Importações

O código começa importando bibliotecas essenciais para a operação, como:

- `cv2` para visão computacional.
- `pickle` para salvar e carregar posições.
- `numpy` para manipulação de arrays.
- `datetime` para lidar com datas e tempos.
- `tkinter` para criar uma interface gráfica.

```
import cv2
import pickle
import numpy as np
import datetime
from tkinter import messagebox, Tk
```

Configuração inicial

Função `show_popup()`: Essa função exibe uma janela pop-up perguntando ao usuário se ele deseja realizar uma nova configuração de vagas.

```
# Função que exibe uma janela pop-up
def show_popup():
    root = Tk()
    root.withdraw()
    result = messagebox.askyesno("Setup", "Deseja realizar uma nova con
    root.destroy()
    return result
```

Caso a opção de configuração seja selecionada, o código tenta carregar as posições das vagas de estacionamento a partir de um arquivo já existente (última calibração). Se o arquivo não for encontrado, uma lista vazia é criada.

```
# Resposta da janela pop-up de configuração
setup_needed = show_popup()

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

Definição das dimensões de vagas: Várias dimensões de vagas são pré-definidas para serem usadas no layout das vagas no estacionamento.

```
# Define dimensões
dimensions1 = (30, 60)
dimensions2 = (60, 120)
dimensions3 = (90, 180)
dimensions4 = (60, 30)
dimensions5 = (120, 60)
dimensions6 = (180, 90)
current_dimensions = dimensions1  # Começa com a primeira configuração
start_times = {}  # Armazena o tempo de inicio para cada vaga
count = 1
```

Função `mouseClick()`: Essa função gerencia as interações do usuário através dos cliques do mouse, permitindo adicionar, remover e alterar a dimensão das vagas de estacionamento.

- Adição de Vagas: Utiliza-se o clique esquerdo do mouse para adicionar uma vaga com as dimensões atuais.
- Remoção de Vagas: Utiliza-se o clique direito do mouse para remover uma vaga existente.
- Alteração das Dimensões: Utiliza-se o clique do meio do mouse para alternar entre diferentes predefinições de dimensões.

```
def mouseClick(events, x, y, flags, params):
    global current_dimensions
    global count

    if events == cv2.EVENT_LBUTTONDOWN:
        # Add parking spot with current dimensions
        posList.append((x, y, current_dimensions))
        with open('CarParkPos', 'wb') as f:
            pickle.dump(posList, f)
    elif events == cv2.EVENT_RBUTTONDOWN:
        for i, pos in enumerate(posList):
            if len(pos) == 3:  # Ensure pos has 3 elements
                px, py, (w, h) = pos
                if px < x < px + w and py < y < py + h:
                    posList.pop(i)
                    with open('CarParkPos', 'wb') as f:
                        pickle.dump(posList, f)
                    break
    elif events == cv2.EVENT_MBUTTONDOWN:
        count = (count % 6) + 1
        if count == 1:
            current_dimensions = dimensions1
        elif count == 2:
            current_dimensions = dimensions2
        elif count == 3:
            current_dimensions = dimensions3
        elif count == 4:
            current_dimensions = dimensions4
        elif count == 5:
            current_dimensions = dimensions5
        elif count == 6:
            current_dimensions = dimensions6

        print(f"Dimensões invertidas: largura={current_dimensions[0]},
```

Loop de configuração

O loop de configuração verifica a disponibilidade e ativa a câmera para permitir que o usuário adicione, remova ou rotacione vagas, desenhando as instruções e as vagas atuais na tela.

```
if setup_needed:
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Error: Unable to access the webcam.")
        exit()

    while True:
        success, frame = cap.read()
if not success:
print("Error: Unable to read from the webcam.")
break

# Draw instructions on the frame
cv2.putText(frame, f'Numero de vagas: {len(posList)}', (10, 50), cv2.FO
cv2.putText(frame, "Left Click: Adiciona vaga", (10, frame.shape[0] - 8
cv2.putText(frame, "Right Click: Remove vaga", (10, frame.shape[0] - 50
cv2.putText(frame, "Scroll Click: Rotaciona vaga", (10, frame.shape[0]

for pos in posList:
if len(pos) == 3:  # Ensure pos has 3 elements
    x, y, dimensions = pos
    w, h = dimensions
    cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 255), 2)

cv2.imshow("Estacionamento Setup", frame)
cv2.setMouseCallback("Estacionamento Setup", mouseClick)
key = cv2.waitKey(1) & 0xFF
if key == 13:  # Enter key
break

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

Função `checkParkingSpace()`

Essa função verifica as vagas de estacionamento, marcando-as como ocupadas ou livres, e desenha retângulos coloridos ao redor das vagas.

```
def checkParkingSpace(imgPro):
    spaceCounter = 0
    for pos in posList:
        if len(pos) == 3:  # Ensure pos has 3 elements
            x, y, dimensions = pos
            w, h = dimensions
            imgCrop = imgPro[y:y + h, x:x + w]
            count = cv2.countNonZero(imgCrop)
            if count < 500:
                color = (0, 255, 0)  # Verde (vaga livre)
                thickness = 1
                spaceCounter += 1
                start_times.pop((x, y, dimensions), None)
            else:
                color = (0, 255, 255)  # Amarela (vaga ocupada)
                thickness = 2
                if pos not in start_times:
                    start_times[pos] = datetime.datetime.now()
                elapsed_time = datetime.datetime.now() - start_times[po
                if elapsed_time.total_seconds() >= 10:
                    color = (0, 0, 255) if int(elapsed_time.total_secon
                elif elapsed_time.total_seconds() >= 5:
                    color = (0, 0, 255)  # Vermelha (vaga ocupada acima
            cv2.rectangle(img, (x, y), (x + w, y + h), color, thickness
            cv2.putText(img, str(count), (x, y + h - 3), cv2.FONT_HERSH

    cv2.putText(img, f'Free: {spaceCounter}/{len(posList)}', (10, 50),
    for pos, start_time in start_times.items():
        if len(pos) == 3:  # Garante que pos tenha 3 elementos
            x, y, dimensions = pos
            w, h = dimensions
            elapsed_time = datetime.datetime.now() - start_time
            cv2.putText(img, f"Time: {elapsed_time.seconds}s", (x, y -
```

Loop principal de detecção

O loop principal usa a webcam para capturar imagens em tempo real e aplicar uma série de transformações para identificar vagas de estacionamento livres e ocupadas. A detecção é feita em tempo real, e as vagas são atualizadas conforme necessário.

Finalização: O loop é interrompido quando a tecla "Esc" é pressionada, liberando a webcam e fechando todas as janelas abertas pelo OpenCV.

```
# Main parking detection loop
cap = cv2.VideoCapture(0)
if not cap.isOpened():
print("Error: Unable to access the webcam.")
exit()

while True:
success, img = cap.read()
if not success:
print("Error: Unable to read from the webcam.")
break
```

```
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
imgBlur = cv2.GaussianBlur(imgGray, (3, 3), 1)
imgThreshold = cv2.adaptiveThreshold(imgBlur, 255, cv2.ADAPTIVE_THRESH_
imgMedian = cv2.medianBlur(imgThreshold, 5)
kernel = np.ones((3, 3), np.uint8)
imgDilate = cv2.dilate(imgMedian, kernel, iterations=1)

checkParkingSpace(imgDilate)

cv2.imshow("Estacionamento", img)
key = cv2.waitKey(10) & 0xFF
if key == 27:  # Escape key
break

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