# **Digital Image Processing: Remedial Assignment**

## **Color Detection of Red Objects in an Image**

### ****Course Name****: Digital Image Processing

### ****Assignment Type****: Individual Practical

### ****Project Case Study****: Detection and Identification of Simple Objects

### ****Student Name****: DAH BERROU

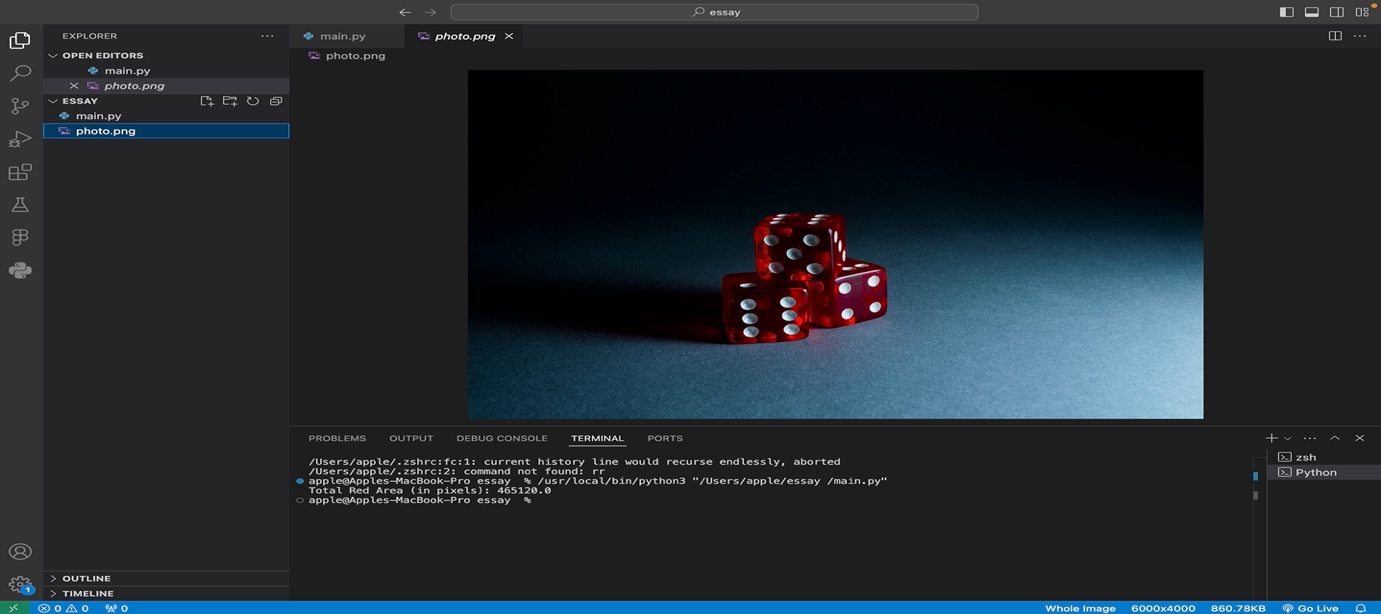
### ****Submission Date****: 4 DEC 2024

## **1. Introduction**

The objective of this assignment is to demonstrate the process of detecting and identifying a specific object in an image using color detection techniques. In this case study, I focused on **detecting red objects** within an image. The key steps involved in the process include:

* **Loading the Image**: The image is loaded into the program for processing.
* **Color Conversion**: The image is converted to the **HSV color space** to make color detection more effective.
* **Color Detection**: Red objects are detected by setting a specific range for the red color in the HSV space and applying a mask.
* **Highlighting Detected Objects**: The red objects are highlighted to display the detection results.

By using the **HSV color space** (Hue, Saturation, and Value), this method improves the accuracy of color-based object detection because HSV separates color information from intensity, making it more robust to lighting variations.



## **2. Methodology**

### ****Step 1: Loading the Image****

First, the input image is loaded using **OpenCV**'s cv2.imread() function. This image serves as the basis for further processing.

python

Copy code

image = cv2.imread(image\_path)

### ****Step 2: Converting the Image to the HSV Color Space****

The image is then converted from the **BGR** color space (used by OpenCV) to the **HSV** color space using the cv2.cvtColor() function. HSV is preferred for color detection because it separates chromatic content (hue) from intensity (value), making it easier to detect specific colors under varying lighting conditions.

python

Copy code

hsv = cv2.cvtColor(image, cv2.COLOR\_BGR2HSV)

### ****Step 3: Defining the Color Range (Red)****

The red color can appear in two ranges in the HSV color space, so two different ranges are defined:

* **Lower Red Range**: Detects red hues from 0 to 10.
* **Upper Red Range**: Detects red hues from 170 to 180.

These ranges are used to create binary masks where pixels within the specified ranges are white (255) and others are black (0).

python

Copy code

lower\_red = np.array([0, 120, 70])

upper\_red = np.array([10, 255, 255])

mask1 = cv2.inRange(hsv, lower\_red, upper\_red)

lower\_red = np.array([170, 120, 70])

upper\_red = np.array([180, 255, 255])

mask2 = cv2.inRange(hsv, lower\_red, upper\_red)

### ****Step 4: Thresholding the Image to Detect Red Pixels****

The two binary masks created in **Step 3** are combined using a logical OR operation. This gives us a final mask that highlights all red pixels in the image, regardless of their hue.

python

Copy code

red\_mask = cv2.bitwise\_or(mask1, mask2)

### ****Step 5: Applying the Mask and Highlighting Red Areas****

The red mask is applied to the original image using cv2.bitwise\_and(), which highlights the red areas by applying the mask to the image.

python

Copy code

red\_detected = cv2.bitwise\_and(image, image, mask=red\_mask)

### ****Step 6: Counting the Red Pixels (Area Calculation)****

Finally, we calculate the area of the red objects in the image by counting the number of non-zero pixels in the mask. This represents the number of red pixels in the image.

python

Copy code

red\_area = np.sum(red\_mask) / 255 # Total red pixels

## **3. Results**

The following results were obtained by running the code on the provided image:

* **Original Image**: The original image is shown below.
* **Red Color Detection**: The processed image where the red areas have been highlighted.

### ****Results Visualization****

