SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2218

SUSTAV ZA NADZOR I UPRAVLJANJE JAVNIM POVRŠINAMA TEMELJEN NA RAČUNALNOM VIDU

Karlo Svetec

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2218

SUSTAV ZA NADZOR I UPRAVLJANJE JAVNIM POVRŠINAMA TEMELJEN NA RAČUNALNOM VIDU

Karlo Svetec

Zagreb, lipanj 2020.

SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Zagreb, 13. ožujka 2020.

DIPLOMSKI ZADATAK br. 2218

Pristupnik:	**Karlo Svetec (0036494454)**
Studij:	Računarstvo
Profil:	Računarska znanost
Mentor:	prof. dr. sc. Davor Petrinović

Zadatak: **Sustav za nadzor i upravljanje javnim površinama temeljen na računalnom vidu**

Opis zadatka:

U okviru diplomskog rada potrebno je istražiti algoritme računalnog vida koji omogućavaju promjenu perspektive slike prostora na temelju kombinacije ulaza s više kamera koje osiguravaju geometrijski različite poglede istog prostora. Iz tako rekonstruirane slike pogodno odabranog pogleda potrebno je provesti prepoznavanje objekata na proizvoljnim ili definiranim pozicijama. Primjer uporabe takvog sustava koji će biti obrađen u ovom radu je prepoznavanje parkiranih automobila i praznih mjesta na javnom ili privatnom parkiralištu, odnosno na javnim površinama koje nisu predviđene za parkiranje. Kako bi se ostvarila potrebna točnost rada, potrebno je riješiti probleme uzrokovane prirodnim pojavama koji su prisutni kod uporabe računalnog vida u vanjskom prostoru, a neki od njih su: različiti intenziteti svjetlosti i različit balans boja, ovisno o dobu dana ili godine, vremenske nepogode koje mijenjaju izgled prostora i općenito nekontrolirani uvjeti u vanjskom prostoru. Glavni dio sustava bit će statički pozicionirane kamere s pogledom na parkiralište, računalo zaduženo za obradu dobivene slike i web aplikacija zadužena za vizualizaciju podataka dobivenih od tog računala. Rezultat rada bit će autonomni sustav za nadzor parkirališta.

Rok za predaju rada: 30. lipnja 2020.

Sadržaj

Uvod

Jedan od najvećih problema infrastrukture modernih gradskih i prigradskih središta je dostupnost parkirnih mjesta i upravljanje javnim parkiralištima. U vrijeme prometne špice pronalazak parkirnog mjesta često je dugotrajan zadatak koji vozačima stvara probleme. Kod privatnih parkirališta u trgovačkim centrima i sličnim objektima situacija je u vrijeme gužve otprilike ista.

U radu je predloženo rješenje u obliku automatiziranog sustava za upravljanje parkiralištima koje bi korisnicima ponudilo pregled parkirališta u stvarnom vremenu kako bi dobili informaciju o dostupnim parkirnim mjestima, što bi značajno skratilo vrijeme pronalaska parkirnog mjesta.

Inicijalna ideja bila je napraviti sustav sastavljen od kombinacije ulaza sa više kamera koje osiguravaju geometrijski različite poglede istog prostora. Takva konfiguracija omogućila bi veću točnost sustava uslijed dodatne količine ulaznih informacija. Zbog tehničke zahtjevnosti i nemogućnosti izvedbe takvog rješenja u eksperimentalnom okruženju, implementacija se svela na iskorištavanje pogleda s jedne kamere. Unatoč tome, dobiveni su zadovoljavajući rezultati temeljeni na jednostavnijoj implementaciji.

Ovaj sustav bio bi alternativa trenutnim sustavima za upravljanje parkiralištima koji uobičajeno imaju senzore za svako parkirno mjesto. Cilj ovakve implementacije je smanjiti troškove i kompleksnost uvođenja sustava, ali i ponuditi novu dimenziju iskoristivosti u okviru prezentacije rezultata detekcije. S napretkom tehnologije ljudi su se sve više počeli oslanjati na mobilne uređaje za pristup raznim informacijama koje im pomažu u svakodnevnom životu. U tom duhu, rezultati detekcije ovog sustava predstavljaju se korisniku putem lako pristupačne Web aplikacije.

1. Komponente sustava

U ovom poglavlju dan je pregled komponenata sustava s visoke razine. Komponente sustava možemo podijeliti na sklopovske komponente i komponente programske podrške. Sklopovske komponente su:

- Ugradbeno računalo Raspberry Pi 3 Model B+
- Akcijska kamera SJCAM SJ4000
- Virtualni poslužitelj na platformi Google Cloud

Komponente programske podrške čine:

- Aplikacija za prijenos slike uživo sa Raspberry-a na poslužitelj
- Aplikacija za konfiguraciju parkirališta na poslužitelju
- Pozadinski proces za obradu slika sa kamere na poslužitelju
- Web aplikacija za vizualizaciju popunjenosti parkirališta
- Sql Server baza podataka

U nastavku je podrobniji opis svake komponente zasebno.

1.1. Sklopovske komponente

U nastavku je dan pregled dviju sklopovskih komponenata korištenih u izradi sustava (Raspberry Pi i kamera) i virtualnog poslužitelja koji nije čista sklopovska komponenta zbog prirode poslužitelja u oblaku, ali je uvršten ovdje jer su poslužitelji konvencionalno sklopovska oprema.

1.1.1. Raspberry Pi 3 Model B+

Ugradbeno računalo Raspberry Pi 3 Model B+ služi za dohvat slike sa kamere i prijenos slike putem mreže na poslužitelj u svrhu daljnje obrade. Ovo računalo izabrano je prvenstveno zbog svoje veličine koje ga čini praktičnim za primjenu u ugradbenim sustavima. Specifikacije računala su slijedeće [1]:

- Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64 bit SoC, 1.4 GHz

- 1 GB LPDDR2 SDRAM
- Ethernet port, WiFi

1.1.2. Akcijska kamera SJCAM SJ4000

U svrhu postizanja što veće preciznosti detekcije krajnjeg sustava, bilo je potrebno izabrati kvalitetnu kameru uz uvjet postizanja dobrog omjera uloženo/dobiveno. Izabrana je kamera kojoj je povezivanje na računalo tek sekundarna uloga, ali se pokazala kao optimalan izbor za prethodno navedene uvjete. Neke od karakteristika koje idu u prilog izboru su [2]:

- Rezolucija: 720p 1280*720 u PC Camera načinu rada
- Kut leće: 170°
    - omogućava obuhvaćanje veće površine parkirališta sa jednom kamerom
- Dimenzije: 59.2 x 29.8 x 41 mm
    - U skladu sa malim dimenzijama računala

1.1.3. Virtualni poslužitelj na platformi Google Cloud

Usko grlo sustava po pitanju potrebe za računalnom moći je detekcija i segmentacija automobila sa slike, koja je izvedena pomoću duboke konvolucijske neuronske mreže „Mask R-CNN“. Pošto Raspberry Pi nema dovoljno resursa da bi se detekcija izvela u ciljanom vremenu od maksimalno 5 sekundi po slici, bilo je potrebno dimenzionirati drugo računalo koje će obavljati obradu u tom vremenu. Izabran je virtualni poslužitelj na Google Cloud platformi sa slijedećim specifikacijama:

- 4 vCPU jezgri
- 15 GB RAM

Na poslužitelju je podešena statička IP adresa. Jedan razlog za to je što svaka instanca ugradbenog računala koja šalje slike prema poslužitelju mora znati gdje se nalazi poslužitelj, a drugi razlog je što je na poslužitelju objavljena i prezentacijska web aplikacija kojoj se također mora moći pristupiti uniformno.

U nastavku je prikazana iskorištenost resursa na poslužitelju sa aktivnim procesom za detekciju (detekcija je pokrenuta u 10 i 40):

Slika 1.1 Iskorištenost CPU-a tijekom detekcije

Slika 1.2 Iskorištenost mreže tijekom detekcije

1.2. Komponente programske podrške

Komponente programske podrške prema namjeni možemo podijeliti na dvije vrste: pozadinske servise i aplikaciju za interakciju s korisnikom. Pozadinski servisi ovog sustava su servis za prijenos slike, servis za obradu slike i detekciju te baza podataka. Web aplikacija zajedno sa servisom za dohvat podataka predstavlja sučelje prema korisniku. Konfiguracija se obavlja prije pokretanja servisa za detekciju i predstavlja preduvjet za uspješnu detekciju.

1.2.1. Servis za prijenos slike uživo sa Raspberry-a na poslužitelj

Ovo je jednostavan pozadinski servis pisan u Pythonu. Namjena mu je čitanje slike sa kamere koja je spojena na Raspberry Pi putem USB-a i slanje pročitane slike na poslužitelj putem mreže.

Na poslužitelju je implementiran modul zadužen za primanje slika sa Raspberry-a. Kako ne bi došlo do zasićenja ulaznog mrežnog spremnika uslijed različite frekvencije primanja slika i njihove obrade, u sklopu tog modula implementiran je interni spremnik u koji se spremaju dohvaćene slike kako bi se mrežni spremnik praznio kontinuirano. Taj spremnik osigurava da u svakom trenutku dohvaćamo najažurniju sliku sa kamere, dok se ostale brišu.

1.2.2. Aplikacija za konfiguraciju parkirališta na poslužitelju

Konfiguracija parkirališta podrazumijeva određivanje četverokutnih parkirnih regija sastavljenih od N parkirnih mjesta i definiranje parkirnih mjesta na tako određenim regijama. Korak definiranja parkirnih regija uveden je kako bi mogli dobiti pogled odozgora na svaku parkirnu regiju. To nam je važno zbog lakšeg označavanja parkirnih mjesta, pošto su sva mjesta na tako dobivenoj slici otprilike jednake veličine. Parkirna mjesta označena su da bi mogli vršiti usporedbu detektiranih automobila s njima. Svako parkirno mjesto je jedinstveno, a razaznajemo ih prema rednom broju koji se u aplikaciji određuje automatski.

Konfiguracija je izvedena direktno na poslužitelju u obliku samostalne Python aplikacije.

1.2.3. Pozadinski proces za obradu slika sa kamere na poslužitelju

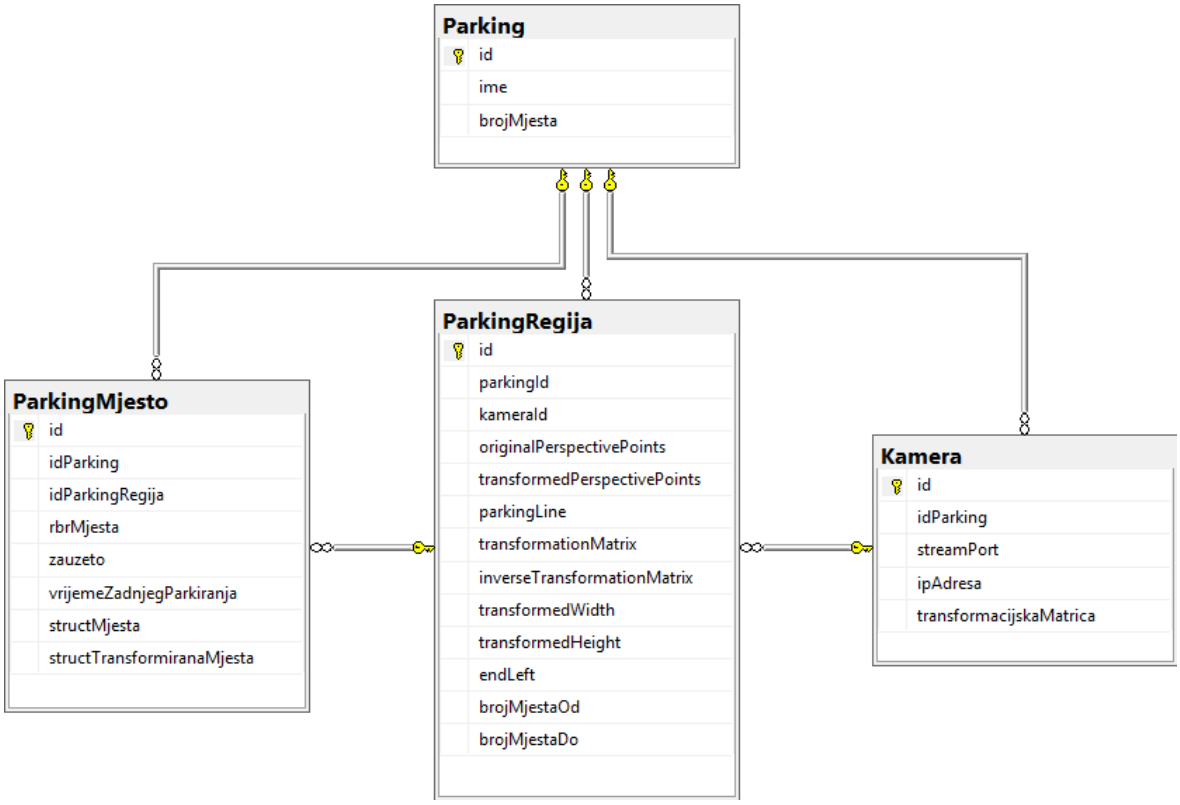
Na slikama dohvaćenim sa parkirališta potrebno je obaviti detekciju automobila i usporedbu lokacija automobila sa lokacijama parkirnih mjesta. Detekcija i segmentacija automobila implementirana je pomoću duboke konvolucijske neuronske mreže „Mask R-CNN“. Izlaz mreže su okviri detekcije za svaki od automobila na slici i binarne bit maske preko površine automobila. Nakon detekcije radi se geometrijska projekcija detektiranih okvira u ravninu parkirališta i usporedba s definiranim parkirnim mjestima. Izlaz iz samog procesa je rječnik parkirnih mjesta gdje je svakom mjestu pridijeljena oznaka popunjeno/prazno, broj zauzetih i ukupan broj mjesta te slika parkirališta sa označenim parkirnim mjestima. Prazna mjesta označena su zelenom bojom, a popunjena crvenom bojom.

1.2.4. Web aplikacija za vizualizaciju popunjenosti parkirališta

Korištenje sustava od strane korisnika osigurano je kroz Web aplikaciju koja nudi informacije o popunjenosti parkirališta kao i pogled uživo na parkiralište sa označenim popunjenim i praznim mjestima.

1.2.5. Sql Server baza podataka

Jednom konfiguriran sustav u pravilu ne treba više konfigurirati ukoliko ne dođe do pomaka kamere ili nekih drugih problema, što znači da se konfiguracijski parametri moraju spremiti u permanentnu memoriju. Osim konfiguracijskih parametra, tijekom rada sustava potrebno je spremati i detekcijske podatke kao i dnevnike događaja koji olakšavaju praćenje grešaka u sustavu. Za ovaj dio sustava izabrana je Sql Server baza podataka koja omogućava strukturiranu pohranu svih spomenutih podataka. U nastavku je prikazan ER model baze.



Slika 1.3 ER dijagram baze podataka

2. Pregled tehnologija korištenih za ostvarenje programske podrške

U realizaciji sustava bilo je potrebno odabrati tehnologije s kojima je moguće ostvariti obradu slike i primjenu algoritama računalnog vida na tim slikama, pohranu konfiguracijskih i detekcijskih podataka i prijenos slike sa ugradbenog računala na poslužitelj. Pored toga, bilo je potrebno odabrati programski jezik za izvedbu programske podrške i operacijske sustave za poslužitelj i ugradbeno računalo.

2.1. Obrada slike i primjena algoritama računalnog vida

Ovaj dio sustava moguće je realizirati pisanjem svojih algoritama ili korištenjem gotovih programskih modula koji znatno ubrzavaju razvoj. Dva najpopularnija alata kada govorimo o gotovim programskim modulima su Matlab sa svojim Computer Vision Toolboxom i OpenCV.

Neke od značajki Matlab Computer Vision Toolboxa su lakoća korištenja sa mnogo gotovih aplikacija za neke od uobičajenih zadataka (npr. kalibracija kamere, označavanje skupa podataka za treniranje). [5] Jedan od nedostataka je cijena s obzirom na to da je Matlab komercijalan alat. Matlab je također i programski jezik, pa je još jedan nedostatak potreba za prevođenjem Matlab koda u neki drugi jezik kako bi se algoritmi uklopili u sustav pisan u drugom jeziku.

S druge strane, OpenCV (Open Source Computer Vision Library) je biblioteka otvorenog koda pisana u programskom jeziku C++ u kojoj je implementirano nekoliko stotina algoritama računalnog vida i smatra se jednom od najobuhvatnijih biblioteka za računalni vid. [6] Nedostatak u odnosu na Matlab je dugotrajnija implementacija stvari koje u Matlabu dolaze kao gotov proizvod.

U ovom radu korišten je OpenCV u kombinaciji s Pythonom.

2.2. Pohrana konfiguracijskih i detekcijskih podataka

Zahtjevi na ovaj dio sustava bili su lakoća korištenja, mogućnost proširenja sustava na više parkirališta, zaštita od gubitka podataka i očuvanje integriteta podataka.

Dva razmatrana rješenja bila su korištenje datotečnog sustava uz spremanje podataka u datoteke ili korištenje baza podataka.

Dvije vrste podataka koje spremamo tijekom rada sustava su:

- Konfiguracijski podaci
    - Mala količina podataka, ugl. serijalizirani binarni podaci, rijetko se mijenjaju
- Izlazni podaci iz detektora
    - Promjene stanja parkirnih mjesta, dnevnici tih događaja, promjene su česte

Konfiguracijski podaci su pogodni za spremanje u datoteke, posebno kod sustava odgovornog samo za jedno parkiralište, međutim, baze podataka uveliko olakšavaju rad sa podacima velikih volumena koji se često mijenjaju.

Zbog navedenih svojstava, izabrana je Sql Server baza podataka u koju se spremaju svi podaci. Čak i u slučaju konfiguracijskih podataka, prednost baze nad datotekama je očuvanje integriteta podataka povezivanjem sa odgovarajućim ključevima na entitete koji predstavljaju pojedino parkiralište.

2.3. Prijenos slike sa ugradbenog računala na poslužitelj

Za pouzdanost sustava iznimno je bitan pravovremeni i robustan mehanizam prijenosa slike u stvarnome vremenu s parkirališta na poslužitelj. Prekid prijenosa doveo bi do prekida rada cijelog sustava, a kašnjenje slike uzrokovalo bi netočne informacije.

Tehnički zahtjevi za prijenos su: frekvencija prijenosa od najmanje 1 slike na 5 sekundi i što manji zahtjevi na propusnost mreže. Naime, parkirališta se često nalaze na lokacijama koje nemaju pristup širokopojasnom internetu, pa je potrebno implementirati prijenos preko mobilne mreže koja uobičajeno donosi i veće troškove.

Uzimajući u obzir sve prethodno navedene zahtjeve, za implementaciju prijenosa razmotreni su protokoli UDP i TCP.

UDP je bez konekcijski protokol koji ne garantira dostavu paketa na odredište, kao ni redoslijed dostave paketa. U praksi se često koristi kod prijenosa videa u stvarnom vremenu

gdje je iznimno bitno prenijeti što više sadržaja kako bi reprodukcija bila nesmetana, ali se dozvoljavaju greške prilikom prijenosa.

TCP je s druge strane konekcijski protokol koji garantira dostavljanje paketa na odredište i ispravan poredak poslanih paketa, što rezultira pouzdanijim prijenosom. TCP je generalno sporiji i troši više resursa od UDP protokola upravo zbog mehanizama koji osiguravaju navedena svojstva.

Imajući u vidu nisku frekvenciju prijenosa, odabran je TCP protokol, kako bi se pouzdanost sustava održala na visokoj razini, nauštrb malo veće količine generiranog prometa.

2.4. Programski jezik

Originalna implementacija OpenCV-a, najbitnijeg modula za rad sustava je u programskom jeziku C++, međutim, evolucijom navedenog modula napisani su omotači za razne programske jezike. U ovom radu razmotreni su omotači za Python i C#. Proučavanjem dokumentacije zaključeno je da je omotač za Python iznimno dobro dokumentiran i postoji velika zajednica korisnika koji koriste OpenCV u Pythonu, dok je omotač za C# slabije dokumentiran. Iz tog razloga, glavni dijelovi sustava (prijenos slike sa ugradbenog računala, konfiguracija sustava, detekcija i Web API) pisani su u Pythonu.

Inicijalna ideja bila je koristiti C# (u okviru ASP .Net Core razvojnog okvira) za pisanje Web API-a i Web aplikacije. ASP .Net Core je novi razvojni okvir otvorenog koda tvrtke Microsoft. Njegove glavne značajke su: visoke performanse i podrška za više platformi. [7] Web aplikacija je napisana u C#-u, a Web API je zbog jednostavnosti pisan u Pythonu u okviru Flask razvojnog okvira, kako bi se izbjegla potreba za među procesnom komunikacijom.

2.5. Operacijski sustavi

Na ugradbenom računalu Raspberry Pi instaliran je Raspberry Pi OS. To je službeni operacijski sustav za Raspberry Pi, pa je kao takav predstavljao logičan izbor po pitanju podrške i optimizacije za specifično sklopovlje. Sam operacijski sustav temelji se na Debianu, besplatan je i dolazi sa velikom količinom pred instaliranih paketa koji olakšavaju početnu konfiguraciju. [8]

Što se tiče poslužiteljskog dijela sustava, izabran je operacijski sustav Windows Server 2019 Datacenter Edition. Izbor ovog operacijskog sustava temelji se isključivo na jednostavnoj početnoj konfiguraciji, što omogućava brzo puštanje sustava u rad. Nedostatak je što je to komercijalan operacijski sustav koji nije besplatan, ali platforma Google Cloud nudi model plaćanja licence operacijskog sustava po satu korištenja. Na taj način može se optimizirati cijena gašenjem sustava po noći ako vidljivost na parkiralištu ne dozvoljava detekciju u to doba. Naravno, ovaj OS može se zamijeniti i sa nekim besplatnim sustavom u kasnijim iteracija razvoja jer su sve komponente pisane u jezicima koji podržavaju više platformi.

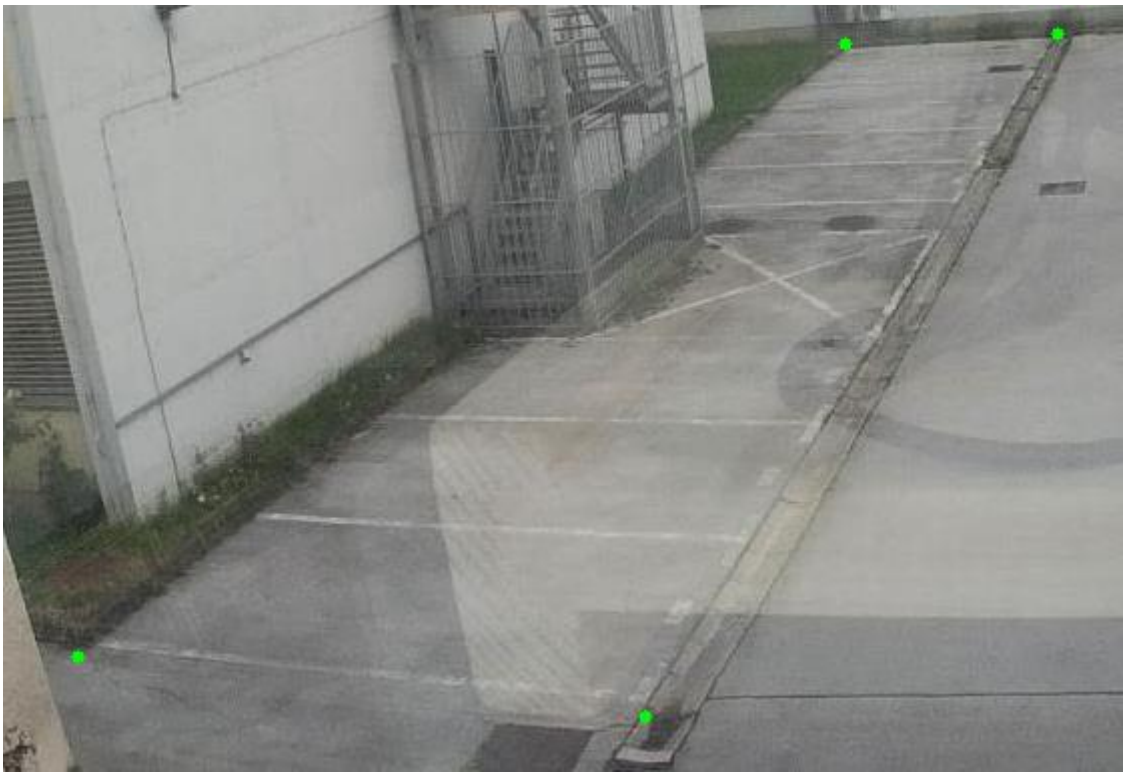
3. Konfiguracija sustava

Preduvjet za uspješnu detekciju zauzeća parkirnih mjesta je statička kamera za koju je implementirana ručna konfiguracija topografije parkirališta. Konfiguracija je izvedena u dva glavna koraka koji se mogu ponavljati. Za svaku kameru na parkiralištu određuje se potreban broj regija parkirnih mjesta u obliku konveksnog četverokuta. Nakon što je regija određena, napravljena je perspektivna transformacija te regije u ptičju perspektivu kako bi lakše označili parkirna mjesta. Parkirna mjesta su također numerirani konveksni četverokuti.

Konfiguracija je izvedena kao samostalna aplikacija i potrebno ju je izvršiti prije puštanja sustava u rad. Ako dođe do promjene u poziciji kamere, konfiguracija se može ponoviti u svakom trenutku, čime bi zamijenili prethodnu konfiguraciju.

3.1. Određivanje regije parkirališta

Kao što je prethodno spomenuto, svaka parkirališna regija predstavljena je u sustavu kao konveksni četverokut. Definicija regije sastoji se od odabira četiri točke koje čine četverokut, perspektivne transformacije tih točaka na novu sliku kako bi dobili pogled iz ptičje perspektive, vizualizacije rezultata konfiguracije i spremanja konfiguracijskih postavki u bazu podataka. Za definiciju regije zadužen je krajnji korisnik.



Slika 3.1 Slika parkirališta sa definiranim točkama parkirne regije

Slika 3.2 Transformirana regija parkinga sa prethodne slike

3.1.1. Perspektivna transformacija

Perspektivnom transformacijom parkirališne regije sa ulazne slike želimo dobiti pogled na regiju odozgora. Ovaj korak uveden je kako bi olakšali proces označavanja parkirnih mjesta. Kod pogleda iz ptičje perspektive sva mjesta imaju otprilike jednaku širinu, dok su iz perspektive kamere udaljena mjesta poprilično mala u odnosu na bliža mjesta, pa se time otežava njihovo definiranje.

Planarna perspektivna transformacija (projektivna transformacija, homografija) je linearna transformacija nad homogenim trodimenzionalnim vektorima ostvarena pomoću regularne matrice veličine 3x3 [4]. U nastavku su dane jednadžbe za perspektivnu transformaciju:

$$\boldsymbol{x'} = \boldsymbol{Hx} \tag{1}$$

$$\begin{pmatrix} x_1{}' \\ x_2{}' \\ x_3{}' \end{pmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{2}$$

Ako izmnožimo prethodnu jednadžbu i podijelimo sa homogenom koordinatom dobivamo:

$$x' = \frac{x'_1}{x'_3} = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}} \tag{3}$$

$$y' = \frac{x'_2}{x'_3} = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}} \tag{4}$$

Množenjem svake od jednadžbi s nazivnicima dobivamo:

$$x'(h_{31}x + h_{32}y + h_{33}) = h_{11}x + h_{12}y + h_{13} \tag{5}$$

$$y'(h_{31}x + h_{32}y + h_{33}) = h_{21}x + h_{22}y + h_{23} \tag{6}$$

Uvrštavanjem 4 para točaka u (5) i (6) imamo 8 nezavisnih linearnih jednadžbi čijim rješavanjem možemo dobiti elemente matrice **H** koji ostaju zavisni samo o jednom nepoznatom parametru. Međutim, matrica **H** je homogena pa nam jedna preostala nepoznanica definira samo homogenu koordinatu čija vrijednost nije bitna za rješenje sustava. [3]

Jedino svojstvo koje perspektivna transformacija čuva je ravnost linija (ravne linije ostaju ravne). Slijedeća svojstva nisu očuvana: orijentacija, dužine, kutovi i paralelne linije.

3.1.2. Implementacija konfiguracije regije

Korisniku prikazujemo sliku dohvaćenu sa kamere. Kada je korisnik odabrao četiri točke, spremamo ih u objekt tipa *ConvexQuadriliteral* koji je implementiran u okviru modula *geometry*. Razred *ConvexQuadriliteral* sprema sve točke konveksnog četverokuta i razvrstava ih po poziciji u četverokutu. (Kod 3.1)

```
self._points = points.copy()
points.sort(key=lambda x: x.getY())

self._topLeft = points[0]
if points[1].getX() < self._topLeft.getX():
    self._topLeft = points[1]
    self._topRight = points[0]
else:
    self._topRight = points[1]

    self._bottomLeft = points[2]
if points[3].getX() < self._bottomLeft.getX():
    self._bottomLeft = points[3]
    self._bottomRight = points[2]
else:
    self._bottomRight = points[3]
```

Kod 3.1 Inicijalizacija razreda *ConvexQuadriliteral*

Pozicioniranje točaka u četverokutu izvedeno je na slijedeći način: sortiramo sve točke prema y koordinati. Proglašavamo gornju lijevu točku točkom s najmanjom y koordinatom. To ne znači nužno da je točka s najmanjom y koordinatom gornja lijeva, ali znamo da su sigurno prve dvije točke s najmanjim y koordinatama gornje točke. Nakon toga provjeravamo ima li slijedeća točka (2. najmanja y koordinata) manju x koordinatu od trenutno proglašene gornje lijeve točke. Ako ima, nju proglašavamo lijevom točkom a prethodnu desnom točkom. Isti postupak ponavljamo za donje dvije točke.

Poredak točaka u četverokutu je iznimno bitan zbog određivanja perspektivne transformacijske matrice u pogled odozgora. Kako bi odredili transformacijsku matricu, prvo je potrebno odrediti veličinu nove slike koja nastaje primjenom te matrice na ulaznu sliku. (Kod 3.2)

```
def getMaxTopBottomCoordinateDistance(self):
      return max(self.coordinateDistance(self._topLeft,
      self._topRight),
      self.coordinateDistance(self._bottomLeft,
      self._bottomRight))

def getMaxLeftRightCoordinateDistance(self):
    return max(self.coordinateDistance(self._topLeft,
    self._bottomLeft),
    self.coordinateDistance(self._topRight,
    self._bottomRight))

@staticmethod
def coordinateDistance(point1, point2):
    return math.sqrt((point2.getX() - point1.getX())**2 +
    (point2.getY() - point1.getY())**2)
```

Kod 3.2 Određivanje veličine transformirane slike

Veličina se određuje pozivom gore navedenih funkcija. Visinu nove slike dobivamo tako da uzmemo parove točaka četverokuta (gornja lijeva, donja lijeva) i (gornja desna, donja desna) te za svaki par računamo udaljenost između točaka u koordinatnom sustavu pomoću formule (7).

$$dist((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_1 - y_2)^2} \tag{7}$$

Na isti način računa se i širina slike.

Kada smo odredili dimenzije nove slike, kreiramo listu točaka koje predstavljaju uglove nove slike. Nakon toga zovemo funkciju *getPerspectiveTransformation()* iz modula OpenCV, koja rješava sustav jednadžbi (5) i (6) i kao rezultat vraća transformacijsku matricu. (Kod 3.3) U ovom koraku vrlo je bitno da su točke na ulaznoj slici i točke u koje ih želimo transformirati zadane istim poretkom. Funkcija *getPointsAsNpArray()* razreda *ConvexQuadriliteral* osigurava nam dohvaćanje točaka u poretku: gornja lijeva, gornja desna, donja lijeva, donja desna. Na kraju zovemo funkciju *warpPerspective()* da bi primijenili dobivenu transformacijsku matricu na ulaznu sliku.

```
imagePoints = np.float32([[0, 0], [w, 0], [0, h], [w, h]])
if self._transformationMatrix is None:
```

```
    self._transformationMatrix =
cv2.getPerspectiveTransform(self._region.getPointsAsNpArray()
, imagePoints)
    transformed_image = cv2.warpPerspective(img,
self._transformationMatrix, (w, h))
```

Kod 3.3 Određivanje transformacijske matrice

3.2. Definiranje parkirnih mjesta

Nakon izvršenog postupka iz prethodnog potpoglavlja, regija parkirališta je definirana i prikazana u zasebnom prozoru. Slijedeći korak je definicija parkirnih mjesta na tako definiranoj regiji. Parkirna mjesta također se označavaju pomoću četiri točke i predstavljena su razredom *ConvexQuadriliteral.* Svako parkirno mjesto označeno je rednim brojem koji mu se pridjeljuje automatski. Time postižemo jedinstvenost svake instance parkirnog mjesta i omogućavamo spremanje konfiguracije parkirnih mjesta u bazu podataka. Razred koji predstavlja jedno parkirno mjesto naziva se *ParkingSpaceRepresentation.* On sadrži dvije instance razreda *ConvexQuadriliteral.* Jedna instanca predstavlja parkirno mjesto u perspektivi na kojoj je označeno mjesto (transformirana perspektiva parkirne regije), a druga je to isto mjesto u originalnoj perspektivi ulazne slike.

Kako bi dobili točke u originalnoj perspektivi, potrebno je izračunati inverz transformacijske matrice. Taj inverz dajemo kao parametar funkciji *perspectiveTransform* modula OpenCV, uz točke na transformiranoj slici, a kao rezultat dobivamo pozicije točaka na originalnoj slici.

Slika 3.3 Definiranje parkirnih mjesta

4. Detekcija i segmentacija automobila

Kako bi mogli odrediti popunjenost parkirnih mjesta, potrebno je identificirati sve automobile na parkiralištu i napraviti usporedbu njihovih lokacija sa lokacijama parkirnih mjesta. Identifikacija objekata na slici podijeljena je u 3 logička koraka. Prvi korak je detekcija objekata na slici. Za svaki od prepoznatih objekata želimo dobiti njegovu lokaciju. Rezultat detekcije su 4 točke koje definiraju pravokutnik koji omeđuje svaki od objekata. Kada smo dobili okvire objekata, slijedeći korak je klasifikacija tih objekata, odnosno svrstavanje svakog objekta u jednu od prethodno definiranih diskretnih klasa. Nas konkretno zanima klasa automobil. Posljednji korak je segmentacija. Segmentacijom postižemo razdvajanje objekata iste klase kako bi ih mogli razlikovati. Rezultat segmentacije je binarna bit maska preko vidljive površine objekta.

Za izvedbu detekcije i segmentacije izabrana je duboka konvolucijska neuronska mreža „Mask R-CNN“ koja u jednom prolazu mreže objedinjuje detekciju, segmentaciju i klasifikaciju. Taj model mreže nastao je kroz evoluciju originalne „R-CNN“ mreže, od čijeg ćemo opisa krenuti u nastavku.

4.1. Evolucija R-CNN-a

Razvoj dubokih konvolucijskih mreža donosi poboljšanje u klasifikaciji objekata sa slika u odnosu na konvencionalne metode. Primjer jedne od takvih mreža je AlexNet. [9] Ideja R-CNN mreže bila je nadograditi postojeću AlexNet mrežu funkcionalnošću detekcije pozicija objekata uz već razvijenu klasifikaciju.

Ulaz u R-CNN mrežu je slika, a izlaz su klasificirani okviri detekcija. Kako bi mreža znala gdje se nalaze objekti, prvi korak detekcije je predlaganje okvira za detekciju. Predlaganje okvira temelji se na procesu selektivne pretrage u kojem kroz sliku prolazimo prozorima raznih veličina i za svaki prozor pokušamo grupirati zajedno piksele prema teksturi, boji ili intenzitetu, kako bi dobili regiju koju zauzima jedan objekt. Dobiveni prijedlozi regija propuštaju se kroz modificiranu verziju AlexNet mreže. U predzadnjem sloju implementiran je SVM koji radi klasifikaciju. Za svaki okvir na izlazu iz SVM-a dobijemo klasifikaciju tog okvira sa vjerojatnošću predikcije. Zadnji korak mreže implementiran je kao model linearne regresije koji služi za podešavanje koordinati okvira detekcije kako bi bolje odgovarao stvarnoj veličini objekta i smanjio greške lokalizacije. [10]

4.1.1. Fast R-CNN

Fast R-CNN nastao je kao nadogradnja na R-CNN zbog poboljšanja brzine detekcije. Glavni problemi originalne mreže su [11]:

- Za svaki od predloženih okvira potreban je jedan prolaz mreže (u prosjeku 2000 prolaza po slici), što uvelike utječe na performanse detekcije
- Potrebno je trenirati tri zasebna modela: konvolucijsku mrežu za ekstrakciju značajki iz slike, SVM za klasifikaciju i model linearne regresije za smanjenje grešaka lokalizacije, što povećava kompleksnost treniranja

Kako bi doskočili problemu performansi, autori Fast R-CNN-a iskoristili su tehniku RoI pooling (Region of Interest pooling). Prednost ove tehnike je što zahtjeva samo jedan prolaz mreže za čitavu sliku. Nakon što je napravljen prolaz i dobivena mapa značajki za svaku klasu i čitavu sliku, uzimaju se prijedlozi okvira dobiveni na početku te se tehnikom RoI pooling pronalaze okviri detekcije.

4.1.2. Faster R-CNN

Fast R-CNN mreža djelomično je riješila problem performansi iz originalne mreže, međutim, ostalo je još jedno usko grlo vezano uz performanse, a to je algoritam selektivne pretrage za predlaganje okvira detekcija. Nakon razmatranja, autori mreže shvatili su da mogu riješiti problem izbacivanjem tog algoritma, uz iskorištavanje činjenice da prijedlozi okvira ovise o značajkama slike koje su već izračunate u prolazu mreže. Na izlazu mreže dodana je konvolucijska mreža nazvana Region Proposal Network (RPN). RPN radi na način da prolazi klizećim prozorom po slici i za svaku lokaciju prozora generira okvire detekcije i rezultat koji govori koliko je koji okvir detekcije dobar. Ideja iza okvira detekcije koji se generiraju u ovoj mreži je da uzmemo česte omjere veličina okvira kako bi smanjili prostor pretraživanja i radimo RoI pooling za svaki od takvih okvira kako bi dobili rezultate detekcije. Uporište ideje je u tome da objekti koje želimo detektirati obično imaju neke standardne omjere veličina okvira (npr. malo je vjerojatno da neki objekt zauzima visoki okvir širine 1 piksel). [12]

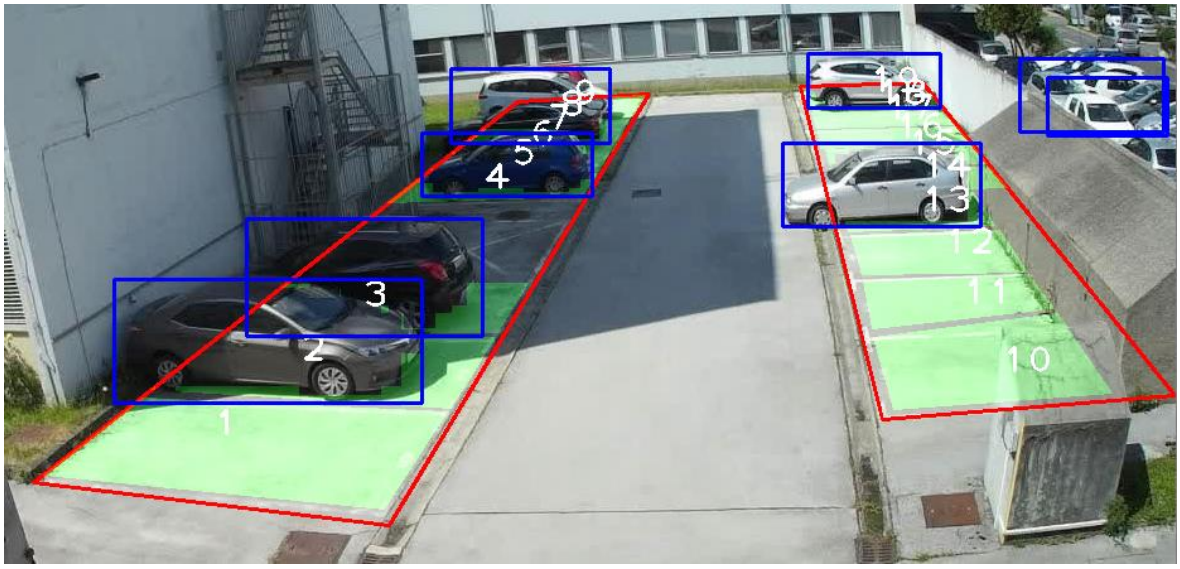
4.1.3. Mask R-CNN

Zadnja iteracija evolucije R-CNN-a je Mask R-CNN. U ovoj mreži dodana je još jedna konvolucijska neuronska mreža koja služi za generiranje binarne bit maske preko vidljive površine detektiranog objekta. [13]

Što se tiče detekcijskog dijela sustava, korištena je upravo Mask R-CNN mreža. Sama mogućnost segmentacije iskorištena je u prezentacijskom sloju kako bi se maska parkirnih mjesta prikazala samo na parkiralištu, a ne na automobilima. U idućem poglavlju detaljno je opisan način na koji se detektirani okviri automobila uspoređuju sa definiranim parkirnim mjestima kako bi dobili zauzeće svakog parkirnog mjesta.

5. Određivanje stanja parkirnog mjesta

Slijedeći korak koji bi trebao zaokružiti detekcijski dio sustava je usporedba rezultata detekcije automobila sa definiranim parkirnim mjestima. Rezultat usporedbe trebao bi biti postotak zauzeća svakog parkirnog mjesta. Vidimo da se okviri detektiranih automobila nalaze u ravnini slike, a parkirna mjesta su označena u ravnini parkirališta (Slika 5.1). Ta razlika u perspektivama nam predstavlja problem jer ne možemo napraviti direktnu usporedbu detekcija i parkirnih mjesta. Kako bi premostili ovaj problem, potrebno je napraviti projekciju detektiranih okvira u ravninu parkirališta. Kada bi imali kameru postavljenu direktno iznad parkirališta, ovoga problema ne bi bilo jer bi nam ravnina slike ujedno bila i ravnina parkirališta, međutim, ova izvedba je u većini primjena nerealistična jer zahtjeva nešto kompliciraniju infrastrukturu za postavljanje kamere. U nastavku je predložen pristup rješenju ovog problema sa pozicijom kamere pod malim kutom u odnosu na ravninu parkirališta.



Slika 5.1 Prikaz detekcije automobila, regija parkirališta i parkirnih mjesta

5.1. Projekcija detektiranog okvira na ravninu parkirališta

Glavna pretpostavka na kojoj se temelji osmišljeni pristup projekcije je da je širina svakog detektiranog automobila na određenoj poziciji u prostoru otprilike slična kao i širina njemu najbližeg parkirnog mjesta. Kako bismo definirali projekciju, prvo definiramo dva pravca koji određuju smjer parkirne regije (pravac koji prolazi po svim vanjskim rubovima

parkirnih mjesta). Jedan pravac spaja donji lijevi i gornji lijevi ugao regije, a drugi spaja donji desni i gornji desni ugao regije. U kodu je pravac definiran razredom *Line* u modulu *geometry*, a sadrži jednu točku koja pripada pravcu i koeficijent smjera pravca. Koeficijent smjera opisuje nagib pravca u odnosu na pozitivan dio x osi. Iz njega možemo dobiti kut koji pravac zatvara sa pozitivnim dijelom x osi. Nadalje, iz toga kuta možemo dobiti kut koji pravci zatvaraju sa osi y.

Slijedeći korak je pronalazak kandidata parkirnih mjesta za usporedbu sa detekcijskim okvirima automobila. Za svaki od okvira određujemo u kojoj parkirnoj regiji se nalazi.

```
dict = []
for idx, region in enumerate(regions):
    dist =
      ConvexQuadrilateral.coordinateDistance(
      boundingBox.topLeft(),
      region.originalPerspectivePoints.topLeft()) +
      ConvexQuadrilateral.coordinateDistance(
      boundingBox.topRight(),
      region.originalPerspectivePoints.topRight()) +
      ConvexQuadrilateral.coordinateDistance(
      boundingBox.bottomLeft(),
      region.originalPerspectivePoints.bottomLeft()) +
      ConvexQuadrilateral.coordinateDistance(
      boundingBox.bottomRight(),
      region.originalPerspectivePoints.bottomRight())

    dict.append([idx, dist])
dict.sort(key=lambda x: x[1])
return dict[0][0]
```

Kod 5.1 Funkcija *getRegionForBoundingBox*

Prolaskom kroz sve regije računamo udaljenosti svakog od uglova detekcijskog okvira automobila do odgovarajućih uglova parkirne regije. Nakon toga sortiramo te udaljenosti rastućim redoslijedom i vraćamo identifikator regije sa najmanjom udaljenosti do detekcijskog okvira. Kada znamo kojoj regiji pripada okvir, idući korak je traženje odgovarajućeg parkirnog mjesta. Temeljem kuta kamere i orijentacije parkirališta u odnosu na kameru možemo pretpostaviti da bi donji uglovi detekcijskog okvira trebali biti u blizini bližeg ruba parkirnog mjesta u odnosu na kameru. Kod za pronalazak parkirnog mjesta u odnosu na okvir detekcije sličan je kao i kod za pronalazak regije, jedina razlika je što

gledamo samo udaljenosti donjeg lijevog i desnog ugla detekcije do uglova ruba parkirnog mjesta koji je bliže kameri. Kada smo pronašli odgovarajuće parkirno mjesto, računamo njegovu širinu. (Kod 5.2)

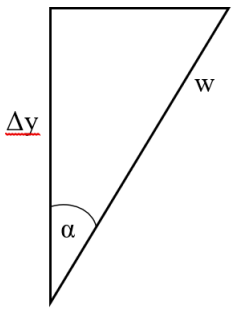
```
def getSpaceWidthInPixels(space):
    return
      int((ConvexQuadrilateral.coordinateDistance(space.struc
      tTransformiranjaMjesta.topRight(),
      space.structTransformiranjaMjesta.bottomRight()) +
      ConvexQuadrilateral.coordinateDistance(space.structTran
      sformiranjaMjesta.topLeft(),
      space.structTransformiranjaMjesta.bottomLeft())) / 2)
```

Kod 5.2 Određivanje širine parkirnog mjesta

Konačno, želimo proširiti lijevi i desni donji rub detekcijskog okvira kako bi dobili okvir u ravnini parkirališta. Ideja je da uzmemo prethodno izračunate koeficijente smjera bočnih pravaca parkirne regije i na temelju njih odredimo dvije nove točke detekcijskog okvira. Nove točke nalaze se na pravcima koji su određeni donjim lijevim i desnim uglovima detekcijskog okvira i prethodno spomenutim koeficijentima smjera. Udaljenost od tih točaka do novih točaka biti će upravo širina parkirnog mjesta. Jednadžba pravca zadana jednom točkom i koeficijentom smjera glasi:

$$y - y_1 = k(x - x_1) \tag{8}$$

Da bi dobili novu točku na temelju prethodno opisanih podataka, prvo je potrebno izračunati y koordinatu nove točke. Ta koordinata ovisi o širini parkirnog mjesta.



Slika 5.2 Ovisnost promjene y koordinate o širini parkirnog mjesta

Hipotenuza pravokutnog trokuta leži na pravcu sa prethodno definiranim koeficijentom smjera (koeficijent smjera jednog od bočnih pravaca parkirne regije), w je širina parkirnog mjesta, a Δy je promjena y koordinate od trenutne točke do točke koju računamo (Slika 5.2).

Koeficijent smjera definiran je kao tangens kuta koji pravac zatvara sa pozitivnim dijelom osi x. Slijedi formula za računanje kuta α:

$$\alpha = \pi - \tan^{-1}(k) \tag{9}$$

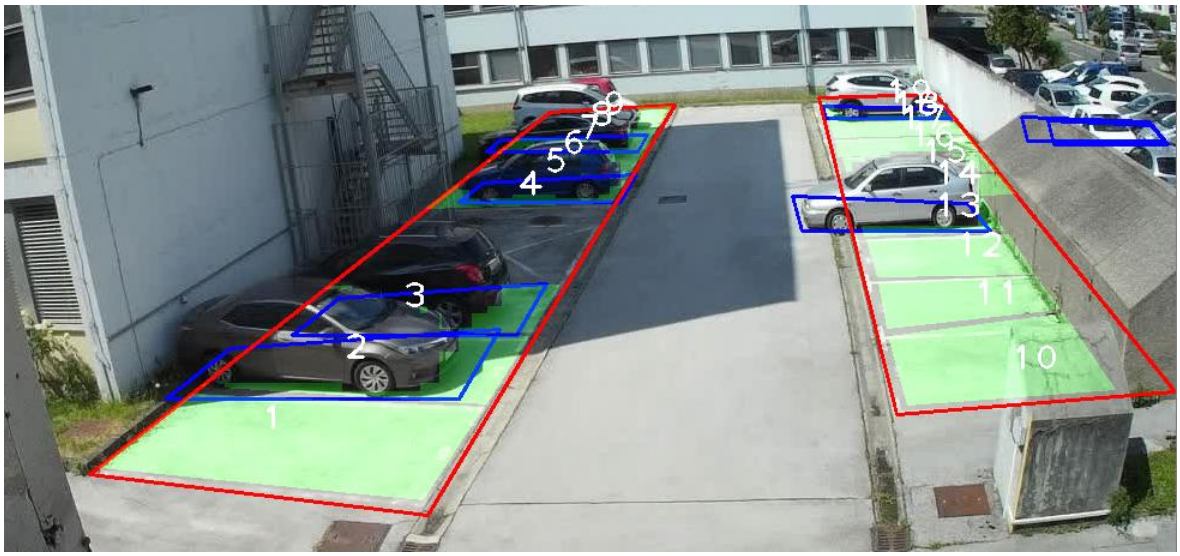
Iz pravokutnog trokuta vidimo ovisnost promjene y koordinate o širini parkirnog mjesta i kutu α:

$$\Delta y = |w \cos \alpha| \tag{10}$$

Dobivenu promjenu y koordinate oduzimamo od početne točke okvira detekcije kako bi dobili y koordinatu nove točke. Ovdje vrijedi spomenuti da je koordinatni sustav slike na računalu dijelom različit od uobičajenog Kartezijevog koordinatnog sustava. Ishodište je u gornjoj lijevoj točci, x koordinate rastu na desno, a y koordinate rastu prema dolje. Zbog toga promjenu y koordinate oduzimamo od početne točke, dok bi u Kartezijevom sustavu sa ishodištem u donjoj lijevoj točci zbrajali tu promjenu. Novu y koordinatu uvrštavamo u jednadžbu pravca iz koje smo izrazili x:

$$x = \frac{y - y_1 + kx_1}{k} \tag{11}$$

te smo na taj način izračunali x i y koordinate nove točke.



Slika 5.3 Rezultat projekcije

Iz rezultata projekcije vidljivo je da je ovom metodom postignuto dobro preklapanje parkirnih mjesta i odgovarajućih projekcija detekcijskih okvira (Slika 5.3).

5.2. Usporedba projekcije okvira s parkirnim mjestom

Popunjenost parkirnog mjesta definiramo kao kvocijent površine presjeka okvira detekcije sa površinom parkirnog mjesta i površine parkirnog mjesta.

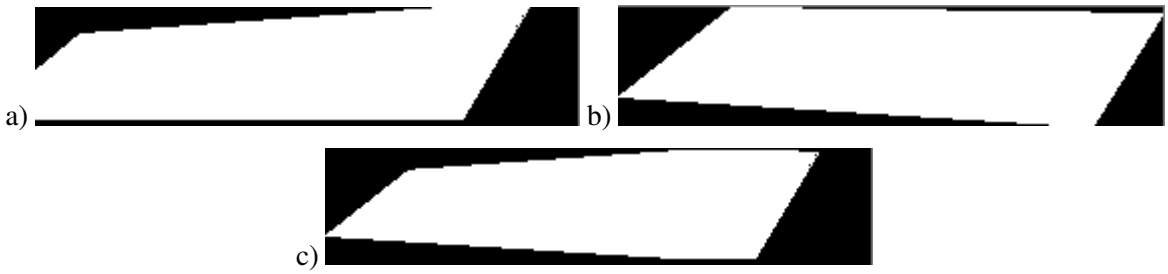
```
for i in (spaceId - 1, spaceId, spaceId + 1):
    if i < 0 or i >= len(spaces[id]):
        continue

    detectionProjection =
      ConvexQuadrilateral([boundingBox.bottomRight(),
      boundingBox.bottomLeft(), Point(leftUpperX,
      leftUpperY), Point(rightUpperX, rightUpperY)])
    h, w, translation =
      findComparisonImageSize(detectionProjection,
      spaces[id][i].structTransformiranjaMjesta)

    detection_mask = np.zeros((h, w, 1), dtype=np.uint8)
    space_mask = np.zeros((h, w, 1), dtype=np.uint8)
    cv2.fillPoly(detection_mask,
      [np.subtract(detectionProjection.getPointsAsNpIntArray(
      ), translation)], 255)
    cv2.fillPoly(space_mask,
      [np.subtract(spaces[id][i].structTransformiranjaMjesta.
      getPointsAsNpIntArray(), translation)], 255)

    space_area = np.count_nonzero(space_mask)
    res = cv2.bitwise_and(detection_mask, space_mask)
    intersection_area = np.count_nonzero(res)

    popunjeno = intersection_area / space_area
```

Kod 5.3 Izračunavanje popunjenosti parkirnog mjesta

Kako bi usporedili okvir detekcije i parkirno mjesto, kreiramo dvije binarne crno bijele slike koje predstavljaju okvir detekcije i parkirno mjesto (Slika 5.4). Binarna slika je karakteristična po tome što ima samo dvije moguće vrijednosti piksela. Površina objekata predstavljena je bijelim pikselima (vrijednost 255), a ostatak slike je crn (vrijednost 0). Veličinu tih slika računamo kao veličinu parkirnog mjesta, pošto nam je jedino njegova površina bitna, odnosno onaj dio detektiranog okvira koji pokriva tu površinu.

Površinu parkirnog mjesta u pikselima računamo kao broj piksela koji nije crn (a to su svi bijeli pikseli). Nakon toga radimo logičku funkciju „i“ između slika. Rezultat te operacije je presjek površina okvira detekcije i parkirnog mjesta. Površinu presjeka računamo na isti način kao i površinu mjesta te ih na kraju podijelimo da bi dobili zauzeće parkirnog mjesta. To je broj u intervalu [0, 1].

Ovdje vrijedi napomenuti i da pridjeljivanje parkirnog mjesta okviru detekcije iz prethodnog koraka ne radi uvijek savršeno zbog nesavršenosti algoritma za detekciju. Iz tog razloga u koraku usporedbe uz parkirno mjesto iz prethodnog koraka uspoređujemo i njemu susjedna parkirna mjesta. Na kraju uzimamo mjesto sa najvećim zauzećem i njega označimo zauzetim.

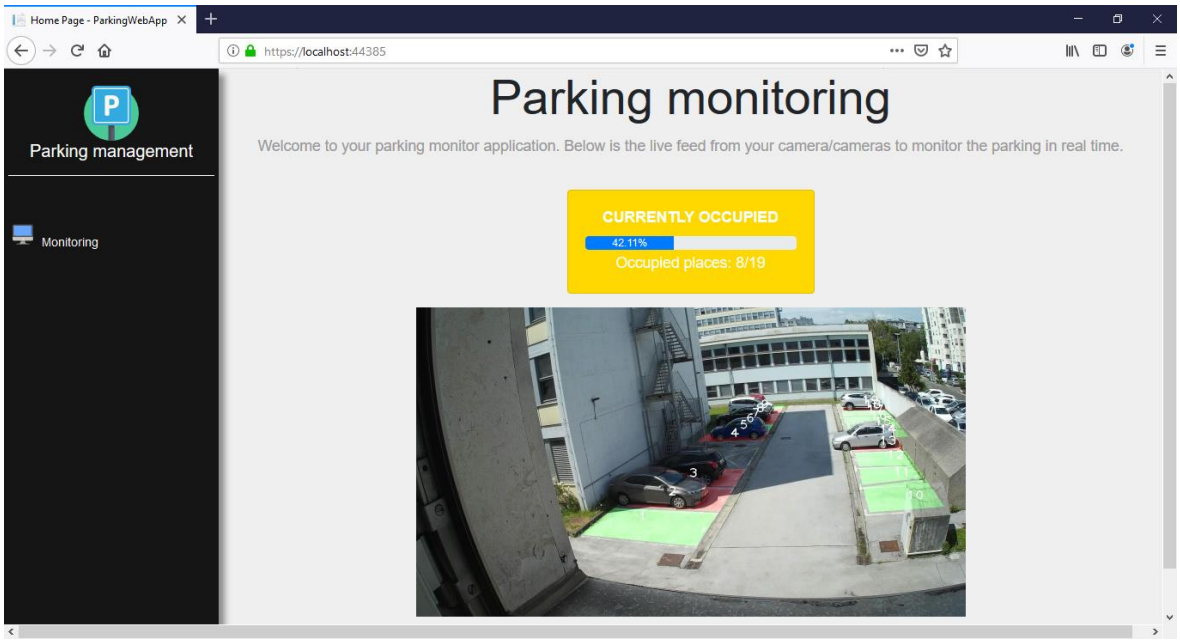


Slika 5.4 Primjer binarnih maski: a) okvira detekcije, b) parkirnog mjesta i c) njihovog presjeka

Slika 5.5 Odgovarajuće područje stvarne slike za prethodno prikazane binarne maske (mjesto 2)

6. Prezentacijski sloj

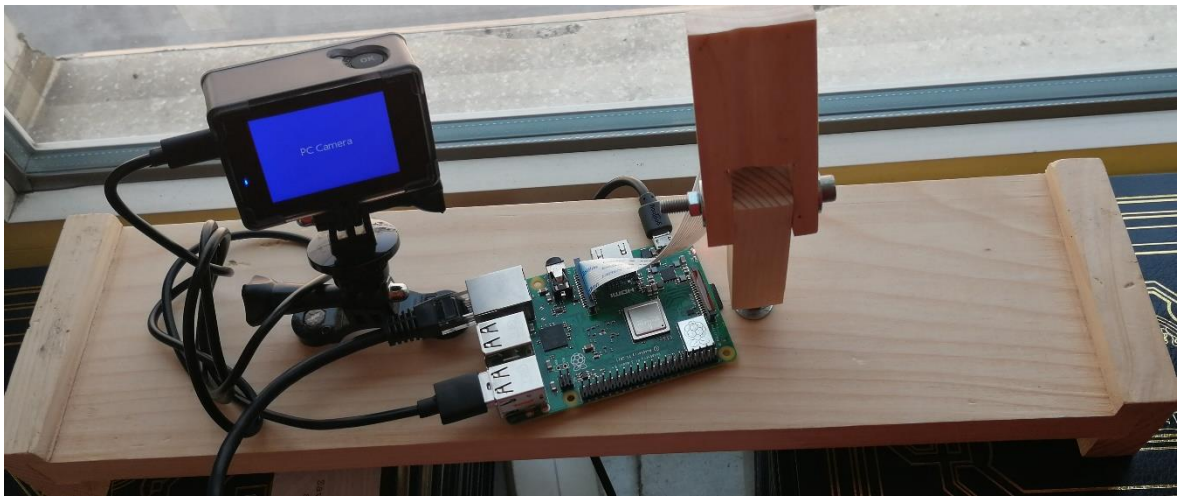
Kako bi rezultati detekcije bili korisni, potrebno je prezentirati podatke korisniku na razumljiv i organiziran način. Prezentacijski sloj sustava izveden je kao Web aplikacija u tehnologiji ASP .Net Core. Aplikacija podatke dobiva iz REST Web servisa napisanog u Pythonu pomoću razvojnog okvira Flask. Aplikacija se sastoji od indikatora zauzeća parkirališta i slike uživo s parkirališta, na koju je dodana maska koja vizualizira zauzeće svakog pojedinog mjesta.



Slika 6.1 Parking monitoring aplikacija

7. Eksperimentalna implementacija i obrada rezultata

U svrhu testiranja implementiranog rješenja i kvalitativne obrade rezultata za procjenu točnosti, sustav je postavljen u prostorijama fakulteta sa pogledom na jedno od fakultetskih parkirališta. Napravljen je stalak za inicijalnu verziju konfiguracije kamera, međutim, kasnije je odlučeno da će se koristiti samo jedna kamera, a ne više njih kako je zamišljeno na početku.

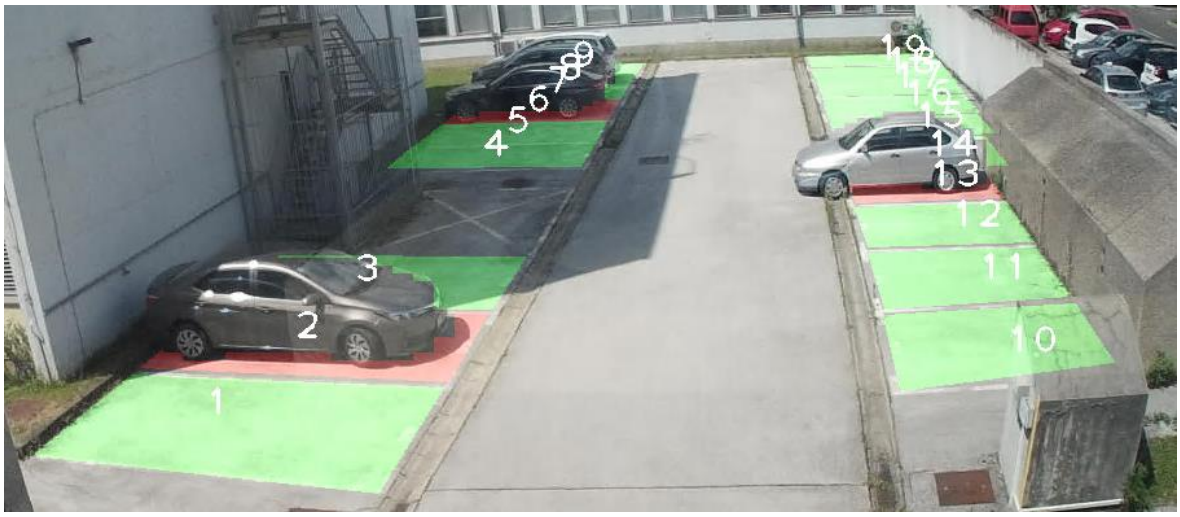


Slika 7.1 Stalak za kamere – zadnja strana

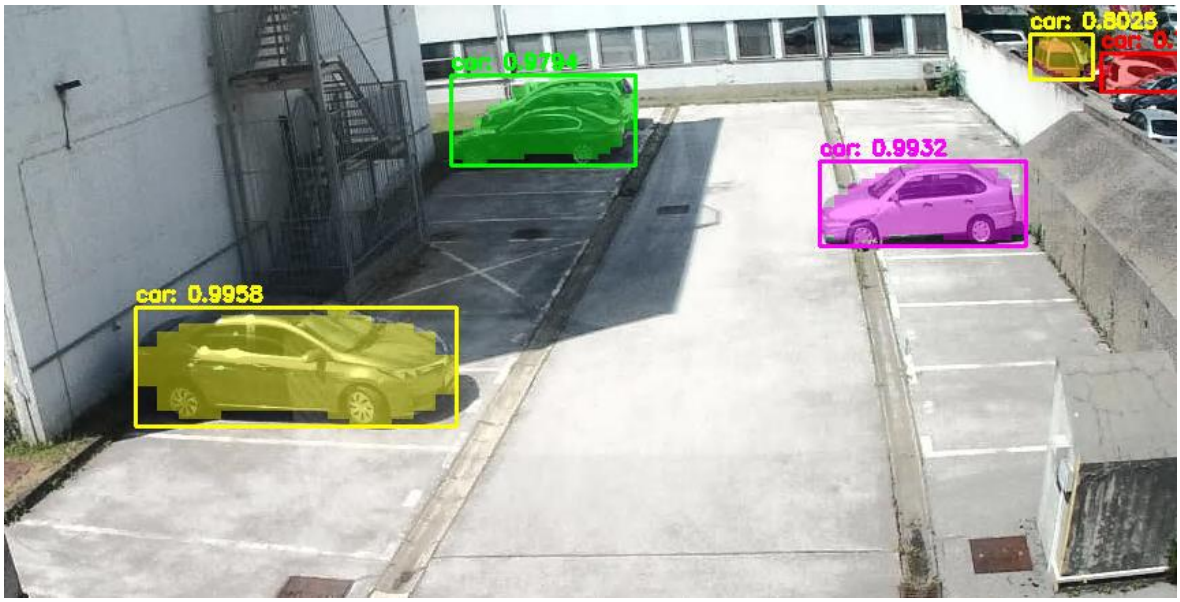
Slika 7.2 Stalak za kamere – prednja strana

Kamere su pozicionirane ispred prozora u prostoru zavodske knjižnice. Jedan od problema koji je prouzročilo ovakvo pozicioniranje je pojava refleksije prozorskog stakla na slici koja je utjecala na rezultate detekcije konvolucijske mreže. Primjer refleksije vidljiv je u području

parkirnih mjesta 2 i 3. (Slika 7.3) Prijedlog za rješenje ovog problema je izrada vodonepropusnog kućišta kako bi se kamera i ugradbeno računalo mogli postaviti vani. Kućište bi ispred leće trebalo imati anti reflektivno staklo kako bi izbjegli problem refleksije. Tako bi postigli veću točnost detekcije i otpornost na različite svjetlosne uvjete scene.



Slika 7.3 Rezultat detekcije sa vidljivom refleksijom i pogreškama u detekciji



Slika 7.4 Uzrok detekcijske pogreške

Drugi problem koji uzrokuje pogreške u radu sustava je djelomično ili potpuno zaklanjanje automobila drugim automobilima ili objektima na sceni. U eksperimentalnoj implementaciji najviše pogrešaka uzrokovanih zaklanjanjem događa se u gornjoj lijevoj regiji parkirališta (parkirna mjesta 7, 8, 9). Na slici gore vidimo da je u toj regiji jedino mjesto 6 označeno kao zauzeto, iako su zauzeta i mjesta 8 i 9. (Slika 7.3) Postoje dva uzroka ovog problema: udaljenost objekata od kamere i kut kamere u odnosu na ravninu parkirališta. Djelomično

zaklanjanje automobila na parkirnim mjestima 8 i 9, uz malu površinu slike koju ti automobili zauzimaju zajedno sa automobilom na mjestu 6 uzrokovali su pogrešku detektora. (Slika 7.4) Detektor ih je grupirao zajedno i dobiven je detekcijski okvir koji omeđuje sva tri automobila, kao da se radi o jednom automobilu. Za ovaj problem postoji više nezavisnih rješenja koja bi mogla dati bolje rezultate detekcije. U nastavku će biti dan prijedlog nekih rješenja, od kojih svako zahtjeva određene promjene u pozicioniranju kamera.

Znamo da detektor dobro radi za mjesta koja su blizu kamere, pa bi mogli postaviti jednu kameru sa suprotne strane parkirališta koja bi bila zadužena za udaljenu regiju za koju ova kamera radi pogreške. Za svaku kameru bi tada definirali regiju koja bi obuhvaćala otprilike pola parkirališta i na kraju bi se rezultati detekcije objedinili. Drugo moguće rješenje bilo bi postaviti kameru na veću visinu u odnosu na parkiralište kako bi dobili veći kut kamere naspram ravnine parkirališta. Na taj način smanjile bi se zaklonjene površine objekata i dobili bi bolje rezultate detekcije. Treća ideja je postaviti kameru bočno u odnosu na parkiralište tako da gleda na automobile sa zadnje ili prednje strane. Tada ne bi uopće trebalo biti zaklanjanja, ali preduvjet za ovakvu konfiguraciju je mogućnost postavljanja kamere na dovoljnu udaljenost da se obuhvati cijela širina parkirališta. Najbolje rješenje, ali i tehnički najzahtjevnije bilo bi postaviti kameru direktno iznad parkirališta, tako da se ravnina slike i ravnina parkirališta poklapaju. Na ovaj način nije moguća pojava zaklanjanja jednog automobila drugim.

U eksperimentalnoj implementaciji ovog rada nisu postojale tehničke mogućnosti za izvedbu niti jednog od prethodno opisanih rješenja, ali su dobiveni rezultati na temelju kojih su izvedene ideje za ta rješenja.

Zaključak

U okviru ovog rada razvijen je autonomni sustav za upravljanje parkiralištem, proširiv na neograničen broj parkirališta uz popratna ulaganja u infrastrukturu. Cilj rada bio je proizvesti što jednostavniji i jeftiniji sustav kao alternativu postojećim sustavima koji uobičajeno uključuju postavljanje senzora na svako pojedinačno parkirno mjesto. Osnovu sustava čine ugradbeno računalo Raspberry Pi Model 3 B+, pripadajuća kamera koja dohvaća slike parkirališta u stvarnom vremenu i poslužitelj koji radi obradu podataka i posluživanje obrađenih podataka prema korisnicima.

Zadovoljeni su uvjeti rada sustava u stvarnom vremenu uz prosječno kašnjenje sustava od otprilike 6 sekundi, što je zadovoljavajuće za navedenu primjenu. Točnost detekcije je odlična za regije parkirališta koje se nalaze bliže kameri, a opada sa udaljenošću od kamere. Prijedlog za rješenje tog problema je postavljanje još kamera u regijama koje nisu dobro pokrivene sa primarnom kamerom.

U radu su iskorištene postojeće tehnike računalnog vida za ostvarenje cjevovoda koji se sastoji od dohvata slike, prijenosa slike na poslužitelj, detekcije automobila, usporedbe lokacije automobila sa definiranim parkirnim mjestima i dobivanja rezultata usporedbe koji predstavljaju izlaz iz sustava.

Prostor za poboljšanje postoji u prezentacijskom dijelu, u vidu implementacije mobilne aplikacije za pregled stanja pojedinog parkirališta i iscrtavanja parkirališta pomoću računalne grafike kako bi dobili pregledniju reprezentaciju parkirališta.

Administracijski dio sustava također bi se mogao poboljšati uvođenjem administratorske Web aplikacije koja bi služila za početnu konfiguraciju sustava.

Literatura

[1] Raspberry Pi službena stranica.
Poveznica: https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/; pristupljeno 1. lipnja 2020.

[2] SJCAM SJ4000 – stranica proizvođača.
Poveznica: https://sjcam.com/product/sj4000/; pristupljeno 3. lipnja 2020.

[3] Szeliski, R., *Computer Vision: Algorithms and Applications*. London: Springer, 2011.

[4] Hartley R., Zisserman A., *Multiple View Geometry in Computer Vision.* 2. izdanje, Cambridge University Press, 2004.

[5] Matlab Computer Vision Toolbox
Poveznica: https://www.mathworks.com/products/computer-vision.html; pristupljeno: 7. lipnja 2020.

[6] OpenCV dokumentacija: Uvod
Poveznica: https://docs.opencv.org/4.3.0/d1/dfb/intro.html; pristupljeno: 7. lipnja 2020.

[7] Introduction to ASP.NET Core
Poveznica: https://docs.microsoft.com/en-us/aspnet/core/introduction-to-aspnet-core?view=aspnetcore-3.1; pristupljeno: 10. lipnja 2020.

[8] Raspberry Pi OS
Poveznica: https://www.raspberrypi.org/documentation/raspbian/; pristupljeno: 10 lipnja 2020.

[9] Krizhevsky A., Sutskever I., Hinton G., E., *ImageNet Classification with Deep Convolutional Neural Networks*

[10] Girshick R., Donahue J., Darell T., Malik J., *Rich feature hierarchies for accurate object detection and semantic segmentation,* UC Berkley, 22. listopad 2014.

[11] Girshick R., *Fast R-CNN,* Microsoft Research, 27. rujna 2015.

[12] Ren S., He K., Girshick R., Sun J., *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,* 6. siječnja 2016.

[13] He K., Gkioxari G., Dollar P., Girshick R., *Mask R-CNN*, 24. siječnja 2018.

Sažetak

U radu je opisana implementacija sustava za autonomno upravljanje parkiralištem temeljeno na računalnom vidu. Sustav se sastoji od ugradbenog računala Raspberry Pi sa pripadajućom kamerom usmjerenom na površinu parkirališta i poslužitelja zaduženog za obradu i prezentaciju rezultata obrade korisniku. Algoritmi računalnog vida implementirani su uz pomoć razvojnog okvira OpenCV. Postupak detekcije sastoji se od dohvaćanja slike sa kamere, prijenosa slike na poslužitelj, detekcije automobila na slici i usporedbe sa definiranim parkirnim mjestima. Detekcija automobila implementirana je pomoću duboke konvolucijske neuronske mreže „Mask R-CNN“. Rezultat detekcije je informacija o popunjenosti svakog parkirnog mjesta i vizualizacija tih informacija na slici dobivenoj sa kamere.

Ključne riječi: parkiralište, parkirna mjesta, OpenCV, Mask R-CNN, računalni vid, Raspberry Pi

Summary

This paper describes the implementation of an autonomous computer vision system for managing parking lots. Systems main components are Raspberry Pi embedded computer, camera overseeing the parking lot connected to the Raspberry Pi and cloud server for processing parking lot images and presenting results to the user. Computer Vision algorithms are implemented using Open CV. Detection pipeline consists of grabbing images from the camera, transferring images to the server, detecting cars on that images and comparing detection results with predefined parking spaces. Car detection and segmentation is implemented using deep convolutional neural network Mask R-CNN. Results of detection include information about availability of each parking space and visualization of that information on images grabbed from the camera.

Keywords: parking, parking space, OpenCV, Mask R-CNN, Computer Vision, Raspberry Pi