# Smart parking

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Activitate | Durata | Start | Responsabil | Resurse | Rezultat |
| Stadiu actual | 2 sapt. | 10.10.2022 | - | - |  |
| Arhitectura sistemului | 4 sapt. | 25.10.2022 | Roșanu Alexandu-Ilie |  |  |
| Dezvoltare experimentala | 4 sapt. | 22.11.2022 | Lungu Fernando-Marian |  |  |
| Testare | 2 sapt. | 20.12.2022 | Mercic Cristian |  |  |
| Concluzii | 2 sapt. | 5.12.2022 | Nistor Antonio-Iulian |  |  |

Parcarile inteligente sunt o soluție modernă la problema aglomerării în zonele urbane. Aceste parcări utilizează tehnologie avansată pentru a gestiona eficient fluxul de vehicule și a oferi locuri de parcare disponibile pentru cei care au nevoie.

Unul dintre avantajele principale ale parcărilor inteligente este că ele ajută la reducerea timpului petrecut căutând un loc de parcare. Sistemele de parcare inteligentă folosesc senzori și alte tehnologii pentru a monitoriza locurile de parcare disponibile și pentru a afișa aceste informații în timp real pentru soferi. Acest lucru permite soferilor să găsească rapid un loc de parcare disponibil, ceea ce înseamnă că nu trebuie să înconjoare blocuri întregi sau să se lupte cu alți soferi pentru un loc.

O altă beneficie a parcărilor inteligente este că ele pot ajuta la reducerea poluării și a emisiilor de dioxid de carbon. Atunci când soferii petrec mult timp căutând un loc de parcare, acest lucru duce la o creștere a emisiilor de dioxid de carbon din cauza mașinilor care rulează în gol. Cu ajutorul parcărilor inteligente, soferii pot găsi rapid un loc de parcare, ceea ce înseamnă că vor petrece mai puțin timp rulând în gol și, astfel, vor contribui la reducerea emisiilor de dioxid de carbon.

De asemenea, parcările inteligente pot contribui la creșterea siguranței în trafic. Sistemele de parcare inteligente pot monitoriza îndeaproape fluxul de vehicule și pot alerta soferii atunci când există pericole sau obstacole pe care ar trebui să le evite. De asemenea, aceste sisteme pot ajuta la prevenirea accidentelor și la reducerea congestiei în zonele aglomerate.

În concluzie, parcările inteligente sunt o soluție modernă la problema aglomerării în zonele urbane. Acestea oferă o serie de avantaje, cum ar fi reducerea timpului petrecut căutând un loc de parcare, reducerea poluării.

1. **Stadiu actual**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Denumire | Concluzie | Bibliografie |
| 1 | Smart City Data Analysis | S-a folosit limbajul Python pentru a analiza traficul cu ajutorul unor camere | https://sci-hub.se/10.1145/3279996.3280029 |
| 2 | Industry 4.0 Urban Mobility: goNpark Smart Parking  Tracking Module | Acest IoT este menit ca un  infrastructura informațională pentru apariția Industriei 4.0. Acesta poate  ajuta consiliul orașului să monitorizeze prețurile pe străzi pentru a încuraja  șoferii să evite fie zonele cu prețuri mari (și foarte aglomerate). | https://sci-hub.se/10.1145/3162957.3163042 |
| 3 | A Vision of Smart Traffic Infrastructure for  Traditional, Connected, and Autonomous Vehicles | Acest studiu depășește cu mult viziune și algoritm  evoluții în dezvoltarea de prototipuri, implementare și  testarea pe un sistem la scară largă din lumea reală. Disponibilitatea mișcărilor de întoarcere în timp continuu după tipul de vehicul (inclusiv pietoni) ne va permite  dezvoltarea strategiilor dinamice de operare a traficului (sincronizarea semnalului | https://sci-hub.se/10.1109/MetroCAD48866.2020.00008 |
| 4 | Low-Cost Smart Surveillance System  for Smart Cities | Această lucrare prezintă un sistem de supraveghere inteligent la preț redus care integrează analiza video  pentru detectarea mișcării în timp real. | https://sci-hub.se/10.1007/978-3-319-92052-8\_26 |
| 5 | System for Monitoring and Guarding Vehicles  on Parking Areas | Este clar că există mulți factori de mediu perturbatori, cum ar fi condițiile meteorologice nefavorabile, iluminarea redusă a scenei în timpul nopții etc., care pot afecta semnificativ  sistemul bazat pe video-detecție. În cazul nostru examinăm dependența  fiabilitatea sistemului privind rezoluția camerei și debitul de datecare este necesar pentru a furniza flux video brut în modul online.  Principalul avantaj al sistemului automatizat de monitorizare și pază a vehiculelor propus și implementat este flexibilitatea și varietatea terminalelor de afișare a ieșirii, descentralizarea și accesul la distanță în subsistemul de comunicații în timp real, ceea ce scade nevoia de serviciu de pază fizică (personală) în parcări auto. | https://sci-hub.se/10.1007/978-3-319-66251-0\_25 |
| 6 | Real‑time video fre/smoke detection based on CNN in antifre  surveillance systems | Scopul acestei lucrări este de a dezvolta dispozitive inteligente IoT pentru detectarea incendiului/fumului în medii interioare și exterioare. Algoritmul este simplu de proiectat și poate fi antrenat rapid. Soluția propusă a obținut rezultate promițătoare pentru acuratețe în comparație cu stadiul tehnicii. | https://sci-hub.se/10.1007/s11554-020-01044-0 |
| 7 | Video-Based Parking Occupancy Detection for Smart  Control System | Acest sistem propus de detectare a ocupării parcării se bazează pe detectarea obiectelor. Este proiectat un model de detectare a obiectelor folosind algoritmul YOLO v3 derivat din MobileNet v2. Acest model este ușor și poate funcționa pe sistemul încorporat Jetson TX2. Sistemul încorporat poate fi integrat cu ușurință în stâlpi de iluminare existenti, iar un dispozitiv care deservește multe grile de parcare poate reduce cerințele de întreținere. | https://sci-hub.se/10.3390/app10031079 |
| 8 | Implementation of Smart Parking Solution by Image Analysis | Rezultatele arată că precizia clasificării nu depinde în mod semnificativ de numărul de mostre de imagini utilizate  pentru antrenamentul modelului.Dezavantajul abordării de monitorizare a parcării bazată pe video este că, dacă mașina nu este parcata direct la lotul considerat, sistemul nu va detecta mașina corect. În cazul în care parcările sunt orientate diferit față de cameră, mai multe modele trebuie pregătite pentru fiecare orientare de parcare. | https://www.scitepress.org/Papers/2018/66297/66297.pdf |
| 9 | Smart Parking Guidance System Using 360o  Camera and Haar-Cascade Classifier on IoT  System | Obiectivul major al acestei cercetări a fost acela de a construi un sistem ușor și eficient de ghidare a parcării, care să poată fi utilizat pentru găsirea și rezervarea spațiului de parcare liber cu ajutorul lor mobil. În această încercare, un dispozitiv prototip care include o cameră de 360 de grade a fost modificat pentru a captura imagini și a fost creat un set de date în scopuri de instruire și testare. Utilizând metoda Haar-Cascade, a fost dezvoltat sistemul care este fiabil și capabil să detecteze prezența mașinii și disponibilitatea locului de parcare cu o precizie de 99,74% în medie, ceea ce este mult mai bun decât alte sisteme. | http://irep.iium.edu.my/79794/1/79794\_Smart%20parking%20guidance%20system.pdf |
| 10 | An Elaborative Study of Smart Parking Systems | Lucrarea oferă o comparație elaborată a cercetărilor disponibile privind sistemele de parcare inteligente și vorbește despre diverși algoritmi cu rezultatele, avantajele și dezavantajele acestora. Aceasta oferă clasificarea lucrărilor de cercetare pe baza metodologiilor utilizate. În această lucrare sunt implementati doi algoritmi pe două seturi de date și am măsurat performanța acestora în termeni de scor de precizie, scor de reamintire și scor F1. În urma analizei întregului studiu, se constată că sistemul de parcare inteligentă implementat folosind mai multe abordări va fi mai eficient și va avea o precizie ridicată în ceea ce privește rezultatele sale. | https://www.ijert.org/research/an-elaborative-study-of-smart-parking-systems-IJERTV10IS100056.pdf |

1. **Arhitectura sistemului**

Arhitectura sistemului pentru a detecta locurile de parcare dintr-un videoclip ar putea include urmatoarele componente:

Capturarea datelor: sistemul ar fi configurat pentru a captura si a prelucra datele din videoclip. Acest lucru ar putea include utilizarea unui camera sau a unui software pentru a prelua videoclipul.

Preprocesarea datelor: acest pas ar implica prelucrarea datelor din videoclip pentru a le face mai usor de analizat. Aceasta ar putea include redimensionarea imaginilor, eliminarea zgomotului sau a altor informatii inutile.

Detectarea locurilor de parcare: acest pas ar implica utilizarea algoritmilor de procesare a imaginilor sau algoritmi de analiza a imaginilor pentru a detecta zonele care ar putea fi locuri de parcare.

Analizarea datelor: aceasta etapa ar implica utilizarea algoritmilor de analiza a imaginilor pentru a determina daca zonele detectate sunt ocupate sau libere.

Raportarea rezultatelor: sistemul ar genera un raport cu privire la locurile de parcare detectate si starea lor(ocupat sau liber) si ar putea prezenta aceste informatii intr-o interfata utilizator sau prin alte mijloace cum ar fi afisaje sau notificari.

1. **Dezvoltare experimentala**

* Codul pentru desenarea dreptunghiului

import cv2  
import pickle  
  
width, height = 150,75  
  
try:  
 with open('CarParkPos1', 'rb') as f:  
 posList = pickle.load(f)  
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('CarParkPos1', 'wb') as f:  
 pickle.dump(posList, f)  
  
  
while True:  
 img = cv2.imread('image1.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)

Codul de mai sus foloseste biblioteca OpenCV pentru a afișa o imagine numită 'image1.png' și permite utilizatorului să selecteze regiuni specifice din imagine prin apăsarea și tragerea mouse-ului. Regiunile selectate sunt reprezentate prin cutii dreptunghiulare pe imagine. Pozițiile acestor cutii sunt stocate într-o listă numită 'posList'.

Codul foloseste si biblioteca pickle pentru a salva si incarca 'posList' in si din un fisier numit 'CarParkPos1'. Acest lucru permite utilizatorului sa salveze pozitiile cutiilor si sa continue sa lucreze la ele mai tarziu.

Functia mouseClick este apelată ori de câte ori utilizatorul apasă butonul stâng sau drept al mouse-ului. Dacă butonul stâng este apăsat, coordonatele curente ale mouse-ului sunt adăugate la 'posList' și dacă butonul drept este apăsat, codul verifică dacă coordonatele mouse-ului sunt în interiorul uneia dintre cutiile din 'posList' și, dacă este așa, elimină acea cutie din listă.

Bucla while afișează în mod continuu imaginea cu cutiile desenate pe ea și așteaptă ca utilizatorul să apese mouse-ul. Functia cv2.setMouseCallback seteaza functia mouseClick ca functie callback care este apelată atunci când mouse-ul este apăsat. Functia cv2.waitKey(1) este utilizată pentru a aștepta o apăsare a tastei și permite utilizatorului să închidă fereastra apăsând orice tastă.

* Codul pentru detectarea locurilor libere dintr-o parcare

import cv2  
import pickle  
import cvzone  
import numpy as np  
# Video feed  
cap = cv2.VideoCapture('video2\_480.mp4')  
  
with open('CarParkPos', 'rb') as f:  
 posList = pickle.load(f)  
  
width, height = 150, 150  
  
  
def checkParkingSpace(imgPro):  
 spaceCounter = 0  
  
 for pos in posList:  
 x, y = pos  
  
 imgCrop = imgPro[y:y + height, x:x + width]  
 # cv2.imshow(str(x \* y), imgCrop)  
 count = cv2.countNonZero(imgCrop)  
  
  
 if count < 1000:  
 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)  
 cvzone.putTextRect(img, str(count), (x, y + height - 3), scale=1,  
 thickness=2, offset=0, colorR=color)  
 with open("example.txt", "w") as file:  
 file.write(str(spaceCounter))  
 cvzone.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()  
 imgGray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
 imgBlur = cv2.GaussianBlur(imgGray, (3, 3), 1)  
 imgThreshold = cv2.adaptiveThreshold(imgBlur, 255, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C,  
 cv2.THRESH\_BINARY\_INV, 25, 16)  
 imgMedian = cv2.medianBlur(imgThreshold, 5)  
 kernel = np.ones((3, 3), np.uint8)  
 imgDilate = cv2.dilate(imgMedian, kernel, iterations=1)  
  
 checkParkingSpace(imgDilate)  
 cv2.imshow("Parcare 2", img)  
 # cv2.imshow("ImageBlur", imgBlur)  
 # cv2.imshow("ImageThres", imgMedian)  
 cv2.waitKey(10)

Codul începe prin a utiliza funcția VideoCapture din OpenCV pentru a captura video-ul din fișierul "video2\_480.mp4". Apoi, folosește biblioteca pickle pentru a deschide și a citi coordonatele spațiilor de parcare din fișierul "CarParkPos". Aceste coordonate vor fi utilizate mai târziu pentru a decupa imagini din video și a verifica dacă spațiile de parcare sunt ocupate sau libere.

După aceea, imaginea capturată din video este prelucrată prin mai multe tehnici de prelucrare a imaginilor. Primul pas este de a converti imaginea în tonuri de gri folosind funcția cvtColor din OpenCV. Apoi, se aplică un blur Gaussian folosind funcția GaussianBlur pentru a elimina zgomotul din imagine. Urmează etapa de thresholding, unde se utilizează funcția adaptiveThreshold pentru a converti imaginea în imagine binară. În continuare se face utilizarea unui filtru medianBlur pentru a elimina orice zgomot rămas.

Se aplica dilatarea imaginii prin utilizarea functiei dilate și a unui kernel (matrice) pentru a extinde zonele albe din imagine.

După ce imaginea este prelucrată, este apoi trecută la funcția "checkParkingSpace", care folosește coordonatele din fișierul "CarParkPos" pentru a decupa imaginea în mai multe zone care reprezintă spațiile de parcare individuale. Pentru fiecare decupaj, se numără numărul de pixeli care nu sunt zero și se compara cu un prag. Dacă numărul de pixeli este sub acest prag, se consideră că spațiul de parcare este liber, în caz contrar este considerat ocupat.

In acest cod, pixelii care sunt considerati "zero" sunt cei care au valoarea 0 in imaginea binara rezultata din thresholding. In procesul de thresholding, se compara fiecare pixel din imaginea originala cu o valoare de prag si daca valoarea pixelului este mai mica decat pragul, acesta este setat la 0, iar daca valoarea pixelului este mai mare decat pragul, acesta este setat la 255. Prin urmare, in acest cod, pixelii cu valoarea 0 sunt considerati "zero" si sunt utilizati pentru a determina daca un spatiu de parcare este ocupat sau liber.

Pentru fiecare decupaj, se desenează un dreptunghi în jurul acestuia și se afișează numărul de pixeli non-zero în interiorul dreptunghiului. În final, se scrie numărul de spații de parcare libere într-un fișier numit "example.txt" și se afișează imaginea prelucrată cu dreptunghiuri și numere pe ecran. Utilizatorul poate apasa pe tasta "q" pentru a opri executia programului.

* Cod aplicatie care afiseaza locurile de parcare

from kivy.uix.boxlayout import BoxLayout  
from kivy\_garden.mapview import MapView, MapMarker  
from kivy.app import App  
from kivy.uix.popup import Popup  
from kivy.uix.label import Label  
from kivy.uix.button import Button  
  
class MapViewApp(App):  
 def build(self):  
 mapview = MapView(zoom=15, lat=47.156275, lon=27.604345)  
 marker1 = MapMarker(lat=47.156275, lon=27.604345)  
 marker1.bind(on\_press=self.show\_data)  
 mapview.add\_marker(marker1)  
 marker2 = MapMarker(lat=47.154515, lon=27.596602) ##47.154515, 27.596602  
 marker2.bind(on\_press=self.show\_data1)  
 mapview.add\_marker(marker2)  
 return mapview  
  
 def show\_data1(self, instance):  
 # Create a button that closes the popup when pressed  
 button = Button(text='Back')  
 button.bind(on\_press=self.dismis\_markdata)  
 with open("ex.txt", "r") as file:  
 for line in reversed(file.readlines()):  
 numarlocuri1 = line  
 break  
 print(numarlocuri1)  
 # Create a BoxLayout to organize the widgets  
 layout = BoxLayout(orientation='vertical')  
  
 # Add the button and data to the BoxLayout  
 layout.add\_widget(Label(text='Parcare 1'))  
 layout.add\_widget(Label(text='Numar locuri:'))  
 layout.add\_widget(Label(text=numarlocuri1))  
 layout.add\_widget(button)  
  
 # Create the popup and set its content to the BoxLayout  
 self.popup = Popup(title='Marker Data', content=layout, size\_hint=(None, None), size=(200, 200))  
 self.popup.open()  
  
 def show\_data(self,instance):  
 # Create a button that closes the popup when pressed  
 button = Button(text='Back')  
 button.bind(on\_press=self.dismis\_markdata)  
 with open("example.txt", "r") as file:  
 for line in reversed(file.readlines()):  
 numarlocuri = line  
 break  
 print(numarlocuri)  
 # Create a BoxLayout to organize the widgets  
 layout = BoxLayout(orientation='vertical')  
  
 # Add the button and data to the BoxLayout  
 layout.add\_widget(Label(text='Parcare 2'))  
 layout.add\_widget(Label(text='Numar locuri:'))  
 layout.add\_widget(Label(text=numarlocuri))  
 layout.add\_widget(button)  
  
 # Create the popup and set its content to the BoxLayout  
 self.popup = Popup(title='Marker Data', content=layout, size\_hint=(None, None), size=(200, 200))  
 self.popup.open()  
  
  
  
 def dismis\_markdata(self, instance):  
 {  
 self.popup.dismiss()  
 }  
MapViewApp().run()

Codul de mai sus creeaza o aplicatie MapView folosind modulul kivy\_garden.mapview. Seteaza nivelul de zoom initial si coordonatele la o anumita locatie (latitudine: 47.156275, longitudine: 27.604345). Apoi creeaza doua obiecte MapMarker, unul pentru fiecare locatie si se leaga un eveniment on\_press fiecarui marcator. Atunci cand marcatoarele sunt apasate, apeleaza metodele show\_data si show\_data1 respectiv.

Metodele show\_data si show\_data1 deschid o fereastra Popup care afiseaza datele pentru fiecare marcator. De asemenea, creeaza un buton care inchide fereastra Popup cand este apasat. Metoda show\_data citeste date din fisierul numit "example.txt" iar metoda show\_data1 citeste date din fisierul numit "ex.txt". Apoi afiseaza datele in fereastra Popup. Metoda dismis\_markdata este apelata cand butonul este apasat si inchide fereastra Popup.

In ansamblu, acest cod creeaza o aplicatie MapView care afiseaza doua marcatoare pe o harta si deschide o fereastra Popup cu date pentru fiecare marcator atunci cand este apasat.

1. **Testare**

Pentru a testa acest program, trebuie urmati urmatorii pasi:

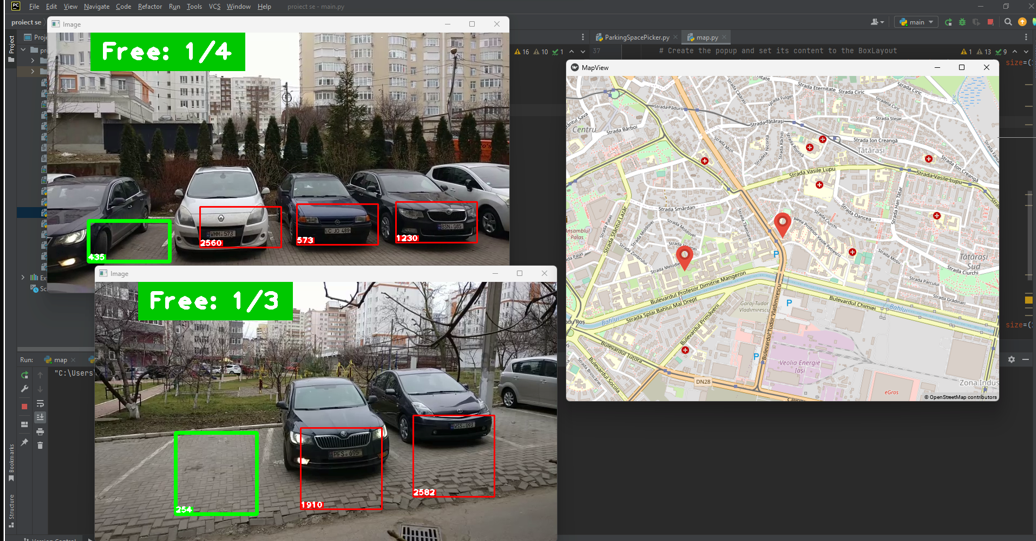
Ne asiguram ca avem toate librariile necesare instalate (OpenCV, pickle, cvzone, numpy).

Verificam daca exista un fisier cu numele 'CarParkPos1' in directorul in care ruleaza programul si daca exista, verificam daca acesta contine pozitiile locurilor de parcare. Daca nu exista, cream unul si adaugam pozitiile dorite sau putem utiliza mouse-ul pentru a marca pozitiile in timp ce programul ruleaza.

Ne asiguram ca exista un video cu numele 'video2\_480.mp4' in directorul in care ruleaza programul .

Rulam programul si verificam daca pozitiile locurilor de parcare sunt marcate corect in imagine si daca numarul de locuri libere este afisat corect. Putetim modifica parametrii din program pentru a vedea cum acestea afecteaza rezultatul final. De exemplu, putem modifica pragul de detectie pentru a vedea cum acest lucru afecteaza detectia locurilor libere. Observam cum se comporta programul in diferite conditii de iluminare si in cazul in care exista obstacole sau vehicule care se misca prin parcare.

Pentru testarea acestui am folosit mai multe imagini atat gasite online, cat si filmate de noi. Pentru fiecare din aceste imagini am modificat parametri pentru desenarea dreptunghiurilor si parametri in care un loc de parcare este liber sau nu.



1. **Concluzii**

In urma testarii am realizat faptul ca si codul ar putea fi imbunatatit. Imbunatirile pe care le-am putea aduce acestui proiect sunt:

* Folosirea unui trepied si o camera mai performanta pentru o calitate mai mare al inregistrarii si acuratete mai mare;
* Implementarea de machine learning pentru a inlatura necesitatea de a desena manual fiecare loc de parcare;
* Posibilitatea de adaugare a traseului pana la cea mai apropiata parcare;
* Dezvoltarea aplicatiei pentru a fi compatibila cu un telefon mobil.

Ce am reusit sa realizam:

* Sistemul detecteaza daca locul de parcare este ocupat sau nu prin analizarea imaginii;
* Trimiterea datelor spre o aplicatie externa cu scopul de a vizualiza in timp real parcarile disponibile;