Project Report: Live Cam Detection of Red and Blue Objects

Group Members

Abdul Rafay

21-NTU-CS-1197

Arooba Zaman

21-NTU-CS-1211

Department of Computer Science

National Textile University Faisalabad **15-05-2024**

Introduction

This documentation provides a comprehensive overview of a digital image processing project implemented in MATLAB. The project involves developing a program to detect blue and red objects within an image or live webcam feed. The program employs various image processing techniques such as thresholding, morphological operations, and connected component analysis to achieve accurate object detection.

Purpose

The purpose of this project is to demonstrate the practical application of digital image processing techniques in real-world scenarios. By detecting blue and red objects within images or video feeds, the program can be useful in applications such as quality control in manufacturing, automated sorting, and real-time monitoring.

Project Components:

1. Image Selection Function:

- Allows users to select an image file (jpg, png, jpeg) for color detection.
- Processes the selected image using color detection algorithms.
- Displays the original image and segmented areas corresponding to detected colors.

2. Webcam Capture Function:

- Captures a single image from a connected webcam for real-time color detection.
- Applies color detection algorithms to the captured image.
- Displays the live video feed along with segmented areas of detected colors.

Methodologies:

Image Selection Function (Select_ImageButtonPushed):

1. Image Loading:

- Prompts the user to select an image file.

- Reads the selected image and converts it to the HSV color space.

2. **Color Detection:**
    - Defines thresholds for blue and red colors based on chosen histogram values.
    - Creates masks for blue and red colors using the defined thresholds.
    - Applies morphological operations to enhance color segmentation.
    - Counts the number of objects detected for each color.
3. **Visualization:**
    - Displays the original image and segmented areas of detected blue and red colors.
    - Overlays bounding boxes around detected objects for visual identification.

Webcam Capture Function (Real-time Color Detection):

1. **Image Acquisition:**
    a. Creates a webcam object and captures a single image from the webcam.
    b. Converts the captured image to the HSV color space.
2. **Color Detection:**
    a. Defines thresholds for blue and red colors similar to the image selection function.
    b. Creates masks for blue and red colors and applies morphological operations.
3. **Object Counting and Visualization:**
    a. Counts the number of objects detected for blue and red colors.
    b. Displays the original image, segmented areas of blue and red colors, and bounding boxes around detected objects.

Usage:

- Users can choose between image selection mode and real-time webcam capture mode.
- The application provides visual feedback on the detected colors and the number of objects detected.
- Users can interact with the graphical user interface (GUI) to select images, view results, and adjust parameters if needed.

Code Implementation

Image Selection:

```
% Step 1: Prompt user to select an image file
[fileName, pathName] = uigetfile({'*.jpg;*.png;*.jpeg', 'Image files'}, 'Select Image');

% Step 2: Check if a file was selected
if isequal(fileName, 0)
    return; % No file selected, do nothing
end

% Step 3: Read the selected image
imagePath = fullfile(pathName, fileName);
image = im2double(imread(imagePath));

% Step 4: Convert RGB image to HSV color space
I = rgb2hsv(image);

% Step 5: Define thresholds for blue color
blueChannel1Min = 0.544;
blueChannel1Max = 0.677;
blueChannel2Min = 0.513;
blueChannel2Max = 1.000;
blueChannel3Min = 0.000;
blueChannel3Max = 1.000;
```

```
% Step 6: Create blue mask based on chosen histogram thresholds
blueMask = (I(:,:,1) >= blueChannel1Min) & (I(:,:,1) <= blueChannel1Max) & ...
        (I(:,:,2) >= blueChannel2Min) & (I(:,:,2) <= blueChannel2Max) & ...
        (I(:,:,3) >= blueChannel3Min) & (I(:,:,3) <= blueChannel3Max);

% Step 7: Define thresholds for red color
redChannel1Min = 0.954;
redChannel1Max = 0.009;
redChannel2Min = 0.709;
redChannel2Max = 1.000;
redChannel3Min = 0.000;
redChannel3Max = 1.000;

% Step 8: Create red mask based on chosen histogram thresholds
redMask = ((I(:,:,1) >= redChannel1Min) | (I(:,:,1) <= redChannel1Max)) & ...
        (I(:,:,2) >= redChannel2Min) & (I(:,:,2) <= redChannel2Max) & ...
        (I(:,:,3) >= redChannel3Min) & (I(:,:,3) <= redChannel3Max);

% Step 9: Apply morphological operations
se = strel('disk', 4);
afterOpeningBlue = imopen(blueMask, se);
afterClosingBlue = imclose(afterOpeningBlue, se);
afterOpeningRed = imopen(redMask, se);
afterClosingRed = imclose(afterOpeningRed, se);

% Step 10: Object Counting
```

```
[~, blueObjectsCount] = bwlabel(afterClosingBlue);
[~, redObjectsCount] = bwlabel(afterClosingRed);

% Step 11: Display results
imshow(image, 'Parent', app.UIAxes);
imshow(afterClosingBlue, 'Parent', app.UIAxes_2);
imshow(afterClosingRed, 'Parent', app.UIAxes2);
imshow(image, 'Parent', app.UIAxes_3);
hold(app.UIAxes_3, 'on');

% Step 12: Display bounding boxes around detected blue objects
[Bblue, Lblue] = bwboundaries(afterClosingBlue, 'noholes');
for k = 1:length(Bblue)
   boundary = Bblue{k};
   plot(app.UIAxes_3, boundary(:,2), boundary(:,1), 'b', 'LineWidth', 2);
end

% Step 13: Display bounding boxes around detected red objects
[Bred, Lred] = bwboundaries(afterClosingRed, 'noholes');
for k = 1:length(Bred)
   boundary = Bred{k};
   plot(app.UIAxes_3, boundary(:,2), boundary(:,1), 'r', 'LineWidth', 2);
end
hold(app.UIAxes_3, 'off');

% Step 14: Debugging - Display counts
disp(['blueObjectsCount: ', num2str(blueObjectsCount)]);
```

```
disp(['redObjectsCount: ', num2str(redObjectsCount)]);

% Step 15: Ensure the counts are scalars
if ~isscalar(blueObjectsCount)
    error('blueObjectsCount is not a scalar.');
end

if ~isscalar(redObjectsCount)
    error('redObjectsCount is not a scalar.');
end

% Step 16: Convert counts to strings
blueCountStr = num2str(blueObjectsCount);
redCountStr = num2str(redObjectsCount);

% Step 17: Display counts in the GUI
app.BlueObjectLabel.Text = ['', blueCountStr];
app.RedObjectLabel.Text = ['', redCountStr];
```

Webcam Capture:

```
% Step 1: Create webcam object
cam = webcam;

% Step 2: Capture single image from webcam
image = snapshot(cam);

% Step 3: Convert RGB image to HSV color space
```

```
I = rgb2hsv(image);

% Step 4: Define thresholds for blue color
blueChannel1Min = 0.544;
blueChannel1Max = 0.677;
blueChannel2Min = 0.513;
blueChannel2Max = 1.000;
blueChannel3Min = 0.000;
blueChannel3Max = 1.000;

% Step 5: Create blue mask based on chosen histogram thresholds
blueMask = (I(:,:,1) >= blueChannel1Min) & (I(:,:,1) <= blueChannel1Max) & ...
        (I(:,:,2) >= blueChannel2Min) & (I(:,:,2) <= blueChannel2Max) & ...
        (I(:,:,3) >= blueChannel3Min) & (I(:,:,3) <= blueChannel3Max);

% Step 6: Define thresholds for red color
redChannel1Min = 0.954;
redChannel1Max = 0.009;
redChannel2Min = 0.709;
redChannel2Max = 1.000;
redChannel3Min = 0.000;
redChannel3Max = 1.000;

% Step 7: Create red mask based on chosen histogram thresholds
redMask = ((I(:,:,1) >= redChannel1Min) | (I(:,:,1) <= redChannel1Max)) & ...
        (I(:,:,2) >= redChannel2Min) & (I(:,:,2) <= redChannel2Max) & ...
        (I(:,:,3) >= redChannel3Min) & (I(:,:,3) <= redChannel3Max);
```

```
% Step 8: Apply morphological operations
se = strel('disk', 4);
afterOpeningBlue = imopen(blueMask, se);
afterClosingBlue = imclose(afterOpeningBlue, se);
afterOpeningRed = imopen(redMask, se);
afterClosingRed = imclose(afterOpeningRed, se);

% Step 9: Remove small objects
minObjectSize = 100; % Adjust this value based on the size of the objects you want to keep
filteredBlueMask = bwareaopen(afterClosingBlue, minObjectSize);
filteredRedMask = bwareaopen(afterClosingRed, minObjectSize);

% Step 10: Object Counting
[~, blueObjectsCount] = bwlabel(filteredBlueMask);
[~, redObjectsCount] = bwlabel(filteredRedMask);

% Step 11: Display results
figure;

subplot(2, 2, 1);
imshow(image);
title('Original Image');

subplot(2, 2, 2);
imshow(filteredBlueMask);
title('Blue Area');
```

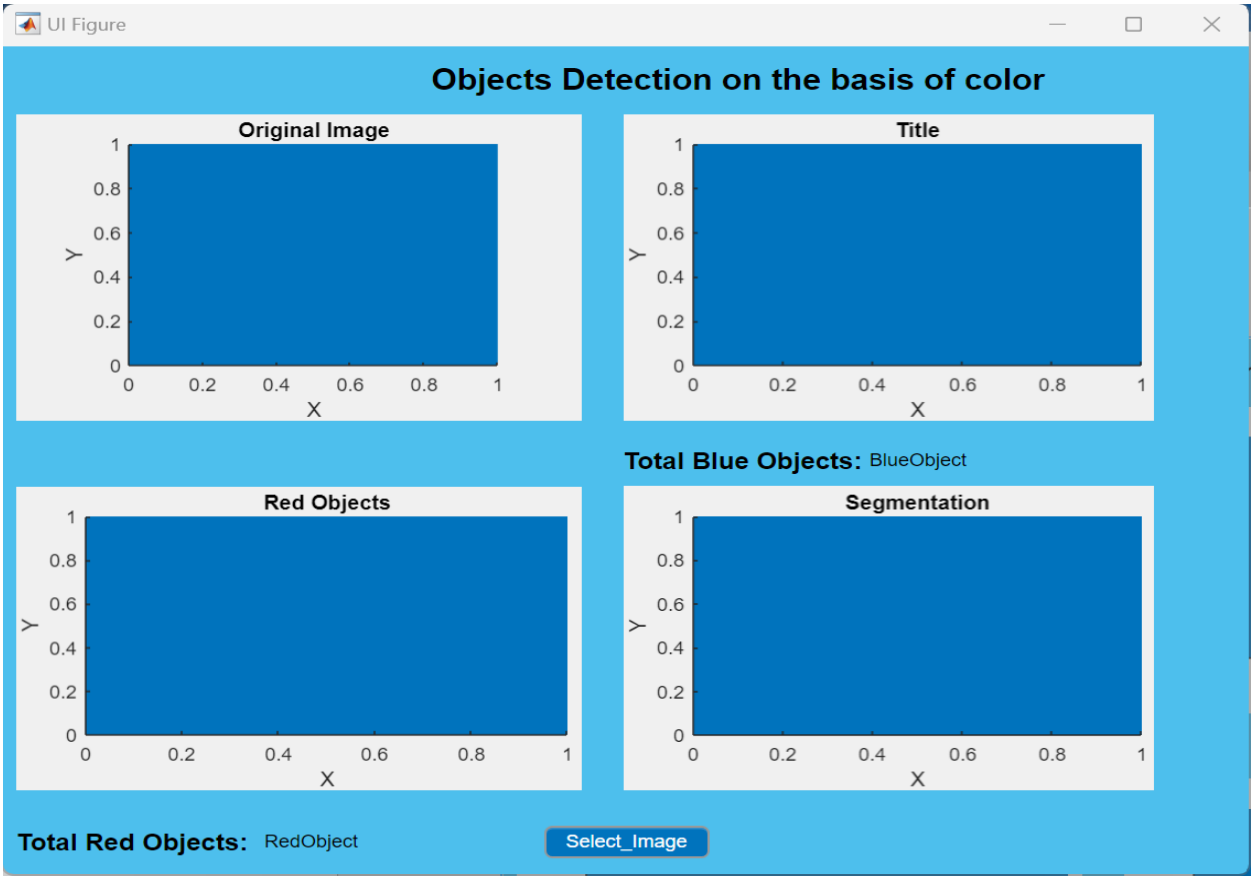
```
subplot(2, 2, 3);
imshow(filteredRedMask);
title('Red Area');

% Step 12: Display bounding boxes around detected blue objects
[Bblue, ~] = bwboundaries(filteredBlueMask, 'noholes');
subplot(2, 2, 1);
hold on;
for k = 1:length(Bblue)
   boundary = Bblue{k};
   plot(boundary(:,2), boundary(:,1), 'b', 'LineWidth', 2);
end
hold off;

% Step 13: Display bounding boxes around detected red objects
[Bred, ~] = bwboundaries(filteredRedMask, 'noholes');
subplot(2, 2, 1);
hold on;
for k = 1:length(Bred)
   boundary = Bred{k};
   plot(boundary(:,2), boundary(:,1), 'r', 'LineWidth', 2);
end
hold off;

% Step 14: Close webcam object when done
clear cam;
```

App Design



Webcam Output

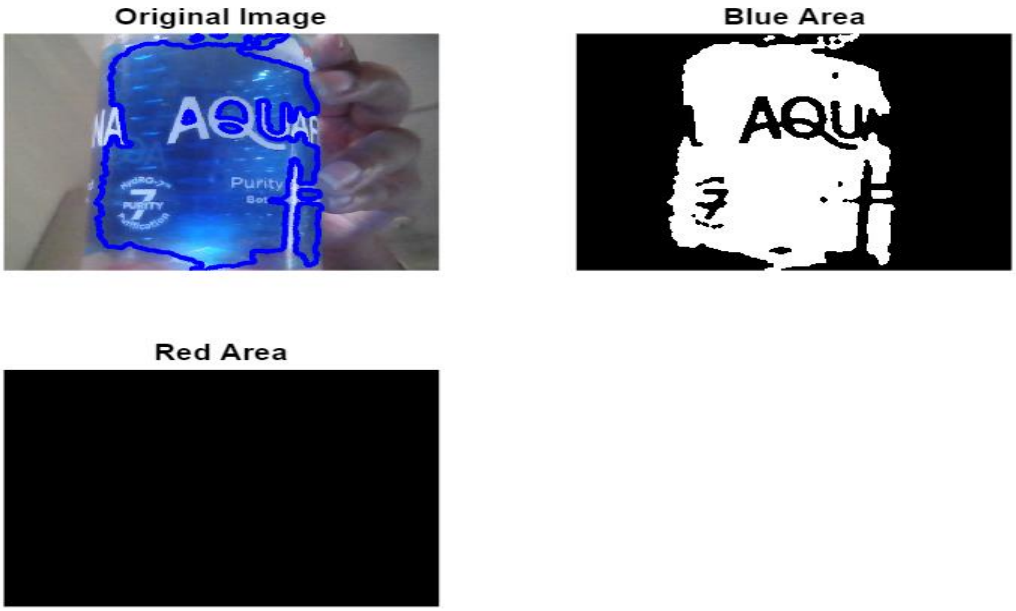
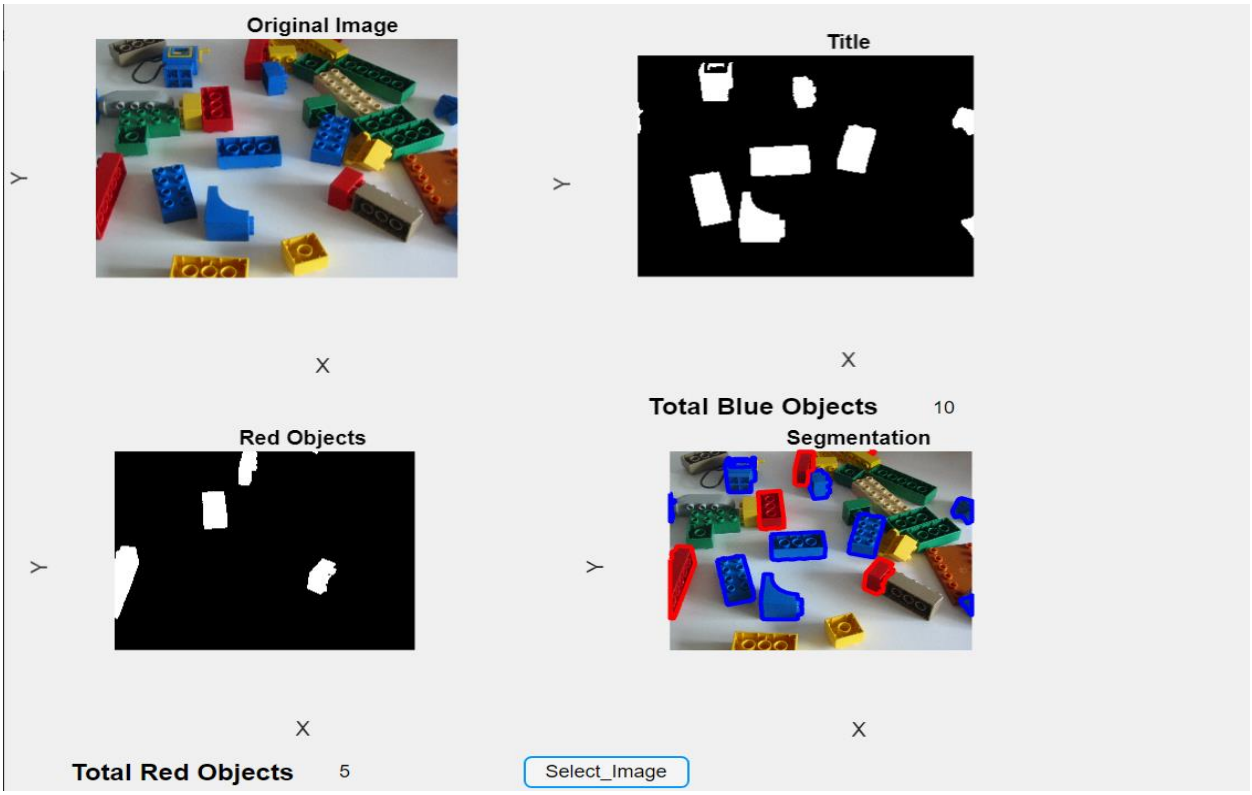
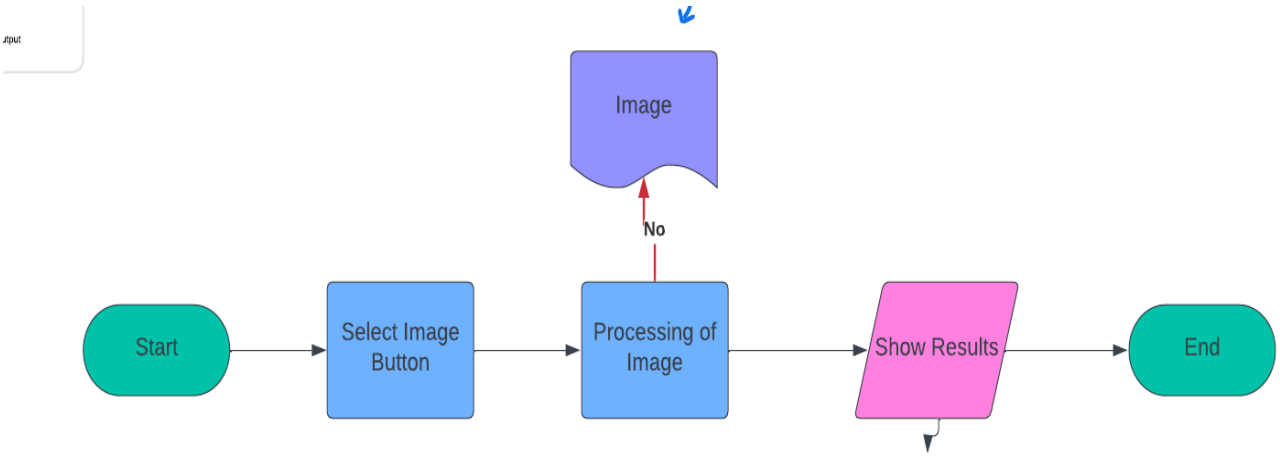


Image Selection Output



Flow Chart



Applications:

The project finds applications across various domains where object detection and counting play a crucial role:

1. **Manufacturing Industry**: It can be integrated into quality control processes to inspect products on assembly lines, ensuring accurate assembly and detecting defects.
2. **Biomedical Imaging**: In medical diagnostics, it can assist in cell counting and identifying specific structures in biological samples, aiding in disease diagnosis and research.
3. **Environmental Monitoring**: It can be utilized for analyzing environmental samples such as water or soil, helping in tasks like counting particles or organisms for environmental studies and pollution monitoring.
4. **Traffic Management**: In transportation systems, it can be employed for traffic monitoring and management by counting vehicles of different colors, supporting traffic flow analysis and planning.
5. **Agriculture**: It finds applications in precision agriculture for monitoring crop health, counting fruits or flowers, and detecting pests or diseases, contributing to optimized farming practices.
6. **Security and Surveillance**: It can enhance security systems by detecting and tracking specific objects or individuals of interest in surveillance footage, improving threat detection and response.

These applications demonstrate the versatility and practical utility of the project in addressing various real-world challenges across different industries and sectors.

Conclusion:

In conclusion, the MATLAB application successfully detects and counts blue and red objects in images, offering valuable quantitative information for various applications. By leveraging color segmentation and morphological operations, it provides an efficient solution for object analysis.