Virtual Color Sorting Machine in Unity

1. Introduction

Automated sorting systems play a crucial role in industries by classifying and organizing objects based on specific attributes. This project presents a **mechatronics-based virtual simulation** in **Unity**, demonstrating an **automated color sorting system**.

Application of Mechatronics Principles

- **Integration of sensors, actuators, and control systems**: The project uses a **color-detecting sensor** to analyze the color of objects in real time.
- **Effective use of embedded systems and automation techniques**: Sorting decisions are automated through Unity scripts.
- **Mechanical design of the sorting mechanism (stability, reliability)**: The system ensures stability during operation through a well-structured sorting algorithm.

2. Objectives

- **To design a functional and automated sorting system in Unity.**
- **To implement a robust color detection mechanism for VIBGYOR colors.**
- **To optimize sorting efficiency and reduce errors in classification.**
- **To ensure real-time sorting through AI/ML algorithms.**

AI Implementation

- **Use of AI/ML for object classification**: The project utilizes **HSL-based color matching** for accurate classification.
- **Real-time performance (speed & accuracy of sorting decisions)**: The color detection system operates efficiently, ensuring minimal delays in sorting.
- **Adaptability to new object types or dynamic conditions**: The system can be expanded to accommodate **additional color categories**.

3. Components and Working Principle

3.1 Components Used

- **Conveyor Belt**: Moves the cubes towards the color detection sensor.
- **Cube Spawning Area**: Randomly generates cubes of **VIBGYOR** colors.
- **Color Detecting Sensor**: Identifies the color of each cube.
- **Sorting Mechanism**: Redirects cubes based on detected colors.

- **Collection Boxes**: Dedicated bins for sorted cubes.

3.2 Working of the System

1. **Spawning of Colored Cubes**:
    - Randomly generated cubes are transported via a **conveyor belt**.
2. **Color Detection Process**:
    - The **sensor detects HSL-based color values**.
    - The detection algorithm maps RGB values to VIBGYOR.
3. **Sorting Mechanism**:
    - The system directs cubes into the **appropriate collection bins** using rotation and real-time decision-making.

Innovation:

- **Creative solutions to object variability**: The project handles different color intensities and variations using advanced **HSL color-matching algorithms**.
- **Unique approaches in integrating mechatronics & AI**: A **multi-dictionary color reference system** ensures adaptability in sorting decisions.

4. System Performance and Efficiency

- **Sorting accuracy**: The system ensures **high precision in color classification**.
- **Processing time per object**: Optimized Unity scripts **minimize sorting delay**.
- **Robustness**: The system maintains stability even with extended runtime.

5. Additional Features for Enhancement

- **User Interface (UI) Dashboard**: Displays real-time sorting statistics.
- **Adjustable Conveyor Speed**: Allows fine-tuning for performance testing.
- **Error Handling Mechanism**: Detects and corrects misplaced cubes.
- **Data Logging System**: Tracks system efficiency over time.

6. Conclusion

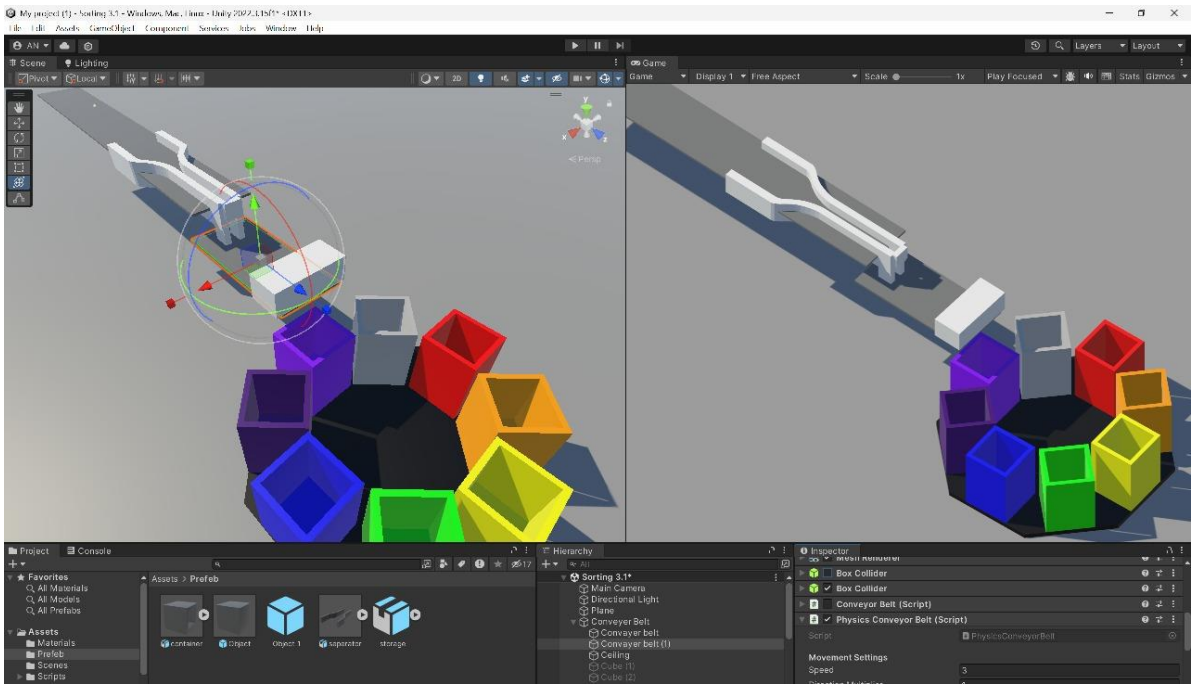
This project successfully simulates an industrial **color-sorting machine**, integrating **mechatronics and AI-based automation**. Future improvements could include **machine learning-driven self-correcting sorting mechanisms** and **advanced AI-based object recognition** for enhanced classification.

Project Description: *Virtual Color Sorting Machine in Unity*

This project is a mechatronics-based virtual simulation developed in Unity, demonstrating an automated color sorting system. The system consists of a conveyor belt that transports randomly spawned cubes with unique HSL-based colors. A sensor detects the color of each cube and categorizes it into one of eight predefined color groups: **VIBGYOR (Violet, Indigo, Blue, Green, Yellow, Orange, Red) and Grayscale**.

Upon color detection, the appropriate container on the right side of the system rotates to align with the cube's trajectory, ensuring it is sorted into the correct bin. The sorting mechanism is automated using Unity scripts, which control the conveyor movement, color detection, bin rotation, and object spawning.

This project effectively simulates an industrial color-sorting machine and showcases the integration of automation and machine perception within a virtual environment.



Title: Automated Sorting System for VIBGYOR Colored Cubes

1. Introduction

Automated sorting systems are widely used in industries to classify and organize objects based on specific attributes. In this project, we design and implement an automated sorting system that sorts cubes of different VIBGYOR (Violet, Indigo, Blue, Green, Yellow, Orange, Red) colors. This system utilizes a conveyor belt, a color-detecting sensor, and sorting mechanisms to categorize cubes into designated collection boxes efficiently.

2. Objectives

The primary objectives of this project are:

- To design a functional and automated sorting system in Unity.
- To implement a color detection mechanism that correctly identifies VIBGYOR colors.
- To ensure precise movement of cubes using a conveyor belt.
- To develop a sorting mechanism that directs cubes into the appropriate collection boxes.
- To optimize sorting efficiency and reduce errors in classification.

3. Components and Working Principle

3.1 Components Used

1. Conveyor Belt: Moves the cubes towards the color detection sensor.
2. Cube Spawning Area: Generates random cubes of VIBGYOR colors at specific intervals.
3. Color Detecting Sensor: Scans and identifies the color of each cube.
4. Sorting Mechanism: Redirects cubes based on detected colors.
5. Collection Boxes: Seven boxes placed at the end, each designated for a specific color.

3.2 Working of the System

1. Spawning of Colored Cubes
    - Cubes are randomly generated at the spawn area with predefined colors (VIBGYOR).
    - The cubes are placed onto the conveyor belt for transportation.
2. Movement via Conveyor Belt
    - The conveyor belt moves continuously, ensuring smooth transport of cubes to the detection area.
    - The speed of the belt can be adjusted for testing and optimization.

3. Color Detection Process
    - A sensor placed along the conveyor detects the color of the cube.
    - The detection algorithm identifies the RGB values or material properties of each cube.
    - Based on the detected color, the system determines the correct sorting path.
4. Sorting Mechanism
    - Once the cube's color is identified, a sorting mechanism (e.g., robotic arm, air pusher, rail switch) directs the cube toward the appropriate collection box.
    - The mechanism ensures each cube is accurately placed in the corresponding box.
5. Final Collection
    - The sorted cubes are stored in one of the seven designated collection boxes based on their color.
    - This allows for organized classification and easy retrieval.

4. How the Color Sensor Works

A color sensor detects the color of objects by analyzing the light reflected from their surfaces. The working principle of a color sensor typically follows these steps:

1. Light Emission
    - The sensor emits white light (or RGB light) onto the cube.
    - The cube absorbs some light and reflects the rest.
2. Light Reflection and Detection
    - The sensor detects the reflected light using photodetectors.
    - Different colors reflect different wavelengths of light (e.g., red reflects more red light, blue reflects more blue light).
3. Color Identification
    - The sensor measures the intensity of Red, Green, and Blue (RGB) values.
    - It compares the values to predefined thresholds to determine the cube’s color.
    - For example, if Red is dominant, the cube is categorized as Red.
4. Signal Processing and Sorting
    - Once the color is identified, the system sends a signal to the sorting mechanism.

- The sorting mechanism activates and directs the cube into the correct bin.

5. Additional Features for Enhancement

To improve functionality and efficiency, the system can be enhanced with the following features:

- User Interface (UI) Dashboard: Displays sorting statistics and real-time cube classification.
- Adjustable Conveyor Speed: Enables fine-tuning of cube movement for better sorting.
- Randomized Cube Generation: Adds variability to the testing process.
- Error Handling Mechanism: Detects misplaced cubes and corrects sorting errors.
- Data Logging System: Tracks sorting performance and efficiency over time.

6. Conclusion

This automated sorting system provides an efficient and accurate method for classifying colored cubes. The project demonstrates fundamental automation principles and can be further expanded with advanced sensors, AI integration, and real-world applications in industrial automation. Future improvements could include implementing machine learning algorithms for enhanced object recognition and self-correcting sorting mechanisms.

Code Analysis for Documentation

1. ColorDetector.cs

Main Algorithm:

The ColorDetector script identifies colors of objects in Unity using HSL (Hue, Saturation, Lightness) color space comparison. The core algorithm:

1. **Color Space Conversion**: Converts RGB colors from Unity's material system to HSL for more intuitive color matching
2. **Multi-Dictionary Approach**: Maintains three color reference dictionaries:
    - Named colors (extensive color names with HSL values)
    - VIBGYOR spectrum colors (the rainbow spectrum)
    - Grayscale colors (varying lightness levels)
3. **Distance Calculation in HSL Space**:

- For each color comparison, calculates weighted distances between the target color and reference colors
- Uses different weighting schemes for named colors vs. spectrum colors
- For grayscale comparison, only considers lightness

4. **Saturation-Based Classification**:
    - Determines if a color is more grayscale or colorful based on a saturation threshold (0.15)
    - Reports both the specific color name and broader category

The script also provides utility methods to test colors using direct HSL values or hex color codes without requiring collision detection.

2. PhysicsConveyorBelt.cs

Main Algorithm:

The PhysicsConveyorBelt script simulates physically realistic conveyor belt movement in Unity:

1. **Physics-Based Movement**:
    - Uses a zero-friction physics material to minimize sliding resistance
    - Detects objects on the belt using two methods:
        - Physics.OverlapBox for continuous detection
        - Collision detection as a backup method
2. **Velocity Control**:
    - Calculates a target velocity based on the conveyor's forward direction and speed
    - Applies forces to adjust object velocity to match the conveyor's movement
    - Preserves vertical movement while controlling horizontal velocity
3. **Object Stabilization**:
    - Applies slight downward force to keep objects on the belt
    - Ensures objects stay in contact with the conveyor surface
4. **Material Animation**:
    - Scrolls the conveyor's texture based on speed and direction

- Synchronizes visual feedback with physical movement

5. **Visual Debugging**:
    - Provides visual gizmos showing direction and boundaries
    - Includes helpful debugging information during development

The script is designed to handle various object types and behaviors realistically while using Unity's physics system.

3. PhysicsConveyorBeltWithEdges.cs

Main Algorithm:

This enhanced version of the conveyor belt adds edge detection for realistic object behavior:

1. **Edge Detection System**:
    - Calculates object position relative to the conveyor
    - Identifies when objects approach the end of the belt
    - Uses a configurable distance threshold to determine the "edge zone"
2. **Edge Handling Behavior**:
    - Stops applying conveyor forces when objects reach the edge
    - Optionally applies a small push force to help objects fall naturally
    - Creates a realistic transition from belt to free-fall
3. **Relative Position Calculation**:
    - Transforms object positions to the conveyor's local space
    - Uses directional logic to handle both forward and backward movement
    - Evaluates the z-position relative to conveyor length
4. **Enhanced Visual Debugging**:
    - Adds red visualization for edge detection zones
    - Clearly shows the active areas of the conveyor
    - Helps developers fine-tune the edge detection parameters

The core conveyor movement algorithm remains the same as the base version, but with this additional edge awareness to create more realistic behavior at the conveyor ends.

5. ColorGenerator.cs

Main Algorithm:

The ColorGenerator script provides various methods for generating colors in Unity with specific characteristics and intelligent handling of problematic colors:

1. **Random Color Generation with Dark Color Handling**:
    - Offers different approaches to generate random colors with varying levels of control
    - Implements a dark color detection system with a brightness threshold (0.2)
    - Maintains a historical record of problematic dark colors to inform future color selections
2. **Multiple Generation Strategies**:
    - `GenerateFullRandomColor()`: Creates entirely random colors with no constraints
    - `GenerateControlledRandomColor()`: Produces random colors with parameters tuned for better visibility
    - `GeneratePaletteColor()`: Selects from a predefined array of common colors
    - `GeneratePastelColor()`: Creates soft, high-brightness, low-saturation colors
    - `GenerateColorWithAlpha()`: Generates colors with transparency
    - `GenerateColorInRange()`: Creates colors within a specific hue range
    - `GenerateComplementaryColor()`: Produces colors opposite to a base color on the color wheel
3. **Dark Color Learning System**:
    - Tracks colors that were too dark in a history list (limited to 20 entries)
    - Uses color similarity detection to identify and adjust colors similar to previously problematic ones
    - Applies HSV (Hue, Saturation, Value) color space analysis to evaluate color brightness
4. **Brightness Enhancement Logic**:
    - When dark colors are detected, either returns pure black (for truly dark colors)
    - Or increases brightness for colors similar to previously dark ones
    - Color similarity is determined by comparing hue and saturation values

This algorithm provides a robust system for generating visually appropriate colors for different use cases while automatically handling edge cases like excessively dark colors that might be difficult to see.