School of Electrical and Electronic Engineering,
Engineering Campus, University Science Malaysia

Mini Project Report

Group 23 (Thursday Morning)

TITLE:
Automated Colour Filter Production Line
(Track: Automated System)

NAME	*MATRIC NUMBER*
LIEW MING HENG	*161439*
LIM JIA XIANG	*161615*

LECTURER:
ASSOCIATE PROFESSOR DR. ZAINI ABDUL HALIM

SUBMISSION DATE:
11th July 2024

TABLE OF CONTENTS

Acknowledgement

We are glad to claim that all the objectives of the mini project have been met and thus the project is complete. The knowledge and skills that we got from EEE 226 Microprocessor I both from the physical class and lab sessions are well applied in this mini project. Also, the research and knowledge about different subjects have also helped in enhancing the practical knowledge and implementation.

Firstly, the authors would like to express their gratitude to all the lecturers, especially Dr. Zaini Abdul Halim, Encik Ahmad Nazri Bin Ali, and Dr. Syed Sahal Nazri Alhady Bin Syed Hassan for providing teaching materials and helpful guidance in the lab sessions and during the Tuesday lectures. A special thanks to Dr. Zaini Abdul Halim for her continuous advice and support throughout our mini project.

Finally, we would like to thank the staff and technicians of the PPKEE for their friendliness and cooperation. They have always been helpful to us whenever we needed them to help us, they have been supplying us with equipment for the lab and technical assistance. We are grateful that they agreed to look at the 8051 board, VisionFive 2 development board, and other parts whenever we had problems.

Abstract

During the age of industry 4. 0, industrial automation is used as an analytical tool to measure the level of technology used in production systems. Machine vision technology is commonly used in modern industries together with the PLCs as a means of automating some of the production lines. The goal of this mini project is to create a mimicked industrial automation filter circuit on the 8051 Microcontroller with the help of VisionFive 2 development board for the machine vision part. Arduino Nano, Keil uVision5, and Avrdudess applications help in compiling the assembly language code into the 8051 microcontroller for running the program codes.

The circuit prototype includes an LCD module for displaying counts, two servos for moving objects to designated boxes, four IR sensors for detecting object presence, one ADC0804, and a potentiometer for adjusting the conveyor belt speed. Additionally, a webcam, in conjunction with the VisionFive 2 development board, performs colour filtering using OpenCV. The system successfully mimics an industrial automation process, counting objects based on colour and displaying the total units on the LCD module. The two servos, equipped with 3D-printed fan blades, push the objects into specific storage boxes. Overall, this mini project prototype effectively demonstrates an industrial colour filter automation production line, similar to those used in industrial settings.

1.0 Introduction

In today's dynamic industrial landscape, the demand for efficient and precise manufacturing processes has spurred the development of automated systems across various sectors. One such innovative application is the Automated Colour Filter Production Line. This advanced system integrates cutting-edge technology, including machine vision and automated sorting mechanisms, to revolutionise the production of colour filters used in various industrial applications.

The primary objective of the Automated Colour Filter Production Line is to streamline the manufacturing process while ensuring consistency and quality in the output. By leveraging machine vision technology, the system can accurately detect and classify objects based on their colour properties. This capability eliminates the need for manual sorting, significantly reducing labour costs and minimising human error by replacing it with automation processes.

Furthermore, the automation of the production line enhances operational efficiency by optimising throughput and reducing production time. Tasks that traditionally required extensive manual intervention, such as colour inspection and sorting, are now performed swiftly and precisely by the integrated machine vision system. This not only accelerates the production cycle but also enhances the overall quality and uniformity of the colour filters manufactured.

In summary, the Automated Colour Filter Production Line represents a significant advancement in industrial automation technology. By combining advanced machine vision, and automated sorting, this system not only enhances manufacturing efficiency but also ensures consistent quality and reliability in colour filter production. As industries continue to embrace automation for improved productivity and competitiveness, systems like the Automated Colour Filter Production Line underscore the transformative impact of technology on modern manufacturing practices.

1.1 Aims

- To design an industrial colour filter production line using the 8051 microcontroller and VisionFive 2 development board.
- To integrate a webcam into the VisionFive 2 development board for machine vision applications.
- To use servos to control the movement of objects into specific storage boxes.
- To use an LCD module for real-time monitoring of the total units as well as the count of each different colour of units.

1.2 Objectives

- Design a real-time monitoring system for tracking the units of total and each different colour of objects
- Design a mechanism for pushing the object to specific storage boxes based on the colour specifications
- Develop a machine vision system for classifying the colour of the objects

1.3 Problem Statements

In the current era of Industry 4.0, industrial automation plays a crucial role in enhancing production efficiency and maintaining technological competitiveness. Traditional manual sorting methods are time-consuming, prone to human error, and inefficient for handling large volumes of products. The need for a more reliable and automated solution is paramount, especially in industries where precision and speed are critical. Additionally, the lack of real-time monitoring and accurate counting of sorted units further complicates the production process, leading to potential bottlenecks and increased operational costs. It is crucial to offset these difficulties with the help of integrated machine vision and accurate control systems to achieve high-quality, non-stop production.

1.4 Proposed Solutions

To address the challenges posed by traditional manual sorting methods, we propose the development of an automated colour filter production line utilising the 8051 microcontroller and the VisionFive 2 development board. This system aims to automate the sorting process, thereby increasing efficiency and accuracy.

The integration of machine vision is a key component of our solution. By incorporating a webcam into the VisionFive 2 development board, the system can utilise OpenCV for colour filtering, enabling accurate and efficient sorting of objects based on their colour. This technological advancement ensures that objects are correctly categorised without human intervention, significantly reducing errors and speeding up the sorting process.

A servo-controlled sorting mechanism further enhances the precision of the system. The use of servos to control the movement of objects into specific storage boxes ensures precise placement and reduces the likelihood of errors. The system employs 3D-printed fan blades attached to the servos to push objects into designated compartments. This method not only ensures accuracy but also increases the system's reliability and durability.

Real-time monitoring and counting are facilitated by an LCD module included in the design. This module provides real-time updates on the total number of units sorted and the count of each colour category. This feature enhances transparency and allows for immediate adjustments if necessary, ensuring that the production line operates smoothly and efficiently.

The system also features adjustable conveyor belt speed, achieved through the integration of an ADC0804 and a potentiometer. This allows for the adjustment of the conveyor belt speed, providing flexibility to adapt to varying production requirements and ensuring smooth operation. This adaptability is crucial for handling different types of products and varying production volumes.

Four IR sensors are strategically placed to detect the presence of objects, ensuring accurate sorting and minimising the risk of jams or misplacements. These sensors play a critical role in maintaining the efficiency and reliability of the system, as they provide real-time feedback on the position of objects, enabling the system to make necessary adjustments promptly.

Additionally, the system includes a power-saving mode activated by pressing a button. This mode allows the system to enter a low-power state, conserving energy when the production line is not in use. Importantly, the memory is preserved during this sleep mode, ensuring that no data is lost and the system can quickly resume full operation when needed. This feature enhances the overall efficiency and sustainability of the production line, reducing operational costs and extending the lifespan of the equipment.

By implementing these advanced technologies, the proposed automated colour filter production line will significantly enhance the efficiency, accuracy, and reliability of the sorting process. This system addresses the limitations of manual methods and meets the demands of modern industrial production, offering a robust solution to improve productivity and reduce operational costs.

2.0 Literature Review

2.1 Introduction and Evolution of Machine Vision

Machine vision technology has undergone significant evolution since its inception in the 1980s, as defined by the Automated Imaging Association (AIA). Initially emerging as a field focused on using hardware and software to provide operational guidance based on image capture and processing, machine vision has grown into a critical component across both industrial and non-industrial applications. This evolution has been driven by advancements in digital sensors housed within specialised industrial cameras, equipped with optics designed to acquire precise images. These images are subsequently processed, analysed, and measured by sophisticated computer hardware and software systems, facilitating real-time decision-making processes.

In the 1990s, the emergence of a number of sensors and interfaces may be regarded as the basic prerequisites for further advancements in the sphere of machine vision. This period was characterised by a slow evolution in the functions and uses of machine vision systems but integrating sensors especially the CCD sensors was still a hard task. Nevertheless, general purpose machine vision cameras, as represented by 1.3MP monochrome USB3 cameras, were widely used across various industries such as semiconductor manufacturing, metrology, food inspection, and life sciences.

The 2000s represented a pivotal era where the demand for automation and advanced computer vision solutions surged across industries. This surge was propelled by a growing need to enhance production capabilities and operational efficiencies. Despite the increasing availability of technology, industrial machine vision systems demanded greater robustness, reliability, and stability compared to their academic and governmental counterparts. This requirement underscored the necessity for low-cost solutions that maintained acceptable accuracy while withstanding challenging industrial environments, including mechanical stress and varying temperatures.

Machine vision systems today are distinguished by their ability to perform precise measurements, counting, decoding, and location functions which were stated in the table below. These capabilities are crucial for tasks such as quality control in manufacturing, where machine vision can detect defects or deviations from specifications with high accuracy. The integration of advanced optics, high-resolution sensors, and sophisticated vision processing algorithms has further refined the capabilities of these systems, enabling them to meet stringent industrial standards and operational requirements.

Function	***Description***
Precise Measurements	*Process required to take dimensions or features of an object and check if they are in conformity with the set standards for*

	tolerances. Some of the applications include use in manufacturing procedures such as the measurement of length, width, angles,and diameter.
Counting	*Used to count the number of objects or components in a certain area or line of production to check the right number. Especially in industries where it is crucial to check the items per package, for example, in packaging industries.*
Decoding	*This includes understanding barcodes, QR codes, data matrix codes, and OCR fonts among others. This function plays an important role in identifying and tracing products in the supply chain.*
Location	*Defines the exact location and attitude of objects or components in a specific area of operation. Applied in the assembly lines to guarantee that the parts are well aligned and located for other operations or joining.*

Table 2.1*: Classification of Modern Machine Vision based on Functions*

In a nutshell, the development of machine vision technology from the general purpose to the specialised and from the initial simple systems to the more complex and reliable one demonstrates that the field is constantly growing to address the growing needs of industrial automation and quality control. With the progression of technologies in the future, machine vision will be more and more significant in improving production efficiency, improving quality of products, and promoting the development of various industries around the world.

2.2 Types of Machine Vision System

Machine vision systems are pivotal technologies categorised into three main types: 1D, 2D, and 3D, all of which are categorised according to the purpose of imaging and the industry in which it will be used. 1D vision systems work at the line-scan approach where the system scans the image one line at a time hence suitable for continuous production such as paper production. They are proficient in real-time defect identification and categorisation which is vital in quality and speed control in fast production lines.

Besides that, 2D vision systems employ digital cameras to capture two-dimensional images in the format of (X, Y) and analyse the reflected intensity patterns. Used in almost all industries, these systems are crucial for activities such as inspection, reading barcodes, tracking objects and surveillance. However, issues like depth perception and variation in the intensity of light can be a hindrance in some cases.

On the other hand, 3D vision systems use several cameras or laser displacement sensors for creating accurate three-dimensional data (X, Y, Z) along with the rotation data. These systems are a necessity in any processes that involve measurement and especially in detecting defects, dimensional measurements, and in guiding robots in manufacturing. It can be seen that they offer spatial perspective and volume calculation which improves operational precision and productivity across various industries.

The impact of 2D and 3D vision systems extends across various industries, transforming manufacturing, automotive, electronics, healthcare, pharmaceuticals, and food sectors. In manufacturing, 2D systems ensure stringent quality control and surface inspection, while 3D systems guarantee precise assembly and dimensional accuracy of complex components. Automotive industries leverage both technologies for verifying part accuracy and facilitating assembly processes with exceptional precision.

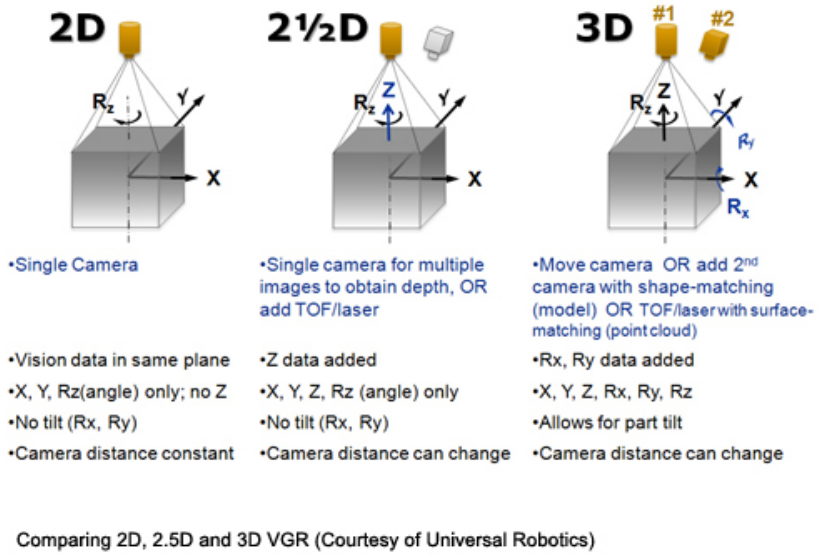

***Figure 2.1**: 2D, 2.5D and 3D Machine Vision Systems*

In conclusion, machine vision systems represent a transformative force in modern manufacturing, enhancing quality control, operational efficiency, and product innovation. This comprehensive literature review underscores their diverse applications, technological capabilities, and future prospects for continued advancement and integration in industrial automation and beyond.

3.0 Methodology

3.1 List of Components

No.	*Component*	*Quantity*
1	*Arduino Nano*	*1*
2	*8051 microcontroller*	*1*
3	*Crystal 11.0592MHz*	*1*
4	*16x2 LCD*	*1*
5	*Push button*	*1*
6	*10k Potentiometer*	*1*
7	*IR Sensor*	*4*
8	*Servo Motor*	*2*
9	*VisionFive 2 RISC-V Board*	*1*
10	*Webcam*	*1*
11	*Motor Driver (10A - Single Channel)*	*1*
12	*Motor*	*1*

Table 3.1*: Component lists*

3.2 Introduction of Components Used

1. Arduino Nano

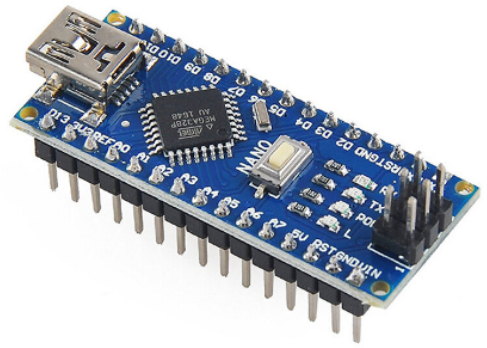

***Figure 3.1**: Arduino Nano*

The Arduino Nano is one of the world's most popular and affordable digital control controllers. The Arduino Nano is a microcontroller-based device with 16 digital pins that can be used for various purposes. It can be used for almost every task, from minor to massive industrial-scale projects. It can also be used for prototyping and developing new applications. Below is the specification of Arduino Nano.

Board	
Name	Arduino® Nano
SKU	A000005
Microcontroller	
ATmega328	
USB connector	
Mini-B USB	
Pins	
Built-in LED Pin	13
Digital I/O Pins	14
Analog input pins	8
PWM pins	6
Communication	
UART	RX/TX

I2C	A4 (SDA), A5 (SCL)
SPI	D11 (COPI), D12 (CIPO), D13 (SCK). Use any GPIO for Chip Select (CS).
Power	
I/O Voltage	5V
Input voltage (nominal)	7-12V
DC Current per I/O Pin	20 mA
Clock speed	
Processor	ATmega328 16 MHz
Memory	
ATmega328P	2KB SRAM, 32KB Flash, 1KB EEPROM
Dimensions	
Weight	7 g
Width	18 mm
Length	45 mm

2. 8051 Microcontroller

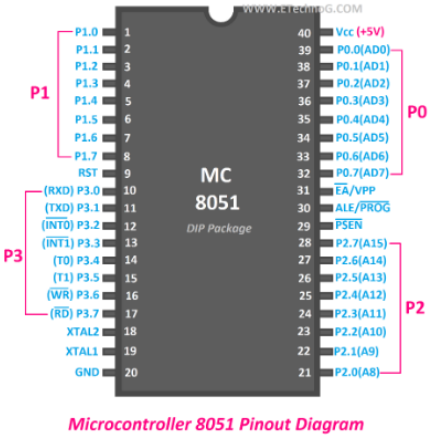

***Figure 3.2**: Pinout Diagram of 8051 Microcontroller*

The 8051-microcontroller chip, which serves as the central processing unit, is integrated into the development board for 8051 microprocessors (CPU). The board also has I/O (input/output) devices such timers, interrupt controllers, serial, and parallel ports, as well as Read Only Memory (ROM), Random Access Memory (RAM), Erasable Programmable Read Only Memory (EPROM/PROM), and ROM.

Ports 0, 1, 2, and 3 are the four ports of an 8051 chip. However, there is only one input/output port available on the 8051-microprocessor development board, designated as PORT 1, which takes up a total of 8 pins. Since it is already built in, a pull-up resistor is not necessary. As it directs the functionality of the electrical devices attached to it, the 8051-microcontroller chip serves as the "brain" of the entire circuit. In order to build a smart security system, we may command the 8051 using computer programming. Below are the features of 8051 microcontroller.

Features:

- 8-bit CPU using Registers A and B.
- Internal ROM is an 8K bytes flash memory that facilitates system programming.
- Internal RAM with 256 Bytes, where the initial RAM's 128 Bytes from 00H to 7FH are once more divided into four banks with 8 registers each, addressable registers with a 16-bit resolution, and general-purpose registers with an 80-bit resolution.
- Special Function Registers are contained in the final 128 bytes of the RAM, from 80H to FFH (SFRs). These registers manage a variety of peripherals, including all I/O ports, serial ports, and timers.
- Internal-3 and External-2 interrupts
- CLK Circuit and the oscillator.
- IE, IP, TMOD, SCON, TMOD, and other control registers.
- 16-bit counters or timers, such as T0 and T1.
- 16-bit programme counter with DPRT (Data Pointer).
- 32 I/O pins, which are organised into four ports (P0, P1, P2, and P3).
- 8-bit and PSW Stack Pointers (Processor Status Word).
- Full-Duplex Operation with Serial Data Tx & Rx

3. Crystal (11.059 MHz)

***Figure 3.3**: Crystal Oscillator*

A crystal oscillator is an electronic oscillator circuit that uses a piezoelectric crystal as a frequency-selective element. The oscillator frequency is often used to keep track of time, as in quartz wristwatches, to provide a stable clock signal for digital integrated circuits, and to stabilise frequencies for radio transmitters and receivers. The most common type of piezoelectric resonator used is a quartz crystal, so oscillator circuits incorporating them became known as crystal oscillators. However, other piezoelectric materials including polycrystalline ceramics are used in similar circuits.

A crystal oscillator relies on the slight change in shape of a quartz crystal under an electric field, a property known as inverse piezoelectricity. A voltage applied to the electrodes on the crystal causes it to change shape; when the voltage is removed, the crystal generates a small voltage as it elastically returns to its original shape. The quartz oscillates at a stable resonant frequency, behaving like an RLC circuit, but with a much higher Q factor (less energy loss on each cycle of oscillation). Once a quartz crystal is adjusted to a particular frequency (which is affected by the mass of electrodes attached to the crystal, the orientation of the crystal, temperature and other factors), it maintains that frequency with high stability.

Quartz crystals are manufactured for frequencies from a few tens of kilohertz to hundreds of megahertz. As of 2003, around two billion crystals are manufactured annually. Most are used for consumer devices such as wristwatches, clocks, radios, computers, and cellphones. However, in applications where small size and weight is needed crystals can be replaced by thin-film bulk acoustic resonators, specifically if ultra-high frequency (more than roughly 1.5 GHz) resonance is needed. Quartz crystals are also found inside test and measurement equipment, such as counters, signal generators, and oscilloscopes.

4. 16 X 2 LCD

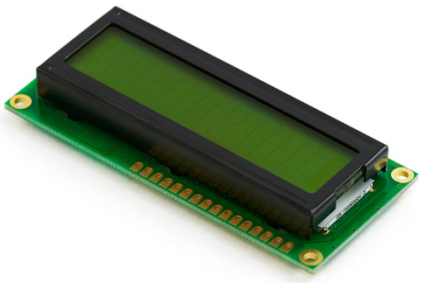

Figure 3.4: *16X2 LCD Module*

An LCD screen is an electronic display module that uses liquid crystal to produce a visible image. The 16×2 LCD display is a very basic module commonly used in DIYs and circuits. The 16×2 translates a display of 16 characters per line into 2 such lines. In this LCD, each character is displayed in a 5×7 pixel matrix.

Pin No.	Function	Name
1	Ground (0V)	Ground
2	Supply voltage; 5V (4.7V – 5.3V)	Vcc
3	Contrast adjustment; the best way is to use a variable resistor such as a potentiometer. The output of the potentiometer is connected to this pin. Rotate the potentiometer knob forward and backward to adjust the LCD contrast.	Vo / VEE
4	Selects command register when low, and data register when high	RS
5	Low to write to the register; High to read from the register	Read/write
6	Sends data to data pins when a high to low pulse is given; Extra voltage push is required to execute the instruction and EN(enable) signal is used for this purpose. Usually, we set en=0, when we want to execute the instruction we make it high en=1 for some milliseconds. After this we again make it ground, that is, en=0.	Enable
7	8-bit data pins	DB0
8		DB1
9		DB2
10		DB3
11		DB4
12		DB5
13		DB6
14		DB7

15	LED Backlight VCC (5V)	Led+
16	LED Backlight Ground (0V)	Led-

5. Push Button

Figure 3.5: *Tactile Switch*

When you press the button, it makes or breaks an electrical connection, allowing or interrupting the flow of electricity through the circuit. Pushbutton switches are commonly used in various electronic devices, control panels and applications where a user needs to activate or deactivate a function or operation.

6. 10k Potentiometer

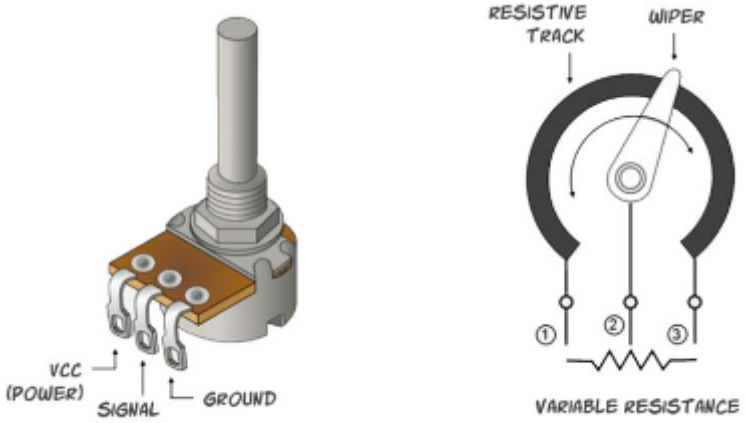

Figure 3.6: *Connection Diagram and Internal Structure of Potentiometer*

A potentiometer is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. If only two terminals are used, one end and the wiper, it acts as a variable resistor or rheostat. The measuring instrument called a potentiometer is essentially a voltage divider used for measuring electric potential (voltage); the component is an implementation of the same principle, hence its name. Potentiometers are commonly used to control electrical devices such as volume controls on audio equipment. It is also used in speed control of fans.

7. IR Sensor

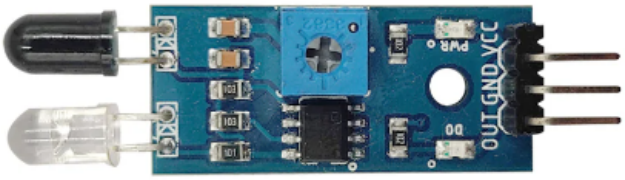

Figure 3.7: *IR Module*

An infrared (IR) sensor is an electronic device that measures and detects infrared radiation in its surrounding environment. Infrared radiation was accidentally discovered by an astronomer named William Herchel in 1800. While measuring the temperature of each color of light (separated by a prism), he noticed that the temperature just beyond the red light was highest. IR is invisible to the human eye, as its wavelength is longer than that of visible light (though it is still on the same electromagnetic spectrum). Anything that emits heat (everything that has a temperature above around five degrees Kelvin) gives off infrared radiation.

There are two types of infrared sensors: active and passive. Active infrared sensors both emit and detect infrared radiation. Active IR sensors have two parts: a light emitting diode (LED) and a receiver. When an object comes close to the sensor, the infrared light from the LED reflects off of the object and is detected by the receiver. Active IR sensors act as proximity sensors, and they are commonly used in obstacle detection systems (such as in robots).

8. Servo Motor

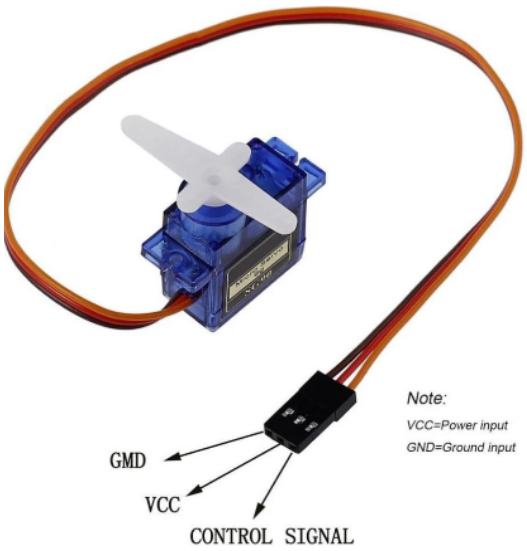

Figure 3.8: *Servo Motor*

A **servo motor** is a type of motor that can rotate with great precision. Normally this type of motor consists of a control circuit that provides feedback on the current position of the motor shaft, this feedback allows the servo motors to rotate with great precision. If you

want to rotate an object at some specific angles or distance, then you use a servo motor. It is just made up of a simple motor which runs through a **servo mechanism**.

Since the servo motor only has three connecting wires, two of which (red and black) are used to power it up and a third (orange) is used to receive signals, its operating principle and method of operation are quite straightforward. The angular position is determined by the width of the pulse at the control input and is controlled by pulse width modulated (PWM) waves. In this project, we are utilising a servo motor with a rotational angle that ranges from 0° to 180° and an angle that can be regulated by changing the duty cycle between 1ms and 2ms.

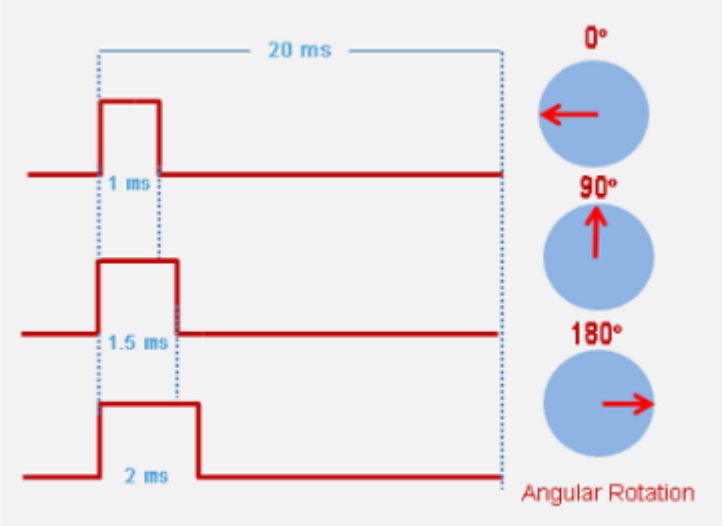

Figure 3.9: *Working Principle of Servo Motor*

9. VisionFive 2 RISC-V Board

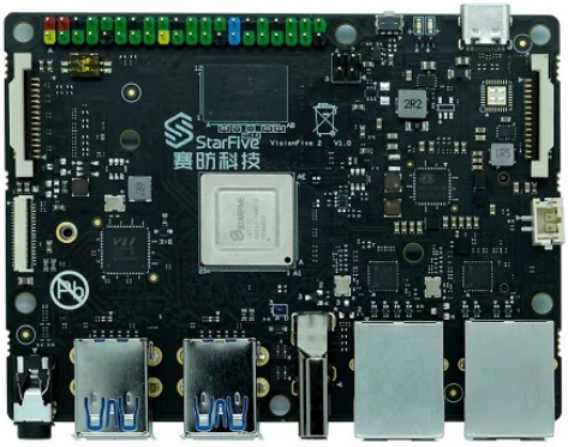

Figure 3.10: *VisionFive 2 Development Board*

Serves as the central controller for the system. It processes inputs from the sensors, controls the actuators, and manages the display and indicators. The VisionFive 2 board's robust performance and versatile connectivity options (GPIO, I2C, PWM) make it ideal for integrating various components required for this mini project. The specification of VisionFive 2 RISC-V Board as below.

Item	Description
Processor/SOC	StarFive JH7110 64bit SoC with RV64GC, up to 1.5GHz
Memory	LPDDR4, 2GB/4GB/8GB
Storage	TF card slot
	Flash for Uboot
Video output	HDMI 2.0
	MIPI-DSI
Multimedia	Camera with MIPI CSI, up to 4k@30fps
	H.264 & H.265 4k@60fps Decoding H.265 1080p@30fps Encoding JPEG encoder/decoder
	1/ 4-pole stereo audio 2. HDMI
Connectivity	2x RJ45 Gigabit Ethernet
	2x USB2.0 + 2x USB 3.0
	M.2 M-Key
Power	USB-C port, 5V DC via USB-C with PD, up to 30W
	GPIO Power in, 5V DC via GPIO header (minimum 3A+)
	PoE
GPIO	40 pin GPIO header
Dimensions	100 x 72mm
Compliance	RoHS, FCC, CE
Button	Reset Button
Other	Debug Pin Headers

10. Webcam

Figure 3.11: *Webcam Used*

A webcam is a digital camera that captures video and audio data and transmits it in real-time over the internet. It is commonly used for video conferencing, live streaming, online meetings, and recording videos. Webcams are typically connected to computers or laptops via

universal serial bus (USB) ports and are often built into devices such as laptops or external monitors.

11. Motor Driver (10A - Single Channel)

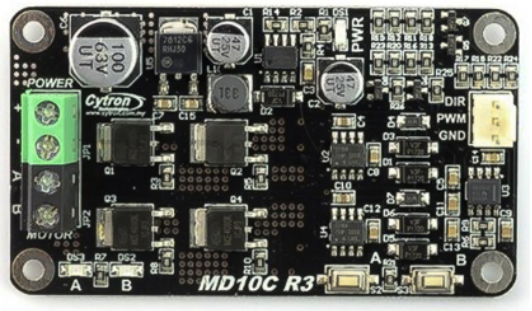

Figure 3.12: *Motor Driver (Cytron)*

Motor drive is the control device for a motor in the system. The motor driver controls the direction of the motor by giving 1 or 0 to the DIR pin of the motor driver when the signals are received from the main board.

12. Motor

Figure 3.12: *12V DC Motor (Cytron)*

Motor used as the actuator for the conveyor belt to move. In addition, it used to drive the conveyor transfer of an object from one place to another place in the production system.

3.3 Circuit Diagram

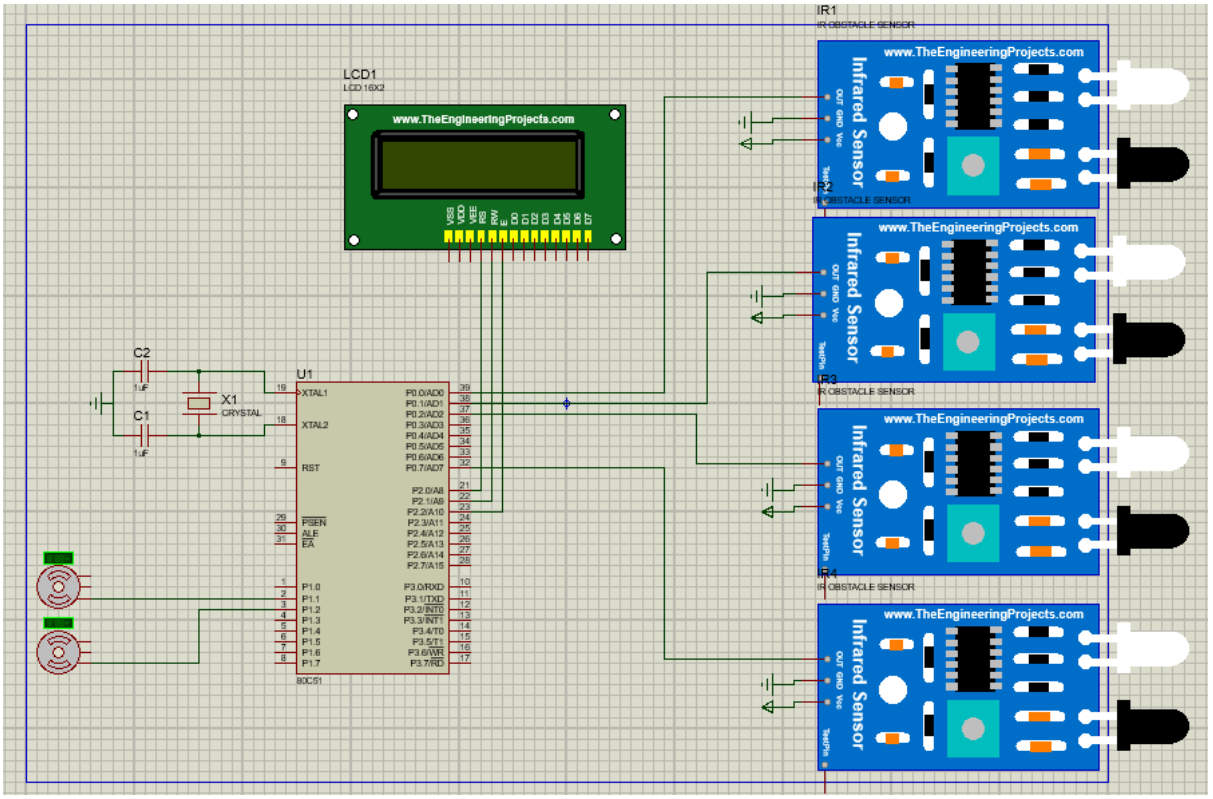

Figure 3.13: *Schematic Diagram of the 8051 Microcontroller*

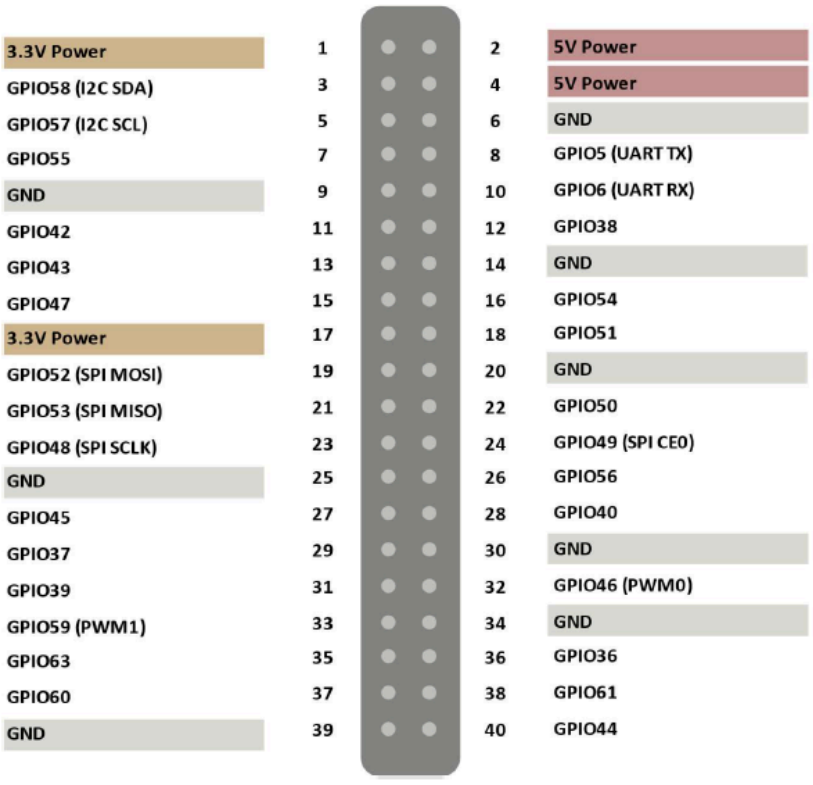

Figure 3.14: *Pinout Diagram of VisionFive 2 Development Board*

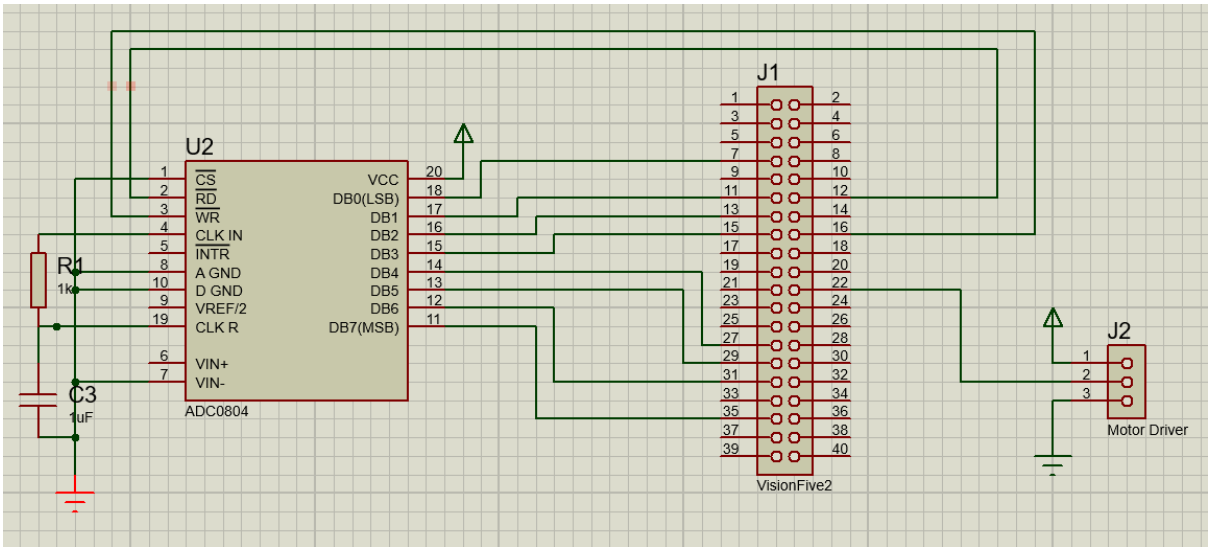

Figure 3.15:*Schematic Diagram of the VisionFive 2 Development Board*

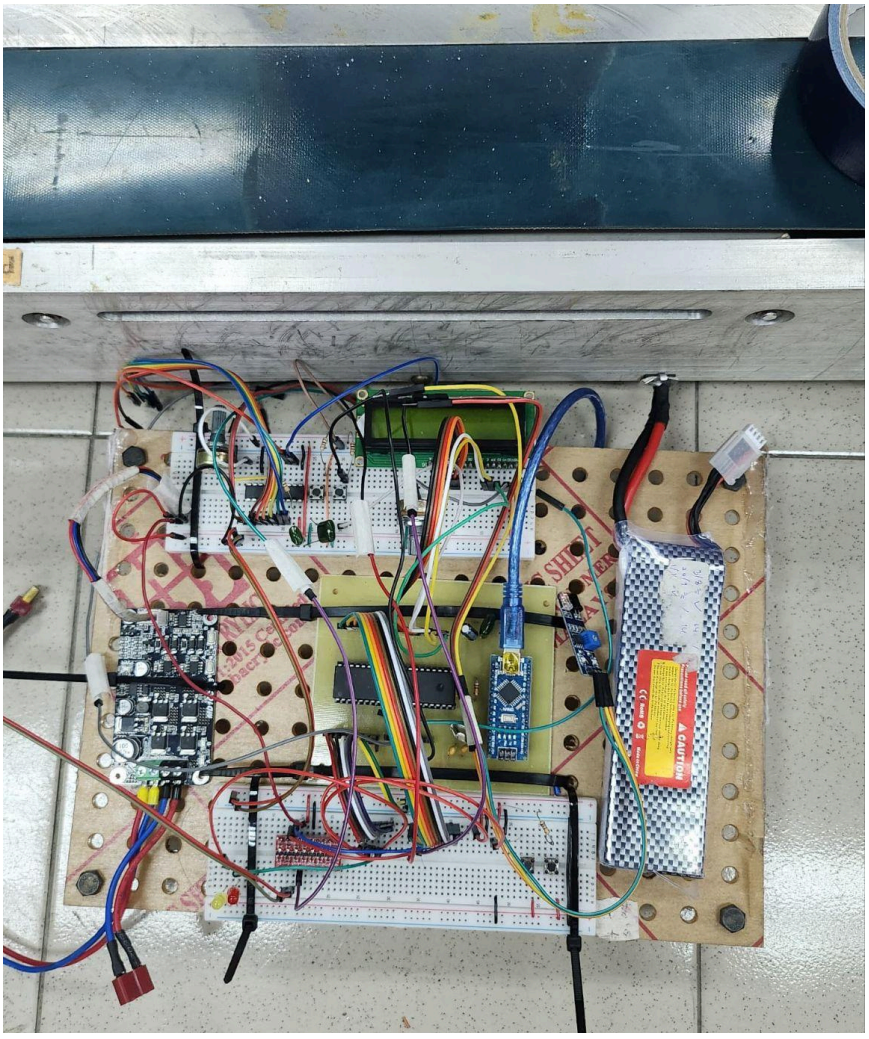

Figure 3.16: *Physical Circuit Diagram*

Port	Connected Pin
0.0	RED
0.1	BLUE
0.2	IRBLUE
0.3	BLUEBOX
0.4	IRRED
0.5	-
0.6	-
0.7	REDBOX
1.0	-
1.1	RED SERVO
1.2	BLUE SERVO
1.3	-
1.4	-
1.5	-
1.6	-
1.7	-
2.0	RS
2.1	RW
2.2	EN
2.3	WK
2.4	-
2.5	-
2.6	-

2.7	-
3.0	-
3.1	-
3.2	-
3.3	-
3.4	-
3.5	INT from VisionFive board
3.6	-
3.7	-

3.4 Program Flow

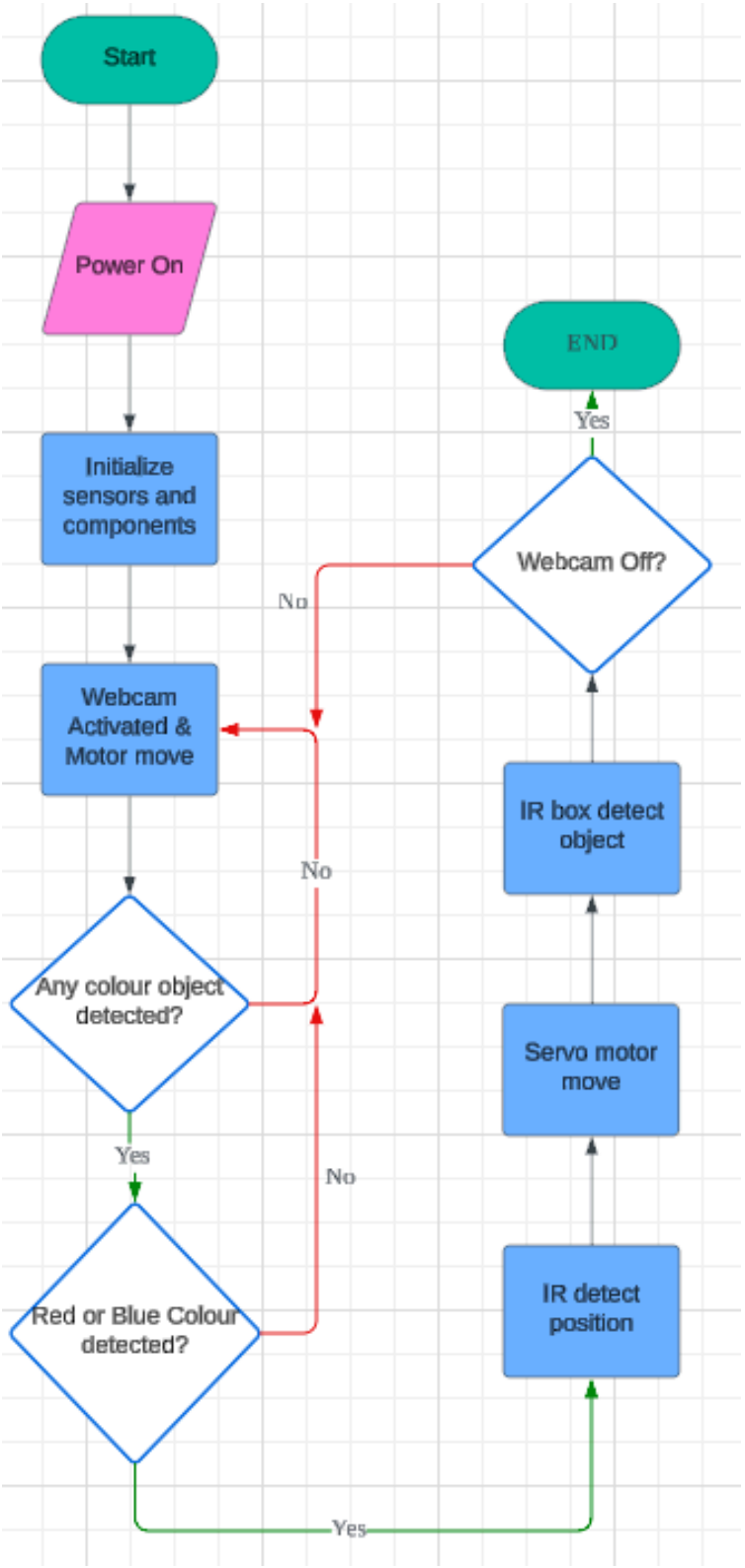

Figure 3.17: *FlowChart of Program*

3.5 System Analysis

Our project is an automated colour filter production line which automatically filters red and blue colour. In the first, all components and sensors are initialised while the power is turned ON. For example, LCD is initialised and displays the initial quantities of red and blue objects that have been filtered. Next, the motor started to move and drive the motion of the conveyor belt when the webcam was activated , ready to scan for colours.

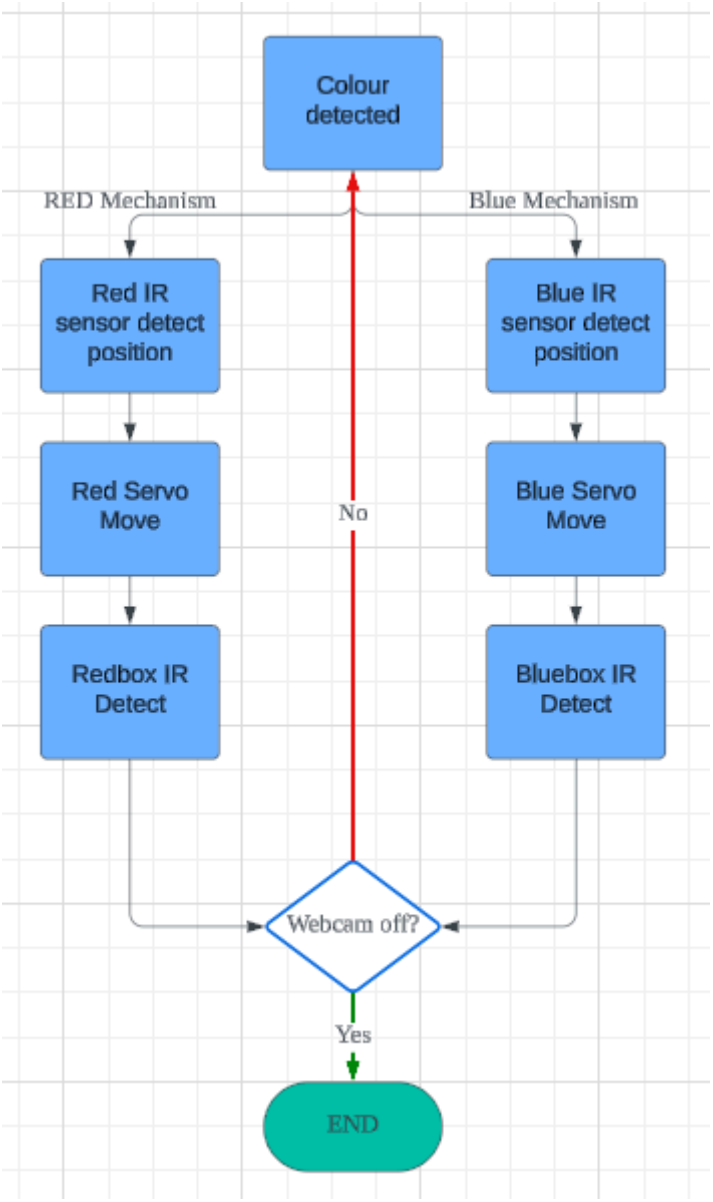

Figure 3.18: *FlowChart of Colour Detection System*

When the colour detected is red, the red mechanism will be triggered to filter red colour objects while vice versa when blue colour is detected. These functions are done by using 2 GPIO pins from VisionFive 2 RISC-V board to 8051 microcontroller. Webcam is connected to the VisionFive board, pin 26 is sending high signal to 8051 microcontroller when red is detected while pin 228 is sending high signal to 8051 microcontroller when blue is detected. An IR sensor will be used to make sure the object comes to the correct position, once the IR sensor detects the object, the servo motor moves to filter out the object into a

box. Next, another IR sensor is placed at the box to detect the object falling into the box, the servo motor moves back to initial position while the IR sensor at the box senses the object. The webcam is ready to scan for the second object colour once it receives the signal from the IR sensor in the box which represents the object falling into the box. After the object is falling to the box, the quantity on the LCD display +1 to the respected colour detected and the total unit +1. For example, a red object is filtered and falls to the box, the red unit on the LCD unit increases from 0 to 1 and Total unit also changes from 0 to 1.

To adjust the speed of the motor that drives the conveyor belt, an ADC0804 is connected to VisionFive board to convert analog signal to digital signal which can be read for VisionFive board. A potentiometer is connected to the ADC0804 to represent and adjust the speed of the motor by changing the voltage input to the V+ pin of the ADC0804.

4.0 Results Presentation and Interpretation

4.1 Result Presentation

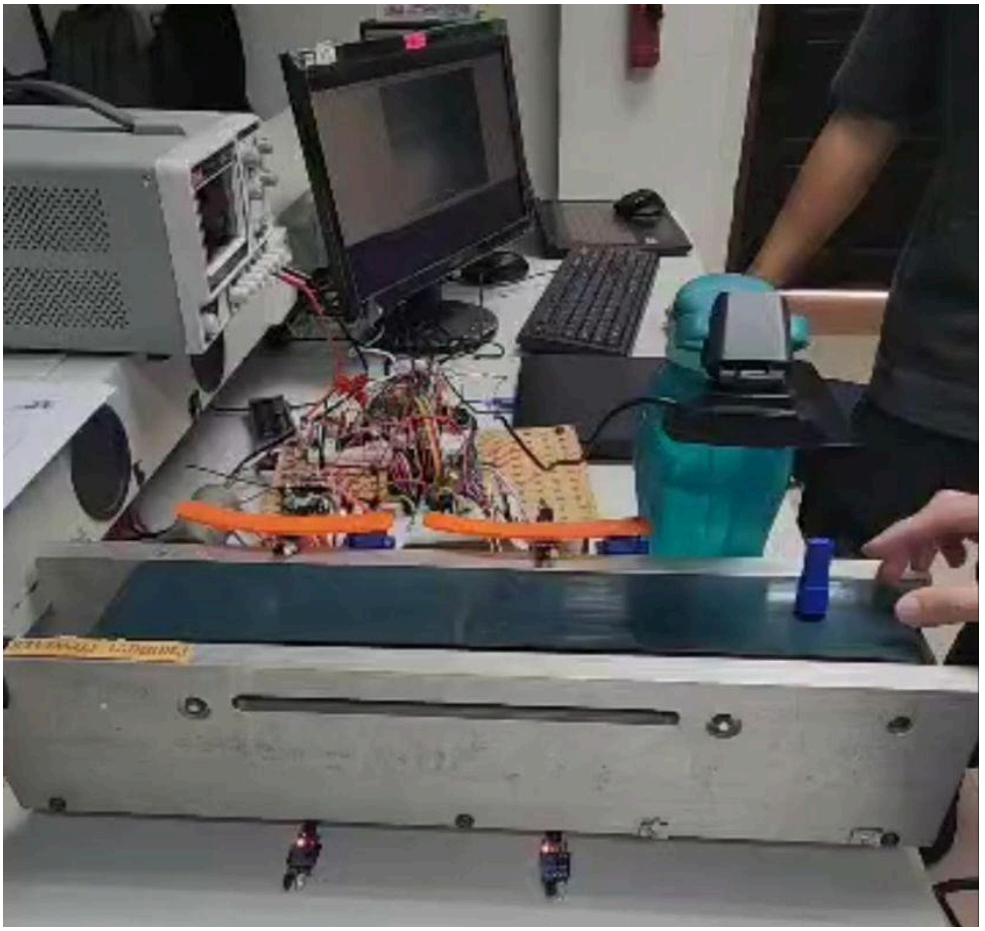

Figure 4.1: *Project Setup*

At first, the project is set up as shown in the figure above. A webcam is set up at the top of the conveyor belt to scan colour. 2 servo motors with the orange colour filtering mechanism are set to filter 2 colours. 2 IR sensors are located below the orange mechanism of the servo motor which is used to detect the position of the object so that it will come to the correct position. Another 2 IR sensors are located below the conveyor belt and on the table are used to detect the object that falls to the box.

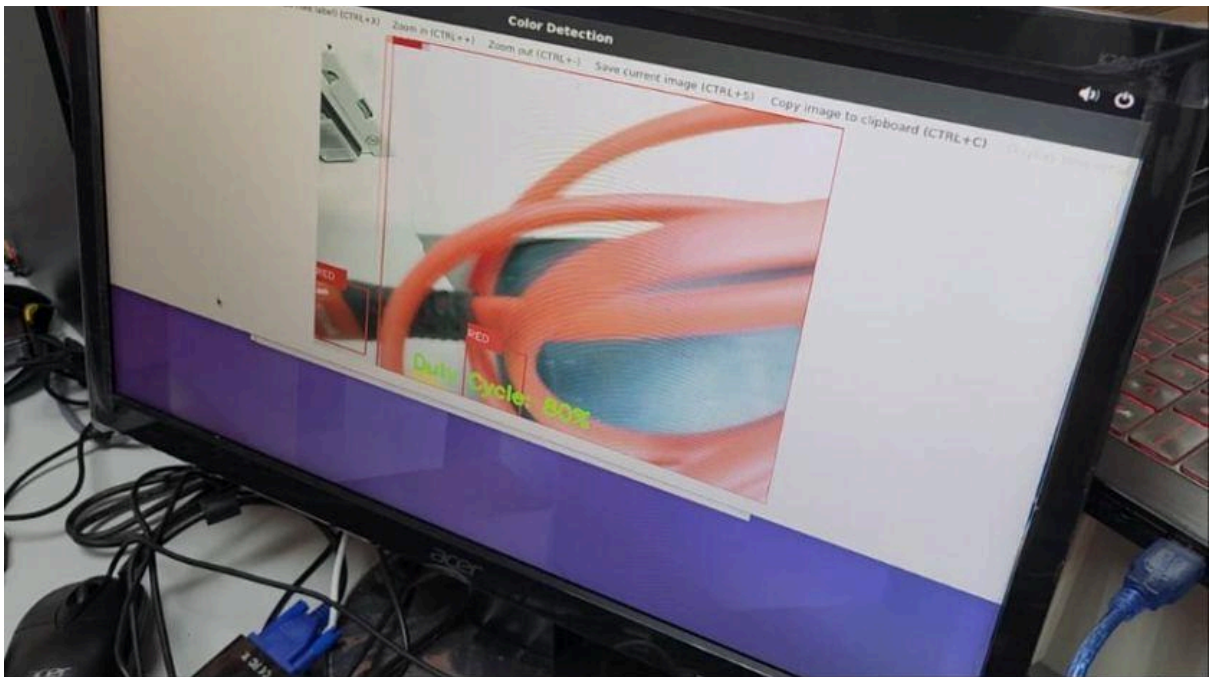

Figure 4.2: *Red Colour Detected and shown on the screen*

Figure above shows the red colour is detected by the webcam and shown on the monitor screen.

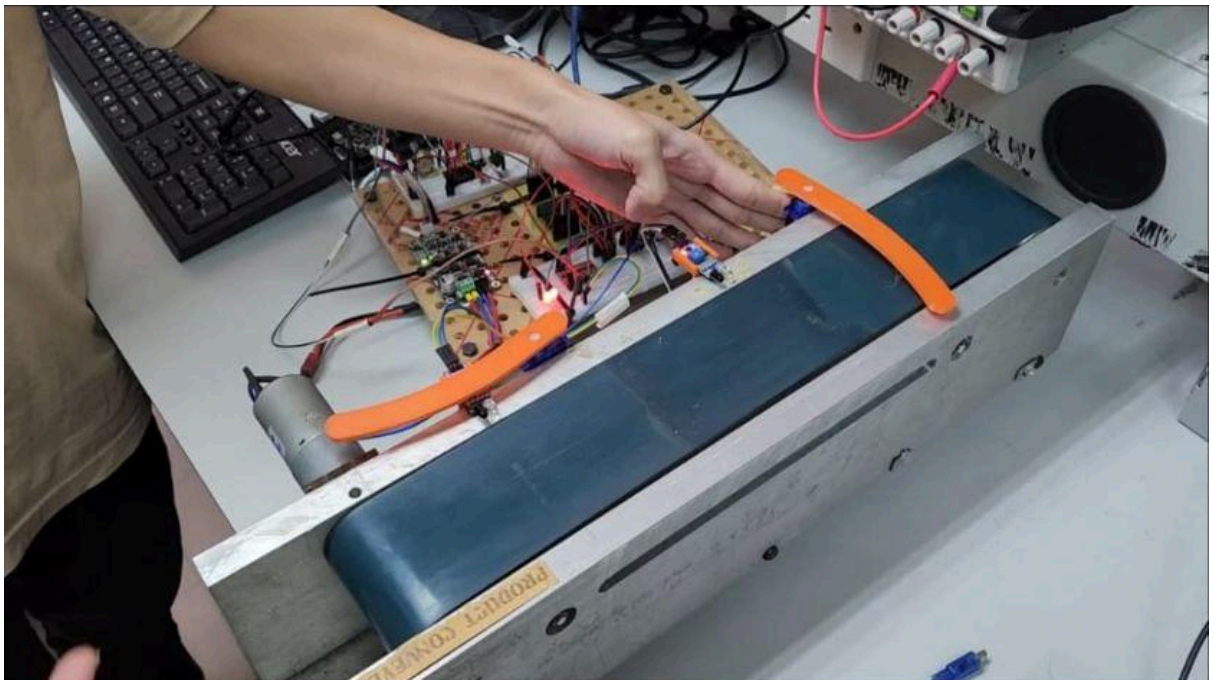

Figure 4.3: *Red Servo Mechanism Activated*

Figure above shows the red mechanism is triggered when the red object is detected. The servo motor will start to move once the IR sensor is detecting the object. Nex, the servo motor will remain in the position until the bottom IR sensor detects that the object falls into the box and the servo motor turns back to its original position.

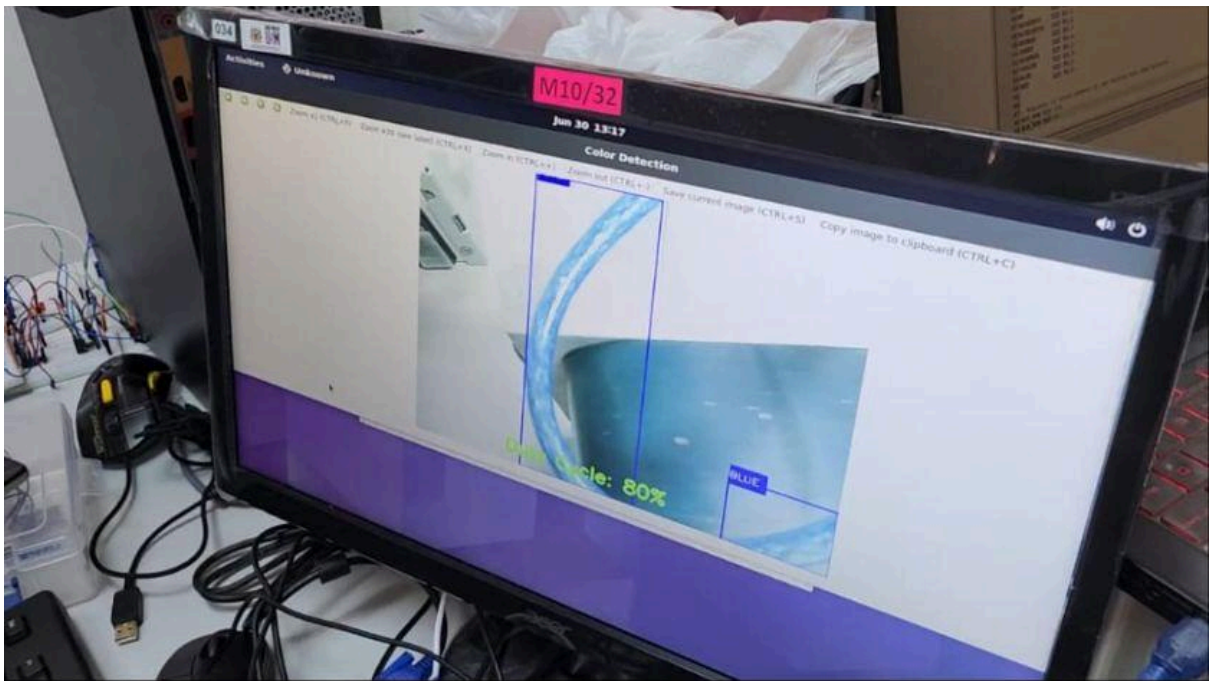

Figure 4.4: *Blue Colour detected and shown on the screen*

Figure above shows the red colour is detected by the webcam and shown on the monitor screen.

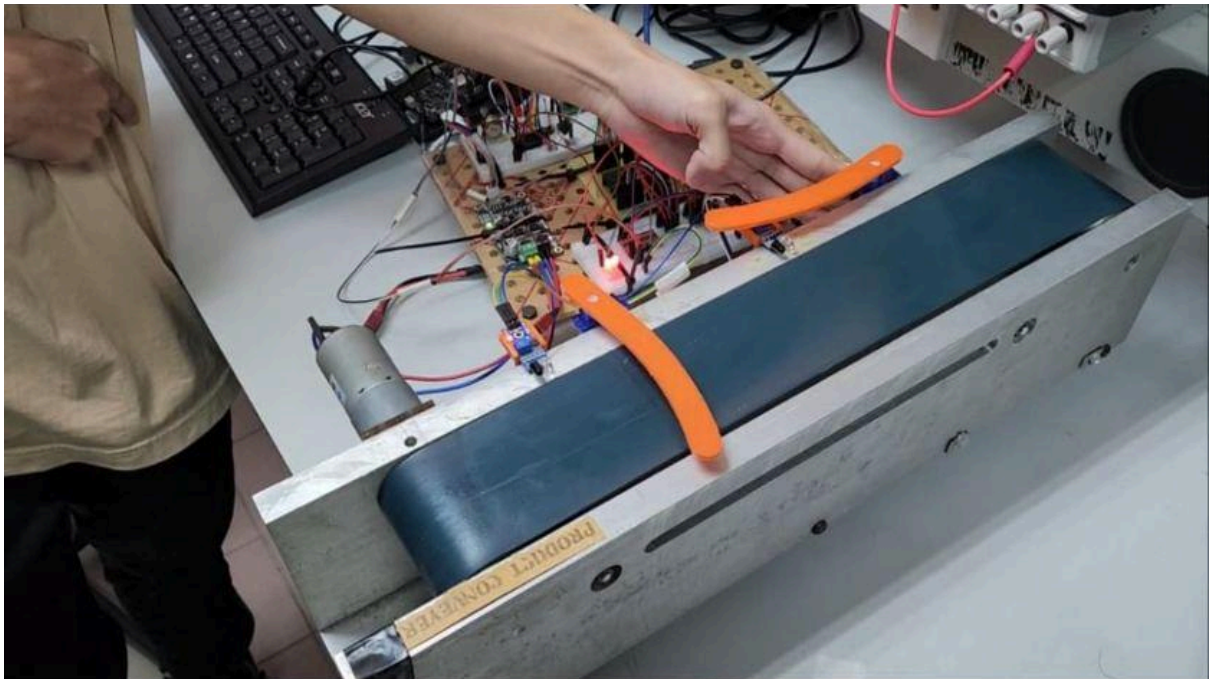

Figure 4.5: *Blue Servo Mechanism Activated*

Figure above shows the blue mechanism is triggered when the blue object is detected. The servo motor will start to move once the IR sensor is detecting the object. Next, the servo motor will remain in the position until the bottom IR sensor detects that the object falls into the box and the servo motor turns back to its original position.

4.2 Result Interpretation

4.2.1 8051 Microcontroller Assembly Codes

I. ISR Program Setup

```
ORG 0000H
LJMP INITIAL

; ---- ISR0 PROGRAM ----
ORG  0003H                    ; INT0 Memory Location
ACALL WAKEUP
RETI

; ---- ISR1 PROGRAM ----
ORG 0013H                     ; INT1 Memory Location
LOOP5:
        CLR P1.4
        LCALL DELAY
        JB RED,  RM1 ;Red colour detected
        JB BLUE, BM1; Blue colour detected
        SETB P1.4
        LCALL DELAY
        SJMP LOOP5

RM1:
        LCALL RM
        RETI

BM1:
        LCALL BM
        RETI
```

This 8051 assembly code begins execution at address `0000H` with a jump instruction to the `INITIAL` subroutine, initialising system states and configuring necessary peripherals. The program defines two interrupt service routines (ISRs): `ISR0` at address `0003H` for `INT0` and `ISR1` at address `0013H` for `INT1`. `ISR0` calls the `WAKEUP` subroutine upon interrupt and returns, likely handling wake-up events or external triggers. `ISR1` continuously loops (`LOOP5`), toggling `P1.4` and delaying with `DELAY` calls, checking conditions for red (`RED`) and blue (`BLUE`) detections. Upon detection, it branches to `RM1` or `BM1` subroutines, which presumably handle specific actions for red or blue detections before returning from interrupt. This structure suggests a real-time monitoring or control application where interrupts are used for responsive handling of external events, such as colour detection signals.

II. Pin and Variables Initialization

```
ORG 0100H
; ---- Pin Initialization and Configurations ----
; Define SFRs and bit variables
//SLP                   EQU P1.0      ; SLEEP PIN
RS              EQU P2.0
RW              EQU P2.1
EN              EQU P2.2
WK              EQU P2.3      ; IDLE AND WAKEUP
REDSERVO   EQU P1.1
BLUESERVO          EQU P1.2
REDBOX     EQU P0.7
IRRED      EQU P0.4
BLUEBOX    EQU P0.3
IRBLUE     EQU P0.2
BLUE       EQU P0.1
RED        EQU P0.0


; Register to count number of red objects and blue objects
RED_NUM EQU 00H
BLE_NUM EQU R5

; ---- VARIABLES INITIALIZATION ----
INITIAL:
        MOV RED_NUM, #0x00  ; Initialize red count to 0
        MOV BLE_NUM, #0x00  ; Initialize yellow count to 0
        MOV TMOD, #01H ; Set Timer 0 to 16 bit mode

        ; Clear P1 and P2
        MOV P1, #0FFH
        MOV P2, #00FH
```

This 8051 assembly code snippet begins at memory address 0100H and initialises various special function registers (SFRs) and bit variables for pin configurations. Symbols like RS, RW, EN, WK, REDSERVO, BLUESERVO, REDBOX, IRRED, BLUEBOX, IRBLUE, BLUE, and RED are defined to correspond to specific GPIO pins for tasks such as control signals, sensor inputs, and indicator LEDs. Two registers, RED_NUM and BLE_NUM, are allocated for counting red and blue objects, respectively. The INITIAL subroutine initialises RED_NUM and BLE_NUM to zero and sets TMOD to configure Timer 0 for 16-bit mode. Additionally, it clears ports P1 and P2 for subsequent operations such as input/output control or data manipulation in the main program flow.

III. Starting Program (Only Run Once for Interrupt, Timer and LCD Initialization)

```
; ---- MAIN PROGRAM ----
START:
  MOV IE,  #85H            ; Enable global interrupt, INT0, INT1
       ; Select INT0 and INT1 on falling edge
       MOV TCON, #05H

  ; Initialization of LCD Module
  MOV A, #02H
  ACALL SEND_COMMAND
  ACALL MS_DELAY
  MOV A, #28H
  ACALL SEND_COMMAND
  ACALL MS_DELAY
  MOV A, #0CH       ; Display ON, Cursor OFF
  ACALL SEND_COMMAND
  ACALL MS_DELAY
  MOV A, #01H
  ACALL SEND_COMMAND
  ACALL MS_DELAY
  MOV A, #80H       ; Force cursor to beginning of first line
  ACALL SEND_COMMAND
  ACALL MS_DELAY
       SJMP MAIN

POWER:
       LJMP SLEEPMODE
```

Assembly code above begins execution at the START label, initialising the 8051 microcontroller for interrupt handling and setting up an LCD module. The MOV IE, #85H instruction enables global interrupts and specifically enables interrupts INT0 and INT1 on the falling edge, configuring them to trigger interrupt service routines (ISRs) when the specified conditions are met. Subsequently, the MOV TCON, #05H sets the control register TCON to enable INT0 and INT1 interrupts on the falling edge. The code then proceeds to initialise the LCD module by sending a series of commands (02H, 28H, 0CH, 01H, 80H) through a subroutine SEND_COMMAND, followed by delays (ACALL MS_DELAY) to ensure proper timing for each command execution. After initialization, control transfers to MAIN for further program execution. The POWER subroutine, invoked by LJMP SLEEPMODE for handling power management or sleep mode operations which will be explained on the latter program.

IV. Main Program

```
MAIN:
        CPL P1.3
        ; Entering IDLE Mode or not
        JNB P1.0, POWER

    MOV A, #80H        ; Continuously Refresh without blinking
    ACALL SEND_COMMAND
    ACALL MS_DELAY

    ; First Line of Words
    MOV DPTR, #LINE1
WRITE_FIRST:
        CLR A
    MOVC A, @A+DPTR
        JZ TOTALUNIT
        INC DPTR
        ACALL SEND_DATA
        SJMP WRITE_FIRST

TOTALUNIT:
    ; Write Total Unit
    MOV A, RED_NUM
    ADD A, BLE_NUM
    ACALL DISPLAY_NUMBER

    ; Second Line
    MOV A, #0C0H
    ACALL SEND_COMMAND
    ACALL MS_DELAY

    ; Second Line of Words
    MOV DPTR, #LINE2
WRITE_SECOND:
        CLR A
        MOVC A, @A+DPTR
        JZ BLUEUNIT
        INC DPTR
        ACALL SEND_DATA
        SJMP WRITE_SECOND

BLUEUNIT:
    ; Write Blue Number Units
    MOV A, BLE_NUM
    ACALL DISPLAY_NUMBER

    ; Spacebar
    MOV A, #20H        ; ASCII code for space
    ACALL SEND_DATA

    ; Third Line of Words
    MOV DPTR, #LINE3
WRITE_THIRD:
        CLR A
        MOVC A, @A+DPTR
        JZ REDUNIT
        INC DPTR
        ACALL SEND_DATA
        SJMP WRITE_THIRD

REDUNIT:
    ; Write Red Number Units
    MOV A, RED_NUM
    ACALL DISPLAY_NUMBER

    ACALL MS_DELAY
    LJMP MAIN

// ~~~~~~~~~~~~~~ END OF PROGRAM ~~~~~~~~~~~~~~~
// ========================================
```

In the `MAIN` subroutine of this 8051 assembly code snippet, the program begins by toggling the state of pin P1.3 using the `CPL P1.3` instruction. It then checks if pin P1.0 is not active low (`JNB P1.0, POWER`), indicating a condition to enter power-saving or idle mode, where control is transferred to the `POWER` subroutine for further handling. If P1.0 is active low, the program proceeds to refresh the LCD display continuously without blinking by sending the command `MOV A, #80H` to position the cursor at the beginning of the first line (`ACALL SEND_COMMAND` and `ACALL MS_DELAY` ensure proper timing).

Next, the program iterates through three lines of text stored in memory (`LINE1`, `LINE2`, `LINE3`) using a loop (`WRITE_FIRST`, `WRITE_SECOND`, `WRITE_THIRD`). It reads characters sequentially from memory pointed by `DPTR`, sends each character to the LCD display using `ACALL SEND_DATA`, and increments `DPTR` until a null character (`JZ TOTALUNIT`, `JZ BLUEUNIT`, `JZ REDUNIT`) is encountered, signalling the end of each line. After displaying the lines of text, the program calculates the total count of units (`RED_NUM` and `BLE_NUM`), displays them sequentially on the LCD (`ACALL

DISPLAY_NUMBER`), and continues this process indefinitely (`LJMP MAIN`), ensuring continuous display update and unit counting operations.

V. Servo Mechanism

```
// ~~~~~~~~~~~~~~ Servo Mechanism ~~~~~~~~~~~~~~~
;Red mechanism ready
RM:
        JB IRRED, $ ; Wait for IR to detect
        INC RED_NUM
        BACK: JB REDBOX, RSERVO_90 ; Servo turn to 90 degree until red box read high
        ACALL RSERVO_0 ; turn back to 0 degree
        SETB P1.4
        RET

;Blue mechanism ready
BM:
        JB IRBLUE, $ ; Wait for IR to detect
        INC BLE_NUM
        BACK2:JB BLUEBOX, BSERVO_90 ; Servo turn to 90 degree until blue box read
high
        ACALL BSERVO_0 ; turn back to 0 degree
        SETB P1.4
        RET

;Set up for servo turn to 90 degree
RSERVO_90:
        SETB REDSERVO
        ACALL DELAY1_4ms
        CLR REDSERVO
        ACALL DELAY18_6ms
        SJMP BACK

BSERVO_90:
        SETB BLUESERVO
        ACALL DELAY1_4ms
        CLR BLUESERVO
        ACALL DELAY18_6ms
        SJMP BACK2

;Set up for servo turn to 0 degree
RSERVO_0:
        MOV R1, #20D ; repeat for 20 times to send 20 pulse
        Loop:
                SETB REDSERVO
                ACALL DELAY1ms
                CLR REDSERVO
                ACALL DELAY19ms
                DJNZ R1, Loop
                RET

BSERVO_0:
        MOV R1, #20D ; repeat for 20 times to send 20 pulse
        Loop3:
                SETB BLUESERVO
                ACALL DELAY1ms
                CLR BLUESERVO
                ACALL DELAY19ms
                DJNZ R1, Loop3
                RET

;Set up for 1ms delay
DELAY1ms:
        MOV TH0, #0FCH
        MOV TL0, #67H
        SETB TR0 ; Start Timer0
        JNB TF0, $ ;Wait until Timer0 finish counting
        CLR TR0 ; Clear Timer0
        CLR TF0 ; Clear Timer Flag 0
        RET

;Set up for 1.4ms delay
DELAY1_4ms:
        MOV TH0, #0F8H
        MOV TL0, #0ADH
        SETB TR0 ; Start Timer0
        JNB TF0, $ ;Wait until Timer0 finish counting
        CLR TR0 ; Clear Timer0
        CLR TF0 ; Clear Timer Flag 0
        RET

;Set up for 19ms delay
DELAY19ms:
        MOV TH0, #0BBH
        MOV TL0, #99H
        SETB TR0 ; Start Timer0
        JNB TF0, $ ;Wait until Timer0 finish counting
```

```
        CLR TR0 ; Clear Timer0
        CLR TF0 ; Clear Timer Flag 0
        RET

;Set up for 18.6ms delay
DELAY18_6ms:
        MOV TH0, #0B3H
        MOV TL0, #0A6H
        SETB TR0 ; Start Timer0
        JNB TF0, $ ;Wait until Timer0 finish counting
        CLR TR0 ; Clear Timer0
        CLR TF0 ; Clear Timer Flag 0
        RET

DELAY:
        MOV R2,#3D
LOOP2:MOV R3, #255D
LOOP1:MOV R4, #255D
        DJNZ R4, $
        DJNZ R3, LOOP1
        DJNZ R2, LOOP2
        RET
```

The 8051 assembly code above outlines the servo mechanism control for both the red and blue mechanisms based on infrared (IR) detection and input signals from respective boxes. Each mechanism (RM for red and BM for blue) begins by waiting for the IR sensor (IRRED or IRBLUE) to detect an object, incrementing the count (RED_NUM or BLE_NUM), and then executing servo movements based on box inputs (REDBOX or BLUEBOX). The RSERVO_90 and BSERVO_90 subroutines set the corresponding servo pins (REDSERVO or BLUESERVO) high for a specific delay (DELAY1_4ms) to turn the servo to 90 degrees, followed by setting the pin low after another delay (DELAY18_6ms). If the box input is not detected (JB REDBOX, RSERVO_90 or JB BLUEBOX, BSERVO_90), the servo resets to 0 degrees using RSERVO_0 or BSERVO_0, which sends a series of pulses (LOOP and DJNZ) to achieve the desired position. The delay routines (DELAY1ms, DELAY1_4ms, DELAY19ms, DELAY18_6ms) utilise the 8051's timer (TH0, TL0, TR0, TF0) to manage precise timing intervals necessary for servo control, ensuring accurate and synchronised movements in the servo mechanism operations.

VI. Power Mode (SLEEP and WAKEUP)

```
; ~~~~~~~~~~~~~~~~~~~~~~~~~~ POWER MODE ~~~~~~~~~~~~~~
; SLEEPMODE, WAKEUP
SLEEPMODE:
        ; ---- CLEAR DISPLAY SCREEN ----
        MOV A, #01H
   ACALL SEND_COMMAND
   ACALL MS_DELAY
   MOV A, #80H        ; Force cursor to beginning of first line
   ACALL SEND_COMMAND
   ACALL MS_DELAY
        ; ---- FINISIH CLEARING ----
        ; ---- SLEEP WORDS ----
   MOV DPTR, #SLEEP
        WRITE_SLEEP:
        CLR A
        MOVC A, @A+DPTR
        INC DPTR
        ACALL SEND_DATA
        JZ ENTERSLP
        SJMP WRITE_SLEEP
        ACALL S_DELAY
        ENTERSLP:
        ; ----
        MOV A, #0FFH
        ; ---- ENTER SLEEP MODE ----
        ORL 87H, #01H       ; PCON (IDLE MODE)

JUMP:
        LJMP MAIN
```

```
WAKEUP:
        ; ---- MISPRESS ----
        CJNE A, #0FFH, JUMP
        ; ---- WAKING UP FROM SLEEP MODE ----
        ANL 87H, #00H
        ; ---- CLEAR DISPLAY SCREEN ----
        MOV A, #01H
   ACALL SEND_COMMAND
   MOV A, #80H        ; Force cursor to beginning of first line
   ACALL SEND_COMMAND
        ; ---- FINISIH CLEARING ----
        ; ---- WAKEUP WORDS ----
   MOV DPTR, #WAKE
        WRITE_WAKE:
        CLR A
        MOVC A, @A+DPTR
        INC DPTR
        JZ JUMPBACK
        ACALL SEND_DATA
        SJMP WRITE_WAKE
        JUMPBACK:
        ACALL S_DELAY
        MOV A, #01H
   ACALL SEND_COMMAND
        RET
```

This segment of 8051 assembly code manages the sleep mode functionality for the system. In the `SLEEPMODE` subroutine, the display screen is first cleared by sending commands (`SEND_COMMAND`) to the LCD module to clear the display (`MOV A, #01H`) and position the cursor at the beginning of the first line (`MOV A, #80H`). Subsequently, it displays the word "SLEEP" on the LCD by fetching characters from the `SLEEP` string using `MOVC A, @A+DPTR` and sending them to the display (`SEND_DATA`). After displaying the sleep message, a short delay (`S_DELAY`) is introduced before the microcontroller enters sleep mode. This is achieved by setting bit 0 (`ORL 87H, #01H`) in the `PCON` (Power Control) register (`87H`), which triggers the microcontroller to enter idle mode (`IDLE MODE`).

The `WAKEUP` subroutine handles waking up from sleep mode. It begins with a check to ensure that the wake condition (`CJNE A, #0FFH, JUMP`) is met before proceeding. Upon waking, it clears the display screen again (`MOV A, #01H` followed by `SEND_COMMAND` to position the cursor), displays the word "WAKEUP" on the LCD (`WRITE_WAKE` subroutine fetching from `WAKE` string), introduces a delay (`S_DELAY`), and finally clears the display again (`MOV A, #01H`). It then returns to the main program (`RET`). This sequence ensures that upon detecting a wake condition, the system gracefully wakes up, refreshes the display, and resumes normal operation from the `MAIN` subroutine.

VII. Sending Command and Data to DDRAM in LCD Module (4 Bit Communication Mode)

```
; ~~~~~~~~~~~~~~~~~~~~~~~~~~ LCD MODULE ~~~~~~~~
; SEND_COMMAND, SEND_DATA
SEND_COMMAND:
   MOV R6, A          ; Save original value of A
   ANL A, #0F0H       ; Mask lower nibble
   ANL P2, #0x0F      ; Clear upper nibble bits
   ORL P2, A          ; Send upper nibble to LCD
   CLR RS             ; Command mode
   CLR RW             ; Write mode
   SETB EN            ; Enable pulse
   ACALL MS_DELAY     ; Delay
   CLR EN             ; Disable pulse

   MOV A, R6          ; Restore original value of A
   SWAP A             ; Swap nibbles to send lower nibble
   ANL A, #0F0H       ; Mask lower nibble
   ANL P2, #0x0F      ; Clear upper nibble bits
   ORL P2, A          ; Send lower nibble to LCD
   CLR RS             ; Command mode
   CLR RW             ; Write mode
   SETB EN            ; Enable pulse
   ACALL MS_DELAY     ; Delay
   CLR EN             ; Disable pulse
   RET
```

```
SEND_DATA:
   MOV R6, A          ; Save original value of A
   ANL A, #0F0H       ; Mask lower nibble
   ANL P2, #0x0F      ; Clear upper nibble bits
   ORL P2, A          ; Send upper nibble to LCD
   SETB RS            ; Data mode
   CLR RW             ; Write mode
   SETB EN            ; Enable pulse
   ACALL MS_DELAY     ; Delay
   CLR EN             ; Disable pulse

   MOV A, R6          ; Restore original value of A
   SWAP A             ; Swap nibbles to send lower nibble
   ANL A, #0F0H       ; Mask lower nibble
   ANL P2, #0x0F      ; Clear upper nibble bits
   ORL P2, A          ; Send lower nibble to LCD
   SETB RS            ; Data mode
   CLR RW             ; Write mode
   SETB EN            ; Enable pulse
   ACALL MS_DELAY     ; Delay
   CLR EN             ; Disable pulse
   RET
```

The SEND_COMMAND and SEND_DATA subroutines in this 8051 assembly code are essential for interfacing with an LCD module, enabling the microcontroller to send commands and data for display control. Both subroutines utilise a ***4-bit communication mode (to save more GPIO pins on 8051 microcontroller with disadvantages of slower data transfer rates)***, leveraging the P2 port to transmit upper and lower nibbles of the command or data byte to the LCD module.

In SEND_COMMAND, the subroutine starts by isolating and preparing the upper nibble of the command (A) with masking operations (ANL A, #0F0H). It then clears the appropriate bits in P2 and sends the upper nibble by ORing it with P2. The LCD is set to command mode (RS low), and EN is pulsed high briefly to signal the LCD to read the command. After a delay (ACALL MS_DELAY), the lower nibble of the command is processed similarly, ensuring the LCD receives the complete command before disabling EN and returning from the subroutine (RET).

Similarly, SEND_DATA handles the transmission of data to the LCD. It follows a nearly identical process to SEND_COMMAND, except that RS is set high to indicate data mode, distinguishing it from commands. This allows the microcontroller to send alphanumeric characters or other data for display. Timing and synchronisation with the LCD's operational requirements are maintained through carefully timed delays (ACALL MS_DELAY), ensuring that data integrity and display stability are maintained throughout the operation..

VIII. Sending Command and Data to DDRAM in LCD Module (4 Bit Communication Mode)

```
// ~~~~~~~~~~ ASCII NUMBER to LCD ~~~~~~~~~~~~~~
DISPLAY_NUMBER:
   MOV B, #10       ; Divide by 10
   DIV AB           ; A = quotient, B = remainder
   ADD A, #30H      ; Convert to ASCII
   ACALL SEND_DATA   ; Send tens place
   MOV A, B         ; Get remainder
   ADD A, #30H      ; Convert to ASCII
   ACALL SEND_DATA   ; Send units place
   RET

// ~~~~~ DELAY SUBROUTINES (MS_DELAY & S_DELAY) ~~~~
MS_DELAY:
   MOV R1, #100
DELAY_LOOP:
   MOV R2, #255
INNER_DELAY_LOOP:
   DJNZ R2, INNER_DELAY_LOOP
   DJNZ R1, DELAY_LOOP
   RET

S_DELAY:
   MOV R1, #7
SDELAY_LOOP1: MOV R2, #255
SDELAY_LOOP2: MOV R3, #255
INNER_SLOOP:
        DJNZ R3,$
   DJNZ R2, SDELAY_LOOP2
   DJNZ R1, SDELAY_LOOP1
   RET
```

```
// ~~~~~~~~~~~~~~~~~~ STRING DISPLAY ~~~~~~~~~~~~~~~
// (Total Unit, RED, BLE, INITIALIZING, SLEEP MODE)
LINE1:
        DB 'TOTAL_UNIT:',0

LINE2:
        DB 'BLE:',0

LINE3:
        DB 'RED:', 0

WAKE:
        DB 'INITIALIZING...', 0

SLEEP:
        DB 'SLEEP MODE', 0

END
```

The assembly code snippet above is structured to facilitate the interaction between a microcontroller and an LCD display, focusing on several key functionalities. The DISPLAY_NUMBER subroutine stands out as it handles the conversion of numerical values into ASCII characters suitable for display on the LCD. By dividing the number by 10 and converting the quotient and remainder into ASCII codes (ADD A, #30H), the subroutine prepares the digits for output via the SEND_DATA subroutine, which likely interfaces directly with the LCD hardware. This process enables the microcontroller to dynamically display numeric data such as sensor readings or operational counts, enhancing the device's capability to provide real-time feedback in embedded systems applications.

Additionally, the code includes essential delay routines: `MS_DELAY` and `S_DELAY`, crucial for timing operations within embedded systems. `MS_DELAY` provides a short delay loop, while `S_DELAY` offers a longer delay, both implemented through nested loops using the microcontroller's registers. These routines are vital for coordinating tasks requiring precise timing, such as synchronisation with external events or ensuring consistent operation across varying environmental conditions. The decision not to use a timer for generating delay routines is to avoid conflicting use of the same timer for tasks like generating PWM signals for servos while also handling delay operations simultaneously. This approach helps maintain separate and dedicated resources for specific tasks, ensuring reliable performance and avoiding potential conflicts in timing-sensitive applications.

4.2.2 VisionFive 2 Python Codes

I. Python Module Setup

```
import cv2
import numpy as np
import VisionFive.gpio as GPIO
import sys
import time
import logging
```

Code above imports the python modules for further usage and the function of each python module has been stated in the table below.

Python Modules	***Usage***
cv2 ***numpy***	*Used for OpenCV and handles image processing.*
VisionFive.gpio	*Used for setup the VisionFive 2 GPIO*
sys ***time***	*Used for basic system and time functions*
logging	*Used for debugging*

II. Arrays, Variables and GPIO Pin Setup

```
# ADC Pins
First Sensor = [7,11,13,15,27,29,31,35]
RD_PIN = 12
WR_PIN = 16

# Motor Driver PWM Pins
PWM_PIN = 22
threshold_pwm = 5
last_voltage_percent = 0

# Interface with 8051 (First Frame)
INT1    = 37  # Interrupt Output for 8051
BLE_BOX = 38  # Continue Capture Another Frame after falling down
RED_BOX = 40  # to boxes (active low)
TRIGGER = 0

# Arrays
Values_Array = [0,0,0,0,0,0,0,0]
First_Sensor_Status = False

# Output Pins
red_led  = 26
blue_led = 28

MIN_AREA = 1200
```

First of all, the code above configuring 8 input pins for the ADC0804 output in the "First Sensor" Array, and assigns GPIO pins 12 and 16 for the "RD" and "WR" functions respectively, crucial for voltage conversion within the ADC0804. Additionally, GPIO pins are allocated for the motor driver, interrupt handling, and control of specific boxes labelled "blue_box" and "red_box". Output values from the ADC0804 are stored in an array named "values_array". Furthermore, GPIO pins 26 and 28 are designated for output LEDs, specifically "red_led" and "blue_led". Lastly, a constant "MIN_AREA" is defined to establish a threshold area for subsequent image processing operations.

III. GPIO Input and Output Setup

```
def gpio_setup(Sensor_Array):
    GPIO.setmode(GPIO.BOARD)
    GPIO.setwarnings(False)

    GPIO.setup(red_led,GPIO.OUT)
    GPIO.setup(blue_led,GPIO.OUT)

    GPIO.setup(INT1,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    GPIO.setup(BLE_BOX,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    GPIO.setup(RED_BOX,GPIO.IN,pull_up_down=GPIO.PUD_UP)

    GPIO.setup(RD_PIN,GPIO.OUT)
    GPIO.setup(WR_PIN,GPIO.OUT)

    for pin in Sensor_Array:
        GPIO.setup(pin,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)
```

The function `gpio_setup` configures GPIO pins on a VisionFive 2 using the `VisionFive.gpio` library. It begins by setting the GPIO mode to board numbering and disables GPIO warnings to streamline operation. Output pins are then defined for controlling LEDs (`red_led` and `blue_led`). Several input pins (`INT1`, `BLE_BOX`, `RED_BOX`, `RD_PIN`, `WR_PIN`, and those in `Sensor_Array`) are configured with pull-up resistors (`GPIO.PUD_UP`), ensuring stable high states when no external signal is present. Each pin in `Sensor_Array` is set as an input with a pull-down resistor (`GPIO.PUD_DOWN`), facilitating reliable signal reading.

IV. GPIO Input and Output Setup

```
def gpio_detect(cond1,cond2):
  # Interrupt (Capture First Frame)
  global last_time, TRIGGER
  if(((cond1 == 1) or (cond2 == 1)) and ((TRIGGER==2) or (TRIGGER ==0))):
    GPIO.setup(INT1,GPIO.OUT)
    # GPIO.output(INT1,True)
    # time.sleep(0.001)
    GPIO.output(INT1,False)
    time.sleep(0.001)
    GPIO.output(INT1,True)
    TRIGGER = 1
    print("Interrupt Activated")
    last_time = time.time()

  # Simple Continuous Color Detection
  if cond1:
    GPIO.output(red_led,True)
    print("RED COLOR DETECT")
  else:
    GPIO.output(red_led,False)
  if cond2:
    GPIO.output(blue_led,True)
    print("BLUE COLOR DETECT")
  else:
    GPIO.output(blue_led,False)

      if(((GPIO.input(RED_BOX) == GPIO.LOW) or (GPIO.input(BLE_BOX) ==
GPIO.LOW)) and TRIGGER ==1):
    TRIGGER == 2
    print("Interrupt is deactivated")
```

The `gpio_detect` function is designed to manage GPIO-based operations and conditions for a VisionFive 2 development board. It starts by checking two conditions (`cond1` and `cond2`) along with the `TRIGGER` state to determine if an interrupt should be activated. If the conditions are met, it sets up the `INT1` pin as an output, toggles its state to capture the first frame, sets the `TRIGGER` to 1, and records the current time (create an interrupt output for the 8051 microcontroller). The function also performs continuous colour detection by turning on the `red_led` if `cond1` is true and the `blue_led` if `cond2` is true, providing immediate visual feedback for detected colours. Additionally, it checks the state of the `RED_BOX` and `BLE_BOX` pins to deactivate the interrupt by setting `TRIGGER` to 2 if either is in a low state (when the object is falling into the boxes and detected by IR sensors).

V. Setup Camera

```
# Camera Setup
def build_gst_pipeline(gst_pipeline):
    cap = cv2.VideoCapture(gst_pipeline,cv2.CAP_GSTREAMER)
    if not cap.isOpened():
        raise Exception("Failed to open video capture")
    return cap
```

The `build_gst_pipeline` function initialises a video capture object using OpenCV and GStreamer. It takes a GStreamer pipeline string (`gst_pipeline`) as an input. The function creates a `VideoCapture` object with the provided pipeline, specifying the use of GStreamer for capturing video streams. It then checks if the video capture object has been successfully opened. If the capture object fails to open, an exception is raised, indicating that the video capture process could not be initiated. If successful, the function returns the `VideoCapture` object, ready for further video processing operations.

VI. Duty Cycle Printing on the Screen

```
# DUTY CYCLE SETUP
def DUTYCYCLE_TEXT(frame, Sensor_Array, last_voltage_percent,pwm):
    voltage = 0

    # Capture data
    GPIO.output(WR_PIN,GPIO.LOW)
    GPIO.output(WR_PIN,GPIO.HIGH)
    GPIO.output(RD_PIN,GPIO.LOW)

    # Read ADC Values
    for i in range (len(Sensor_Array)):
        Output = GPIO.input(Sensor_Array[i])
        if Output == GPIO.HIGH:
            Values_Array[i] = 1
        else:
            Values_Array[i] = 0

    # Calculation
    for i in range(len(Values_Array)):
        voltage += Values_Array[i] * (2**i)
    volt_to_percent = (voltage/255)*100
                                        cv2.putText(frame,f"Duty          Cycle:
{volt_to_percent:.0f}%",(160,460),cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),3)

    # Change Duty Cycle
    if(abs(volt_to_percent - last_voltage_percent) > threshold_pwm):
        pwm.ChangeDutyRatio(int(volt_to_percent))
        last_voltage_percent = volt_to_percent
    return frame
```

The `DUTYCYCLE_TEXT` function reads analog data from an ADC using GPIO pins, calculates the voltage percentage, displays this information on a video frame, and adjusts the PWM duty cycle accordingly. It begins by initializing the voltage variable and setting the WR (Write) and RD (Read) pins to capture data from the ADC. The function then iterates over the `Sensor_Array` to read each sensor's value, storing the results in `Values_Array`. The voltage is calculated by summing the values in `Values_Array`, each weighted by a power of 2, and converting the result to a percentage (`volt_to_percent`) (Using binary values with increasing power of 2 to calculate the hexadecimal value and convert to percentage value). This percentage is displayed on the video frame using `cv2.putText`. Finally, if the new duty cycle percentage differs significantly from the previous value, the function updates the PWM duty cycle and returns the modified frame.

VII. Draw boxes on the detected colour objects

```
# Image Processing
def
detect_and_draw(frame,lower_bound,upper_bound,color_name,box_color,condition,min_are
a=MIN_AREA):
   hsv_frame=cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
   mask=cv2.inRange(hsv_frame,lower_bound,upper_bound)
                                                          contours,_                =
cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

   for contour in contours:
      area = cv2.contourArea(contour)
      if area > min_area:
         x,y,w,h = cv2.boundingRect(contour)
         cv2.rectangle(frame,(x,y),(x+w,y+h),box_color,2)

         text_offset_x = 10
         text_offset_y = 10

         box_coords = ((x,y+text_offset_y),(x+50,y+text_offset_y-30))

         cv2.rectangle(frame,box_coords[0],box_coords[1],box_color,-1)

cv2.putText(frame,color_name,(x,y+text_offset_y-10),cv2.FONT_HERSHEY_SIMPLEX,0.
5,(255,255,255),1)
         condition = True

      return frame,condition
```

The `detect_and_draw` function processes an image frame to detect objects of a specific colour and draw bounding boxes around them. It converts the frame from BGR to HSV colour space, creates a mask based on the specified colour bounds (`lower_bound` and `upper_bound`), and finds contours within this mask. For each contour, if its area exceeds the minimum area (`min_area`), the function draws a bounding box around the detected object and overlays a coloured rectangle with the object's colour name. It also updates a `condition` flag to `True` if any object is detected. The function returns the modified frame with the drawn boxes and the updated `condition` flag.

VIII. Read each frames and shows the result (box and colour detected) on the screen

```
def read_and_display_frames(cap,pwm):
    lower_red = np.array([0,120,70])
    upper_red = np.array([10,255,255])

    lower_blue = np.array([100,150,0])
    upper_red = np.array([140,255,255])

    display_flag = False

    while True:
        ret,frame = cap.read()
        if not ret:
            print("Failed to read frames")
            break

        key = cv2.waitKey(1) & 0xFF
        if key == ord(' '):
            display_flag = not display_flag

        if display_flag:
            RED_D = False
            BLUE_D = False
                                                                    frame,      RED_D      =
detect_and_draw(frame,lower_red,upper_red,"RED",(0,0,255),RED_D)
              frame, BLUE_D = detect_and_draw(frame, lower_blue, upper_blue, "BLUE", (255,
0, 0, BLUE_D)
            frame = DUTYCYCLE_TEXT(frame,First_Sensor,last_voltage_percent,pwm)
            gpio_detect(RED_D,BLUE_D)
            cv2.imshow("Color Detection",frame)
            print(f"{TRIGGER}")
        else:
            blank_frame = np.zeros_like(frame)
            cv2.imshow("Color Detection",blank_frame)
            pwm.ChangeDutyRatio(0)

        if key == ord('q'):
            break
```

The read_and_display_frames function captures video frames from a video stream (cap), processes the frames for red and blue colour detection, displays the results, and adjusts the PWM duty cycle based on detected colours. The function defines colour ranges for red and blue in HSV colour space and initialises a display_flag to control display toggling. In a continuous loop, it reads frames from the video stream and checks for user input to toggle the display or quit. If the display is active, it detects red and blue colours in the frame using the detect_and_draw function, updates the frame with the detected colours, and displays the duty cycle percentage using DUTYCYCLE_TEXT. The gpio_detect function is called to handle GPIO-related tasks based on the detection results. The processed frame is then shown in a window. If the display is inactive, a blank frame is shown, and the PWM duty cycle is set to 0. The loop exits when the 'q' key is pressed.

IX. Integrate all functions into Main Program

```
def main():
    gst_pipeline = build_gst_pipeline()
    cap = None
    try:
        global threshold_pwm
        gpio_setup(PWM_PIN,GPIO.OUT)
        GPIO.setup(PWM_PIN,GPIO.HIGH)
        GPIO.output(PWM_PIN,GPIO.HIGH)
        pwm.start(0)
        cap = open_video_capture(gst_pipeline)
        cap.set(cv2.CAP_PROP_FRAME_WIDTH,640)
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT,480)
        read_and_display_frames(cap,pwm)
    except Exception as e:
        print(e)
    finally:
        if cap:
            cap.release()
        pwm.stop()
        GPIO.remove_event_detect(BLE_BOX)
        GPIO.remove_event_detect(RED_BOX)
        cv2.destroyAllWindows()
        GPIO.cleanup()

if __name__ == "__main__":
    main()
```

The `main` function initialises the video capture and PWM setup for a machine vision application. It starts by building a GStreamer pipeline with `build_gst_pipeline` and declares a `cap` variable for video capture. Within a try block, it sets the `threshold_pwm` and configures the GPIO pin for PWM output, starting the PWM with a 0% duty cycle. The function then opens the video capture device using `open_video_capture`, sets the frame dimensions to 640x480 pixels, and calls `read_and_display_frames` to process and display the frames while adjusting the PWM based on detected colours. If any exceptions occur, they are caught and printed. Finally, in the `finally` block, it releases the video capture, stops the PWM, removes GPIO event detections for specific pins, closes all OpenCV windows, and cleans up the GPIO settings.

5.0 Conclusion And Future Work

5.1 Conclusion

In conclusion, an automated colour filter production line is successfully implemented in our project. There are several features that are successfully implemented in our project such as colour detection by using OpenCv in VisionFive, VisionFive board interfaces with ADC0804 to achieve adjustable duty cycle for controlling the speed of the motor in driving the conveyor belt, 4 bit LCD mode by using 8051 microcontroller, different PWM output in controlling servo motor by using 8051 microcontroller and interfacing of 8051 microcontroller and IR sensors as inputs to the system.

By going through the mini project, deeper understanding in interfacing with different sensors and components are provided. Hands-on experience is also being strengthened as the demo session required students to show a completely working project.

Lastly, our project increases the efficiency in the colour filter production line and reduces the labour cost for the process as the project is an automated colour filter production line which requires no one to operate. In addition, the LCD display screen summarises and shows the result of the quantities of different colours and total units to the user as a summary.

5.2 Future Work

To make the project more advanced and smarter, there are some future improvements which are suggested to improvise the project.

1. ***Enhance Sensor Technology***
   Sensors with more sensitivity and accuracy are used instead of cheap sensors. The current IR sensor has low range for object detection. So, it is suggested to change the IR sensor to proximity sensor for object detection.

2. ***Improve User Interface or Control System***
   Current User Interfaces are just LCD displays and one potentiometer to adjust the speed of the conveyor belt. It is better if there is a control panel or dashboard that can show the data and control the speed of the motor directly by touching buttons on the screen.

3. ***Integration of Advanced Machine Learning Algorithms***
   Implement machine learning techniques to enhance colour recognition accuracy. Use algorithms for adaptive learning to handle new colour variations and defects.

6.0 Reference

1. I. The MathWorks, Color Detection - MATLAB & Simulink - MathWorks Deutschland (n.d.). https://de.mathworks.com/help/supportpkg/android/ref/color-detection.html

2. Joy, D.T., Kaur, G., Chugh, A., Bajaj, S.B.: Computer vision for color detection. Int. J. Innov. Res. Comput. Sci. Technol. 9, 53–59 (2021). https://doi.org/10.21276/ijircst.2021.9.3.9

3. Amin, U., Ahmad, G., Liaqat, N., Ahmed, M., Zahoor, S.: Detection & distinction of colors using color sorting robotic arm in a pick & place mechanism. Int. J. Sci. Res. 3, 1164–1168 (2012). https://www.ijsr.net/archive/v3i6/MDIwMTMxODgy.pdf

4. Dewi, T., Risma, P., Oktarina, Y.: Fruit sorting robot based on color and size for an agricultural product packaging system. Bull. Electr. Eng. Inform. 9, 1438–1445 (2020). https://doi.org/10.11591/eei.v9i4.2353

5. Kieselbach, K.K., Nöthen, M., Heuer, H.: Development of a visual inspection system and the corresponding algorithm for the detection and subsequent classification of paint defects on car bodies in the automotive industry. J. Coat. Technol. Res. 16(4), 1033–1042 (2019). https://doi.org/10.1007/s11998-018-00178-y

6. Mohamad, F.S., Manaf, A.A., Chuprat, S.: Histogram matching for color detection: a preliminary study. In: Proceedings of the 2010 International Symposium on Information Technology - System Development and Application and Knowledge Society, ITSim 2010, vol. 3, pp. 1679–1684 (2010). https://doi.org/10.1109/ITSIM.2010.5561637

7. Li, Q., Lu, W., Yang, J.: A hybrid thresholding algorithm for cloud detection on ground-based color images. J. Atmos. Ocean. Technol. 28, 1286–1296 (2011). https://doi.org/10.1175/JTECH-D-11-00009.1

8. Castillo-Martínez, M., Gallegos-Funes, F.J., Carvajal-Gámez, B.E., Urriolagoitia-Sosa, G., Rosales-Silva, A.J.: Color index based thresholding method for background and foreground segmentation of plant images. Comput. Electron. Agric. 178, 105783 (2020). https://doi.org/10.1016/J.COMPAG.2020.105783

9. Basar, S., Ali, M., Id, G.O., Id, M.Z.: Unsupervised color image segmentation: a case of RGB histogram based K-means clustering initialization. PLoS ONE 15, 1–21 (2020). https://doi.org/10.1371/journal.pone.0240015

10. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You Only Look Once: Unified, Real-Time Object Detection (n.d.). http://pjreddie.com/yolo/

11. Medojevic, I., Veg, E., Joksimovic, A., Ilic, J.: Promotion of color sorting in industrial systems using a deep learning algorithm. Appl. Sci. 12, 12817 (2022). https://doi.org/10.3390/app122412817

12. Jiang, P., Ergu, D., Liu, F., Cai, Y., Ma, B.: A review of Yolo algorithm developments. Proc. Comput. Sci. 199, 1066–1073 (2021). https://doi.org/10.1016/j.procs.2022.01.135

13. Ultralytics, YOLOv8 – Ultralytics—Revolutionizing the World of Vision AI (n.d.). https://ultralytics.com/yolov8

14. Hanmandlu, M., Arora, S., Gupta, G., Singh, L.: Underexposed and overexposed colour image enhancement using information set theory. Imaging Sci. J. 64, 1–13 (2016). https://doi.org/10.1080/13682199.2016.1215063

15. iiM AG measurement + engineering, Useful Facts About Machine Vision Lighting Systems (2022). https://www.iim-ag.com/fileadmin/user_upload/lumimax/en/useful-facts/pdf/LUMIMAX_knowledge_base_website.pdf

16. Navada, B.R., Santhosh, K.V., Prajwal, S., Shetty, H.B.: An image processing technique for color detection and distinguish patterns with similar color: an aid for color blind people. In: Proceedings of the International Conference on Circuits, Communication, Control and Computing, I4C 2014, pp. 333–336 (2014). https://doi.org/10.1109/CIMCA.2014.7057818

17. O'Mahony, N., et al.: Deep learning vs. traditional computer vision. In: Arai, K., Kapoor, S. (eds.) CVC 2019. AISC, vol. 943, pp. 128–144. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-17795-9_10

18. Zhang, X., Qiu, Z., Huang, P., Hu, J., Luo, J.: Application research of YOLO v2 combined with color identification. In: Proceedings of the 2018 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2018, pp. 138–141 (2019). https://doi.org/10.1109/CyberC.2018.00036

7.0 Appendix

Appendix A: Assembly Codes for 8051 Microcontroller

```
ORG 0000H
LJMP INITIAL

; ---- ISR0 PROGRAM ----
ORG  0003H                          ; INT0 Memory Location
ACALL WAKEUP
RETI

; ---- ISR1 PROGRAM ----
ORG 0013H                           ; INT1 Memory Location
LOOP5:
        CLR P1.4
        LCALL DELAY
        JB RED,  RM1 ;Red colour detected
        JB BLUE, BM1; Blue colour detected
        SETB P1.4
        LCALL DELAY
        SJMP LOOP5

RM1:
        LCALL RM
        RETI

BM1:
        LCALL BM
        RETI

ORG 0100H
; ---- Pin Initialization and Configurations ----
; Define SFRs and bit variables
//SLP                   EQU P1.0      ; SLEEP PIN
RS              EQU P2.0
RW              EQU P2.1
EN              EQU P2.2
WK              EQU P2.3      ; IDLE AND WAKEUP
REDSERVO   EQU P1.1
BLUESERVO          EQU P1.2
REDBOX     EQU P0.7
```

```
IRRED       EQU P0.4
BLUEBOX     EQU P0.3
IRBLUE      EQU P0.2
BLUE        EQU P0.1
RED         EQU P0.0


; Register to count number of red objects and blue objects
RED_NUM EQU 00H
BLE_NUM EQU R5

; ---- VARIABLES INITIALIZATION ----
INITIAL:
        MOV RED_NUM, #0x00  ; Initialize red count to 0
        MOV BLE_NUM, #0x00  ; Initialize yellow count to 0
        MOV TMOD, #01H ; Set Timer 0 to 16 bit mode

        ; Clear P1 and P2
        MOV P1, #0FFH
        MOV P2, #00FH

; ---- MAIN PROGRAM ----
START:
   MOV IE,   #85H               ; Enable global interrupt, INT0, INT1
        ; Select INT0 and INT1 on falling edge
        MOV TCON, #05H

   ; Initialization of LCD Module
   MOV A, #02H
   ACALL SEND_COMMAND
   ACALL MS_DELAY
   MOV A, #28H
   ACALL SEND_COMMAND
   ACALL MS_DELAY
   MOV A, #0CH         ; Display ON, Cursor OFF
   ACALL SEND_COMMAND
   ACALL MS_DELAY
   MOV A, #01H
   ACALL SEND_COMMAND
   ACALL MS_DELAY
   MOV A, #80H         ; Force cursor to beginning of first line
```

```
   ACALL SEND_COMMAND
   ACALL MS_DELAY
        SJMP MAIN

POWER:
        LJMP SLEEPMODE

MAIN:
        CPL P1.3
        ; Entering IDLE Mode or not
        JNB P1.0, POWER

   MOV A, #80H         ; Continuously Refresh without blinking
   ACALL SEND_COMMAND
   ACALL MS_DELAY

   ; First Line of Words
   MOV DPTR, #LINE1
WRITE_FIRST:
        CLR A
   MOVC A, @A+DPTR
        JZ TOTALUNIT
        INC DPTR
        ACALL SEND_DATA
        SJMP WRITE_FIRST

TOTALUNIT:
   ; Write Total Unit
   MOV A, RED_NUM
   ADD A, BLE_NUM
   ACALL DISPLAY_NUMBER

   ; Second Line
   MOV A, #0C0H
   ACALL SEND_COMMAND
   ACALL MS_DELAY

   ; Second Line of Words
   MOV DPTR, #LINE2
WRITE_SECOND:
        CLR A
```

```
        MOVC A, @A+DPTR
        JZ BLUEUNIT
        INC DPTR
        ACALL SEND_DATA
        SJMP WRITE_SECOND

BLUEUNIT:
   ; Write Blue Number Units
   MOV A, BLE_NUM
   ACALL DISPLAY_NUMBER

   ; Spacebar
   MOV A, #20H         ; ASCII code for space
   ACALL SEND_DATA

   ; Third Line of Words
   MOV DPTR, #LINE3
WRITE_THIRD:
        CLR A
        MOVC A, @A+DPTR
        JZ REDUNIT
        INC DPTR
        ACALL SEND_DATA
        SJMP WRITE_THIRD

REDUNIT:
   ; Write Red Number Units
   MOV A, RED_NUM
   ACALL DISPLAY_NUMBER

   ACALL MS_DELAY
   LJMP MAIN

// ~~~~~~~~~~~~~~ END OF PROGRAM ~~~~~~~~~~~~~~~
// ===========================================

// ~~~~~~~~~~~~~~ Servo Mechanism ~~~~~~~~~~~~~~~
;Red mechanism ready
RM:
        JB IRRED, $ ; Wait for IR to detect
        INC RED_NUM
```

```
        BACK: JB REDBOX, RSERVO_90 ; Servo turn to 90 degree until red box read high
        ACALL RSERVO_0 ; turn back to 0 degree
        SETB P1.4
        RET

;Blue mechanism ready
BM:
        JB IRBLUE, $ ; Wait for IR to detect
        INC BLE_NUM
        BACK2:JB BLUEBOX, BSERVO_90 ; Servo turn to 90 degree until blue box read
high
        ACALL BSERVO_0 ; turn back to 0 degree
        SETB P1.4
        RET

;Set up for servo turn to 90 degree
RSERVO_90:
        SETB REDSERVO
        ACALL DELAY1_4ms
        CLR REDSERVO
        ACALL DELAY18_6ms
        SJMP BACK

BSERVO_90:
        SETB BLUESERVO
        ACALL DELAY1_4ms
        CLR BLUESERVO
        ACALL DELAY18_6ms
        SJMP BACK2

;Set up for servo turn to 0 degree
RSERVO_0:
        MOV R1, #20D ; repeat for 20 times to send 20 pulse
        Loop:
                SETB REDSERVO
                ACALL DELAY1ms
                CLR REDSERVO
                ACALL DELAY19ms
                DJNZ R1, Loop
                RET
```

```
BSERVO_0:
        MOV R1, #20D ; repeat for 20 times to send 20 pulse
        Loop3:
                SETB BLUESERVO
                ACALL DELAY1ms
                CLR BLUESERVO
                ACALL DELAY19ms
                DJNZ R1, Loop3
                RET

;Set up for 1ms delay
DELAY1ms:
        MOV TH0, #0FCH
        MOV TL0, #67H
        SETB TR0 ; Start Timer0
        JNB TF0, $ ;Wait until Timer0 finish counting
        CLR TR0 ; Clear Timer0
        CLR TF0 ; Clear Timer Flag 0
        RET

;Set up for 1.4ms delay
DELAY1_4ms:
        MOV TH0, #0F8H
        MOV TL0, #0ADH
        SETB TR0 ; Start Timer0
        JNB TF0, $ ;Wait until Timer0 finish counting
        CLR TR0 ; Clear Timer0
        CLR TF0 ; Clear Timer Flag 0
        RET

;Set up for 19ms delay
DELAY19ms:
        MOV TH0, #0BBH
        MOV TL0, #99H
        SETB TR0 ; Start Timer0
        JNB TF0, $ ;Wait until Timer0 finish counting
        CLR TR0 ; Clear Timer0
        CLR TF0 ; Clear Timer Flag 0
        RET

;Set up for 18.6ms delay
```

```
DELAY18_6ms:
        MOV TH0, #0B3H
        MOV TL0, #0A6H
        SETB TR0 ; Start Timer0
        JNB TF0, $ ;Wait until Timer0 finish counting
        CLR TR0 ; Clear Timer0
        CLR TF0 ; Clear Timer Flag 0
        RET

DELAY:
        MOV R2,#3D
LOOP2:MOV R3, #255D
LOOP1:MOV R4, #255D
        DJNZ R4, $
        DJNZ R3, LOOP1
        DJNZ R2, LOOP2
        RET

; ~~~~~~~~~~~~~~~~~~~~~~~~~~ POWER MODE ~~~~~~~~~~~~~~~~~~~~~~~~~~
; SLEEPMODE, WAKEUP
SLEEPMODE:
        ; ---- CLEAR DISPLAY SCREEN ----
        MOV A, #01H
   ACALL SEND_COMMAND
   ACALL MS_DELAY
   MOV A, #80H         ; Force cursor to beginning of first line
   ACALL SEND_COMMAND
   ACALL MS_DELAY
        ; ---- FINISIH CLEARING ----
        ; ---- SLEEP WORDS ----
   MOV DPTR, #SLEEP
        WRITE_SLEEP:
        CLR A
        MOVC A, @A+DPTR
        INC DPTR
        ACALL SEND_DATA
        JZ ENTERSLP
        SJMP WRITE_SLEEP
        ACALL S_DELAY
        ENTERSLP:
        ; ----
```

```
        MOV A, #0FFH
        ; ---- ENTER SLEEP MODE ----
        ORL 87H, #01H        ; PCON (IDLE MODE)

JUMP:
        LJMP MAIN

WAKEUP:
        ; ---- MISPRESS ----
        CJNE A, #0FFH, JUMP
        ; ---- WAKING UP FROM SLEEP MODE ----
        ANL 87H, #00H
        ; ---- CLEAR DISPLAY SCREEN ----
        MOV A, #01H
   ACALL SEND_COMMAND
   MOV A, #80H         ; Force cursor to beginning of first line
   ACALL SEND_COMMAND
        ; ---- FINISIH CLEARING ----
        ; ---- WAKEUP WORDS ----
   MOV DPTR, #WAKE
        WRITE_WAKE:
        CLR A
        MOVC A, @A+DPTR
        INC DPTR
        JZ JUMPBACK
        ACALL SEND_DATA
        SJMP WRITE_WAKE
        JUMPBACK:
        ACALL S_DELAY
        MOV A, #01H
   ACALL SEND_COMMAND
        RET




; ~~~~~~~~~~~~~~~~~~~~~~~~~~ LCD MODULE ~~~~~~~~~~~~~~~~~~~~~~~~~~
; SEND_COMMAND, SEND_DATA
SEND_COMMAND:
   MOV R6, A           ; Save original value of A
   ANL A, #0F0H        ; Mask lower nibble
```

```
    ANL P2, #0x0F       ; Clear upper nibble bits
    ORL P2, A           ; Send upper nibble to LCD
    CLR RS              ; Command mode
    CLR RW              ; Write mode
    SETB EN             ; Enable pulse
    ACALL MS_DELAY      ; Delay
    CLR EN              ; Disable pulse

    MOV A, R6           ; Restore original value of A
    SWAP A              ; Swap nibbles to send lower nibble
    ANL A, #0F0H        ; Mask lower nibble
    ANL P2, #0x0F       ; Clear upper nibble bits
    ORL P2, A           ; Send lower nibble to LCD
    CLR RS              ; Command mode
    CLR RW              ; Write mode
    SETB EN             ; Enable pulse
    ACALL MS_DELAY      ; Delay
    CLR EN              ; Disable pulse
    RET

SEND_DATA:
    MOV R6, A           ; Save original value of A
    ANL A, #0F0H        ; Mask lower nibble
    ANL P2, #0x0F       ; Clear upper nibble bits
    ORL P2, A           ; Send upper nibble to LCD
    SETB RS             ; Data mode
    CLR RW              ; Write mode
    SETB EN             ; Enable pulse
    ACALL MS_DELAY      ; Delay
    CLR EN              ; Disable pulse

    MOV A, R6           ; Restore original value of A
    SWAP A              ; Swap nibbles to send lower nibble
    ANL A, #0F0H        ; Mask lower nibble
    ANL P2, #0x0F       ; Clear upper nibble bits
    ORL P2, A           ; Send lower nibble to LCD
    SETB RS             ; Data mode
    CLR RW              ; Write mode
    SETB EN             ; Enable pulse
    ACALL MS_DELAY      ; Delay
    CLR EN              ; Disable pulse
```

```
   RET

// ~~~~~~~~~ ASCII NUMBER to LCD ~~~~~~~~~~~~~
DISPLAY_NUMBER:
   MOV B, #10       ; Divide by 10
   DIV AB           ; A = quotient, B = remainder
   ADD A, #30H      ; Convert to ASCII
   ACALL SEND_DATA  ; Send tens place
   MOV A, B         ; Get remainder
   ADD A, #30H      ; Convert to ASCII
   ACALL SEND_DATA  ; Send units place
   RET

// ~~~~~ DELAY SUBROUTINES (MS_DELAY & S_DELAY) ~~~~~~~~~~
MS_DELAY:
   MOV R1, #100
DELAY_LOOP:
   MOV R2, #255
INNER_DELAY_LOOP:
   DJNZ R2, INNER_DELAY_LOOP
   DJNZ R1, DELAY_LOOP
   RET

S_DELAY:
   MOV R1, #7
SDELAY_LOOP1: MOV R2, #255
SDELAY_LOOP2: MOV R3, #255
INNER_SLOOP:
        DJNZ R3,$
   DJNZ R2, SDELAY_LOOP2
   DJNZ R1, SDELAY_LOOP1
   RET

// ~~~~~~~~~~~~~~~~~~ STRING DISPLAY ~~~~~~~~~~~~~~~
// (Total Unit, RED, BLE, INITIALIZING, SLEEP MODE)
LINE1:
        DB 'TOTAL_UNIT:',0

LINE2:
        DB 'BLE:',0
```

```
LINE3:
        DB 'RED:', 0

WAKE:
        DB 'INITIALIZING...', 0

SLEEP:
        DB 'SLEEP MODE', 0

END
```

Appendix B: Python Codes for VisionFive 2 Development Board

```
import cv2
import numpy as np
import VisionFive.gpio as GPIO
import sys
import time
import logging

# ADC Pins
First Sensor = [7,11,13,15,27,29,31,35]
RD_PIN = 12
WR_PIN = 16

# Motor Driver PWM Pins
PWM_PIN = 22
threshold_pwm = 5
last_voltage_percent = 0

# Interface with 8051 (First Frame)
INT1    = 37  # Interrupt Output for 8051
BLE_BOX = 38  # Continue Capture Another Frame after falling down
RED_BOX = 40  # to boxes (active low)
TRIGGER = 0

# Arrays
Values_Array = [0,0,0,0,0,0,0,0]
First_Sensor_Status = False

# Output Pins
red_led  = 26
blue_led = 28

MIN_AREA = 1200

# GPIO Setup and Condition Meet
def gpio_setup(Sensor_Array):
    GPIO.setmode(GPIO.BOARD)
    GPIO.setwarnings(False)

    GPIO.setup(red_led,GPIO.OUT)
    GPIO.setup(blue_led,GPIO.OUT)
```

```
    GPIO.setup(INT1,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    GPIO.setup(BLE_BOX,GPIO.IN,pull_up_down=GPIO.PUD_UP)
    GPIO.setup(RED_BOX,GPIO.IN,pull_up_down=GPIO.PUD_UP)

    GPIO.setup(RD_PIN,GPIO.OUT)
    GPIO.setup(WR_PIN,GPIO.OUT)

    for pin in Sensor_Array:
        GPIO.setup(pin,GPIO.IN,pull_up_down=GPIO.PUD_DOWN)

def gpio_detect(cond1,cond2):
    # Interrupt (Capture First Frame)
    global last_time, TRIGGER
    if(((cond1 == 1) or (cond2 == 1)) and ((TRIGGER==2) or (TRIGGER ==0))):
        GPIO.setup(INT1,GPIO.OUT)
        # GPIO.output(INT1,True)
        # time.sleep(0.001)
        GPIO.output(INT1,False)
        time.sleep(0.001)
        GPIO.output(INT1,True)
        TRIGGER = 1
        print("Interrupt Activated")
        last_time = time.time()

    # Simple Continuous Color Detection
    if cond1:
        GPIO.output(red_led,True)
        print("RED COLOR DETECT")
    else:
        GPIO.output(red_led,False)
    if cond2:
        GPIO.output(blue_led,True)
        print("BLUE COLOR DETECT")
    else:
        GPIO.output(blue_led,False)

            if(((GPIO.input(RED_BOX) == GPIO.LOW) or (GPIO.input(BLE_BOX) ==
GPIO.LOW)) and TRIGGER ==1):
        TRIGGER == 2
        print("Interrupt is deactivated")
```

```
# Camera Setup
def build_gst_pipeline(gst_pipeline):
    cap = cv2.VideoCapture(gst_pipeline,cv2.CAP_GSTREAMER)
    if not cap.isOpened():
        raise Exception("Failed to open video capture")
    return cap

# DUTY CYCLE SETUP
def DUTYCYCLE_TEXT(frame, Sensor_Array, last_voltage_percent,pwm):
    voltage = 0

    # Capture data
    GPIO.output(WR_PIN,GPIO.LOW)
    GPIO.output(WR_PIN,GPIO.HIGH)
    GPIO.output(RD_PIN,GPIO.LOW)

    # Read ADC Values
    for i in range (len(Sensor_Array)):
        Output = GPIO.input(Sensor_Array[i])
        if Output == GPIO.HIGH:
            Values_Array[i] = 1
        else:
            Values_Array[i] = 0

    # Calculation
    for i in range(len(Values_Array)):
        voltage += Values_Array[i] * (2**i)
    volt_to_percent = (voltage/255)*100
                                                cv2.putText(frame,f"Duty                Cycle:
{volt_to_percent:.0f}%",(160,460),cv2.FONT_HERSHEY_SIMPLEX,1,(0,255,0),3)

    # Change Duty Cycle
    if(abs(volt_to_percent - last_voltage_percent) > threshold_pwm):
        pwm.ChangeDutyRatio(int(volt_to_percent))
        last_voltage_percent = volt_to_percent
    return frame

# Image Processing
```

```
def
detect_and_draw(frame,lower_bound,upper_bound,color_name,box_color,condition,min_are
a=MIN_AREA):
   hsv_frame=cv2.cvtColor(frame,cv2.COLOR_BGR2HSV)
   mask=cv2.inRange(hsv_frame,lower_bound,upper_bound)
                                                                    contours,_          =
cv2.findContours(mask,cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)

   for contour in contours:
      area = cv2.contourArea(contour)
      if area > min_area:
         x,y,w,h = cv2.boundingRect(contour)
         cv2.rectangle(frame,(x,y),(x+w,y+h),box_color,2)

         text_offset_x = 10
         text_offset_y = 10

         box_coords = ((x,y+text_offset_y),(x+50,y+text_offset_y-30))

         cv2.rectangle(frame,box_coords[0],box_coords[1],box_color,-1)

cv2.putText(frame,color_name,(x,y+text_offset_y-10),cv2.FONT_HERSHEY_SIMPLEX,0.
5,(255,255,255),1)
         condition = True

      return frame,condition

def read_and_display_frames(cap,pwm):
   lower_red = np.array([0,120,70])
   upper_red = np.array([10,255,255])

   lower_blue = np.array([100,150,0])
   upper_red = np.array([140,255,255])

   display_flag = False

   while True:
      ret,frame = cap.read()
      if not ret:
         print("Failed to read frames")
         break
```

```
        key = cv2.waitKey(1) & 0xFF
        if key == ord(' '):
            display_flag = not display_flag

        if display_flag:
            RED_D = False
            BLUE_D = False
                                                                frame,  RED_D  =
detect_and_draw(frame,lower_red,upper_red,"RED",(0,0,255),RED_D)
              frame, BLUE_D = detect_and_draw(frame, lower_blue, upper_blue, "BLUE", (255,
0, 0, BLUE_D)
            frame = DUTYCYCLE_TEXT(frame,First_Sensor,last_voltage_percent,pwm)
            gpio_detect(RED_D,BLUE_D)
            cv2.imshow("Color Detection",frame)
            print(f"{TRIGGER}")
        else:
            blank_frame = np.zeros_like(frame)
            cv2.imshow("Color Detection",blank_frame)
            pwm.ChangeDutyRatio(0)

        if key == ord('q'):
            break

def main():
    gst_pipeline = build_gst_pipeline()
    cap = None
    try:
        global threshold_pwm
        gpio_setup(PWM_PIN,GPIO.OUT)
        GPIO.setup(PWM_PIN,GPIO.HIGH)
        GPIO.output(PWM_PIN,GPIO.HIGH)
        pwm.start(0)
        cap = open_video_capture(gst_pipeline)
        cap.set(cv2.CAP_PROP_FRAME_WIDTH,640)
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT,480)
        read_and_display_frames(cap,pwm)
    except Exception as e:
        print(e)
    finally:
        if cap:
```

```
            cap.release()
        pwm.stop()
        GPIO.remove_event_detect(BLE_BOX)
        GPIO.remove_event_detect(RED_BOX)
        cv2.destroyAllWindows()
        GPIO.cleanup()

if __name__ == "__main__":
    main()
```