Automated Sorting System using Mechatronics principle

Detailed Documentation

---

1. Introduction

The Automated Mechatronic Sorting System is a Python-based project designed to mimic an industrial robotic arm's sorting mechanism. The robotic arm picks up objects with different colors, shapes, and sizes and places them into their respective bins. This simulation utilizes OpenCV for visualization, ensuring smooth robotic movement and object interactions.

2. Problem Statement

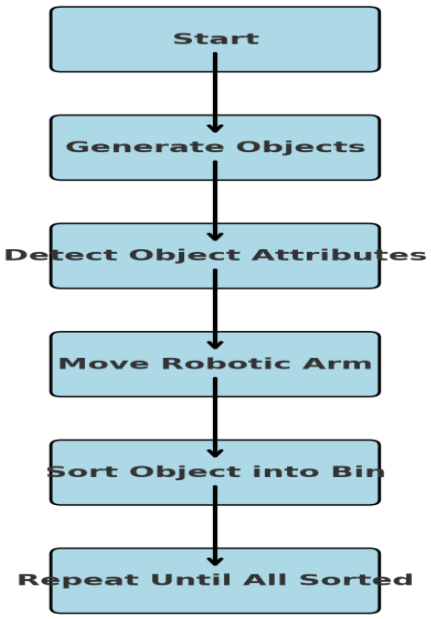
The challenge in industrial automation is to efficiently sort objects based on predefined attributes such as color, shape, or size. Manual sorting is inefficient and prone to errors, making automated sorting systems an essential component of modern manufacturing units. This project aims to simulate a robotic sorting system that autonomously identifies and sorts objects into predefined bins using mechatronics principles.

---

3. Approach

The approach involves:

- **Object Recognition: Randomly generating objects with different properties.**
- **Path Planning: Simulating a robotic arm’s movement to pick and place objects efficiently.**
- **Animation and Visualization: Using OpenCV to create smooth and realistic robotic animations.**
- **Sorting Logic: Sorting objects into color-coded bins based on their attributes.**



.

---

4. Architecture

The simulation follows a modular architecture that includes:

4.1 Object Generation Module

- **Randomly generates objects with attributes:**
  - **Colors: Red, Green, Blue**
  - **Shapes: Circle, Square, Triangle**
  - **Sizes: Small to Large**

4.2 Robotic Arm Control Module

- **Simulates robotic arm movement using:**
  - **Joint Rotation Calculations for smooth transitions**
  - **Gripper Mechanism for picking objects**
  - **Step-by-Step Motion Planning**

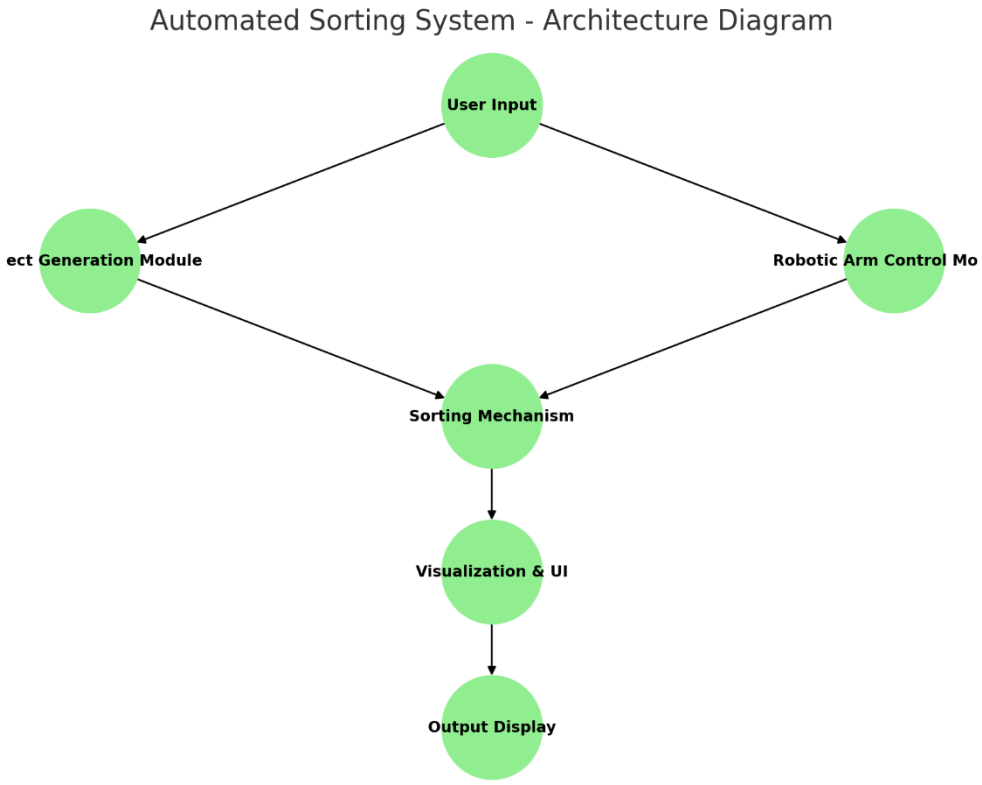
4.3 Sorting Mechanism

- **Detects object attributes and places them in respective bins:**

- Red Objects → Red Bin
- Green Objects → Green Bin
- Blue Objects → Blue Bin

4.4 Visualization & Animation

- **Warehouse Layout with bright and vibrant backgrounds**
- **Enhanced UI with labeled bins and smooth robotic movement**

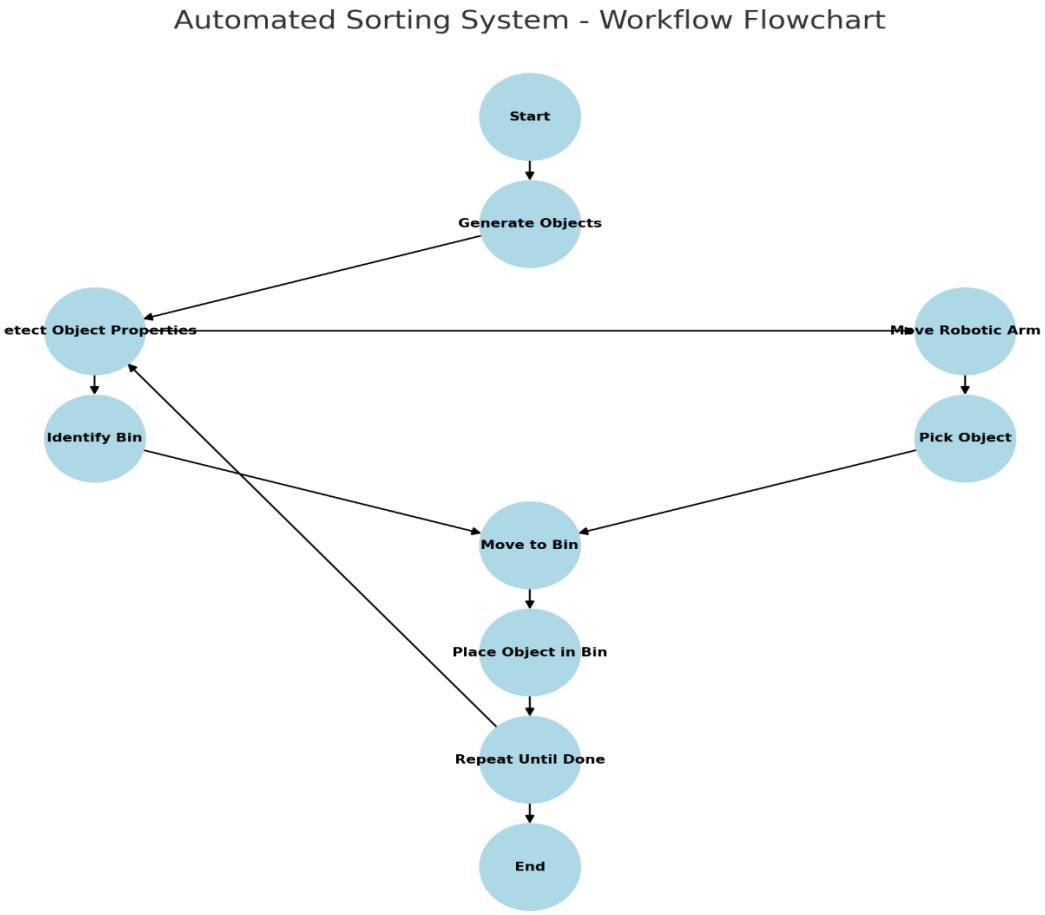


---

5. Workflow

1. **Object Generation: Randomly create objects with different colors, shapes, and sizes.**
2. **Robotic Arm Activation: The arm starts in an idle state, awaiting object pickup.**
3. **Object Detection & Sorting: The robotic arm identifies an object's color and moves to the respective bin.**
4. **Placement in Bins: Objects are placed in corresponding bins according to color.**

5. **Repeat Process: The cycle continues until all objects are sorted.**



---

6. Features

✅ **Realistic Robotic Arm Movement**

- The robotic arm moves smoothly to pick and place objects.
- Objects are scattered instead of stacked, making sorting more challenging.

✅ **Random Object Generation**

- The simulation randomly generates objects with different attributes.

✅ **Sorting into Three Compartments**

- Objects are placed into three separate bins instead of a stack.

✅ **Enhanced UI with Vibrant Background**

- The simulation uses bright colors for better visualization.

✅ **Realistic Robotic Hand with Wires**

- **The robotic hand is drawn with multiple wire-like structures for a mechanical feel.**

---

7. Technologies Used

Technology	Purpose
Python	Core programming language
OpenCV	Computer vision and object visualization
NumPy	Handling object positions and movement calculations
Random Module	Generating randomized object properties

---

8. Code Structure

8.1 File Structure

/automated-sorting-sim/

|— robotic_sorting.py # Main script

|— README.md # Documentation

|— assets/ # Store any images if needed

8.2 Functions Overview

Function	Purpose
generate_objects(num_objects=5)	Generates random objects with color, shape, and size
draw_shape(img, shape, position, size, color)	Draws circle, square, or triangle on the screen
draw_robotic_hand(img, position)	Simulates a robotic hand with wire-like structures
robotic_arm_simulation(objects)	Moves the robotic arm to pick and place objects

---

9. Installation & Setup

9.1 Prerequisites

Ensure you have Python 3.7 or above installed on your system.

9.2 Install Dependencies

Run the following command to install the necessary libraries:

pip install opencv-python numpy

9.3 Running the Simulation

Once the dependencies are installed, you can execute the script:

python robotic_sorting.py

This will launch the sorting simulation window.

---

10. Working Mechanism

10.1 Object Generation

1. **Randomly generates objects with different shapes, sizes, and colors.**
2. **Objects appear in scattered locations rather than stacked.**

10.2 Robotic Arm Movement

1. **The arm moves down to pick an object.**
2. **It lifts the object and moves toward the corresponding color bin.**
3. **Finally, it places the object in the bin and moves back to the next object.**

10.3 Sorting Process

Objects are sorted into three bins based on their colors.

CODE:

```
import cv2
import numpy as np
import random
import time


# Define object properties
COLORS = {'red': (0, 0, 255), 'green': (0, 255, 0), 'blue': (255, 0, 0)}
```

```
SHAPES = ['circle', 'square', 'triangle']
PICK_POSITION = (300, 150)
PLACE_POSITIONS = {'red': (100, 300), 'green': (300, 300), 'blue': (500, 300)}


# Generate stacked objects
def generate_objects(num_objects=5):
    objects = []
    stack_x, stack_y = 300, 100
    for i in range(num_objects):
        color = random.choice(list(COLORS.keys()))
        shape = random.choice(SHAPES)
        size = random.randint(20, 50)
        objects.append({'color': color, 'shape': shape, 'size': size, 'position': (stack_x, stack_y - i * 30)})
    return objects


# Draw different shapes
def draw_shape(img, shape, position, size, color):
    if shape == 'circle':
        cv2.circle(img, position, size, color, -1)
    elif shape == 'square':
        top_left = (position[0] - size, position[1] - size)
        bottom_right = (position[0] + size, position[1] + size)
        cv2.rectangle(img, top_left, bottom_right, color, -1)
    elif shape == 'triangle':
        pts = np.array([[position[0], position[1] - size],
                        [position[0] - size, position[1] + size],
                        [position[0] + size, position[1] + size]], np.int32)
```

```
        cv2.fillPoly(img, [pts], color)


# Draw a warehouse-like background
def draw_warehouse_background(img):
    img[:] = (210, 210, 210)  # Light grey background
    cv2.rectangle(img, (50, 50), (550, 350), (180, 180, 180), -1)  # Warehouse floor
    cv2.rectangle(img, (100, 50), (500, 80), (120, 120, 120), -1)  # Shelf top
    cv2.rectangle(img, (100, 80), (120, 250), (100, 100, 100), -1)  # Left shelf
    cv2.rectangle(img, (480, 80), (500, 250), (100, 100, 100), -1)  # Right shelf
    cv2.putText(img, "Warehouse Sorting System", (140, 30), cv2.FONT_HERSHEY_SIMPLEX,
0.8, (50, 50, 50), 2)


# Draw a realistic robotic arm with joints
def draw_robotic_arm(img, shoulder, elbow, wrist, gripper):
    # Draw arm segments
    cv2.line(img, shoulder, elbow, (50, 50, 50), 8)
    cv2.line(img, elbow, wrist, (50, 50, 50), 6)


    # Draw joints
    cv2.circle(img, shoulder, 10, (0, 0, 0), -1)
    cv2.circle(img, elbow, 8, (0, 0, 0), -1)
    cv2.circle(img, wrist, 6, (0, 0, 0), -1)


    # Draw gripper
    cv2.rectangle(img, (gripper[0] - 10, gripper[1] - 5), (gripper[0] + 10, gripper[1] + 5), (0, 0,
0), -1)


# Simulate robotic arm picking and placing
def robotic_arm_simulation(objects):
```

```
    for obj in objects:
        print(f"Picking Object -> Color: {obj['color'].capitalize()}, Shape: {obj['shape'].capitalize()}, Size: {obj['size']}")

        # Create warehouse background
        img = np.zeros((400, 600, 3), dtype=np.uint8)
        draw_warehouse_background(img)

        color_bgr = COLORS[obj['color']]
        place_position = PLACE_POSITIONS[obj['color']]

        # Draw stacked objects
        for stacked_obj in objects:
            draw_shape(img, stacked_obj['shape'], stacked_obj['position'], stacked_obj['size'], COLORS[stacked_obj['color']])

        # Draw bins
        cv2.rectangle(img, (50, 280), (150, 350), (0, 0, 0), 2)
        cv2.rectangle(img, (250, 280), (350, 350), (0, 0, 0), 2)
        cv2.rectangle(img, (450, 280), (550, 350), (0, 0, 0), 2)
        cv2.putText(img, "Red Bin", (60, 370), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 2)
        cv2.putText(img, "Green Bin", (260, 370), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 2)
        cv2.putText(img, "Blue Bin", (460, 370), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 0), 2)

        # Move robotic arm step by step
        shoulder = (300, 50)
        elbow = (300, 100)
        wrist = (obj['position'][0], 150)
```

```
    gripper = obj['position']

    for step in range(150, 250, 10):
        frame = img.copy()
        wrist = (wrist[0], step)
        gripper = (gripper[0], step)
        draw_robotic_arm(frame, shoulder, elbow, wrist, gripper)
        draw_shape(frame, obj['shape'], gripper, obj['size'], color_bgr)
        cv2.imshow("Robotic Arm Simulation", frame)
        cv2.waitKey(100)

    # Move to placement bin
    for step_x in range(gripper[0], place_position[0], 10 if place_position[0] > gripper[0] else -10):
        frame = img.copy()
        wrist = (step_x, 250)
        gripper = (step_x, 250)
        draw_robotic_arm(frame, shoulder, elbow, wrist, gripper)
        draw_shape(frame, obj['shape'], gripper, obj['size'], color_bgr)
        cv2.imshow("Robotic Arm Simulation", frame)
        cv2.waitKey(100)

    # Final placement
    frame = img.copy()
    draw_robotic_arm(frame, shoulder, elbow, wrist, place_position)
    draw_shape(frame, obj['shape'], place_position, obj['size'] - 5, color_bgr)  # Shrink object slightly
    cv2.imshow("Robotic Arm Simulation", frame)
    cv2.waitKey(500)
```

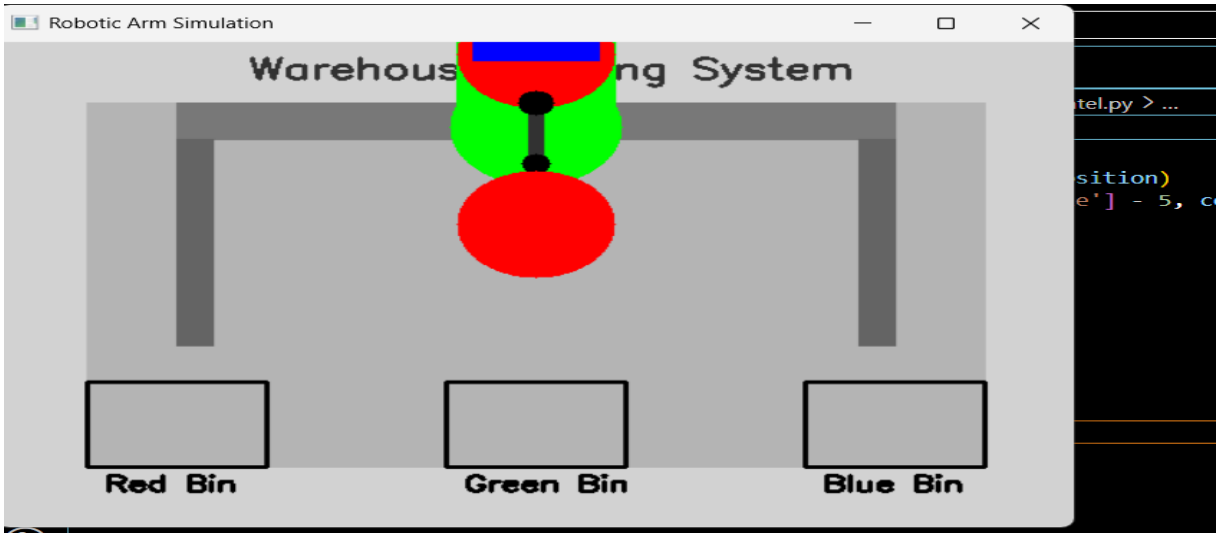
```
        print(f"Placed in {obj['color'].capitalize()} Bin ✅")
        print("Objects are being sorted...")
        time.sleep(0.5)

    cv2.destroyAllWindows()

# Generate and simulate
objects = generate_objects(5)
robotic_arm_simulation(objects)
```

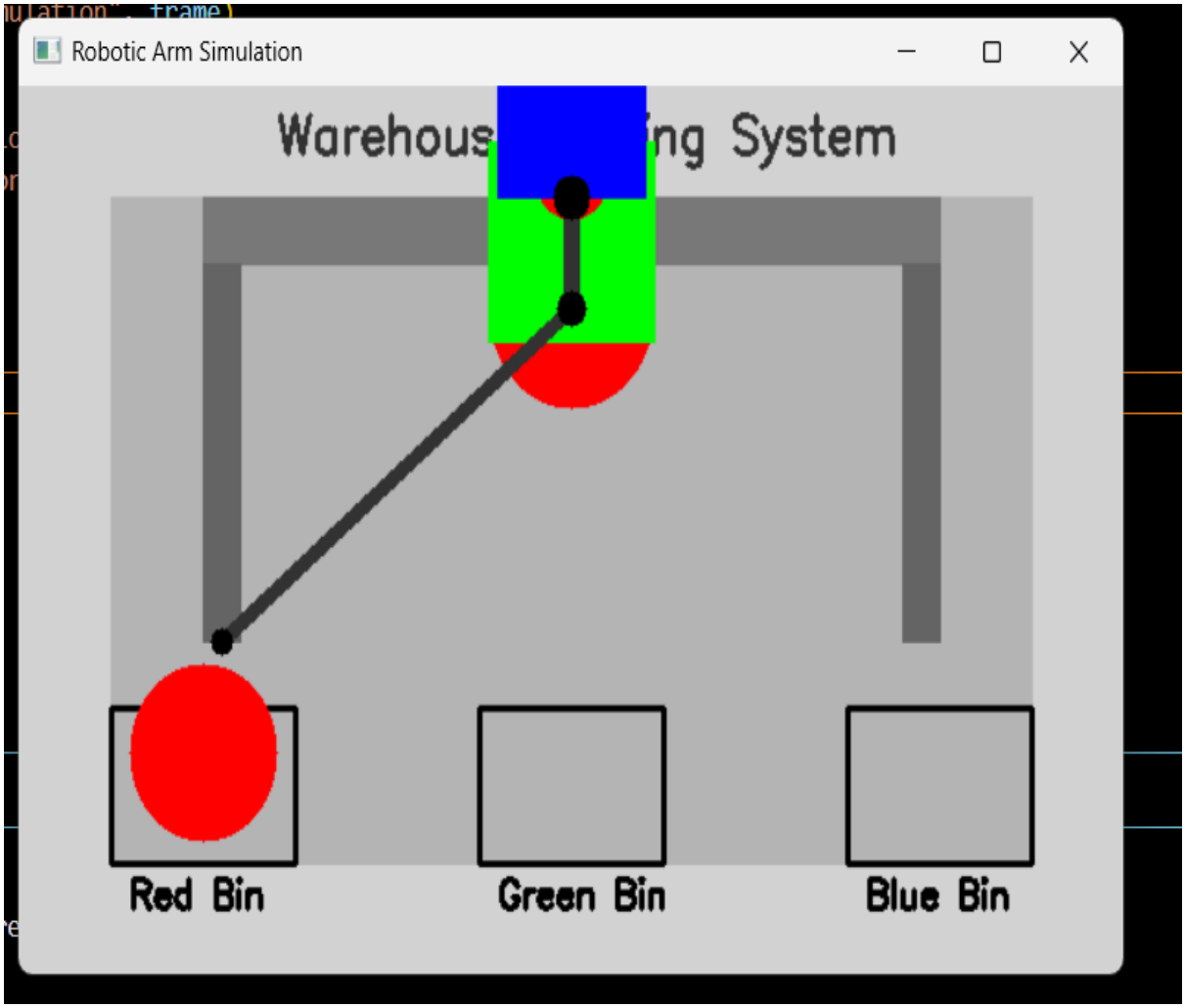
---

11. Output

The simulation will display:

- **A warehouse environment with a robotic arm and sorting bins.**
- **Objects appearing in random locations.**
- **The robotic arm moving smoothly to pick and place objects.**
- **Objects correctly sorted into their bins.**





---

12. Enhancements & Future Improvements

🚀 **Machine Learning Integration: Use AI-based object recognition for better decision-making.**

🚀 **3D Simulation in Unity: Enhance the simulation with 3D modeling for real-world application.**

🚀 **Physics-Based Simulation: Implement ROS (Robot Operating System) for real physics interaction.**

🚀 **User Interaction: Allow users to manually control the robotic arm using keyboard/mouse.**

---

13. Conclusion

This project effectively simulates an automated sorting process using a robotic arm in Python. By combining OpenCV animations, random object generation, and robotic movement, it provides an excellent demonstration of real-world robotic applications.

VIDEO LINK : https://youtu.be/NB4fNkeCd9E