IMPERIAL

Vision-Assisted Mechatronic Component Sorter

MEng Final Year Report

Author
Shaheen Amin
CID: 01866464

Supervised by
Dr. Ed Stott
Prof. -

A thesis submitted in fulfilment of requirements for the degree of
Master of Electronics 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

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

add acknowledgements

Contents

List of Todos Todo list

1 PROJECT SPECIFICATION

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.

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's deliverables are as follows:

- **Vision System:** A computer vision system capable of real-time classification of electrical components.
- **Mechatronic System:** A mechatronic system that can sort the classified components into designated bins set by the user.
- **User Interface:** An interface from which to observe and control the system's state.

Samples of the different electrical components commonly used in the Level 1 labs were collected, and they are as follows:

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

The repository can be found in the Appendix 9.1.

Conform more to fyp report guidelines

fix all pictures

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 Existing Computer Vision Systems

A range of computer vision systems have been explored in the literature, ranging from PCA (Principle Component Analysis) Dhenge et al. [2] to more modern computer vision techniques like CNNs as used by Xu et al. [3], Chand and Lal [4]. However, 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.

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, which are very distinct components. In this paper, PCA is used as a feature extractor and then fed directly into, though not explicitly mentioned, a FCL (Fully Connected Layer), as the classifier. Although a valid approach, 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, so this approach is not viable for the project.

The authors Xu et al. [3] use the SqueezeNet CNN architecture to identify 22 different subcategories of electronic components, specifically resistors, capacitors, and inductors, achieving 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 is an impressive result. This work shows that a CNN is a viable approach to the problem of component identification, helping to inform the design of the project's computer vision system.

The paper by Muminovic and Sokic [5] 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 (86%, however, the test images used are resistors that are positioned far away from the camera, as shown in Figure 3, and so the bands are very small, which will not be the case in the project's vision system, so the accuracy will likely be higher) which makes it a very promising resource for the project.

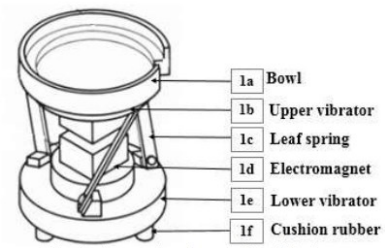

Figure 1: VBF [6]

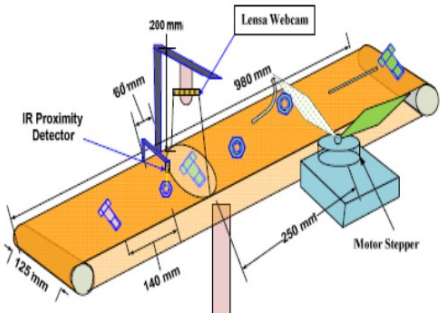

Figure 2: Conveyor Bel [2]

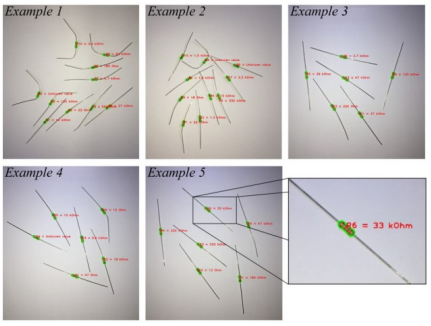

Figure 3: Resistor Test Set [5]

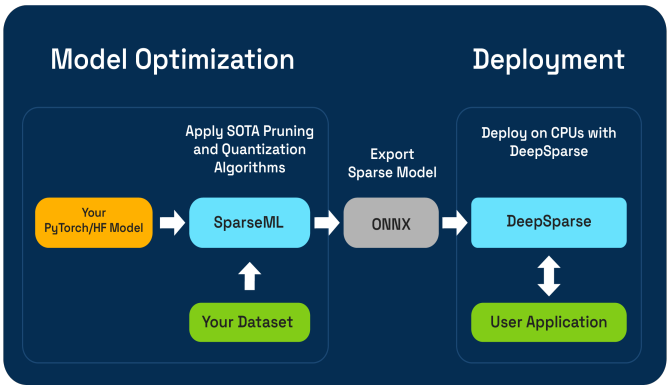

Figure 4: SparseML Pipeline [7]

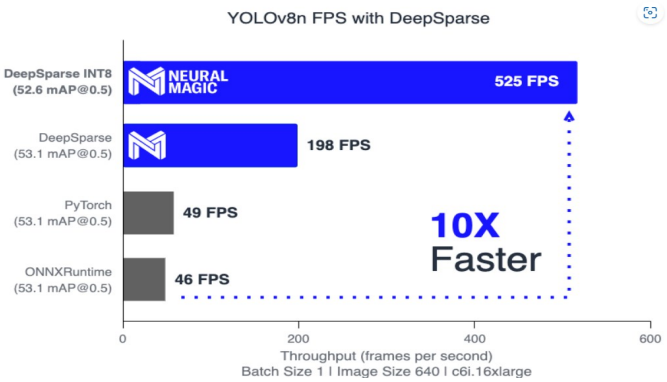

Figure 5: DeepSparse Performance [8]

properly place pictures (from two column format)

2.2 Computer Vision Architectures

For the problem of component identification, the computer vision system must be able to identify components in real-time, as the components may eventually be moving on a conveyor belt, captured using a camera facing down at the components. This means that the computer vision system must be able to identify components in a very short amount of time, and so the system must be computationally efficient. The envisioned system is designed to be self-contained, and as described in Section 4 (Implementation), the system will run on a Raspberry Pi 4, which has a 1.8GHz quad-core 64-bit ARM Cortex-A72 CPU and up to 8GB of RAM [9], and must also be accurate and able to control the other systems in the project in real-time.

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%.

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 [10]. 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. [11] 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. [12] 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.

However, efforts made by Neural Magic [8] in the optimisation of YOLOv5 and YOLOv8 show performance improvements of up to 10x on CPUs, through their open-source optimisation toolkit SparseML [7] and CPU inferencing runtime, DeepSparse [13]; a very promising result.

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-labeled 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. [14] proposes many different training techniques, the most promising of which is semi-supervised learning. In this approach, the model is trained on a small labeled 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 labeled 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 labeling the entire dataset.

2.4 Mechanical Design

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 2, 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.

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. [15] 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.

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 [16], which use precise vibrations to orientate and transport components, and pneumatic systems from Abe et al. [17] (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 achiev-

able 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. Initially, the aim was for the system to be semi-autonomous, with a design allowing the camera to face upwards. This configuration would enable the user to place the component on an acrylic plate directly above the camera for immediate identification, enabling easy user interaction as the view of the component is not obstructed. Having the camera face-down would block the user's view of the component, which would be an inconvenience, and as such, the current design of the system features an up-facing camera. However, this will be changed to reflect the research outlined above, in the next stage of the project.

2.4.2 Bowl Feeders

While researching for this project, and especially in videos [18], it seems most industrial sorting machines use a Vibratory Bowl Feeder (VBF) to help feed components into the sorting system; as shown in Figure 1, the VBF consists of a bowl that vibrates coupled with a spring and electromagnet. The paper by Nam et al. [6] 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 [19] delve into a mathematical model of a VBF, optimising more on the overall performance of the VBF rather than efficiency, and Silversides et al. [20] 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 [21] 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 Commerical Systems

complete this section with a few youtube videos

2.6 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. Additionally, the YOLO architecture may be covered in the modules Deep Learning COMP70010 and Machine Learning COMP70014, enriching the understanding of this architecture.

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 computer vision system will be based on the research outlined above in the next stage of the project.

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 semi-supervised learning 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 expose 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.

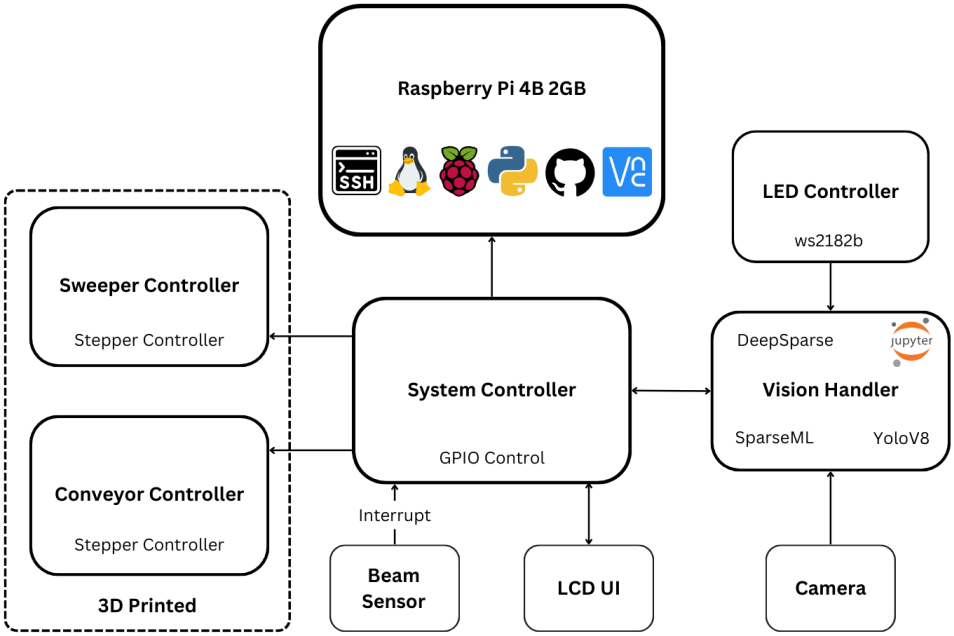

Figure 6: System Architecture Diagram

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 [9] 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.

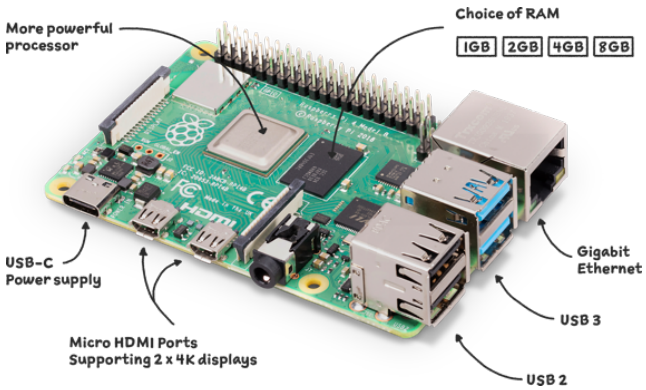

Figure 7: Raspberry Pi 4 Model B 2GB [9]

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 [22], 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 [9]. 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 [23] 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 49.

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.

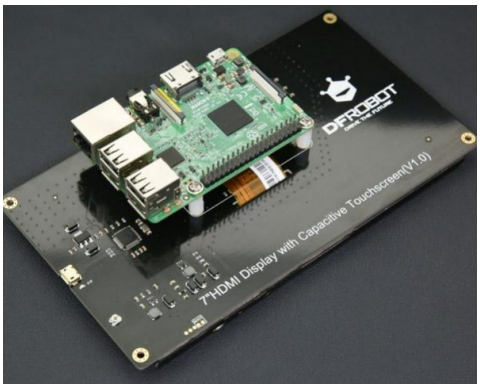

Figure 8: DFRobot 7″ Touchscreen Display [24]

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 [25], 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.

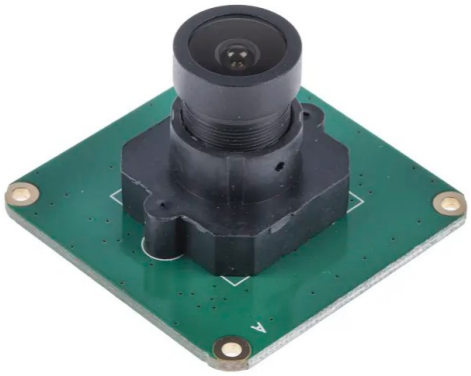

Figure 9: Okdo Adjustable Focus OV5647 Camera [26]

Other cameras were considered, such as the Raspberry Pi Camera Module V2 [27], 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 [28] 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 [29] 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.

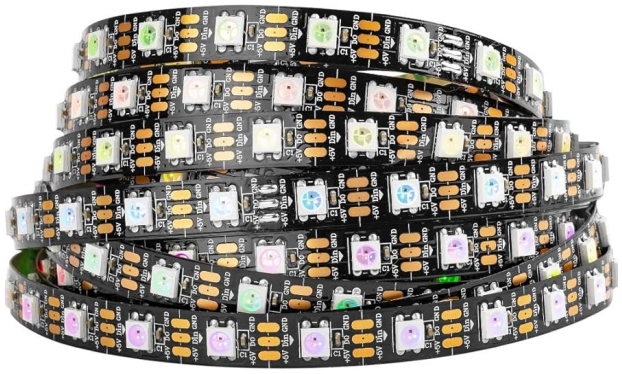

Figure 10: WS2812B LEDs [29]

Figure 11: 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 [30] as they are widely used and have a large amount of support and documentation, also typically used in FDM 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 12: NEMA17 Stepper Motor [30]

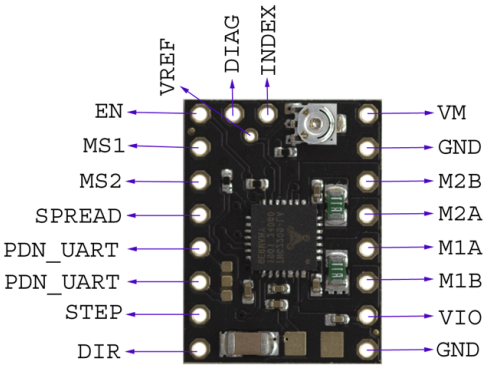

Figure 13: TMC2209 Stepper Motor Driver [31]

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 [32], 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 [33] 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 14: IR Beam Break Sensor [33]

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 [34] 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 15: Power consumption of the system

Figure 16: 24V DC 6.25A Power Supply Unit [35]

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 15. 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.

add power consumption measurements from usb for complete power - implementation maybe?

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.

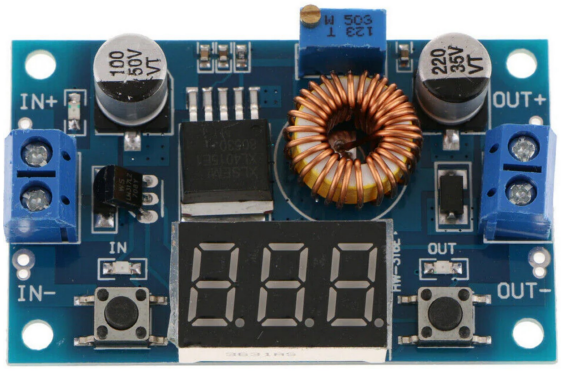

Figure 17: XL4015 High Power Step Down Convertor [36]

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.

3.3 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 [37], 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 [38]. 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.

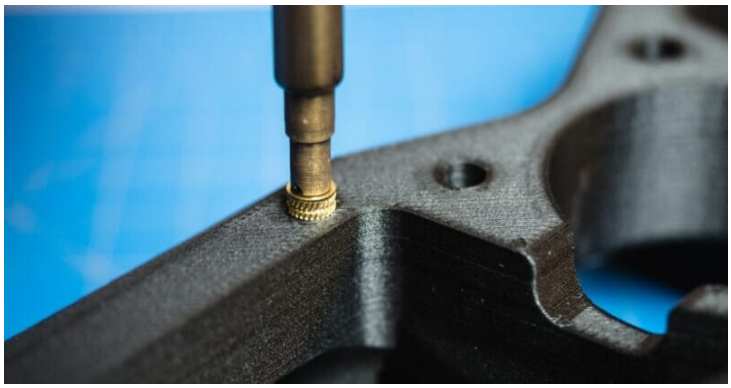

Figure 18: Heat Set Inserts [39]

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.

include nema17 and tmc2209 specs

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

talk about camera distortion correction

YOLOv8's training framework [40] 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 [40], 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 [13] and SparseML [7] 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 would have introduced latency and would have required an internet connection. However, during the implementation phase, the typical YOLOv8 model not found to be suitable for the task.

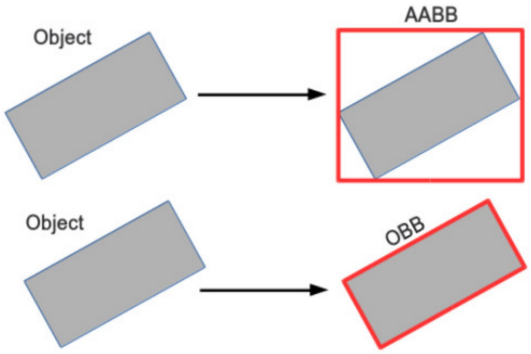

Figure 19: OBB vs AABB Bounding Boxes [41]

Common Computer Vision architectures typically draw AABBs (Axis Aligned Bounding Boxes) when detecting the object in the camera frame, as shown in Figure 19. 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 [40], which will be used for this task.

Model	**mAP^{50}**	**Speed 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 [40]

The OBB models come pretrained on the DOTOv1 dataset [42], 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 [43]. The performance of the different sized YOLOv8-obb models is shown in Table 2. The most crucial parameters to take into consideration is the mAPval (mean average precision) and the CPU ONNX (inference latency). 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 $A \cap B$ being the intersection of the two bounding boxes, and $A \cup B$ being the union of the two bounding boxes.

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

From Table 2, the YOLOv8x-obb model has the best performance, with a mAP^{50} of 81.36 and a CPU ONNX of 1759.10ms. The YOLOv8n-obb model has the worst performance, with a mAP^{50} of 78.0 and a CPU ONNX 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 [13] and SparseML [7] libraries to optimise the model for the Raspberry Pi, the YOLOv8n-obb model will be used for the project.

justify why training component detcection and resistor detection model

flow diagram of cv pipeline?

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 [44], due to its ease of use and large amount of support and libraries available, and will be run entirely on the Raspberry Pi. Git [45] is used for version control to ensure that changes to the software can be tracked and reverted if necessary, Pylint [46] is used for code linting to ensure that the code is clean and readable, RealVNC [47] is used for remote access to the Raspberry Pi, and Visual Studio Code [48] is used as the IDE for development. As mentioned in Subsection 3.3, FreeCAD [37] has been used for the mechanical design of the system.

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 [49] and Pygame GUI [50], 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 [51]. 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 [52], 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 [37], a parametric 3D modelling software. The below figures show the system in its entirety, with the front and back views of the system.

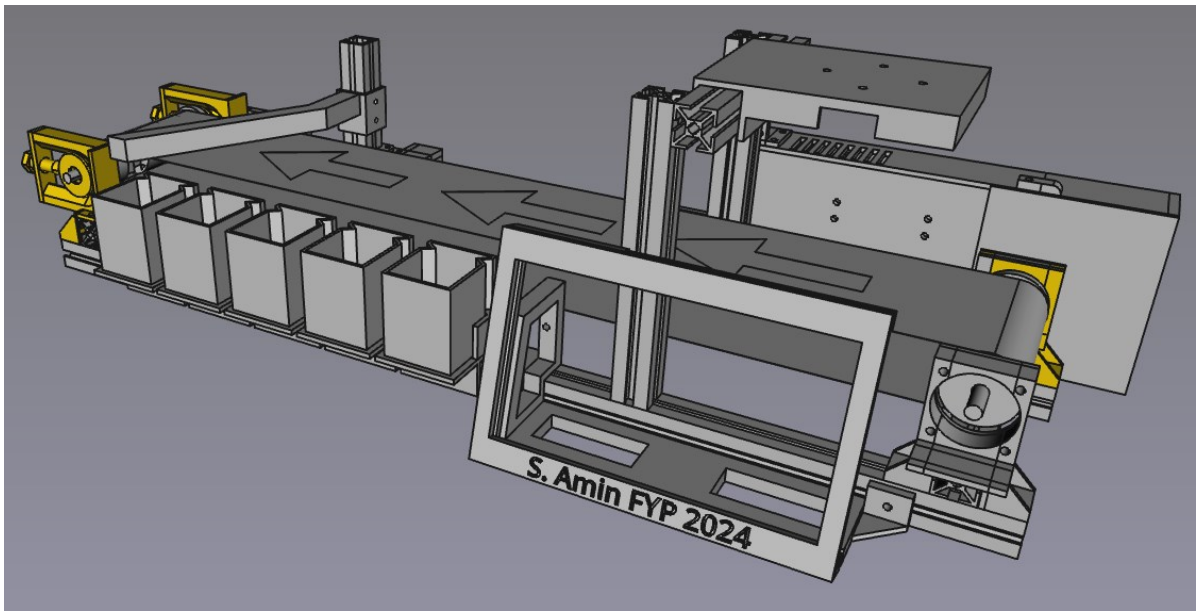

Figure 20: Front View of the Mechanical Design

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.

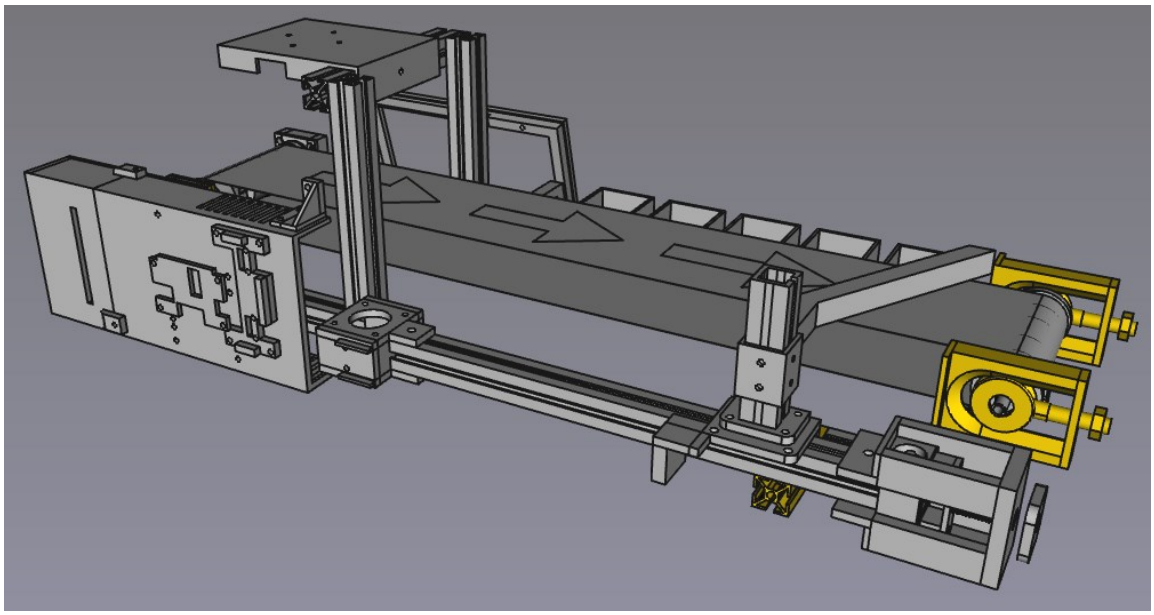

Figure 21: Back View of the Mechanical Design

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.

show actual system

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 a 3D printed case, which prevents accidental contact with the PSU's terminals.

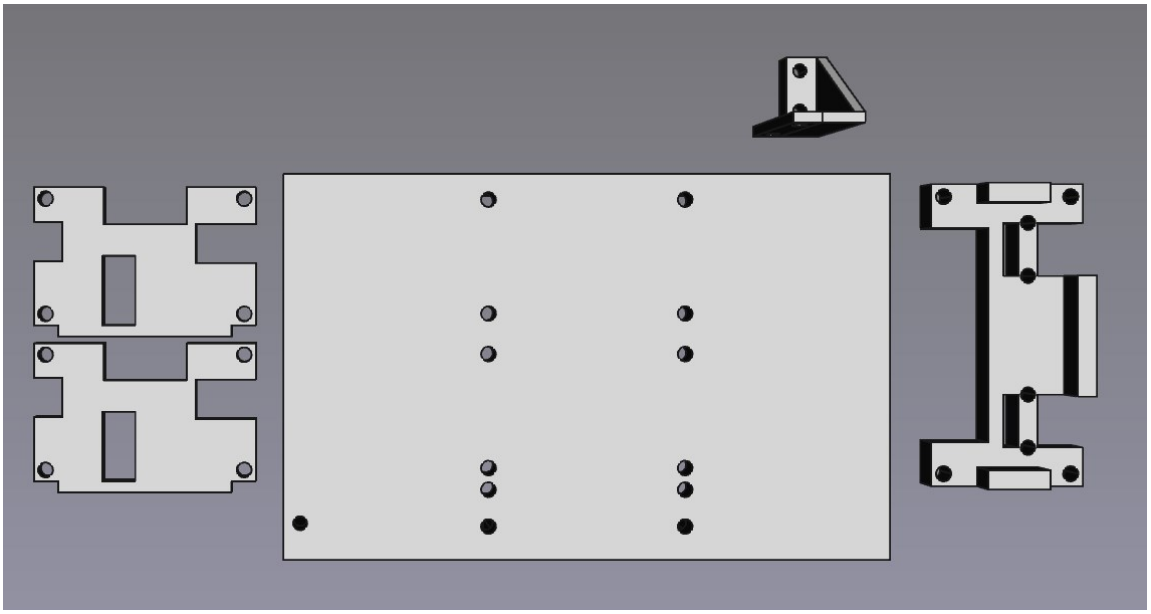

Figure 22: PSU Enclosure

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.

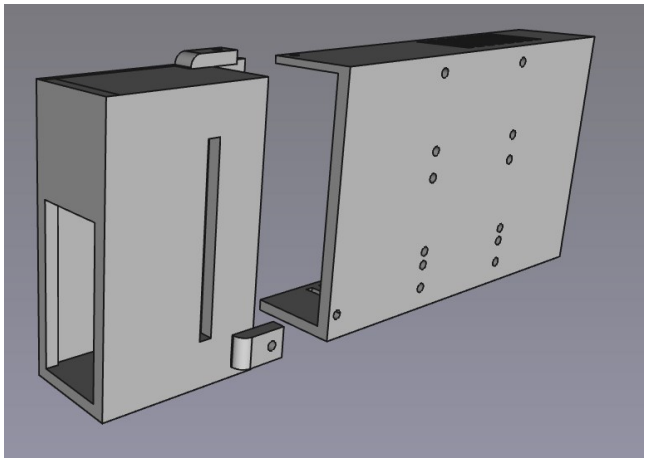

Figure 23: PSU Switch Housing

Figure 24: IEC C14 Power Socket [53]

A standard IEC C14 power connector and an RS Pro Snap-In Fused Rocker Switch [53] 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 [54], 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.

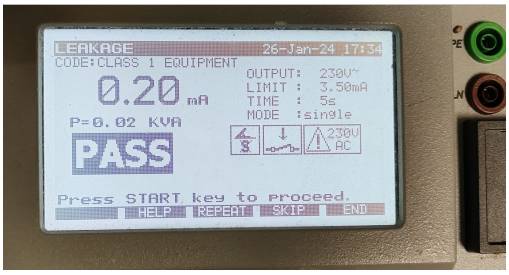

Table 4: PAT Testing Results and PAT Machine

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 [55, 56]. The results of the PAT test are shown in Table 4.

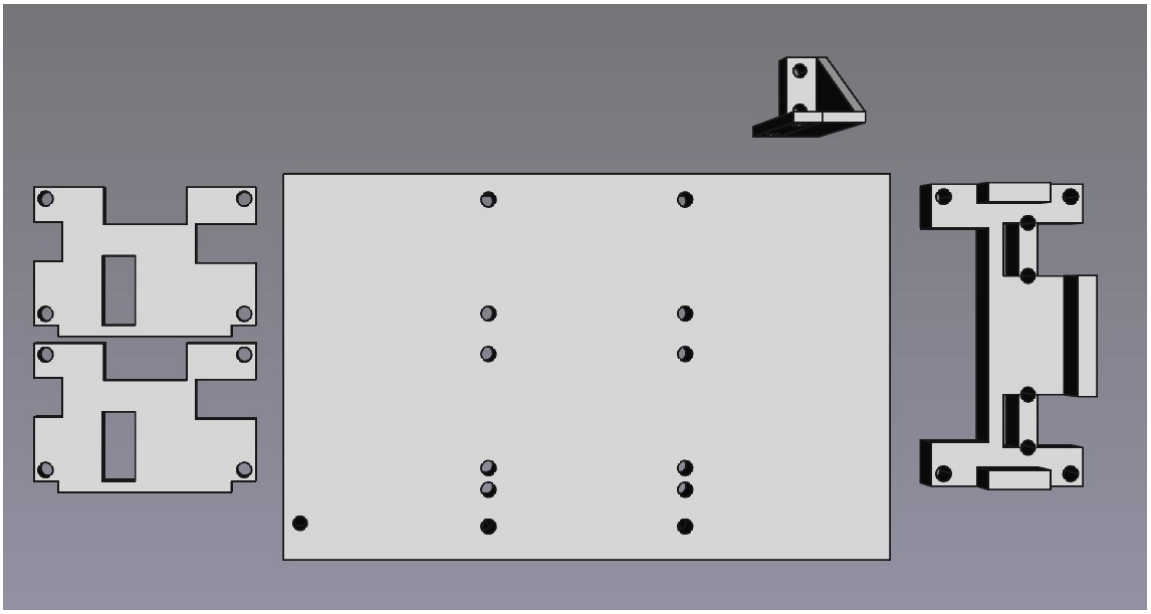

Figure 25: Final PSU Enclosure

show actual PSU

As shown in Figure 25, the final PSU enclosure is shown to be working with the 5V and 12V outputs connected to the mounted USB breakout boards.

4.1.3 LCD Mount

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.

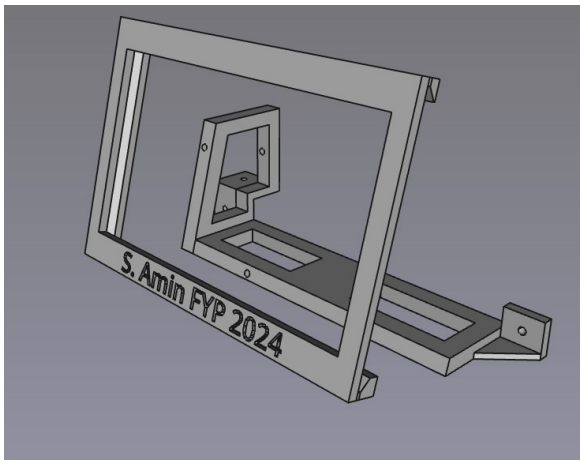

Figure 26: LCD Mount

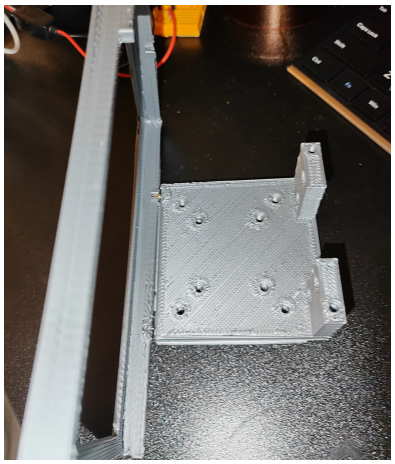

Figure 27: Unbraced LCD Mount

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, as it did in previous iterations, as shown in Figure 27.

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.

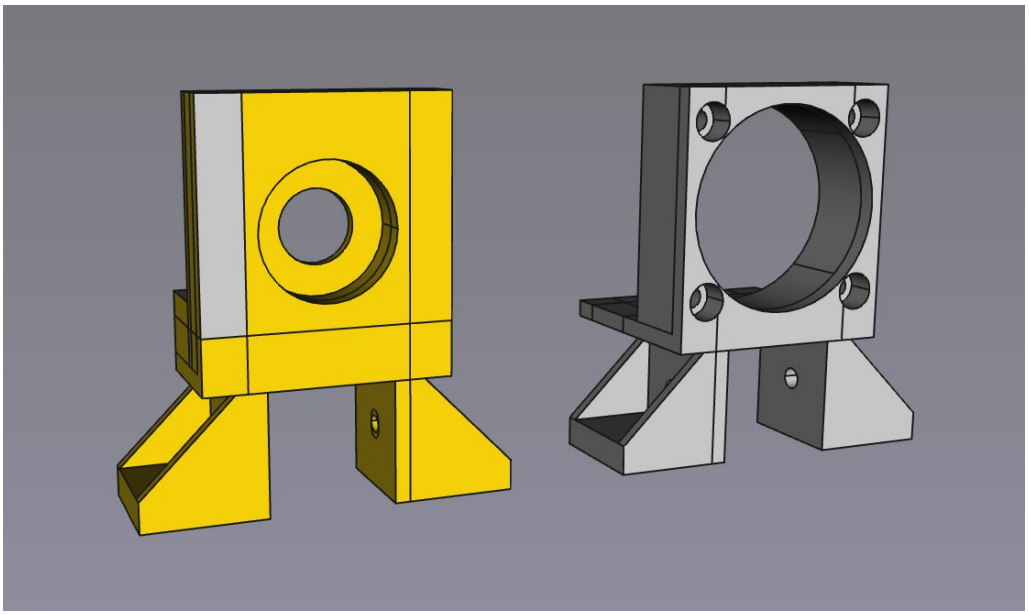

Figure 28: Driven Conveyor Roller

Figure 28 shows the mounts for the driven conveyor roller, the right side mounting the NEMA17 stepper through four M3 screws, and the left side mounting the other side of the roller by allowing it to rotate freely in the mount. The left side 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.

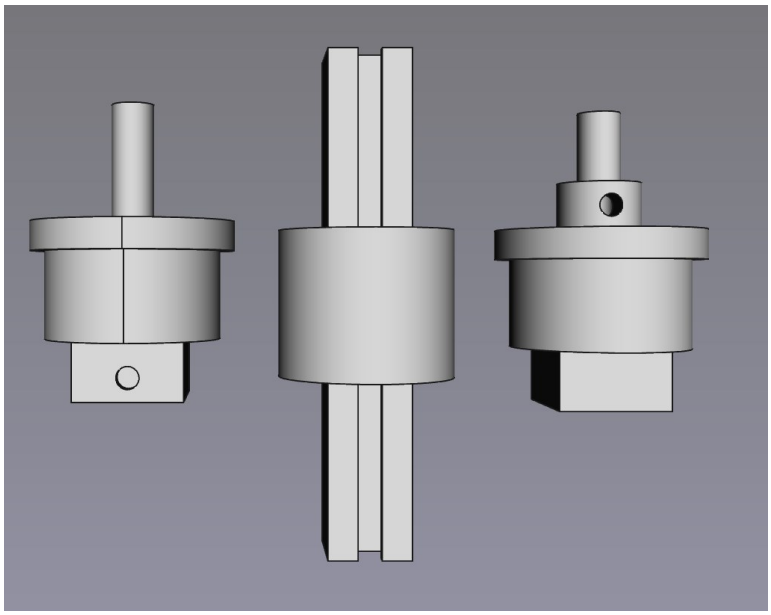

Figure 29: Conveyor Belt Rollers

Figure 29 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 30 shows the adjustable roller design.

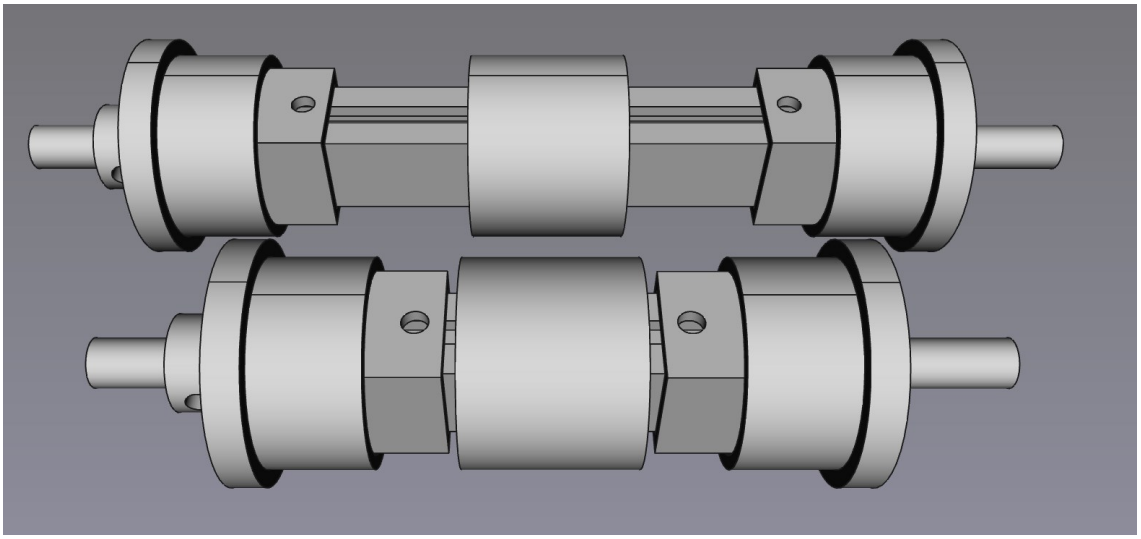

Figure 30: Adjustable Conveyor Rollers

On the other side of the conveyor belt, the belt tensioner is used to ensure the belt is taut.

explain belt tensioner design

4.1.5 Sweeping Mechanism

show nema17 mount and endstop

show belt driven arm design, inspired by 3D printer designs

show off gantry, ender 3 design inspiration with v rollers

show belt tensioner design

show adjustable arm height design

show bin design

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

show wiring diagram

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.

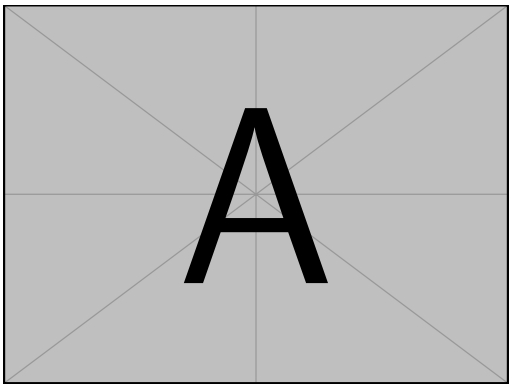

Figure 31: WS2812B LED Strip

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 GND pin is connected to the GND pin on the Pi, and the 5V pin is connected to the 5V output of the power supply. It has been cut into a square shape to fit around the camera, allowing it to uniformly illuminate the components on the conveyor belt.

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 12 and Figure 13. 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 [30], and the TMC2209 can handle between 4.75V and 29V [31]. 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.

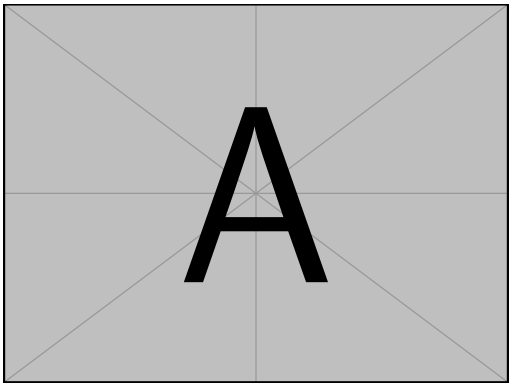

Figure 32: Schematic for NEMA17 Stepper Motor Control designed in Fritzing [57]

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.

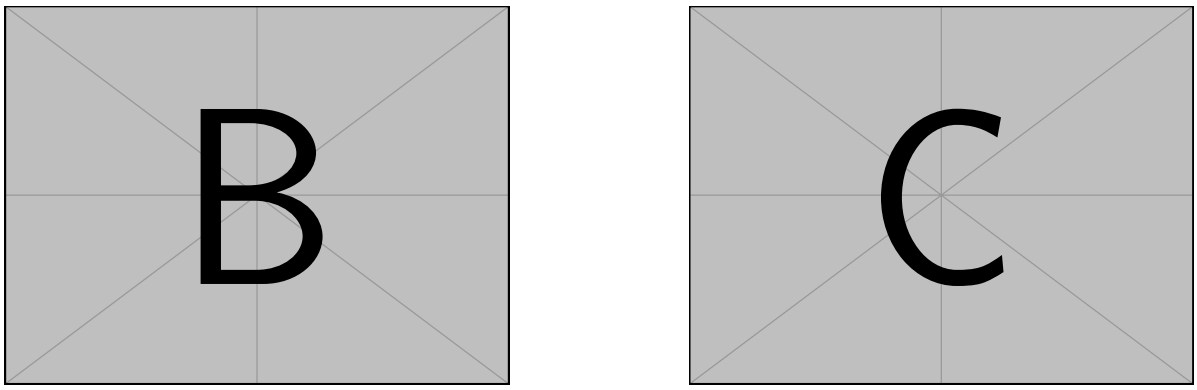

Figure 33: PCB for NEMA17 Stepper Motor Control

include diagram and picture of motor driver

4.3 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 [46] 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 [48] 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 1: 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 1.

```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 2: Example simulation of the GPIO library

As shown in Code Snippet 2, the `simulate.py` file contains a class called `GPIO` that emulates the GPIO library.

The Pi uses Git [45] for version control, and the repository is hosted on GitHub [58], 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.

4.3.1 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. From here, the user is able to…

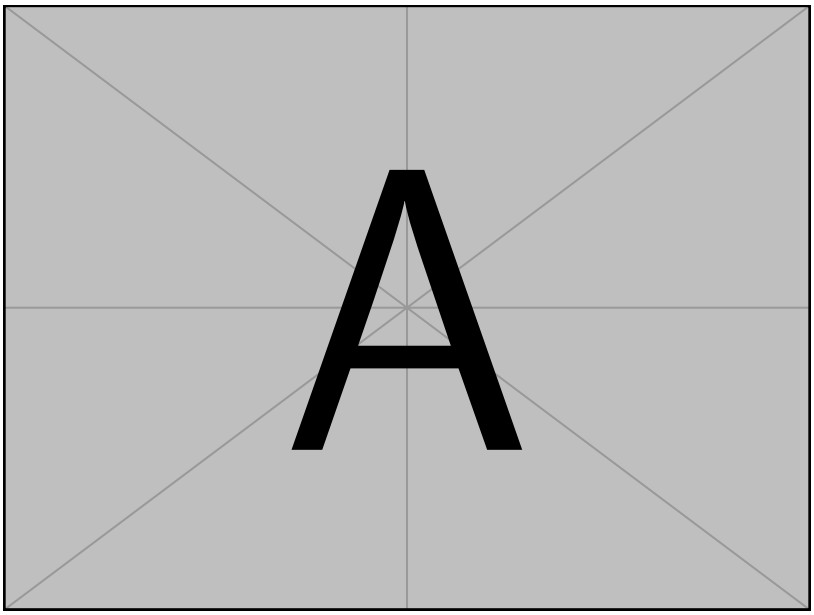

Figure 34: User Interface

As shown in Figure 34, the user interface consists…

Written in pygame [49] and pygame_gui [50], 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 Section 5.

The user interface also supports a "training mode", where the camera feed is enlarged in order to capture images of components to be used for training the computer vision system, done programmatically by toggling a flag called `TRAININGMODE` in the `LCD UI` class. This is only used during the dataset collection phase of the project, and is not used during the normal operation of the system. This is discussed in more detail in Subsubsection 4.3.4.

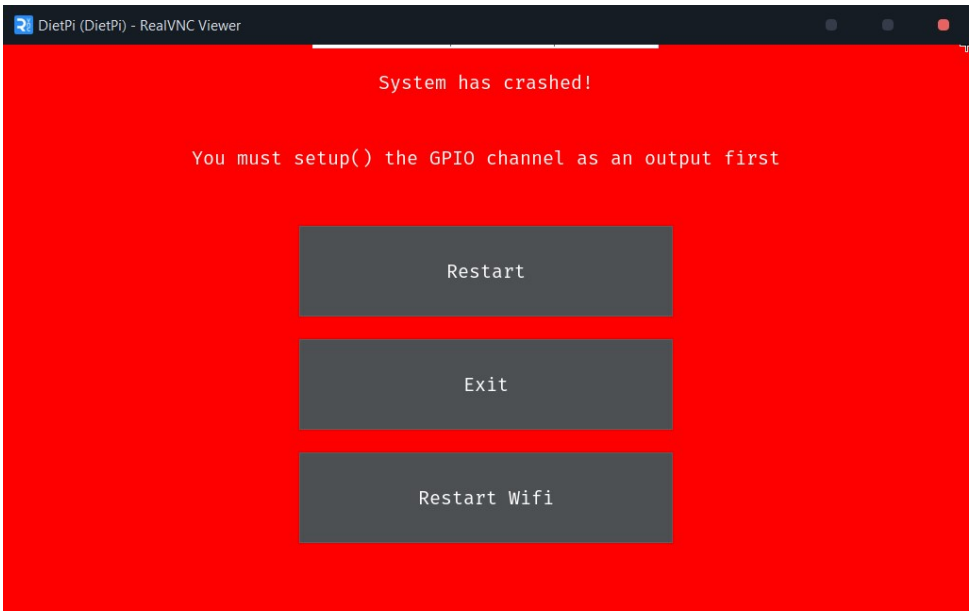

Figure 35: System crash during development

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 35. 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 4.3.2.

Complete section after fully implementing the user interface

4.3.2 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 36 shows what the user interface displays when the camera is disconnected.

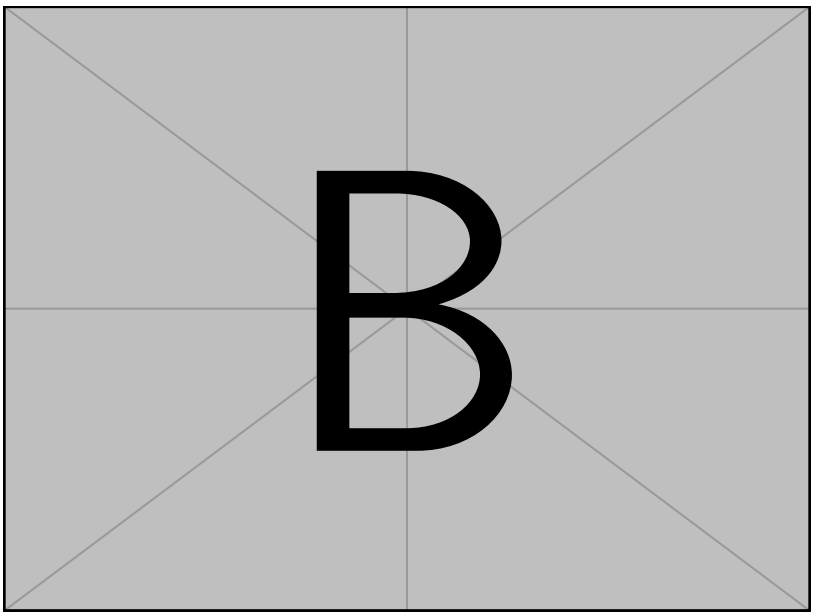

Figure 36: Camera disconnect error

4.3.3 System Controller

diagram of interupt/data flow?

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

finish section

4.3.4 Dataset Collection

As mentioned in Subsubsection 3.5.2, 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.

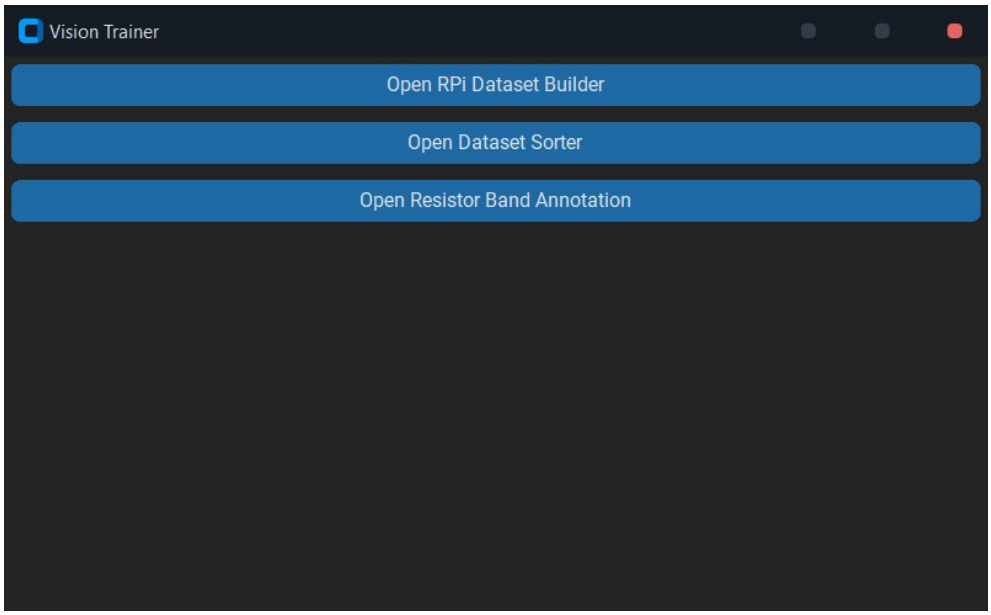

Figure 37: Vision Trainer Menu

Upon launch, the user is able to select which mode they would like to use, as shown in Figure 37. 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 7: Main buttons for the dataset collection tool

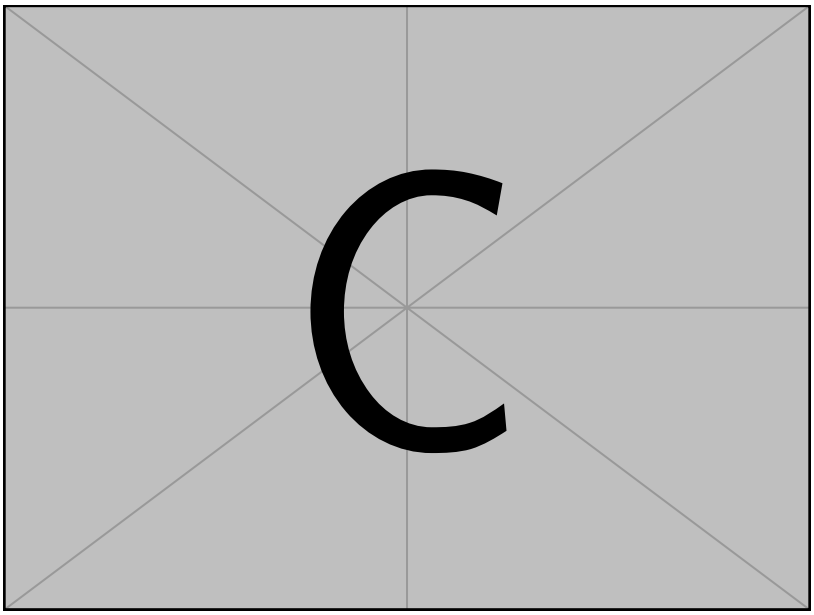

Figure 38: Training mode screen

As shown in Figure 38, 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 [47], 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 8.

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 8: Main keybinds for the dataset collection tool

The dataset collection tool uses PyGetWindow [59] to identify the VNC window, and PyAutoGUI [60] to capture the images. OpenCV's [34] `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 3: Camera feed extraction from VNC window

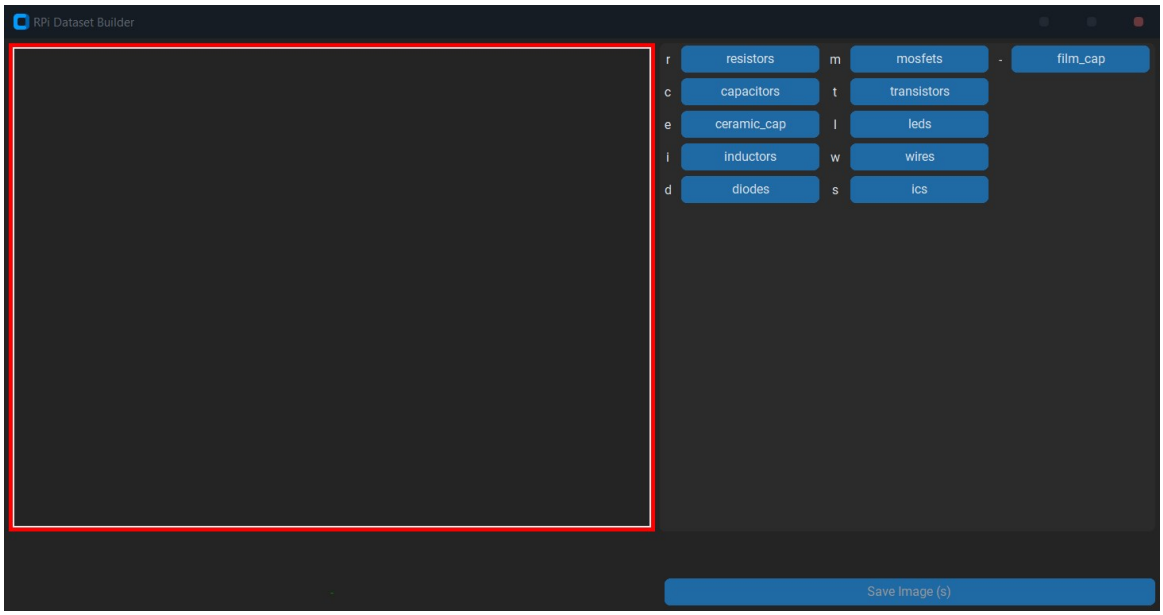

Figure 39: Uncaptured camera feed with red border

If the camera feed is not captured (it may be obscured), the border of the camera feed will turn red, as shown in Figure 39, indicating that the camera feed was not captured, otherwise it will turn green, as shown in Figure 40. In Figure 39, 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.

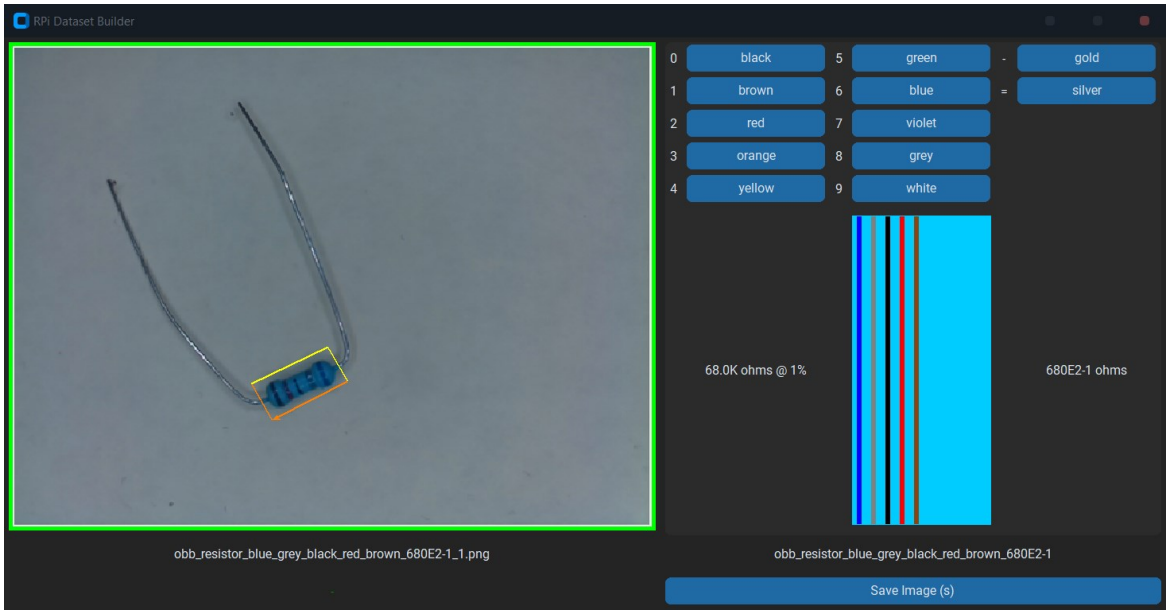

Figure 40: Successfully captured camera feed with green border and annotated resistor

In Figure 40, 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 40, 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 was discovered during the training of the OBB model, as discussed in Subsection 4.4

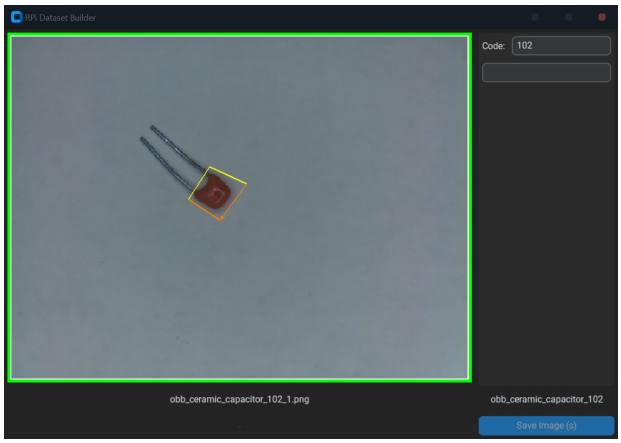

Figure 41: Annotated Ceramic Capacitor

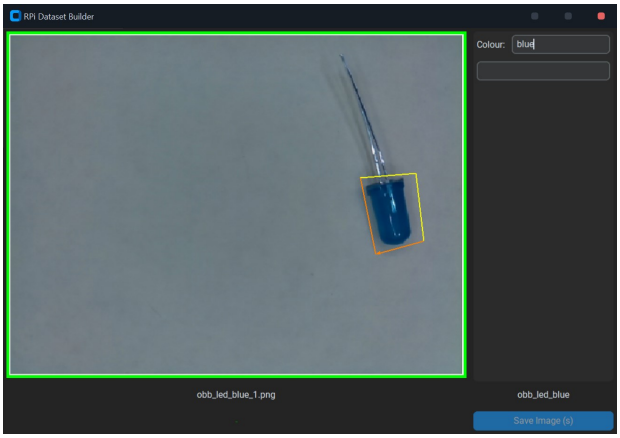

Figure 42: Annotated LED

Figure 41 shows an annotated ceramic capacitor, and Figure 42 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 4.4. 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.

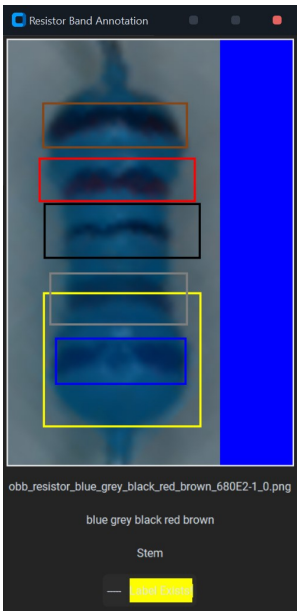

Figure 43: Resistor band annotation tool

As shown in Figure 43, 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 the component identification model will always produce resistors upright). The format is discussed in more detail in Subsection 4.4.

4.4 Computer Vision

As mentioned in Section 2 and Subsection 3.4, the YOLO-OBB model [40] is used to classify and detect the electrical components, and the regular YOLO model was used for resistor value detection.

4.4.1 YOLO Format and Dataset Collection Process

To train a YOLO model with the Ultralytics library [40], 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 4.3.4 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 4, 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 4: 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 5: 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.

4.4.2 Component Identification

Using the dataset collection tool discussed in Subsubsection 4.3.4, a total of 974 images were collected across 7 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

Table 9: Distribution of the dataset

It is important to note that the 7 classes 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 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 [61], 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 10: 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.

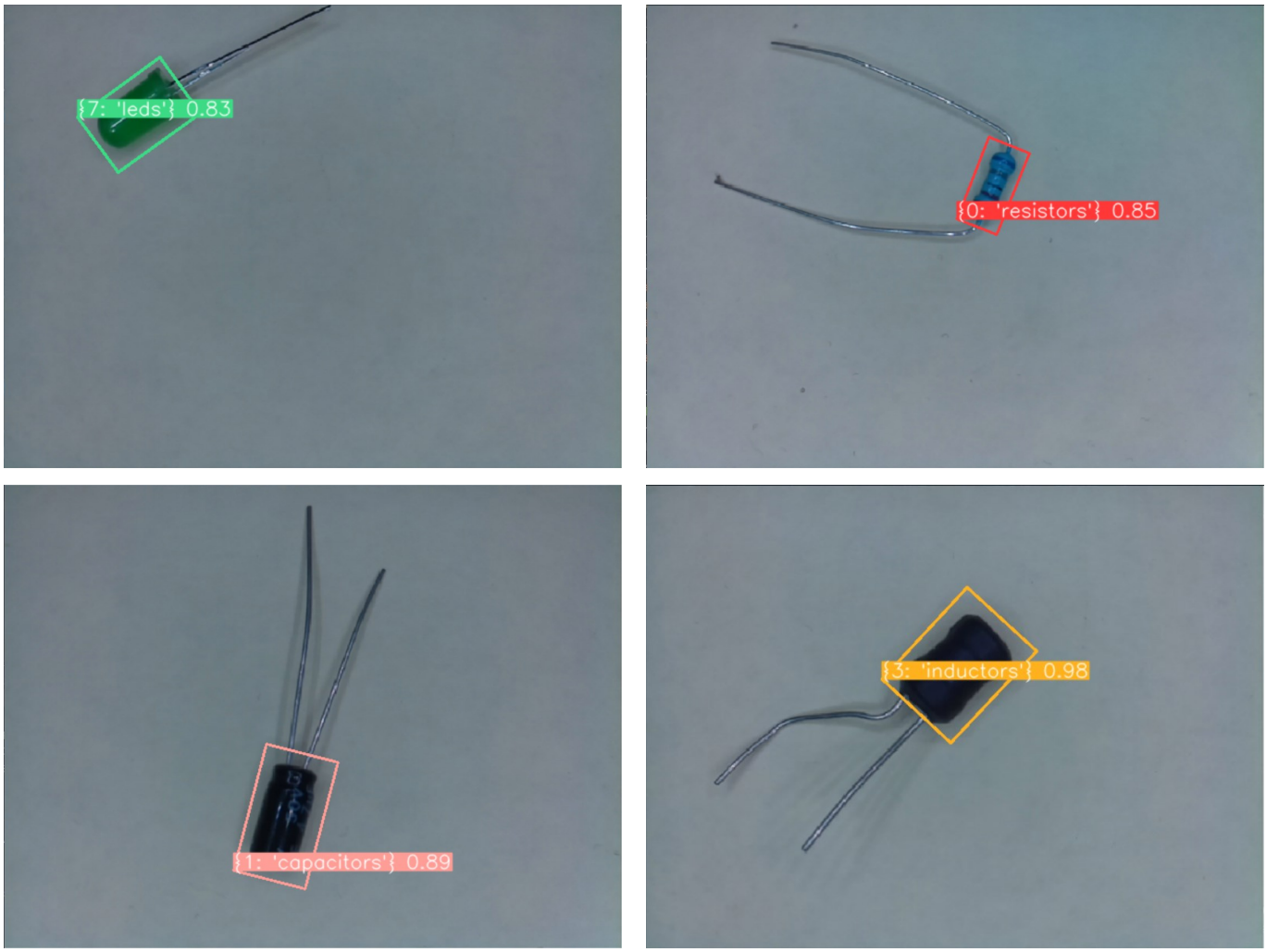

Figure 44: Inference ran using the `detection-trainer.ipynb` notebook, showing an LED, a resistor, capacitor and inductor, being correctly detected and classified.

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.

4.4.3 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 4.3.4.

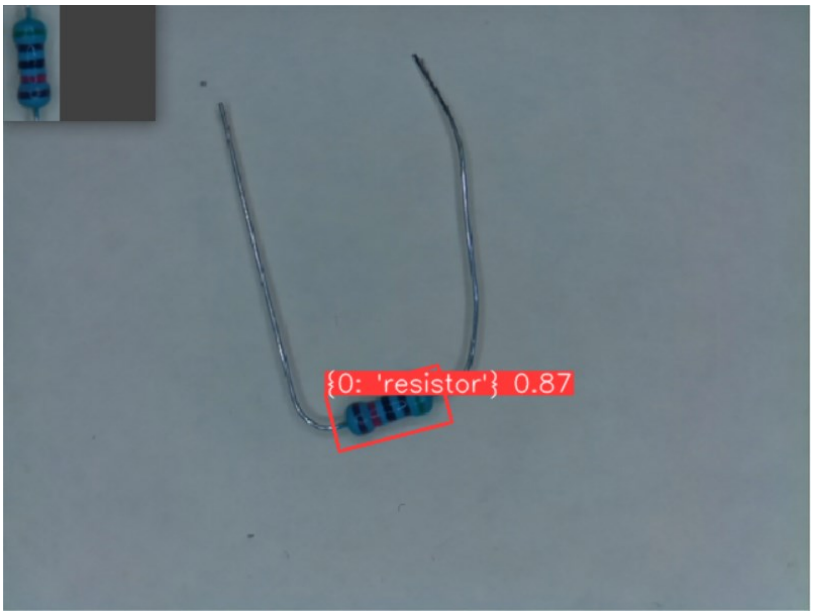

Figure 45: Resistor Identification and Cropping

As shown in Figure 45, a small window opens up that shows the cropped resistor image using OpenCV [34]. The images were then saved to a folder structure matching Code Snippet 5, 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.

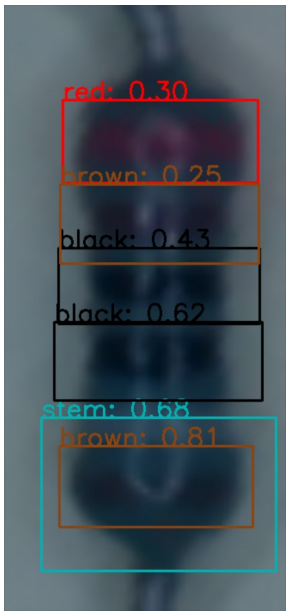

Figure 46: Resistor Band Detection

4.5 Sparsification and Deployment

talk about 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 [62], 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. This process is not as efficient as sparsification, as the model must be retrained, however it is still a viable option for reducing the model size and increasing inference speed.

4.6 Problems and Changes

5 RESULTS AND EVALUATION

Contents

This chapter will focus on the evaluation of the different subsystems of the project, and offer a critical analysis of the project as a whole. The evaluation will be based on the project's ability to meet the requirements set out in the project specification, as well as the project's performance in terms of accuracy, efficiency, and usability. The evaluation will also consider the limitations of the project and suggest areas for future work.

5.1 Mechanical Design

5.2 Electronics and Software

5.3 Computer Vision

This section will evaluate the Computer Vision system, as discussed in Subsection 3.4 and implemented in Subsection 4.4.

5.3.1 Component Detection Model

The Component Detection model was trained on a dataset of 976 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 10.

Figure 47: IoU Example [63]

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 47.

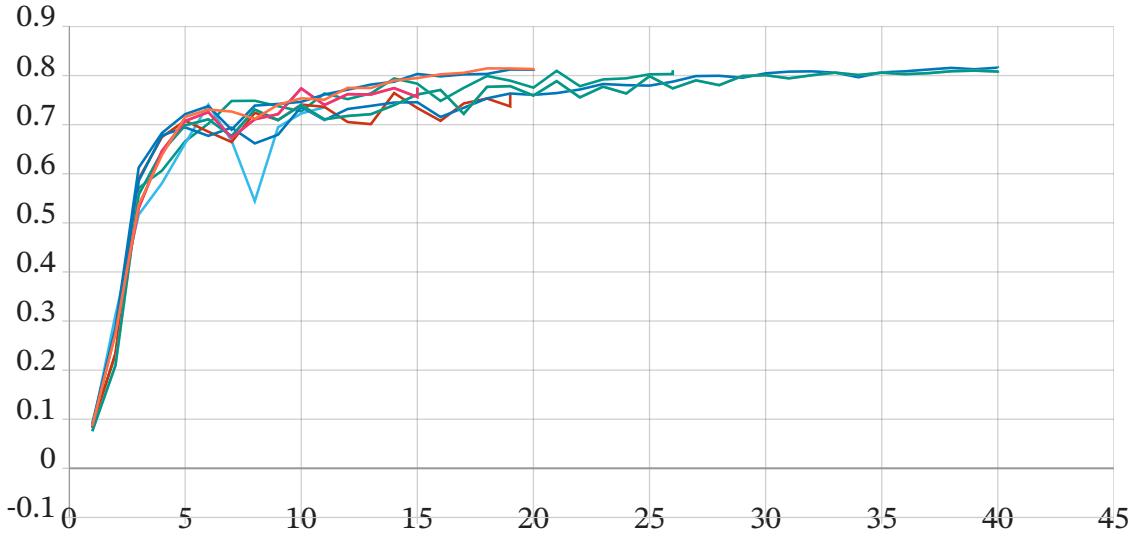

Figure 48: TensorBoard graphing: all runs $mAP^{50\text{-}95}$ against epochs

In Figure 48, 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% $mAP^{50\text{-}95}$. 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 evaluate the model's performance.

talk about tensor board

show metrics

On the GTX 3080 Ti,

Interestingly, if Class Loss is set to 1.0, Box Loss is set to 4.0, and DFL is set to 1.5, the model converges in around 15 epochs, taking only 2 minutes to train. However, the model's performance is slightly impacted, calculating metrics on the test set yields an $mAP^{50\text{-}95}$ of 77.8%, but it was decided that the time savings were not worth it given it was only a few minutes of training time. The model was therefore trained with the parameters shown in Table 10.

show pictures of model in action

5.3.2 Resistor Value Classification Model

6 TESTING

7 CONCLUSION

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] 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.

[6] 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.

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

[8] ——, "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.

[9] 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.

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

[11] 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

[12] 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.

[13] N. Magic, "Deepsparse," https://github.com/neuralmagic/deepsparse, n.d.

[14] 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

[15] 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.

[16] 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

[17] 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

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

[19] 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

[20] 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

[21] 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

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

[23] 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.

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

[25] 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.

[26] ——, "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.

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

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

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

[30] 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.

[31] 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.

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

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

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

[35] 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.

[36] 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.

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

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

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

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

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

[42] 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.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[61] 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

[62] Jan 2024, accessed: 08-06-2024. [Online]. Available: https://github.com/orgs/ultralytics/discussions/7472

[63] 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/

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

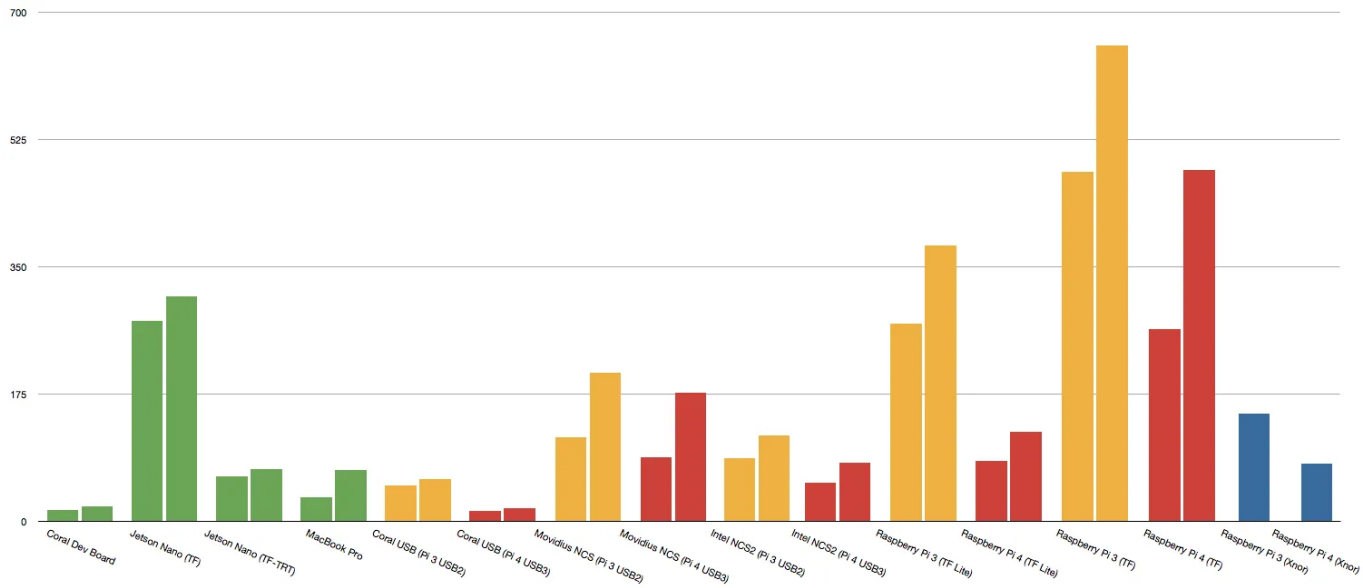

Figure 49: Raspberry Pi Benchmarking