCD in place. It contains a small gap that the LCD fits into, preventing it from falling out with the help of friction.

LCD Extrusion Mount: The LCD extrusion mount is designed to be attached to the 2020 extrusion frame of the system. It contains screw holes for the LCD Cover to be attached to, and screw holes for the 2020 extrusion frame. It also contains a support for the LCD cover to rest on, preventing it from breaking off.

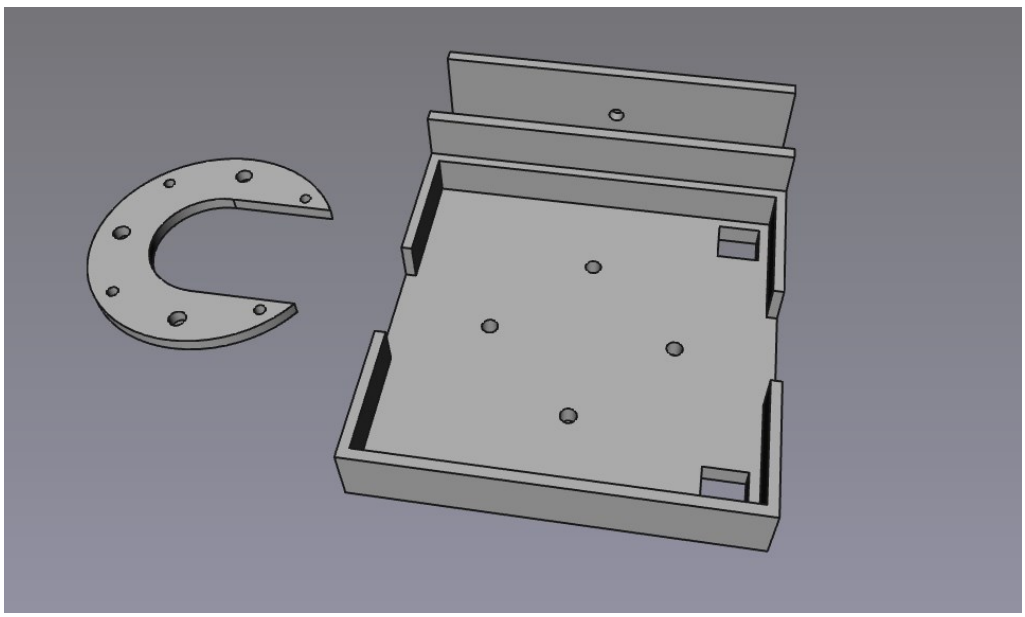


Figure 33: Camera mount and plate

For the Okdo OV5647 Camera, a mount was designed to hold the camera face down above the conveyor belt. The camera is mounted on the top of the system, with LED lights inside the camera mount. The mount consists of two parts as shown in Figure 33;

Extrusion Mount: The extrusion mount is designed to be attached to the 2020 extrusion frame of the system. It contains a hole for the WS2812B LED strip wires to pass through, and screw holes for the camera plate to be attached to. The mount fits onto the extrusion, however this is not strictly necessary as the camera mount is able to fit on the extrusion without falling off.

Camera Plate: The camera plate is designed to hold the camera in place. It contains four holes for the Okdo OV5647 Camera to be attached to, and a slot for the camera ribbon cable to pass through, ensuring that the camera remains flat.

4.1.4 Conveyor Belt

The conveyor belt is a key component of the system, as it is used to transport components from the input side to the bins. As such, it is the largest component of the system, and so makes use of 2020 aluminium extrusion as its skeleton. The conveyor belt has two rollers; one actively driven by a NEMA17 stepper motor, and the other is a free roller. The conveyor belt relies on friction to turn the idle roller, so it is essential that the belt is taut. To ensure this, belt tensioners are used on the idler side of the conveyor belt.

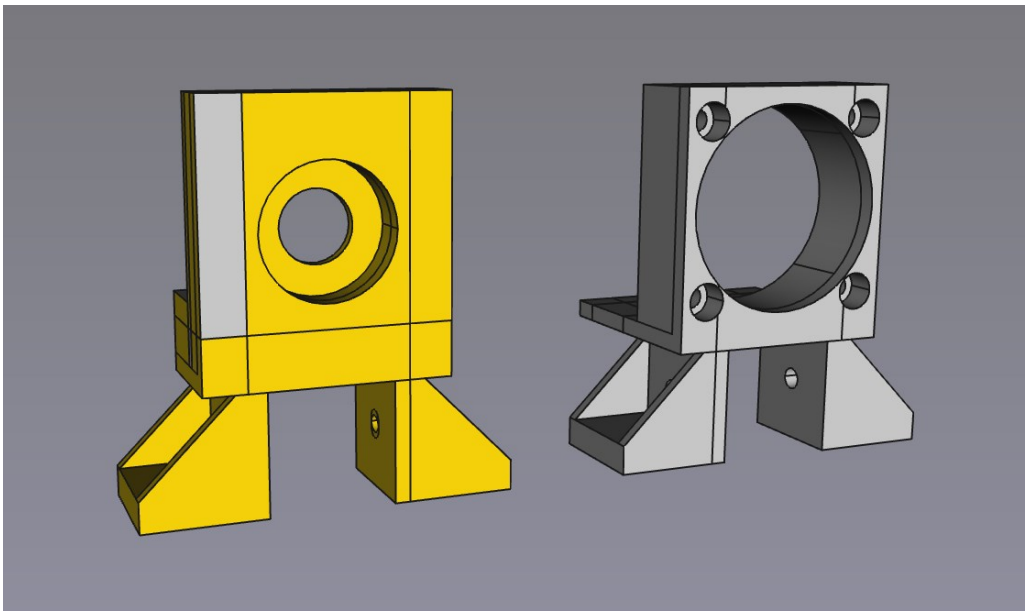


Figure 34: Driven Conveyor Roller

Figure 34 shows the mounts for the driven conveyor roller, the right mount will mount the NEMA17 stepper through four M3 screws, and the left mount holds the other side of the roller by allowing it to rotate freely in the mount. The left mount contains a space for a 608ZZ bearing to be inserted, which allows the roller to rotate freely without friction (8x22x7mm). There is a clear 20x20mm slot for the 2020 extrusion frame to be inserted into, allowing the mounts to attach to the frame, securing it in place. The left mount can also be disassembled to allow for the belt and roller to be inserted into the system easily.

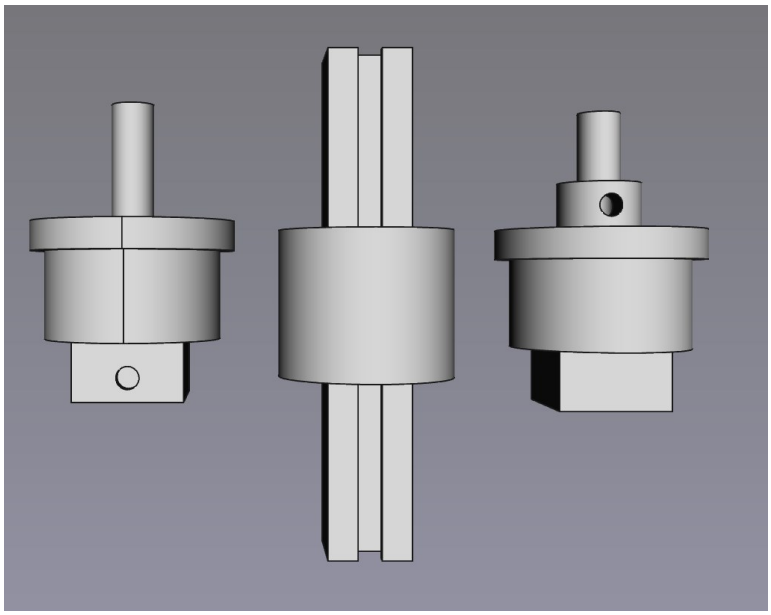


Figure 35: Conveyor Belt Rollers

Figure 35 shows the rollers themselves, with non-driven end rollers on the left, the adjustable-length roller body in the middle, and the motor-coupled roller on the right. During initial development, it was unsure what the width of the conveyor belt would be, so a design was made to allow for the length of the rollers to be adjusted. This is done by using a screw on the roller ends to adjust where it locks onto the roller body, enabling the length of the roller to be adjusted. The roller is coupled to the motor by screwing an M3 screw into the side of the roller which pushes against the motor shaft, allowing the motor to drive the roller. Figure 36 shows the adjustable roller design.

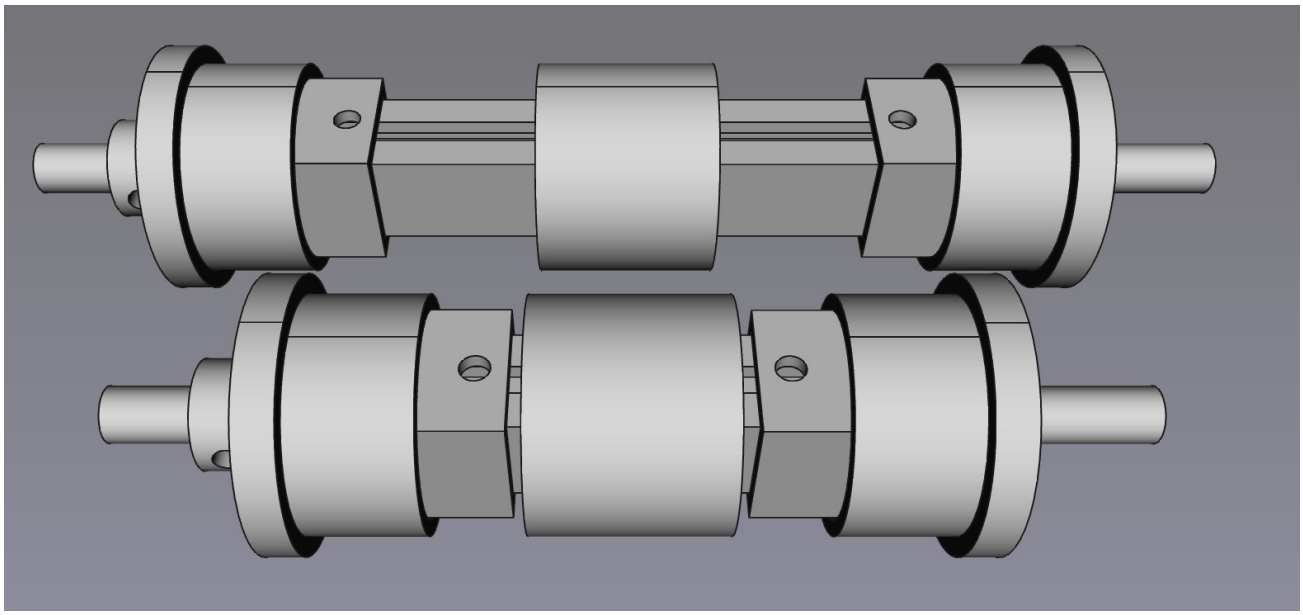


Figure 36: Adjustable Conveyor Rollers

On the other side of the conveyor belt, the belt tensioner is used to ensure the belt is taut as shown in Figure 37.

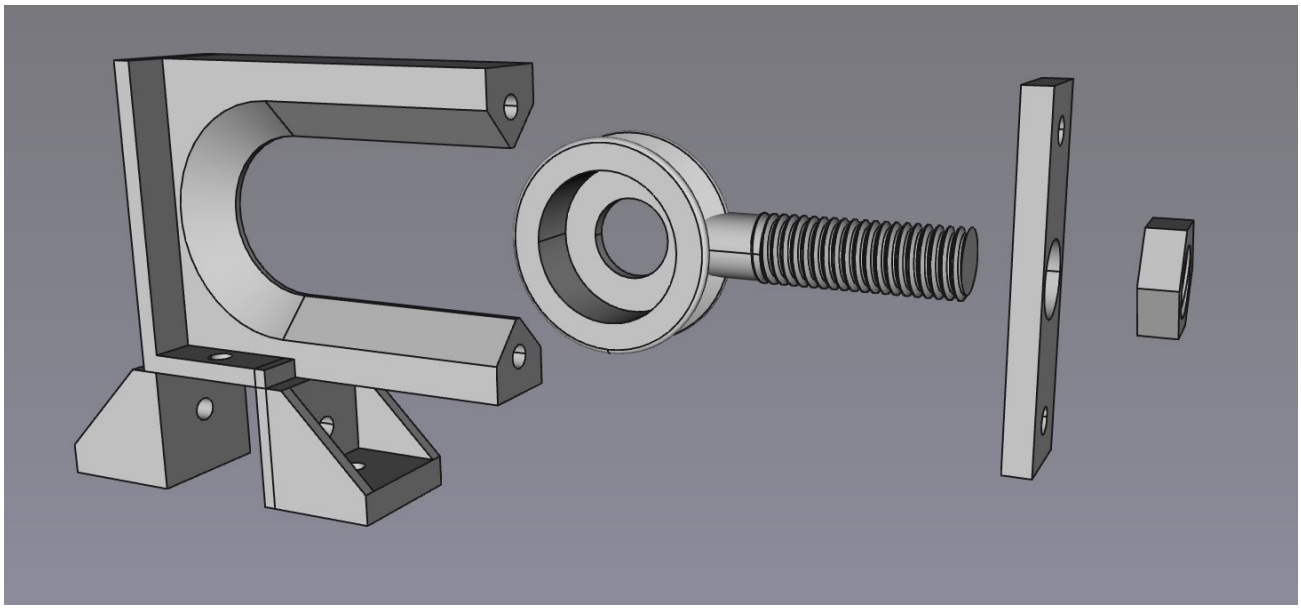


Figure 37: Belt tensioner design

As usual, the belt tensioner is mounted to the 2020 extrusion frame using M3 screws and T-nuts. The tensioner is made of 4 distinct parts;

Belt Tensioner Mount: The mount is used to attach the tensioner to the 2020 extrusion frame. The mount contains a bevelled edge to allow the tensioner to slide in and out of the mount.

Belt Tensioner: The tensioner is used to tension the conveyor belt. It slots into the mount, and features a 608ZZ bearing to allow the rollers to rotate freely. It is threaded to allow the belt tensioner nut to be screwed in.

Belt Tensioner Cover: The cover is used to ensure that the tensioner does not slip out, and provides something for the nut to press against to tension the belt.

Belt Tensioner Nut: The nut is used to tension the belt. It is screwed into the tensioner cover, and presses against the tensioner to tension the belt.

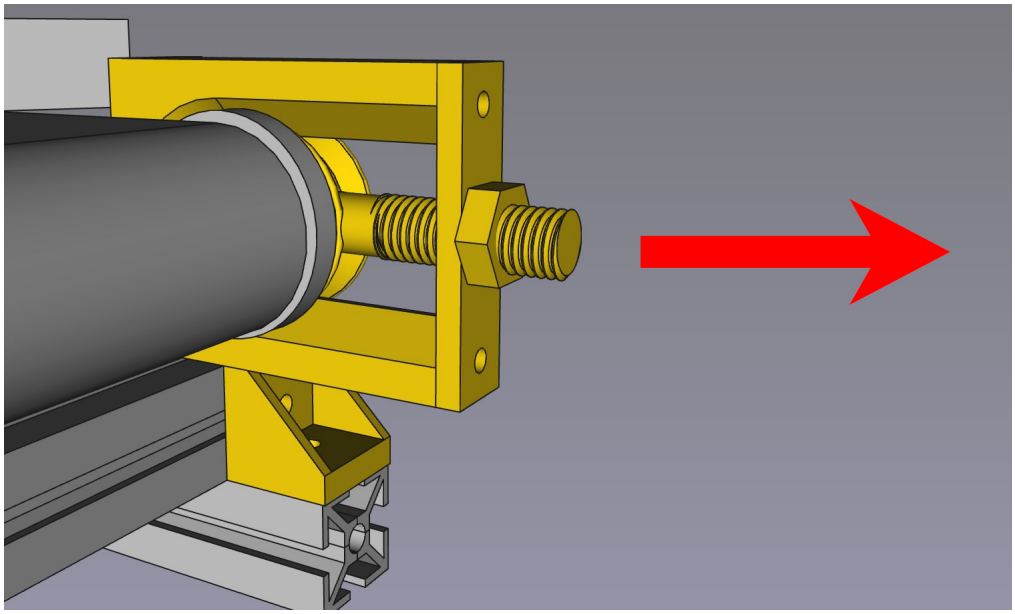


Figure 38: Belt tensioner in use

To tension the belt, the nut is simply turned which forces the tensioner to retract or extend in the mount,

tensioning the belt. As shown in Figure 38, tightening the nut would force the tensioner to move in the direction of the arrow, causing the belt to be tensioned.

4.1.5 Sweeping Mechanism

The sweeping mechanism of the system contains many intricate parts. It is responsible for pushing components off the conveyor belt into the sorting bins. The mechanism is driven by a NEMA17 stepper motor, which is mounted to the 2020 extrusion frame of the system. The motor then drives a GT2 belt, which is connected to the sweeping arm, allowing it to move linearly down the length of the conveyor belt.

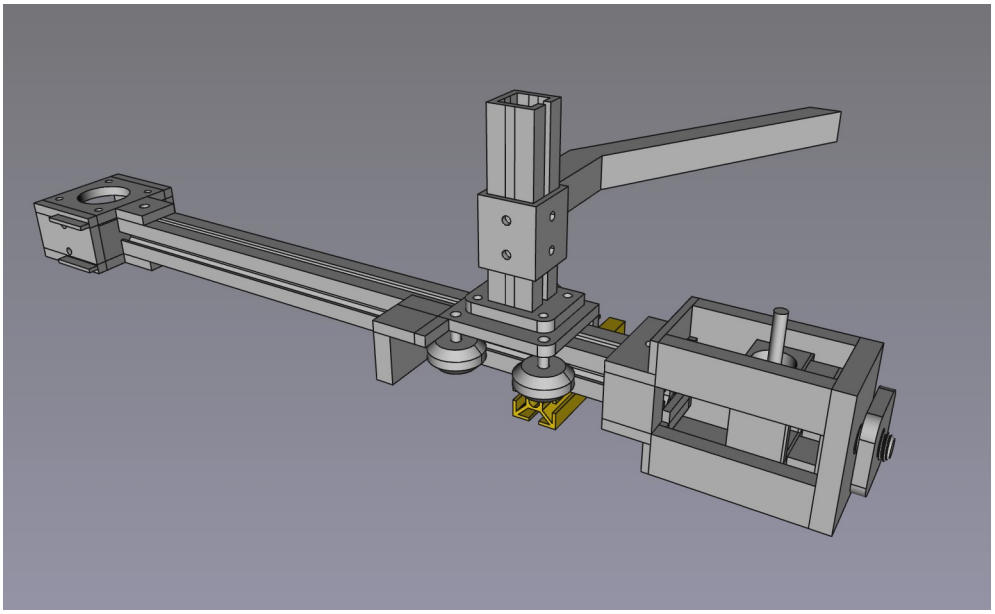


Figure 39: Full Sweeping Mechanism

The arm makes use of v-rollers to move smoothly along the 2020 extrusion frame, inspired by traditional x-gantry designs of 3D printers, as shown in Figure 40. The arm is mounted to the v-rollers using M3 screws and T-nuts, allowing it to be adjusted to the correct height.

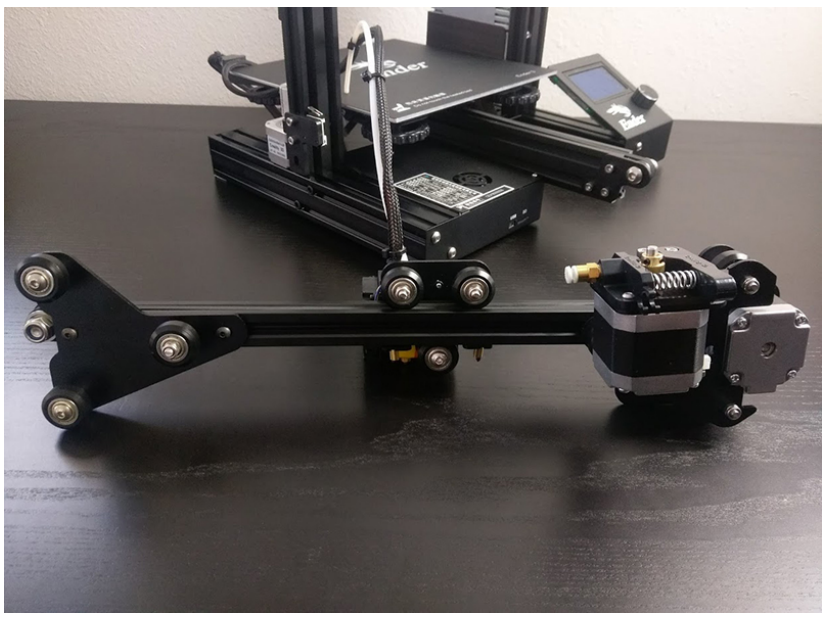


Figure 40: Ender 3 X-Gantry [63]

The sweeper contains several parts;

Motor and Endstop Mount: The mount is used to attach the NEMA17 stepper motor to the 2020 extrusion frame. It contains four screw holes for the motor, and two screw holes for the endstop. The endstop is used to detect when the sweeper arm has reached the end of its travel, and is used to prevent the motor from damaging the system. This allows precise control of the sweeper arm's position. In Figure 41, the motor and endstop mount is shown on the far left.

Belt Tensioner: The belt tensioner is used to tension the GT2 belt that drives the sweeper arm. It is mounted to the 2020 extrusion frame using M3 screws and T-nuts, and contains a 608ZZ bearing to allow the belt to rotate freely. The tensioner is used to tension the belt, ensuring that the belt is taut and does not slip. This design is similar to the conveyor belt tensioner in that it uses a nut to tension the belt. Inside the tensioner, there is a GT2 gear that the belt wraps around, and it is affixed to a 5mm shaft that is allowed to rotate freely using two 608ZZ bearings. The tensioner is allowed to move in only direction as it is housed in a "container" — similar to the conveyor belt tensioner. This is clearly shown in Figure 41.

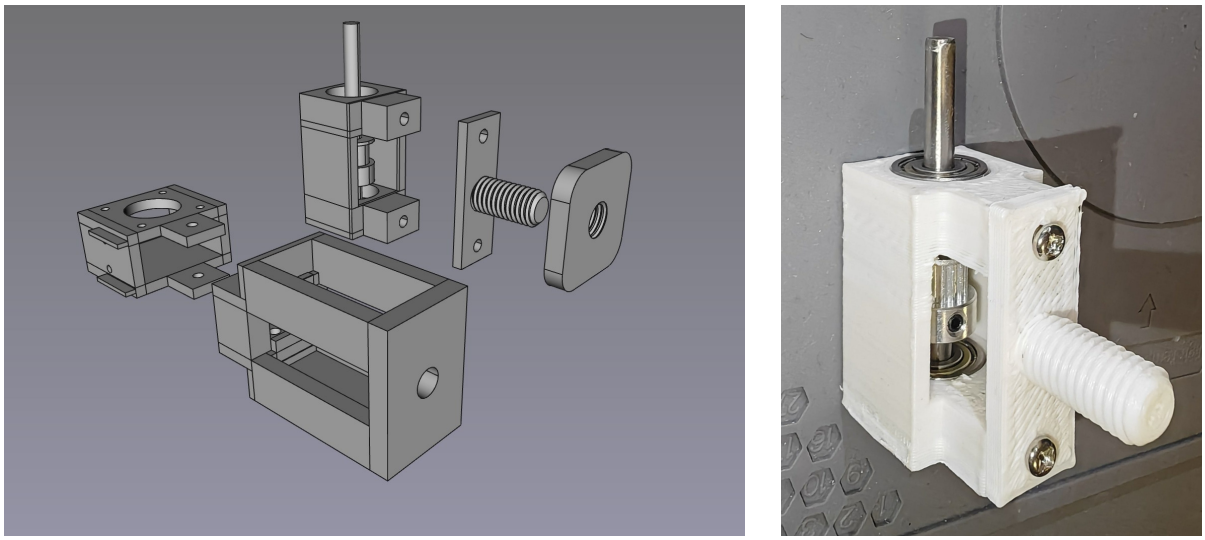


Figure 41: Motor and endstop mount, and belt tensioner

Gantry: The gantry rolls along the 2020 extrusion frame using v-rollers, and provides a mount for the sweeper arm and also triggers the endstop. The design is inspired by the Ender 3 as shown in Figure 40, and is used to provide a smooth linear motion for the sweeper arm. Three M4 holes are present in the gantry to mount the v-rollers, and there are slots for the timing belt to be inserted into. Four M3 holes are present to mount the sweeper arm, and the gantry has a large bumper on the end to trigger the endstop. The gantry is shown in Figure 42.

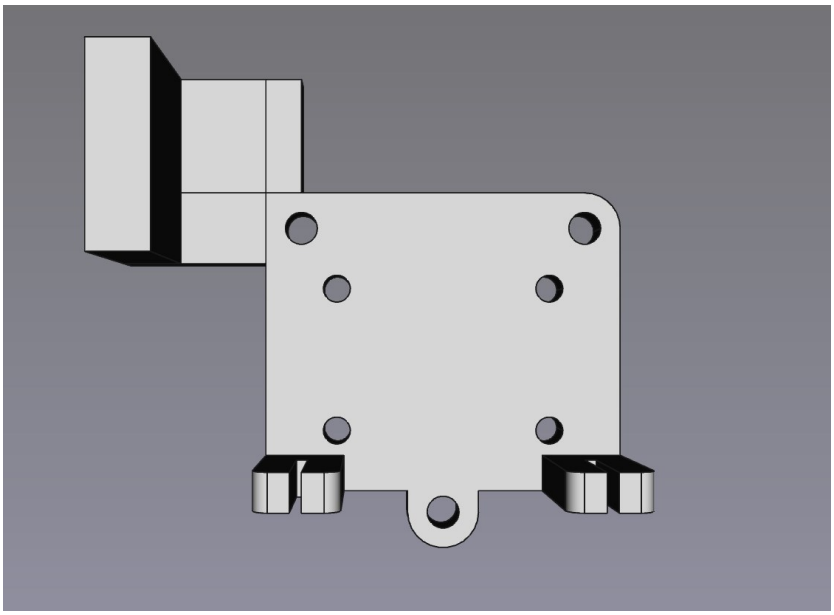


Figure 42: Gantry

Sweeper Arm: The sweeper arm is the part of the system that pushes components off the conveyor belt into the sorting bins. In Figure 43, it attaches vertically to a mount using M3 screws and T-nuts, and mounts directly to the gantry. The v-rollers on the gantry are also visible for reference.

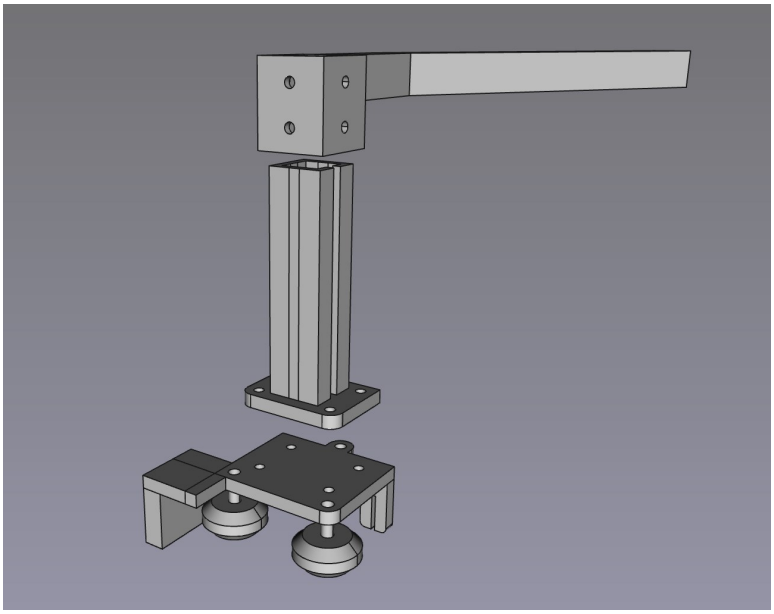


Figure 43: Sweeper arm with v-rollers

Bin: The bin is used to catch components that are pushed off the conveyor belt by the sweeper arm. There are two parts to the bin; the bin itself and the extrusion mount. Originally, it was designed to have the extrusion mount be secured to the extrusion using M3 screws and T-nuts, but this was found to be unnecessary as the tight fit of the extrusion mount was sufficient to hold it in place. The bin is shown in Figure 44.

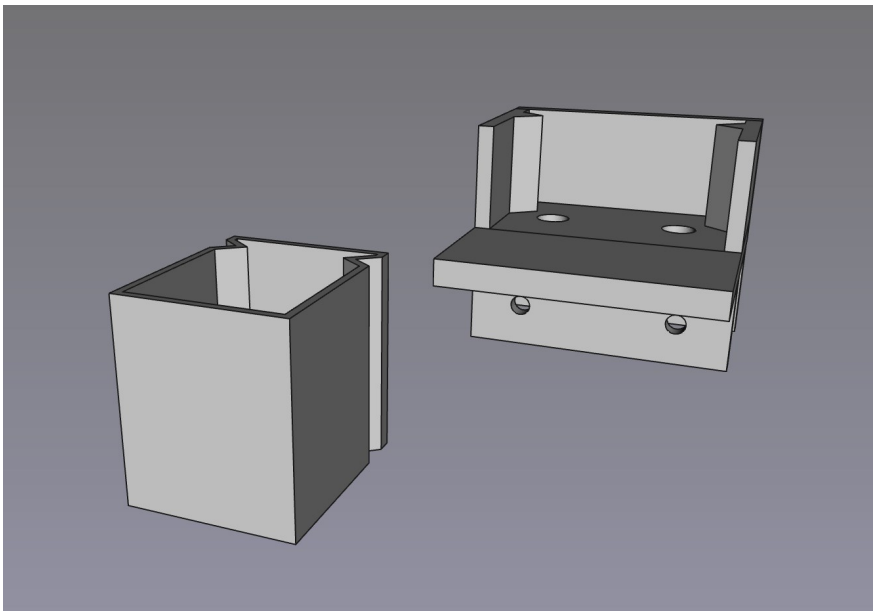


Figure 44: Bin and extrusion mount

As shown in Figure 44, the bin simply slides into the extrusion mount due to the triangle insets, constraining it to only move directly up and down. There are also countersunk holes in the bin to allow for M3 screws to be inserted to secure the bin to the extrusion mount, but this was found to be unnecessary.

4.2 Electronics and Wiring

As the system requires the use of moving parts, such as the conveyor belt and sweeper arm, there is a need to design electronics that can control these parts. For future reference, the following pins are connected to the following components:

Component	Pin	Description
WS2812B LED Strip	GPIO10	Controls the colour of the LEDs on the strip.
NEMA17 Stepper Motor (Conveyor Belt)	GPIO27 (DIR) GPIO18 (STEP)	Controls the direction and steps of the motor.
NEMA17 Stepper Motor (Sweeper Arm)	GPIO6 (DIR) GPIO3 (STEP)	Controls the direction and steps of the motor.
IR Break Beam Sensor	GPIO7	Detects when a component is on the conveyor belt.
Limit Switch	GPIO17	Detects when the sweeper arm is at its minimum position.
Okdo OV5647 Camera	CSI-2	Captures images for the computer vision system.
LCD Display	HDMI USB	Displays the user interface, powered directly from the Pi.

Table 5: Component Connections

For signal connections, standard 28AWG jumper wire is used to connect the components to the Pi.

4.2.1 Power Supply

As mentioned in Subsubsection 3.2.7, there are two XL4015 buck converters used to reduce the 24V output of the power supply to 5V and 12V. 18 AWG wire is used to both connect the 24V input to the buck converters, and to connect the output of the converters to the USB breakout boards. A standard 3-pin power cable connects the PSU's IEC C14 socket to the mains power.

4.2.2 WS2812B LED Strip

To illuminate the components on the conveyor belt, a WS2812B LED strip was used. It is powered with 5V, however it cannot be powered directly from the Pi due to the high current draw of the strip. Instead, a USB-C breakout board is connected to the 5V input pins, and then a USB-C cable is connected from the power supply to the breakout board, allowing the LED strip to be powered from the 5V output of the power supply. A USB hub is used to allow both the Pi and the LED strip to be powered from the same USB port on the power supply.

Figure 45: LED rainbow startup sequence

The SIG pin of the LED strip is connected to GPIO18 on the Pi, which is used to control the colour of the LEDs. The strip has been cut into a square shape to fit around the camera, allowing it to uniformly illuminate the components on the conveyor belt.

As shown in Figure 45, the LED strip has a rainbow startup sequence, which is used to indicate that the system is ready to start. This was added as there were previously issues with the SPI interface, which caused the LEDs to not light up correctly, as mentioned in Subsection 4.5. The rainbow sequence indicates that the system is ready to start.

4.2.3 Motor Control

As mentioned in Subsubsection 3.2.5, the system requires two NEMA17 stepper motors to control the conveyor belt and sweeper arm, and they controlled by TMC2209s as shown in Figure 17 and Figure 18. The TMC2209 requires the following connections:

Connection	Description
VIO	A voltage matching the controller's logic level (3.3V or 5V). For the Pi, this is 3.3V.
GND	Ground connection.
VM	Motor supply voltage. The NEMA17s can be powered by 12-36V [34], and the TMC2209 can handle between 4.75V and 29V [35]. The 12V from the power supply is used.
STEP	Step input for the motor, to be driven by the controller.
DIR	Direction input for the motor, to be driven by the controller.
EN	Enable input for the motor, to be driven by the controller, but is directly connected to VIO to enable the motor.

Table 6: Stepper Motor Driver Connections

The motor is driven by the controller by sending pulses to the STEP pin, and the direction of the motor is controlled by the DIR pin, as such the UART connections are not needed. The Pi only has one UART port, so to maintain consistency, both motors are instead controlled using STEP and DIR. The precision provided by microstepping is not required for the system, so the microstepping pins are not connected.

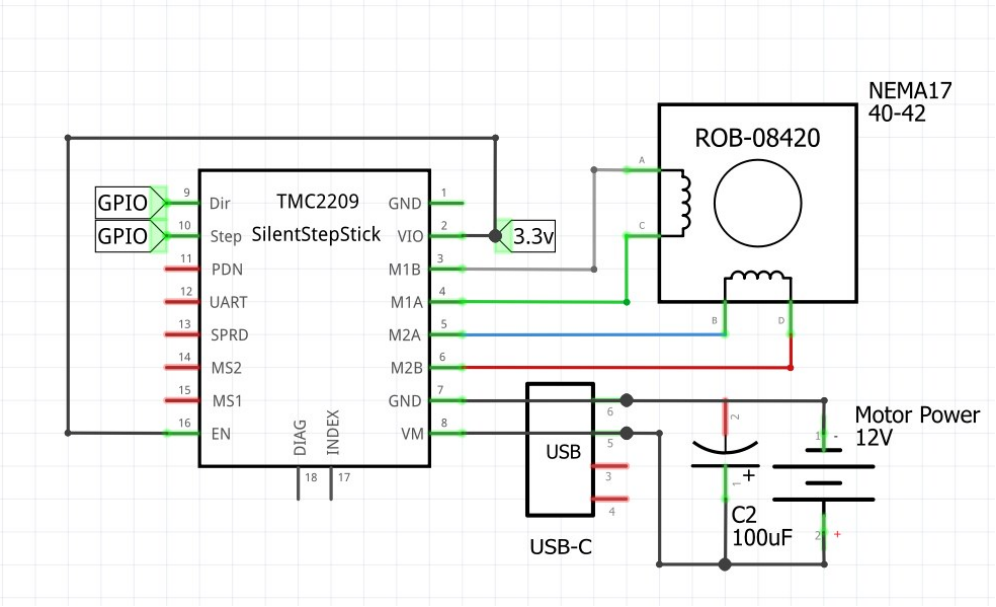
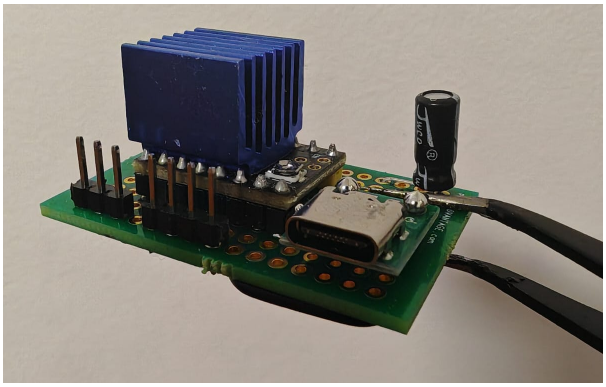


Figure 46: Schematic for NEMA17 Stepper Motor Control designed in Fritzing [64]

After designing the schematic, the connections were then realised on a prototyping board, making use of the TMC2209 driver, a breakout board for a USB-C connection to provide the 12V power from the PSU, and pin headers to allow connection to the Pi and the motor.



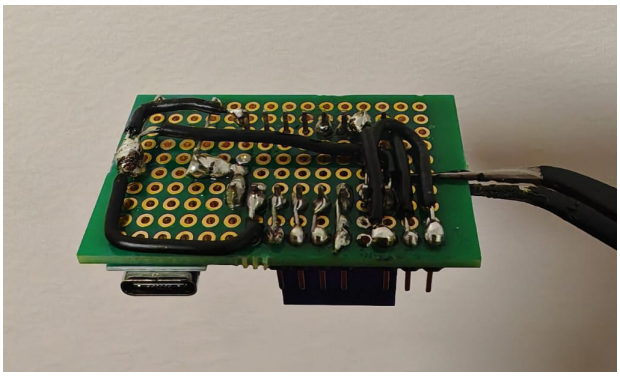


Figure 47: PCB for NEMA17 Stepper Motor Control

Two of these PCBs were produced due to the need for two stepper motors, and they were then connected to the Pi using the GPIO pins as shown in Table 5. The boards receive power using standard USB-C cables from the PSU's 12V USB output, and the motors are connected to the boards using standard 4-pin stepper motor cables.

4.3 Computer Vision

As mentioned in Section 2 and Subsection 3.4, the YOLO-OBB model [46] is used to classify and detect the electrical components, and the regular YOLO model was used for resistor value detection.

4.3.1 YOLO Format and Dataset Collection Process

To train a YOLO model with the Ultralytics library [46], it is necessary to properly structure the dataset in the YOLO format. The YOLO format is a text file that contains the class label and the bounding box coordinates of the object in the image, and multiple lines imply that there are multiple objects in the image. As the dataset collection tool discussed in Subsubsection 4.4.4 automatically saves the images and labels in the YOLO format, the dataset was already in the correct format for training.

Each dataset has a corresponding `.yaml` file that contains the class names as shown in Code Snippet 1, and relative file paths to the images and labels. The `.yaml` file is used to load the dataset into the YOLO training process. Note that there is no test set in the `.yaml` file, as YOLO does not explicitly support a test set. To get around this, another `.yaml` file was created that named the test set as the validation set, and validation is then run on the model using this file.

```yaml
  names:
- 0: resistor
- 1: capacitor
- 2: ceramic_cap
- 3: inductors
- 4: diodes
- 5: mosfet
- 6: transistor
- 7: leds
- 8: wire
- 9: ics
- 10: film_cap
path: ./full/current
train: images/train
val: images/val
```

Code Snippet 1: Dataset `.yaml` file

Depending on the YOLO model used, the format of the YOLO file can change. For the YOLO-OBB model, the format is as follows:

```
<class_num> <x0> <y0> <x1> <y1> <x2> <y2> <x3> <y3>
```

Where `class_num` is the numerical value of the class defined in the dataset's `.yaml` file, and `x0, y0, x1, y1, x2, y2, x3, y3` are the coordinates of the bounding box in the image normalised between 0 and 1. The normalisation ensures that the model is scale-invariant and image size agnostic.

For the regular AABB YOLO model, the format is as follows:

```
<class_num> <x_center> <y_center> <width> <height>
```

Where `x_center, y_center, width, height` are all normalised between 0 and 1.

```
root/
|
|---images/
|    |-- train/
|    |    |-- resistor_1.jpg
|    |    |-- capacitor_1.jpg
|    |    |-- ...
|    |
|    |-- val/
|    |    |-- capacitor_2.jpg
|    |    |-- inductor_1.jpg
|    |    |-- ...
|    |
|---labels/
|    |-- train/
|    |    |-- resistor_1.txt
|    |    |-- capacitor_1.txt
|    |    |-- ...
|    |
|    |-- val/
|    |    |-- capacitor_2.txt
|    |    |-- inductor_1.txt
|    |    |-- ...
```

Code Snippet 2: Dataset Folder Structure

The images and labels must have the same basename (i.e. the same filename excluding the extension) to ensure that the labels are correctly matched to the images. The images must be organised in folders similiar to the above structure.

4.3.2 Component Identification

Using the dataset collection tool discussed in Subsubsection 4.4.4, a total of 974 + 100 background images were collected across 8 classes. The distribution of the dataset is shown in the following table:

Class	**Number of Images**
Resistors	259
Capacitors	164
Ceramic Capacitors	264
Film Capacitors	66
Inductors	52
LEDs	96
Wires	75
Background	100

Table 7: Distribution of the dataset

It is important to note that the 7 classes, and background, shown above are not the same type of components that were discussed in Section 1. It was decided to do a smaller subset of classes due to the time it takes to collect a dataset, and the fact that the model can be extended to include more classes in the future. This model therefore serves as a proof of concept, and the model can be easily extended to

include more classes in the future, due to the flexibility of the dataset collection tool and the YOLOv8 models.

A notebook called `detection-trainer.ipynb` was developed that handled the organisation of the dataset (include train-val-test splitting), the training of the model, the evaluation of the model, and the ability to run inference on a particular dataset to see how the model performs. The notebook makes it incredibly trivial to adjust hyperparameters and change specifically what the model is trained on. The notebook ensures that the original dataset remains untouched and organises the desired images and labels into a folder called 'current' to ensure that there is no risk of accidental data deletion, which could be catastrophic given the time it takes to collect a dataset. The notebook also automatically saves the best model weights and leverages the TensorBoard [65] support that YOLOv8 provides to visualise the training process.

For the training of the model, and to determine model performance, the dataset was split into a 70-20-10 split for training, validation, and testing respectively. Data augmentation was also used as part of the training process to increase the diversity of the dataset and improve the model's generalisation capabilities. The specific augmentations used were discussed in Subsubsection 3.4.1.

The following training parameters were set:

Parameter	**Value**	**Explanation**
Epochs	50	The number of times the model will see the entire dataset. Due to early stopping, the model will not necessarily train for the full 50 epochs.
Patience	5	The number of epochs to wait before early stopping; this helps to mitigate overfitting by cutting off training when the relative improvement between epochs is below a certain threshold.
Cosine LR Scheduler	True	A learning rate scheduler that adjusts the learning rate according to a cosine function. This helps to prevent the model from getting stuck in local minima during training and reaching a suboptimal solution.
Batch Size	-1	The number of images to be processed in one iteration. The YOLO training process automatically detects the optimal batch size based on the GPU memory available.
Class Loss	1.2	The weight given to the classification loss. It was more important that the model correctly classify the components than it was to detect the bounding boxes accurately.
Box Loss	1.0	The weight given to the bounding box loss.
DFL (Dynamic Focal Loss)	2.0	A loss function that helps to improve the accuracy of bounding box regression by focussing on the distribution of the predicted bounding box coordinates [66], rather than the actual coordinates themselves. This has the effect of also helping to manage class imbalance as the model is made to focus on the more rare classes.

Table 8: Training Parameters

The model was trained using an NVIDIA GTX 3080 Ti GPU with 16GB of VRAM, which is more than capable of handling the training process. The TensorBoard logs were carefully observed during training

to ensure that the model was converging as expected. The model trained from only 26 epochs due to early stopping, however, YOLO's validation testing discovered that epoch 21 performed the best, and the model was saved at this epoch. This model took only 3:18 minutes to train, a testament to YOLOs efficiency.

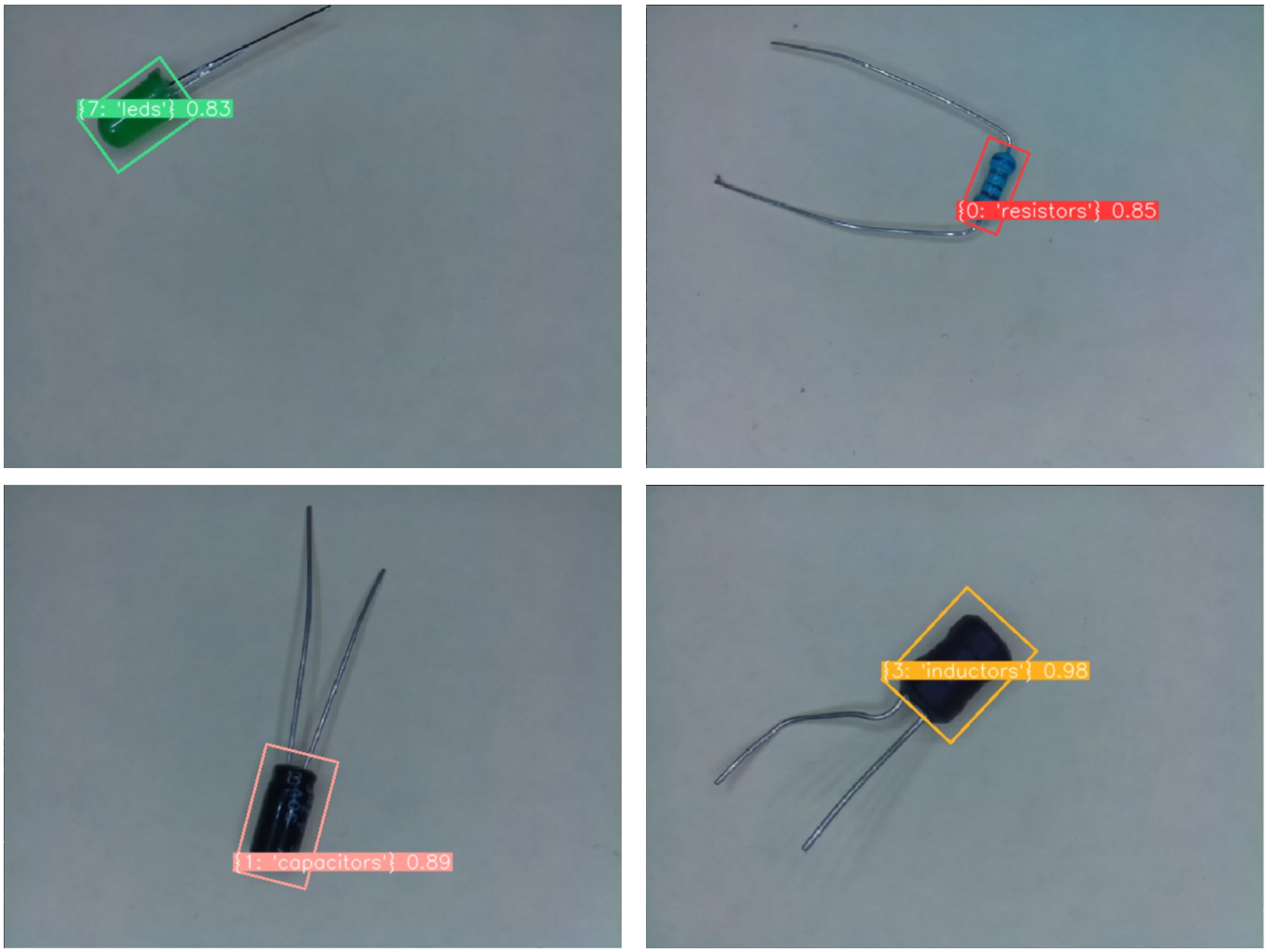


Figure 48: Inference ran using the `detection-trainer.ipynb` notebook, showing an LED, a resistor, capacitor and inductor, being correctly detected and classified.

For greater clarity, the accuracy of the above inferences are shown in the following table:

Component	**Confidence**	**Class**
LED	83%	LED
Resistor	85%	Resistor
Capacitor	89%	Capacitor
Inductor	98%	Inductor

Clearly, the YOLO-OBB model is able to identify and classify the components with a high degree of accuracy with tight, properly oriented bounding boxes. The model is also able to detect components at different orientations and scales, and is able to detect multiple components in the same image. A comprehensive evaluation is conducted in Subsection 5.3.

4.3.3 Component Value Identification

While the YOLOv8-obb model can identify components, it cannot determine the value of the component, for example the resistance of a resistor or the capacitance of a capacitor.

Originally, the plan was to use the YOLO-OBB model to properly orient the component so that they could be passed into another model for value identification. For regular components, like ceramic capacitors, the value is printed on the component in a standard format, which can be read using OCR (Optical Character Recognition) models and some image preprocessing. However, for resistors, the value is encoded in the colour bands on the resistor, which are relatively small. For this reason, a regular YOLO model was trained to detect and classify colour bands on resistors, and then determine the value of the resistor programmatically, as done in the resistor band data annotation tool discussed in Figure 61.

4.3.4 Resistor Value Detection

In order to generate the images for the resistor value detection model, the model from the previous section was used to detect the resistors in the images. The generated bounding boxes were then used to crop the images and generate a dataset of resistor images, which were already labelled with the resistor values as explained in Subsubsection 4.4.4.

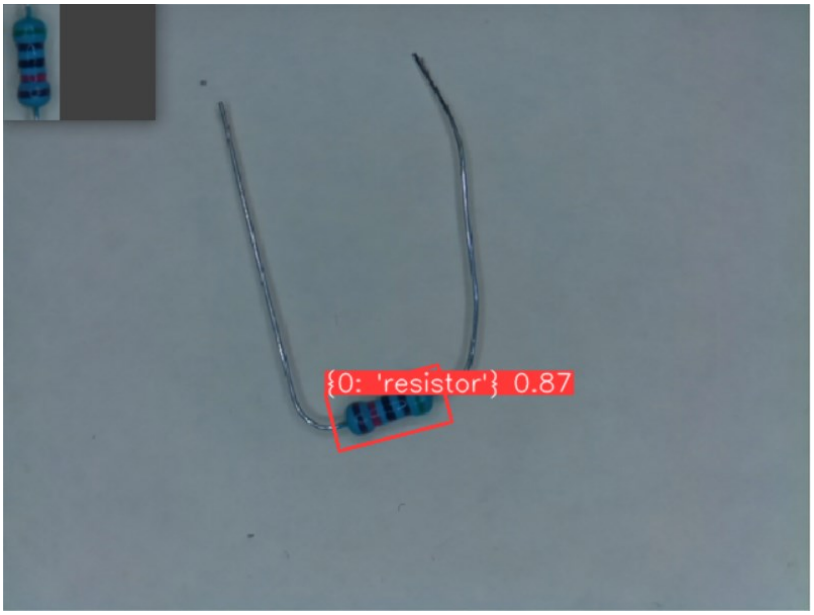


Figure 49: Resistor Identification and Cropping

As shown in Figure 49, a small window opens up that shows the cropped resistor image using OpenCV [38]. The images were then saved to a folder structure matching Code Snippet 2, allowing the resistor value detection model to be trained.

Similiar to the component identification model, the resistor value detection model made use of a `resistor_trainer.ipynb` notebook that handled the organisation of the dataset, the training of the model, the evaluation of the model, and the ability to run inference on the test set. The notebook also automatically saved the best model weights and leveraged the TensorBoard support that YOLOv8 provides to visualise the training process.

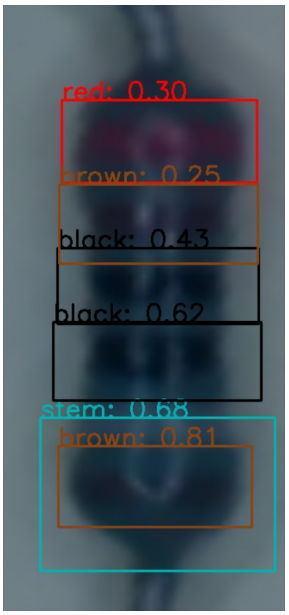


Figure 50: Resistor Band Detection

4.3.5 Sparsification and Deployment

Unfortunately, although SparseML is an incredibly powerful tool, it is not yet compatible with the YOLOv8-OBB model; usually, the SparseML framework integrates with Ultralytics' YOLO models, providing a wrapper that neatly allows sparsification to happen with minimal effort, however sparsification from scratch requires that the optimiser be exposed and a manual training and validation loop be written. This would be possible, however the YOLOv8-OBB model does not expose the optimiser during training, so it would be necessary to rewrite the training loop from scratch, and given the complexity of having to manage the three seperate losses (classification, bounding box, and dfl loss), this would be a significant amount of work. Given the time constraints of the project, it was decided that the model would not be sparsified. This was one of the major design decisions that was made during the project due to the OBB model only being released in January 2024 [67], after the Background Research phase had been completed.

However, although sparsification through SparseML was not possible, the model can still be pruned to achieve a similar effect. Pruning is the process of removing weights from the model that are close to zero, and can be done using the PyTorch pruning library. The model can be pruned to a certain sparsity level, and then fine-tuned to recover the lost accuracy. However, anecdotal evidence seems to suggest that the manual pruning of the YOLOv8 models causes the models to have a slower inference time [68]. This could be due to YOLO's complex and efficient architecture, making use of parallel computation and residual connections to speed up inference time [11]. Pruning the model could disrupt this architecture, and cause the model to be less efficient.

Despite this, DeepSparse [16] is still a viable deployment runtime for the model. As discussed in Section 2, the model does not need to be sparsified as a prerequisite for deployment, and can be deployed directly to DeepSparse, albeit without the benefits of sparsification. In Subsubsection 5.3.3, the model is deployed to DeepSparse and the inference latency is evaluated.

4.4 Software

As mentioned in Section 3, the software for the system is written in Python and is run entirely on the Raspberry Pi 4B. The software is divided into three main components: the user interface, the computer vision system, and the system controller which manages the hardware components of the system. The software is designed to be modular, with each component being able to run independently of the others.

This allows for easier debugging and testing of the system, as well as making it easier to add new features in the future. All written Python code adheres to a style defined in a `.pylintrc` file, which is a configuration file for the pylint [52] linter. This ensures that the code is consistent and readable. The code is also documented using docstrings for readability and maintainability.

All software constants are defined in a `constants.py` file, which is imported by all other Python files, allowing for easy modification of constants that define the behaviour of the system which is useful for testing and debugging.

Additionally, a lot of thought went into streamlining the development process to ensure that the system is easy to develop and maintain. For example, Visual Studio Code's [54] Remote SSH extension is used to develop the system remotely, as it allows developing the system on a much more powerful laptop, while using the familiar VSCode environment with any extensions that help streamline the development process. This is crucial as the Pi is not very powerful, so compiling and running the system on the Pi may be slow and cumbersome.

During development, the Pi connects to the laptop's Wi-Fi hotspot and is configured to be discoverable with the Pi's hostname, facilitating easy access to the Pi through SSH and a VNC Viewer without needing the Pi's IP address, which is dynamic. The Pi also makes use of SSH keys, allowing connection to the Pi without needing to enter a password every time, ensuring quick and easy access.

```python
# Allow development on non-Raspberry Pi devices
try:
    import RPi.GPIO as GPIO # type: ignore
    from rpi_ws281x import PixelStrip, Color
    GPIO.setmode(GPIO.BCM)
    print("Using real hardware!")
except ImportError:
    from src.common.simulate import GPIO
    from src.common.simulate import PixelStrip, Color
    print("Simulating missing hardware!")
```

Code Snippet 3: Cross-platform GPIO import

Additionally, as development is done on both a Windows laptop and the Pi, the code is written to be cross-platform, however certain libraries like the GPIO library are only available on the Pi, so a `simulate.py` file is used to simulate the GPIO pins on the laptop, allowing for development of the system without needing to be on the Pi as shown in Code Snippet 3.

```python
# Allow development on non-Raspberry Pi devices
class GPIO:
    BCM = 0
    IN, OUT = 0, 0
    HIGH, LOW = 0, 0
    PUD_DOWN, PUD_UP = 0, 0
    FALLING, RISING = 0, 0
    def setmode(_) -> None: pass
    def setup(_, *__, **___) -> None: pass
    def cleanup() -> None: pass
    def output(_, __) -> None: pass
    def add_event_detect(_, *__, **___) -> None: pass
    # PWM emulation
    class PWM:
        def __init__(self, _, __) -> None: pass
        def stop(self) -> None: pass
        def start(self, _) -> None: pass
        def ChangeDutyCycle(self, _) -> None: pass
        def ChangeFrequency(self, _) -> None: pass
```

Code Snippet 4: Example simulation of the GPIO library

As shown in Code Snippet 4, the `simulate.py` file contains a class called `GPIO` that emulates the GPIO library.

The Pi uses Git [51] for version control, and the repository is hosted on GitHub [69], with a dedicated branch for the Pi that is regularly updated with the main branch. The vision system and the laptop also maintain their own Git branches which are regularly updated with the main branch, ensuring that all systems are running the same code. This is crucial as it enables development on a more powerful laptop, and then synchronises the changes to the Pi, without having to manually copy any files over. The repository can be found in the Appendix 9.1.

4.4.1 User Interface

The user interface allows the user to view and command the state of the system by interacting with the touchscreen on the DFRobot 7" display.

Written in pygame [55] and pygame_gui [56], the user interface is controlled by the `LCD UI` class, which is responsible for drawing the various elements of the user interface, such as the camera feed, the buttons, and the text. The system operates at 30Hz, allow for responsiveness without overloading the system. A profiler was used to determine the performance of the system, and it was found that the system was able to run at 30Hz without any issues as discussed in Subsection 5.2

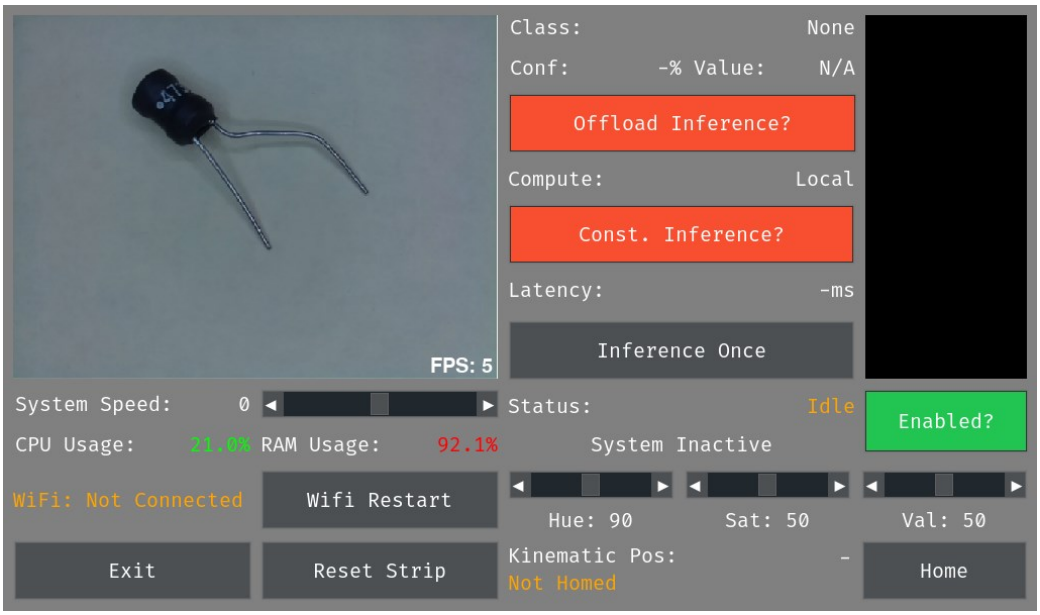


Figure 51: User Interface

As shown in Figure 51, the user interface consists of a multitude of features that allow the user to interact with the system. The user is able to view the live camera feed, where an inductor is clearly shown on screen. The user is able to adjust the overall system speed by adjusting the slider, which directly affects the System Controller. The CPU and RAM usage of the system is also shown on screen as these can help to judge the load of the system during development, and especially during inference. The colours of important system elements change depending on their value to indicate the state of the system, for example, the CPU and RAM usage are coloured red when they are high, and green when they are low.

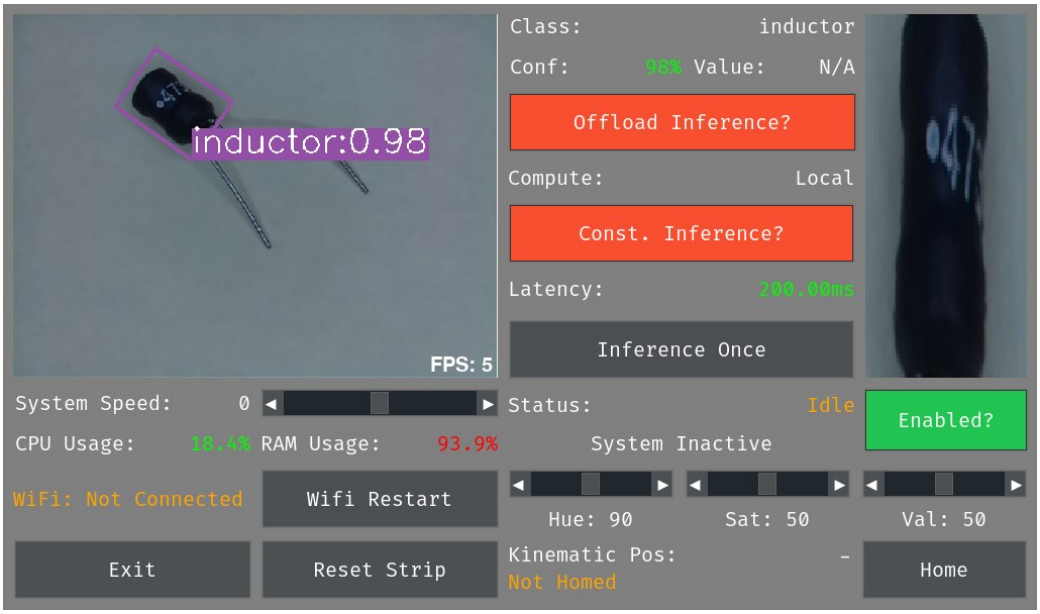


Figure 52: Inference Example

On the right side of the screen is information relating to the vision system, for example, what the classification of the component is, its value, and the confidence of the model in its inference. The component is typically displayed in the black area in the top right, as shown in Figure 52. As shown in Figure 52, the inductor from Figure 51 from the live camera feed has been classified as an inductor with

a confidence of 98%. The user interface also shows that inference latency is 200ms. The component is then cropped from its bounding box and then displayed in the top right corner of the screen.

As explained Subsection 4.4, the Pi is connected to a more powerful machine with SSH to facilitate faster development. For this reason, a Wi-Fi button is present on the UI to allow it to attempt to re-connect in the event of connection issues. The user interface also supports streaming frames to the machine where inference can be done much quicker, due to the presence of a GPU. This is done by toggling the "Offload Inference?" button. The "Const. Inference?" button means the Vision system is constantly processing frames rather than only when a beam break is detected. The "Inference Once" button allows the user to manually trigger inference, and performs inference on the current frame.

This was achieved using Python's `sockets`, `pickle` and `multiprocessing` library to facilitate the connection and data transfer between the two devices, and is discussed in more detail in Subsubsection 4.4.5.

Additionally, during the development of the system, the WS2812B needed to be calibrated as explained in Subsubsection 4.2.2. Due to this, there are HSV sliders for the user to adjust the colour of the LED strip, allowing for easy calibration of the LED strip. There is also a "Reset Strip" button to reinitialise the LED strip and display the LED start up sequence as shown in Figure 45.

The user interface also supports a "training mode" as discussed in Subsubsection 4.4.4, which is done programmatically by toggling a flag called `TRAININGMODE` in the `LCD UI` class.

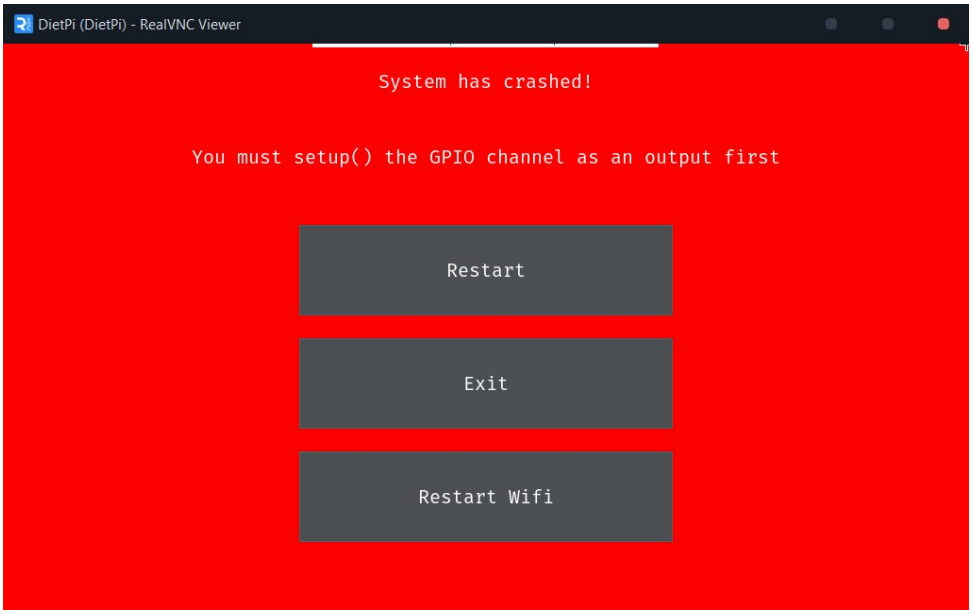


Figure 53: System crash during development

The LCD UI class is also capable of gracefully handling errors that may occur during the operation of the system, and the handling of these errors is delegated to the specific subsystem that caused the error. If a much more catastrophic error occurs, the system will still gracefully display an error screen, and allow the system to be restarted as shown in Figure 53. This facilitates not only the development process, but also prevents the system from having downtime in real-world applications as it prevents the need for a system restart. This is implemented by making use of Python's exception handling mechanism, and then switching to the error screen if an exception is raised.

The camera feed in the top left is controlled by the Vision Handler, which is responsible for delivering frames to the user interface. The Vision Handler is discussed in more detail in Subsubsection 4.4.2.

4.4.2 Vision Handler

The Vision Handler is responsible for managing the camera feed, and performs the following tasks:

- Capturing frames from the camera
- Performing inference by passing the frames to the computer vision model
- Gracefully handling errors that may occur during inference
- Gracefully handling capture errors (such as the camera being disconnected)
- The ability to live capture from the `TRAININGMODE=True` UI mode to allow for development off the Raspberry Pi

Figure 54 shows what the user interface displays when the camera is disconnected.

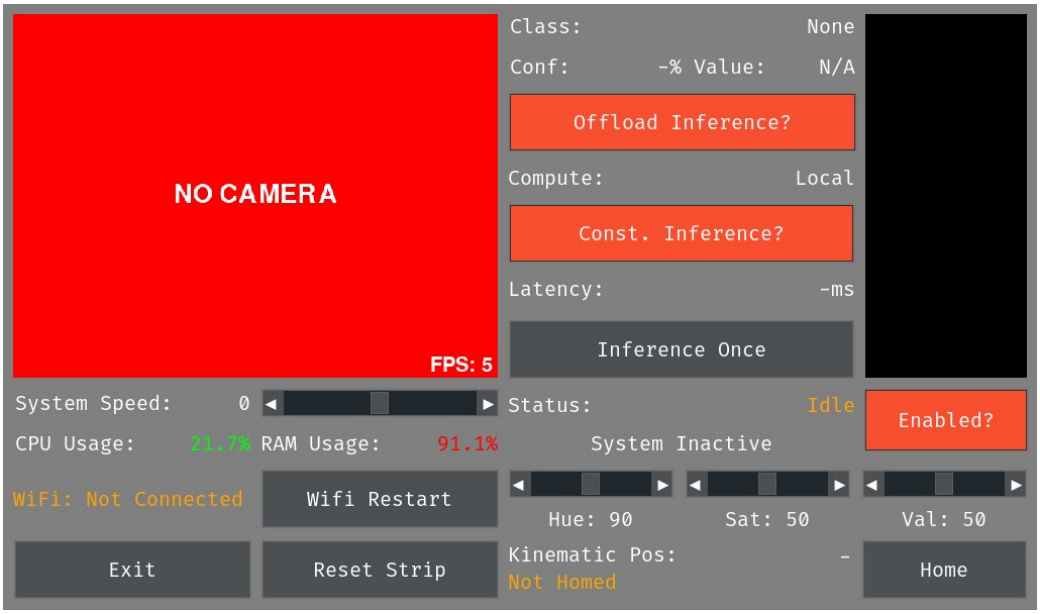


Figure 54: Seamless camera disconnect handling

4.4.3 System Controller

The System Controller heavily leverages Python's `Multiprocessing` library to allow for the system to run multiple processes concurrently, allowing for the system to be more responsive while displaying the `LCD UI`. The System Controller is responsible for the following tasks:

- Detecting a beam break and triggering procedure to sort the component.
- Coordinating the movement of the `Sweeper Controller` to sort the component.
- Relaying information back to the `LCD UI` to display the current state of the system.
- Handling the event of the beam break sensor being triggered while the system is in the process of sorting a component.
- Handling the event of the endstop being triggered when it is not supposed to be.

For the system to reactively sort components, the System Controller uses a beam break sensor to detect when a component is in the sorting area. When the beam break sensor is triggered, the System Controller triggers the `Sweeper Controller` to sort the component. This is handled using an interrupt, which is a signal that is sent to the CPU to indicate that a system event has occurred, and the CPU should take immediate action. Likewise, the endstop used to detect the sweeper arm's position is also handled using an interrupt to allow it to react in time to prevent damage to the system from the motors. The concurrency of the system is discussed in more detail in Subsubsection 4.4.5.

4.4.4 Dataset Collection

As mentioned in Figure 4.5.3, a custom dataset annotation tool was used to collect and annotate the dataset required to train the YOLO-OBB model used for component identification, and then extended to train a regular YOLO model for resistor value detection.

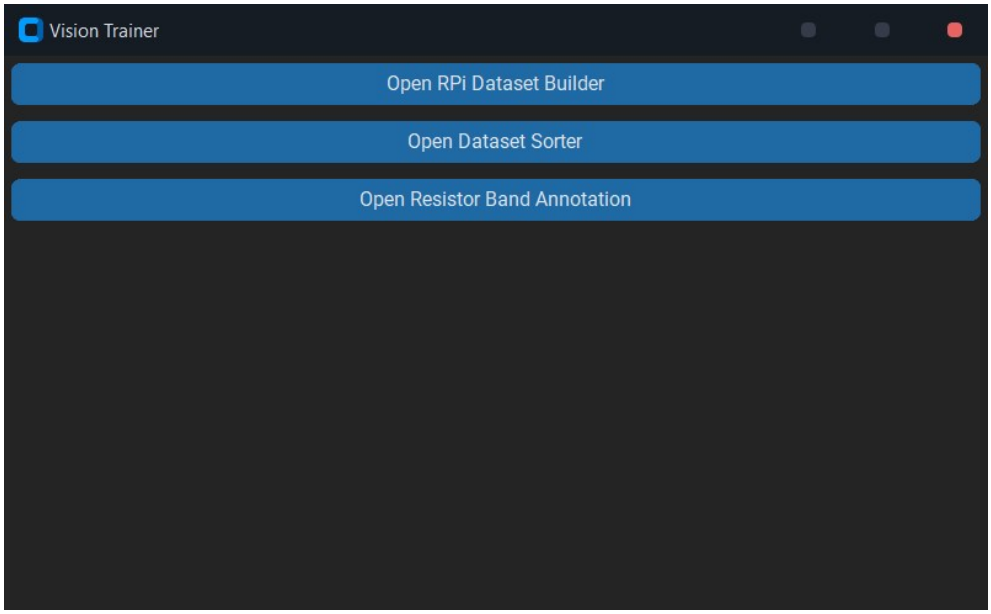


Figure 55: Vision Trainer Menu

Upon launch, the user is able to select which mode they would like to use, as shown in Figure 55. The user can select the mode by using the following buttons:

Button	Action
`RPi Dataset Builder`	Opens the dataset collection tool, capturing images from the VNC window connected to the Pi
`Open Dataset Sorter`	Opens the dataset sorter tool that sorts a local dataset and reads the labels if they exist
`Open Resistor Band Annotation`	Opens the resistor band annotation tool

Table 9: Main buttons for the dataset collection tool

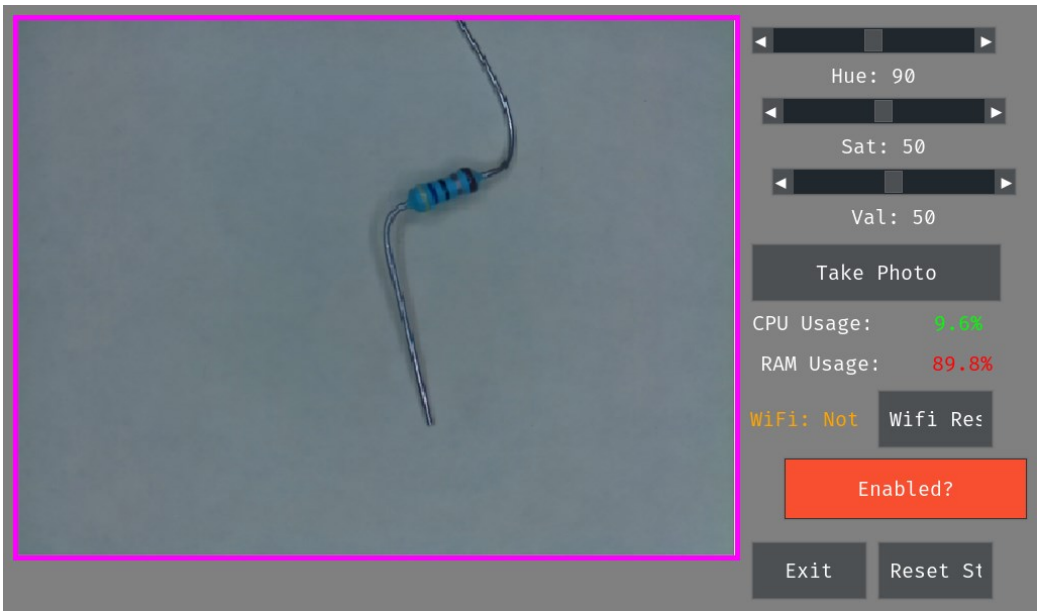


Figure 56: Training Mode Screen

As shown in Figure 56, to facilitate the collection of the dataset, the user interface was modified to include a "training mode" that allows the user to capture images of components to be used for training the computer vision system. As the camera feed must originate from the Pi, dataset collection is done on a machine connected to the Pi with VNC [53], and the user interface is displayed on the VNC client. When `TRAININGMODE` is enabled, a purple border surrounds the camera feed, enabling the dataset tool to identify where the camera feed is located on the VNC window.

The tool makes use of keybinds to capture images, to allow the user to capture images of components quickly, detailed in the Table 10.

Key	**Action**
`Space`	Capture from VNC window
`Enter`	Save image but only if the image is captured and labelled
`Escape`	Return to the component selection screen
`Mouse Left Click`	Draw line for OBB
`Mouse Middle Click`	Cancel OBB

Table 10: Main keybinds for the dataset collection tool

The dataset collection tool uses PyGetWindow [70] to identify the VNC window, and PyAutoGUI [71] to capture the images. OpenCV's [38] `cv2.findContours` function is used to find the largest contour within the image that is bordered by the purple rectangle, allowing it to effectively extract the camera feed from the VNC window.

```
    try:
        realVNCWindow = pygetwindow.getWindowsWithTitle(REALVNC_WINDOW_NAME)[0]
        realVNCWindow.activate()
        pygetwindow.getWindowsWithTitle("RPi Dataset Builder")[0].activate()
    except:
        try:
            realVNCWindow = pygetwindow.getWindowsWithTitle("Component Sorter")[0]
            realVNCWindow.activate()
            pygetwindow.getWindowsWithTitle("RPi Dataset Builder")[0].activate()
        except:
            self.imgBorder.configure(bg_color=BORDER_COLOUR_FAILED)
            print("No Window Found")
            return
    # Capture the image
    screenshotPil = pyautogui.screenshot(region=(realVNCWindow.left, realVNCWindow.top,
    ↪  realVNCWindow.width, realVNCWindow.height))
    # Convert to OpenCV format
    screenshotCv = numpy.array(screenshotPil)
    # Find the contours defined by the pink square
    mask = cv2.inRange(cv2.cvtColor(screenshotCv, cv2.COLOR_BGR2HSV), LOWER_THRESHOLD,
    ↪  UPPER_THRESHOLD)
    contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

Code Snippet 5: Camera feed extraction from VNC window

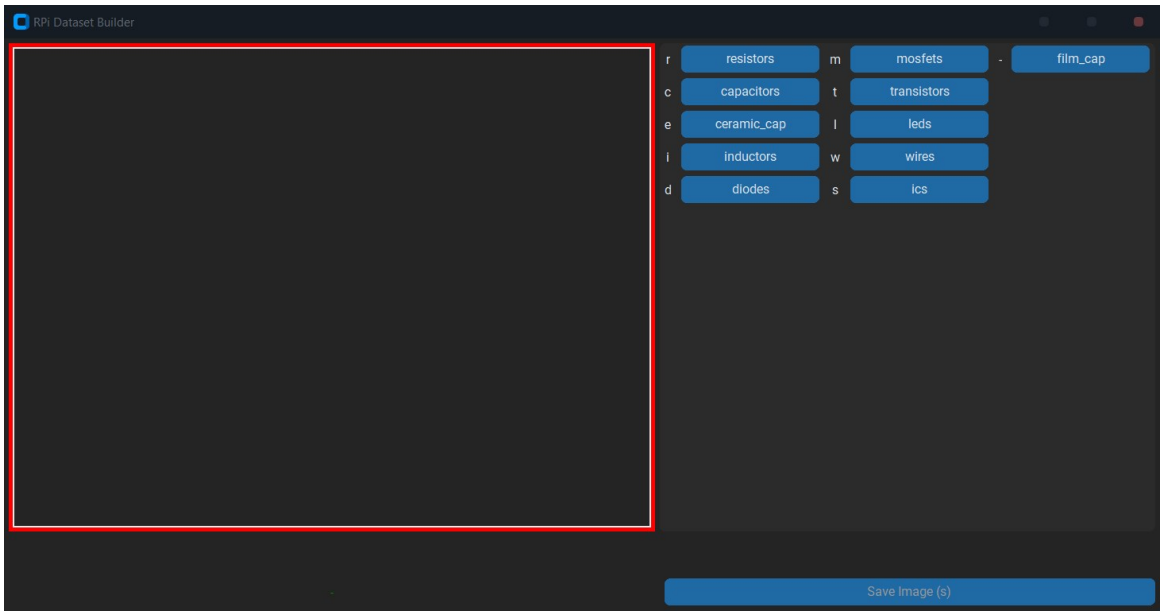


Figure 57: Uncaptured camera feed with red border

If the camera feed is not captured (it may be obscured), the border of the camera feed will turn red, as shown in Figure 57, indicating that the camera feed was not captured, otherwise it will turn green, as shown in Figure 58. In Figure 57, the user is selecting which component to sort on the component selection screen, and has a selection from the following components:

- Resistors
- Capacitors
- Ceramic Capacitors
- Inductors
- Diodes
- MOSFETs
- Transistors
- LEDs
- Wires
- ICs
- Film Capacitors

The user can select the component to sort by clicking on the component button or by using the hotkey indicated to the left of the button.

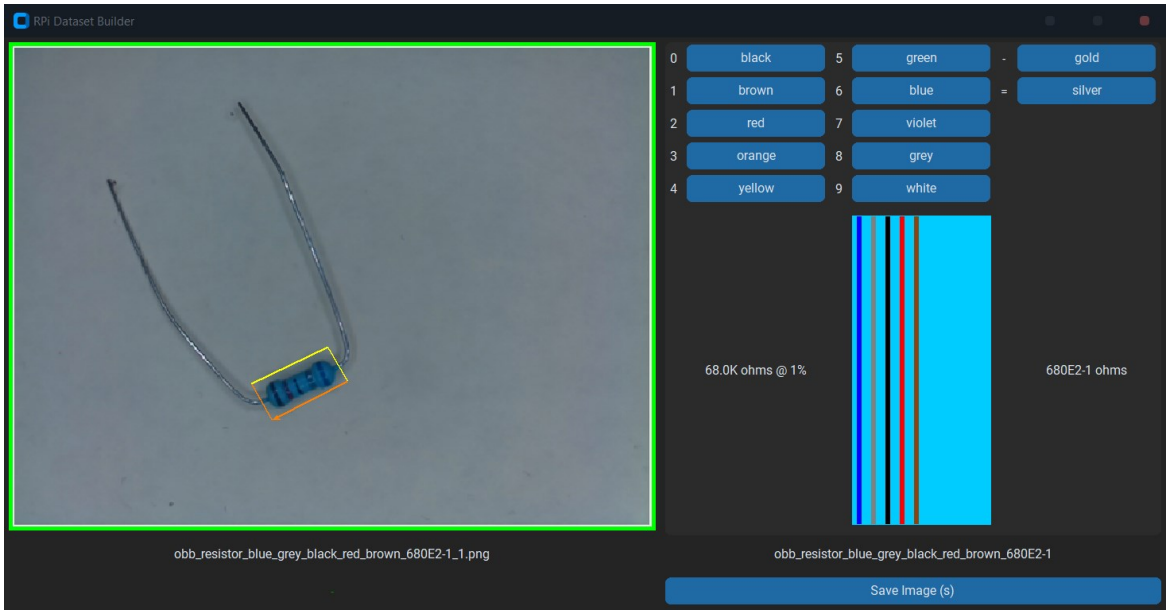


Figure 58: Successfully captured camera feed with green border and annotated resistor

In Figure 58, the user has successfully captured the camera feed, and has annotated a resistor. The tool has hotkeys for quickly annotating the band and retains the configuration of the band between images, allowing for quick annotation of resistors (as the user would take multiple angles of the same resistor). During band annotation, the program automatically calculates what the value of the resistor should be, including tolerances, and displays this on the screen, to help prevent the mislabelling of the resistors. The bands and value are then saved in the filename of the image and label, allowing the resistor value model to be trained based off the file names. For eaxmple, in Figure 58, the resistor will be saved as `obb_resistor_blue_grey_black_red_brown_680E2-1_1.png` ; the value of the resistor is 68 kΩ, with a tolerance of 1% and is the first image of this value in the dataset.

The user can also draw the OBB by clicking and dragging the mouse, and can cancel the OBB by clicking the middle mouse button. The tool automatically connects the first and last points to form the OBB, showing a gray dotted line so the user can verify the OBB is correct. The user can then save the image by pressing the `Enter` key, or return to the component selection screen by pressing the `Escape` key.

Originally, it was thought that the OBB model learns the correct left-right orientation of objects by the order of which the vertexes are joined, and this is represented in the tool with two orange arrows, showing the desired right-side up orientation of the resistor. However, this was found to be incorrect, and the OBB model only learns the bounding box of the object, and not the orientation of the object. This is discussed more in Subsection 4.5.

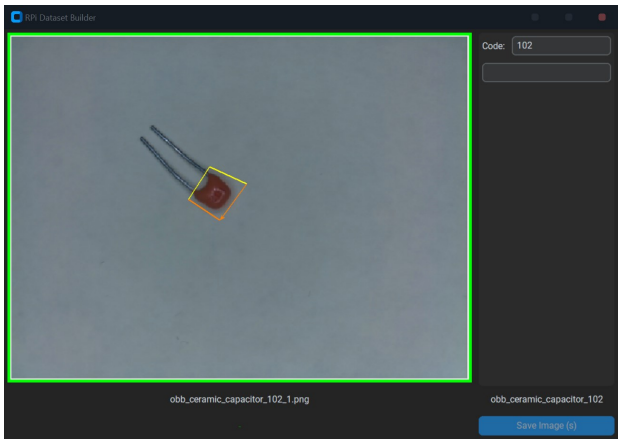


Figure 59: Annotated Ceramic Capacitor

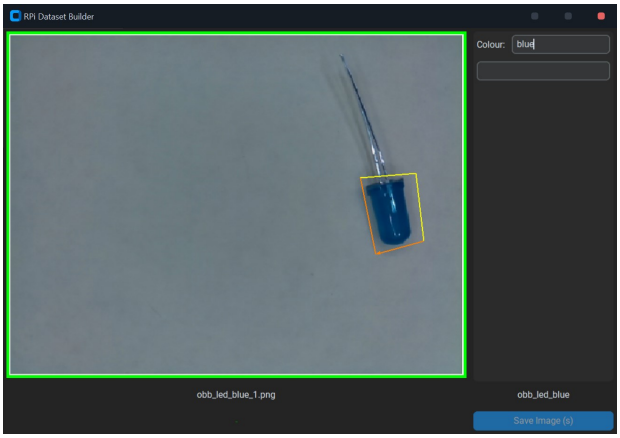


Figure 60: Annotated LED

Figure 59 shows an annotated ceramic capacitor, and Figure 60 shows an annotated LED, demonstrating the tool's ability to annotate components of different shapes and sizes.

In addition to regular component identification using the YOLO-OBB model, a separate model was trained to identify the value of resistors, which is discussed in more detail in Subsection 4.3. This dataset is collected by using the classification model to identify the resistor, and then the bounding box is used to crop the image to the resistor. The cropped image then contains the neccessary information to train the resistor value model, such as the bands and the value of the resistor, but is missing the location of the bands in the image, and the orientation of the resistor, which would help it determine what the first band is. This means a tool is still required to annotate the resistor bands.

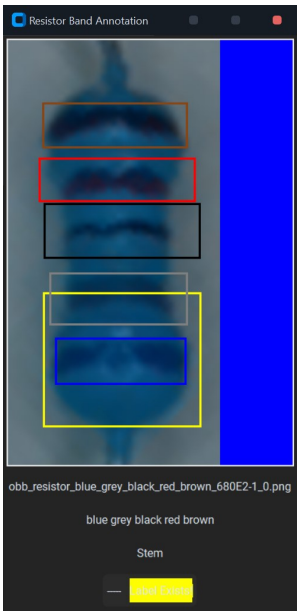


Figure 61: Resistor band annotation tool

As shown in Figure 61, the tool clearly shows a resistor being annotated, with the bands being drawn on the resistor. The user only has to click on the location of the bands, and does not need to manually annotate what band the user is clicking on; the information of the order of the bands in encoded in the filename as discussed earlier, so the tool can automatically determine what band the user is clicking on. The user only needs to indicate where the 'stem' of the resistor is, indicated by the yellow box, enabling the tool to identify the orientation of the resistor, and then allowing the resistor value model to be trained. Clicking on the resistor automatically draws a bounding box centered at the band location with the current band colour, and then saves the image and label in the same way in the YOLO format

(not the YOLO-OBB format as only AABB are needed as we care more for classification accuracy). The format is discussed in more detail in Subsection 4.3.

4.4.5 Concurrency and Optimisation

To ensure that the system is responsive and can handle multiple tasks concurrently, the system makes use of Python's `multiprocessing` library to run multiple processes concurrently. The system is divided into three main processes: the main process, the vision process, and the system controller process.

The main process is responsible for running the user interface, and it has the highest priority as it is responsible for displaying the state of the system to the user and must be responsive to user input. This is achieved by using an event loop to handle user input, and then updating the user interface accordingly, and also ensuring that no blocking operations are performed in the main process. A blocking process is an operation that pauses the execution of code until it is complete, and would prevent the event loop from executing and would "freeze" the user interface. To prevent this, all blocking operations are delegated to other processes, and the main process only waits for the results of these operations.

Pygame [55] and Pygamegui [56] make event handling easy, as they provide an event queue that allows the system to consume events as they occur, and then update the user interface accordingly. This allows the system to be responsive and handle multiple tasks concurrently.

The vision process is responsible for capturing frames from the camera, and then passing these frames to the computer vision model for inference. Inference must be run on a separate process due to the fact that inference is computationally expensive, making it a blocking operation as the system must wait for the inference to complete before it can continue. This has been achieved by using Python's `multiprocessing` library to run the vision process concurrently with the main process, allowing the system to capture frames and perform inference at the same time.

```
model = YOLO(modelPath)
print("Loaded YOLO model!")
while True:
    print("Waiting for frame")
    frame = frameQueue.get()
    start = time.time()
    print("Got frame")
    # Inference
    prediction = model.predict(frame)
    results = draw_results(frame, prediction)
    resultQueue.put(results)
    busyInference.clear()
    print(f"Inference took {time.time()-start:.2f}s")
```

```
# Consume the result
if not self.resultQueue.empty():
    results = self.resultQueue.get()
    endTime = time.time()
    # Update the LCD
    ...
# Produce a frame
if (self.doInference.is_set() or self.constInference.is_set()) and not self.busyInference.is_set():
    self.busyInference.set()
    if self.frameQueue.empty():
        self.startTime = time.time()
        self.frameQueue.put(pygame.surfarray.array3d(frame).swapaxes(0,1))
        self.doInference.clear()
```

Code Snippet 6: Producer and Consumer mechanism for the Vision Handler

In Code Snippet 6, the vision process is shown to be a producer-consumer mechanism, where the inference process (top) is the producer, and the main process (bottom) is the consumer. The mechanism makes use of multiprocessing queues and events which are multiprocessing synchronisation primitives

that allow for the communication between processes without the need for locks to ensure data integrity. If the same data is accessed by multiple processes at the same time, it can lead to data corruption, or race conditions, which can cause the system to behave unpredictably. The queues and events ensure that the data is accessed in a multiprocessing-safe manner, and that the processes are able to communicate effectively.

In this mechanism, the main process ensures that the frame queue is empty before putting a frame into the queue, and then sets the `doInference` event, which signals to the vision process that a frame is ready for inference. This check is necessary as the "put" operation is blocking, and will wait until the queue is empty before putting the frame into the queue, as its maximum length was set to 1. The vision process then waits for the `doInference` event to be set, and then performs inference on the frame, and then puts the result into the result queue. The main process then consumes the result from the result queue, checking if the result queue is empty for the same reason, before consuming the result. This mechanism successfully allows the vision process to perform inference concurrently with the main process, and ensures that the system is responsive and can handle multiple tasks concurrently.

Note that Python's `threading` library cannot be used for this purpose, as Python's Global Interpreter Lock (GIL) prevents multiple threads from executing Python code concurrently, and only allows one thread to execute Python code at a time. This makes it useful for I/O-bound tasks, but not for CPU-bound tasks, as the GIL would prevent the inference from taking place while the main process is running.

For offloaded inference to a more powerful machine, the system uses Python's `sockets` library to establish a connection between the Pi and the machine, and then uses Python's `pickle` library to serialise the frame and send it over the network. As the Pi is connected to the machine via its own Wi-Fi hotspot, a connection is easily established and frames can be sent over the network. The machine then deserialises the frame, performs inference, and then sends the results back to the Pi, which then displays the results on the user interface. This operation is I/O-bound, and could be threaded using Python's `threading` library, but as the system is already using `multiprocessing` and supports toggling between offloaded and on-device inference, it was decided to keep using `multiprocessing` for development simplicity.

For the system controller, the system controller process is responsible for detecting a beam break, and then triggering the system to sort the component. The beam-break is detected by the system controller using an interrupt using Python's `RPi.GPIO`, and then the system controller triggers a process to begin sorting the component.

Additionally, to ensure that the system has all the computational resources it needs to run effectively, the system runs off a minimal environment, with no desktop environment. To display the user interface, the system uses a custom "startx" session that only starts the necessary processes, and reduces the overhead of running a full desktop environment. "startx" is a program that starts the X Window System server and the X Window System, allowing GUI applications to be run on the system. Pygame has the capability to display to a minimal X server, and this helped to influence the design choice of picking Pygame as the user interface library.

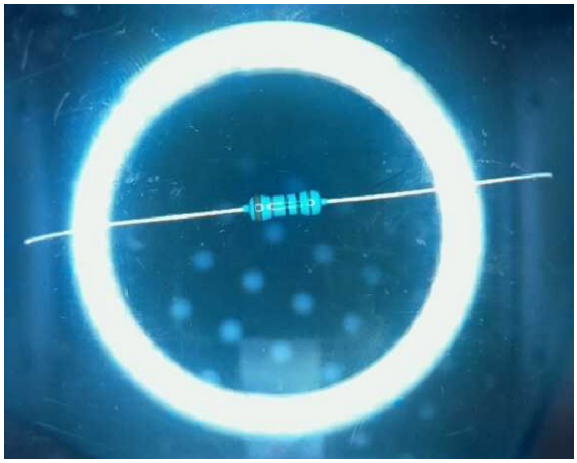
4.5 Problems and Changes

During the implementation of the system, several problems were encountered. Each subsystem had its own set of problems, and the solutions to these problems are discussed below.

4.5.1 Mechanical Design

FDM printing or 3D printing is typically an iterative process, where the design is printed, tested, and then modified based on the results. This further reinforces the need for a modular and parametric de-

sign, as it allows for easy modification of each component. After a few iterations with errors due to tolerances and slightly incorrect measurements, (not relevant to the technical content of this report) a design was produced that physically fit together and mounted the components as intended, however when combined with the software and electronics, more problems were encountered that were eventually overcome by making some design decisions.



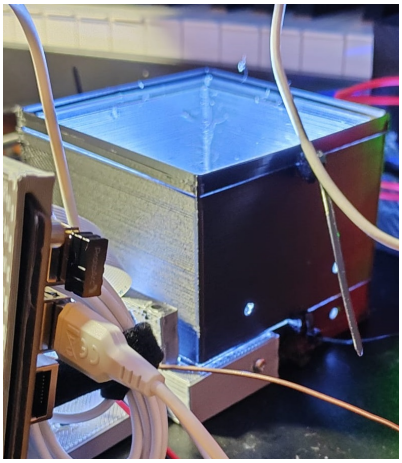


Figure 62: Glare from LED Ring and old camera housing

Originally, the system was to use a face-up camera with components mounted on an acrylic plate to allow the user to easily change the components as shown in Figure 62. However, this caused a problem with the LED ring, as the light would reflect off the acrylic plate, causing a glare that would obscure the image of the component. This design was then abandoned for the conveyor system that the system currently uses, as discussed in Subsection 4.1.

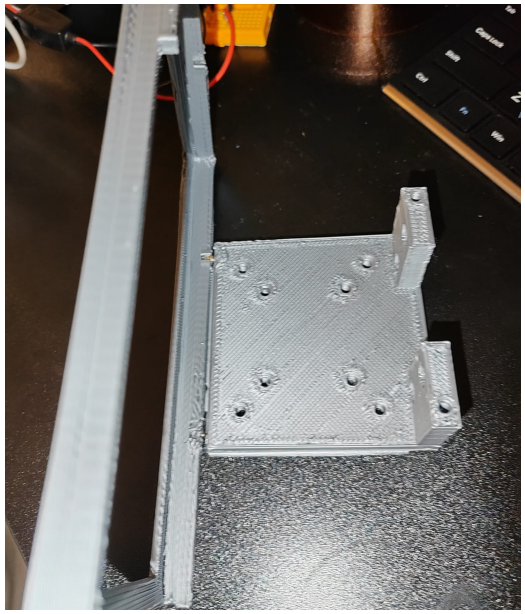


Figure 63: Unbraced LCD Mount

The initial design of the LCD cover was only mounted to the extrusion mount using two M3 screws at the base, which made it unstable and prone to wobbling. Over time, this would put stress on the plastic, causing it to break as shown in Figure 63. To solve this, a third mounting point was added to the LCD cover in the form of a brace that reaches halfway up the LCD cover, increasing the stability of the LCD, and preventing it from wobbling, as seen in Figure 32.

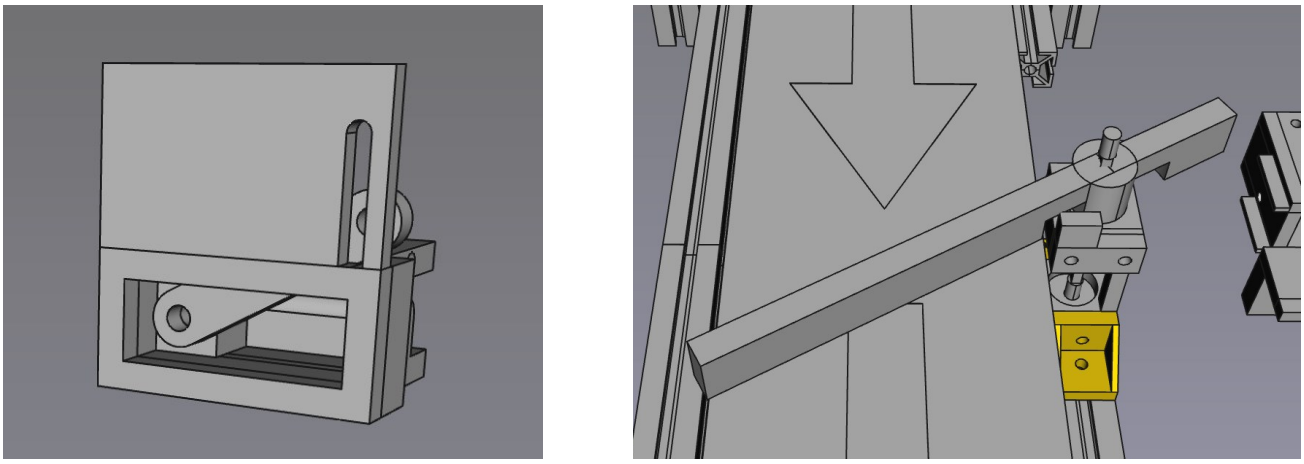


Figure 64: Linkage mechanism and old sweeper design

The bin sweeper also had several design iterations, however the main working principle was that the motor would drive a GT2 timing belt to move some other mechanism that would sweep the components, and this remained unchanged. The original design had several sweepers positioned at the bin locations, and would 'activate' like gates when the driven arm moved passed it. A flexible polyurethane rod was going to be used to allow the gates to be 'opened' by the arm, however the sweeper would then have to translate horizontal motion to vertical motion, which requires a complex mechanism using linkages as shown in Figure 64. This was then simplified to a moving sweeper arm that was angled to use the conveyor belt speed to move the components off the belt, and was effective.

4.5.2 Electronics and Wiring

The WS2812B LED strip experienced communication issues when connected to the Raspberry Pi, often not responding to commands or flickering when attempting to do so. Originally, the strip was controlled via PWM (Pulse Width Modulation) however this requires root privileges and is not conventional. The solution was to use the SPI (Serial Peripheral Interface) protocol to control the strip, which does not require root privileges, however there was still flickering issues.

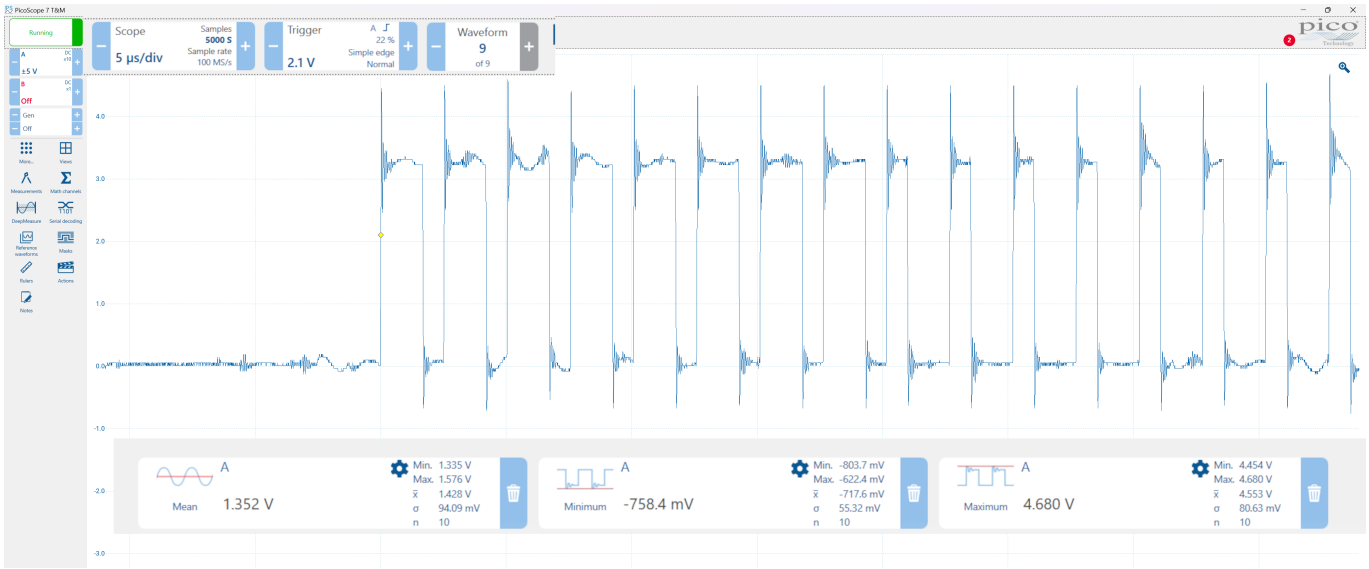


Figure 65: PicoScope signal analysis

After several attempts to debug the issue programmatically, a PicoScope [72] was used to analyse the signal, and it was found that it was severely beneath the expected 5V signal, which was causing the

flickering and general unresponsiveness of the strip. This can be seen in Figure 65 (note the image has been edited to allow for better readability), with a maximum voltage of only 4.68V. The solution was to simply use thicker wires to connect the Raspberry Pi to the PSU, which solved the issue; despite the step-down converters display 5V, the voltage drop across the wires was causing the Pi to be undervolted, which was also shown on the display of the Pi itself in the corner as as lightning bolt symbol as shown in Figure 66.

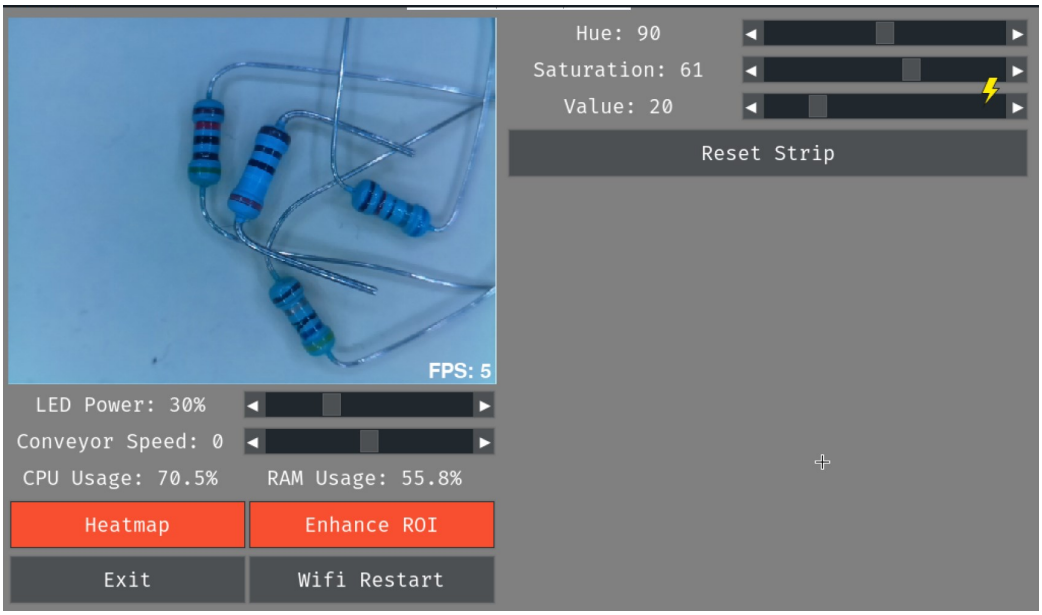


Figure 66: Pi under-voltage warning

.

4.5.3 Computer Vision

During the evaluation of the model, the confusion matrix was examined to determine the model's performance on each class. The confusion matrix is shown in Figure 67. Clearly, there is a strong diagonal line, which is a good sign as it shows that the model is correctly classifying most of the images, however interestingly there is a class that was unaccounted for during training; the background.

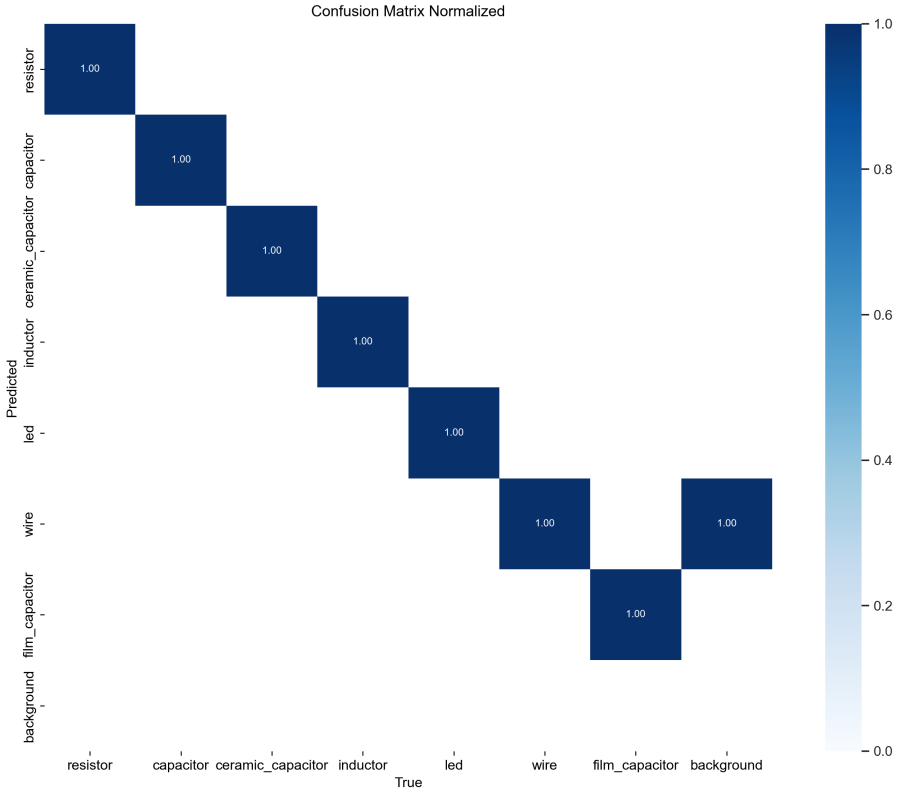


Figure 67: Normalised confusion matrix for the test set

Counterintuitively, the confusion matrix seems to imply that all background images are being classified as wires, but this is actually not what the confusion matrix is showing; it is showing that 100% of false positives are wires, due to the fact that the confusion matrix is normalised. The unnormalised confusion matrix is shown in Figure 68, where there are two false positives, and both are wires.

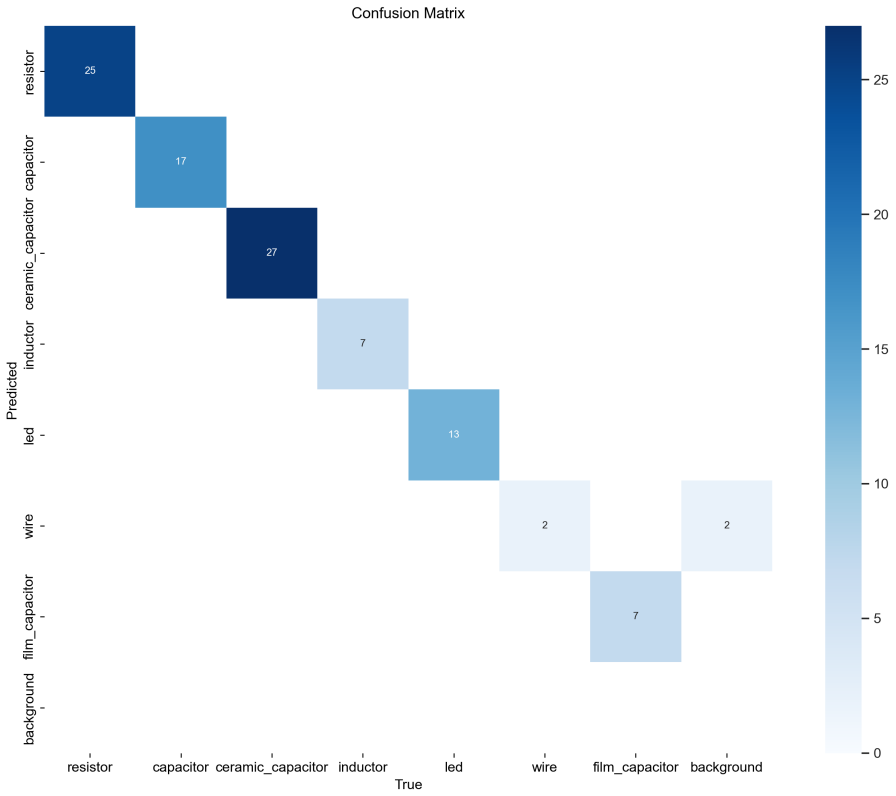


Figure 68: Unnormalised confusion matrix for the test set

However, this highlights an issue; there are no images in the dataset without a component, so the model may have learnt that there is always a component in the image. This is a negative side effect of the dataset, as the model has not learnt that there are images without components, and thus will always place a bounding box, which is clearly not ideal. On Ultralytics' documentation [73] (the creators of YOLO), they state the dataset should contain "0-10% background images to help reduce FPs" and that the COCO dataset "has 1000 background images for reference", which is "1% of the total". Luckily, this was fixed extremely easily by adding images with no labels to the dataset, and the model will learn that there are images without components.

Figure 69: Background image example

As there are ĩ000 images in the training set, 100 background images were added; however adding 100 images of the same static empty conveyor would not be very useful, so instead some images of the conveyor with a variety of objects were added to the dataset as shown in Figure 69, as well as other random images. The model was then retrained on this dataset, and the results are shown in. Naturally, the dataset annotation tool discussed in was extended to allow for the addition of background images.

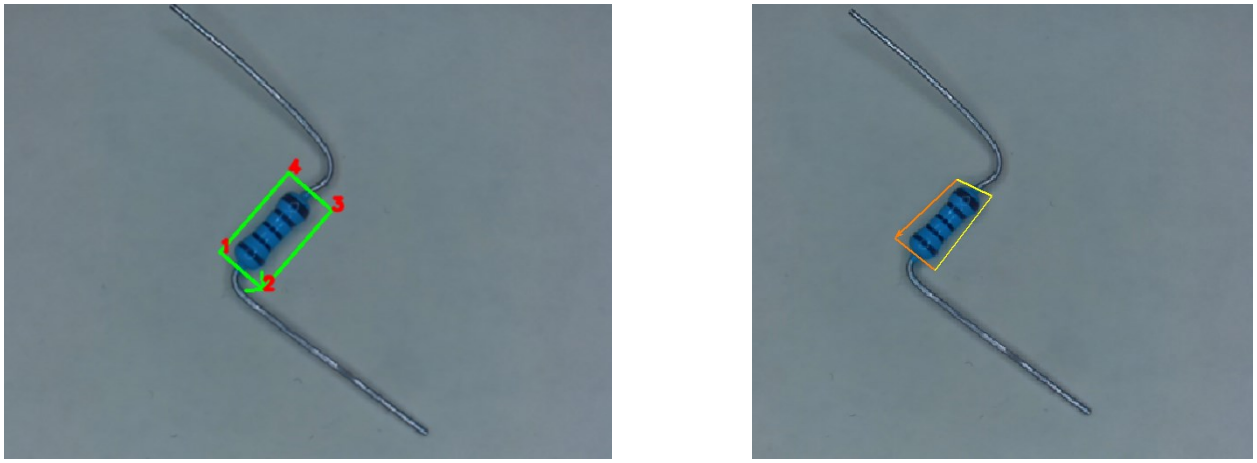


Figure 70: Annotated resistor with orientation and vertex numbers on the same resistor during inference

Also, it was discovered late into development that the OBB model does not learn the orientation of the components and encode it into the bounding box data. For instance, the line drawn between the first and second point does not indicate the orientation of the component, it is simply a line of the bounding box. This is shown more clearly in Figure 70, where the resistor is annotated, with the first line drawn is an orange arrow to indicate the resistor's orientation, but during inference, the first line is a completely

different line. At the time, this was an issue as to determine resistor values, it was necessary to know which way the resistor was facing to read the bands in the correct order. This was circumvented by adding a "stem" class for the resistor model to predict, where it would attempt to predict whether the first band of the resistor was. This is shown in Figure 61 as the large yellow box that is drawn on the base of the resistor.

5 RESULTS AND TESTING

Contents

This chapter will focus on the results of the project, and the testing that was carried out to evaluate the performance of the different subsystems of the project. The chapter will be divided into three sections: Mechanical Design, Electronics and Software, and Computer Vision.

5.1 Mechanical Design

The mechanical design was tested for stability, durability, and ease of use. During its design, it was kept in mind what components would be under stress and how they would be used in the system. Several tweaks were made to some designs to increase their stability and durability, so that they are fit for purpose, as described in Subsection 4.5.

The system was observed while in operation and did not exhibit any signs of instability or wobbling, and the system was able to sort components effectively. Where points of failure were identified, they would be replaced with a more robust design, such as the LCD cover, which was braced to prevent wobbling, and increases in thickness were made to certain parts to resist mechanical stress. For example all mounts that attach to aluminium extrusion were increased in thickness from 3mm to 5mm to prevent them from cracking when a screw was tightened on top of them, a common failure from previous iterations.

Additionally, mechanical failure was prevented by using parts such as bearings to facilitate the movement of the rollers and prevent friction from wearing down the FDM printed parts. Screws and heat-set inserts were used to attach parts to other parts, preventing threads from being stripped with repeated use. The system was also designed to be easily assembled and disassembled, with the use of T-slot nuts and bolts to attach parts to the aluminium extrusion, and screws to attach parts to other parts. This allows for easy maintenance and repair of the system, as parts can be easily replaced if they fail. The mechanical design of the system was successful in achieving its goals, and provides a strong foundation for the rest of the system.

During testing, it was found that the movement of the system could keep up with the vision system, which requires frequent pauses to take clear pictures and perform inference. The system is bottlenecked by this speed.

5.2 Electronics and Software

As mentioned in Section 3, the system is designed to be modular to facilitate easy development and debugging, with each system being able to be tested independently. Each part of the software, the

System Controller, LCD UI, Vision Handler, Camera Feed, etc are all able to be run independently by simply running the Python script that contains the code for that part of the system.

This allows for easy debugging and development of the system, as each part can be tested independently, and any issues can be isolated to a single part of the system. This makes it easier to identify and fix issues, as the issue is likely to be in the part of the system that is being tested. The system was tested by running each part of the system independently, and then running the system as a whole to ensure that all parts of the system work together.

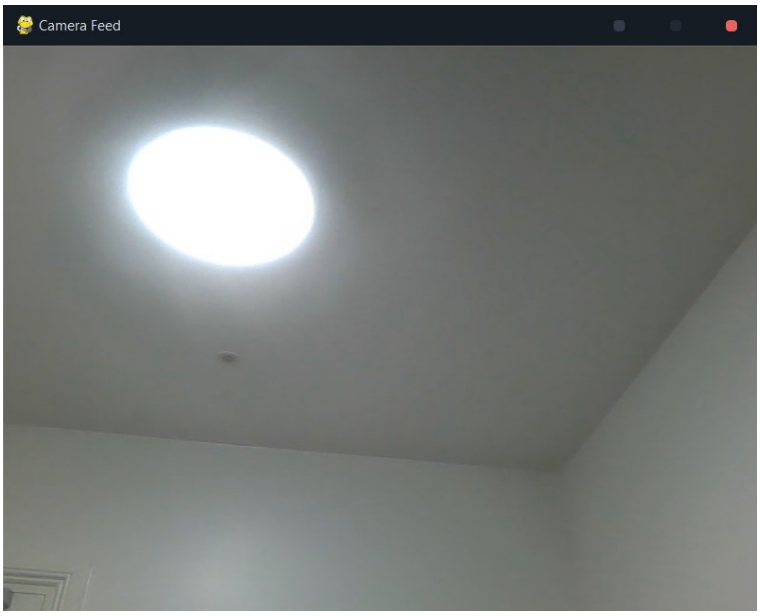


Figure 71: Camera Feed in Pygame running standalone

For example, the Camera Feed is shown in Figure 72, where the camera feed is displayed in a Pygame window completely on its own without any other parts of the system running. It is responsible for handling when the camera disconnects, so that no other part of the system is affected.

Figure 72: Vision Handler running standalone

Likewise, the Vision Handler is shown in Figure 72, where the Vision Handler is running standalone, and is responsible for handling the inference of the component detection model. It is able to operate without the LCD UI, and is able to handle the inference of the model without any other parts of the system running. The Vision Handler detects that it is being run standalone, and enables keypresses for controlling it, which are detailed in the following table:

Key	Action
i	Inference once
c	Toggle continuous inference
f	Toggle force image
v	Toggle capture VNC
r	Select random file

Table 11: Vision Handler standalone keypresses

Unit testing of all parts of the system was carried out to ensure that each part of the system works as expected as shown in the above examples. To test the system as a whole, a profiler was used to examine the system's performance, and to identify any bottlenecks in the system.

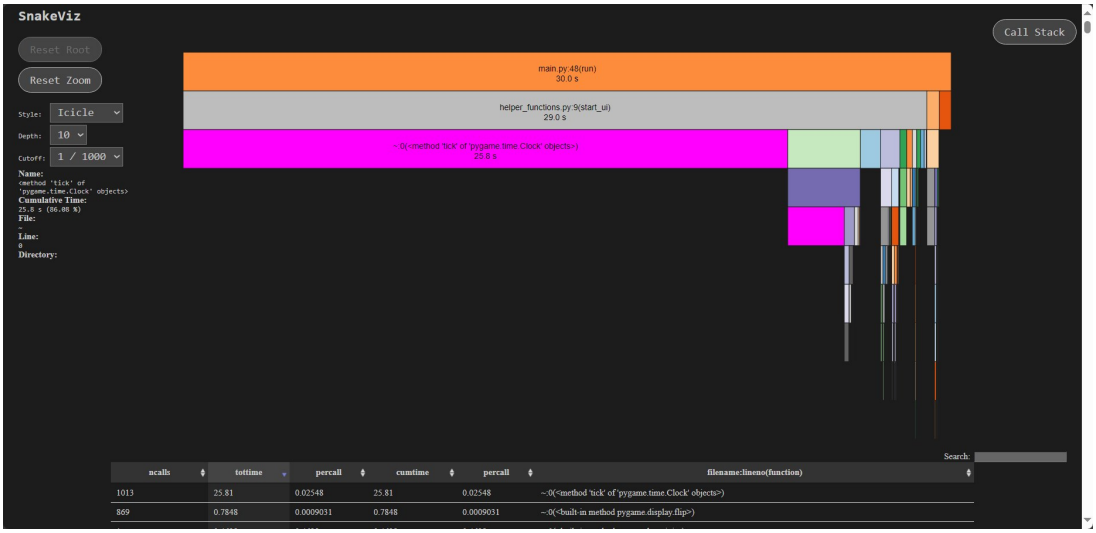


Figure 73: Profiler output for the main process

SnakeViz [74] and cProfiler were used to profile the main process of the system, and the output is shown in Figure 73. Highlighted in purple is the `Pygame.time.Clock tick` function, which is used to control the frame rate of the system. This function simply waits until the next frame is ready to be displayed, and the fact that it takes up 86% of the time in the main process is a good indication that the LCD UI is responsive, as it is not being held up by other parts of the system.

The other parts of the system are shown in green, and take up a small amount of time in the main process, which is a good indication that the system is running efficiently. The second and third largest functions are the `pygame.display.flip` function and `pygame_gui`'s `ui_manager.update` function, both of which are necessary for `pygame` to display the UI, and are expected to take up a large amount of time in the main process. The largest function that is not responsible for the UI is the Vision Handler's `get_frame` function, which takes a miniscule 1.32% of the time in the main process, a good indication that the Vision Handler is running efficiently and using multiprocessing effectively to perform inference.

5.3 Computer Vision

This section will examine the Computer Vision system, as discussed in Subsection 3.4 and implemented in Subsection 4.3.

5.3.1 Component Detection Model

The Component Detection model was trained on a dataset of 1076 images, split into a 70-20-10 split for training, validation, and testing respectively. The dataset was augmented using the augmentations discussed in Subsubsection 3.4.1. The model was trained using the training parameters shown in Table 8.

Figure 74: IoU Example [75]

As discussed in Subsubsection 3.4.2, mAP^{50} is a standard metric used to evaluate object detection models. $mAP^{50\text{-}95}$ is a more rigorous metric that aggregates the mean average precision across all classes at different Intersection over Union (IoU) thresholds; in mAP^{50}, an object is considered as detected if the IoU is greater than 0.5 (the predicted bounding box overlaps the ground truth bounding box by at least 50%), whereas in $mAP^{50\text{-}95}$, the mAP score is aggregated across different IoU thresholds from 0.5 to 0.95 in steps of 0.05. This can be seen more clearly Figure 74, where the second bounding box would be detected in mAP^{50}, but not in mAP^{75}.

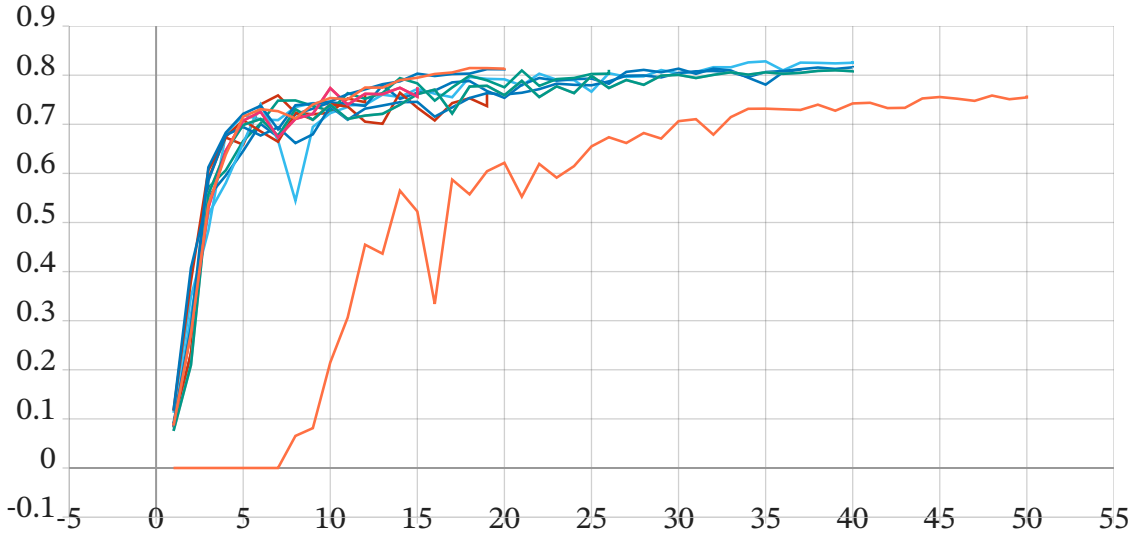


Figure 75: TensorBoard graphing: all runs $mAP^{50\text{-}95}$ against epochs

In Figure 75, the $mAP^{50\text{-}95}$ is shown for all training runs of the model with slightly different training

parameters. It can be seen that the models tend to converge around 20 epochs, and there is a ceiling at around 80% mAP^{50-95}. This is likely due to the small dataset size, and it is made even smaller due to the use of a test set. For deployment, the training set will absorb the test set to increase the size of the dataset and improve the model's performance, however for discussion it is necessary to keep a test set separate to objectively measure the model's performance.

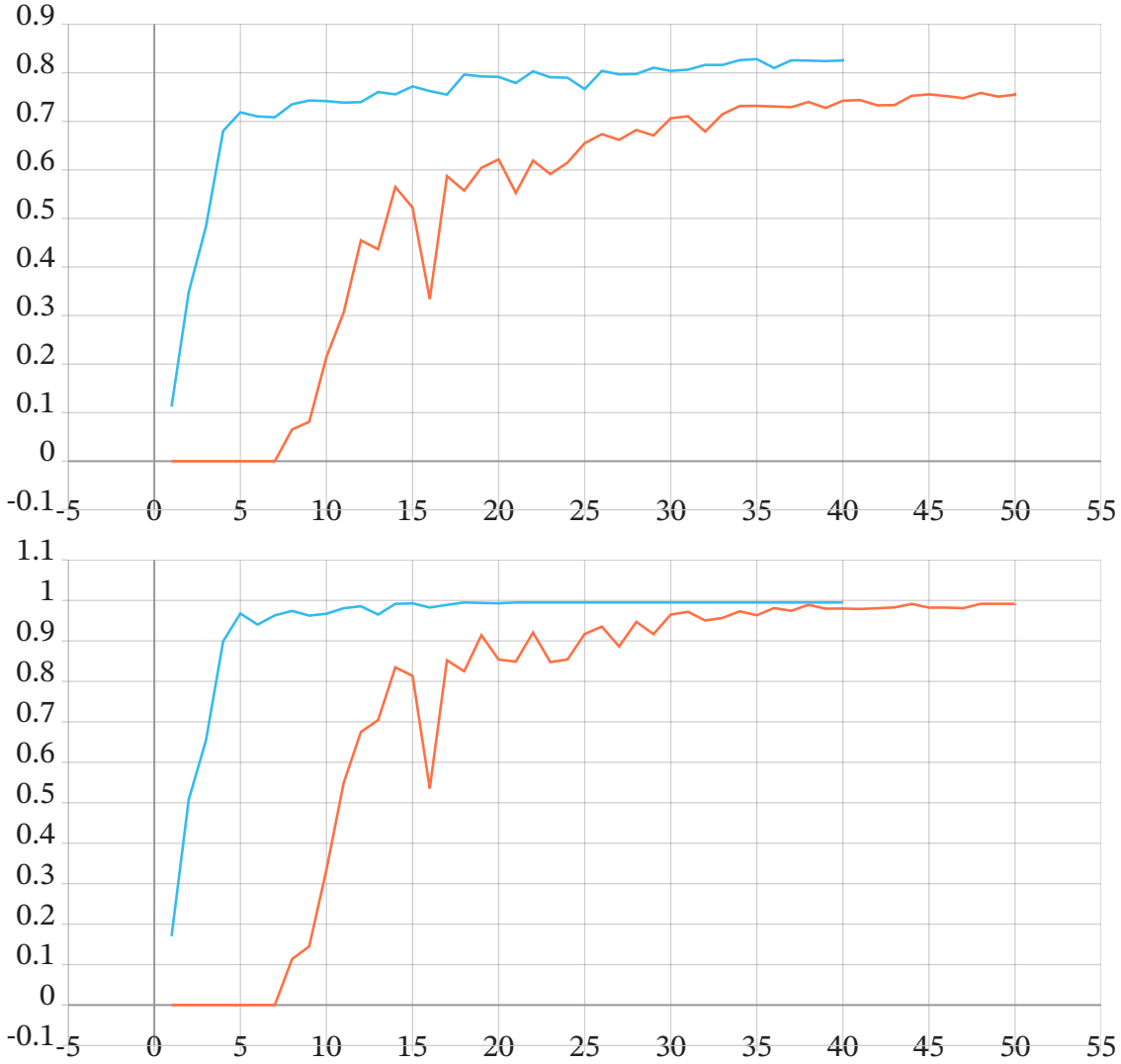


Figure 76: Final Model (blue) and non-pretrained model (orange) mAP^{50-95} and mAP^{50} scores

As shown in Figure 76, the final YOLO-OBB model (blue) achieved an mAP^{50-95} of 82.8% and an incredibly high mAP^{50} of 99.5%. This is an extremely impressive score given the relatively small size of the dataset (and made even smaller due to the need of the test set), which is truly a testament to YOLOv8's capabilities.

An experiment was run to see how the model would perform if the model was **not** pretrained on DOTOv1 [48] given the low training time to see how much of an impact the pretraining had on the model's performance. The model, shown in orange, clearly takes much longer to converge; where the original model reaches 99.5% mAP^{50} for the first time in 18 epochs, the non-pretrained model had an mAP^{50} of 82.5%, and takes 40 epochs to reach 98%. By the time the pretrained model achieves an mAP^{50-95} of at least 80% on epoch 26, the non-pretrained model has an mAP^{50-95} of 67.4%. This is a substantial difference, and it is clear that pretraining on a large dataset is crucial to the model's performance and training time. When the model is pretrained on a large dataset, the model is able to learn generalisable, basic features such as edge and corner detection, which are common in most images. When the model is now trained on a new dataset, it does not need to relearn these basic features, and

can instead focus on learning the more complex features of the new dataset. This is known as transfer learning, and it is a common technique used in deep learning to improve the performance of models on new datasets.

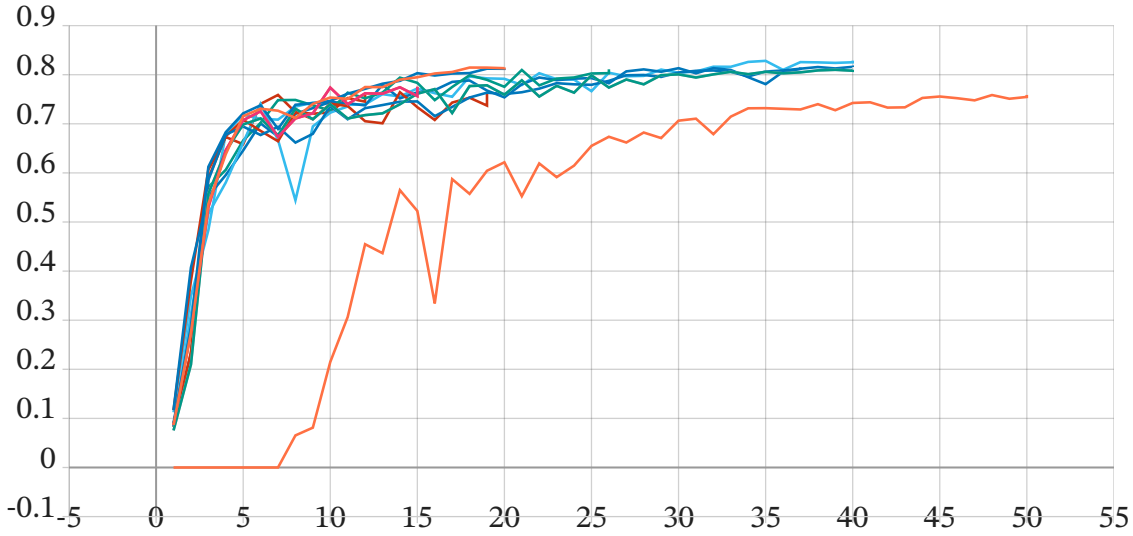


Figure 77: All training runs $mAP^{50\text{-}95}$ against epochs

For completeness, the $mAP^{50\text{-}95}$ for all training runs is shown in Figure 77, where the orange non-pretrained model is clearly lagging behind all other training runs.

To properly evaluate a model's performance, it is important to consider the model's performance on a test set. As described above, 10% of the dataset was split for this, giving 98 + 10 background images for the test set.

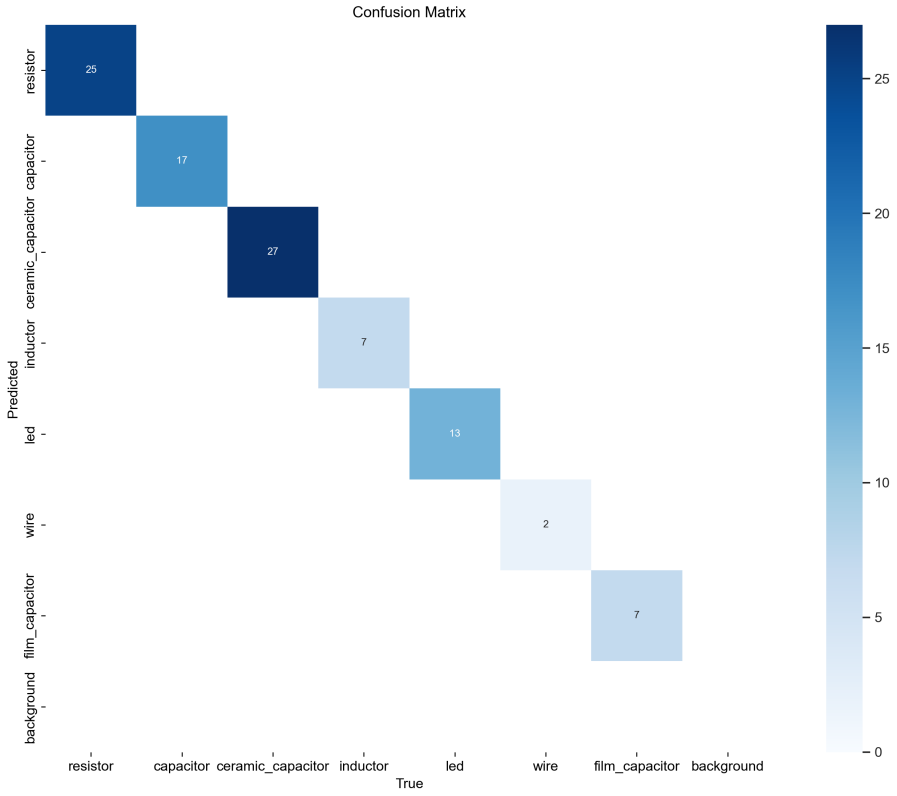


Figure 78: Unnormalised confusion matrix for the test set

In Figure 78, the unnormalised confusion matrix is shown, with only diagonal entries, which shows

100% accuracy on the test set. This is a very good sign, however it is important to remember that the dataset is very small, so the validation and test set has then been used to evaluate the model, as shown in Figure 79. Although the model does not learn on the validation set, it still has seen the validation set during training which may cause the model to learn features that are specific to the validation set, which is why the test set is used to evaluate the model's performance. However, showing the confusion matrix for the validation and test set will still be useful to gauge the model's real-world performance.

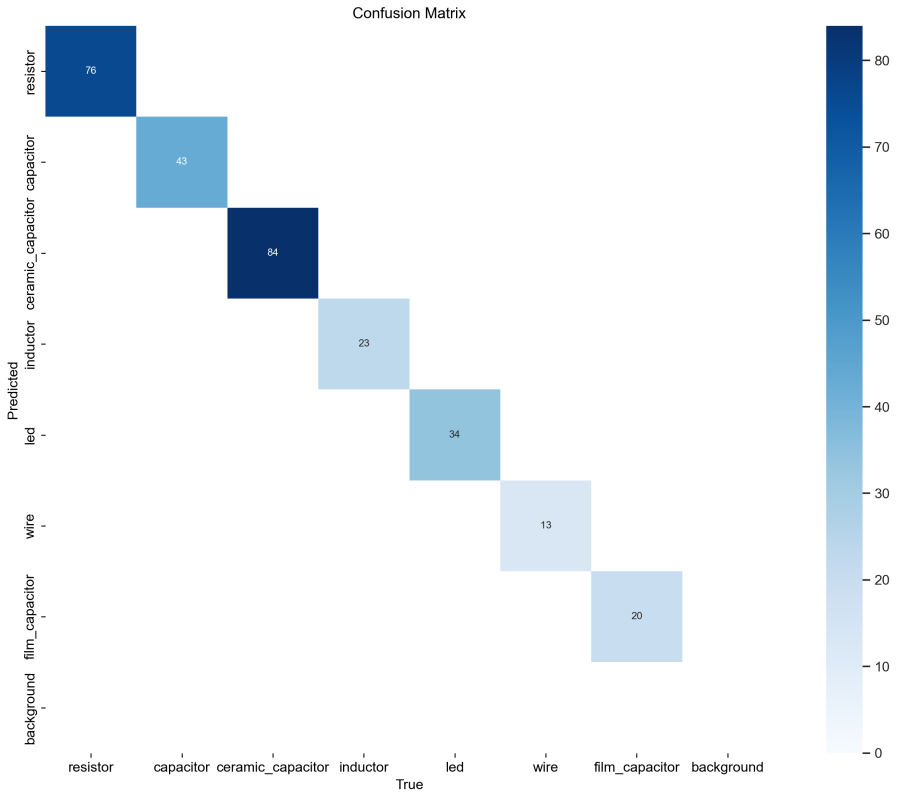


Figure 79: Unnormalised confusion matrix for the validation and test set

This again is an exceptional result with no false positives or false negatives, and an $mAP^{50\text{-}95}$ of 82.8%. The model is able to detect all components in the test set, and is able to classify them correctly. The model is able to generalise well to the test set, and is able to detect components that it has not seen before. In terms of the requirements of the project, the model is able to classify components with a high degree of accuracy.

A mosaic of inferences on the test set is shown in Appendix Figure 85 for a visual representation of the model's performance.

5.3.2 Resistor Value Detection Model

As discussed in Subsubsection 4.3.4, the resistor value detection model is a standard YOLOn model trained on images produced by running inference on all resistor images with the component detection model and then cropping the bounding boxes. This meant that there were only 259 images in the dataset, which was then split 80-20 for training and validation; a test set was not used as the dataset was too small. This left only 208 images for training, and 51 images for validation.

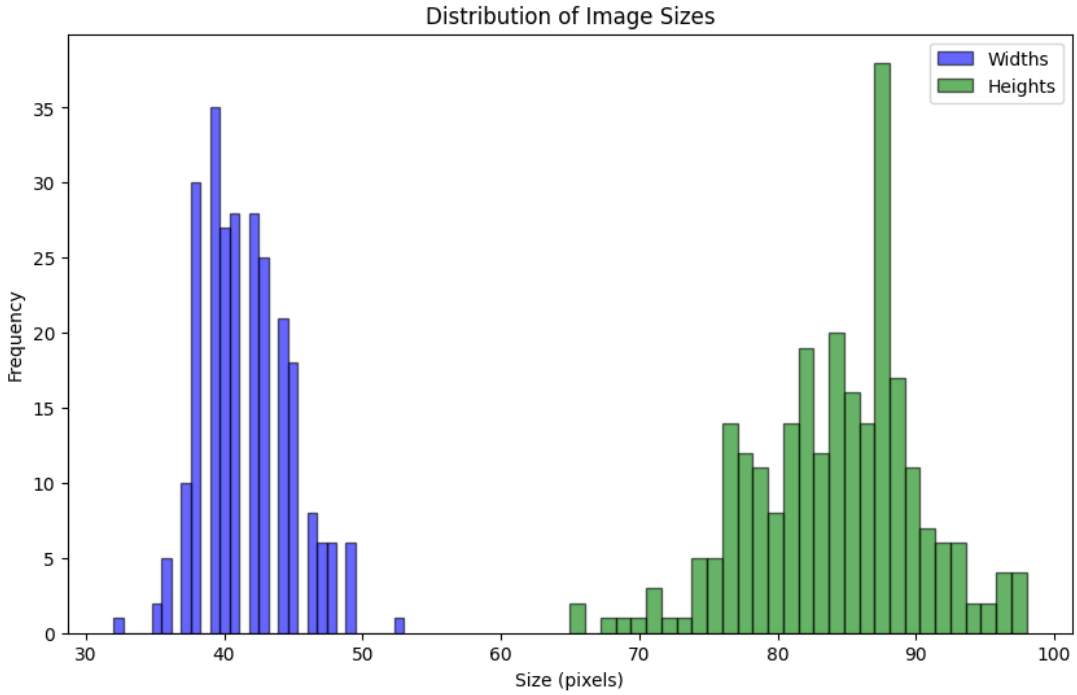


Figure 80: Distribution of image widths and heights

As the images are generated from the component detection model, the image size is no longer fixed, but instead varies based on the size of the bounding box. The distribution of image sizes is shown in Figure 80, where the image sizes vary from 32x65 to 53x98. Due to this, a training parameter was set that would resize all images to a larger size of 128x128, to ensure that the model always works with the same size images.

Class	**Images**	**Instances**	**Precision**	**Recall**	**mAP^{50}**	**mAP$^{50\text{-}95}$**
all	51	326	0.731	0.774	0.837	0.457
black	51	61	0.839	0.967	0.956	0.488
brown	47	91	0.886	0.989	0.981	0.53
red	31	31	0.859	0.968	0.98	0.577
orange	17	17	0.64	0.882	0.837	0.454
yellow	19	19	0.919	1	0.993	0.599
green	12	12	0.852	0.75	0.903	0.417
blue	5	5	0.577	0.4	0.567	0.275
violet	8	8	0.769	1	0.995	0.654
grey	2	2	0	0	0.199	0.0562
white	10	10	0.527	0.9	0.81	0.348
gold	13	13	0.87	0.769	0.779	0.382
silver	6	6	1	0.43	0.885	0.593
stem	51	51	0.763	1	0.995	0.572

Table 12: Resistor value detection model metrics

However, due to the small image sizes and the small dataset, the model only achieved an mAP$^{50\text{-}95}$ of 45.7% and an mAP^{50} of 83.7% as shown in Table 12. As we are more concerned about classifiying the resistor values correctly rather than the exact bounding box location, the mAP^{50} is more important in this context. The model is able to classify some bands with high accuracy, such as brown, yellow, and violet, but struggles with others, such as blue and grey. After observing these results, the generated dataset was inspected more closely.

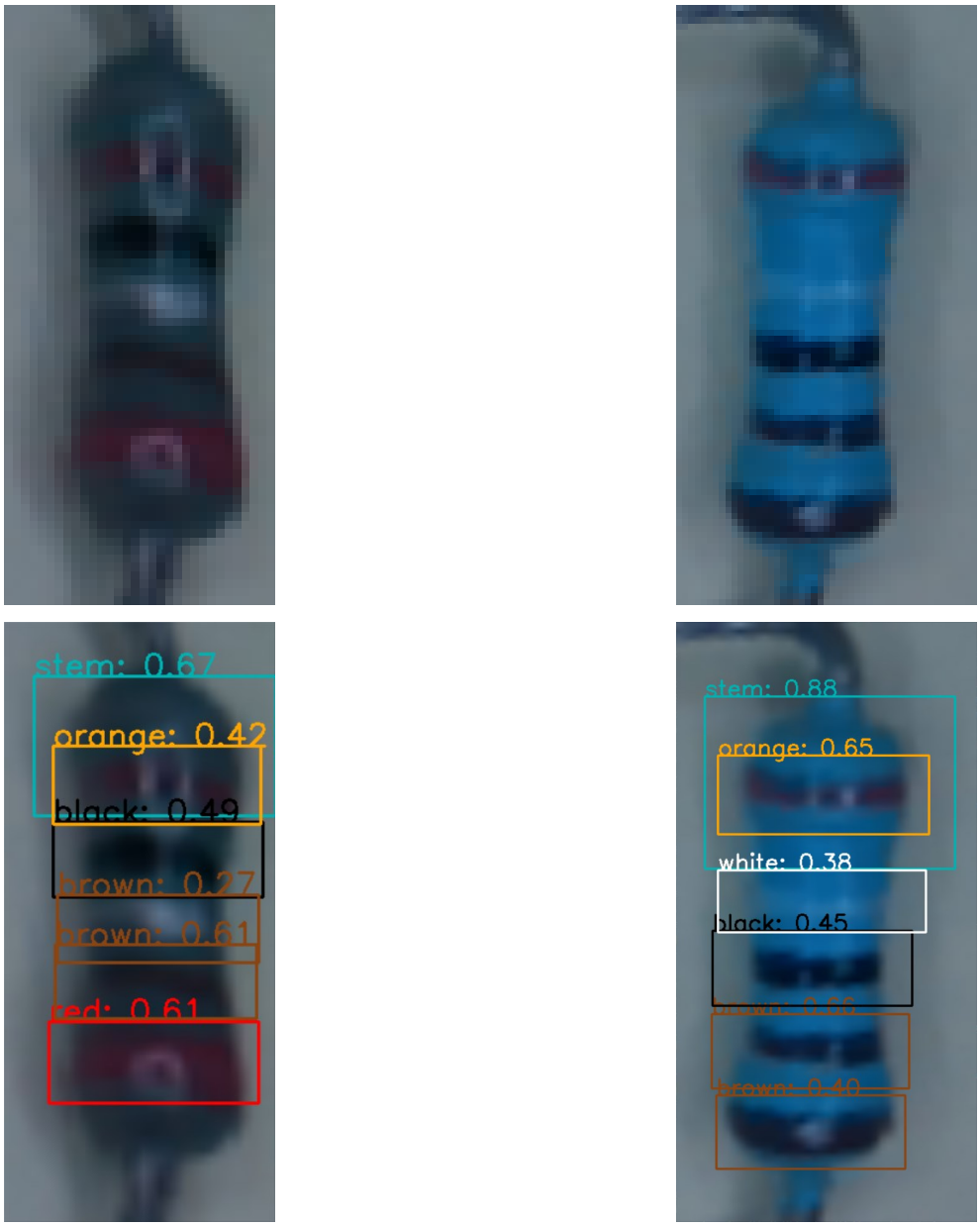


Figure 81: Smallest and largest images in the resistor value dataset with inference results

In Figure 81, the smallest image in the dataset is on the left and the largest on the right. The resolution difference is obvious with the left images exhibiting more pixelation. For the low resolution image, the model was able to identify the red, brown, black and orange band as well as the stem, but fails to correctly determine the silver and grey bands. To human eye, the silver band actually seems white, and the grey band is hardly visible (it is at the top of the resistor). This shows that maybe the dataset itself is not very good, and the model is not at fault — observing a full 640x480 image of the resistor as seen in Figure 82, the resistor is very small and in terms of pixels, only takes up 0.74% of the pixels in the original image.

Figure 82: Raw image from camera of the smallest resistor in the dataset

This is a very small amount of pixels, and it is clear that the model is struggling to classify the bands due to the low resolution of the image, and clearly performs better on the larger image. This would be fixed by moving the camera closer to the belt to increase the relative size of the components of the image at the cost of a smaller field of view, and then a new dataset would have to be generated. Due to time constraints, this is not possible at the time of the report being written, but the model itself is performing well given the constraints of the dataset.

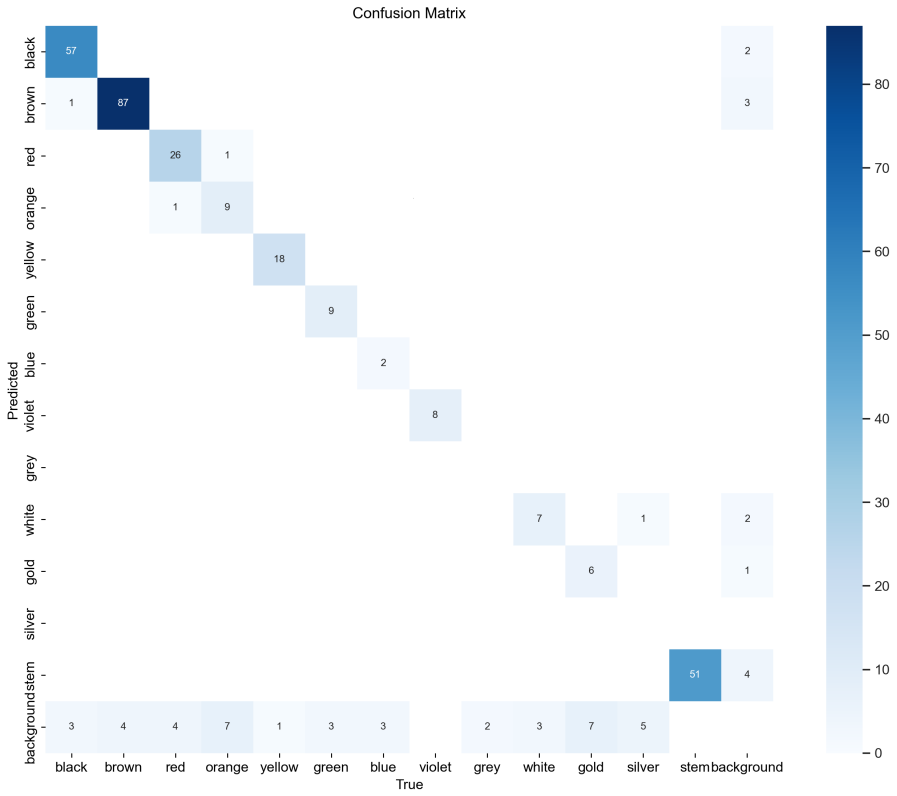


Figure 83: Confusion matrix for the resistor value detection model

For completeness, the confusion matrix of the resistor detection model was shown in Figure 83, and shows a strong diagonal but with many false negatives; multiple bands are not being identified at all, as shown by the row on the bottom. There is also severe class imbalance, with 91 total brown bands across the test set but only 2 are gray and 5 are blue.

Objectively, the model is performing well given the constraints of the dataset, and is able to classify resistor values with a high degree of accuracy. However, the performance of the model is not adequate enough to be deployed in the system as of the time of writing, and further work is needed to improve the model's to an acceptable standard.

5.3.3 Inference Latency

The inference latency of the model is crucial to the system's performance, as the system must be able to classify components in real-time. The Ultralytics and DeepSparse Python libraries come with their own benchmarking capabilities, and they were used to measure the model's performance. The benchmarks were performed on the Raspberry Pi 4, as this is the intended deployment platform for the system. The benchmarks were performed on the YOLO-OBB model, as it more complex than the standard YOLO model used for the resistor value classification model, and thus should be on the lower end of the performance spectrum.

```
YOLOv8n-obb summary (fused): 187 layers, 3079364 parameters, 0 gradients, 8.3 GFLOPs
val: Scanning /home/dietpi/MEngFYP/datasets/full/current/labels/test.cache... 98 images, 7
 ↪  backgrounds, 0 corrupt: 100%|----------| 105/105 [00:00<?, ?i
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):
                  ↪  100%|%|----------| 105/105 [03:10<00:00,  1.81s/it]
                   all        105         98      0.971          1      0.995      0.793
       {0: 'resistor'}         25         25      0.989          1      0.995      0.819
      {1: 'capacitor'}         17         17      0.983          1      0.995       0.85
    {2: 'ceramic_cap'}         27         27      0.957          1      0.995      0.784
      {3: 'inductors'}          7          7      0.975          1      0.995      0.882
           {7: 'leds'}         13         13      0.981          1      0.995      0.882
           {8: 'wire'}          2          2      0.966          1      0.995      0.499
      {10: 'film_cap'}          7          7      0.949          1      0.995      0.837
Speed: 5.5ms preprocess, 1762.0ms inference, 0.0ms loss, 5.3ms postprocess per image
Results saved to runs/obb/val6
```

Code Snippet 7: YOLO model inference latency on Raspberry Pi 4

As shown in Code Snippet 7, the standard YOLO model takes 1762ms to performance inference; this translates to 0.53FPS on the Pi, which is not suitable for real-time object detection.

```
'engine': "ORTEngine
onnx_file_path: ./src/vision/models/final/classifier.onnx  batch_size: 1...
'benchmark_result': {'scenario': 'singlestream', 'items_per_sec': 0.9651499692166781,
 ↪  'seconds_ran': 60.094287778999956, 'iterations': 58, 'median': 727.9371934999972, 'mean':
 ↪  1036.078450206897, 'std': 592.0563771426215, '25.0%': 552.4718009999958, '50.0%':
 ↪  727.9371934999972, '75.0%': 1564.6943927500274, '90.0%': 2038.2221947999483, '95.0%':
 ↪  2091.3465085999687, '99.0%': 2292.9438349999714, '99.9%': 2342.975698999978},
 ↪  'fraction_of_supported_ops': None
```

Code Snippet 8: Onnxruntime model inference latency on Raspberry Pi 4

Code Snippet 8 shows the inference latency of the model using the Onnxruntime library [76], which is a popular library for running ONNX models. ONNX (Open Neural Network Exchange) is an open-source format for AI models, and is supported by a wide range of libraries and frameworks. The model takes 1036ms to perform inference, which translates to 0.96FPS on the Pi, an improvement of 81.1% over the standard YOLO model. This is a significant improvement, however it is still not suitable for real-time object detection.

```
    'engine': 'deepsparse.engine.Engine:
    onnx_file_path: ./src/vision/models/final/classifier.onnx batch_size: 1...
    'benchmark_result': {'scenario': 'singlestream', 'items_per_sec': 1.6153408297780283,
     ↪  'seconds_ran': 60.04924670499986, 'iterations': 97, 'median': 414.9715189998915, 'mean':
     ↪  619.0319354845434, 'std': 360.0698332691641, '25.0%': 404.8420740000438, '50.0%':
     ↪  414.9715189998915, '75.0%': 714.750333000211, '90.0%': 1199.7598698000731, '95.0%':
     ↪  1420.3843886001316, '99.0%': 1713.9762652400166, '99.9%': 1744.4832248240787},
     ↪  'fraction_of_supported_ops': 0.9926
```

Code Snippet 9: Inference latency of the model using the DeepSparse library

The DeepSparse library [16] takes an ONNX model and uses its own engine to perform infernece at higher speeds. Using the library, the model is able to perform inference at 619ms, which translates to 1.62FPS on the Pi, an improvement of 55.2% over the standard Onnxruntime model, and an improvement of 203.8% over the standard YOLO model. This is a significant improvement, and the model is now able to perform inference at a speed that is suitable for real-time object detection.

6 EVALUATION

Contents

It is important to evaluate the system to determine if it meets the requirements set out in Section 1 from a high level, and to determine if the individual components meet their requirements.

6.1 Mechanical System

As discussed in Subsection 5.1, the mechanical design enables the system to move components into designated bins for sorting. The system is able to move components from the conveyor belt into the bins, and the bins are able to be removed and replaced with ease. It is also easily maintained and cleaned. Standard parts were used so the system remains cost-effective and easy to repair.

The main mechanical system is the conveyor belt, and the sweeper and bin system. The conveyor's function requirements outlined in Subsubsection 3.3.1 were met, as the conveyor is able to move components from the input to the sweeper, and provide mechanisms for tensioning and detaching the conveyor belt. It allows the rollers to turn freely though the use of bearings, facilitating smooth operation.

Likewise, the sweeper and bin system design allows the system to move components from the conveyor to the bins, and the bins are able to be removed and replaced with ease. The sweeper is able to move components off the conveyor belt and into the bins, meeting the functional requirements of needing a sorting system for the components.

However, it is important to note that the system can only sort one component at a time, which is a limitation of the system. This is due to the fact that the sorting arm requires the movement of the conveyor belt to move the components off the belt, rather than removing components from the conveyor belt directly. This is a limitation of the system but a difficult problem to solve, as the system would require a more complex mechanism to remove components from the conveyor belt directly, such as a vacuum system which may not be feasible, as outlined in Subsubsection 3.3.2.

The mechanical system is also not fully autonomous — it requires a human operator to place components on the conveyor belt. This is a limitation of the system, as the system was designed to be a proof of concept, and not a fully autonomous system. However, the system could be made autonomous with the addition of a feeder system, as discussed in Section 2, which would allow the system to feed components into the system one by one for sorting. This is an incredibly difficult design and engineering challenge, as it requires a system that can successfully detangle and feed the components into the system, and this was not feasible to implement in the time frame of this project.

Overall, for the mechanical system, the system meets the functional requirements set out in Section 1, with the exception of the system being fully autonomous.

6.2 Software

As explored and thoroughly discussed in Subsection 5.2, the software system is able to operate without significant lag to the user regardless of the load on the system; it successfully utilises Python's `multiprocessing` library to run the computer vision system and the conveyor system concurrently, and supports an extensive range of features for easy debugging and maintenance.

The system is also able to gracefully handle errors and exceptions, and provides a user-friendly interface for the user to interact with the system, fulfilling the requirements set out in Section 1.

6.3 Computer Vision

As discussed in Subsection 5.3, the computer vision system is able to identify components with a high degree of accuracy, with an $mAP^{50\text{-}95}$ of 82.8%, with a low inference time on the Pi of 619ms per image, allowing the system to identify components in real-time. The system is also able to identify components in a variety of orientations, which meets the requirements set out in Section 1.

While the computer vision system is able to identify components with a high degree of accuracy, it does not currently have the ability to perform value identification on components as of the time of writing due to time constraints.

Due to this limitation, in its current state it cannot be a drop-in replacement for the manual sorting of components that the technicians currently perform. The system exists as a proof of concept, and not a fully autonomous system, but provides a strong foundation for future work to build upon.

7 CONCLUSION

Contents

This project was an incredibly complex and challenging project that required a lot of research and development across several engineering disciplines to complete. The project required extensive programming skill to complete to ensure system stability and coherency, as well as advanced CAD skills to be able to fully realise the physical parts of the system.

The project also demanded knowledge of deep-learning to properly identify and train the appropriate model for the task of component identification, as well as sufficient knowledge of electronics to be able to design and implement the electronics system. However, all of these skills were developed and honed throughout the project, and the project was completed largely successfully, but not without its challenges, and it is not without its limitations.

Unfortunately, due to time constraints, the system was not able to be fully autonomous, as it required a human operator to place the components on the conveyor belt and does not yet have a universal component value detection system.

7.1 Future Works

Due to time constraints, there were some features that were not implemented in the final system, but could be implemented in future iterations of the system. One of these features is a feeder system as described in Section 2 in order to autonomously feed the components into the system one by one for sorting. This is an incredibly difficult design and engineering challenge, as it requires a system that can successfully detangle and feed the components into the system, and this was not feasible to implement in the time frame of this project.

Another improvement to the system would be to reduce the inference latency on the trained models by defining custom recipes for the SparseML and DeepSparse libraries, as discussed in Subsubsection 4.3.5, to reduce the model size and increase the inference speed. This would allow for the system to be able to process components faster, and thus sort them faster. Alternatively, an improvement could instead be to use a more powerful device to run the models, such as a Jetson Nano as explored in Subsection 3.2, or even offload the inference to a cloud service, which would allow for the system to be able to process components faster, provided that the internet connection is stable.

Additionally, to perform value identification, deep-learning models capable of OCR could be used to read the values of the components, as discussed in Subsubsection 4.3.3, and the resistor model should be retrained with a higher resolution dataset to improve the accuracy of the model. However, there is still the challenge of needing to orient the components such that their values can be read, which is a difficult problem to solve.

Additionally, again due to time constraints, a comprehensive performance evaluation of the system was not conducted, as this would require the entire system be assembled and in working order. This would involve testing the system with a variety of components to determine the system's accuracy and efficiency, and determine its throughput, and then compare against a human operator to determine if the system is more efficient than a human operator.

Finally, the component identification model should be trained with more component classes to improve

its breadth of identification.

8 REFERENCES

[1] U. of Bristol, "Leaf green lab certification," https://www.imperial.ac.uk/sustainable-imperial/resource-management/energy-use/laboratory-efficiency-assessment-framework-leaf/#:~:text=In%202021%2D22%2C%20we%20awarded,leaving%20procedures%20were%20in%20place, n.d, accessed: 06-02-2024.

[2] A. I. Dhenge, Nāgpur, and A. S. Khobragade, "Mechanical nut-bolt sorting using principle component analysis and artificial neural network," in *n.a*, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:201742258

[3] Y. Xu, G. Yang, J. Luo, J. He, and C. Huang, "An electronic component recognition algorithm based on deep learning with a faster squeezenet," *Mathematical Problems in Engineering*, vol. 2020, p. 2940286, 2020. [Online]. Available: https://doi.org/10.1155/2020/2940286

[4] P. Chand and S. Lal, "Vision-based detection and classification of used electronic parts," *Sensors*, vol. 22, no. 23, 2022. [Online]. Available: https://www.mdpi.com/1424-8220/22/23/9079

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale," 2021.

[6] "A guide to principal component analysis (pca) for machine learning," 2022, accessed: 08-06-2024. [Online]. Available: https://www.keboola.com/blog/pca-machine-learning

[7] M. Muminovic and E. Sokic, "Automatic segmentation and classification of resistors in digital images," in *2019 XXVII International Conference on Information, Communication and Automation Technologies (ICAT)*, 2019, pp. 1–6.

[8] Y. Li, G. Yuan, Y. Wen, J. Hu, G. Evangelidis, S. Tulyakov, Y. Wang, and J. Ren, "Efficientformer: Vision transformers at mobilenet speed," 2022.

[9] D. Ghimire, D. Kil, and S.-h. Kim, "A survey on efficient convolutional neural networks and hardware acceleration," *Electronics*, vol. 11, no. 6, 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/6/945

[10] R. P. Foundation, "Raspberry pi 4 model b," https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/, n.d, accessed: 19-01-2024.

[11] J. Terven and D.-M. Cordova-Esparza, "A comprehensive review of yolo: From yolov1 to yolov8 and beyond," *n.j*, 04 2023.

[12] C. Guo, X. ling Lv, Y. Zhang, and M. lu Zhang, "Improved yolov4-tiny network for real-time electronic component detection," *Scientific Reports*, vol. 11, no. 1, p. 22744, 2021. [Online]. Available: https://doi.org/10.1038/s41598-021-02225-y

[13] P. Sismananda, M. Abdurohman, and A. G. Putrada, "Performance comparison of yolo-lite and yolov3 using raspberry pi and motioneyeos," in *2020 8th International Conference on Information and Communication Technology (ICoICT)*, 2020, pp. 1–7.

[14] N. Magic, "Sparseml," https://github.com/neuralmagic/sparseml, n.d.

[15] ——, "Yolov8 detection 10x faster with deepsparse—over 500 fps on a cpu," https://neuralmagic.com/blog/yolov8-detection-10x-faster-with-deepsparse-500-fps-on-a-cpu/, n.d.

[16] ——, "Deepsparse," https://github.com/neuralmagic/deepsparse, n.d.

[17] X. Yang, Z. Song, I. King, and Z. Xu, "A survey on deep semi-supervised learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 9, p. 8934–8954, Sep. 2023. [Online]. Available: http://dx.doi.org/10.1109/TKDE.2022.3220219

[18] E. Quilloy, C. Delfin, and M. Pepito, "Single-line automated sorter using mechatronics and machine vision system for philippine table eggs," *African Journal of Agricultural Research*, vol. 13, pp. 918–926, 04 2018.

[19] S. Kilikevičius and A. Fedaravičius, "Vibrational transportation on a platform subjected to sinusoidal displacement cycles employing dry friction control," *Sensors*, vol. 21, no. 21, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/21/7280

[20] M. C. Abe, G. A. Gelladuga, C. J. Mendoza, J. M. Natavio, J. S. Zabala, and E. C. R. Lopez, "Pneumatic conveying technology: Recent advances and future outlook," *Engineering Proceedings*, vol. 56, no. 1, 2023. [Online]. Available: https://www.mdpi.com/2673-4591/56/1/205

[21] F. University, "Vibratory feeder basics," https://www.youtube.com/watch?v=m27oD1wfQ0Y, n.d, accessed: 06-02-2024.

[22] L. Nam, N. Mui, and D. Tu, "A method to design vibratory bowl feeder by using fem modal analysis," *Vietnam Journal of Science and Technology*, vol. 57, p. 102, 02 2019.

[23] G. Reinhart and M. Loy, "Design of a modular feeder for optimal operating performance," *CIRP Journal of Manufacturing Science and Technology*, vol. 3, no. 3, pp. 191–195, 2010. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1755581710000908

[24] R. Silversides, J. S. Dai, and L. Seneviratne, "Force Analysis of a Vibratory Bowl Feeder for Automatic Assembly," *Journal of Mechanical Design*, vol. 127, no. 4, pp. 637–645, 08 2004. [Online]. Available: https://doi.org/10.1115/1.1897407

[25] D. o. M. E. Zhengyang Zhang, MIT, *Design and Development of an Automated Sorting and Orienting Machine for Vials*. Massachusetts Institute of Technology, Department of Mechanical Engineering, 2019. [Online]. Available: https://books.google.co.uk/books?id=Mi5WzQEACAAJ

[26] NVIDIA, "Jetson nano," https://developer.nvidia.com/embedded/jetson-nano-developer-kit, n.d, accessed: 06-02-2024.

[27] A. Allan, "Benchmarking the intel neural compute stick on the new raspberry pi 4, model b," https://aallan.medium.com/benchmarking-the-intel-neural-compute-stick-on-the-new-raspberry-pi-4-model-b-e419393f2f97, 08-2019, accessed: 06-02-2024.

[28] DFRobot, "7 inch hdmi display with usb touchscreen," https://www.farnell.com/datasheets/3162025.pdf, n.d, accessed: 19-01-2024.

[29] Okdo, "Okdo, camera module, csi-2 with 2592 x 1944 resolution specification," https://docs.rs-online.com/b064/A700000006917308.pdf, n.d, accessed: 19-01-2024.

[30] ——, "Okdo, camera module, csi-2 with 2592 x 1944 resolution," https://uk.rs-online.com/web/p/raspberry-pi-cameras/2020456/, n.d, accessed: 19-01-2024.

[31] R. P. Foundation, "Raspberry pi camera module v2," https://www.raspberrypi.com/products/camera-module-v2/, n.d, accessed: 24-05-2024.

[32] ——, "Raspberry pi high quality camera," https://www.raspberrypi.com/products/raspberry-pi-high-quality-camera/, n.d, accessed: 24-05-2024.

[33] Worldsemi, "Ws2812b datasheet," https://cdn-shop.adafruit.com/datasheets/WS2812B.pdf, n.d, accessed: 24-05-2024.

[34] P. Linear, "Nema 17 stepper motor," https://pages.pbclinear.com/rs/909-BFY-775/images/Data-Sheet-Stepper-Motor-Support.pdf, n.d, accessed: 24-05-2024.

[35] BIGTREETECH, "Bigtreetech tmc2209 stepper motor driver," https://botland.store/stepstick-stepper-motor-controllers/19883-bigtreetech-tmc2209-v13-stepper-motor-driver-5904422379667.html, n.d, accessed: 24-05-2024.

[36] Pololu, "Pololu drv8825 stepper motor driver," https://www.pololu.com/product/2133, n.d, accessed: 24-05-2024.

[37] T. P. Hut, "Break beam sensor," https://thepihut.com/products/ir-break-beam-sensor-3mm-leds, n.d, accessed: 24-05-2024.

[38] Jun 2024, accessed: 08-06-2024. [Online]. Available: https://opencv.org/

[39] R. Components, "Delta electronics power supply, pmt-24v150w2ba, 24v dc, 6.25a, 150w, 1 output, 90 132v ac input voltage," https://uk.rs-online.com/web/p/switching-power-supplies/2411652?gb=s, n.d, accessed: 24-05-2024.

[40] Eelectronicparts, "Xl4015 5a high power 75w dc-dc adjustable step-down module+led voltmeter power supply module," https://www.eelectronicparts.com/products/xl4015-5a-high-power-75w-dc-dc-adjustable-step-down-module-led-voltmeter-power-supply-module, n.d, accessed: 24-05-2024.

[41] "10pcs v slot 2020 aluminum extrusion european standard 1000mm(39.4″) length anodized linear rail for cnc diy 3d printer and industrial bracket making,silver," 2024, accessed: 08-06-2024. [Online]. Available: https://www.amazon.co.uk/Aluminum-Extrusion-European-Standard-Industrial/dp/B0BM9T2579?th=1

[42] "3 to 1 ratio gt2 pulley and belt set 5mm shafts 2018," 2018, accessed: 08-06-2024. [Online]. Available: https://www.mpja.com/31-Ratio-GT2-Pulley-and-Belt-Set-5mm-Shafts/productinfo/37209%20HD/

[43] FreeCAD, "Freecad," https://www.freecadweb.org/, n.d, accessed: 19-01-2024.

[44] FormLabs, "Guide to 3d printing materials: Types, applications, and properties," https://formlabs.com/uk/blog/3d-printing-materials/, n.d, accessed: 24-05-2024.

[45] L. Shop, "Heat set insert," https://loveiissk.shop/product_details/4437313.html, n.d, accessed: 24-05-2024.

[46] Ultralytics, "Yolov8," https://github.com/ultralytics/ultralytics, n.d, accessed: 06-02-2024.

[47] N. Bhandari, "Aabb obb photo," https://www.namasbhandari.in/post/the-era-of-oriented-bounding-boxes, n.d, accessed: 01-06-2024.

[48] J. Ding, N. Xue, G.-S. Xia, X. Bai, W. Yang, M. Yang, S. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang, "Object detection in aerial images: A large-scale benchmark and challenges," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.

[49] encord, "Pre trained model," https://encord.com/glossary/pre-trained-model-definition/, n.d, accessed: 01-06-2024.

[50] Python, "Python," https://www.python.org/, n.d, accessed: 19-01-2024.

[51] Git, "Git," https://git-scm.com/, n.d, accessed: 22-01-2024.

[52] Pylint, "Pylint," https://github.com/pylint-dev/pylint, n.d, accessed: 19-01-2024.

[53] RealVNC, "Realvnc," https://www.realvnc.com/en/, n.d, accessed: 22-01-2024.

[54] V. S. Code, "Visual studio code," https://code.visualstudio.com/, n.d, accessed: 22-01-2024.

[55] Pygame, "Pygame documentation," https://www.pygame.org/docs/, n.d, accessed: 22-01-2024.

[56] MyreMylar, "Pygame gui," https://github.com/MyreMylar/pygame_gui/tree/0cbf7056518377b455d51a8d20167f4029756ad9, n.d, accessed: 22-01-2024.

[57] Roboflow, "Roboflow," https://roboflow.com/, n.d, accessed: 01-06-2024.

[58] T. Schimansky, "Custom tkinter widgets," https://github.com/TomSchimansky/CustomTkinter, n.d, accessed: 25-01-2024.

[59] R. Components, "Rs pro c14 snap-in iec connector," https://docs.rs-online.com/c6ad/0900766b815867b2.pdf, n.d, accessed: 22-01-2024.

[60] ——, "Rs pro 18 awg hook up wire, pvc insulation," https://docs.rs-online.com/770d/A700000007035534.pdf, n.d, accessed: 22-01-2024.

[61] G. UK, "Pat testing: portable appliance testing," https://www.hse.gov.uk/electricity/faq-portable-appliance-testing.htm, n.d, accessed: 27-01-2024.

[62] P. T. Course, "Pat testing course," https://www.pat-testing-course.com/, n.d, accessed: 28-01-2024.

[63] Brett, "Guide: How to assemble the creality ender-3 - lets print 3d," Apr 2018, accessed: 08-06-2024. [Online]. Available: https://letsprint3d.net/how-to-assemble-creality-ender-3/

[64] Fritzing, "Fritzing," https://fritzing.org/, n.d, accessed: 22-01-2024.

[65] 2024, accessed: 24-05-2024. [Online]. Available: https://www.tensorflow.org/tensorboard

[66] P. Langechuan, "generalized focal loss: learning qualified and distributed bounding boxes for dense object," 2020. [Online]. Available: https://patrick-llgc.github.io/Learning-Deep-Learning/paper_notes/gfocal.html

[67] "v8.1.0 release - yolov8 oriented bounding boxes (obb) ultralytics discussion," Jan 2024, accessed: 08-06-2024. [Online]. Available: https://github.com/orgs/ultralytics/discussions/7472

[68] "how to prune yolov8 model and save it for finetuning issue 3507 ultralytics," Jul 2023, accessed: 08-06-2024. [Online]. Available: https://github.com/ultralytics/ultralytics/issues/3507

[69] GitHub, "Github," https://github.com/, n.d, accessed: 22-01-2024.

[70] Oct 2020. [Online]. Available: https://pypi.org/project/PyGetWindow/

[71] May 2023. [Online]. Available: https://pypi.org/project/PyAutoGUI/

[72] author, "Oscilloscope | pico technology," 2024, accessed: 24-05-2024. [Online]. Available: https://www.picotech.com/products/oscilloscope

[73] Ultralytics, "Tips for best training results," 2023, accessed: 08-06-2024. [Online]. Available: https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/

[74] M. D. (jiffyclub), "Snakeviz," 2024, accessed: 24-05-2024. [Online]. Available: https://jiffyclub.github.io/snakeviz/

[75] A. Rosebrock, "Intersection over union (iou) for object detection - pyimagesearch," Nov 2016, accessed: 08-06-2024. [Online]. Available: https://pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/

[76] O. R. developers, "Onnx runtime," https://onnxruntime.ai/, 2021, version: x.y.z.

9 APPENDIX

Contents

9.1 GitHub Repository

The code and design files for this project can be found at the following link:
https://github.com/samin50/MEngFYP

9.2 Raspberry Pi Benchmarking

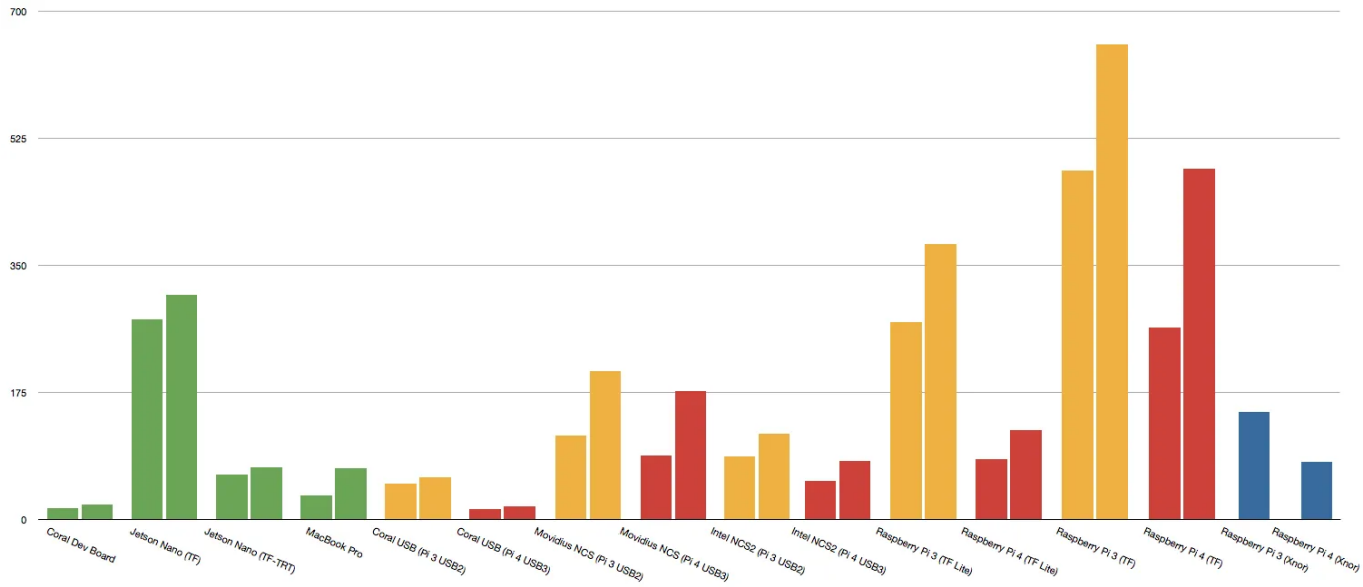


Figure 84: Raspberry Pi Benchmarking

9.3 Component Detection Mosaic

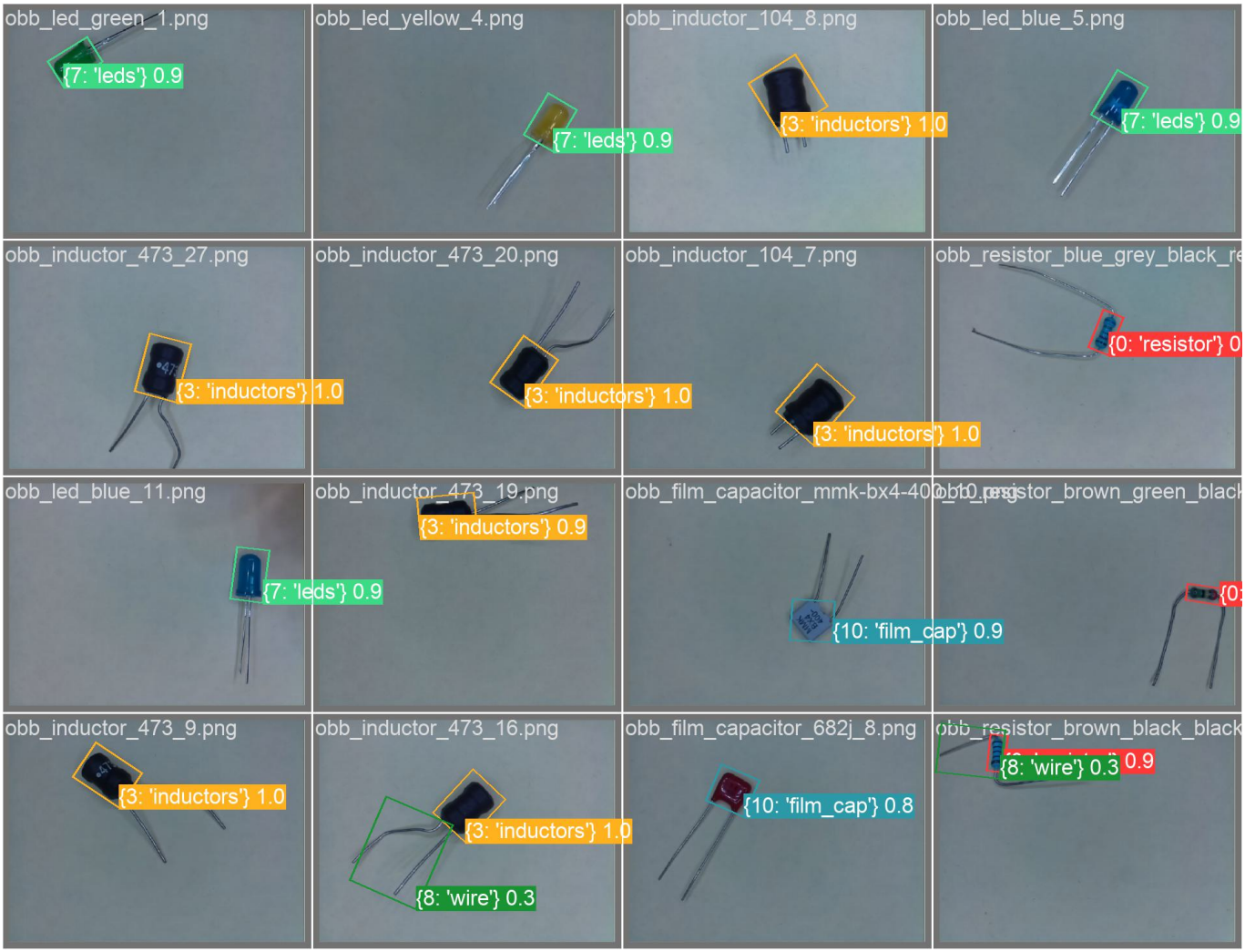


Figure 85: Component Detection Mosaic