Intel® Unnati Industrial Training 2025
Project Report

Title: Automated Object Sorting System Using Mechatronics Principles
Candidate Name: Marcel Basumatary
Mentor: Tandrali Ray (Assistant Professor)
Institute: The Assam Kaziranga University

1. Introduction

This document presents the design and development of an automated object sorting system based on mechatronics principles. The system utilizes a Raspberry Pi microcontroller, camera-based color detection, a proximity sensor, and a six-servo motor robotic arm. A conveyor belt transports various objects—each differing in shape and color (red, green, or blue)—to a sorting station, where they are automatically classified and placed in designated trays.

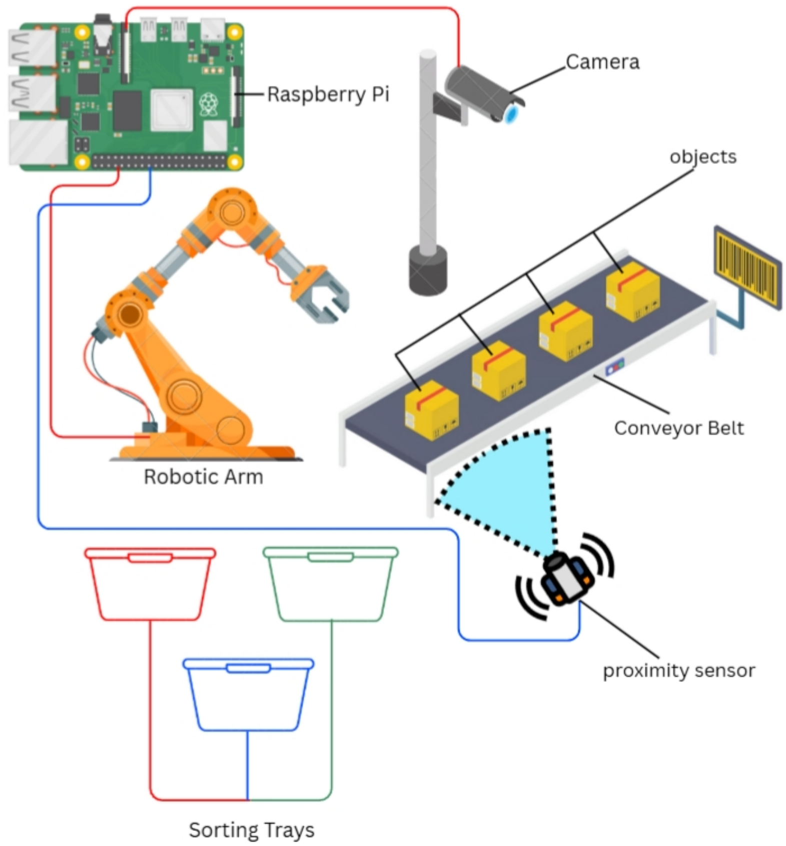
2. System Components

- **Microcontroller:** Raspberry Pi (central control unit)
- **Camera:** Used for real-time detection and classification of object color
- **Proximity Sensor:** Identifies the object's presence at the pickup zone
- **Robotic Arm:** A six degrees of freedom (6-DOF) arm controlled via six servo motors for precise manipulation
- **Conveyor Belt:** Continuously delivers objects to the detection area
- **Sorting Trays:** Labeled bins for Red, Green, and Blue objects
- **Objects:** Shapes such as cubes, cones, and spheres, in red, green, or blue

3. Functional Overview

1. Objects are carried via conveyor belt into the camera's detection zone.
2. The Raspberry Pi captures and processes the image to determine the object's color.
3. When the proximity sensor detects that the object has reached the pickup zone, a signal is sent to the Raspberry Pi.
4. The Raspberry Pi calculates the object's position and commands the robotic arm to pick the object.
5. Depending on the detected color, the robotic arm moves and deposits the object into the appropriate color-coded tray.
6. The arm then resets to its home position, ready to sort the next object.

4. System Diagram



5. Python Implementation

a. Color Detection Using Camera and OpenCV

```
import cv2
import numpy as np

def detect_color(frame):
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    red_mask = cv2.inRange(hsv, (0, 100, 100), (10, 255, 255))
    green_mask = cv2.inRange(hsv, (40, 70, 70), (80, 255, 255))
    blue_mask = cv2.inRange(hsv, (100, 150, 0), (140, 255, 255))

    if red_mask.sum() > green_mask.sum() and red_mask.sum() > blue_mask.sum():
        return 'red'
    elif green_mask.sum() > red_mask.sum() and green_mask.sum() > blue_mask.sum():
        return 'green'
    elif blue_mask.sum() > red_mask.sum() and blue_mask.sum() > green_mask.sum():
        return 'blue'
    else:
        return 'unknown'
```

b. Proximity Sensor Detection Using GPIO

```python
import RPi.GPIO as GPIO
import time

PROXIMITY_PIN = 17
GPIO.setmode(GPIO.BCM)
GPIO.setup(PROXIMITY_PIN, GPIO.IN)

def object_detected():
    return GPIO.input(PROXIMITY_PIN) == GPIO.HIGH
```

c. Robotic Arm Control Using Adafruit ServoKit

```python
from adafruit_servokit import ServoKit
import time

kit = ServoKit(channels=16)

positions = {
    'home': [90, 90, 90, 90, 90, 90],
    'red': [45, 80, 100, 60, 90, 90],
    'green': [60, 70, 90, 65, 90, 90],
    'blue': [75, 60, 85, 70, 90, 90]
}

def move_arm(position):
    for i, angle in enumerate(positions[position]):
        kit.servo[i].angle = angle
        time.sleep(0.1)
```

d. Main Sorting Logic with Real-Time Feedback

```python
import cv2
import time
from proximity_sensor import object_detected
from color_detection import detect_color
from robotic_arm_control import move_arm

sorted_counts = {'red': 0, 'green': 0, 'blue': 0, 'unknown': 0}

cap = cv2.VideoCapture(0)

try:
    while True:
        if object_detected():
            ret, frame = cap.read()
            if not ret:
                print("Camera error: Frame not captured.")
                continue

            color = detect_color(frame)
            print(f"Detected color: {color}")

            if color in ['red', 'green', 'blue']:
                move_arm(color)
                sorted_counts[color] += 1
                print(f"Item placed in {color} tray. Total: {sorted_counts[color]}")
            else:
                sorted_counts['unknown'] += 1
                print("Error: Color could not be classified.")

            time.sleep(1)
            move_arm('home')

except KeyboardInterrupt:
    print("\nSorting stopped by user.")
    print("Final sorting counts:")
    for color, count in sorted_counts.items():
        print(f"{color.capitalize()}: {count}")

    cap.release()
```

6. Conclusion

The automated object sorting system efficiently demonstrates the integration of mechanical, electronic, and software components for real-time decision-making. By automating object classification and manipulation, the system reduces human error and increases operational efficiency. The design is modular, making it suitable for upgrades and expansion.