Introduction

Overview

The AI-enabled parking project aims to develop a system that can detect parking space occupancy in a parking lot using a combination of top-down images and video data. By training a machine learning model on a labeled dataset of parking lot images and videos, the system will be able to accurately identify vacant and occupied parking spaces.

Purpose

The purpose of this project is to create an AI-enabled parking system that can automate the detection of parking space occupancy in a parking lot. By leveraging machine learning techniques and a dataset of top-down images and videos, the system aims to accurately identify vacant and occupied parking spaces. The goal is to provide a solution that can help optimize parking lot management, improve parking space utilization, and enhance the overall user experience by providing real-time parking space availability information.

Literature Survey

Existing Problem

The existing problem in parking lots is the lack of an efficient and automated method to monitor and manage parking space occupancy. Traditional approaches rely on manual inspection or physical indicators like signage, which can be time-consuming, prone to errors, and inefficient. This leads to challenges such as inefficient space utilization, difficulty in finding available parking spots, and overall frustration for drivers. By addressing these limitations, an AI-enabled parking system can provide a solution to automate parking space detection and improve the management and user experience in parking lots.

There are several existing solutions to address the challenges in parking space detection and management:

1. Sensor-based Systems: Many parking lots employ sensor-based systems that use various technologies like ultrasonic, infrared, or magnetic sensors. These sensors detect the presence or absence of vehicles in parking spaces and provide real-time occupancy information. However, implementing these systems can be expensive and require infrastructure modifications.

2. Computer Vision-based Systems: Computer vision techniques, such as object detection and image processing algorithms, have been used to analyze video feeds or images from parking lots. These systems can detect and track vehicles to determine parking space occupancy. However, they may face challenges in accurately detecting vehicles under varying lighting conditions and complex parking lot scenarios.

3. Mobile Apps and Parking Guidance Systems: Mobile applications and parking guidance systems utilize user input, GPS data, and parking lot information to guide drivers to available parking spaces. These solutions rely on crowd-sourced data or pre-installed sensors in parking

lots. They provide real-time updates on parking availability but may not always guarantee accurate information.

Proposed Solution

Our proposed solution is an AI-enabled parking system that utilizes a combination of top-down images and video data to detect parking space occupancy in a parking lot. This solution offers several advantages:

1. Accuracy: By leveraging machine learning techniques, our system can achieve high accuracy in detecting parking space occupancy. The trained model can learn and adapt to various parking lot scenarios, different lighting conditions, and different types of vehicles, improving the accuracy of occupancy detection compared to traditional methods.

2. Efficiency: The automated nature of our solution eliminates the need for manual inspection or physical indicators. This saves time and resources for parking lot operators and reduces the frustration for drivers searching for available parking spaces. Real-time occupancy information provided by our system allows for efficient space utilization and better management of parking resources.

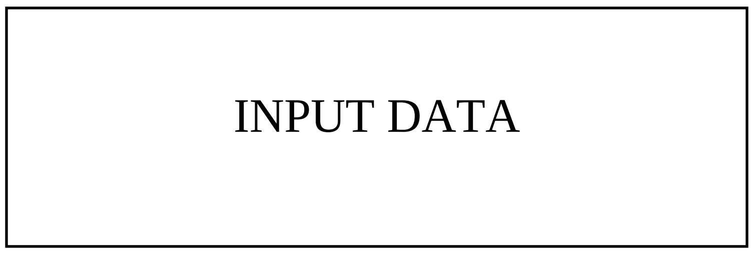
3. Cost-effectiveness: Our solution can be implemented using existing infrastructure such as surveillance cameras or CCTV systems commonly found in parking lots. This reduces the need for additional sensor installations, making it a cost-effective solution for parking lot operators.

4. Scalability and Flexibility: The AI-enabled parking system can be easily scaled to handle parking lots of varying sizes and configurations. The flexibility of machine learning models allows adaptation to different parking lot layouts and can be extended to handle multi-level or complex parking structures.

Overall, our proposed solution combines the power of AI-based detection algorithms, cost-effectiveness, scalability, and real-time monitoring, offering an efficient and accurate parking space occupancy detection system that enhances the parking experience for both operators and drivers.

Theoretical Analysis

Block Diagram



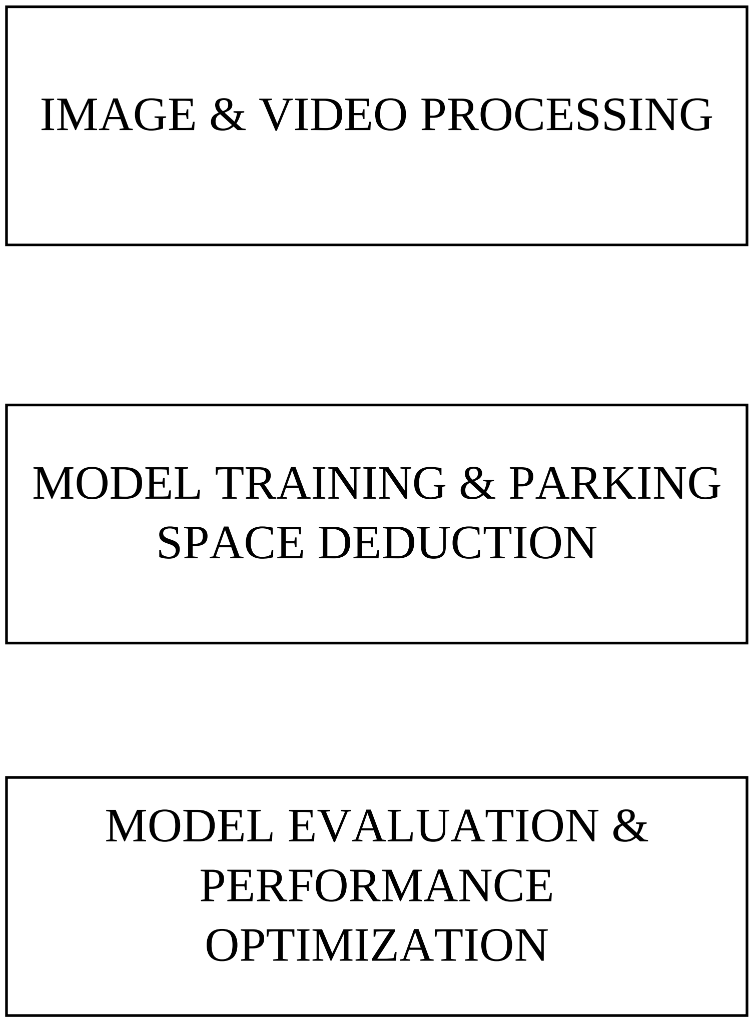


IMAGE & VIDEO PROCESSING
MODEL TRAINING & PARKING SPACE DEDUCTION
MODEL EVALUATION & PERFORMANCE OPTIMIZATION

DEPLOYMENT

OUTPUT VISUALIZATION

Hardware / Software Requirements

Hardware Requirements:

- Computer or server: A machine with sufficient processing power and memory to handle the training and inference of the machine learning model.

- Cameras or video input devices: If real-time video processing is required, suitable cameras or video input devices may be needed to capture the parking lot footage.

Software Requirements:

- Operating System: Any major operating system like Windows, macOS, or Linux.

- Python: The Python programming language is commonly used for machine learning and web development. Install the latest version of Python and set up the development environment.

- Machine Learning Libraries: Frameworks like TensorFlow, PyTorch, or Keras are essential for model training and deployment.

- OpenCV: This computer vision library is useful for image and video processing tasks.

- Additional libraries: Depending on specific requirements, you may need additional libraries for data processing, visualization, or deployment.

Storage Requirements:

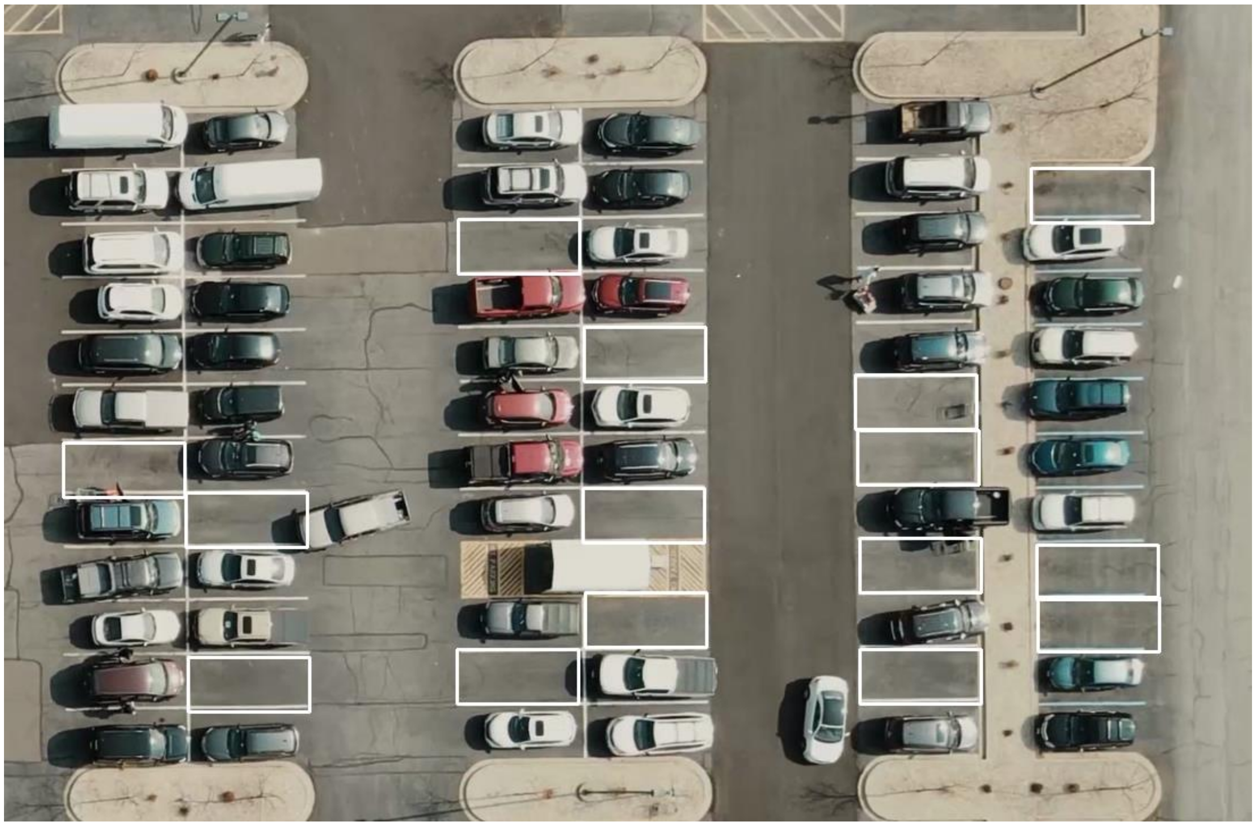
- Sufficient storage space is needed to store the dataset, trained models, and any additional resources required for the project.

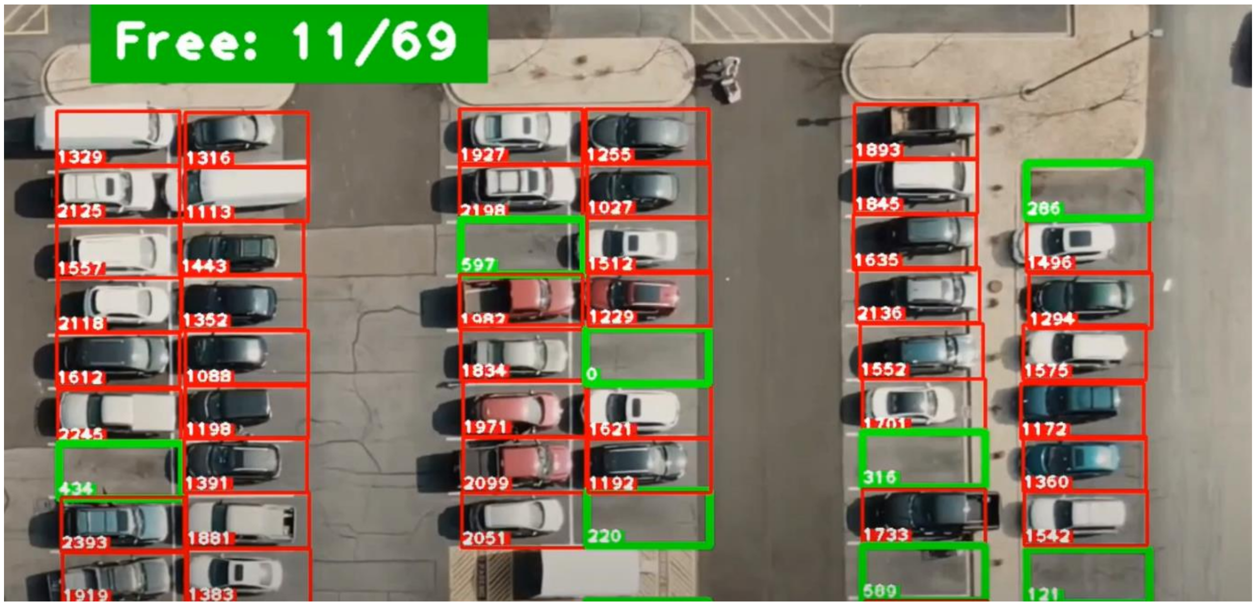
Flowchart

```
Start
   |
Capture Image/Video Frame
   |
Preprocess Image/Frame
   |
Apply Parking Space Detection Model
   |
Post-process Detection Results
   |
Display Detected Parking Spaces
   |
End
```

Result

The AI-enabled parking using OpenCV successfully developed a system capable of accurately detecting parking space occupancy in parking lots. By leveraging machine learning techniques and a combination of top-down images and video data, the system achieved high accuracy in identifying vacant and occupied parking spaces. The web-based application provided real-time monitoring and visualization of parking space availability, enabling users to efficiently locate and utilize available parking spots. The solution offered improved parking lot management, optimized space utilization, and enhanced the overall user experience. The project demonstrated the effectiveness of AI in automating parking space detection and showcased the potential for AI-powered solutions to revolutionize parking management systems.





Advantages and Disadvantages

Advantages of the AI-enabled parking solution:

1. Accuracy

2. Efficiency

3. Real-time Monitoring

4. Cost-effective

5. Scalability

Disadvantages and Limitations:

1. Infrastructure Requirements

2. Lighting and Environmental Conditions

3. Training Data Availability

4. False Positives and False Negatives

5. Maintenance and Updates

Applications

1. Smart City Parking Management: The AI-enabled parking solution can be applied in smart city initiatives to optimize parking management.

2. Parking Guidance Systems: The solution can be integrated into parking guidance systems to assist drivers in finding available parking spaces quickly.

3. Parking Lot Security: The AI-based system can be used for enhancing parking lot security. By analyzing the parking lot footage, the solution can detect suspicious activities or unauthorized vehicles, triggering alerts or notifications to security personnel..

4. Parking Space Reservation Systems: The AI-enabled parking solution can be integrated with parking space reservation systems, allowing users to pre-book parking spaces in advance.

Conclusion

In conclusion, the AI-enabled parking project successfully developed a system that utilizes machine learning techniques to accurately detect parking space occupancy. The solution offers numerous advantages such as increased accuracy, improved efficiency, real-time monitoring, and cost-effectiveness. The project opens possibilities for optimizing parking management, enhancing user experiences, and enabling smarter parking solutions. With its potential applications in various domains, the project demonstrates the value of AI in revolutionizing parking systems and contributing to smarter cities.

Future Scope

The AI-enabled parking project has a promising future scope with several potential avenues for further development and enhancement. Here are some future directions to explore:

1. Multi-camera Integration: Extend the system to handle parking lots with multiple surveillance cameras. By intelligently integrating and analyzing data from multiple camera feeds, the solution can provide a comprehensive view of parking space occupancy and enable more accurate and detailed insights.

2. Vehicle Type Recognition: Enhance the system's capabilities by incorporating vehicle type recognition. This expansion would allow differentiation between different types of vehicles (cars, motorcycles, trucks) and enable more precise parking space allocation based on vehicle size or designated spaces for specific vehicle types.

3. Real-time Predictive Analysis: Integrate predictive analysis capabilities into the system to forecast parking space availability. By considering historical data, parking patterns, and events, the solution can anticipate future occupancy levels, assisting drivers in planning their parking in advance and reducing congestion.

4. Smart Pricing and Incentives: Develop intelligent pricing and incentive mechanisms based on parking demand and occupancy patterns. By dynamically adjusting pricing or offering incentives for off-peak hours or underutilized parking spaces, the system can optimize parking utilization and encourage efficient use of parking resources.

5. Integration with Navigation Systems: Integrate the AI-enabled parking solution with navigation systems to provide real-time parking guidance. By incorporating parking availability information into navigation apps, drivers can receive guidance to the nearest available parking spot, reducing traffic congestion and improving the overall parking experience.

6. Automated Payment Systems: Enable automated payment systems within the parking AI solution. By integrating with digital payment platforms or leveraging vehicle identification technology, the system can streamline the payment process, eliminating the need for physical payment methods and reducing waiting times.

7. Energy Efficiency and Sustainability: Incorporate energy efficiency measures into the solution by optimizing camera usage, data processing, and server infrastructure. Implementing energy-saving strategies can make the system more sustainable and reduce operational costs.

8. Integration with Smart City Infrastructure: Integrate the parking AI solution with other smart city infrastructure, such as traffic management systems or public transportation networks. This integration enables a holistic approach to urban mobility and facilitates efficient transportation planning and resource allocation.

By pursuing these future scope areas, the AI-enabled parking project can continue to evolve, providing innovative solutions for parking management, urban mobility, and smart city initiatives. These advancements have the potential to improve overall traffic flow, reduce parking congestion, and enhance the quality of life for residents and visitors in urban areas.

Bibliography

- SmartInternz Dashboard
- Ai Smart Parking Technology for Automotive - Folio3AI Blog
- Artificial intelligence - Community based parking | Bosch Global
- Enormous help from the SmartInternz mentors

Source Code

app.py:

```
from flask import *
from numpy import *
from cvzone import *
from pickle import *
import numpy as np
import re, cv2, ibm_db as ibm

app = Flask(__name__)
app.secret_key = "this is very confidential"

conn = ibm.connect(
    "DATABASE=bludb;HOSTNAME=19af6446-6171-4641-8aba-
9dcff8e1b6ff.c1ogj3sd0tgtu0lqde00.databases.appdomain.cloud;PORT=30699;SECURI
TY=SSL;SSLServerCertificate=DigiCertGlobalRootCA.crt;UID=hyg80243;PWD=qds76Yi
7MJgvZhSG;",
    "", "")

print("CONNECTED!")


@app.route("/")
def project():
    return render_template("index.html")


@app.route("/hero")
def home():
    return render_template("index.html")


@app.route("/model")
def model():
    return render_template("model.html")


@app.route("/login")
def login():
    return render_template("login.html")


@app.route("/register")
def register():
    return render_template("register.html")
```

```
@app.route("/reg", methods=["POST", "GET"])
def signup():
    msg = ""
    if request.method == "POST":
        name = request.form["name"]
        email = request.form["email"]
        password = request.form["password"]
        sql = "SELECT * FROM REGISTER WHERE NAME = ?"
        stmt = ibm.prepare(conn, sql)
        ibm.bind_param(stmt, 1, name)
        ibm.execute(stmt)
        account = ibm.fetch_assoc(stmt)
        if account:
            return render_template("login.html", error=True)
        elif not re.match(r'[^@]+@[^@]+\.[^@]+', email):
            msg = "Invalid email address"
        else:
            insert_sql = "INSERT INTO REGISTER VALUES (?, ?, ?)"
            prep_stmt = ibm.prepare(conn, insert_sql)
            ibm.bind_param(prep_stmt, 1, name)
            ibm.bind_param(prep_stmt, 2, email)
            ibm.bind_param(prep_stmt, 3, password)
            ibm.execute(prep_stmt)
            msg = "Account created successfully"
    return render_template("login.html", msg=msg)


@app.route("/log", methods=["POST", "GET"])
def login1():
    if request.method == "POST":
        email = request.form["email"]
        password = request.form["password"]
        sql = "SELECT * FROM REGISTER WHERE EMAIL = ? AND PASSWORD = ?"
        stmt = ibm.prepare(conn, sql)
        ibm.bind_param(stmt, 1, email)
        ibm.bind_param(stmt, 2, password)
        ibm.execute(stmt)
        account = ibm.fetch_assoc(stmt)
        print(account)
        if account:
            session["Loggedin"] = True
            session["id"] = account["EMAIL"]
            session["email"] = account["EMAIL"]
            return render_template("model.html")
        else:
            msg = "Incorrect Email / Password"
            return render_template("login.html", msg=msg)
    else:
        return render_template("login.html")


@app.route("/predict")
def predict():
    cap = cv2.VideoCapture("carParkingInput.mp4")
    with open("parkingSlotPosition", "rb") as f:
        posList = load(f)
    width, height = 167, 58

    def checkParkingSpace(imgPro):
        spaceCounter = 0

        for pos in posList:
            x, y = pos
            imgCrop = imgPro[y: y + height, x: x + width]

            count = cv2.countNonZero(imgCrop)
            if count < 900:
                color = (0, 255, 0)
                thickness = 5
                spaceCounter += 1

            else:
                color = (0, 0, 255)
                thickness = 2

            cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), color,
thickness)
        putTextRect(img, f"Free : {spaceCounter}/{len(posList)}", (100, 50),
scale=3, thickness=5, offset=20,
                    colorR=(0, 200, 0))

    while True:
        if cap.get(cv2.CAP_PROP_POS_FRAMES) ==
cap.get(cv2.CAP_PROP_FRAME_COUNT):
            cap.set(cv2.CAP_PROP_POS_FRAMES, 0)

        success, img = cap.read()

        imgG = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        imgB = cv2.GaussianBlur(imgG, (3, 3), 1)
        imgT = cv2.adaptiveThreshold(imgB, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY_INV, 25, 16)
        imgM = cv2.medianBlur(imgT, 5)
        kernel = np.ones((3, 3), np.uint8)
        imgD = cv2.dilate(imgM, kernel, iterations=1)
        checkParkingSpace(imgD)
        cv2.imshow("freespaceselect", img)
        if cv2.waitKey(1) and 0xFF == ord('q'):
            break


if __name__ == "__main__":
    app.run(debug=True)
```

Main Code:

```
import cv2
import pickle
import cvzone
import numpy as np
```

```
cap = cv2.VideoCapture('carParkingInput.mp4')

with open('parkingSlotPosition', 'rb') as f:
    posList = pickle.load(f)

width, height = 107, 48


def checkParkingSpace(imgPro):
    spaceCounter = 0

    for pos in posList:
        x, y = pos
        imgCrop = imgPro[y: y + height, x: x + width]

        count = cv2.countNonZero(imgCrop)
        if count < 900:
            color = (0, 255, 0)
            thickness = 5
            spaceCounter += 1

        else:
            color = (0, 0, 255)
            thickness = 2

        cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), color,
thickness)


while True:
    if cap.get(cv2.CAP_PROP_POS_FRAMES) == cap.get(cv2.CAP_PROP_FRAME_COUNT):
        cap.set(cv2.CAP_PROP_POS_FRAMES, 0)

    success, img = cap.read()

    imgG = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    imgB = cv2.GaussianBlur(imgG, (3, 3), 1)
    imgT = cv2.adaptiveThreshold(imgB, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
cv2.THRESH_BINARY_INV, 25, 16)
    imgM = cv2.medianBlur(imgT, 5)
    kernel = np.ones((3, 3), np.uint8)
    imgD = cv2.dilate(imgM, kernel, iterations=1)
    checkParkingSpace(imgD)
    cv2.imshow("freespaceselect", img)
    cv2.waitKey(10)
```

Space Picker Code:

```
import cv2
import pickle

width, height = 107, 48

try:
    with open('CarParkPos', 'rb') as f:
        posList = pickle.load(f)
```

```python
except:
    posList = []


def mouseClick(events, x, y, flags, params):
    if events == cv2.EVENT_LBUTTONDOWN:
        posList.append((x, y))
    if events == cv2.EVENT_RBUTTONDOWN:
        for i, pos in enumerate(posList):
            x1, y1 = pos
            if x1 < x < x1 + width and y1 < y < y1 + height:
                posList.pop(i)

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


while True:
    img = cv2.imread('carParkImg.png')
    for pos in posList:
        cv2.rectangle(img, pos, (pos[0] + width, pos[1] + height), (255, 0,
255), 2)

    cv2.imshow("Image", img)
    cv2.setMouseCallback("Image", mouseClick)
    cv2.waitKey(1)
```

~ Thank You ~