Design and Implementation of an Automated Sorting System Using Mechatronics Principle

Abstract:

This project presents the design and implementation of an automated sorting system that classifies objects based on their color using mechatronics principles. A Raspberry Pi acts as the main controller, integrated with a web camera for image processing, an IR sensor for object detection, and servo/stepper motors for actuation. When an object is placed on the conveyor belt, the IR sensor stops the belt, allowing the camera to capture and analyze the object's color using OpenCV. After processing, the belt resumes, and the object is sorted into its designated bin via a servo motor. The system offers a low-cost, efficient solution for color-based sorting and can be extended to include parameters like shape, size, or weight for broader industrial applications.

1.Introduction

In modern manufacturing and logistics, automation plays a vital role in improving efficiency, reducing human error, and enhancing accuracy in sorting processes. This project presents the Design and Implementation of an Automated Sorting System based on mechatronics principles, aiming to sort objects by color using a combination of hardware and software components.

The system uses a Raspberry Pi as the controller, integrated with a web camera, IR sensor, servo/stepper motors, and a 5V DC motor to drive a conveyor belt. When a user places an object on the moving belt, the IR sensor detects its presence and halts the conveyor. The web camera captures the image, and OpenCV processes it to detect the color. After a 3-second delay, the belt resumes, and the Raspberry Pi activates the respective servo motor to push the object into the correct bin based on its color (e.g., red, green, or blue).

This prototype aligns with Problem Statement 5, focusing on real-time, automated sorting using color detection. It demonstrates a practical application of IoT, image processing, and mechatronics integration, and can be enhanced with features like shape detection, real-time monitoring, and scalability for industrial use in the future.

2. System Overview

This project, *Automatic Sorting System using Mechatronics*, integrates hardware and software components to detect objects based on color and sort them using stepper motors and servo mechanisms. The system utilizes a Raspberry Pi for control, IR sensors for object detection, a camera for color recognition, and motors for mechanical movement.

2.1 Libraries and Modules Used

To ensure smooth interaction between hardware components and perform image processing, the following Python libraries and modules have been used:

1. **RPi.GPIO**
   Description: Enables GPIO pin control on the Raspberry Pi.
   Role: Used for motor control, reading IR sensor input, and servo positioning.

2. **time**
   Description: Provides time-related functions.
   Role: Used for delays between motor steps, servo movements, and system timing.

3. **threading**
   Description: Allows creation and management of threads.
   Role: Enables concurrent operation of motors, sensors, and web interface.

4. **cv2 (OpenCV)**
   Description: Computer vision library for image processing.
   Role: Handles color detection from camera input using HSV color space.

5. **numpy**
   Description: Fundamental package for scientific computing.
   Role: Used for array operations in image processing and color detection.

6. **os**
   Description: Provides operating system interface functions.
   Role: Sets environment variables for headless OpenCV operation.

7. **web_interface (custom)**
   Description: Shared module for web interface communication.
   Role: Provides system_data dictionary and socketio for real-time updates.

2.2 Sensors and Actuators Used

The project integrates multiple electronic components that serve specific roles in the detection, classification, and mechanical sorting of objects. The table below explains the function of each component, the reason it was chosen, and a brief comparison with alternatives.

1. Raspberry Pi (Main Controller)

- **Function:** Acts as the brain of the system – runs OpenCV for image processing, controls servos and stepper motors based on color detection.
- **Why Raspberry Pi?**
  - Lightweight, affordable single-board computer
  - Supports Python & OpenCV
  - Can handle camera input and GPIO outputs

Component	Pros	Cons
Raspberry Pi	Full Linux OS, OpenCV support, GPIOs	Slower than PC, limited USB ports
Arduino	Real-time control, cheaper	No camera/OpenCV support
Jetson Nano	Faster GPU for AI tasks	Higher cost, more power consumption

2. IR Sensor

- **Function:** Detects the presence of an object on the conveyor belt.
- **Why IR Sensor?**
  - Simple object detection
  - Low power, cost-effective

Sensor Type	Pros	Cons
IR Sensor	Cheap, easy to implement	Affected by ambient light, limited range

Ultrasonic	Better range and accuracy	More expensive, may need calibration
LIDAR	Highly accurate	Expensive, overkill for basic detection

3. Camera (with OpenCV)

- **Function:** Captures object image for color detection.
- **Why Camera with OpenCV?**
    - Enables real-time image analysis
    - Detects RGB values for classification

Camera Type	Pros	Cons
Pi Camera	Plug-and-play with Raspberry Pi	Lower resolution than some USB webcams
USB Webcam	Higher resolution options	May require extra drivers or more power
Industrial Camera	High-speed image capture	Expensive

4. Servo Motors (for Sorting Mechanism)

- **Function:** Moves the item into the correct bin based on color (blue, green, red).
- **Why Servo Motors?**
    - Precise angle control
    - Easy to interface with Raspberry Pi

Motor Type	Pros	Cons
Servo Motor	Precise positioning	Limited torque, angle limited to ~180°
DC + Encoder	High torque, adjustable	Complex control, needs H-bridge & feedback
Solenoid	Fast action, simple	Only linear motion, no precise control

5. Stepper Motors (for Conveyor Belt)

- **Function:** Moves the belt forward at fixed intervals to align items for sorting.
- **Why Stepper Motors?**
    - Precise rotation control
    - No feedback required for fixed-step operations

Motor Type	Pros	Cons
Stepper Motor	Precise control, no encoder needed	Can lose steps under load
DC Motor	Simple, inexpensive	Needs encoder for position control
BLDC Motor	Efficient, high-speed	More complex control

3. Methodology

3.1 Technical System Flow & Data Processing Overview

The system integrates Raspberry Pi GPIO control, stepper/servo motor actuation, infrared (IR) sensing, and OpenCV-based computer vision to automate color-based object sorting. The workflow begins with two stepper motors running in opposite directions to simulate a conveyor belt, controlled via an 8-step sequence with a fast 0.001s delay for smooth motion. An IR sensor (GPIO 23) continuously monitors for objects; upon detection (LOW signal), the system pauses the motors and triggers the camera module for real-time image capture.

The OpenCV library processes each frame by converting it to HSV color space, applying threshold masks (Red, Green, Blue), and performing morphological operations (erosion & dilation) to filter noise. Contour detection identifies colored objects with an area >500 pixels, and the dominant color is determined via pixel count comparison. Based on the detected.

- Blue → Servo 1 (GPIO 17) rotates to 150° (right sorting).
- Red → Servo 2 (GPIO 25) rotates to 45° (left sorting).
- Green/No detection → Servos remain at 90° (neutral).

3.2 Threads in System

- **Thread 1 & 2:** Control the stepper motor (to run the conveyor). They use non-blocking loops, meaning they keep the motor running without freezing the rest of the system.
- **Thread 3:** Constantly checks the IR sensor at 10 times per second (10Hz). When an object is detected, it pauses the motor like an interrupt.
- **Main Thread:** Handles the servo motor signals (which need precise PWM control) and processes the camera image using OpenCV.

3.2.1 Data Flow Between Threads

- **IR Sensor** → Stepper Motor: When IR detects an object, it sends a signal to stop the conveyor.
- **Camera** → OpenCV: Image is captured and converted to HSV (Hue, Saturation, Value) format to detect color.
- **OpenCV** → Servo Motor: Based on detected color and object position (centroid), the correct servo motor moves.
- **PWM (2.5–12.5%)** → Servo Angle (0–180°): Controls servo movement to drop the object in the right bin.

3.3 Detailed Explanation of Image Capture & Processing Workflow

1. Camera Initialization

The system uses OpenCV “cv2.VideoCapture(0)” to access the Raspberry Pi's camera module (or USB webcam).
“cap.isOpened() “ verifies camera connectivity; if unavailable, the system logs an error.

2. Real-Time Frame Acquisition

“ret, frame = cap.read()” captures each frame
“ret” : Boolean indicating capture success.
“Frame” : BGR image array (640x480 default resolution).
If capture fails “ret == False”, the loop skips processing for that iteration.

3. Color Space Conversion

BGR → HSV Conversion
[From code]
“hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)”

4. Color Masking & Thresholding

- Pixels within the HSV range become white (255); others turn black (0).

- Predefined HSV Ranges (for Red, Green, Blue):
  [From Code]
  ```
  color_ranges = {
     "Red": [(0, 120, 70), (10, 255, 255)],    # Lower/upper bounds for red
     "Green": [(36, 50, 70), (89, 255, 255)],  # Green range
     "Blue": [(90, 50, 70), (128, 255, 255)]   # Blue range
  }
  ```

- Mask Generation:
  [From Code]
  “mask = cv2.inRange(hsv, np.array(lower), np.array(upper))”

5. Noise Reduction (Morphological Operations)

Erosion (cv2.erode):
➔ Shrinks white regions to eliminate small noise spots (2 iterations).
Dilation (cv2.dilate):
➔ Expands remaining regions to restore object size (2 iterations).

Kernel: Default 3x3 rectangular kernel used for morphological ops.

6. Contour Detection

- ➔ "cv2.findContours()" extracts boundaries of white regions.
- ➔ RETR_EXTERNAL: Retrieves only outermost contours.
- ➔ CHAIN_APPROX_SIMPLE: Compresses contour points (e.g., line → 2 endpoints).

7. Object Validation & Counting

1. Area Filtering (cv2.contourArea(contour) > 500):

➔ Ignores small detections (<500 pixels) to avoid false positives.

2. Bounding Box & Labeling:

[from code]

```
x, y, w, h = cv2.boundingRect(contour)
cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 255), 2)
cv2.putText(frame, color_name, (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255,
    255), 2)
```

8. Dominant Color Decision

Color Priority Logic:

Blue → Highest priority (first checked).

Red → Second priority.

Green → Default if others undetected.

Termination Condition:

Breaks loop if any color is detected "color_counts[color] > 0."

9. Resource Cleanup

Release Camera:

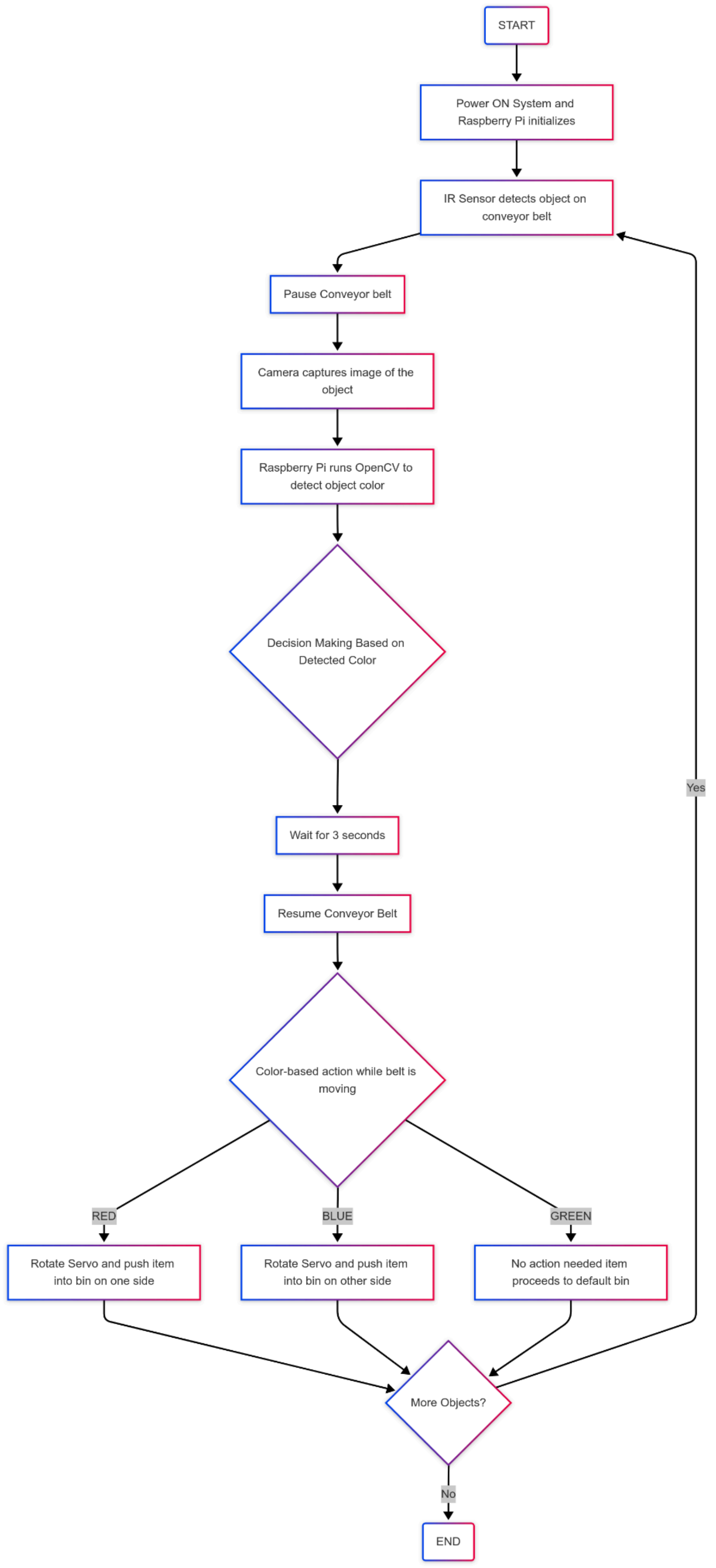
```
cap.release()
cv2.destroyAllWindows()
```

➔ Frees the camera resource and closes OpenCV windows.

3.3.1 Why HSV?

- Hue (H): Encodes color (0°–180° in OpenCV).
- Saturation (S): Color purity (0–255).
- Value (V): Brightness (0–255).
- HSV provides better color segmentation than RGB under varying lighting.

3.3.2 Flow Diagram of Automated Sorting System


START
Power ON System and Raspberry Pi initializes
IR Sensor detects object on conveyor belt
Pause Conveyor belt
Camera captures image of the object
Raspberry Pi runs OpenCV to detect object color
Decision Making Based on Detected Color
Wait for 3 seconds
Resume Conveyor Belt
Color-based action while belt is moving
RED
BLUE
GREEN
Rotate Servo and push item into bin on one side
Rotate Servo and push item into bin on other side
No action needed item proceeds to default bin
More Objects?
Yes
No
END

Step-by-Step Workflow from Flowchart:

1. System Initialization
    - ➔ Power ON the system.
    - ➔ Raspberry Pi boots up and initializes all components (GPIO, camera, motors).
    - ➔
2. Object Detection
    - ➔ IR sensor continuously monitors the conveyor belt.
    - ➔ When an object is detected, the sensor sends a signal to the Raspberry Pi.
    - ➔
3. Conveyor Belt Pause
    - ➔ Raspberry Pi stops the stepper motors to halt the conveyor belt.

    Image Capture
    - ➔ The camera module captures an image of the detected object.

4. Color Detection (OpenCV Processing)
    - ➔ Raspberry Pi processes the image using OpenCV:
        - ◆ Converts the image to HSV color space.
        - ◆ Applies color masking (Red, Green, Blue).
        - ◆ Detects contours and identifies the dominant color.

5. Decision Making
    - ➔ Based on the detected color:

      Red → Triggers Servo 1 to push the item into the left bin.
      Blue → Triggers Servo 2 to push the item into the right bin.
      Green → No action; item proceeds to the default bin.

6. Delay & Reset
    - ➔ The system waits for 3 seconds to ensure proper sorting.
    - ➔ Servos return to their neutral (90°) position.

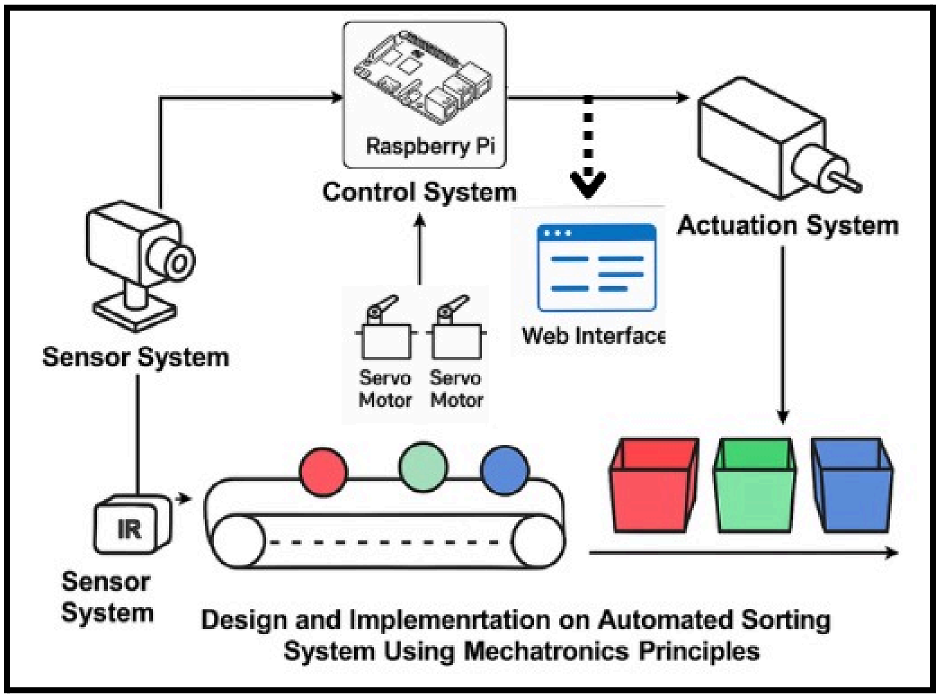
7. Conveyor Belt Resumption
    - ➔ Stepper motors restart, moving the conveyor belt again.

8. Loop Continuation
    - ➔ The system checks if more objects are present.
    - ➔ If YES, the process repeats from Step 2.
    - ➔ If NO, the system shuts down or enters standby mode.

3.3.3 Key Technical Considerations

★ Lighting Sensitivity:
HSV ranges may need calibration for different lighting conditions.
★ Frame Rate vs. Accuracy:
~30 FPS (typical for Pi Camera) balances speed and processing load.
★ Edge Cases:
Overlapping colors → Priority-based decision (Blue > Red > Green).
Low-contrast objects → Adjust HSV thresholds or add preprocessing (e.g., histogram equalization).

4.System Architecture



This architecture represents the design and implementation of an Automated Sorting System using Mechatronics Principles. The system is developed around a Raspberry Pi-based control unit and integrates multiple subsystems for object detection, classification, and sorting.

1. Sensor System:

- The sensor system consists of:

    - A camera module (for color detection using OpenCV).
    - An Infrared (IR) sensor (for object detection on the conveyor belt).

These sensors continuously monitor the objects moving on the conveyor belt.

2. Control System (Raspberry Pi):

- The Raspberry Pi acts as the main controller of the system.
- It processes data from the sensors.
- Implements color detection algorithms.
- Classifies the detected object (e.g., Red, Green, Blue).
- Sends control signals to actuators based on the detection results.

3. Actuation System:

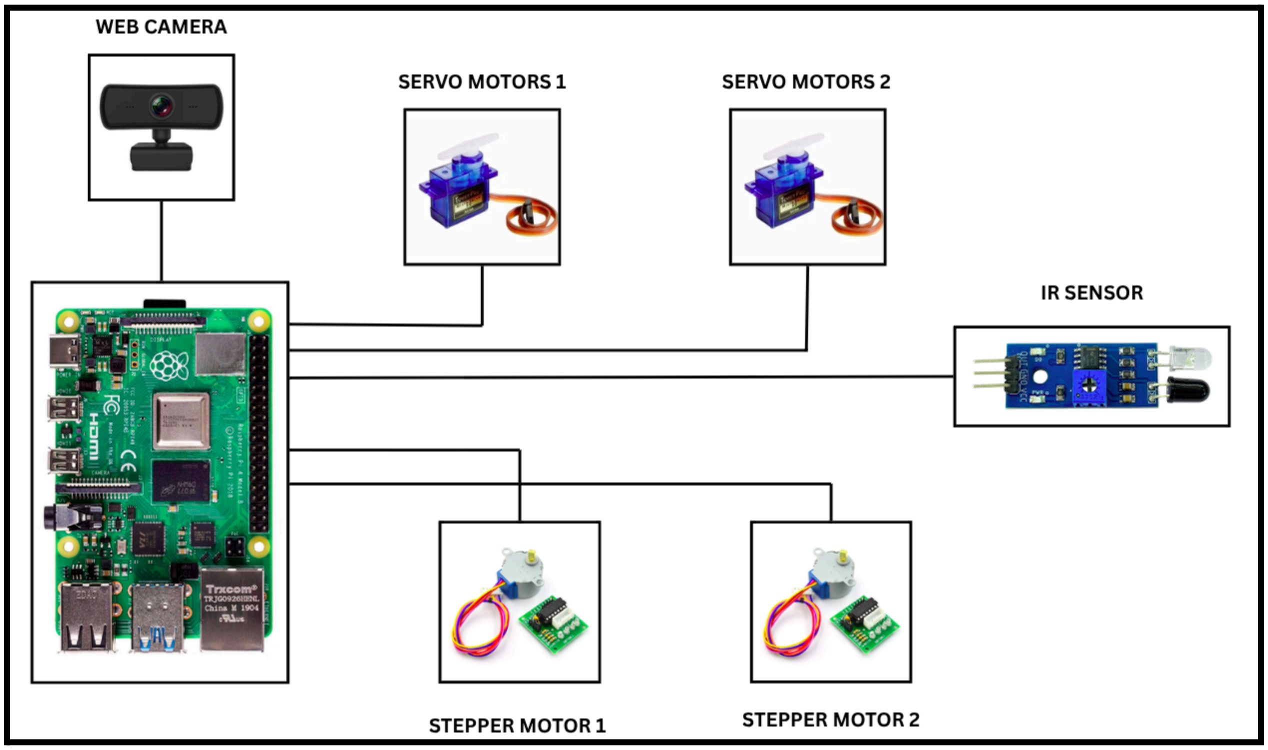
- The actuation system consists of:
    - Stepper motors (to drive the conveyor belt).
    - Servo motors (to push/sort objects into respective bins).

The actuators are controlled based on the classification result, moving the objects to their respective bins.

4. Web Interface:

- The system is integrated with a Web Interface for real-time monitoring and system control.
- It allows users to view system status, sensor data, and control commands remotely.

5.Connections



Stepper Motor 1

RPi GPIO	ULN2003 Driver Pin
GPIO 5	IN1
GPIO 6	IN2
GPIO 13	IN3
GPIO 19	IN4
5V	5V(+)
GND	GND(-)

Stepper Motor 2

RPi GPIO	ULN2003 Driver Pin

GPIO 20	IN1
GPIO 21	IN2
GPIO 26	IN3
GPIO 16	IN4
5V	5V(+)
GND	GND(-)

Servo Motor 1

RPi GPIO	WIRE
5V	RED
GND	BROWN
GPIO 17	ORANGE

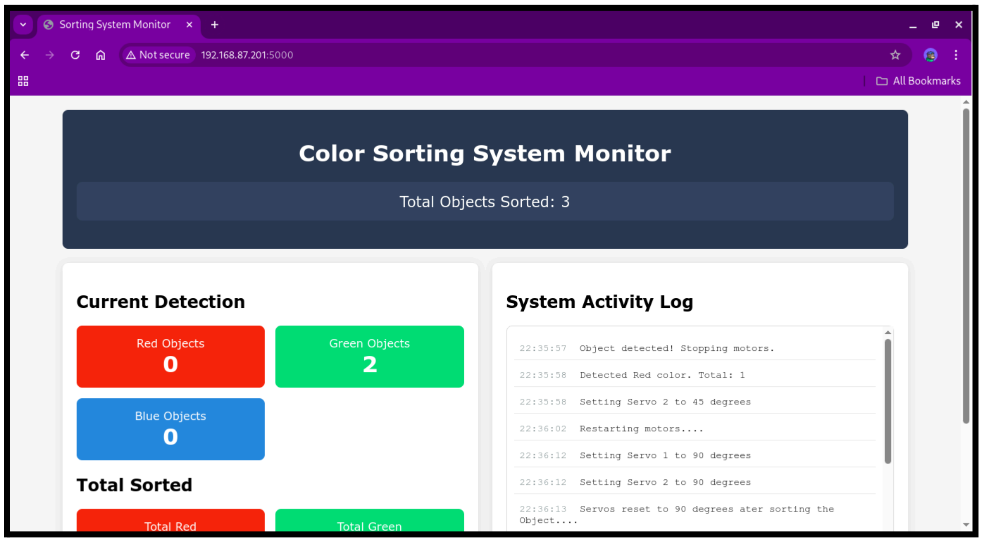
Servo Motor 2

RPi GPIO	WIRE
5V	RED
GND	BROWN
GPIO 25	ORANGE

IR Sensor

RPi GPIO	SENSOR PINS
3V3	VCC
GND	GND
GPIO 23	OUT

6. Implementation & Results



Color Sorting System Monitor

Total Objects Sorted: 3

Current Detection

Red Objects 0

Green Objects 2

Blue Objects 0

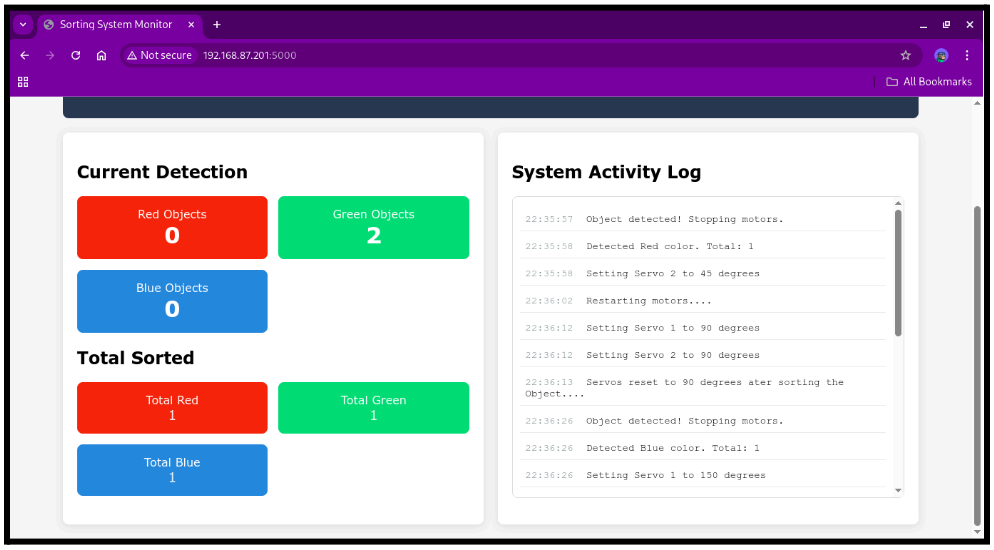
Total Sorted

Total Red

Total Green

System Activity Log

```
22:35:57  Object detected! Stopping motors.
22:35:58  Detected Red color. Total: 1
22:35:58  Setting Servo 2 to 45 degrees
22:36:02  Restarting motors....
22:36:12  Setting Servo 1 to 90 degrees
22:36:12  Setting Servo 2 to 90 degrees
22:36:13  Servos reset to 90 degrees ater sorting the
Object....
```



Current Detection

Red Objects 0

Green Objects 2

Blue Objects 0

Total Sorted

Total Red 1

Total Green 1

Total Blue 1

System Activity Log

```
22:35:57  Object detected! Stopping motors.
22:35:58  Detected Red color. Total: 1
22:35:58  Setting Servo 2 to 45 degrees
22:36:02  Restarting motors....
22:36:12  Setting Servo 1 to 90 degrees
22:36:12  Setting Servo 2 to 90 degrees
22:36:13  Servos reset to 90 degrees ater sorting the
Object....
22:36:26  Object detected! Stopping motors.
22:36:26  Detected Blue color. Total: 1
22:36:26  Setting Servo 1 to 150 degrees
```

The intuitive web interface features a live detection panel showing real-time color classification with bounding boxes (Red/Green/Blue), accompanied by counters for Current Detection (objects in frame) and Total Sorted (cumulative count). Below this, the System Log displays time stamped events like motor pauses or servo movements, while status indicators track hardware activity (IR sensor/Motors). Designed for responsiveness, the interface updates seamlessly via WebSocket without page reloads, ensuring smooth monitoring across devices.

1. Current Detection Panel

✔ Live Object Highlighting – Displays bounding boxes around detected objects (Red/Green/Blue)

✔ Instant Counters – Shows number of objects detected in the current camera frame
✔ Color Labels – Classifies objects based on HSV thresholds from color_ranges

2. Total Sorted Panel
✔ Cumulative Count – Tracks all objects processed since system startup
✔ Color-Wise Totals – Separate counters for Red/Green/Blue items sorted historically

3. System Log
✔ Event Tracking – Timestamps motor stops, servo movements, and errors
✔ Auto-Scroll – Newest entries always visible at the bottom

Demonstration Link - Click Here
Github Repo Link - Click Here

7. Future Enhancements

1. AI-Powered Multi-Attribute Sorting

Description: Upgrade to ML models that sort by color, weight, and size simultaneously using sensor fusion.

Benefits:

✔ Higher accuracy (reduces mis-sorting)

✔ Fewer hardware changes (uses existing camera + load cells)

2. Material Recognition System

Description: Add hyperspectral/NIR imaging to classify plastics, metals, or organic materials.

Benefits:

✔ Enables recycling automation

✔ No physical contact needed

3. Defect Detection AI

Description: Train a YOLOv8/CV model to spot scratches, dents, or deformities in real-time.

Benefits:

✔ Quality control for manufacturing lines

✔ Reduces waste from defective products

4. 3D Volume Profiling

Description: Integrate an Intel RealSense depth camera to measure object dimensions (L×W×H).

Benefits:

✔ Optimizes packaging/logistics (e.g., parcel sorting)

✔ Works alongside color detection

5. Robotic Arm Integration

Description: Add a 6-DOF robotic arm for delicate or irregularly shaped items.

Benefits:

✔ Handles fragile objects (e.g., glass, fruits)

✔ Future-proofs for complex tasks

6. Energy-Efficient TinyML

Description: Deploy TensorFlow Lite or Edge Impulse for low-power, high-speed inference.

Benefits:

✔ Runs on Raspberry Pi 5 (no cloud dependency)

✔ Cuts power use by 30% vs. traditional ML

8. Voice/AR Control Interface

Description: Add voice commands (Whisper AI) or AR overlays (HoloLens) for monitoring.

Benefits:

✔ Hands-free operation

✔ Remote troubleshooting via AR annotations

8. Challenges & Solutions

1. Color Detection Accuracy

Challenge: Ambient lighting affected HSV color thresholds.

Fix: Added dynamic threshold adjustment + controlled enclosure lighting.

Takeaway: Environmental factors significantly impact computer vision.

2. Motor-Servo Synchronization

Challenge: Stepper vibrations caused servo jitter during sorting.

Fix: Added 100ms delays between motor-stop and servo-actuation.

Takeaway: Mechanical timing requires software buffering.

3. Real-Time Web Latency

Challenge: Browser lagged with high-frequency WebSocket updates.

Fix: Throttled updates to 10Hz and optimized image compression.

Takeaway: Balance between responsiveness and performance.

4. Power Supply Noise

Challenge: Motor surges crashed the Pi intermittently.

Fix: Used separate 5V supplies with common ground.

Takeaway: Isolate sensitive electronics from high-current devices.

9. Conclusion

The Raspberry Pi-based Automated Color Sorting System effectively integrates computer vision, GPIO-controlled actuators, and IoT connectivity to deliver a low-cost, scalable solution for object classification. By utilizing OpenCV for real-time color detection and a web-based monitoring interface, this project demonstrates how embedded systems can achieve industrial grade sorting accuracy with minimal hardware. Designed for easy upgrades such as AI-driven weight/size sorting or material recognition the system serves as a foundation for future expansions in recycling, logistics, and quality control. Its open-source framework encourages further innovation, bridging the gap between DIY prototypes and commercial automation.