IMAGE PROCESSING

Parking Space Counter Project Report

NUR SULTAN VASİ 19155058

TEOMAN ÜNAL 17155042

Parking Space Counter Project Report

Introduction

Efficient utilization of parking spaces is of great importance in today's modern cities. This project aims to develop a system that detects empty parking spaces in parking lots and provides this information in real-time. This report will detail the objectives of the project, the methods used, the findings obtained, and potential future improvements.

Project Objective

The primary objective of the project is to automatically detect empty parking spaces in parking lots and provide this information in real-time. This allows for:

Parking management to have real-time information for efficient use of parking areas.

Drivers to reduce the time spent searching for empty parking spaces, thus saving time and fuel.

Reduction of traffic congestion and environmental pollution.

3. Technologies and Methods Used

3.1. Technologies Used

3.1.1. OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source library used for computer vision and machine learning tasks. It has been heavily utilized in this project for performing image processing tasks. The extensive range of functions provided by OpenCV has facilitated the image processing and analysis steps required for the project.

3.1.2. Python

Python is a high-level programming language known for its simple and readable syntax. Python has been preferred for this project due to its extensive library support and ease of use. Additionally, the strong community and continuously updated packages of Python have ensured a fast and effective development process for the project.

3.1.3. Pickle

Python's pickle module enables serialization and deserialization of Python objects. It has been used in this project to store and reload the coordinates of parking spaces. This enhances the flexibility and ease of use of the project.

3.2. Algorithms and Methods Used

The project consists of two main components:

Parking Space Picker (parking_space_picker.py)

Parking Space Counter (parking_space_counter.py)

3.2.1. Parking Space Picker

This module allows the user to identify empty parking spaces in the parking lot using the mouse. The user marks the corner points of parking spaces using the mouse, and the parking space picker saves these coordinates in a file.

Algorithms and Methods Used:

Mouse Events:

The user identifies parking spaces using the mouse. The cv2.EVENT_LBUTTONDOWN and cv2.EVENT_LBUTTONUP events are used to determine the coordinates of parking spaces. This allows the user to easily mark and edit the locations of parking spaces.

Serialization:

The coordinates of parking spaces are stored in a file using the Pickle module and can be reloaded when needed. This method allows for the permanent storage of the locations of parking spaces and quick loading when required.

How It Works?:

a) Create a list named "park_positions" to store the coordinates of parking spaces.

b) Listen for mouse events. The user uses the left mouse button to mark the corner points of parking spaces.

c) Calculate the coordinates of parking spaces and add them to the park_positions list.

d) While continuing interaction with the user through mouse events and keyboard events, parking spaces are continuously marked.

e) The user can delete incorrectly marked parking spaces using the right mouse button.

f) The coordinates of parking spaces are stored in a file using the Pickle module.

Used Code:

```
import cv2

import pickle

from math import sqrt


width, height = 40, 23

pt1_x, pt1_y, pt2_x, pt2_y = None, None, None, None

line_count = 0


try:

    with open('park_positions', 'rb') as f:

        park_positions = pickle.load(f)

except:

    park_positions = []



def parking_line_counter():

    global line_count

    line_count = int((sqrt((pt2_x - pt1_x) ** 2 + (pt2_y - pt1_y) ** 2)) / height)

    return line_count



def mouse_events(event, x, y, flag, param):

    global pt1_x, pt1_y, pt2_x, pt2_y


    if event == cv2.EVENT_LBUTTONDOWN:
        pt1_x, pt1_y = x, y

    elif event == cv2.EVENT_LBUTTONUP:
        pt2_x, pt2_y = x, y
        parking_spaces = parking_line_counter()
        if parking_spaces == 0:
            park_positions.append((x, y))
        else:
            for i in range(parking_spaces):
                park_positions.append((pt1_x, pt1_y + i * height))

    if event == cv2.EVENT_RBUTTONDOWN:
        for i, position in enumerate(park_positions):
            x1, y1 = position
            if x1 < x < x1 + width and y1 < y < y1 + height:
                park_positions.pop(i)

    with open('park_positions', 'wb') as f:
        pickle.dump(park_positions, f)


while True:

    img = cv2.imread('input/parking.png')

    for position in park_positions:
        cv2.rectangle(img, position, (position[0] + width, position[1] + height), (255, 0, 255), 1)

    cv2.namedWindow('image', cv2.WINDOW_NORMAL)
    cv2.setWindowProperty('image', cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)
    cv2.imshow('image', img)
    cv2.setMouseCallback('image', mouse_events)

    key = cv2.waitKey(30)
    if key == 27:
        break

cv2.destroyAllWindows()
```

3.2.2. Parking Space Counter

This module analyzes the occupancy status of parking spaces in each frame of the video stream. Empty parking spaces are detected using image processing techniques, and their numbers are continuously updated.

Algorithms and Methods Used:

Grayscale Conversion: The color image is converted to a grayscale image. This process enables faster and more efficient image processing steps.

Blurring: Gaussian blur is applied to the image to reduce noise. This is important to prevent false positive detections.

Thresholding: An adaptive thresholding method is used to obtain a binary image. This method makes parking spaces more distinct in the image.

Pixel Counting: The number of white pixels in each parking space region is calculated to determine the occupancy rate. The calculated ratio is used to decide whether the parking space is occupied or empty.

How It Works?:

a) A file named "park_positions" stores the coordinates of parking spaces. These coordinates are used for the detection of parking spaces.

b) Processing is performed on each video frame.

c) Grayscale conversion, blurring, and thresholding are applied to the frame to enhance the visibility of parking spaces.

d) The number of white pixels in each parking space is counted.

e) The occupancy rate is calculated and compared with a threshold value.

f) Parking spaces below the threshold value are considered empty, and their numbers are updated.

g) The real-time updated count of empty parking spaces is visualized on the video stream.

Used Code:

```
import cv2

import pickle


cap = cv2.VideoCapture('input/parking.mp4')


with open('park_positions', 'rb') as f:

    park_positions = pickle.load(f)


font = cv2.FONT_HERSHEY_COMPLEX_SMALL


# Parking space parameters

width, height = 40, 20

full = width * height

empty = 0.22




def parking_space_counter(img_processed):

    global counter


    counter = 0


    for position in park_positions:

        x, y = position


        img_crop = img_processed[y:y + height, x:x + width]
        count = cv2.countNonZero(img_crop)

        ratio = count / full

        if ratio < empty:
            color = (0, 255, 0)
            counter += 1
        else:
            color = (0, 0, 255)

        cv2.rectangle(overlay, position, (position[0] + width, position[1] + height), color, -1)
        cv2.putText(overlay, "{:.2f}".format(ratio), (x + 4, y + height - 4), font, 0.7, (255, 255, 255), 1,
cv2.LINE_AA)


while True:

    # Video looping
    if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT):
        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)

    _, frame = cap.read()
    overlay = frame.copy()

    # Frame processing
    img_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    img_blur = cv2.GaussianBlur(img_gray, (3, 3), 1)
    img_thresh = cv2.adaptiveThreshold(img_blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV, 25, 16)

    parking_space_counter(img_thresh)
    alpha = 0.7
    frame_new = cv2.addWeighted(overlay, alpha, frame, 1 - alpha, 0)

    w, h = 220, 60
    cv2.rectangle(frame_new, (0, 0), (w, h), (255, 0, 255), -1)
    cv2.putText(frame_new, f"{counter}/{len(park_positions)}", (int(w / 10), int(h * 3 / 4)), font, 2,
(255, 255, 255), 2, cv2.LINE_AA)

    cv2.namedWindow('frame', cv2.WINDOW_NORMAL)
    cv2.setWindowProperty('frame', cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)

    cv2.imshow('frame', frame_new)
    # cv2.imshow('image_blur', img_blur)
    # cv2.imshow('image_thresh', img_thresh)

    if cv2.waitKey(1) & 0xFF == 27:
        break

cap.release()
cv2.destroyAllWindows()
```

4.Methodology

4.1. Data Collection

At the beginning of the project, the coordinates of parking spaces were manually determined using an image of the parking area, and these coordinates were saved to a file. This process was carried out using the parking_space_picker.py file. The user selects parking spaces with the mouse, and the locations of parking spaces are saved to the park_positions file.

4.2. Image Processing

A series of image processing steps were performed on each frame of the video stream:

Grayscale Conversion: The color image was converted to a grayscale image.

Blurring: Gaussian blur was applied to the image to reduce noise.

Thresholding: An adaptive thresholding method was used to obtain a binary image.

4.3. Parking Space Detection and Counting

Each parking space was cropped using the coordinates obtained from the park_positions file. The number of white pixels in these cropped areas was calculated to determine the occupancy status of each parking space. If the occupancy rate was below a certain threshold, the parking space was considered empty.

5. Conclusion

This project has developed a system that detects empty parking spaces in parking lots using image processing techniques and provides this information in real-time. The system provides significant benefits for parking management and users. With future improvements, the accuracy and stability of the system can be increased, enabling a wider range of applications.

6.Project Outputs

This section will present some sample outputs from the project tests. These outputs visually represent the working principles and accuracy of the system.

Figure 1: Parking space picker

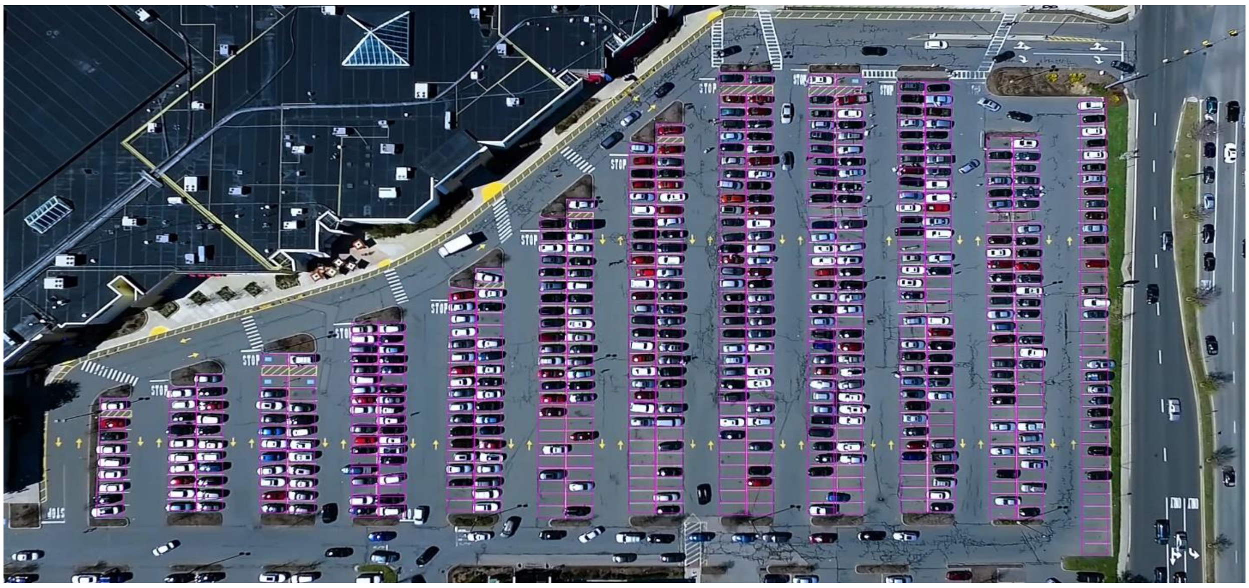
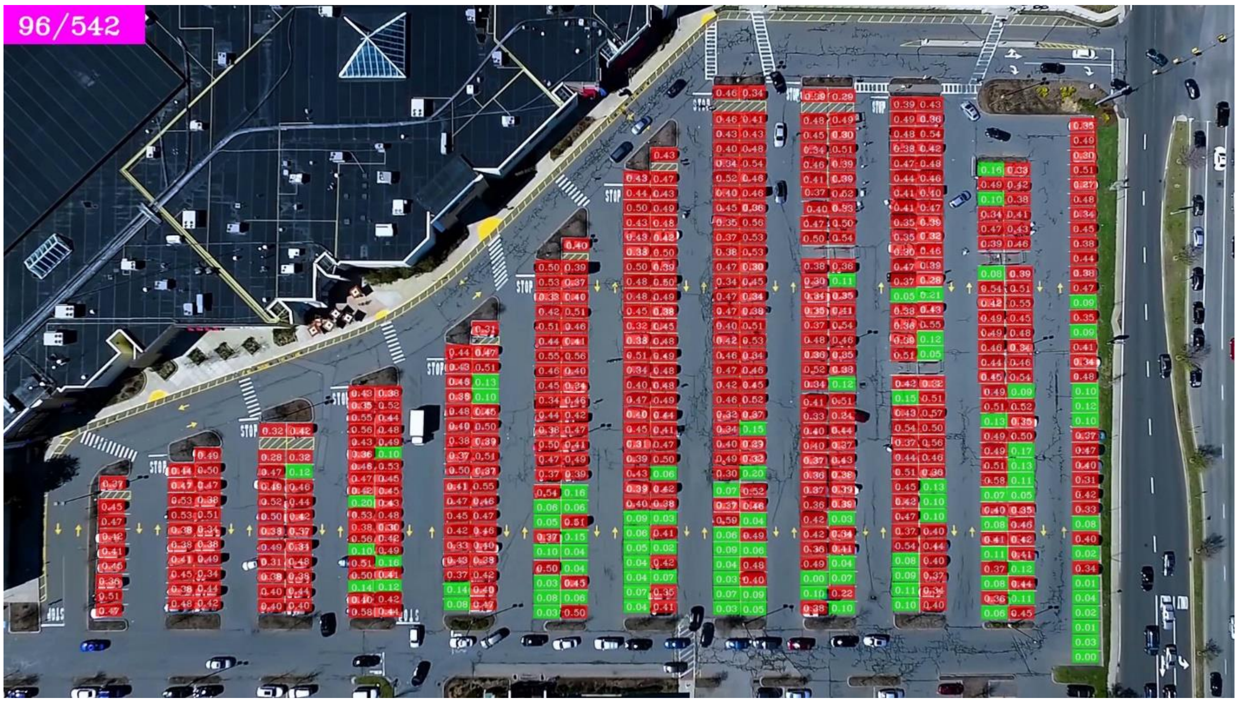


Figure 2: Empty and Occupied parking space detection



References:

https://www.youtube.com/watch?v=F-884J2mnOY&list=PLb49csYFtO2GunSHX38EjYHoYYbJYwrB-&index=1

https://www.youtube.com/watch?v=caKnQlCMIYI

https://github.com/computervisioneng/parking-space-counter

https://www.computervision.zone/courses/parking-space-counter/

PROJECT VİDEO LINK:

https://drive.google.com/drive/folders/10KUnRqdyJex0Hl7ikH6-_G2Nxvng_GsK