OBJECT AND MOTION DETECTION MODEL

Submitted as part of the internship deliverables by:

Suraj Sharma.

Reference Number:

VT20254404

Under the supervision of:

T.N.Verma

Year:

2025

TATA STEEL

INTRODUCTION

This project sets up a basic motion detection system using Python and OpenCV. The main goal is to find and highlight moving objects in a video stream captured from a webcam. The code starts by importing necessary libraries like OpenCV for image processing, datetime for marking motion events, and pandas for storing and exporting motion data. It initializes key variables to track motion status, including a reference frame taken initially as the background, a list to watch for changes in motion status, and a timestamp list to log when motion starts and ends.

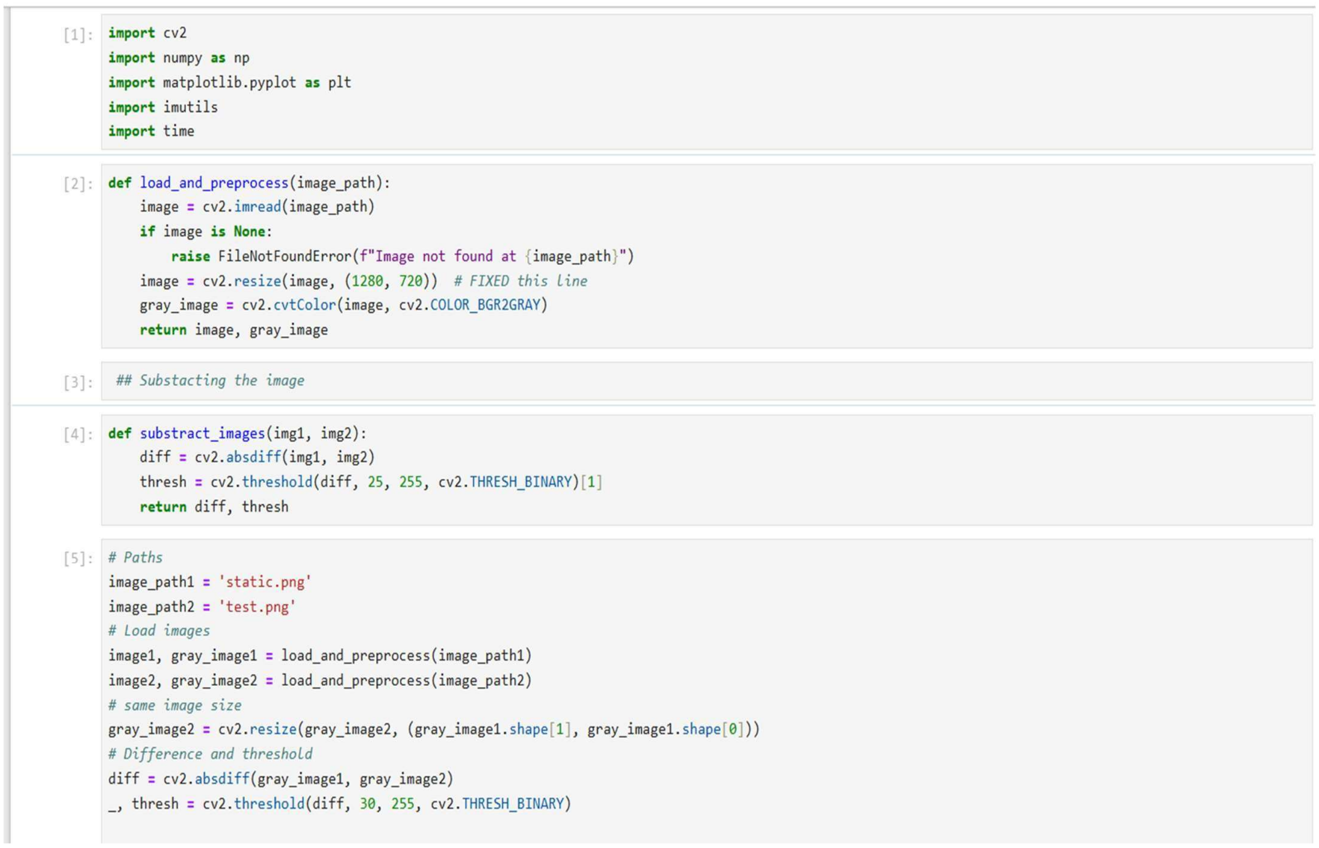
Each frame from the webcam is turned into grayscale and blurred to cut down on noise, making it easier to spot significant changes. Motion is detected by calculating the absolute difference between the current frame and the reference frame, then using thresholding and contour detection. If a large enough contour, signaling an object in motion, is found, the code marks that area with a rectangle and updates the motion status. This system is sensitive enough to notice even slight movements in the scene, such as trees swaying due to the wind, and treats them as motion events.

The system also records the exact moments when motion begins and ends, organizing this data in a structured format. Finally, when you stop the video feed by pressing the 'q' key, the recorded motion timestamps are saved to a CSV file for later analysis. Overall, this project shows a simple yet effective way to carry out real-time motion detection using basic image processing techniques.

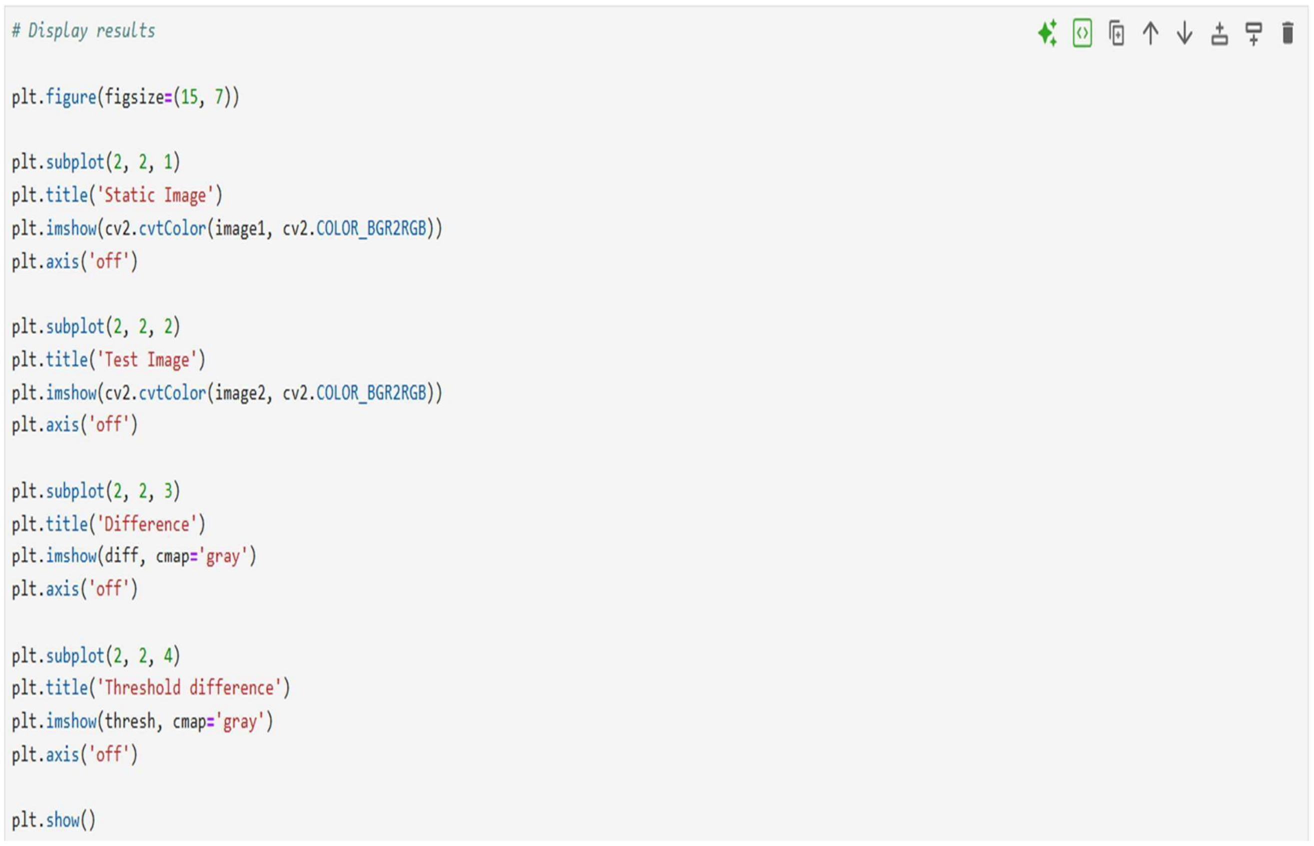
PROBLEM STUDY OF EXISTING SYSTEM

Traditional security systems and surveillance setups often rely on continuous video recording. This generates a large amount of unnecessary footage, most of which shows no relevant activity. These systems usually lack real-time intelligence and cannot tell the difference between actual intrusions and irrelevant movements, such as swaying trees or passing shadows. Consequently, reviewing this footage becomes time-consuming and inefficient. Additionally, many basic systems do not record the exact timestamps of motion events, making it hard to track when specific incidents happened. While advanced motion detection systems are available, they tend to be expensive and need significant computational resources or specialized hardware. Therefore, there is a need for a lightweight, cost-effective solution that can accurately detect motion, log the timing of these events, and even capture subtle movements in the environment, like tree branches moving in the wind. The existing system, in its basic form, lacks automation, efficiency, and smart event tracking.

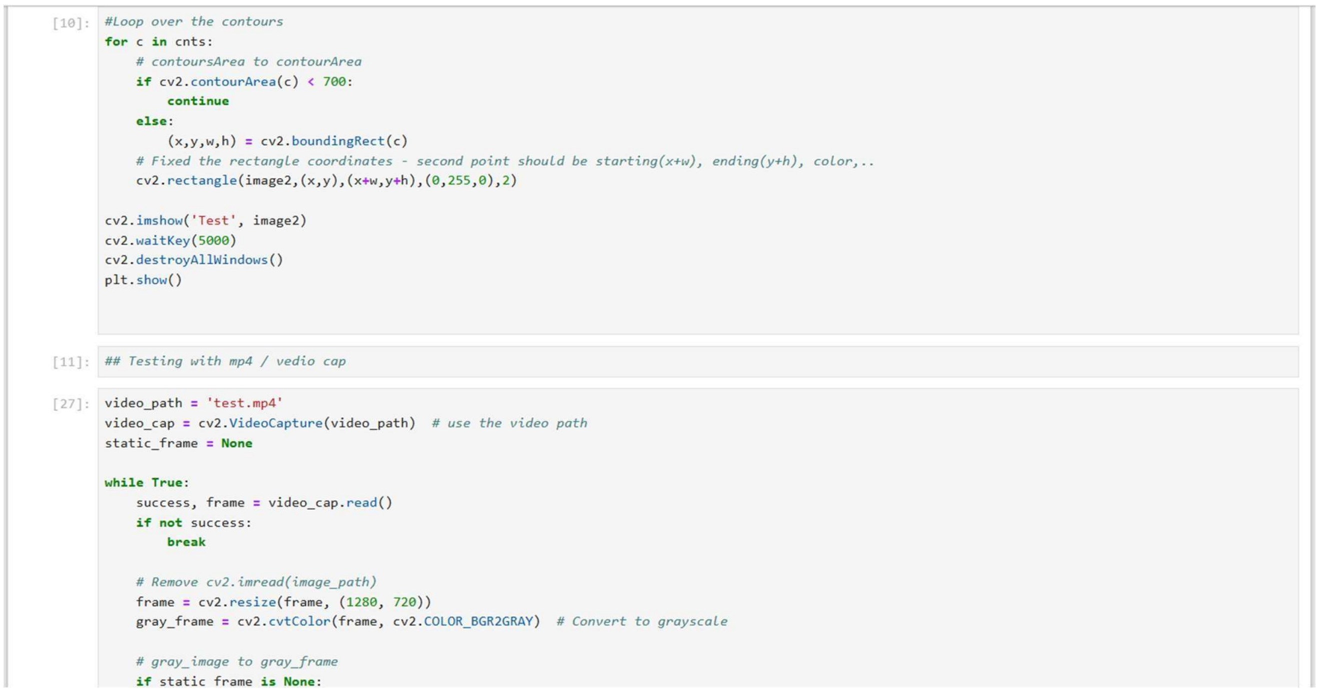
CODE



```
[1]: import cv2
     import numpy as np
     import matplotlib.pyplot as plt
     import imutils
     import time

[2]: def load_and_preprocess(image_path):
         image = cv2.imread(image_path)
         if image is None:
             raise FileNotFoundError(f"Image not found at {image_path}")
         image = cv2.resize(image, (1280, 720))  # FIXED this line
         gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
         return image, gray_image

[3]:  ## Substacting the image

[4]: def substract_images(img1, img2):
         diff = cv2.absdiff(img1, img2)
         thresh = cv2.threshold(diff, 25, 255, cv2.THRESH_BINARY)[1]
         return diff, thresh

[5]: # Paths
     image_path1 = 'static.png'
     image_path2 = 'test.png'
     # Load images
     image1, gray_image1 = load_and_preprocess(image_path1)
     image2, gray_image2 = load_and_preprocess(image_path2)
     # same image size
     gray_image2 = cv2.resize(gray_image2, (gray_image1.shape[1], gray_image1.shape[0]))
     # Difference and threshold
     diff = cv2.absdiff(gray_image1, gray_image2)
     _, thresh = cv2.threshold(diff, 30, 255, cv2.THRESH_BINARY)
```



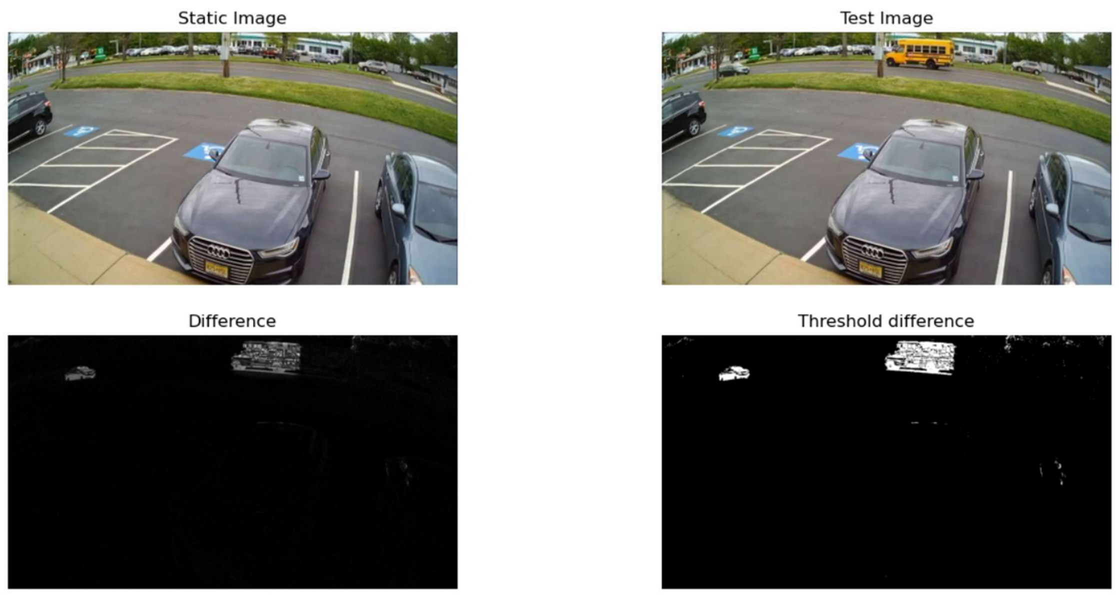
```
# Display results

plt.figure(figsize=(15, 7))

plt.subplot(2, 2, 1)
plt.title('Static Image')
plt.imshow(cv2.cvtColor(image1, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(2, 2, 2)
plt.title('Test Image')
plt.imshow(cv2.cvtColor(image2, cv2.COLOR_BGR2RGB))
plt.axis('off')

plt.subplot(2, 2, 3)
plt.title('Difference')
plt.imshow(diff, cmap='gray')
plt.axis('off')

plt.subplot(2, 2, 4)
plt.title('Threshold difference')
plt.imshow(thresh, cmap='gray')
plt.axis('off')

plt.show()
```

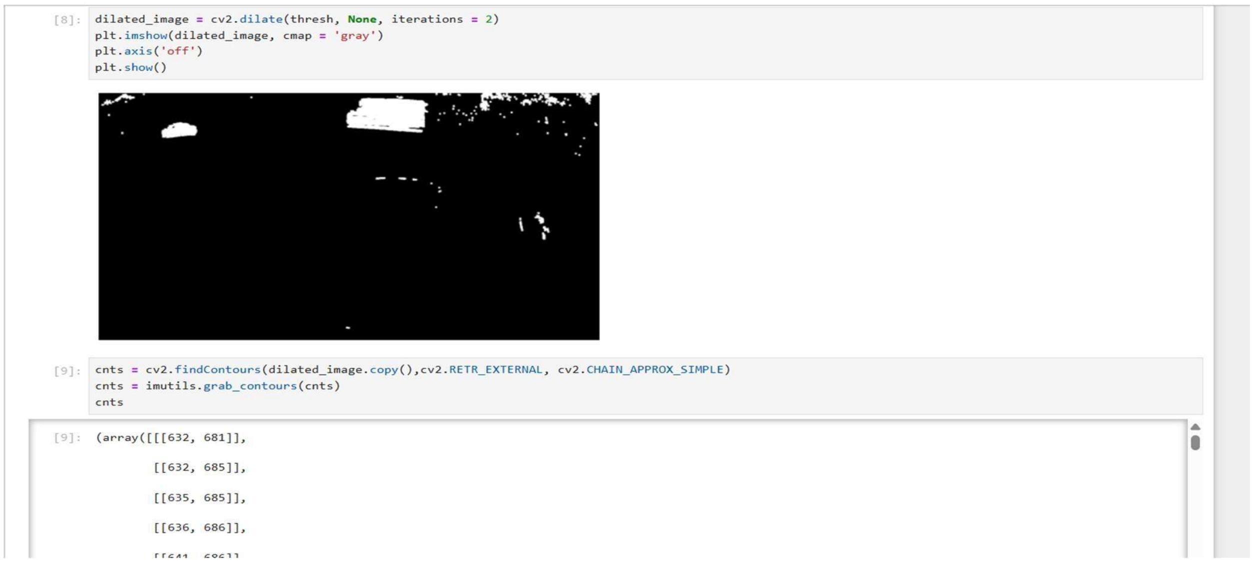


```
[10]: #Loop over the contours
      for c in cnts:
          # contoursArea to contourArea
          if cv2.contourArea(c) < 700:
              continue
          else:
              (x,y,w,h) = cv2.boundingRect(c)
          # Fixed the rectangle coordinates - second point should be starting(x+w), ending(y+h), color,..
          cv2.rectangle(image2,(x,y),(x+w,y+h),(0,255,0),2)

      cv2.imshow('Test', image2)
      cv2.waitKey(5000)
      cv2.destroyAllWindows()
      plt.show()
```

```
[11]: ## Testing with mp4 / vedio cap
```

```
[27]: video_path = 'test.mp4'
      video_cap = cv2.VideoCapture(video_path)  # use the video path
      static_frame = None

      while True:
          success, frame = video_cap.read()
          if not success:
              break

          # Remove cv2.imread(image_path)
          frame = cv2.resize(frame, (1280, 720))
          gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # Convert to grayscale

          # gray_image to gray_frame
          if static_frame is None:
```

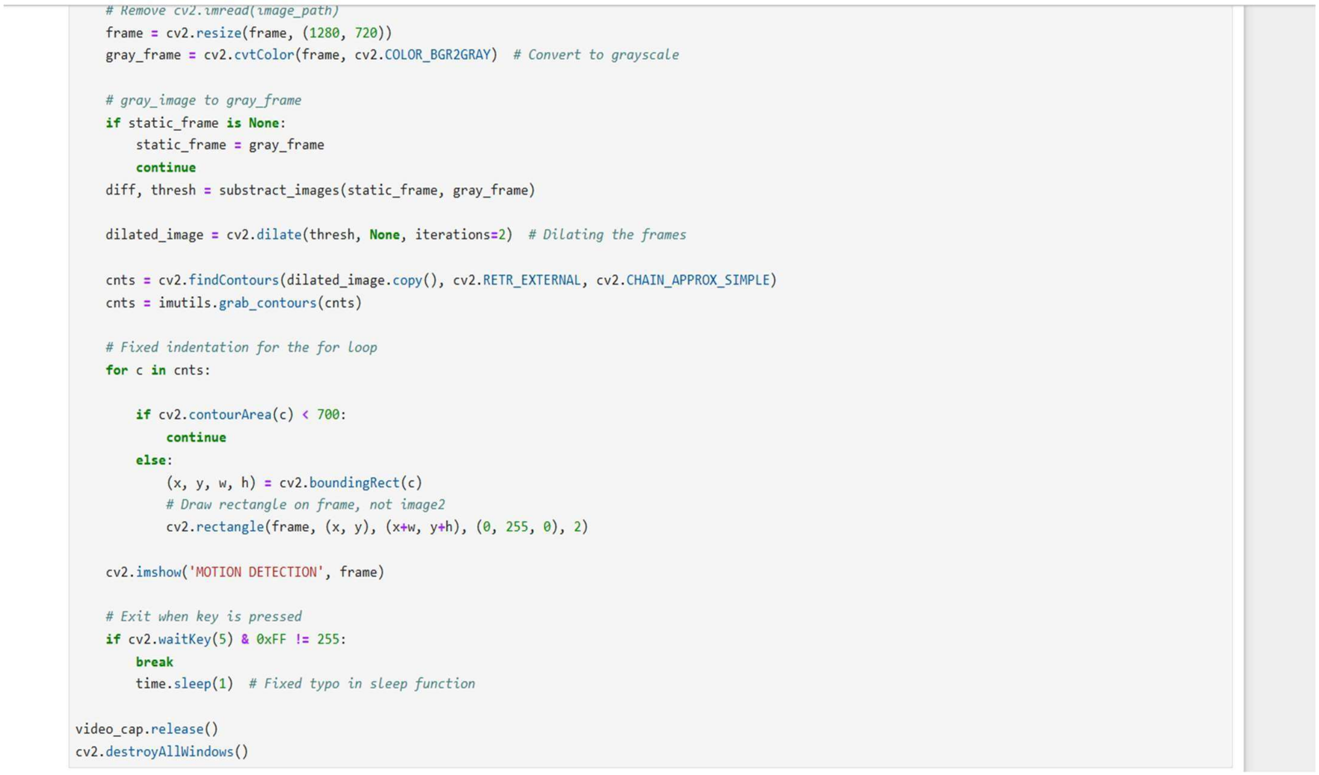




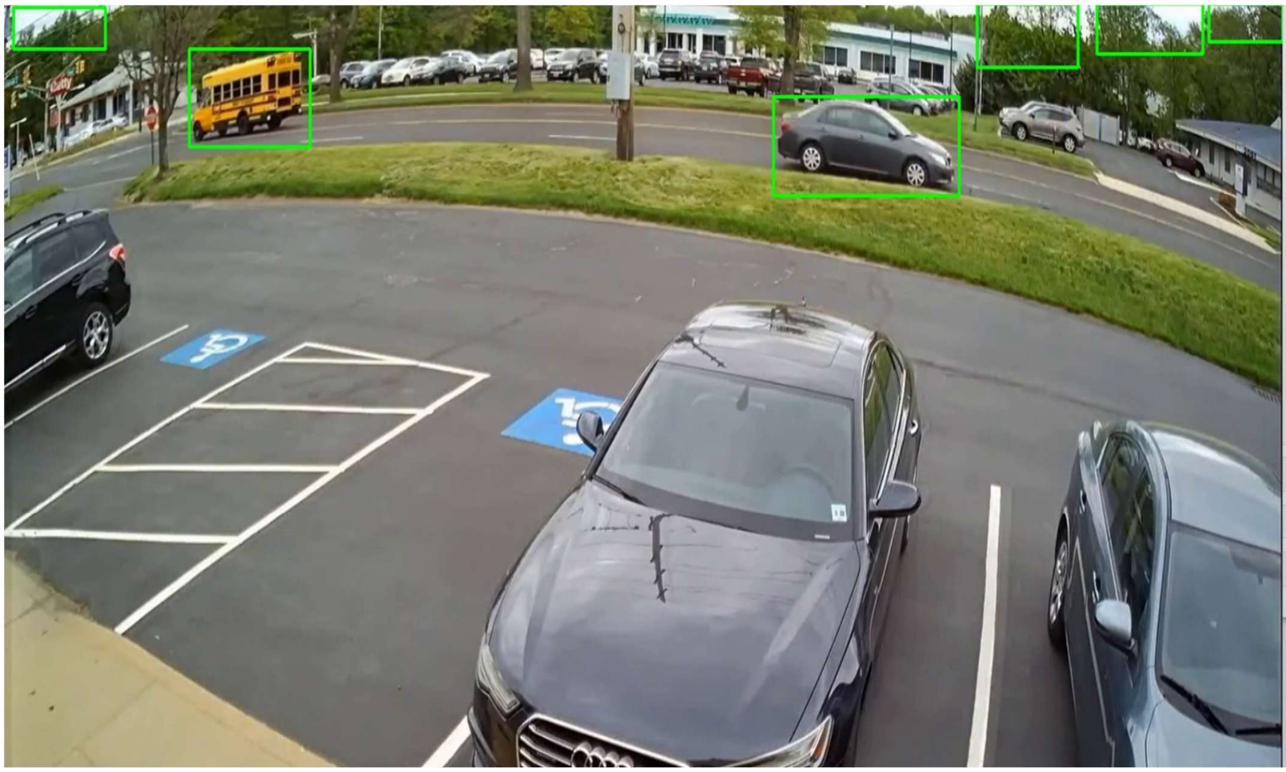
```
[8]: dilated_image = cv2.dilate(thresh, None, iterations = 2)
     plt.imshow(dilated_image, cmap = 'gray')
     plt.axis('off')
     plt.show()
```

```
[9]: cnts = cv2.findContours(dilated_image.copy(),cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
     cnts = imutils.grab_contours(cnts)
     cnts
```

```
[9]: (array([[[632, 681]],

             [[632, 685]],

             [[635, 685]],

             [[636, 686]],

             [[641, 686]]
```



```
    # Remove cv2.imread(image_path)
    frame = cv2.resize(frame, (1280, 720))
    gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # Convert to grayscale

    # gray_image to gray_frame
    if static_frame is None:
        static_frame = gray_frame
        continue
    diff, thresh = substract_images(static_frame, gray_frame)

    dilated_image = cv2.dilate(thresh, None, iterations=2)  # Dilating the frames

    cnts = cv2.findContours(dilated_image.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    cnts = imutils.grab_contours(cnts)

    # Fixed indentation for the for loop
    for c in cnts:

        if cv2.contourArea(c) < 700:
            continue
        else:
            (x, y, w, h) = cv2.boundingRect(c)
            # Draw rectangle on frame, not image2
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)

    cv2.imshow('MOTION DETECTION', frame)

    # Exit when key is pressed
    if cv2.waitKey(5) & 0xFF != 255:
        break
        time.sleep(1)  # Fixed typo in sleep function

video_cap.release()
cv2.destroyAllWindows()
```



CODE AND EXPLAINATION

- Code Block 1: Importing Required Libraries import cv2

```
import numpy as np import matplotlib.pyplot as plt import
imutils import time
```

- Code Block 2: Load and Preprocess the Image def

```
load_and_preprocess(image_path):     image =
cv2.imread(image_path)     if image is None:
        raise FileNotFoundError(f"Image not found at {image_path}")
image = cv2.resize(image, (1280, 720))  # FIXED this line     gray_image =
cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)     return image,
gray_image
```

- Code Block 3: Image Subtraction Function def

```
substract_images(img1, img2):     diff = cv2.absdiff(img1,
img2)     thresh = cv2.threshold(diff, 25, 255,
cv2.THRESH_BINARY)[1]
    return diff, thresh
```

- Code Block 4: Load and Compare Two Images

```
# Paths image_path1 = 'static.png' image_path2 = 'test.png' #
Load images image1, gray_image1 =
load_and_preprocess(image_path1) image2, gray_image2 =
load_and_preprocess(image_path2)
# same image size
gray_image2 = cv2.resize(gray_image2, (gray_image1.shape[1], gray_image1.shape[0]))
# Difference and threshold diff =
cv2.absdiff(gray_image1, gray_image2)
```

_, thresh = cv2.threshold(diff, 30, 255, cv2.THRESH_BINARY)

 Code Block 5: Visualization Using Matplotlib

```
# Display results
plt.figure(figsize=(15, 7))
plt.subplot(2, 2, 1)
plt.title('Static Image')
plt.imshow(cv2.cvtColor(image1, cv2.COLOR_BGR2RGB))
plt.axis('off') plt.subplot(2,
2, 2) plt.title('Test Image')
plt.imshow(cv2.cvtColor(image2, cv2.COLOR_BGR2RGB))
plt.axis('off') plt.subplot(2, 2,
3) plt.title('Difference')
plt.imshow(diff, cmap='gray')
plt.axis('off') plt.subplot(2, 2, 4)
plt.title('Threshold difference')
plt.imshow(thresh, cmap='gray')
plt.axis('off') plt.show()
```

 Code Block 6: Dilation dilated_image = cv2.dilate(thresh, None, iterations = 2)

```
plt.imshow(dilated_image, cmap = 'gray')
plt.axis('off') plt.show()
```

 Code Block 7: Finding Contours

```
cnts = cv2.findContours(dilated_image.copy(),cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts) cnts
```

 Code Block 8: Loop over the contours

```
for c in cnts:
  # contoursArea to contourArea     if
cv2.contourArea(c) < 700:
      continue
else:
      (x,y,w,h) = cv2.boundingRect(c)
  # Fixed the rectangle coordinates - second point should be starting(x+w), ending(y+h), color,..
  cv2.rectangle(image2,(x,y),(x+w,y+h),(0,255,0),2)


cv2.imshow('Test', image2)
cv2.waitKey(5000)
cv2.destroyAllWindows() plt.show()
```

□ Code Block 9: Drawing Bounding Boxes on Motion with mp4 / vedio cap

```
  video_path = 'test.mp4' video_cap = cv2.VideoCapture(video_path)  # use the video
  path  static_frame = None while True:
    success, frame = video_cap.read()     if
  not success:
        break
    # Remove cv2.imread(image_path)     frame = cv2.resize(frame, (1280, 720))     gray_frame
  = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)  # Convert to grayscale
    # gray_image to gray_frame
  if static_frame is None:
  static_frame = gray_frame
  continue     diff, thresh =
  substract_images(static_frame,
  gray_frame)       dilated_image =
  cv2.dilate(thresh, None,
  iterations=2)  # Dilating the frames
    cnts = cv2.findContours(dilated_image.copy(), cv2.RETR_EXTERNAL,
  cv2.CHAIN_APPROX_SIMPLE)
```

```
    cnts = imutils.grab_contours(cnts)
# Fixed indentation for the for loop
for c in cnts:         if cv2.contourArea(c)
< 700:
            continue
else:
            (x, y, w, h) = cv2.boundingRect(c)             # Draw
rectangle on frame, not image2             cv2.rectangle(frame, (x,
y), (x+w, y+h), (0, 255, 0), 2)
    cv2.imshow('MOTION DETECTION', frame)
    # when key is pressed do Exit     if
cv2.waitKey(5) & 0xFF != 255:
        break         time.sleep(1)  # Fixed typo in sleep
function video_cap.release()
cv2.destroyAllWindows()
```

CONCLISION

In this project, I have successfully created a basic but effective Object & motion detection Model using image processing techniques. By using OpenCV, Jupyter NoteBook and Python, the model could detect changes between still and moving frames through grayscale conversion, frame subtraction, thresholding, and contour detection. Further improvements with dilation helped increase accuracy in identifying areas of movement.

This system can be used in real-world applications like surveillance, wildlife monitoring, and automated security systems. It also shows how simple computer vision techniques can solve real problems. Future improvements could include adding background subtraction methods, real-time video feeds, and machine learning models for object classification and motion tracking.