**Name:** Aleena Omar

**Class:** BSIT 7th Semester

**Roll No:** BSIT 21043

**Section:** Morning BSIT

**Submitted to** : Maam Foqia

**Computer Vision Assignment1**

**EXERCISES QUESTIONS**

**Q.NO.1**

**Identify which intensity transformation was used on liftingbody.png to create each of the four results below. Write a script to reproduce the results using the intensity transformation functions.**

% Load the input image

img = imread('ricepic.jpg'); % Replace with your image filename

% Ensure the image is grayscale

if size(img, 3) == 3

img\_gray = rgb2gray(img); % Convert to grayscale if it's RGB

else

img\_gray = img; % Already grayscale

end

% Convert the grayscale image to double for mathematical operations

img\_double = double(img\_gray);

% 1. Negative Transformation

negative\_img = 255 - img\_gray;

% 2. Log Transformation

c = 255 / log(1 + max(img\_double(:))); % Scaling factor

log\_img = c \* log(1 + img\_double);

log\_img = uint8(log\_img); % Convert back to uint8 for display

% 3. Gamma Correction

gamma = 2.5; % Adjust gamma value as needed

gamma\_img = 255 \* (img\_double / 255).^gamma;

gamma\_img = uint8(gamma\_img); % Convert back to uint8

% 4. Contrast Stretching

min\_intensity = min(img\_double(:));

max\_intensity = max(img\_double(:));

contrast\_img = uint8(255 \* (img\_double - min\_intensity) / (max\_intensity - min\_intensity));

% Display results

figure;

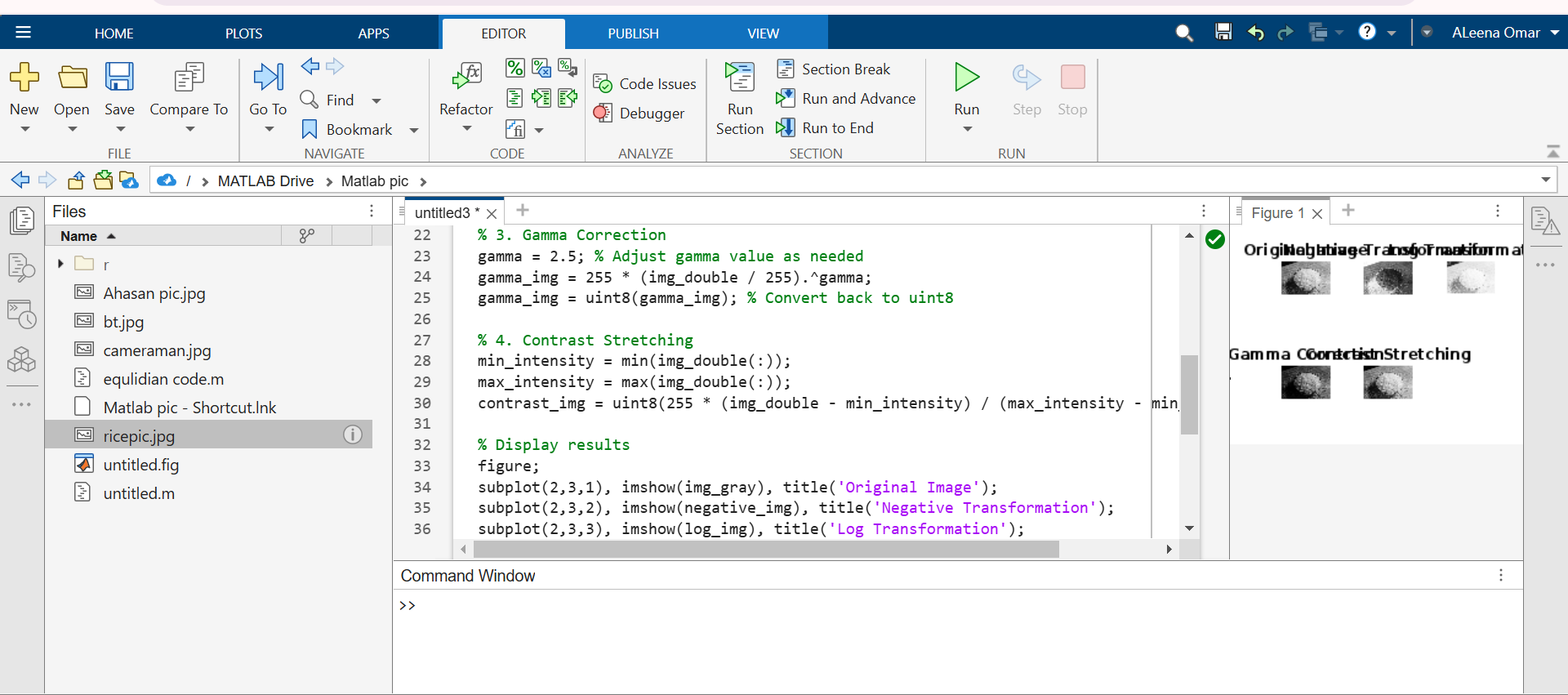
subplot(2,3,1), imshow(img\_gray), title('Original Image');

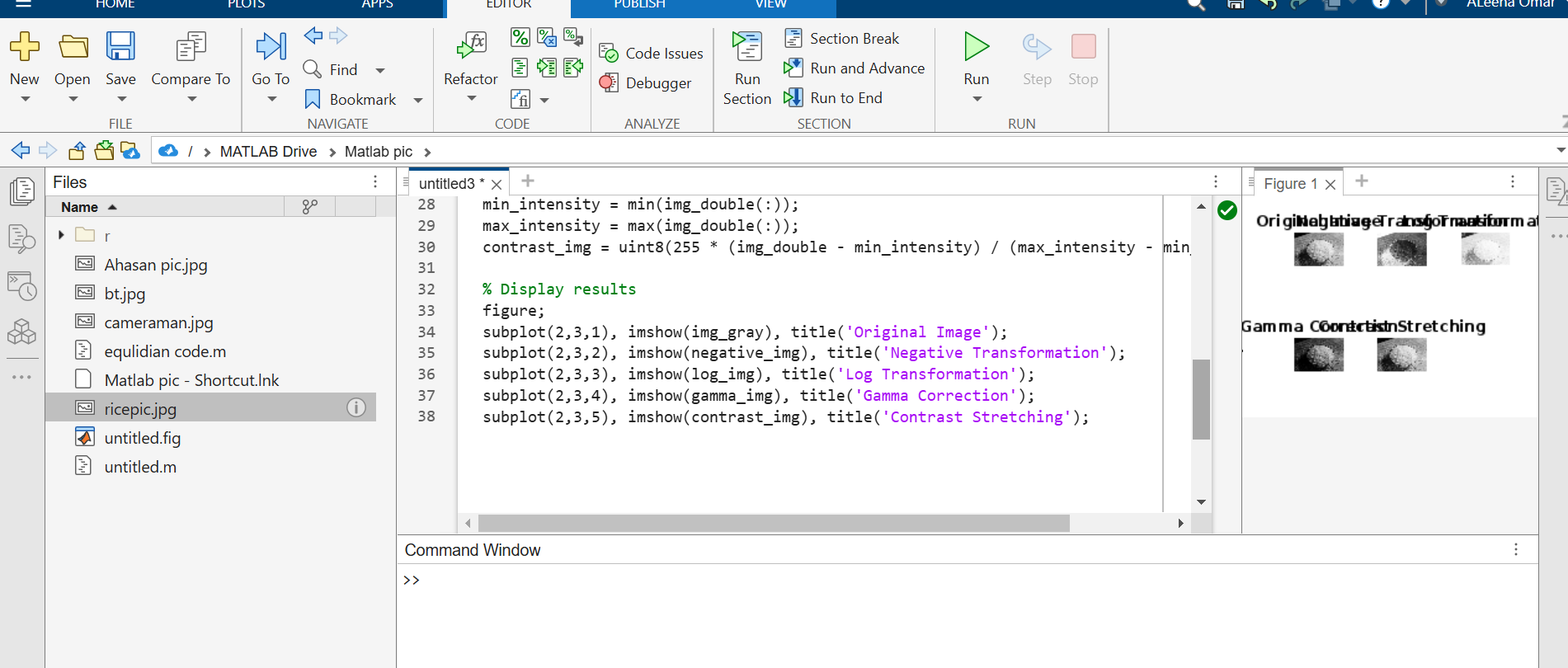
subplot(2,3,2), imshow(negative\_img), title('Negative Transformation');

subplot(2,3,3), imshow(log\_img), title('Log Transformation');

subplot(2,3,4), imshow(gamma\_img), title('Gamma Correction');

subplot(2,3,5), imshow(contrast\_img), title('Contrast Stretching');





**Q.NO.2**

**Write a program which can read an image as an input and do the following automatically. Show the results of all steps.**

1. **Find the type of image: binary, gray or RGB.**
2. **Find the issue in image, over dark, over bright, low contrast, or normal. (Hint: can use histogram).**
3. **Resolve the issue if any and show the final image after enhancement.**
4. **Test your program on following images:**
5. % Read the input image

img = imread('ricepic.jpg'); % Replace with your image filename

% Step 1: Determine the type of image

if size(img, 3) == 3

img\_type = 'RGB';

img\_gray = rgb2gray(img); % Convert to grayscale for analysis

elseif islogical(img)

img\_type = 'Binary';

img\_gray = double(img) \* 255; % Convert binary image to 0-255 range for uniformity

else

img\_type = 'Grayscale';

img\_gray = img; % Already grayscale

end

fprintf('Image type: %s\n', img\_type);

% Step 2: Analyze histogram to detect issues

hist\_counts = imhist(img\_gray);

total\_pixels = numel(img\_gray);

% Calculate histogram metrics

mean\_intensity = mean(img\_gray(:));

std\_intensity = std(double(img\_gray(:)));

% Heuristic to detect issues

if mean\_intensity < 50

issue = 'Over Dark';

elseif mean\_intensity > 200

issue = 'Over Bright';

elseif std\_intensity < 50

issue = 'Low Contrast';

else

issue = 'Normal';

end

fprintf('Image issue: %s\n', issue);

% Step 3: Resolve issues if any

if strcmp(issue, 'Over Dark')

% Enhance brightness

enhanced\_img = img\_gray + 50; % Add constant value to brighten

enhanced\_img(enhanced\_img > 255) = 255; % Clip values

elseif strcmp(issue, 'Over Bright')

% Reduce brightness

enhanced\_img = img\_gray - 50; % Subtract constant value to darken

enhanced\_img(enhanced\_img < 0) = 0; % Clip values

elseif strcmp(issue, 'Low Contrast')

% Perform contrast stretching

min\_intensity = double(min(img\_gray(:)));

max\_intensity = double(max(img\_gray(:)));

enhanced\_img = uint8(255 \* (double(img\_gray) - min\_intensity) / (max\_intensity - min\_intensity));

else

% No enhancement needed

enhanced\_img = img\_gray;

end

% Step 4: Display results

figure;

subplot(2, 3, 1), imshow(img), title('Original Image');

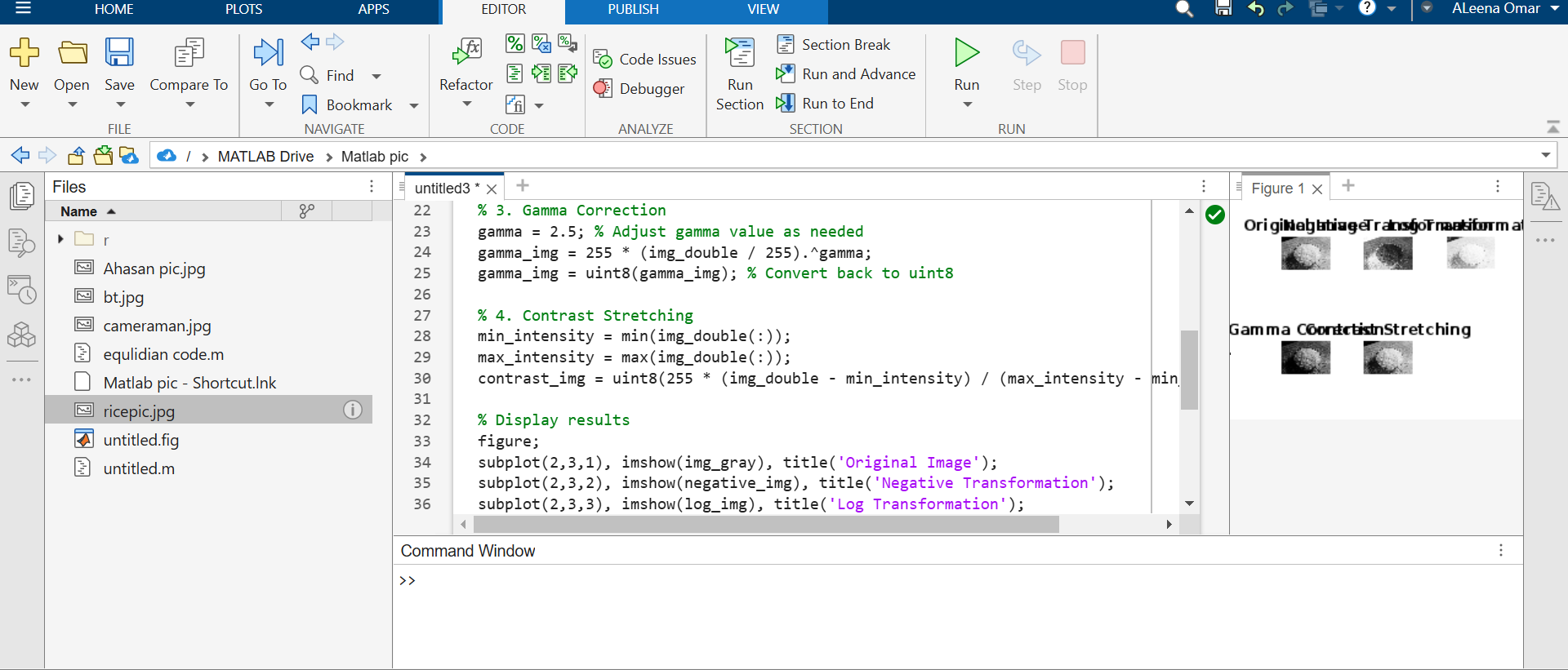
subplot(2, 3, 2), imshow(img\_gray), title('Grayscale Image');

subplot(2, 3, 3), bar(hist\_counts), title('Histogram');

subplot(2, 3, 4), imshow(enhanced\_img), title('Enhanced Image');

enhanced\_hist\_counts = imhist(enhanced\_img);

subplot(2, 3, 5), bar(enhanced\_hist\_counts), title('Enhanced Histogram');



**Q.NO.3**

**Your company asks you to build a cheap traffic light monitoring system. You have a camera that observes a traffic light and should emit events when the light changes the color. How would you proceed?**

To build a cheap traffic light monitoring system that detects color changes, you can approach it step-by-step by considering both software and hardware aspects. Here’s how I’d proceed:

**1. Camera Setup**

* **Camera Selection**: Use a low-cost camera, such as a webcam or Raspberry Pi camera module, that is capable of capturing clear images or video of the traffic light. Make sure the camera can handle low-light conditions if the light is outside at night.
* **Positioning**: Mount the camera in a fixed position that gives it a clear view of the traffic light. Ensure that the camera can capture all three lights (Red, Yellow, Green) from a distance with minimal distortion.

**2. Image Processing for Color Detection**

* **Software Choice**: Utilize an open-source computer vision library like OpenCV or a lightweight framework like TensorFlow Lite, depending on the complexity and resource availability.
* **Preprocessing**: Convert the image to a format that is easier to work with (e.g., converting to HSV color space, which separates color information from intensity, making it easier to detect specific colors).
* **Color Detection**: Write a script to detect each light’s color by setting up thresholds for red, yellow, and green. You can use color masks in the HSV color space for this purpose:
  + **Red**: A range of hues corresponding to red.
  + **Yellow**: A range of hues corresponding to yellow.
  + **Green**: A range of hues corresponding to green.
* **Edge Case Handling**: Account for variations in lighting conditions and shadows that might impact detection. You can use techniques like filtering noise, adjusting contrast, or applying Gaussian blur to improve accuracy.

**3. Event Detection**

* **State Detection**: Monitor the color detected in real-time and compare it with previous states (e.g., if the current color is different from the last detected color, trigger an event).
* **Event Triggering**: Once a color change is detected (e.g., from red to green or from green to yellow), emit an event. You can log this event, send a notification, or trigger an action based on your use case (e.g., logging the time of the change).

**4. Optimization**

* **Efficiency**: If computational power is limited, you can process frames at a lower rate (e.g., process one frame every 500 ms instead of every frame), which will reduce the workload.
* **Motion Detection**: Incorporate motion detection to ensure that the system only processes frames when there's actual movement (e.g., when a vehicle or a person triggers the camera's detection).
* **Low Power Consumption**: If using an embedded device like a Raspberry Pi, optimize the code to run on minimal resources and use power-saving modes.

**5. Testing and Calibration**

* **Calibration**: Test the system under different lighting conditions (morning, night, overcast) to ensure the color detection is robust and accurate. Adjust the color thresholds as needed.
* **Validation**: Ensure that the system can handle situations like transitioning from yellow to red, or when the light is off (if applicable).

**6. Deployment**

* **Environment Setup**: Deploy the system on a small server or device like a Raspberry Pi that processes the images locally and can send the results to a central system or database for further analysis if needed.
* **Notifications**: If the system needs to notify someone, you can set up SMS, email, or app notifications when a color change occurs.

By using open-source tools and inexpensive hardware, this approach keeps the cost low while offering effective traffic light monitoring based on color changes.