Alaa Hosni 223101325 / Lujain Mohammed 223101330 / Shahd Saleh 223101361

Car Detection Using OpenCV and Image Processing Techniques

Introduction

This document explains a computer vision application for detecting cars in images by comparing a scene with and without cars (background subtraction). The implementation uses manual image processing techniques along with OpenCV functions to identify and highlight cars in parking areas.

Code Overview

The code performs the following steps:

1. Loads two images: one with cars and one without (empty reference)
2. Converts images to grayscale using manual calculation
3. Computes the difference between images
4. Applies thresholding to highlight significant changes
5. Uses morphological operations (manually implemented) to clean up the result
6. Extracts contours and identifies car-sized objects
7. Draws bounding boxes around detected cars

Detailed Implementation

Image Loading and Preprocessing

python

```python
import cv2
import numpy as np
from matplotlib import pyplot as plt

# Load the images
img_with = cv2.imread("img.jpeg")        # Image with cars
img_empty = cv2.imread("empty.jpeg")     # Empty reference image

# Ensure both images are the same size
img_with = cv2.resize(img_with, (img_empty.shape[1], img_empty.shape[0]))
```

Manual Grayscale Conversion

Instead of using OpenCV's built-in function, a manual approach is implemented using the standard RGB to grayscale formula:

python

```python
def to_grayscale(image):
    # Standard RGB to grayscale conversion formula:
    # Gray = 0.299*R + 0.587*G + 0.114*B
    return (0.299 * image[:, :, 2] + 0.587 * image[:, :, 1] + 0.114 * image[:, :, 0]).astyp

gray_with = to_grayscale(img_with)
gray_empty = to_grayscale(img_empty)
```

Background Subtraction

The difference between the two grayscale images is calculated to identify changes (potential cars):

python

```python
# Calculate the absolute difference between images
diff = np.abs(gray_with.astype(np.int16) - gray_empty.astype(np.int16)).astype(np.uint8)

# Apply manual thresholding (pixel values > 30 become 255, others become 0)
thresh = np.where(diff > 30, 255, 0).astype(np.uint8)
```

Morphological Operations

Morphological operations are implemented manually to clean up the binary image:

```python
# Define a 5x5 kernel
kernel = np.ones((5, 5), np.uint8)

# Manual implementation of dilation
def dilate(img, kernel):
    pad = kernel.shape[0] // 2
    padded = np.pad(img, pad, mode='constant', constant_values=0)
    out = np.zeros_like(img)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            region = padded[i:i+kernel.shape[0], j:j+kernel.shape[1]]
            out[i, j] = np.max(region * kernel)
    return out

# Manual implementation of erosion
def erode(img, kernel):
    pad = kernel.shape[0] // 2
    padded = np.pad(img, pad, mode='constant', constant_values=255)
    out = np.zeros_like(img)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            region = padded[i:i+kernel.shape[0], j:j+kernel.shape[1]]
            out[i, j] = np.min(region * kernel)
    return out

# Apply morphological closing (dilation followed by erosion)
closed = dilate(thresh, kernel)
closed = erode(closed, kernel)

# Apply morphological opening (erosion followed by dilation)
opened = erode(closed, kernel)
opened = dilate(opened, kernel)
```

Contour Detection and Car Identification

Finally, the code detects contours in the processed image and filters them based on size to identify cars:

python

```python
# Extract contours
contours, _ = cv2.findContours(opened, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

# Draw bounding boxes around car-sized objects
for cnt in contours:
    area = cv2.contourArea(cnt)
    # Filter contours by area to identify cars (area between 100 and 2500 pixels)
    if 100 < area < 2500:
        x, y, w, h = cv2.boundingRect(cnt)
        cv2.rectangle(img_with, (x, y), (x+w, y+h), (255, 0, 0), 2)  # Draw blue rectangle
```

Visualization

The result is displayed using Matplotlib:

python

```python
# Display the result
plt.figure(figsize=(14, 10))
plt.imshow(cv2.cvtColor(img_with, cv2.COLOR_BGR2RGB))  # Convert BGR to RGB for correct dis
plt.axis("off")
plt.title("Detected Cars (Manual Processing)")
plt.show()
```

Technical Notes

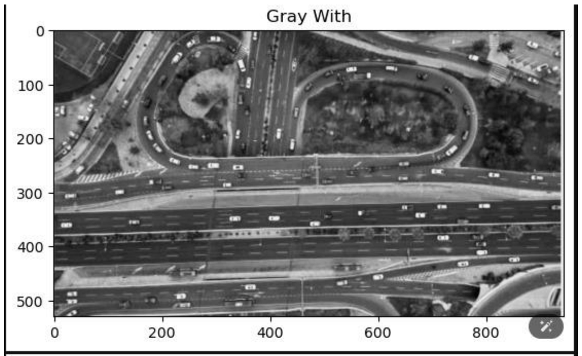
1. **Manual Implementation vs. OpenCV Functions**:
   - The code implements several standard image processing operations manually instead of using built-in OpenCV functions
   - This approach provides greater transparency into the algorithm's workings but is computationally less efficient
2. **Performance Considerations**:
   - The manual implementations of morphological operations are significantly slower than OpenCV's optimized versions
   - For production applications, using built-in functions would be recommended
3. **Detection Parameters**:
   - Threshold value (30) determines sensitivity to changes between images

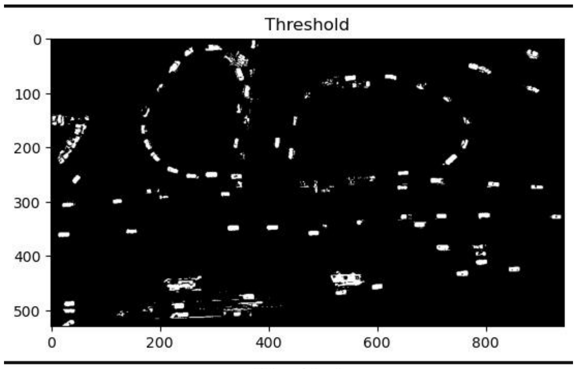
- Contour area filtering (100-2500) determines what size objects are considered cars
- These parameters may need adjustment for different scenes

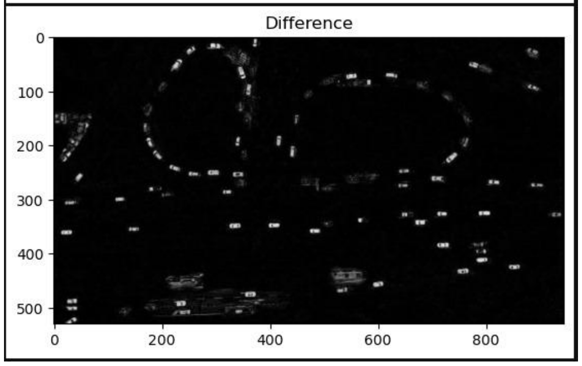
Conclusion

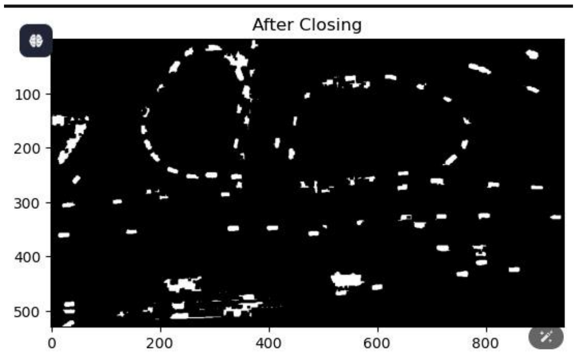
This implementation demonstrates a basic approach to car detection using background subtraction and image processing techniques. The manual implementation of standard algorithms provides educational insight into how these operations work at a fundamental level.

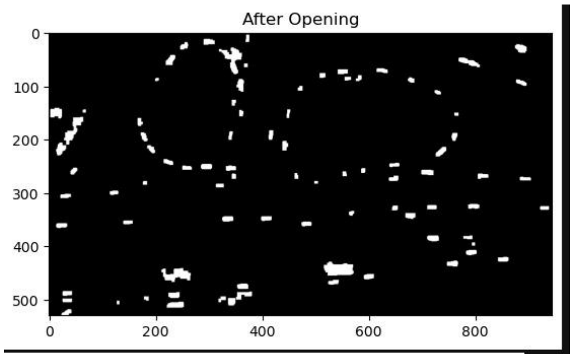
For practical applications, the code could be optimized by leveraging OpenCV's built-in functions and further refined by implementing additional filtering techniques to improve detection accuracy.




Threshold
0
100
200
300
400
500
0
200
400
600
800


Difference
0
100
200
300
400
500
0
200
400
600
800


After Closing
100
200
300
400
500
0
200
400
600
800


After Opening
0
100
200
300
400
500
0
200
400
600
800