**A Project Report**

On

# IMAGE PROCESSING USING OPENCV

**BY**

**YASH GARG**

**2022A3PS1470H**

**Under the supervision of**

PROF. PONNALAGU R.N.

**SUBMITTED IN FULLFILLMENT OF THE REQUIREMENTS OF**

**EEE F266: STUDY PROJECT**

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)**

**HYDERABAD CAMPUS**

**(APRIL 2025)**

**ACKNOWLEDGMENTS**

I would like to sincerely thank Prof. Ponnalagu RN and Prof. Anakhi Hazarika for their constant support, patience, guidance and giving me the opportunity to work on this project.. I would like to express my gratitude towards my fellow peers who worked on this project with me.

**Birla Institute of Technology and Science-Pilani,**

**Hyderabad Campus**

## Certificate

This is to certify that the project report entitled **"IMAGE PROCESSING USING OPENCV"** submitted by MR. YASH GARG (ID No. 2022A3PS1470H) in fulfillment of the requirements of the course EEE F266, Study Project Course, embodies the work done by him under my supervision and guidance.

**Date:** **(PROF. PONNALAGU R.N.)**

**BITS- Pilani, Hyderabad Campus**

## ABSTRACT

For the second phase of the project, we have concentrated on integrating a variety of classical image processing techniques available in the OpenCV library to analyze a dataset that contained image for potholes, roads, and speedbreakers. Rather than employing object recognition models based on machine learning or deep learning, we went for a non-learning approach, which involved progressing through a series of procedures to analyze and extract features from the images. The aim was to identify the distinguishing features, such as edges, contours, and other characteristics, that would help in assessing these constituents based on classical computer vision methods. The methodology employed grayscale conversion, Gaussian blur, adaptive Canny edge detection, morphological operations, and using morphology and hierarchy for filtering contours and data across the whole pipeline. By doing so, we sought to separate and demonstrate the possible locations for potholes and speedbreakers in the images, thereby demonstrating that it is possible to detect objects with proper image pre-processing and engineering steps without having to train models and invest time in data accumulation and machine learning.

**CONTENTS**

## I) Introduction:

The following report discusses the methodology and systematic approach followed to pre-process a dataset of road surface images to detect potholes, speed breakers and other structural anomalies. The dataset comprises scenarios of real-world roads where the said anomalies are present under contrasting light, texture, and environmental conditions. The purpose was to develop a pre-processing pipeline capable of highlighting primarily potholes and speed breakers in the images using classical image processing techniques available in OpenCV. Each image was first transformed from BGR color space to HSV color space, then followed by noise reduction and edge detection using various OpenCV functions. The resultant images are then processed to locate and highlight major regions of interest in the images using bounding boxes in green color which could be used as an input to develop further systems such as road maintenance bots, traffic safety bots, etc.

## II) Pre-processing:

a) Grayscaling:

- The first step in the preprocessing pipeline is changing the input color image (typically represented as a BGR image in the OpenCV library) into a grayscale image. This is necessary as it transforms the image into an intensity image, which simplifies it. Indeed, a color image has three color channels; that is, it presents the intensity of colors in all three RGB channels. However, the processing of most standard image processing operations such as edge detection, thresholding, and morphological operations is based on the intensity gradient of the image rather than color information.

Reducing the images to grey scale reduces the computational complexity and processing time of computer algorithms, particularly when dealing with a large set of images or performing real-time operations. In addition, it improves the effectiveness of subsequent operations by removing color noise and emphasizing information on pixel intensity changes. These changes are critical for detecting the edges, contours, and other structures of the objects seen in the image, such as potholes, speed breakers, etc. When starting with grey scale, the search for object identification in road scenes is only focused on the visual contrast and shape information only.

```
# 1. Enhanced Preprocessing
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

b) Dynamic Gaussian Blur based on image size:

Blurring is an essential pre-processing step to get rid of the noise and small variations in the pixel values, which can give false or break detections. In this project, Gaussian Blur has been used which is a common smoothing technique in which the pixel value is averaged with its neighbors and weighed according to the Gaussian distribution to ensure that the nearer pixels have a greater influence than those further away, preserving the general image structure and reducing noise.

A dynamic approach to Gaussian blurring was introduced by varying the kernel size to optimize the balance between noise removal and edge retention. The kernel sizes used during the cross-validation were 3×3, 7×7, 9×9, and 11×11. After comparison, a 5×5 kernel provided the best results for the detection of edge details, while eliminating minor noise. The size of the image is analyzed to ensure that the size of the kernel is appropriate for the size of the object. Subsequently, the image is blurred.

`

```
# Dynamic Gaussian Blur based on image size
img_h, img_w = gray.shape
kernel_size = 5
blur = cv2.GaussianBlur(gray, (kernel_size, kernel_size), 2)
```

c) Canny Edge Detection with adaptive thresholds:

The Canny edge detection algorithm is one of the most effective tools for this purpose. However, its performance heavily depends on the choice of threshold values. To make the edge detection robust across a diverse set of images with varying lighting and contrast, we implemented an adaptive thresholding approach.

It all starts with determining the median pixel intensity of a blurred grayscale image. The median is a stable indicator of overall lightness and helps establish context-sensitive thresholds. Based on it, a lower bound (commonly 0.66 × median) and an upper bound (commonly 1.33 × median) are calculated and then passed to the Canny algorithm. This approach allows for reliable edge detection in various images regardless of their brightness or shadows.

The produced edge map can accurately give the edge of structures such as pothole, and speed breakers, with minimal noise and other unnecessary features. These detected edges form the basis for other processes such as morphological transformation and contour analysis making this process important for accurate feature detection.

```
# 2. Canny Edge Detection with adaptive thresholds
median = np.median(blur)
lower = int(max(0, 0.6 * median))
upper = int(min(255, 1.4 * median))
edges = cv2.Canny(blur, lower, upper)
```

d) Dynamic Parameter Adjustment:

After edge detection, many contours may be detected—some meaningful, others irrelevant. To reduce noise and false positives, we implement area-based contour filtering, where contours are retained only if their areas fall within a specific range. This step is crucial to exclude small fragments caused by noise and overly large regions that may span unrelated structures.

Rather than using fixed thresholds, we calculate them dynamically based on the image's total area. A lower bound (e.g., 0.0005 × image area) helps eliminate minor irregularities, while an upper bound (e.g., 0.25 × image area) filters out excessively large shapes. This dynamic scaling ensures that the filtering remains effective regardless of the image resolution or scale of objects.

By adapting to image size, this approach makes the pipeline resolution-independent, ensuring consistent detection accuracy across datasets with varying dimensions and content complexity.

```
# 4. Dynamic Parameter Adjustment
img_area = img_h * img_w
min_area = (img_area // 150) #* scale_factor  # Adjusted baseline
max_area = img_area // 3
```

e) Contour processing with hierarchy analysis

Contours are a very useful tool in the field of image treatment. They are curves joined all the way along a boundary which share the same color or intensity. The **hierarchy information** is particularly important in contours, as it provides a bit of additional structure to the information and stores information of parent and child relationship between contours. With this information, we can distinguish between nested objects.For instance, an object is inside another, the outer one is the parent and the inner is the child.

In OpenCv, the `findContours` function is used in order to get contours of a binary image in which the pixels are in two colors, black and white. This function does not only find the contours but also returns the **hierarchy**. The hierarchy can be used for extracting the contours according to the order in the contour tree, and it can be used to extract the outer most contour if there are more than one contours for one shape, the hierarchy can also be used to treat the child contours.

```
# 5. Contour processing with hierarchy analysis
contours, hierarchy = cv2.findContours(closed, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
```

f) Filtering and Drawing Bounding Boxes:

Parent Contours: Parent contours are used to prevent redundant or nested detections by leveraging hierarchy details. For instance, if a tiny object is contained within a larger one, the parent contour will enable us to concentrate on the outer border while avoiding the identification of irrelevant inner contours.

Area Filtering:Area filtering is a method of choosing specific contours based on their areas. Filters can be applied to both small and big contours. Masks and object recognition programs can be made to ignore small contours and focus on the main subjects of interest.

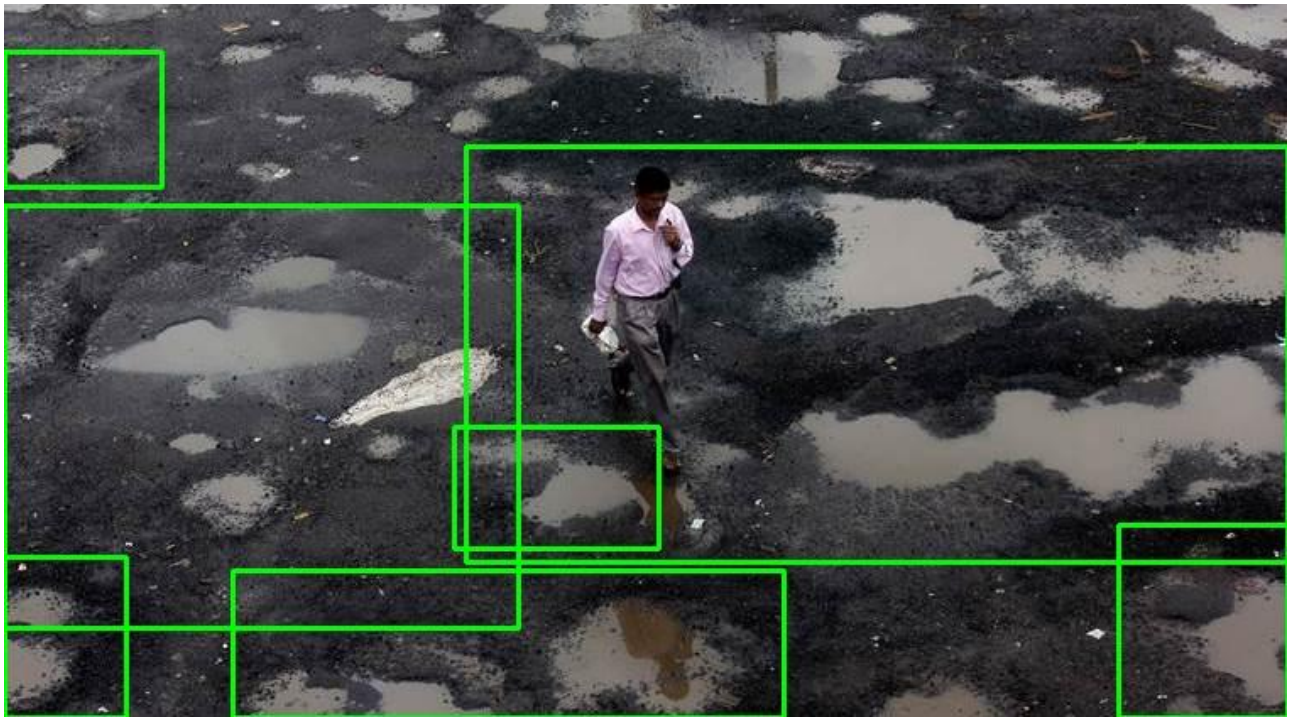
Aspect Ratio Filtering: This Method Focuses on Contours with Certain Shapes. For instance, if we are detecting potholes or speed breakers, they usually have a specific width-to-height ratio. By using this method, we can concentrate on contours that match this ratio and pay no attention to the objects that have other forms.

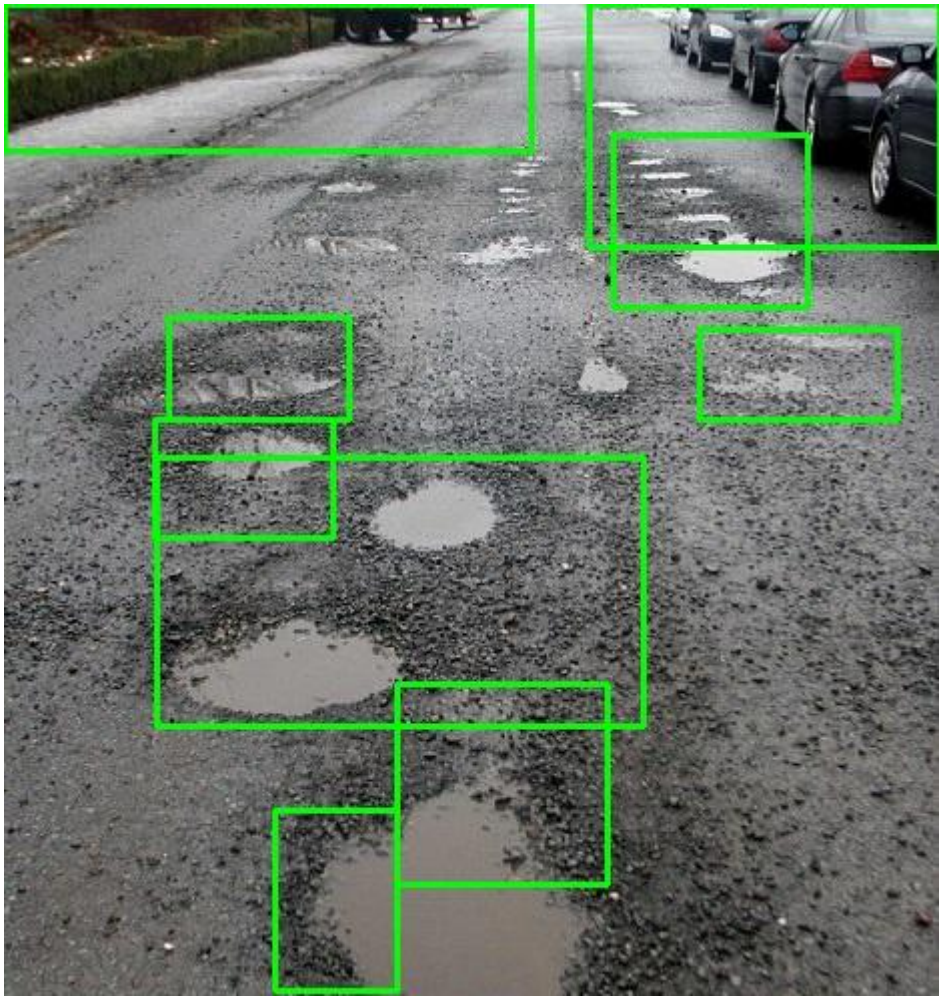
Drawing Rectangles: To visualize the image of an object, the contours surrounding the object are enclosed in rectangles. This way, the dimensions and extension of the identified parts can be seen, thereby making it easier for further analysis and assessment of the item.

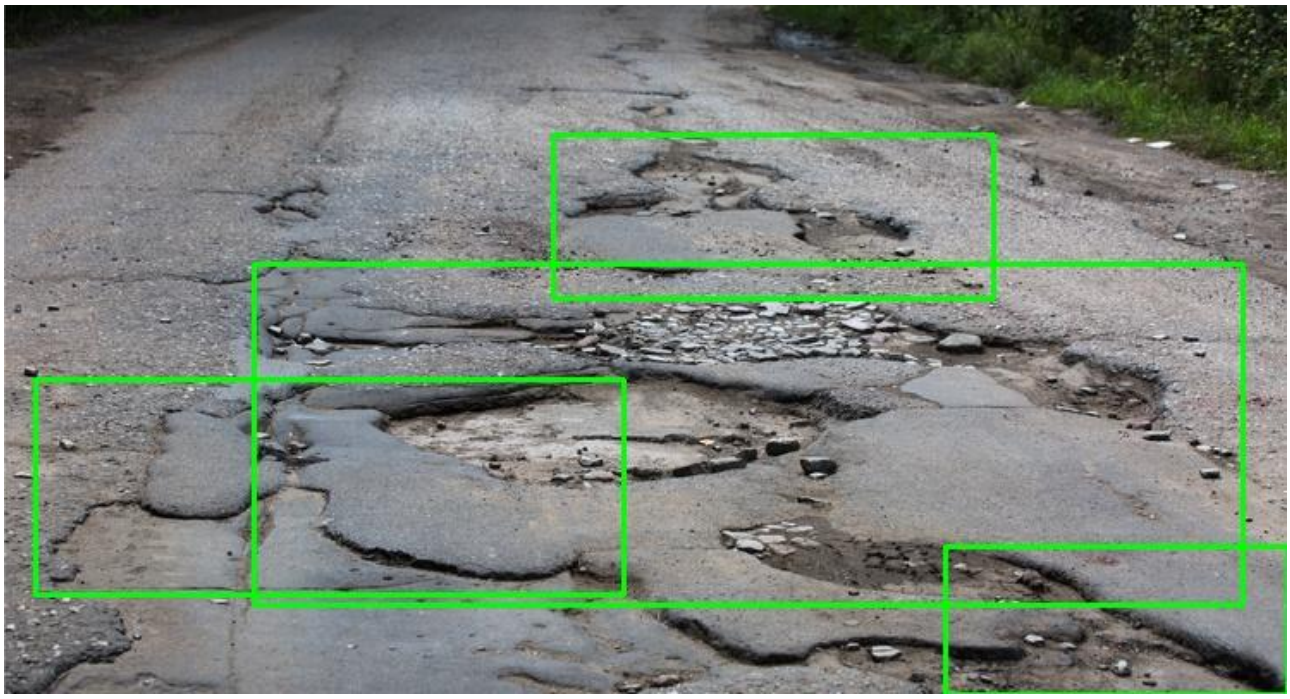
```
for i, cnt in enumerate(contours):
    if hierarchy[0][i][3] == -1:  # Filter parent contours only
        area = cv2.contourArea(cnt)
        if min_area < area < max_area:
            x, y, w, h = cv2.boundingRect(cnt)
            aspect_ratio = w / float(h)
            if 0.2 < aspect_ratio < 5:  # Filter by shape
                cv2.rectangle(image, (x, y), (x+w, y+h), (0, 255, 0), 2)
```

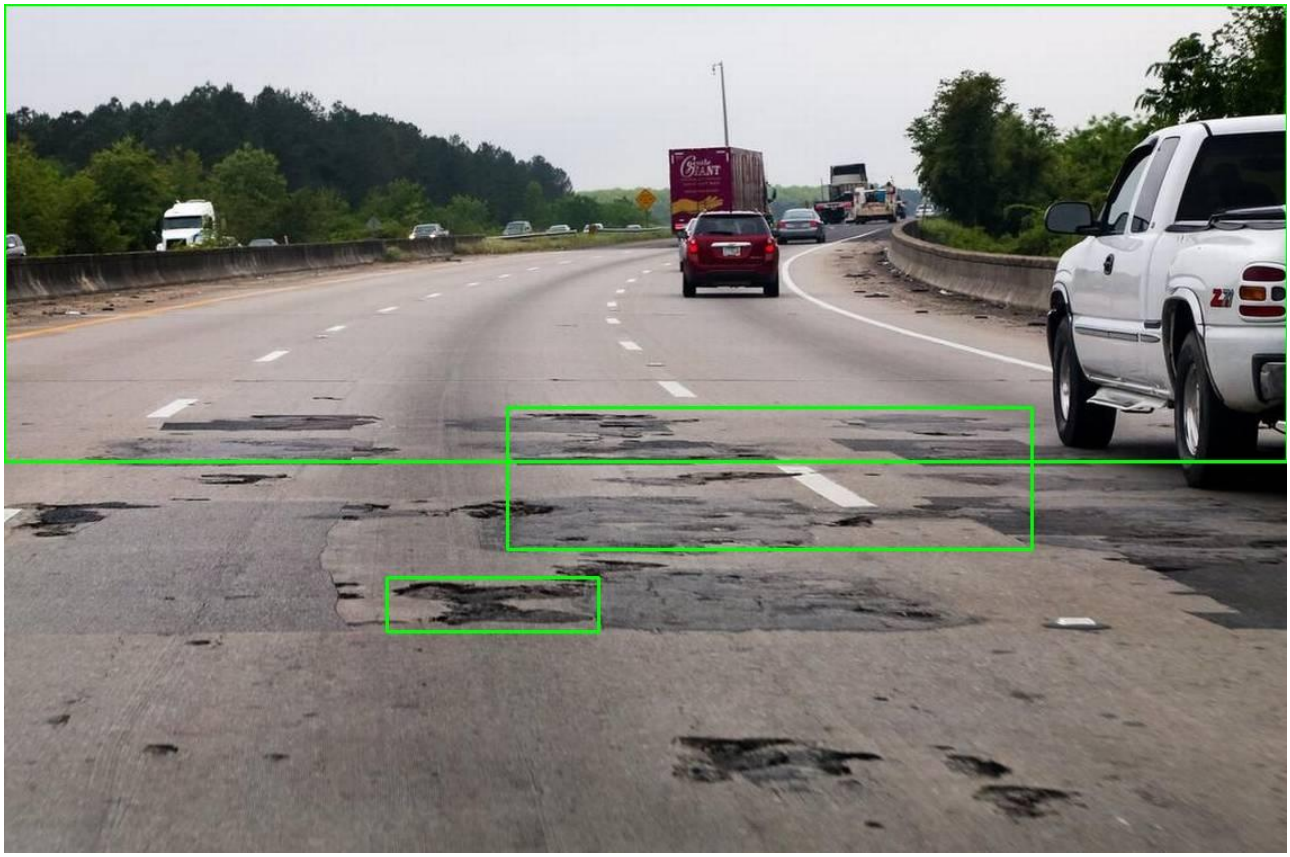
## III) RESULTS:

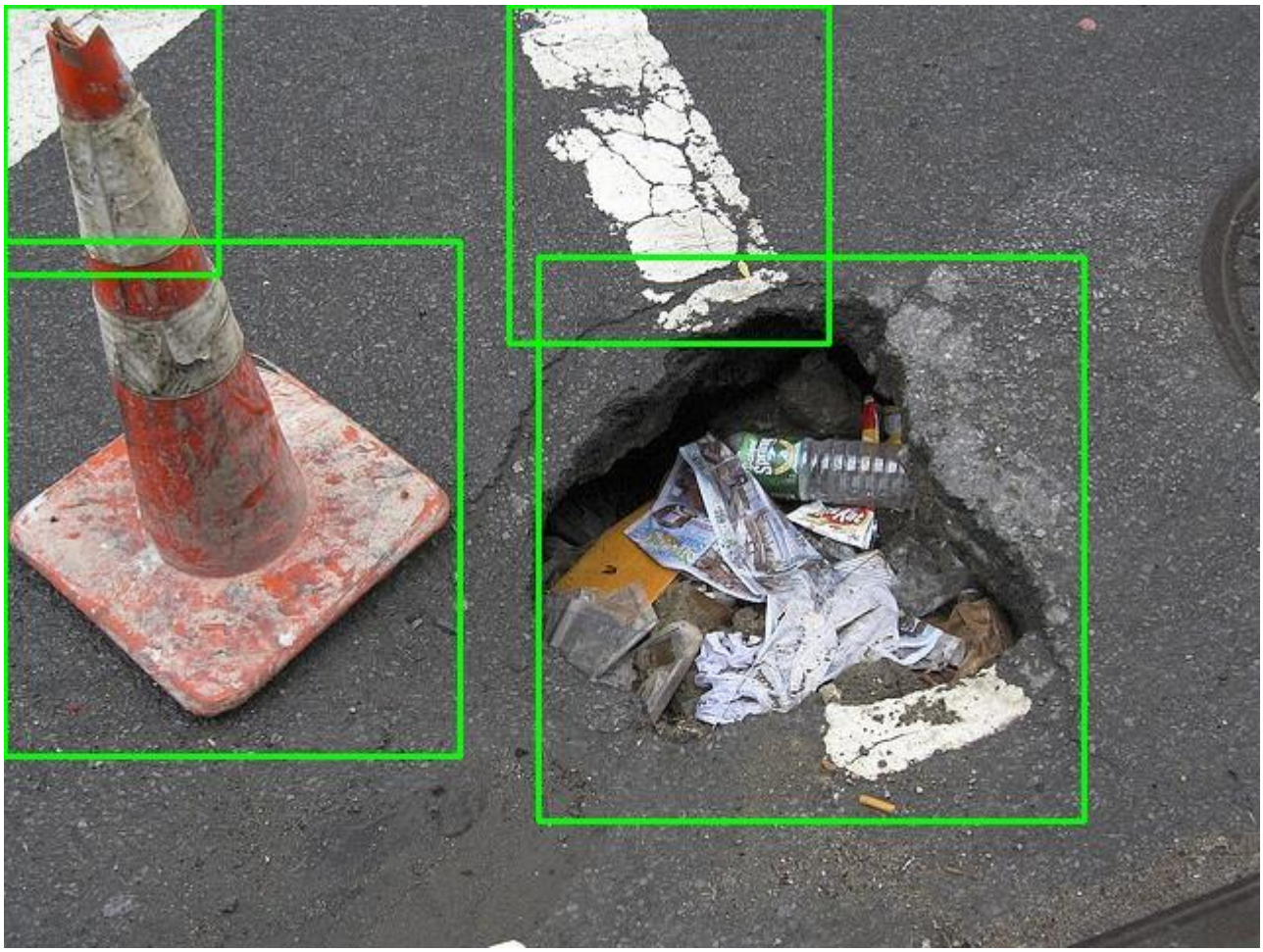
Following are some results after applying the image pre-processing and contrast identification techniques.











## IV) OBSERVATIONS:

The bounding boxes drawn around detected regions closely matched the actual locations and shapes of potholes and speedbreakers in most cases, providing clear visual confirmation of the pipeline's effectiveness. The visualization step not only aided in verifying the accuracy of detections but also made it easier to interpret and communicate results

Some limitations were observed: in images with complex backgrounds or poor lighting, the pipeline occasionally produced false positives or missed subtle features. This is consistent with broader literature, where classical image processing methods may underperform compared to deep learning approaches in highly variable real-world conditions.

## V) CONCLUSION:

These techniques show promise to enhance the efficiency and accuracy of ML models potentially trained on such a pre-processed dataset to detect objects, potholes, speedbreakers, etc.

These techniquies could potentially reduce the training time by having enhanced features that make it more obvious for the model to predict or detect the same.

These and similar techniques can be applied on varying datasets to enhance identification and model response, potentially in fields of geo-spatial imagery, astronomy, marin biology, etc.

## IV) REFERENCES:


1. OpenCV Documentation: Image Processing Modules.
   [https://docs.opencv.org/4.x/d2/d96/tutorial_py_table_of_contents_imgproc.html][2]
2. Application of Image Processing Techniques for Pothole Detection.
   Sathyabama Institute of Science and Technology.
   [https://sist.sathyabama.ac.in/sist_naac/documents/1.3.4/1822-b.e-cse-batchno-42.pdf][3]
3. Real-Time Pothole Detection Using Deep Learning.
   arXiv preprint arXiv:2107.06356.
   [https://arxiv.org/pdf/2107.06356.pdf][4]
4. Image Processing in OpenCV.
   International Journal for Research in Applied Science and Engineering Technology (IJRASET).
   [https://www.ijraset.com/research-paper/image-processing-in-open-cv][5]
5. Image Detection Using OpenCV Python.
   International Research Journal of Modernization in Engineering Technology and Science, Vol 5, Issue 10, October 2023.
   [https://www.irjmets.com/uploadedfiles/paper/issue_10_october_2023/45466/final/fin_irjmets1698244602.pdf][6]