FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Development of an automatic equipment to separate medical capsules by colour

Pedro Teixeira de Freitas

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Armando Araújo

October 31, 2024

Resumo

A indústria farmacêutica está constantemente à procura de formas de aumentar a eficiência de produção e o controlo de qualidade, especialmente quando se trata de classificação de cápsulas por cor e quantidade. Esta tese descreve o design, desenvolvimento e validação de um sistema automatizado de classificação de cápsulas destinado a melhorar a precisão e eficiência da produção. O sistema utiliza uma tecnologia moderna de processamento de imagem combinada com automatização mecânica para classificar as cápsulas a taxas mais rápidas do que as abordagens manuais, conseguindo uma excelente taxa de precisão na classificação de cores sob configurações reguladas.

O sistema de classificação de cápsulas destina-se especialmente a pequenas áreas industriais, utilizando uma câmara de alta resolução e algoritmos de software especializados para reconhecer e categorizar as cápsulas com precisão. Os testes e validação extensiva revelaram que o sistema pode processar até três cápsulas por segundo, excedendo os métodos típicos de classificação humana. Além disso, o sistema manteve um tempo de atividade quase perfeito com um tempo de inatividade operacional mínimo, confirmando a sua fiabilidade e eficácia.

No entanto, foram encontrados vários obstáculos, como a sensibilidade às mudanças nas condições de iluminação e a fiabilidade do hardware durante a operação contínua. Para ultrapassar estes problemas, as soluções incluem a incorporação de sistemas de iluminação adaptativos, a atualização para componentes mecânicos mais duradouros e a utilização de algoritmos de processamento de imagem mais sofisticados. O desenvolvimento futuro centrar-se-á na melhoria do processo de classificação mecânica com portas automatizadas para classificação física direta, melhorando a precisão da contagem e alargando a adaptabilidade do sistema a diferentes condições de funcionamento.

Este estudo contribui para os campos da automação e da visão mecânica no fabrico farmacêutico, propondo uma alternativa escalável, eficiente e precisa à classificação manual de cápsulas que pode reduzir drasticamente os custos e melhorar os padrões de produção.

Palavras-Chave: Automação, classificação de cápsulas, processamento de imagem, reconhecimento de cor.

Abstract

The pharmaceutical business is constantly looking for ways to increase manufacturing efficiency and quality control, particularly when it comes to capsule sorting by color and amount. This thesis describes the design, development, and validation of an automated capsule sorting system intended to improve production accuracy and efficiency. The system uses modern image processing technology paired with mechanical automation to sort capsules at rates faster than manual approaches, reaching an excellent accuracy rate in color sorting under regulated settings.

The capsule sorting system is particularly intended to fit into small industrial areas, using a high-resolution camera and powerful software algorithms to precisely recognize and categorize capsules. Extensive testing and validation revealed that the system can handle up to three capsules per second, exceeding typical human sorting methods. Furthermore, the system maintained near-perfect uptime with minimum operational downtime, confirming its dependability and effectiveness.

However, the encountered discovered various obstacles, such as sensitivity to changing lighting conditions and hardware dependability during continuous operation. To overcome these issues, solutions include incorporating adaptive lighting systems, updating to more lasting mechanical components, and using more powerful image processing algorithms. Future development will center on improving the mechanical sorting process with automated doors for direct physical sorting, improving counting accuracy, and broadening the system's adaptability to different operational conditions.

This study contributes to the fields of automation and machine vision in pharmaceutical manufacturing by proposing a scalable, efficient, and accurate alternative to manual capsule sorting that might drastically cut costs and improve production standards.

Keywords: Automation, capsule sorting, image processing, colour recognition.

United Nations Sustainable Development Goals

The project "Development of Automatic Equipment to Separate Medical Capsules by Colour" makes a substantial contribution to various United Nations Sustainable Development Goals (SDG). This section describes how the project connects with particular SDGs, the objectives it addresses, the contributions it makes, and the performance measures used to assess its impact.

This project addresses the following SDGs directly or indirectly:

- **SDG 3: Good Health and Well-being**: The initiative improves pharmaceutical quality and safety by assuring proper capsule sorting, which reduces the possibility of dispensing mistakes.
- **SDG 9: Industry, Innovation, and Infrastructure**: The project's development of automated sorting equipment represents a key advance in the pharmaceutical production process, enhancing efficiency and reliability.
- **SDG 12: Responsible Consumption and Production**: The improved sorting process minimizes waste and enhances the sustainability of pharmaceutical production.

Within each identified SDG, the project contributes to the following specific targets:

- **SDG 3.8: Access to quality essential health-care services and access to safe, effective, quality, and affordable essential medicines and vaccines for all.** The initiative provides high standards in pharmaceutical manufacture, hence increasing the availability of safe and effective pharmaceuticals.
- **SDG 9.4: Upgrade infrastructure and retrofit industries to make them sustainable, with increased resource-use efficiency and greater adoption of clean and environmentally sound technologies.** The project improves production efficiency and resource utilisation by implementing modern automation technologies.
- **SDG 12.2: By 2030, achieve the sustainable management and efficient use of natural resources.** The optimized sorting process reduces waste and promotes the efficient use of materials.

The initiative improves quality of life, environmental preservation, social equality, and other SDG-related factors in the following ways:

- **Quality of Life**: Ensuring that patients receive the right medication at the appropriate dosage has a direct influence on patient safety and treatment efficacy, improving overall health outcomes.

- **Environmental Protection**: Automated sorting reduces waste generated from incorrect sorting and ensures that resources are used more efficiently.
- **Economic Efficiency**: Automation of a previously manual process lowers labor costs and enhances production speeds, resulting in economic benefits for pharmaceutical companies.

To assess the project's impact on the SDGs, the following performance indicators and metrics were identified:

- **Accuracy Rate of Capsule Sorting**: Measures the percentage of correctly sorted capsules to ensure the system's reliability and accuracy.
- **Throughput Rate**: Evaluates the number of capsules sorted per hour to measure the system's efficiency.
- **Reduction in Sorting Errors**: Tracks the reduction in sorting mistakes before and after using the automated system.
- **Operational Cost Savings**: Calculates the labor and error-related cost savings that can be achieved by automation.

Table 1: SDG Development Goals

SDG	Goal	Contribution	Performance Indicators and Metrics
3	3.8	Enhanced patient safety and treatment efficacy	Accuracy Rate of Capsule Sorting, Reduction in Sorting Errors
9	9.4	Increased efficiency and reduced resource usage	Throughput Rate, Operational Cost Savings
12	12.2	Sustainable management of resources	Reduction in waste, Efficiency metrics

Using this structured method, the project clearly illustrates its compatibility with and contributions to the United Nations Sustainable Development Goals. It emphasizes the project's positive impact on health, industry, and sustainability while also offering a solid foundation for assessing its long-term performance and benefits.

Agradecimentos

A conclusão da tese e consequentemente do mestrado simboliza uma etapa da minha vida que está agora a terminar.

Quero começar por agradecer à minha família pois sem eles nada disto seria possível, à minha namorada por tudo o que faz por mim, a todos os meus amigos que tornaram este percurso sem sombra de dúvida muito melhor e a todas as pessoas que de alguma forma contribuíram para o meu crescimento pessoal.

O meu muito obrigado.

Pedro Teixeira de Freitas

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This document portrays the development of a project within the scope of the Dissertation course in the Master's degree in Electrical and Computer Engineering at Faculdade de Engenharia da Universidade do Porto, under the supervision of professor Armando Araújo. This project was proposed by Farmácia Couto, a pharmacy located in Vila Nova de Gaia, and was guided and supervised by Doctor Pedro Coelho.

Farmácia Couto is responsible for preparing compounded medication and one of the pharmaceutical forms frequently prepared in the compounded medicine preparation laboratory are hard capsules. The development of an automatic equipment to separate medical capsules by colour represents a strategic initiative aimed at optimizing pharmaceutical operations and ensuring the integrity of medication distribution. Rooted in the recognition of the significance of accurate medication sorting and dispensation, this project seeks to leverage advanced technological solutions to address this critical aspect of pharmaceutical practice.

At its core, this project addresses the challenge of quickly and efficiently sorting medical capsules based on colour, a task that is currently performed manually and is prone to errors and inefficiencies. By automating this process, we aim to not only enhance the speed and accuracy of medication sorting but also to minimize the risk of human error and ensure consistency in pharmaceutical operations.

Moreover, this project aligns with broader efforts to enhance patient safety and optimize healthcare delivery. By facilitating the precise sorting of medical capsules, we contribute to the overarching goal of ensuring that patients receive the right medication in the right dosage and at the right time, (which is) a fundamental aspect of quality pharmaceutical care.

1.1 Contextualization

In the realm of pharmaceutical industry and operations, segregation and distribution are paramount tasks that demand the utmost accuracy and efficiency. The origin of this necessity is deeply rooted in the critical role that pharmacies play in the healthcare delivery. Medications, particularly

capsules, are often distinguished by colour which represent different compounds or dosages, therefore pharmacists and patients can use this colour code as a visual cue in order to ensure the correct medication is dispensed and consumed. However, the traditional method of manually sorting capsules is loaded with challenges, including the propensity for human error, the time-consuming nature of the task, and the physical strain on pharmacy staff, which collectively compromise operational efficiency and, more critically, patient safety.

The introduction of an Automated Capsule Colour Sorting System into this environment is predicated on the urgent need for a solution that not only mitigates these challenges but also aligns with the evolving demands of modern healthcare practices. This need is further amplified by the increasing volume of prescriptions pharmacies handle daily, coupled with the growing complexity of medication regimens. Thus the development of this system is contextualized within this pressing need, promising a paradigm shift in how pharmacies operate by introducing precision, speed, and reliability into the capsule sorting process.

Moreover, the proposal for the sorting system project is informed by a broader trend toward automation and digital transformation in healthcare which is driven by an understanding that technology, when thoughtfully integrated into this practices, can significantly enhance service delivery and patient outcomes. Automation technologies, in particular, have shown immense potential in reducing manual labor, minimizing errors, and improving process efficiencies hence, in the context of pharmacies, the capsule sorting system represents a specific application of these broader technological advancements, tailored to address a key operational challenge.

1.2 Problem Definition

In the pharmaceutical sector, precise capsule sorting by colour and amount is critical for maintaining quality control and efficiency in packing and delivery. Many facilities have traditionally used manual sorting methods, which are prone to human error, inefficiency, and inconsistency. Furthermore, rising demand for pharmaceutical items mandates a more quick and dependable system to keep up with market demands. Due to speed and accuracy constraints, conventional manual procedures are frequently unable to satisfy these needs. Furthermore, space limits in production facilities need small but efficient equipment to maximize the utilization of available space.

1.2.1 Objectives

The project aims to address these challenges through the following specific objectives:

1. **Development of a Capsule Separation System by Colour**: To develop an automated method for precisely classifying capsules based on their colour. This system will use modern image processing technology to recognise and categorise capsules, lowering the risk of mistakes associated with human sorting.

2. **Development of a Capsule Separation System by Quantity**: To create a mechanism within the system that can count and classify capsules based on certain amounts. This feature is critical for meeting packaging specifications that require exact capsule counts, increasing productivity, and decreasing waste.

3. **Product Size Adjusted to the Bench Space**: To create a system with a small footprint that can be installed in the constrained places available in pharmaceutical manufacturing facilities. The technology will be designed to enhance efficiency while maintaining performance, allowing it to effortlessly fit into existing operations.

4. **Speed of Executing Tasks Greater Than That of Humans**: To guarantee that the system functions faster than manual sorting methods. This improvement in speed will greatly improve throughput, allowing pharmaceutical businesses to fulfill greater production objectives while lowering personnel expenses associated with manual sorting.

1.2.2 Expected Results

By meeting these objectives, the project expects to achieve the following results:

- **Enhanced Accuracy and Efficiency**: The automated system will reduce human mistakes in sorting, resulting in more precision and consistency in capsule processing. This reliability is critical for meeting pharmaceutical quality requirements.

- **Increased Productivity**: With a faster pace than manual sorting, the system will handle much more capsules per hour, increasing total productivity.

- **Space Optimization**: The system's small form will allow it to be integrated into a variety of industrial scenarios without requiring considerable rearrangement or extension of existing space.

- **Cost Reduction**: Automating the capsule sorting process will lower the labor expenses involved with manual sorting, as well as the overhead costs caused by mistakes and inefficiencies.

1.3 Dissertation Structure

This dissertation is divided into six chapters besides the introduction, each focusing on a distinct element of the development and assessment of an automated capsule sorting system for pharmaceutical applications. In Chapter 2, Literature Study, is conducted a comprehensive analysis of present technologies and approaches in automated sorting systems, investigating current solutions, their limits, and how advances in image processing and automation have impacted system design. Chapter 3 describes the capsule sorting system design and idea in depth, discussing the system architecture, which includes both the hardware configuration (cameras, conveyor belts, and sorting devices) and the software algorithms used for color identification and sorting. The integration of

these components into a coherent system designed for small areas is also covered. Afterwards, Chapter 4 outlines the approaches used to design the system discussing the technical aspects of software creation, such as programming languages and tools utilized, as well as hardware assembly. The methods for testing and calibrating system components to ensure they work as intended are also described. Later on, Chapter 5 focuses on testing and validating the completed system describing the testing environment and assessment criteria, such as efficiency, precision, and dependability. The results are provided to show how the system meets or exceeds the performance metrics specified, and any discrepancies or concerns discovered during testing are investigated. Finally, the paper concludes in Chapter 6 with some finishing remarks about the problem, challenges faced and future work.

Chapter 2

Literature Study

The automation of sorting operations, particularly in the pharmaceutical business, offers a huge step forward in maintaining product quality and operating efficiency. Because pharmaceutical items, particularly capsules and pills, must be sorted and packaged with extreme precision due to safety concerns, the industry is always seeking novel technology to improve these processes. The combination of modern image processing technologies, small yet powerful computer systems and software tools like OpenCV provide a solid foundation for constructing an autonomous capsule sorting system. Such systems are critical not only for guaranteeing accurate medicine delivery, but also for sustaining high throughput in production lines.

2.1 Colour Recognition Technologies

2.1.1 Colour Image Processing Fundamentals

At the heart of any colour-based sorting system is the method by which colour is perceived and processed. Traditional RGB (Red, Green, Blue) colour space, while prevalent in general imaging applications, might be less efficient in industrial environments where lighting conditions change and accuracy is crucial, which is where HSV (Hue, Saturation, Value) colour space shines. HSV enables more consistent colour identification across diverse lighting conditions by separating colour information (hue) from lighting (value). This functionality is especially beneficial in contexts such as pharmaceutical manufacturing, where constant and accurate colour identification is mandatory.

While the RGB colour space is widely used, it's important to note that some platforms, such as OpenCV, initially handle images in BGR (Blue, Green, Red) format. Although this distinction might seem minor, it can affect how images are processed and analysed. For applications utilising OpenCV, images are often converted from BGR to RGB before further processing to align with more conventional RGB-based methods or directly converted from BGR to HSV for improved colour detection efficiency.

By converting RGB to HSV (or straight from BGR to HSV in some circumstances, such as when using OpenCV), systems gain improved ability to handle illumination fluctuations, which is critical for accurate and consistent colour sorting. Such conversions guarantee that the hue (colour type) is constantly identified, regardless of variations in illumination intensity or colour, making HSV especially useful in applications that need exact colour separation.

2.1.2 Computational Process of BGR to HSV Conversion

In BGR colour space, colours are represented as a combination of three primary colours, red, green, and blue. Each colour channel has values typically ranging from 0 to 255, indicating how much of each colour is present. BGR is primarily used in OpenCV.

HSV stands for Hue, Saturation, and Value where Hue indicates the colour type and is expressed as a degree on the colour wheel (0 to 360°), Saturation refers to the vibrancy of a colour (0% indicates no colour, i.e., gray, and 100% is the full colour). and Value indicates the brightness of the colour (0% is completely black, and 100% is the brightest).

The conversion from BGR to HSV entails a number of computer procedures that modify how colour information is recorded, making it more consistent with how humans perceive colour differences (Saravanan, Yamuna, and Nandhini [10]). The conversion can be mathematically described as the following steps:

1. Normalize BGR values: The BGR values, which are usually in the range of 0 to 255, are normalized to a range of 0 to 1 by dividing each by 255.

$$B', G', R' = \frac{B}{255}, \frac{G}{255}, \frac{R}{255}$$

2. Calculating $C_{\max}$, $C_{\min}$, and Δ, helps in determining the Value and Saturation components in the HSV model:

$$C_{\max} = \max(B', G', R'), \quad C_{\min} = \min(B', G', R'), \quad \Delta = C_{\max} - C_{\min}$$

3. Compute Hue (H): Hue calculation depends on which BGR component has the maximum value.

$$H = \begin{cases} 60^{\circ} \times \left(\frac{G'-R'}{\Delta} + 2 \right) & \text{if } C_{\max} = B' \\ 60 \times \left(\frac{R'-B'}{\Delta} + 4 \right) & \text{if } C_{\max} = G' \\ 60 \times \left(\frac{B'-G'}{\Delta} \mod 6 \right) & \text{if } C_{\max} = R' \end{cases} \tag{2.1}$$

If $\Delta = 0$ (meaning the colour is a shade of gray), set $H = 0$.

4. Compute Saturation (S):

$$S = \begin{cases} 0 & \text{if } C_{\max} = 0 \\ \frac{\Delta}{C_{\max}} & \text{otherwise} \end{cases}$$

5. Compute Value (V): This is the maximum of the normalized B, G, R values. Value determines the brightness and is crucial for understanding how light or dark a colour is.

$$V = C_{\max}$$

2.1.3 Advantages of HSV over RGB

Several studies have highlighted the superiority of HSV over RGB for tasks that require reliable colour differentiation under varied lighting. For instance, a study by A. Ajmal, C. Hollitt, M. Frean and H. Al-Sahaf (2018) [1] demonstrates that HSV-based colour detection systems are more robust to changes in light, reducing the error rate in colour-based sorting applications. This robustness is attributed to HSV's ability to isolate luminance (brightness) from colour information, thereby minimizing the impact of lighting variations on colour perception.

The HSV space simplifies the task of isolating colours through segmentation and thresholding techniques. By working with hue and saturation, it is easier to define ranges for specific colours, which is crucial for sorting capsules where precise colour identification is essential for correct categorisation. Also, HSV aligns more closely with human vision, where colour and brightness are perceived independently, making HSV particularly useful in applications where human visual assessment is the standard, ensuring that the machine's perception of colour closely matches what a human would see.

In summary, the transformation from RGB to HSV is crucial for applications requiring robust and reliable colour detection, particularly in environments with variable lighting and where precise colour differentiation is essential. This makes it an indispensable tool in a project for sorting medical capsules based on colour.

2.1.4 Thresholding in HSV Space

Thresholding in the HSV colour space is a powerful method used in image processing and computer vision to separate objects based on colour. This approach is especially successful because it isolates colour information (hue) from lighting information (saturation and value), allowing for more accurate colour recognition even under changing lighting circumstances.

The HSV components for thresholding are Hue, Saturation and Value (Brightness). Hue represents the colour type on a 360-degree circle, however it is frequently scaled to 0-180 in certain implementations to fit into an 8-bit picture format. Hue is commonly referred to as "colour". Saturation measures the colour's vibrancy or brilliance, where low saturation might be perceived as washed out or closer to gray, but high saturation indicates that the colour is more pure and vibrant.

Finally, Value indicates how light or dark a certain colour is, meaning that a low value makes a colour seem closer to black while a high value means it is closer to white (Hassan and Gutub [4]).

2.1.4.1 Process of Thesholding

Thresholding in HSV space entails choosing a range of hue, saturation, and value that specifies the colour or colours one wishes to isolate. The first step is to convert the picture from RGB (or BGR if using OpenCV) to HSV facilitating the process of separating colours based on their chromatic and brightness qualities. To isolate a specific colour, one needs to define the range of hue values that correspond to that colour. For example:

- Red might be from 0 to 10 degrees and 170 to 180 degrees (adjusting for how hue values wrap around at the red end of the spectrum).
- Green might be from 40 to 80 degrees.
- Blue might range from 100 to 140 degrees.

After establishing the hue range, adjusting the saturation and value ranges helps to refine the choices. For a bold and brilliant green, one may set:

- Saturation from 50% to 100% to avoid very pale greens.
- Value from 50% to 100% to avoid very dark greens.

The figure B.1 is a visual representation of how the HSV colour space is perceived in order to help users understand how to manage these parameters.

Afterwards, apply a threshold based on the provided ranges to generate a binary picture. In this binary picture, pixels within the defined HSV ranges are white (or 1), while all other pixels are black (or 0). The binary image can now function as a mask (B.3) which may be used on the original image to separate the coloured regions that fit the criteria and isolate them from the rest of the image.

The applications of this method are vast, because lighting may have a considerable impact on colour perception in the RGB space and HSV is useful in contexts with changing lighting, such as outdoors or in lighting-controlled industrial applications. Real-world applications must deal with noise as well as colour fluctuations caused by shadows, highlights, and reflections, and adjusting the thresholds to account for these differences is critical for reliable colour detection. And for applications requiring the detection of multiple colours, separate thresholds for each colour can be set up and combined as needed.

2.1.5 Region of Interest (ROI) Processing

Focusing on specific areas of an image, commonly referred to as defining the Region of Interest (ROI), is a highly effective technique in image processing to improve both the speed and accuracy of tasks such as colour detection. This method narrows down the processing area to the most relevant parts of an image, reducing computational load and enhancing the precision of analysis.

2.1.5.1 Benefits of Using Regions of Interest (ROI)

Focusing on a ROI reduces the amount of pixels to process, considerably lowering computational burden. This is especially useful in real-time processing applications like automated quality control systems and robotic vision, where quick decision-making is critical. Processing only the most important bits of an image allows systems to run at greater frame rates and adapt faster to changing surroundings.

ROIs provide more specific analysis, potentially improving the accuracy of colour identification and other image processing tasks. Algorithms can deliver more trustworthy and exact results by identifying regions likely to contain significant information while decreasing background noise and irrelevant stuff. This tailored technique decreases the number of false positives and negatives that might occur when analysing full photos, especially in cluttered or complex settings.

2.1.5.2 Strategies for Defining and Utilizing ROIs

In applications where the objects of interest are consistently located in the same parts of the images, such as on conveyor belts or fixed camera setups, ROIs can be statically defined. This means setting a fixed pixel boundary or rectangle where the system will always focus its processing efforts.

In increasingly sophisticated cases where the objects of interest may alter location, orientation, or scale, dynamic ROIs are required. Before conducting colour detection, preparatory processes like as motion detection, object detection, or other tracking techniques can be used to determine the location of the relevant sections of the picture. For example, a vision system may employ edge detection to locate objects before dynamically setting ROIs around them for thorough colour analysis (Li et al. [5])(Doukas and Maglogiannis [2]).

ROIs may be efficiently integrated with other image processing techniques to improve overall performance. For example, applying filters or morphological operations within ROIs might preprocess the picture to improve characteristics of interest or decrease noise prior to the primary colour identification work (Li et al. [5])(Doukas and Maglogiannis [2]).

In adaptive systems, the results of ROI processing can feed back into the system, allowing it to constantly modify ROI settings. If a colour detection algorithm repeatedly identifies relevant items in a little different location than originally stated, the system may alter the ROI accordingly, increasing efficiency and accuracy over time.

Thus, determining the optimal size and location of an ROI is crucial and can be challenging, especially in varied environments. Incorrect ROI settings might miss important parts of the image or include too much irrelevant information, reducing the effectiveness of the processing.

2.1.6 Colour Accuracy and Calibration

Accurate colour detection is critical in a variety of applications, including quality control in manufacturing and medical diagnostics. To accomplish exact colour identification, the camera and lighting configuration must be calibrated appropriately. This calibration guarantees that the colours obtained in photographs are as similar to the genuine colours of the objects as possible, regardless of the ambient conditions.

2.1.6.1 Importance of Camera and Lighting Calibration

Camera sensors and lighting setups can vary significantly, leading to discrepancies in colour reproduction. This is why calibration is important to make sure that the output remains consistent regardless of the camera or lighting equipment used. Changes in ambient lighting, such as shifting between natural and artificial light sources or varying light intensity throughout the day, may have a substantial impact on how a camera perceives colours. Proper calibration compensates for these differences, ensuring that colour detection remains reliable across a variety of environmental situations. For industries where precise colour differentiation is required, such as pharmaceuticals, textiles, and food processing, calibration is critical for maintaining quality and safety standards. It ensures that the colours detected are accurate, leading to reliable sorting, grading, and quality control.

Calibration typically starts with capturing an image of a colour chart, which contains patches of standardised colours, to help in setting the baseline for colour values that the camera should detect under ideal lighting conditions. Also, it very important to adjust the camera's exposure and gain to minimize overexposure and underexposure, both of which can distort colour perception. Proper exposure ensures that the camera gets the entire spectrum of colours without losing information in very bright or dark regions. Regularly recalibrating the camera and lighting setup is necessary as changes over time in the camera sensors and lights can alter how colours are captured.

The accuracy and reliability of colour identification may be considerably improved by carefully calibrating both the camera and the illumination, as well as taking precautions to reduce environmental influences. This is critical not just for preserving product quality and safety, but also for assuring data integrity in research and other sensitive applications.

2.1.7 Advantages of Cameras over Colour Sensors for colour detection

When sorting capsules by colour, especially those with two distinct hues, using cameras rather than specialized colour sensors provides numerous significant benefits. These advantages arise from cameras' intrinsic capacity to perform complicated visual tasks, versatility, and integration with sophisticated processing technology.

Cameras can catch the whole spectrum of colours in a single frame, allowing them to identify and distinguish several colours on a single capsule. This capacity is critical for capsules with two

unique colours since cameras can reliably distinguish where one colour stops and another begins, which is sometimes difficult for colour sensors that are calibrated to detect particular wavelengths more broadly.

The integration of cameras with powerful image processing software enables comprehensive analysis that extends beyond colour recognition. This involves identifying patterns, forms, and even textures, which are necessary for thorough quality control, for example, using machine learning algorithms, cameras may be trained to identify certain patterns of colour distribution on the capsule, improving sorting accuracy and reliability.

Cameras can adapt to changes in lighting and ambient circumstances by adjusting settings such as exposure, ISO and white balance, to provide consistent operation under varied production settings, which colour sensors typically struggle with because they frequently require stable and regulated illumination to work properly.

Cameras are able to sort by colour as well as do other visual examinations at the same time which enables the identification of additional possible flaws such as malformed capsules, fractures, and incorrect seals, all of which are key quality aspects in pharmaceutical manufacture. Using a single camera system to accomplish numerous inspection functions lowers equipment costs and streamlines production line setup. These systems are scalable and can be modified with additional software to handle other types of capsules or sorting criteria as the product line evolves. This makes cameras a more versatile and future-proof investment than specialized colour sensors.

Given the broader capabilities and lower cost of digital imaging technology, cameras provide a cost-effective option for complicated sorting operations. They offer more capability per unit cost than specialist colour sensors, which may require numerous units or kinds to complete the same job.

Choosing cameras over colour sensors for capsule sorting, particularly multi-coloured ones, provides improved detection capabilities, more operational flexibility, and thorough quality control. The capacity to swiftly adjust to new needs, along with the economic and functional benefits, makes cameras a useful tool in modern production environments where precision, efficiency, and adaptability are critical.

2.2 Computer Vision in Automation

2.2.1 Image Acquisition

When choosing cameras for applications such as sorting capsules, key considerations like resolution, frame rate, and the use of macro lenses play pivotal roles in ensuring optimal performance. Each of these aspects helps the camera create high-quality, detailed pictures that are required for exact identification and sorting of small objects.

The number of pixels in a picture, known as resolution, has a direct impact on the image's detail level, where higher resolutions indicate more pixels, which results in crisper and more detailed

pictures. For tiny things like capsules, great resolution guarantees that even the smallest features are recorded, being critical for properly identifying small changes in colour, shape, or marks on capsules, which are required for proper sorting and quality control. Higher resolutions also require more processing power and storage space, which is why balancing resolution with the processing capabilities of the sorting system and its real-time response needs is essential to ensure efficient operations without sacrificing quality.

Frame rate, or the frequency with which successive pictures or frames are taken, is expressed in frames per second (fps). A greater frame rate enables smoother transitions between pictures, which is especially useful in dynamic scenes where things move quickly, for example in a capsule sorting arrangement, where capsules travel fast along a conveyor belt, a high frame rate guarantees that each capsule is clearly caught despite its velocity. This eliminates motion blur, allowing for accurate colour and form analysis for each capsule. Higher frame rates also means a boost to the system's data flow and processing demand. In order to maintain system balance ensuring that the camera's frame rate is in sync with the conveyor's speed and processing capability is a requirement.

A macro lens is particularly intended to concentrate on things that are very near to the lens, resulting in increased detail and magnification which is perfect for capturing little objects such as capsules. Macro lenses provide extreme close-ups with more clarity and detail and when sorting capsules, even modest changes in colour bands, minuscule writing, or identification symbols may be readily recognized and identified. Macro lenses usually have a small depth of field, which implies that the region in fine focus is fairly limited and while this is useful for separating the subject from a distracting background, it necessitates careful control over the camera's and object's position to achieve consistent focus across all capsules.

Consistent and uniform lighting is also fundamental for effective image processing, particularly in applications like capsule inspection where precise colour and shape detection are crucial. Proper lighting ensures that the camera captures images with minimal shadows and reflections, which can obscure details or distort colours.

Consistent lighting conditions are essential for sustaining colour fidelity over several batches of photos. Variations in illumination can cause considerable differences in how colours are seen and recorded by the camera, potentially leading to inaccuracies in colour-based sorting algorithms. Uniform illumination reduces shadows and glare on the capsules, which can hide important features like text, borders, and colour transitions, while Good lighting improves the overall quality of the image by offering more contrast and clarity which is especially crucial for automated systems that use image processing algorithms to make choices.

2.2.2 Image Processing Algorithms

2.2.2.1 OpenCV

OpenCV (Open Source Computer Vision Library) is an important tool in the field of computer vision, providing comprehensive support for a variety of image processing applications. This library is especially useful in systems such as capsule sorting, where accuracy and efficiency are paramount.

OpenCV helps with colour detection by converting pictures from the RGB/BGR colour space to the HSV colour space. This conversion is critical for capsule sorting because HSV separates colour information (hue) from light intensity (value), allowing for more consistent colour identification under a variety of lighting situations. Following colour space conversion, OpenCV provides tools for implementing colour thresholding, which is required for segregating capsules of various hues. This approach entails selecting a range of hue, saturation, and value to highlight some colours while excluding others, which is especially beneficial for detecting capsules that fit specified colour requirements.

OpenCV offers a number of edge detection techniques, including Sobel and Canny, which are critical for defining each capsule's bounds. Accurate edge detection aids in differentiating separate capsules from one another, which is especially crucial when capsules are adjacent or overlapping. After recognizing edges, OpenCV may detect contours or capsule outlines. These contours are examined to identify the form and size of each capsule, assisting in the sorting process by ensuring that only capsules that satisfy preset shape parameters are acceptable.

OpenCV offers comprehensive support for morphological processes like erosion and dilation. These processes are employed to improve the forms acquired in the image, where erosion removes tiny noise, while dilation restores the size and shape of objects after erosion. In capsule sorting, these modifications aid in cleaning up the picture of capsules, making them more distinguishable for subsequent processing.

OpenCV is meant to be efficient, making it ideal for real-time image processing applications. It can swiftly interpret and analyze pictures recorded from a camera stream, enabling for high-speed capsule sorting with minimal delays. OpenCV works flawlessly with a variety of hardware platforms, including Raspberry Pi, which is frequently used to operate the mechanics of sorting devices. This connection is necessary to synchronize the image processing software with the physical movement of the sorting gear.

Beyond typical image processing, OpenCV offers a variety of machine learning methods. This feature may be used to increase capsule sorting accuracy over time by learning from previous sorting data, detecting subtle trends, and dynamically adjusting the sorting criteria in response to fresh observations. OpenCV's versatility enables developers to tweak and enhance the library's functionality to match the capsule sorting system's specific demands, such as creating new algorithms for recognizing unique capsule traits or flaws.

OpenCV is an indispensable tool in the creation of sophisticated systems. Its wide set of features facilitates critical jobs ranging from basic image modification to advanced machine learning applications. Using OpenCV, developers may ensure that the system processes are not only efficient and rapid, but also resistant to fluctuations in appearance and environmental circumstances. This makes OpenCV an essential component in the quest for more automated, precise, and dependable drug production processes.

2.2.2.2 Edge Detection

Edge detection is a key approach in image processing that identifies locations in a digital picture where the brightness of pixel intensities fluctuates dramatically. This approach is critical in applications such as capsule inspection, where exact item boundaries are required for counting, sorting, and quality control. The Canny edge detector and the Sobel operator are two of the most used edge detection methods.

The Sobel operator is used to calculate the gradient of picture intensity at each pixel. It calculates the gradient of picture intensity at each pixel in an image, determining the direction of the largest possible rise from light to dark and the rate of change in that direction. The Sobel operator employs two 3x3 convolution masks (kernels), one for horizontal changes (Sobel X) and another for vertical changes (Sobel Y). The kernels are applied to the picture to compute derivative approximations, one for horizontal changes (Gx) and one for vertical changes (Gy). Ideal for applications that do not require great edge sensitivity and is simpler and quicker, making it appropriate for real-time processing applications requiring rapid edge identification without comprehensive refining, such as initial edge detection in capsule inspection (Vincent, Folorunso, et al. [13]).

The Canny edge detector is a multistage algorithm that detects a broad variety of edges in photos. It is particularly notable for its ability to identify weak edges while being less susceptible to noise than other approaches. Typically, the picture is smoothed first with Gaussian blurring to decrease noise and prevent erroneous edge detection. The gradient magnitude and direction are determined using more exact approaches, which may involve the Sobel operator. To thin out the edges, the method employs non-maximum suppression, which refines all possible edges to thin lines by suppressing all gradient values (making them zero) except for local maxima, which represent areas with the sharpest shift in intensity values. Canny employs two thresholds to distinguish between strong and weak edges. Strong edges have an intensity gradient that exceeds the high threshold, whereas weak edges are those that are above the low threshold but below the high threshold. Finally, weak edges that are related to strong edges are kept, but all other weak edges are deleted. Highly effective for activities demanding high precision and noise resistance, making it perfect for thorough and accurate boundary detection in applications such as capsule inspection, where precise edges are crucial for successful sorting or quality control (Xu, Baojie, and Guoxin [14])(Rong et al. [8])(Lu et al. [6]).

Both the Sobel operator and the Canny edge detector are useful tools for edge detection in image processing applications. The decision between them is determined by the application's

unique needs for speed, precision, and noise robustness. For separating touched capsules, Canny's approach may be better because to its resilience and accuracy, while Sobel's simplicity and quickness make it suitable for rapid or less extensive inspections.

2.2.2.3 Morphological Operations

Erosion and Dilation are important morphological procedures in image processing that refine observed forms within an image. These techniques are especially important in applications such as capsule sorting, where exact form determination may dramatically enhance sorting accuracy.

Erosion is a morphological process that removes the borders of a foreground item (often white). It is used to minimize the size of foreground items and is implemented by involving a kernel (a tiny shape or template known as a structuring element) over the picture, retaining just the pixel at the center of the kernel if all pixels under the kernel match the structural element. It operates by moving the kernel over the image and checking if all of the pixels under the kernel match the structuring element at each place. If they are, the pixel in the kernel's center is set to white (or keeps its value); otherwise, it is assigned to black (or changes). This procedure eliminates small-scale noise and reduces the size of things. It can also separate barely touching items, which is essential for recognising and sorting capsules that may cluster together (Said and Jambek [9])(Soille [11]).

Dilation is the opposite of erosion; it adds pixels to the edges of items in a picture and is commonly used to enhance the size of foreground objects. This procedure likewise uses a kernel that is involved with the picture, but unlike erosion, a pixel at the center of the kernel is set to white if at least one pixel below it is white. As the kernel advances across the image, if any pixel underneath it is white, the pixel in the kernel's center is set to white. This extends the sections with white pixels. Dilation is used to highlight features and fill minor gaps inside objects. It can also expand the size of items that have been too decreased due to erosion (Said and Jambek [9])(Soille [11]).

These two techniques have a wide range of applications, including capsule sorting. For example, erosion can assist separate capsules that are lightly touching or overlapping by thinning the connecting regions, as well as eliminating any minute particles or flaws that are not part of the capsules, ensuring that such noise does not interfere with sorting accuracy. After erosion, dilatation can help fill tiny gaps within objects or between linked components of an item, which can be advantageous if erosion has damaged sections of a capsule's perimeter. Dilation may also make narrow features bigger and more noticeable for further processing, which is useful for feature extraction tasks in image analysis.

Thus, erosion and dilation efficiently improve the shapes observed in pictures, boosting sorting accuracy by guaranteeing that individual capsules are clearly identified and quantified without influence from noise or slight interactions with other capsules. These processes are critical tools

in the preprocessing phases of image analysis for applications requiring great accuracy, such as pharmaceutical capsule sorting.

2.2.3 Machine Vision vs. Human Inspection

When it comes to jobs like continuous operation in industrial settings, both machine vision systems and human visual inspection have different benefits and limits.

Machine vision systems thrive at producing consistent and dependable outcomes, particularly in controlled situations when activities are repeated. These systems examine and interpret pictures using cameras and image processing algorithms, allowing them to do the same tasks consistently over time. Machine vision is not affected by the physical or psychological states that people experience, such as exhaustion or loss of attention. It can reliably identify minute characteristics that human inspectors frequently overlook, such as minor flaws in items or subtle colour differences that are crucial in quality control operations. However, while machine vision is very consistent, its dependability can be jeopardised by changes in the environment that have not been included into its programming, such as unexpected illumination changes or new product varieties that it has not been trained to detect.

Machine vision systems can handle thousands of items per hour, outpacing human capabilities in terms of speed, particularly in applications requiring quick visual assessment, such as sorting or assembly line inspections. Machine vision systems require initial configuration and training, especially when utilizing AI or ML algorithms. They require vast amounts of training data to achieve high reliability, and adjusting to new sorts of tasks may necessitate extensive reprogramming and recalibration.

Humans thrive at tasks that demand flexible thinking, adaptability to new contexts, and cognitive breakthroughs based on unexpected discoveries. Human inspectors can rapidly adapt to new product kinds or changes in quality standards, eliminating the need for costly training. Humans can utilize their judgment to overlook little faults that are insignificant, allowing them to see the context or larger picture in ways that machine vision cannot currently match. Human inspection is valuable because of its capacity to analyse complicated and unstructured situations and is naturally less constant and trustworthy in repetitive operations. Fatigue, boredom, and psychological issues can all contribute to performance variability, lowering inspection effectiveness over time or in high-throughput circumstances.

Human performance can degrade over time due to fatigue, which affects visual acuity and concentration. In continuous operations, this deterioration might result in higher mistake rates and inspection quality fluctuation. Each human inspector may have a somewhat varied threshold for what constitutes a flaw, resulting in discrepancies in inspection results across persons or even the same individual at various times (Sylla [12]).

To summarize, machine vision systems offer improved consistency, speed, and reliability for repeated and high-speed activities, making them vital in current industrial environments where

accuracy and efficiency are critical. They are especially successful in settings where variables can be controlled and activities do not need high degrees of cognitive judgment.

Human inspectors, on the other hand, provide unparalleled flexibility and cognitive understanding, which is especially useful in less organized contexts or when decision-making requires complicated criteria that current AI cannot fully mimic. However, their vulnerability to fatigue and unpredictability presents a substantial disadvantage in continuous operations.

The decision between machine vision and human inspection is frequently influenced by the task's unique needs and the trade-off between speed and consistency compared to flexibility and cognitive understanding. In many circumstances, combining the two systems, with robots handling the majority of repetitive labor and people managing quality assurance and complicated decision-making, may provide the best of both worlds.

Chapter 3

System Concept

3.1 System Overview

The capsule sorting system is a complex automated technology that simplifies the process of sorting pharmaceutical capsules based on colour, size and shape. Using modern image processing technology such as computer vision algorithms and machine learning capabilities, the system improves the efficiency and accuracy of pharmaceutical production operations.

The major goal of this system is to guarantee that pharmaceutical capsules are correctly sorted according to established requirements. This is key for ensuring uniformity in medicine dose and presentation, which is essential for patient safety and regulatory compliance. The system is intended to identify, count and separate capsules to reduce the likelihood of incorrect medicine packing.

In the pharmaceutical sector, accuracy and reliability are essential. Capsule sorting errors can pose considerable health hazards to patients while also causing significant financial and reputational damage to manufacturers. While manual inspection is successful, it is labor-intensive and prone to human mistake, particularly when faced with high-volume manufacturing needs.

The capsule sorting system solves these problems by automating the sorting process to not only minimize the need for manual labor, but also to improve throughput and output uniformity. Automation in capsule sorting may considerably reduce the danger of human mistake while increasing production speed and lowering operating expenses. It also enables pharmaceutical businesses to devote more human resources to important quality control and oversight responsibilities, hence increasing overall productivity and safety.

The introduction of an automated capsule sorting system is particularly responsive to several industry-specific challenges:

1. **Quality Assurance**: Automated sorting guarantees that each capsule is inspected in accordance with standardized criteria, hence enhancing overall product quality.

2. **Regulatory Compliance**: By delivering accurate sorting based on exact criteria, the technology assists producers in meeting severe regulatory standards for pharmaceuticals.

3. **Operational Efficiency**: Automation speeds up the capsule sorting process, allowing for shorter production cycles and prompt fulfillment of market needs.

4. **Scalability**: As production numbers expand, the system may be scaled up to accommodate more output while maintaining quality and efficiency.

The capsule sorting system is a game-changing development in pharmaceutical production, combining cutting-edge technology with industry requirements to create a high-performance solution. Its creation and deployment mark a significant step forward in the industry's continuous attempts to adopt automation, which ensures drug safety and efficacy while improving production procedures.

3.2 Technological Framework

The capsule sorting system combines many critical technologies to ensure an efficient and precise operation. This section will go over the key components, which include OpenCV for image processing, the Raspberry Pi 3 as the control unit, a high-quality Raspberry Pi camera with a macro lens, a conveyor belt mechanism for transporting capsules, a lighting system to improve image capture, and a Nextion display for user interaction.

The Raspberry Pi was chosen over other control units like the Arduino and the ESP32, for example, since the Raspberry Pi 3 has a Quad-Core ARM Cortex-A53 CPU and 1 GB of RAM, making it far more powerful than an Arduino, which typically has an 8-bit microcontroller (such as the Atmega328 in the Arduino Uno). This increased processing capability enables the Raspberry Pi to undertake demanding activities like real-time image processing with OpenCV. In contrast, the Arduino lacks the processing capability for activities like real-time colour identification and sorting, which would need other modules and increased programming complexity. The Raspberry Pi 3 has several communication choices, including HDMI, USB ports, GPIO pins, Ethernet, and a specialized Camera Serial Interface (CSI). This makes it suitable for incorporating peripherals such as the HQ camera and Nextion display. On the other hand, Arduino boards frequently lack built-in network connectivity, and sophisticated capabilities necessitate additional components, increasing the system's complexity. For the ESP32, while it has built-in Wi-Fi and Bluetooth and is a fantastic microcontroller for Internet of Things (IoT) applications, it lacks the same amount of software ecosystem and community support for image processing as the Raspberry Pi 3. The Raspberry Pi's comprehensive library support for OpenCV and other Python-based software tools simplifies development, making it a more practical solution for image-based activities such as sorting medicinal capsules.

The Nextion display was selected because, unlike standard TFT LCDs, has its own inbuilt microprocessor that handles user interface (UI) processing chores, freeing them from the Raspberry Pi, which frees up the Raspberry Pi's processing resources for more important activities such as picture analysis and capsule sorting. Standard TFT LCDs would need more programming and resources on the Raspberry Pi to handle UI chores, potentially slowing down the system. The Nextion display has built-in touchscreen functionality, which makes the user interface more engaging and accessible. Basic TFT displays without touch capabilities would require extra buttons or peripherals for interaction, increasing the complexity of the design. Also, compared to the OLED displays, the Nextion display features proprietary software for building personalized graphical user interfaces (GUIs). This provides for a great degree of freedom in the interface design, which is especially optimized for capsule sorting and monitoring. OLEDs lack this GUI capabilities, making them less appropriate for complicated interactions and real-time data display.

About the camera of choice, the Raspberry Pi HQ Camera's 12-megapixel Sony IMX477 sensor provides a much improved resolution and image quality than the Raspberry Pi Camera Module V3. The HQ Camera's ability to withstand low-light circumstances and produce clearer macro photos makes it perfect for close-up inspections of medicinal capsules, where minor changes in colour and form must be detected. The V3 module struggled with focus and clarity in this application, resulting in misidentification of capsule colours. The HQ Camera has interchangeable lenses, including the 6mm 3MP macro lens used in this project, which enables for more exact focus on small objects like capsules. The V3 module has a fixed lens, limiting its versatility for activities needing varied focal lengths. The ability to utilize a macro lens was vital for getting the necessary amount of detail. Furthermore, the HQ Camera connects directly to the Raspberry Pi via the Camera Serial Interface (CSI), enabling for faster data transfers and better real-time processing than ordinary USB cameras. USB cameras frequently cause delay, which might be inefficient for real-time sorting operations. Direct CSI connection makes the system more dependable and responsive.

Finally, for the Conveyor Belt System, a continuous belt system was chosen because it can deliver medicine capsules smoothly and reliably. This design reduces the likelihood of capsules turning over or becoming misaligned during transportation. Unlike roller or chain conveyors, a continuous belt provides a uniform surface, ensuring that capsules travel in a regulated manner through the sorting system, which is critical for proper image capture and classification by the camera. The conveyor belt moves at a speed of 50 mm/second. This speed is suitable for syncing the capsules' movements with the camera's picture processor. The selected speed balances throughput and sorting accuracy, allowing the system to handle capsules swiftly without losing precision. The motor was chosen to provide enough power and torque to operate the conveyor belt system at the appropriate speed while carrying the capsules. It has a voltage of 24V, a speed of 30 RPM, a current of 2A (limited by the L298N motor driver) and provides around 15.29 Nm of torque, which is enough to move the conveyor belt at 50 mm/sec. This torque guarantees that the motor can overcome the frictional forces in the system and carry the capsules without stalling,

resulting in smooth, constant functioning. A motor driver, the L298N, was also selected was selected to control the 24V motor. The L298N supports a voltage input of up to 46V, making it compatible with the 24V motor and it can supply a maximum current of 2A per channel. It also allows precise motor control utilising Pulse Width Modulation (PWM) signals. This control guarantees that the conveyor belt may be started, stopped and speed-adjusted smoothly in accordance with the system's operating needs. To avoid capsule sorting mistakes, the belt's movement must be synchronized with the image processing system using accurate speed control.

Here's how these components interact to form an efficient capsule sorting system:

1. **Raspberry Pi**
    - **Role and Capabilities**: The Raspberry Pi is the capsule sorting system's core control unit. It is a strong yet low-cost minicomputer that can handle a variety of tasks, including image processing and peripheral device control.
    - **Integration**: The Raspberry Pi 3 handles all computational tasks, including running the OpenCV software for image analysis, controlling the speed and operation of the conveyor belt, managing the lighting system, and interfacing with the Nextion display.
2. **HQ PiCamera with Macro Lens**
    - **High-Quality Image Capture**: The Raspberry Pi High Quality Camera is equipped with a macro lens that was particularly chosen for its ability to take detailed photographs of tiny objects at close range. This is crucial for correctly detecting the colours and shapes of different capsules.
    - **Functionality**: The camera continuously captures images of the capsules as they move past the inspection area on the conveyor belt. The photos are then analyzed in real time to sort the capsules according to predetermined parameters.
3. **OpenCV for Image Processing**
    - **Image Processing Software**: OpenCV (Open Source Computer Vision Library) is used for the essential image processing tasks within the system that includes converting the images from the camera from RGB to HSV colour space, which is particularly effective for colour detection in varying lighting conditions.
    - **Applications**: OpenCV algorithms perform tasks such as colour thresholding to separate capsules of various hues and edge detection to determine each capsule's borders. These actions are critical for accurately sorting the capsules.
4. **Conveyor Belt**
    - **Transport Mechanism**: The conveyor belt is the actual device that moves the capsules past the camera for inspection. It is crucial for maintaining a steady flow of capsules and presenting them to the camera in a regulated manner.

- **Control and Synchronization**: The Raspberry Pi regulates the conveyor belt's speed, syncing it with the camera's frame rate to guarantee that each capsule is captured under ideal conditions for precise sorting.

5. **Lighting System**

   - **LED Lighting**: A breadboard-mounted set of white LEDs illuminates the capsules as they move through the inspection area. The use of LEDs provides bright, consistent, and energy-efficient lighting.
   - **Role in Image Quality**: Proper illumination is essential for high-quality image capture. The white LEDs keep the capsules consistently lit, reducing shadows and reflections that may interfere with image analysis.

6. **Nextion Display**

   - **User Interface**: The Nextion display is incorporated into the capsule sorting mechanism, creating an interactive user interface. This touchscreen display enables users to directly manage the system and alter settings without the need for an external computer.
   - **Functionality**: The display is utilised for a variety of important operations, including starting and halting the conveyor belt and defining colour thresholds for sorting. It also shows real-time data which improve user participation and control over the sorting process.

7. **System Integration**

   - **Cohesive Operation**: All of these components are combined within a wooden box that acts as the system's structural framework. This shell not only holds the hardware, but it also protects the capsules from external light and interference, improving image processing accuracy.
   - **Data Flow and Processing**: The Raspberry Pi takes image data from the camera, processes it using OpenCV, and then utilizes the results to drive the sorting mechanism.

The capsule sorting system's technical architecture is optimized for efficiency and accuracy. The system delivers high-speed, dependable pharmaceutical capsule sorting by combining the computing power of the Raspberry Pi 3, the precision of the HQ PiCamera with a macro lens, OpenCV's powerful image processing capabilities, and the interactive Nextion display.

3.3 System Breakdown Structure

In order to better explain each sub-part of the system, figure 3.1 has a detailed breakdown:

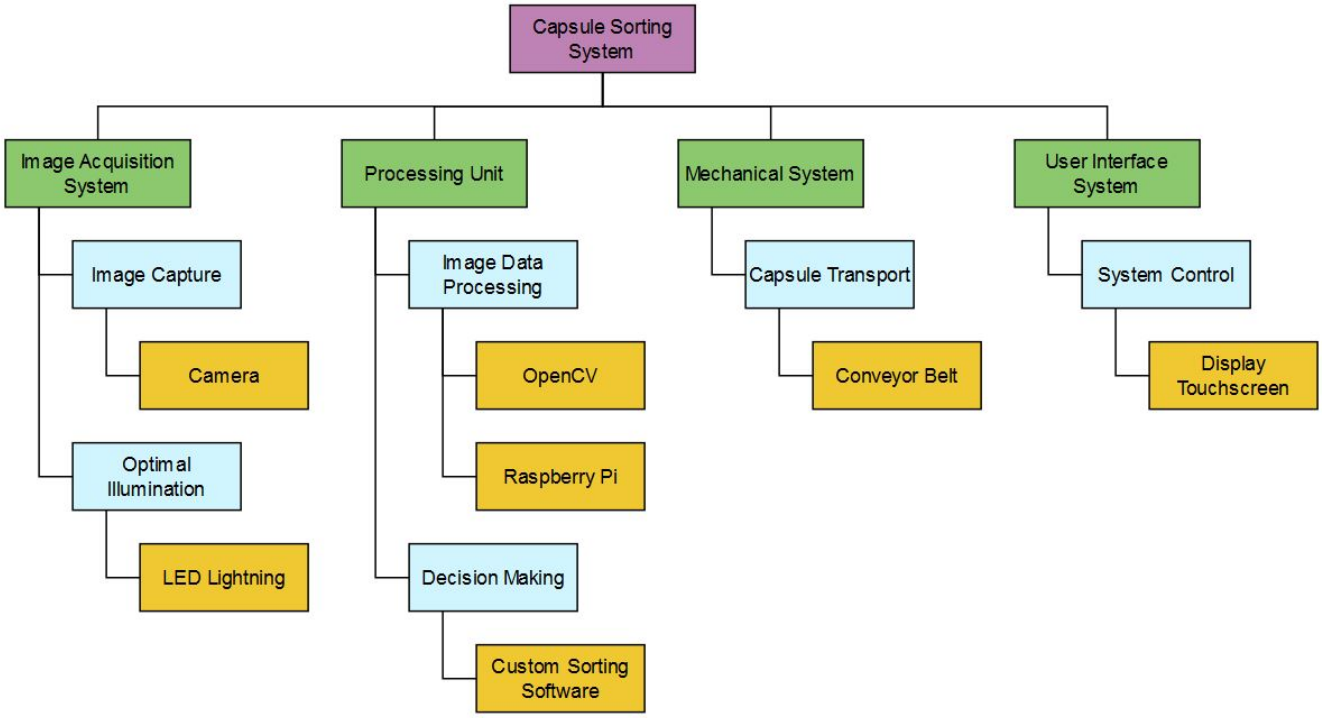


Figure 3.1: System Breakdown Structure diagram.

Each colour represents a different level of abstraction.

- Purple - 1st High-level representation of the product
- Green - 2nd level of abstraction, showcasing different subsystems
- Blue - 3rd level of abstraction, referring to particular activities or objectives of each subsystem.
- Yellow - 4th and lowest level of abstraction, outlining the many technologies/components to fulfill the purpose of the 3rd level.

3.4 Functional Architecture

Functional architecture defines the flow of information and processes between a system's hardware and software components. It describes how hardware sensors gather and analyse data, how software algorithms process that data, and how the results are communicated to the user. By breaking down the system into discrete functions and describing their interactions, one is able to gain a clearer understanding of how the complete system worked. With this in mind, the following diagram was created.

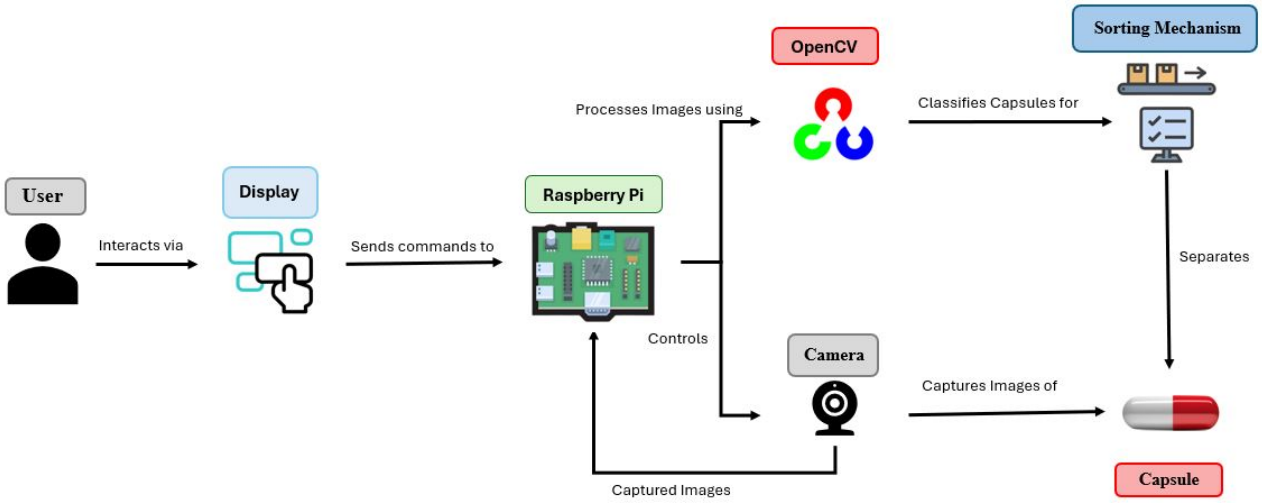


Figure 3.2: Functional Architecture

1. **User Interaction**

   - **Starting Point**: The user interacts with the system through the Nextion display. This interaction includes starting the sorting process (both colour sorting and counting), adjusting settings, and stopping the process based on user input or system feedback.
   - **Control Commands**: The user sends commands to the Raspberry Pi via the display. These commands can include operational instructions (start/stop) and parameter adjustments (e.g., changing threshold values for image processing).

2. **Raspberry Pi**

   - **Central Processing Unit**: The Raspberry Pi works as the system's brain, accepting user commands and processing image data by controlling the camera to take capsule photos and processing them using the OpenCV library.
   - **Decision Making**: After processing the images, the Raspberry Pi determines how each capsule should be sorted based on predefined criteria.

3. **Camera**

   - **Image Capture**: The camera, which has a macro lens for precise close-ups, takes pictures of the capsules as they travel along the conveyor belt. These photos are immediately transmitted to the Raspberry Pi for examination.
   - **Continuous Operation**: The camera continuously captures images, ensuring that every capsule is analysed without missing any.

4. **OpenCV**

- **Image Processing**: OpenCV processes the images received from the camera which involves converting the colour space, applying filters for noise reduction, and using algorithms to detect edges, shapes, and colours.
- **Classification**: OpenCV classifies each capsule based on the analysis, determining the characteristics that dictate the sorting decision.

5. **Sorting Mechanism**

   - **Action Execution**: Based on the Raspberry Pi's decisions, the sorting mechanism physically sorts the capsules which might include leading the capsules to various bins or channels.
   - **Final Output**: The sorted capsules are the output of the system, each correctly placed according to its classification.

3.5 KPIs (Key Performance Indicators)

To ensure the Capsule Sorting System's success in pharmacy contexts, clear, measurable Key Performance Indicators (KPIs) must be established. These KPIs are critical for evaluating the system's efficiency, accuracy, and overall influence on pharmacy operations, enabling collaborators to measure the system's benefits, track its effectiveness, and lead ongoing improvement efforts. The following are the key performance metrics necessary for assessing the efficacy of the Capsule Sorting System project:

1. **Accuracy Rate of Capsule Sorting**: This KPI calculates the percentage of capsules appropriately sorted by colour and by quantity compared to the total number of capsules processed. It is a clear measure of the system's ability to differentiate and categorise capsules, which is critical to ensure patient safety and prescription compliance.

2. **Throughput Rate**: Throughput rate refers to the number of capsules sorted per unit of time (e.g., per hour). This KPI is essential for determining the system's capacity to handle the volume of work typical in a pharmacy setting and its ability to improve operational efficiency.

3. **Downtime and Reliability**: Measured as the percentage of time the system is operational versus the time it is unavailable due to maintenance or malfunctions. This KPI is critical for evaluating the system's reliability and its impact on pharmacy operations, particularly in high-volume environments where downtime can significantly affect service delivery.

4. **User Satisfaction**: User satisfaction among pharmacy workers who engage with the system is measured via surveys and feedback systems. This qualitative KPI evaluates the system's usability, integration with current workflows, and influence on employee happiness.

5. **Error Reduction Rate**: This KPI tracks the reduction in distribution errors related to incorrect capsule colour sorting before and after system's implementation, highlighting the system's effectiveness in enhancing patient safety and reducing the risk of medication errors.

6. **Operational Cost Savings**: Operational cost savings are calculated by reducing the number of worker hours necessary for manual sorting, the frequency of mistakes (and the costs associated with fixing them), and other efficiency advantages. This KPI highlights the system's involvement in optimizing resource allocation in pharmacies.

7. **Power Consumption**: Monitors the amount of electrical energy consumed in Watts (W) by the system during operation, emphasising energy efficiency and cost-effectiveness.

8. **Compliance Rate with Regulatory Standards**: This KPI measures the system's adherence to healthcare and pharmaceutical regulations, especially those related to medication handling and patient safety. Compliance is essential for legal and ethical operations in the healthcare industry/business.

By monitoring these KPIs, stakeholders (collaborators) can effectively evaluate the performance and impact of the automated Capsule Sorting System. This evaluation not only demonstrates the value of the system to pharmacy operations but also guides strategic decisions and improvements to ensure the system meets the evolving needs of the healthcare sector.

3.6 Scalability and Maintenance

To achieve long-term performance and sustainability in pharmaceutical applications, the capsule sorting system must be both scalable and simple to maintain. The design contains aspects that simplify scaling, make maintenance easier, and increase system reliability.

The system is modularly constructed, with components like the camera, conveyor belt, and sorting processes designed to be autonomous but interoperable. This modular architecture enables for easy extension or improvements, such as adding more cameras or extending the conveyor system, without causing substantial interruptions to the present arrangement. This adaptability allows for increasing production demands or adaption to varied capsule kinds and sizes.

The software architecture, which includes image processing and machine control algorithms, is built for future upgrades. The system is built on a common, open-source platform (for example, OpenCV on Raspberry Pi), making it easy to integrate new software features and upgrades. As new sorting algorithms or efficiency improvements become available, they may be effortlessly loaded into the system, ensuring that it remains at the forefront of technology without necessitating major hardware upgrades.

The camera lens and conveyor belts, two components that require frequent maintenance, are easily accessible. This design considerably minimizes the time and effort needed for common maintenance tasks including cleaning, adjustment, and part replacement.

Whenever feasible, the system use standardized, commercially accessible components rather than bespoke ones. Standardized parts guarantee that replacements are widely available and often less expensive than custom components. This also streamlines the procurement process and lowers maintenance downtime.

To address the changing demands of pharmaceutical manufacturing, the capsule sorting system is designed with scalability and ease of maintenance in mind. By integrating modular components, assuring software upgrade ability reducing maintenance procedures, and automated processes, the system is well-prepared to ensure long-term, reliable performance while responding to future demands.

3.7 Risk Assessment

A risk is defined as any incident that has a negative impact on the system's operation. A thorough risk assessment is required to ensure the capsule sorting system's reliability and safety, as it aids in the identification of possible failure spots and the mitigation of hazards that might impair system performance and safety.

To build an evaluation system that quantifies each risk, a grading system was developed based on the likelihood of the problem and its impact on the project. This score ranges from 1 to 5, and it may be used to conduct a Risk Management Assessment presented on the table 3.2.

Table 3.1: Risk matrix

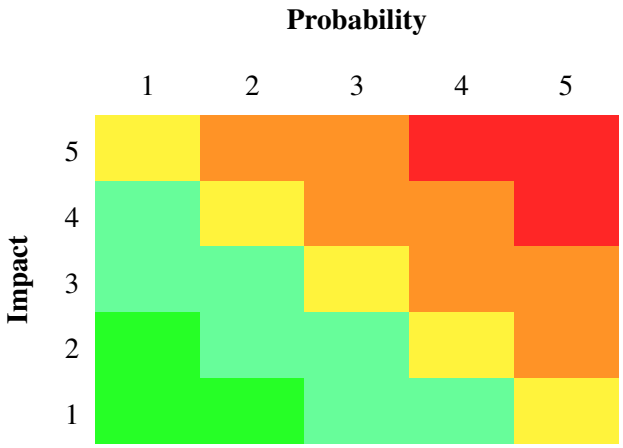


Table 3.2: Risk management assessment

Due to	Risks	Impact	Prob.	Solution
Mechanical Failure	Mechanical parts such as conveyor belts or sorting mechanisms can fail due to wear and tear, improper alignment, or mechanical overload.	4	3	Implement a routine maintenance schedule to inspect, clean, and repair mechanical components.
Software Errors	Software glitches, bugs, or system crashes could lead to incorrect sorting, system stoppages, or data losses.	4	2	Conduct thorough testing phases, integration tests, and system-level stress tests to identify and fix bugs before deployment.
Camera Failure	Failures in the camera or image processing algorithms could result in inaccurate capsule sorting due to poor image quality or misinterpretation of data.	3	2	Use high-resolution cameras and reliable lighting systems to ensure consistent image quality. Regularly validate and calibrate image processing algorithms to maintain accuracy.
	Camera doesn't meet standards	5	3	Find alternatives as soon as possible.
User Error	Incorrect operation by users due to misunderstanding or errors in judgment could lead to system misuse or operational inefficiency.	3	4	Hold regular feedback sessions with users to learn about potential confusion or operational difficulties and adjust training or interface design accordingly.

From all this risks the most impactful was the fact that the first camera chosen didn't have the appropriate qualities to this specific project and thus a different camera had to be selected as soon as possible in order to accomplish the project's objectives.

A proactive risk management approach is crucial for any project's, in this case the capsule sorting system's, successful deployment and operation. By recognizing possible risks and adopting strategic mitigations, the system may achieve high reliability, maintain safety standards, and meet regulatory requirements.

Chapter 4

Methodology

This chapter describes the extensive approaches used in the development, assembly, and validation of an automatic capsule sorting system intended to improve efficiency and accuracy in pharmaceutical operations. The emphasis here is on a full explanation of the technical and operational procedures that support the whole project. This chapter aims to provide a clear blueprint of how the capsule sorting system was brought from conceptualization to functional reality, from the initial system design incorporating both hardware and software elements, to the intricate processes of assembly and rigorous phases of testing and validation.

4.1 Hardware Setup

The hardware setup of the capsule sorting system is a critical component of its overall functionality, designed to ensure reliability, efficiency, and ease of use in pharmaceutical environments. This section details each component that has been carefully selected and integrated into the system based on specific operational requirements and technical specifications.

4.1.1 Micro Computer

Serving as the central control unit, the Raspberry Pi 3 was chosen for its robust processing capabilities, extensive community support, and compatibility with various peripherals. It features a Quad-Core ARM Cortex-A53 processor, 1 GB of RAM, and comprehensive connectivity options including HDMI, USB ports, Ethernet, GPIO pins which facilitate direct control over peripheral devices, and it has a Camera Serial Interface (CSI) port that will be fundamental to establish connection with the camera of choice. It interfaces with the camera to process images of the capsules, controls the conveyor belt speed and direction, manages the LED lighting system, and communicates with the user interface for system monitoring and adjustments. The choice of the Raspberry Pi 3 allows for sophisticated image processing tasks that are crucial for the accurate sorting of capsules. This includes using libraries such as OpenCV (Richardson and Wallace [7])(Halfacree and Upton [3]).

The selection of the Raspberry Pi 3 for the capsule sorting system is justified by its ability to handle demanding image processing tasks, its compatibility with essential programming libraries like OpenCV, and its comprehensive connectivity options that facilitate control over various hardware components. This micro computer provides a strong foundation for a reliable, efficient, and versatile sorting system.

Figure 4.1: Raspberry Pi

4.1.2 Camera

The camera selection is critical in the capsule sorting system because it captures precise photos of the capsules, which are required for accurate sorting. The system utilises a high quality 12-megapixel Raspberry Pi Camera with a 6mm 3MP macro lens designed for close-up shooting.

This piece of equipment offers a 12 megapixels resolution, proving images that capture fine details of each capsule, essential for precise colour and shape analysis. The camera uses a Sony IMX477 sensor, which is noted for its high picture quality and low-light capabilities, resulting in crisp captures under a variety of lighting circumstances. It also comes with a 6mm 3MP macro lens that was particularly developed for close-range photography enabling the camera to focus on small items, such as medicine capsules, and capture the fine details required for effective sorting.

The camera connects to the Raspberry Pi via the Camera Serial Interface (CSI), which supports high data transfer rates, allowing real-time picture processing without latency/lag. Focus and aperture may be manually adjusted to suit specific imaging needs, providing flexibility in capturing images under a variety of operational conditions.

The camera is positioned directly above the conveyor belt and collects high-resolution photos of capsules as they travel through the specified inspection zone. These pictures are essential to the future image processing procedures. The macro lens's ability to capture detailed close-ups ensures that the system can accurately analyse the features of each capsule.

(a) HQ Camera (b) Macro Lens

Figure 4.2: Camera Composition

4.1.3 LEDs

High-brightness white LEDs, more specifically two, are arranged around the camera to provide consistent, uniform, shadow-free lighting to avoid distorted image data. This setup is crucial for capturing high-quality images regardless of ambient lighting conditions. The LEDs are embedded in the frame and wired to the Raspberry Pi involving resitors, allowing software control over their intensity and activation.

Proper illumination is required for the camera to effectively identify colours, which is the foundation for sorting decisions. Inconsistent or bad lighting can cause colour misidentification, leading in sorting problems. Well-lit images minimize the processing workload by reducing the requirement for correction algorithms to adjust for lighting inadequacies. This efficiency is critical for sustaining high throughput rates in industrial environments.

Figure 4.3: Ultra-bright white LEDs

4.1.4 Conveyor Belt

The conveyor belt system is an essential piece of the capsule sorting system, ensuring the controlled transit of capsules through the inspection zone for image collection and subsequently to the sorting mechanism. This system is driven by an L298N H-bridge motor driver and has a 24-volt DC motor with a maximum speed of 50 mm/sec, which is necessary for precise capsule sorting based on colour. However, for capsule counting activities, the speed is set to roughly

31 mm/sec (62% of maximum speed) to maximize counting accuracy. The L298N motor driver provides exact control over the motor's speed and direction via a PWM signal supplied by the Raspberry Pi, which is critical for ensuring that capsules are moved at the right speed for accurate imaging and sorting. This movement must be smooth and steady to avoid dislodging the capsules from their intended course. The motor driver optimises the system by adjusting the speed of the conveyor belt, ensuring that capsules are given to the camera at the ideal pace for continuous picture processing. This synchronization is critical for achieving high throughput while preserving accuracy.

(a) Conveyor Belt (b) L298N Driver

Figure 4.4: Conveyor Belt System

4.1.5 Sorting System

The capsule sorting system uses a novel LED-based sorting mechanism to efficiently identify capsules based on colour combinations. This technology uses visual feedback from LEDs to inform operators about sorting actions and decisions, simplifying system operation while maintaining high accuracy and efficiency.

The sorting mechanism consists of six LED indicators, one for each of the machine's six exits. These LEDs are used to graphically show the various hue combinations of capsules as they are handled. When a capsule is initially identified, its colour is determined and assigned to one of the various LED indicators. This assignment is based on the order in which colours are detected, with the first detected colour allocated to the first LED, and so on. After assigning a colour to an LED, each succeeding capsule with the same colour combination causes the corresponding LED to blink. The flashing indicates that the capsule has been appropriately detected and is being sorted based on its colour classification.

When the colour of a capsule is detected, the appropriate LED indicator blinks to confirm the sorting activity. This fast visual input is critical for monitoring system performance and guaranteeing accurate sorting processes. The system constantly allocates new colour combinations to available LEDs until all six are used. If more than six colour combinations are discovered, the

system can cycle through or prioritize them according to predetermined parameters. If a colour is recognized that does not match any assigned LED (after all LEDs have been used), the system can notify the operator, prompting manual involvement or decision-making. While the system originally supports up to six colour combinations, its architecture allows for simple scalability. Additional LED indicators and exits can be integrated to accommodate more colours as needed, providing flexibility for future expansions.

4.1.6 Display

The integration of the Nextion NX4832T035 HMI (Human-Machine Interface) is also an aspect of the capsule sorting system, providing a user-friendly interface for system control and monitoring. This 3.5-inch TFT LCD touchscreen is specifically chosen for its reliability, ease of use, and robust functionality.

This display has a resolution of 480 x 320 pixels, which offers clear and sharp visuals for interface elements and text, is a resistive touch panel that supports interaction through direct touch inputs, facilitating an intuitive user experience, has a serial communication interface, simplifying connectivity with the Raspberry Pi and minimizing wiring complexity and includes proprietary software that allows for the design of custom user interfaces, which can be updated or modified according to user requirements.

The Nextion display enhances system usability with its graphical interface, which is more intuitive than traditional button-based controls. This reduces training time and increases user comfort. Immediate access to system controls and configurations through the HMI speeds up operational adjustments and troubleshooting, leading to higher overall efficiency and by handling user interface processing on its own, the Nextion display offloads tasks from the Raspberry Pi, allowing it to focus more on image processing and control logic.

Figure 4.5: Display

4.1.7 Power Supply

The power supply is a important component of the capsule sorting system, delivering consistent and dependable power to all electrical and mechanical components. The Raspberry Pi and the conveyor belt motor require different voltages and currents, hence the system uses two independent power supply units.

The power supply for the Raspberry Pi and other components was a standard 5V/3A power supply, commonly used with Raspberry Pi models. The power supply is small and compact, comparable to conventional mobile phone chargers, and it fits smoothly into the system without taking up extra room. It has critical safety features including overvoltage, overcurrent, and short-circuit protection, which ensure the safety of electronic components. It connects directly to the Raspberry Pi via a USB connector, simplifying cabling, and then distributes power to other low-power components like LEDs and the camera.

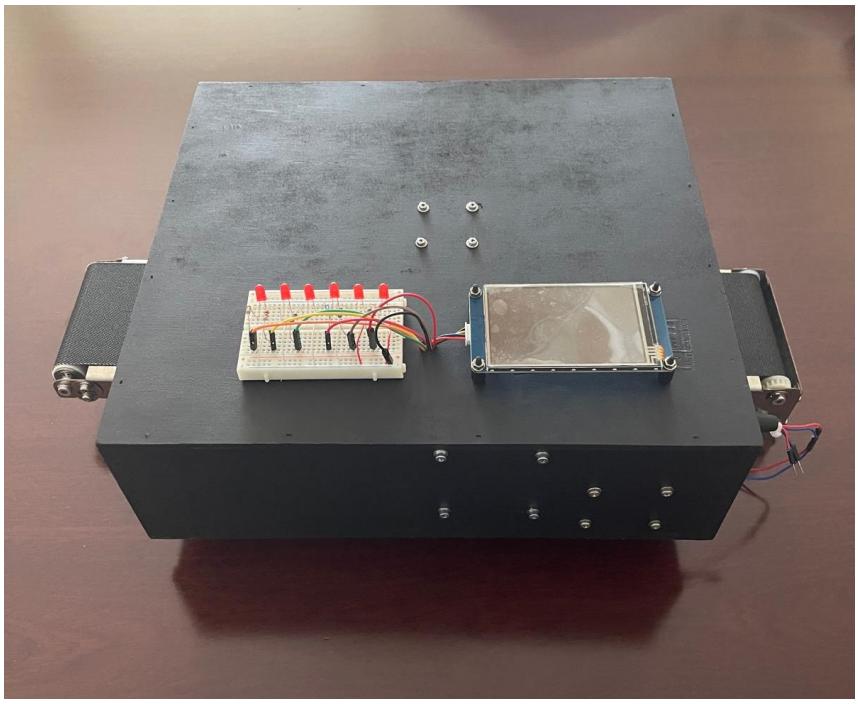
The conveyor belt's 24V DC motor requires a separate 24V power supply unit to satisfy its higher voltage needs. The power unit is capable of supplying enough current to meet the motor's operational requirements without causing voltage drop or power instability. This 24V supply is coupled to the L298N motor driver, which regulates motor operation. The L298N is critical for regulating the power sent to the motor, allowing for fine control of its speed and direction.

The dual power configuration not only assures operating efficiency and safety, but it also improves the system's overall performance and reliability. This structure enables for the system's easy integration into industrial contexts, achieving the project's goal of providing a small, efficient, and dependable automated sorting solution.

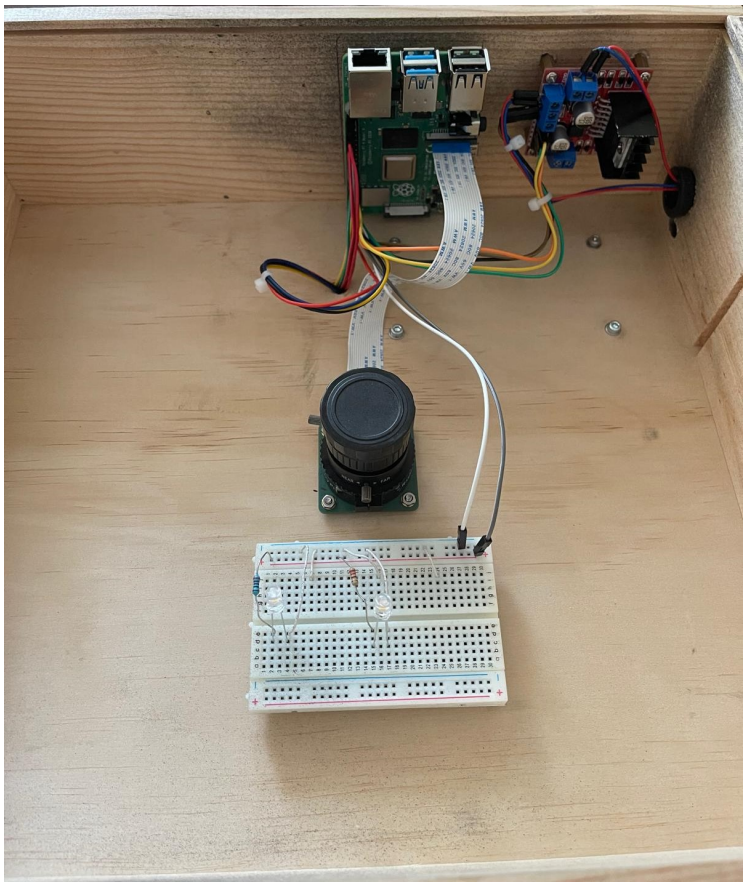
4.1.8 System Structure

A well-designed system structure is critical for ensuring the capsule sorting system's operational integrity and environmental management. The decision to place the complete assembly in a wooden box was deliberate, as it intended to provide a regulated lighting environment, which is crucial for effective picture capture and processing.

The container is designed to accommodate the conveyor belt, camera system, lighting arrangement, Raspberry Pi, display, and all related wiring. Every component was screwed into the wooden box, with the exception of the breadboard with white LEDs, which was glued to the box with velcro.



(a) System



(b) Component Container

Figure 4.6: System Structure

4.1.9 Circuit Schematic

To have a better understanding of how the hardware components are connected between each other the diagram of the circuit schematic presented in the picture 4.7 was made.

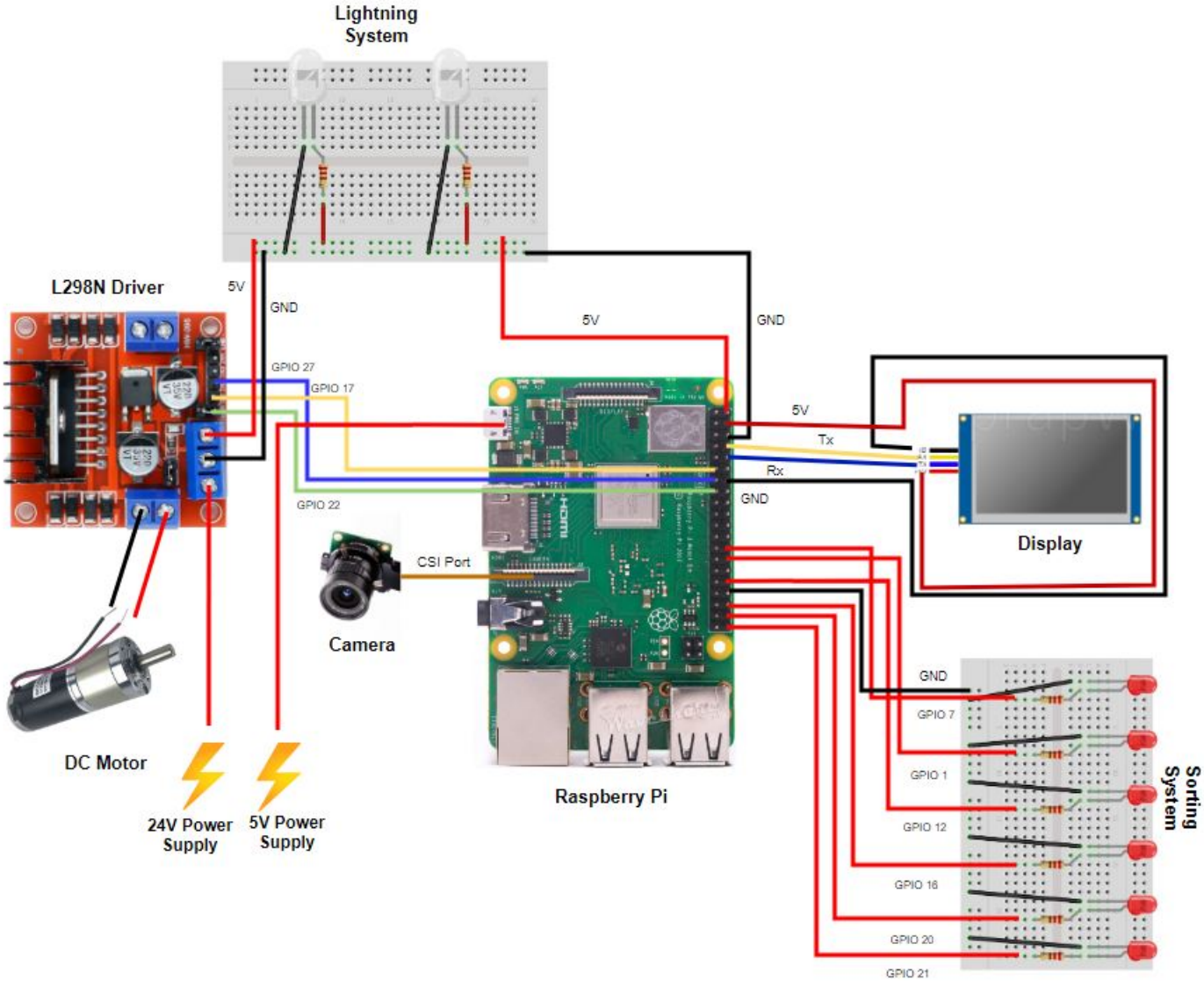


Figure 4.7: Circuit Schematic

4.2 Software Development

The software development of the capsule sorting system is critical to ensuring that the system runs efficiently and successfully. The programming language chosen was Python for its widespread use and compatibility with the Raspberry Pi and for offering a vast set of libraries that facilitate image processing, hardware control, and user interface creation. Its straightforward syntax, ability to handle high-level operations and simplicity of interface with numerous hardware devices make it ideal for this application.

Everything was developed with the help of Visual Studio Code, a powerful Integrated Development Environment (IDE) very commonly used for software development because of its extensive plugin ecosystem and strong support for python development, connected via SSH to the Raspberry Pi allowing an easier and faster software deployment.

About the tools and libraries, the primary library used for image processing tasks was OpenCV (Open Source Computer Vision Library) which provides robust tools for real-time image analysis and is essential for accurately identifying and sorting capsules based on colour and shape. Also leveraged the existence of the Numpy library which is often used in conjunction with OpenCV, supporting large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. This is crucial for handling the data involved in image processing tasks. Last but not the least, the GPIO library was utilized for interfacing with the Raspberry Pi's hardware pins, allowing control over components like the conveyor belt's motor and the LED indicators. This library provides an essential interface for managing the hardware aspects of the system.

The software development for the capsule sorting system includes sophisticated image processing and decision-making algorithms designed to accurately identify and categorise capsules based on colour. The colour ranges for each of the colours presented in the capsules were very well defined, the Hue, Saturation and Value specifically (2.1.4). This way all the colours were able to be detected precisely. Below, in the figures 4.8 and 4.9, is presented a little explanation of the algorithm used, accompanied by flowcharts to illustrate the process flow. For the counting aspect of the system, the code developed is identical to the colour sorting one but with some changes to the image processing part where the functions Erosion and Dilation, previously mentioned in 2.2.2.3, are used to allow a greater detection of the capsules or pills that are closer to each other.

The image processing algorithm involves capturing images of the capsules and detecting their colours.

1. **Capture Image**: Use the HQ camera with a macro lens connected to a Raspberry Pi to capture images of the capsules on the conveyor belt.

2. **Region of Interest (ROI)**: Define a specific area in the captured image where the capsules are expected to be present. An example can be found in the figure B.2.

3. **Colour Conversion**: Convert the captured image from BGR to HSV colour space for better colour segmentation.

4. **Colour Detection**: For each defined colour range, create a mask to detect the presence of that colour in the ROI. The image B.3 is a representation of what is seen when colour is detected.

5. **Contour Detection**: Find contours in the mask to identify the presence and position of capsules. In the figure B.4 is how the camera detects and separates each colour.

6. **Filter Contours**: Filter contours based on area and bounding box size to eliminate noise and small objects.
7. **Store Results**: If valid contours are found, store the detected colours and contours for further processing.

The decision-making algorithm involves assigning detected colours to specific exits (LEDs) and controlling the conveyor belt based on the presence of capsules and user input from the HMI.

1. **Initialize HMI**: Set up the Human-Machine Interface for user interaction.
2. **Initialize LEDs/Exits**: Define the LEDs/exits.
3. **Initialize Conveyor Belt**: Set up the conveyor belt and motor control.
4. **User Input**: Wait for the user to start the conveyor belt through the HMI. The user can also choose to stop when intended.
5. **Detect Colours**: Use the image processing algorithm to detect colours in the ROI.
6. **Assign Exits**: Assign detected colours to the next available exit (LED) if they are not already assigned.
7. **Control Conveyor**: If no capsules are detected for a specified duration, stop the conveyor belt.

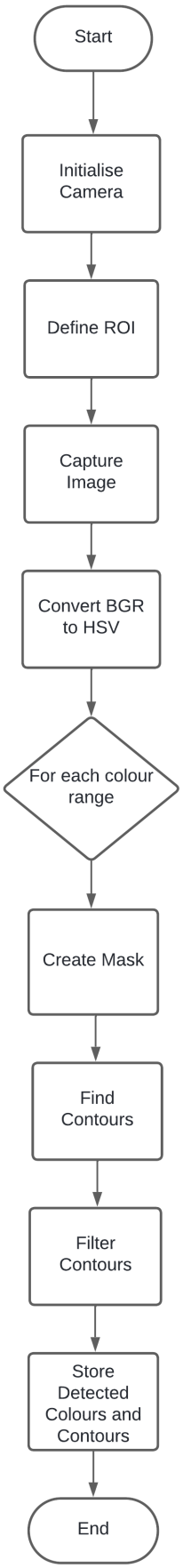


Figure 4.8: Image Processing Flowchart

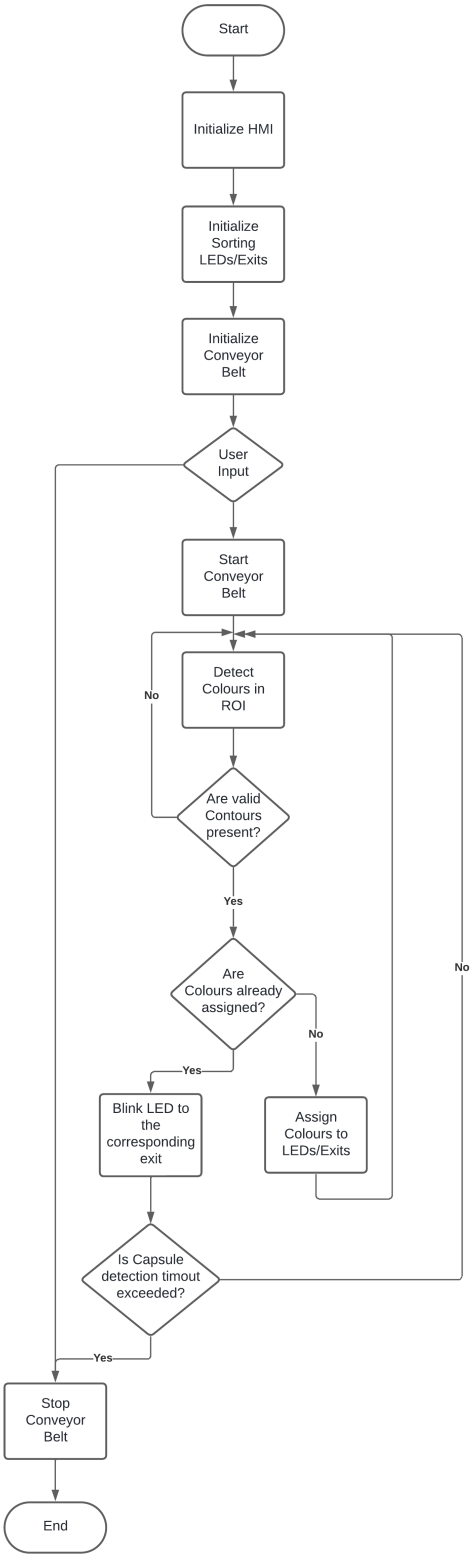


Figure 4.9: Decision-Making Flowchart

Chapter 5

System Testing and Validation

Testing and validation are critical aspects in the development of any engineering project, especially for systems such as the capsule sorting machine, which require precision, reliability, and efficiency. This section details the methodologies employed in testing and validating the system, including the setup of the testing environment and the specific criteria used for evaluation.

5.1 Testing Methods

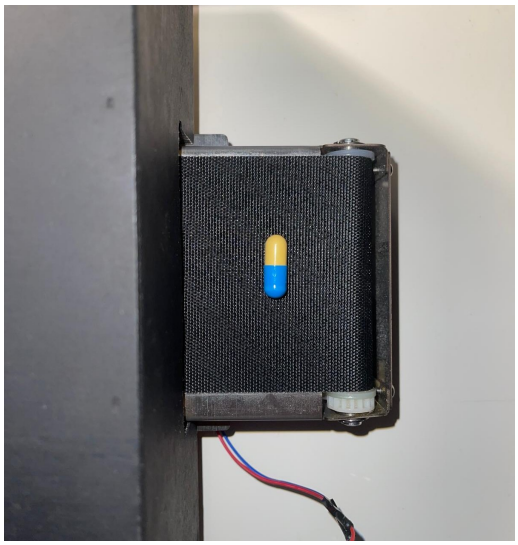
Component-Level Testing involves a thorough evaluation of each component. Every hardware component, including the camera, motor, and LEDs, is individually tested to ensure its proper functioning. On the software side, each module, particularly those responsible for image processing and GPIO interaction, is thoroughly tested to ensure that it executes its intended duties correctly. These tests are crucial for verifying that the system's core pieces are reliable before integrating them into the broader system.

System Integration Testing is performed when all components have demonstrated consistent performance on their own. During this phase, the system was constructed as it would be in an operational context, with all components designed to work together to verify the overall system design. This involves ensuring that the conveyor belt's movement is synchronized with the camera's image acquisition and processing, allowing the system to sort capsules precisely based on real-time data. Furthermore, the user interface on the display is tested under numerous circumstances to guarantee that it offers correct feedback and controls to the user, hence improving the system's usability and interactivity.

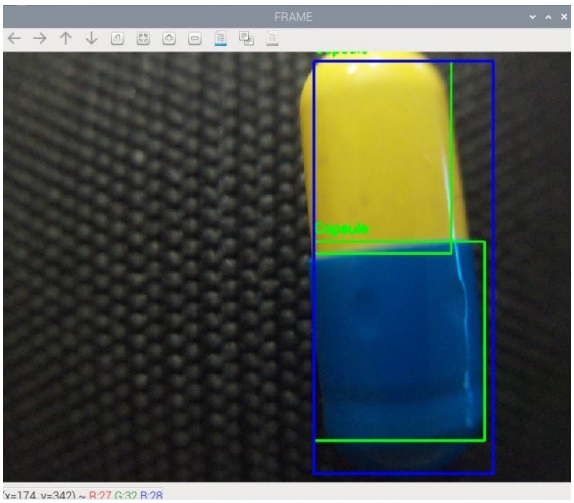
Performance testing evaluates a system's efficiency and efficacy. The system's processing and sorting speed is assessed against established benchmarks to verify if it satisfies the operational criteria for commercial deployment. Sorting accuracy, an important performance indicator, is assessed by measuring the system's ability to accurately detect and sort various coloured capsules.

During the testing phase some difficulties were found while trying to make the camera and the image capture work in an acceptable way so it could be further improved and enhanced to meet the necessary needs and requirements for the project's success. Unfortunately, the camera in question, which was a Raspberry Pi Camera Module V3, was not ideal for the purpose of the project since it didn't provide a clear and focused image when dealing with closed up scenarios and due to that fact most of the colours of the capsules would not be distinguishable. This was a major issue that had to be dealt with and therefore a new camera was acquired to fulfil the demands.

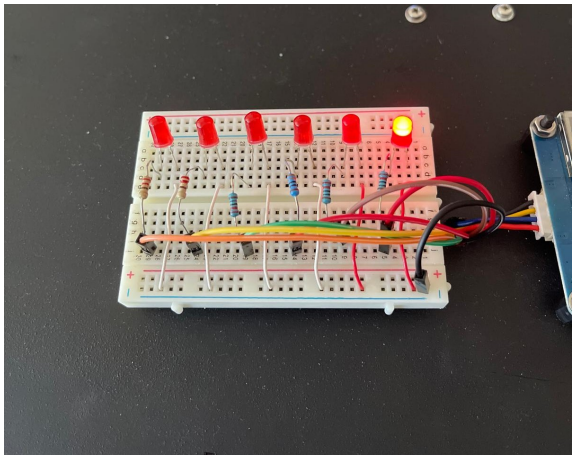
The testing process can be seen and understood through the figures 5.1 and 5.2 which represent the Colour Sorting process and the Counting process respectively. For the Colour Sorting process a large amount of capsules were sent through the conveyor belt to be detected and sorted by the system. In the images is represented the process flow where a capsule is deployed on the conveyor belt (a), then its colours are detected, registered and then assigned to the first exit or LED (b and c), and finally it comes out through the other end (d). If a second, third or fourth different capsule comes along, their colour are also registered and a LED/exit is assigned to that colour until the process is completed. The Counting process was a similar procedure where a sizeable amount of pills or small capsules set on the transporter (a), then before starting the user has the option to select the amount of capsules he wants to be specifically counted (b), and lastly the total count is presented at the end of the process (c).



(a) Capsule Input



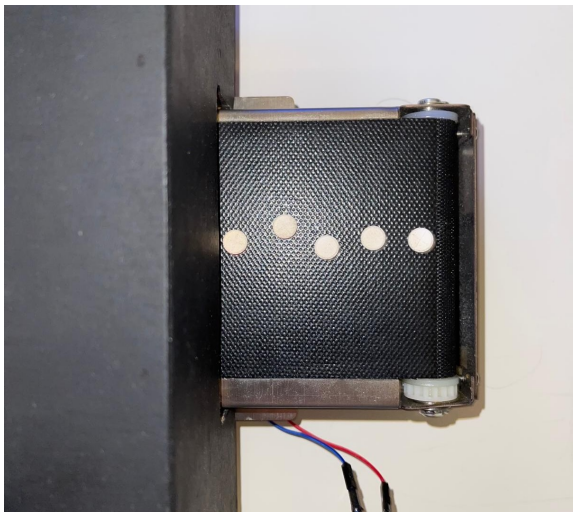
(b) Colour detected and registered



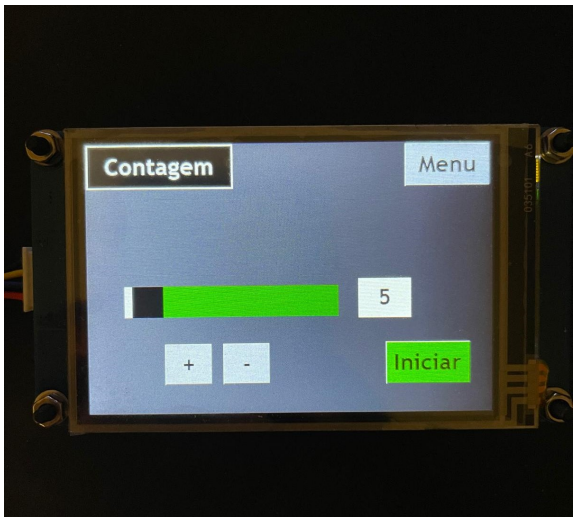
(c) LED Response

(d) Capsule Output

Figure 5.1: Colour Sorting Process



(a) Pills Input



(b) Counter Selection

(c) Total Count

Figure 5.2: Counting Process

5.2 Test Results

The capsule sorting system underwent extensive testing to evaluate its efficiency, accuracy, and reliability. The system's performance was assessed based on several metrics, including its ability to sort and count capsules. These metrics are then compared against predefined Key Performance Indicators (KPIs) to determine if the system meets the project objectives.

Table 5.1: Technical KPIs Analysis

KPI	**Target Value**	**Achieved Value**
Throughput Rate	2 capsules/sec	3 capsules/sec
Sorting Accuracy	98% success rate	99% success rate
Counting Accuracy	99% success rate	98% success rate
System Uptime and Error Rate	99% of uptime and error rate >1%	Almost no downtime and a 2% error rate
Power Consumption	15 W	15 W

5.2.1 Throughput Rate

The system successfully processes up to 3 capsules per second, aligning with the conveyor belt speed and necessary capsule spacing. The capsules also have to be predisposed in a vertical manner in order to avoid multiple colours detected.

The target KPI for throughput rate was at least 2 capsules per second to ensure operational efficiency. Achieving 3 capsules per second surpasses this KPI, highlighting the system's excellent efficiency.

5.2.2 Sorting Accuracy

The system achieved a 99% success rate in correctly sorting capsules by colour. The accuracy rate KPI was set to a 98%, anticipating potential challenges like variations in capsule colour shades or lightning conditions and the surpassing of this KPI demonstrates exceptional sorting capability.

5.2.3 Counting Accuracy

The system exhibits a counting accuracy of approximately 98%, occasionally counting one capsule or pill more or less than the actual number. The expected counting accuracy KPI was 99%. Although the system falls slightly short of this target, it remains within a generally acceptable range for many applications, though it may require adjustments for tasks requiring exact counts.

5.2.4 System Uptime and Error Rate

The system maintained a high uptime with negligible downtimes observed during testing, and an error rate of about 2% primarily in counting. The reliability KPI targeted a 99% uptime and an error rate of less than 1%. The system meets the uptime KPI but the error rate, although minimal, indicates room for improvement.

5.2.5 Power Consumption

When it comes to energy efficiency, Operating on a standard phone charger with an output of 5V/3A, the system's power consumption is measured to be consistently within the 15-watt limit. Given the compact and efficient design of the system, the power consumption was expected to remain under a modest 15 watts during normal operation. The system adheres to this KPI, emphasizing its energy efficiency and making it suitable for extended use without significant energy expenditure.

5.2.6 General Analysis

The testing phase demonstrated that the capsule sorting system excels in most areas, especially sorting efficiency and power consumption, where it meets or surpasses the established KPIs. However, the modest difference in counting accuracy and error rate identifies areas for potential improvement.

The system's near-perfect sorting accuracy proves its value in situations that need great precision, such as pharmaceutical production, where accuracy is important for compliance and safety. Minor counting flaws demand more refining, particularly for applications requiring accurate counts, such as packing or inventory management.

The counting algorithm must be optimised by being refined to increase capsule recognition and distinction, especially when capsules are close together, which might increase counting accuracy. Capsules positioned close one another or overlapping in the viewing area may be identified as a single item, resulting in undercounting. In contrast, quick motions or vibrations may cause the device to detect several counts for the same capsule. The camera may not have sufficient sensitivity or resolution to distinguish each capsule, especially when they pass fast or close together, and the image processing method may struggle to effectively separate closely spaced objects, or there may be faults in the logic that decides when to increment the count.

Overall, the capsule sorting system has tremendous promise and strong performance in the majority of the tested areas. With targeted improvements, notably in the counting process, the system is on track to completely fulfill all operational criteria and KPIs, assuring its effectiveness and reliability in real-world scenarios.

Chapter 6

Conclusion and Future Work

The automatic capsule sorting system described in this thesis marks a significant step forward in the integration of image processing and mechanical engineering. The system's design includes small, efficient components specifically designed for sorting pharmaceutical capsules. A high-resolution camera combined with complex software algorithms for colour identification exemplifies the system's technological sophistication. Mechanical components such as a precisely controlled conveyor belt system provide the smooth movement and handling of capsules, which is critical for sustaining high throughput rates.

6.1 Objective Satisfaction

Extensive testing and validation supplied reliable information on the system's performance. The findings demonstrated outstanding colour sorting accuracy, with a 99% success rate under controlled testing settings. The system's capacity to handle up to three capsules per second demonstrated its efficiency, greatly outperforming manual sorting techniques. In terms of reliability, the system performed nearly flawlessly, with little downtime occurring mainly during scheduled maintenance rather than unscheduled breakdowns. The power usage stayed within the intended parameters, proving the system's energy efficiency.

The automated capsule sorting system was designed with clear objectives aimed at addressing specific challenges within the pharmaceutical manufacturing sector. The technology outperformed manual sorting methods in terms of speed and accuracy. The system's throughput exceeded the objective, resulting in much higher production efficiency. This result not only achieves, but exceeds, the project's goal of assuring a highly dependable sorting process, which is critical for pharmaceutical goods' demanding quality criteria. The system was successfully built to be small, fitting into the restricted area available in industrial environments. This was accomplished without losing operating efficiency, which precisely aligns with the goal of creating a system that is both space-efficient and powerful. Its modular architecture also enables for simple maintenance and scalability, which is especially useful for industrial plants that want to adapt and enhance their capabilities without undergoing significant reconfiguration.

6.2 Future Work

While the current system has shown impressive results, there is always room for improvement and expansion to enhance its functionality and adaptability. One of the most important areas for future improvement is the shift from employing LED indications for sorting confirmation to constructing a more robust mechanical sorting process. The use of automated doors or gates that automatically send capsules into specified bins might significantly enhance the physical sorting process. This device would employ actuators activated by software choices to physically sort capsules, decreasing reliance on visual confirmation and expediting the entire process.

Additional improvements to the image processing algorithms are required, notably to improve capsule counting and colour identification accuracy in varying lighting circumstances and with capsules of diverse hues. Advanced machine learning models might be trained on bigger datasets to better manage these changes, boosting the system's resilience. Upgrading to higher-resolution cameras and adding quicker, more accurate motors might aid in meeting high-throughput demands while preserving precision. These hardware improvements would help the new mechanical sorting mechanisms by providing sharper pictures for processing and more precise motion control for the mechanical components.

Scalability testing is critical for determining the system's efficacy in bigger production environments or with different types of capsules. This would aid in understanding the constraints of the existing design and what changes may be required to accommodate different production sizes. In the future, connecting the capsule sorting system with other production processes, such as packaging and quality inspection, may result in even greater efficiency. This comprehensive strategy would improve the whole production process, minimizing bottlenecks and increasing throughput. Incorporating more advanced AI and machine learning approaches to dynamically change sorting algorithms based on real-time data is another intriguing area of future research. This might include utilizing real-time feedback to automatically update sorting settings, making the system more adaptive to changes in capsule features or operational conditions.

In conclusion, the automated capsule sorting system fulfilled and exceeded its initial objectives, displaying considerable improvements in efficiency, accuracy, and compact design. The future additions indicated above not only aim to expand on the system's present strengths, but also to solve its few weaknesses, opening the way for a next-generation sorting system that has the potential to redefine pharmaceutical production automation norms.

Bibliography

[1] Aisha Ajmal et al. “A Comparison of RGB and HSV Colour Spaces for Visual Attention Models”. In: *2018 International Conference on Image and Vision Computing New Zealand (IVCNZ)*. 2018, pp. 1–6. DOI: 10.1109/IVCNZ.2018.8634752.

[2] Charalampos Doukas and Ilias Maglogiannis. “Region of Interest Coding Techniques for Medical Image Compression”. In: *IEEE Engineering in Medicine and Biology Magazine* 26.5 (2007), pp. 29–35. DOI: 10.1109/EMB.2007.901793.

[3] Gareth Halfacree and Eben Upton. *Raspberry Pi user guide*. John Wiley & Sons, 2012.

[4] Fatuma Saeid Hassan and Adnan Gutub. “Improving data hiding within colour images using hue component of HSV colour space”. In: *CAAI Transactions on Intelligence Technology* 7.1 (2022), pp. 56–68.

[5] Ziyue Li et al. “An ROI Optimization Method Based on Dynamic Estimation Adjustment Model”. In: *Remote Sensing* 15.9 (2023). ISSN: 2072-4292. DOI: 10.3390/rs15092434. URL: https://www.mdpi.com/2072-4292/15/9/2434.

[6] Youcun Lu et al. “Application and improvement of Canny edge-detection algorithm for exterior wall hollowing detection using infrared thermal images”. In: *Energy and Buildings* 274 (2022), p. 112421. ISSN: 0378-7788. DOI: https://doi.org/10.1016/j.enbuild.2022.112421. URL: https://www.sciencedirect.com/science/article/pii/S0378778822005928.

[7] Matt Richardson and Shawn Wallace. *Getting Started with Raspberry Pi: Electronic Projects with Python, Scratch, and Linux*. Maker Media, Inc., 2012.

[8] Weibin Rong et al. “An improved CANNY edge detection algorithm”. In: *2014 IEEE international conference on mechatronics and automation*. IEEE. 2014, pp. 577–582.

[9] K A M Said and A B Jambek. “Analysis of Image Processing Using Morphological Erosion and Dilation”. In: *Journal of Physics: Conference Series* 2071.1 (Oct. 2021), p. 012033. DOI: 10.1088/1742-6596/2071/1/012033. URL: https://dx.doi.org/10.1088/1742-6596/2071/1/012033.

[10] G. Saravanan, G. Yamuna, and S. Nandhini. "Real time implementation of RGB to HSV/HSI/HSL and its reverse color space models". In: *2016 International Conference on Communication and Signal Processing (ICCSP)*. 2016, pp. 0462–0466. DOI: 10.1109/ICCSP.2016.7754179.

[11] Pierre Soille. "Erosion and Dilation". In: *Morphological Image Analysis: Principles and Applications*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 63–103. ISBN: 978-3-662-05088-0. DOI: 10.1007/978-3-662-05088-0_3. URL: https://doi.org/10.1007/978-3-662-05088-0_3.

[12] Cheickna Sylla. "Experimental investigation of human and machine-vision arrangements in inspection tasks". In: *Control Engineering Practice* 10.3 (2002), pp. 347–361. ISSN: 0967-0661. DOI: https://doi.org/10.1016/S0967-0661(01)00151-4. URL: https://www.sciencedirect.com/science/article/pii/S0967066101001514.

[13] O Rebecca Vincent, Olusegun Folorunso, et al. "A descriptive algorithm for sobel image edge detection". In: *Proceedings of informing science & IT education conference (InSITE)*. Vol. 40. 2009, pp. 97–107.

[14] Zhao Xu, Xu Baojie, and Wu Guoxin. "Canny edge detection based on Open CV". In: *2017 13th IEEE international conference on electronic measurement & instruments (ICEMI)*. IEEE. 2017, pp. 53–56.

Appendix A

Overall Project Cost

Component	**Quantity**	**Price (€)**
Raspberry Pi 3	1	39.87
Memory Card 32GB	1	7.65
Raspberry Pi HQ Camera	1	59.9
6mm Macro Lens for Raspberry Pi Camera	1	27.99
Touch HMI Display: Nextion NX4832T035	1	64.5
L298N Motor Driver	1	2.25
Conveyor Belt	1	64
LEDs	6	1.26
Total	13	267.42

Table A.1: List of Components

Appendix B

Images

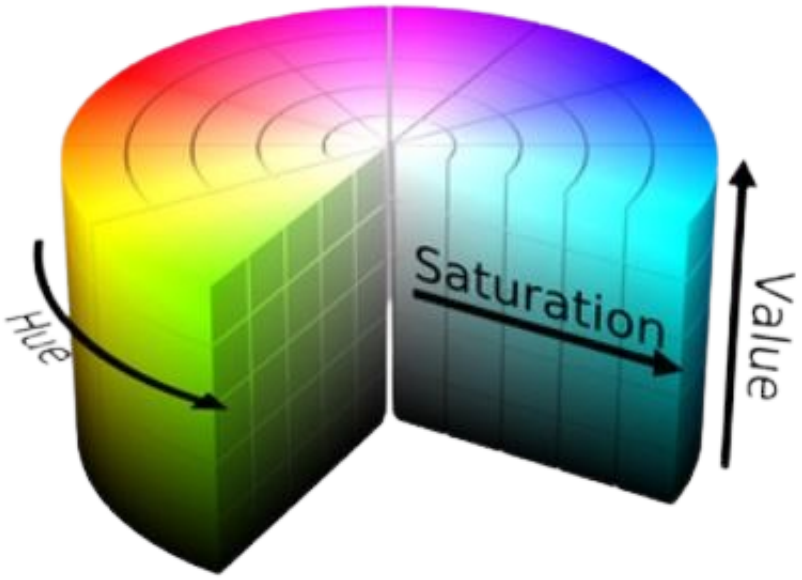


Figure B.1: HSV Colour Space

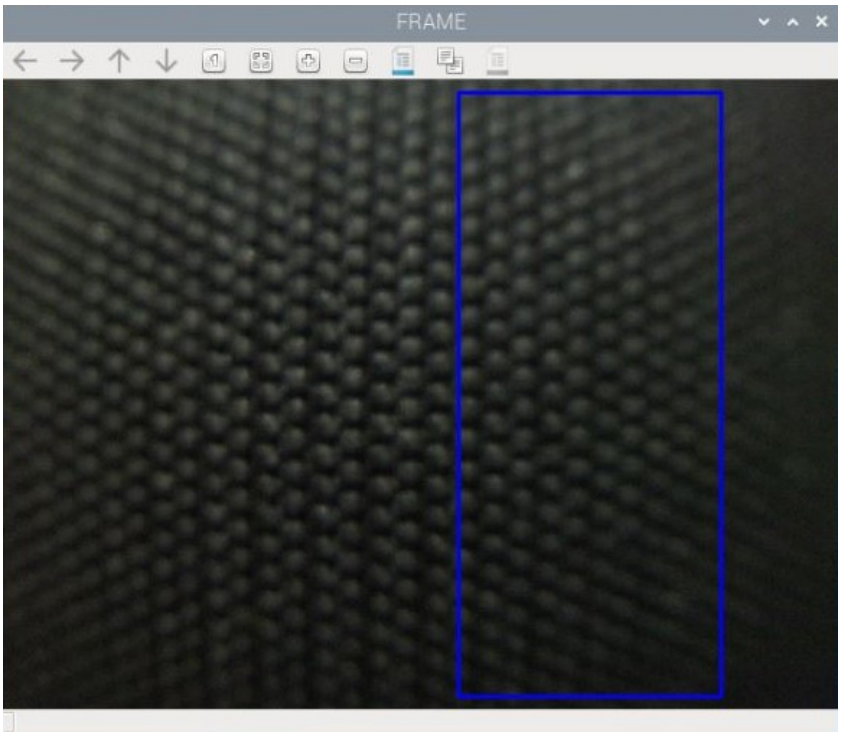


Figure B.2: Region of Interest (ROI)

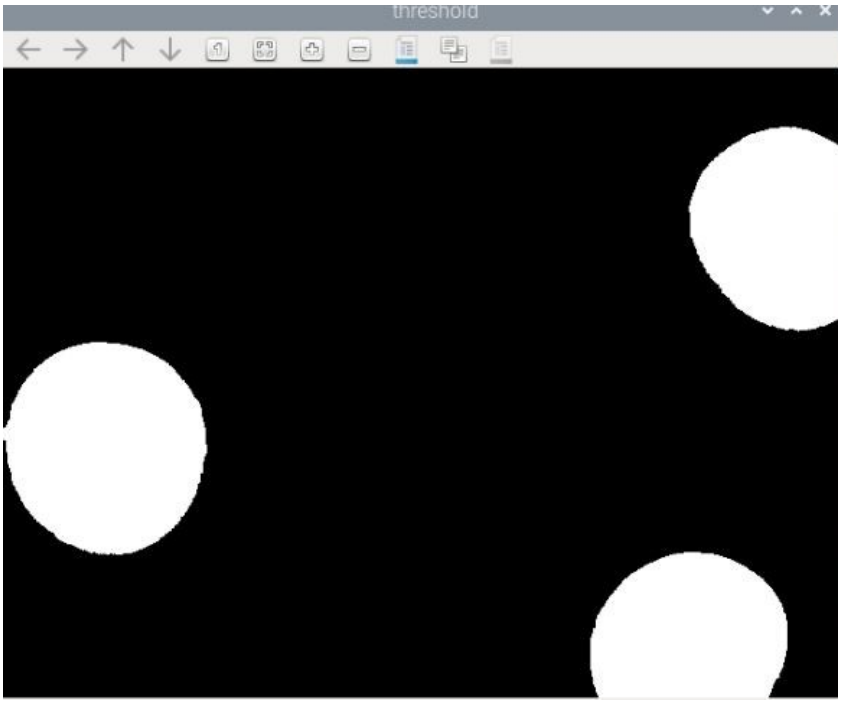


Figure B.3: Mask

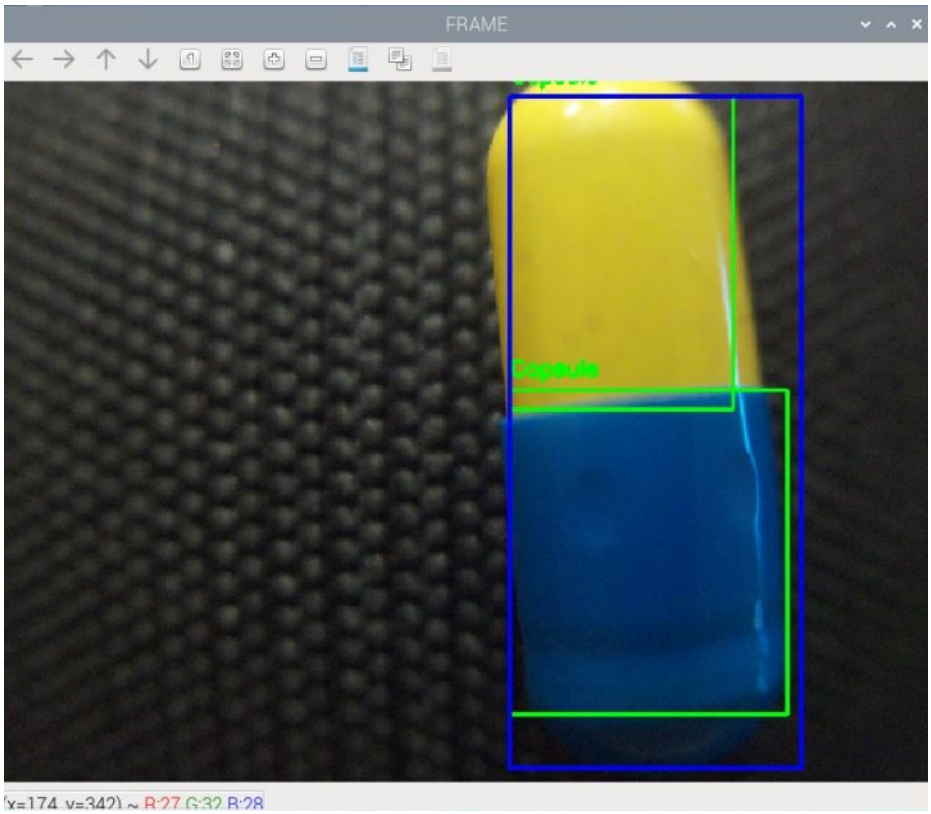


Figure B.4: Contour

Appendix C

Code

```
from picamera2 import Picamera2, Preview
# import cv2
# import numpy as np
# import time
# import RPi.GPIO as GPIO
# import serial
# import os

# GPIO.setwarnings(False)

# # Set the GPIO mode
# GPIO.setmode(GPIO.BCM)

# # Define GPIO pins for LEDs (you can change the pin numbers as
 ↪  needed)
# LED_PINS = [21, 20, 16, 12, 25, 24]

# # Set up GPIO pins for LEDs as outputs
# for pin in LED_PINS:
#     GPIO.setup(pin, GPIO.OUT)
#     GPIO.output(pin, GPIO.LOW)  # Initialize with LEDs off

# # Define GPIO pins
# IN1 = 17
# IN2 = 27
# ENA = 22

# # Set up the GPIO pins
# GPIO.setup(IN1, GPIO.OUT)
# GPIO.setup(IN2, GPIO.OUT)
# GPIO.setup(ENA, GPIO.OUT)
```

```
31
32 # # Set up PWM on the ENA pin
33 # pwm = GPIO.PWM(ENA, 1000)  # PWM at 1kHz frequency
34 # pwm.start(0)  # Start PWM with 0% duty cycle
35
36 # def set_motor_speed(speed):
37 #     """Set motor speed (0 to 100)"""
38 #     if speed > 100:
39 #         speed = 100
40 #     elif speed < 0:
41 #         speed = 0
42 #     pwm.ChangeDutyCycle(speed)
43
44 # def forward():
45 #     """Move motor forward"""
46 #     GPIO.output(IN1, GPIO.HIGH)
47 #     GPIO.output(IN2, GPIO.LOW)
48
49 # def backward():
50 #     """Move motor backward"""
51 #     GPIO.output(IN1, GPIO.LOW)
52 #     GPIO.output(IN2, GPIO.HIGH)
53
54 # def stop():
55 #     """Stop motor"""
56 #     GPIO.output(IN1, GPIO.LOW)
57 #     GPIO.output(IN2, GPIO.LOW)
58
59 # ################################################ DISPLAY
   ↪  ##################################################
60
61 # # Set up the serial connection (adjust port if necessary)
62 # ser = serial.Serial('/dev/ttyS0', baudrate=9600, timeout=1)
63
64 # # Variable to track the current mode (1 = Separação, 2 = Contagem)
65 # current_mode = 1
66
67 # # Variable to track whether sorting should stop
68 # stop_sorting = False
69 # stop_counting = False
70
71 # # Global variables for capsule counting
72 # target_count = 50  # Target number of capsules to count, set by
   ↪  slider/buttons
```

```
# current_count = 0  # Current number of capsules counted

# # Function to send commands to the Nextion display
# def write_to_display(command):
#     ser.write(command.encode('utf-8'))  # Send the command as a string
#     ser.write(b'\xff\xff\xff')  # Send termination bytes (0xFF 0xFF 0xFF)
#     print(f"Sent: {command}")  # For debugging purposes

# # Function to toggle the mode between "Separação" and "Contagem"
# def toggle_mode():
#     global current_mode
#     if current_mode == 1:
#         current_mode = 2  # Switch to "Contagem"
#         write_to_display('b1.txt="Contagem"')  # Update the Mode button text
#         print("Mode switched to Contagem")
#     else:
#         current_mode = 1  # Switch to "Separação"
#         write_to_display('b1.txt="Separacao"')  # Update the Mode button text
#         print("Mode switched to Separacao")

# def init_display():
#     """Initialize the display and set it to page 0"""
#     write_to_display('page 0')  # Send command to go to page 0 on startup

# # Function to handle button press events
# def handle_button_event(event_data):
#     global target_count, stop_sorting, current_count, stop_counting
#     page_id = event_data[1]  # Extract the page ID
#     component_id = event_data[2]  # Extract the component ID (e.g., b0 = 0x00, b1 = 0x01)

#     # Handle the "Mode" button (b1)
#     if page_id == 0x00 and component_id == 0x01:  # Button b1 on Page 0 (Mode button)
#         print("Mode button pressed, toggling mode")
#         toggle_mode()  # Toggle between "Separação" and "Contagem"

#     # Handle the "Initiate" button (b0)
```

```
#     elif page_id == 0x00 and component_id == 0x02:  # Button b0 on Page 0 (Initiate button)
#         print("Initiate button pressed, starting selected process")
#         if current_mode == 1:  # If "Separação" is selected
#             write_to_display('page 1')  # Go to Separation page
#             print("Starting Colour Sorting")
#             stop_sorting = False # Reset stop flag
#             run_colour_sorting()

#         elif current_mode == 2:  # If "Contagem" is selected
#             write_to_display('page 2')  # Go to Counting page
#             print("Choose amount")

#     # Handle the Shutdown button (ID 4 on Page 0)
#     elif page_id == 0x00 and component_id == 0x04:  # Button 4 on Page 0 (Shutdown button)
#         print("Shutdown button pressed, shutting down the Raspberry Pi")
#         write_to_display('page 4')
#         write_to_display('t0.txt="Shutting down..."')  # Show shutdown message on the display
#         time.sleep(1)
#         os.system("sudo shutdown now")  # Shutdown the Raspberry Pi

#     # Handle the Reboot button (ID 3 on Page 0)
#     elif page_id == 0x00 and component_id == 0x03:  # Button 3 on Page 0 (Reboot button)
#         print("Reboot button pressed, rebooting the Raspberry Pi")
#         write_to_display('page 4')
#         write_to_display('t0.txt="Rebooting..."')  # Show reboot message on the display
#         time.sleep(1)
#         os.system("sudo reboot")  # Reboot the Raspberry Pi

#     # Handle the "Stop" button on the Separation and Counting page
#     elif page_id == 0x01 and component_id == 0x03:  # Button 3 on Separation page (Stop button)
#         print("Stop button pressed, stopping colour sorting")
#         stop_sorting = True  # Set stop flag to break out of sorting loop

#     elif page_id == 0x03 and component_id == 0x05:  # Button 5 on Counting page (Stop button)
#         print("Stop button pressed, stopping counting")
```

```
144 #         stop_counting = True  # Set stop flag to break out of
    ↪ counting loop
145
146 #     # Handle the Plus button on the Counting page (button ID 5)
147 #     elif page_id == 0x02 and component_id == 0x05:  # Button 5 on
    ↪ Counting page (Plus button)
148 #         print("Plus button pressed, increasing target count")
149 #         target_count += 1
150 #         update_number(target_count)  # Update the displayed number
    ↪ on the screen
151
152 #     # Handle the Minus button on the Counting page (button ID 6)
153 #     elif page_id == 0x02 and component_id == 0x06:  # Button 6 on
    ↪ Counting page (Minus button)
154 #         print("Minus button pressed, decreasing target count")
155 #         if target_count > 0:
156 #             target_count -= 1
157 #         update_number(target_count)  # Update the displayed number
    ↪ on the screen
158
159 #     # Handle the "Initiate" button on the Counting page (button ID
    ↪ 3)
160 #     elif page_id == 0x02 and component_id == 0x07:  # Button 3 on
    ↪ Counting page (Initiate button)
161 #         print("Initiate button pressed, starting capsule counting")
162 #         current_count = 0  # Reset the current count
163 #         write_to_display('page 3')  # Go to Counting Result page
164 #         stop_counting = False # Reset stop flag
165 #         run_counter()  # Start the counting process
166
167 #     # Handle the "Back to Home" buttons on different pages
168 #     elif (page_id == 0x01 and component_id == 0x02) or \
169 #          (page_id == 0x02 and component_id == 0x04) or \
170 #          (page_id == 0x03 and component_id == 0x02):
171 #         print(f"Back to Home button pressed, returning to Page 0
    ↪ from Page {page_id}")
172
173 #         write_to_display('page 0')  # Go back to Page 0
174 #         if current_mode == 1:
175 #             stop_sorting = True # Also stop the sorting process
176 #             write_to_display('b1.txt="Separacao"')
177 #         elif current_mode == 2:
178 #             stop_counting = True
179 #             write_to_display('b1.txt="Contagem"')
```

```
180
181 # # Function to handle slider value event and update the number field
    ↪ based on slider value
182 # def handle_slider_event(event_data):
183 #     global target_count
184 #     # Slider h0 on Page 2
185 #     slider_value = event_data[1]  # Extract the slider value
    ↪ (assuming it's in byte 3)
186 #     print(f"Slider value: {slider_value}")
187 #     target_count = slider_value  # Update target count based on
    ↪ slider
188 #     update_number(target_count)
189
190 # # Function to update the number field based on the slider value
191 # def update_number(value):
192 #     # Update the number field n0 on Page 2 based on the slider value
193 #     write_to_display(f'n0.val={value}')  # Send command to update
    ↪ number component n0
194
195 # # Function to read and process multiple events from the Nextion
    ↪ display
196 # def read_from_display():
197 #     if ser.in_waiting > 0:
198 #         data = ser.read(ser.in_waiting)  # Read raw bytes
199 #         print(f"Raw Data Received: {data.hex()}")  # Print raw data
    ↪ in hex format for debugging
200
201 #         # Process each event individually
202 #         process_events(data)
203
204 # # Function to process each event from the received data
205 # def process_events(data):
206 #     while data:
207 #         if data.startswith(b'\x65') and len(data) >= 7:  # Handle
    ↪ button press event
208 #             handle_button_event(data[:7])
209 #             data = data[7:]  # Remove the processed event from the
    ↪ data
210
211 #         elif data.startswith(b'\x71') and len(data) >= 7:  # Handle
    ↪ slider value event
212 #             handle_slider_event(data[:7])
213 #             data = data[7:]  # Remove the processed event from the
    ↪ data
```

```
214
215 #         else:
216 #             break  # If data doesn't match known patterns, stop
    ↪ processing
217
218 # ######################################### SEPARATION
    ↪ ########################################################
219
220 # def run_colour_sorting():
221
222 #     picam2 = Picamera2()
223 #     camera_config =
    ↪ picam2.create_preview_configuration(main={"size": (3280, 2464),
    ↪ "format": "RGB888"})
224 #     picam2.preview_configuration.align()
225 #     picam2.configure(camera_config)
226 #     picam2.start()
227 #     time.sleep(1)
228
229 #     global stop_sorting # Access the global stop_sorting flag
230
231 #     # Define the coordinates of the ROI (x, y, width, height)
232 #     roi_x, roi_y, roi_w, roi_h = 350, 10, 200, 460  # Coordinates
233
234 #     # Lower and Upper ranges for different colors
235 #     color_ranges = {
236 #         "red": [(np.array([0, 178, 70]), np.array([6, 255, 255])),
237 #                 (np.array([160, 178, 70]), np.array([179, 255,
    ↪ 255]))],
238 #         "green": [(np.array([77, 110, 30]), np.array([88, 255,
    ↪ 250])),
239 #                 (np.array([89, 110, 30]), np.array([94, 255,
    ↪ 250]))],
240 #         "lightgreen": [(np.array([58, 110, 110]), np.array([67, 255,
    ↪ 250])),
241 #                     (np.array([68, 110, 110]), np.array([70, 255,
    ↪ 250]))],
242 #         "blue": [(np.array([103, 190, 75]), np.array([107, 255,
    ↪ 250])),
243 #                 (np.array([108, 190, 75]), np.array([112, 255,
    ↪ 250]))],
244 #         "yellow": [(np.array([25, 120, 115]), np.array([29, 255,
    ↪ 250])),
```

```
#                 (np.array([30, 120, 115]), np.array([39, 255,
↪ 250]))],
#         "orange": [(np.array([15, 154, 125]), np.array([17, 255,
↪ 250])),
#                 (np.array([18, 154, 125]), np.array([22, 255,
↪ 250]))],
#         "white": [(np.array([25, 0, 102]), np.array([50, 53, 227])),
#                 (np.array([51, 0, 102]), np.array([58, 53, 227]))],
#     }

#     # Database to store unique photos

#     photo_database = []

#     # Keep track of color-to-LED assignments
#     color_to_led = {}  # Dictionary to map colors to specific LED
↪ pins

#     # Function to detect the colors in the frame
#     def detect_colors(frame, color_ranges):
#         hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

#         # Apply histogram equalization on the V channel
#         hsv[:,:,2] = cv2.equalizeHist(hsv[:,:,2])

#         detected_colors = set()
#         valid_contours = []
#         for color_name, ranges in color_ranges.items():
#             masks = [cv2.inRange(hsv, lower, upper) for lower, upper
↪ in ranges]
#             mask = cv2.add(*masks)
#             _, mask = cv2.threshold(mask, 254, 255,
↪ cv2.THRESH_BINARY)

#             cnts, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL,
↪ cv2.CHAIN_APPROX_NONE)
#             for c in cnts:
#                 if cv2.contourArea(c) > 2500:  # Filter based on
↪ contour area
#                     x, y, w, h = cv2.boundingRect(c)
#                     if w > 35 and h > 35:  # Further filter based on
↪ bounding box size
#                         detected_colors.add(color_name)
#                         valid_contours.append(c)
```

```
#         return list(detected_colors), valid_contours

#     # Function to assign colors to LEDs and make the corresponding LED blink
#     def handle_led_blink(detected_colors, color_to_led, available_leds):
#         # Sort the detected colors to ensure consistency in the tuple key
#         color_combination = tuple(sorted(detected_colors))

#         # If the color combination is not yet assigned and there are available LEDs
#         if color_combination not in color_to_led and available_leds:
#                 led_pin = available_leds.pop(0)  # Assign the next available LED
#                 color_to_led[color_combination] = led_pin  # Map the color to the LED pin
#                 print(f"Assigned {color_combination} to LED {led_pin}")

#         # Blink the corresponding LED if the color is assigned
#         if color_combination in color_to_led:
#             led_pin = color_to_led[color_combination]
#             print(f"Blinking LED {led_pin} for color {color_combination}")
#             GPIO.output(led_pin, GPIO.HIGH)
#             time.sleep(0.1)
#             GPIO.output(led_pin, GPIO.LOW)

#     # Function to handle the database separately from LED blinking
#     def handle_photo_database(detected_colors, photo_database):
#         # Check if the detected color combination is already in the database
#         if not is_color_in_database(detected_colors, photo_database):
#             # Add the new photo with unique colors to the database
#             photo_database.append({"colors": detected_colors})
#             print(f"New colour added: {detected_colors}")

#     # Function to check if the detected colors are already in the database
#     def is_color_in_database(detected_colors, database):
#         for entry in database:
#             if set(detected_colors) == set(entry["colors"]):
```

```
313 #                 return True
314 #         return False
315 
316 #     # MOTOR
317 #     forward()
318 #     set_motor_speed(100)
319 #     print("Motor running at 100% speed")
320 
321 #     # Initialize the time of last detection
322 #     last_detection_time = time.time()
323 
324 #     # Initialize available LED pins list (copy of LED_PINS)
325 #     available_leds = LED_PINS[:]
326 
327 #     while True:
328 #         frame = picam2.capture_array()
329 #         frame = cv2.resize(frame, (640, 480))
330 
331 #         read_from_display()
332 
333 #         if stop_sorting:  # Check if the stop button has been
    ↪ pressed
334 #             stop()
335 #             print("Colour sorting stopped by user")
336 #             break
337 
338 #         # Crop the frame to the ROI
339 #         roi_frame = frame[roi_y:roi_y + roi_h, roi_x:roi_x + roi_w]
340 #         detected_colors, valid_contours = detect_colors(roi_frame,
    ↪ color_ranges)
341 
342 #         # Ensure that valid contours are present before adding the
    ↪ image to the database
343 #         if detected_colors and valid_contours:
344 #             last_detection_time = time.time()
345 
346 #             # Handle LED blinking for detected colors
347 #             handle_led_blink(detected_colors, color_to_led,
    ↪ available_leds)
348 
349 #         # Handle the photo database (only add unique color
    ↪ combinations)
350 #         handle_photo_database(detected_colors, photo_database)
351 
```

```
#         # Draw bounding boxes for valid contours for visualization
#         for c in valid_contours:
#             x, y, w, h = cv2.boundingRect(c)
#             #cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
#             cv2.rectangle(frame, (roi_x + x, roi_y + y), (roi_x + x + w, roi_y + y + h), (0, 255, 0), 2)
#             cv2.putText(frame, "Capsule", (roi_x + x, roi_y + y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

#         # Draw the ROI rectangle for visualization
#         cv2.rectangle(frame, (roi_x, roi_y), (roi_x + roi_w, roi_y + roi_h), (255, 0, 0), 2)

#         # Stop the motor if no capsules are detected for 10 seconds
#         if time.time() - last_detection_time > 15:
#             stop()
#             print("Motor stopped due to no detection for 10 seconds")
#             break

#         time.sleep(0.3)

#     picam2.stop()
#     picam2.close()

# ######################################## COUNTING ##################################################

# contours_count = 0

# def nothing(x):
#     pass

# def findContoursInImage(frameImg, showImage, frame):
#     contours, hierarchy = cv2.findContours(frameImg, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
#     try:
#         hierarchy = hierarchy[0]
#     except:
#         hierarchy = []

#     global contours_count

```

```
#     print("Count of pills: ", contours_count)
#     bg = np.zeros((frameImg.shape[0], frameImg.shape[1], 3),
↪  np.uint8)

#     for contour, hier in zip(contours, hierarchy):
#         (x, y, w, h) = cv2.boundingRect(contour)
#         if x > 55 and x < 120:
#             contours_count = contours_count + 1

#     resultText(contours_count, frame)

#     for i in range(0, len(contours)):
#         contour = contours[i]
#         hull = cv2.convexHull(contour, False)
#         area = cv2.contourArea(contour)

#         area_min = 200  # Adjusted for larger pills
#         area_max = 20000  # Adjusted for larger pills

#         if area_min < area < area_max:
#             M = cv2.moments(contour)
#             if M["m00"] != 0:
#                 cX = int(M["m10"] / M["m00"])
#                 cY = int(M["m01"] / M["m00"])

#                 cv2.circle(showImage, (cX, cY), 5, (255, 0, 255),
↪  -1)

#                 epsilon = 0.02 * cv2.arcLength(contour, True)
#                 approx = cv2.approxPolyDP(contour, epsilon, True)
#                 cv2.drawContours(bg, [approx], -1, (0, 0, 255), 3)

#                 ellipse = cv2.fitEllipse(contour)
#                 cv2.ellipse(showImage, ellipse, (0, 255, 0), 3)

#     return contours_count

# def resultText(contours_count, frame):
#     cv2.putText(frame, f'Count of capsules: {contours_count}', (0,
↪  50), cv2.FONT_HERSHEY_SIMPLEX, .8, (0, 255, 0), 1, cv2.LINE_AA)

# def run_counter():

#     picam2 = Picamera2()
```

```
#     camera_config =
↪  picam2.create_preview_configuration(main={"size": (3280, 2464),
↪  "format": "RGB888"})
#     picam2.preview_configuration.align()
#     picam2.configure(camera_config)
#     picam2.start()
#     time.sleep(1)

#     global current_count, target_count, stop_counting,
↪  contours_count
#     contours_count = 0

#     cv2.namedWindow("Mask Settings")
#     cv2.resizeWindow("Mask Settings", 500, 250)

#     cv2.namedWindow("Threshold Settings")
#     cv2.resizeWindow("Threshold Settings", 500, 70)

#     cv2.createTrackbar("Thresh-Min", "Threshold Settings", 0, 255,
↪  nothing)
#     cv2.createTrackbar("Thresh-Max", "Threshold Settings", 0, 255,
↪  nothing)

#     cv2.createTrackbar("Lower-H", "Mask Settings", 0, 180, nothing)
#     cv2.createTrackbar("Lower-S", "Mask Settings", 0, 255, nothing)
#     cv2.createTrackbar("Lower-V", "Mask Settings", 0, 255, nothing)

#     cv2.createTrackbar("Upper-H", "Mask Settings", 0, 180, nothing)
#     cv2.createTrackbar("Upper-S", "Mask Settings", 0, 255, nothing)
#     cv2.createTrackbar("Upper-V", "Mask Settings", 0, 255, nothing)

#     cv2.setTrackbarPos("Thresh-Min", "Threshold Settings", 221)
#     cv2.setTrackbarPos("Thresh-Max", "Threshold Settings", 255)

#     cv2.setTrackbarPos("Lower-H", "Mask Settings", 0)
#     cv2.setTrackbarPos("Lower-S", "Mask Settings", 0)
#     cv2.setTrackbarPos("Lower-V", "Mask Settings", 149)

#     cv2.setTrackbarPos("Upper-H", "Mask Settings", 180)
#     cv2.setTrackbarPos("Upper-S", "Mask Settings", 255)
#     cv2.setTrackbarPos("Upper-V", "Mask Settings", 255)

#     # MOTOR
#     forward()
```

```
469 #     set_motor_speed(40)
470 #     print("Motor running")
471 
472 #     while contours_count < target_count:
473 
474 #         read_from_display()
475 
476 #         frame = picam2.capture_array()
477 #         frame = cv2.resize(frame, (640, 480))
478 #         roi = frame[0:, 0:125]
479 #         hsv = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
480 
481 #         lh = cv2.getTrackbarPos("Lower-H", "Mask Settings")
482 #         ls = cv2.getTrackbarPos("Lower-S", "Mask Settings")
483 #         lv = cv2.getTrackbarPos("Lower-V", "Mask Settings")
484 #         uh = cv2.getTrackbarPos("Upper-H", "Mask Settings")
485 #         us = cv2.getTrackbarPos("Upper-S", "Mask Settings")
486 #         uv = cv2.getTrackbarPos("Upper-V", "Mask Settings")
487 
488 #         lower_b = np.array([lh, ls, lv])
489 #         upper_b = np.array([uh, us, uv])
490 
491 #         if stop_counting:  # Check if the stop button has been
    ↪ pressed
492 #             stop()
493 #             print("Counting stopped by user")
494 #             break
495 
496 #         mask = cv2.inRange(hsv, lower_b, upper_b)
497 
498 #         # Apply morphological transformations with adjusted kernel
    ↪ size and sequence
499 #         kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5,
    ↪ 5))  # Smaller kernel size for finer separation
500 
501 #         # Apply a slight erosion followed by dilation (opening) to
    ↪ separate close objects
502 #         morphOpen = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel,
    ↪ iterations=1)
503 
504 #         # Now apply closing to fill in any small gaps
505 #         morphClose = cv2.morphologyEx(morphOpen, cv2.MORPH_CLOSE,
    ↪ kernel, iterations=1)
506 
```

```
#         # Threshold the image to create a binary image
#         _, thresholdImg = cv2.threshold(morphClose,
↪  cv2.getTrackbarPos("Thresh-Min", "Threshold Settings"),
↪  cv2.getTrackbarPos("Thresh-Max", "Threshold Settings"),
↪  cv2.THRESH_BINARY)

#         contours_count = findContoursInImage(thresholdImg, roi,
↪  frame)

#         # Update the number of counted capsules on the display
#         update_number(contours_count)

#         cv2.line(frame, (48, 0), (48, 500), (0, 255, 255), 2)
#         cv2.line(frame, (125, 0), (125, 500), (0, 255, 255), 2)

#         if contours_count >= target_count:
#             print(f"Target of {target_count} capsules reached.")
#             time.sleep(5)
#             stop()  # Stop the motor
#             break

#         key = cv2.waitKey(30)
#         if key == ord('q'):
#             stop()
#             break

#     picam2.stop()
#     picam2.close()
#     return

# ##################################### BACK TO DISPLAY MAIN LOOP
↪  ####################################

# # Main loop to continuously read from the display
# if __name__ == "__main__":
#     init_display()  # Set the display to page 0 at startup
#     while True:
#         read_from_display()
#         time.sleep(0.1)

# # Close the serial connection when done
# GPIO.cleanup()
# ser.close()
```