Programsko rješenje za pomoć pri parkiranju vozila

Kuprešak, Marko

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

PROGRAMSKO RJEŠENJE ZA POMOĆ PRI PARKIRANJU VOZILA

Diplomski rad

Marko Kuprešak

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za obranu diplomskog rada

Osijek, 10.05.2020.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za obranu diplomskog rada

Ime i prezime studenta:	Marko Kuprešak
Studij, smjer:	Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije
Mat. br. studenta, godina upisa:	D-4ARK, 26.09.2019.
OIB studenta:	26447340052
Mentor:	Izv. prof. dr. sc. Mario Vranješ
Sumentor:	
Sumentor iz tvrtke:	Matija Pul
Predsjednik Povjerenstva:	Doc.dr.sc. Ratko Grbić
Član Povjerenstva:	Izv. prof. dr. sc. Marijan Herceg
Naslov diplomskog rada:	Programsko rješenje za pomoć pri parkiranju vozila
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak diplomskog rada:	Planiranje kretanja autonomnog vozila predstavlja procjenu referentne putanje kojom se vozilo treba kretati kako bi stiglo s trenutne pozicije na željenu. Pri tome su autonomnom vozilu na raspolaganju različiti senzori (LIDAR-a, kamere, GPS i sl.). U okviru diplomskog rada potrebno je osmisliti algoritam koji će na temelju obrade slika dobivenih s kamere na vozilu detektirati parkirna mjesta i utvrditi jesu li ona zauzeta ili slobodna. Nakon detekcije prvog slobodnog parkirnog mjesta, a uz poznate dimenzije samog vozila, potrebno je procijeniti referentnu putanju kojom se vozilo treba kretati kako bi se parkiralo te je potrebno provesti samo autonomno parkiranje vozila. Nakon razvoja algoritma u željenom programskom jeziku, potrebno ga je testirati u prikladnom simulatoru za različite scenarije zauzetosti parkirnih mjesta i različite strane ceste, pri čemu je parkiranje potrebno izvesti vožnjom unaprijed i vožnjom unatrag.. "Tema rezervirana za Marko
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	10.05.2020.

Potpis mentora za predaju konačne verzije rada u Studentsku službu pri završetku studija:	Potpis:
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 14.05.2020.

Ime i prezime studenta:	Marko Kuprešak
Studij:	Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije
Mat. br. studenta, godina upisa:	D-4ARK, 26.09.2019.
Ephorus podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Programsko rješenje za pomoć pri parkiranju vozila**

izrađen pod vodstvom mentora Izv. prof. dr. sc. Mario Vranješ

i sumentora

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.

Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

U posljednje vrijeme dolazi do značajnih promjena u automobilskoj industriji. Sustavi autonomnih funkcija u vozilima ili skraćeno zvani ADAS (engl. *Advanced Driver Assistance Systems*) sve su kompleksniji i sve se više operacija u vožnji prepušta njima, tj. samom vozilu. To će u skorijoj budućnosti u jednom trenutku dovesti do potpune autonomije vozila, gdje se čovjeka (vozača) kao glavnog uzročnika većine prometnih nesreća u potpunosti zamjenjuje naprednim algoritmima koji obavljaju funkcije u vožnji umjesto njega. Kako bi se postigla visoka razina autonomije vozila, potrebni su vrlo moćni računalni sustavi, skup senzora poput kamera, radara, lidara itd., te napredni algoritmi koji su u stanju obrađivati podatke sa senzora u stvarnom vremenu. Jedan od osnovnih koraka u postizanju pune autonomije vozila je autonomno parkiranje vozila, gdje se pri niskim brzinama i u relativno kontroliranim uvjetima mogu koristiti i testirati navedeni sustavi za tu namjenu.

Algoritam za detekciju parkirnog mjesta zasnovan na obradi slike s kamere postavljene na vozilu jedan je od osnovnih sastavnih dijelova sustava za autonomno parkiranje vozila. U posljednje vrijeme sustavi za automatsku detekciju parkirnog mjesta i parkiranje vozila postaju dio osnovne opreme vozila. Cilj ovog algoritma je skratiti vrijeme traženja parkirnog mjesta i pojednostaviti manevar parkiranja. Međutim, iako se u teoriji ovo može činiti kao vrlo jednostavan zadatak, u praski se pokazao znatno težim. Algoritam autonomnog parkiranja može se svesti na tri osnovna problema. Prvi je detekcija slobodnog parkirnog mjesta. Parkirna mjesta nisu svugdje jednakih dimenzija, tipova i oblika. Npr. postoje otvorena i zatvorena parkirna mjesta, kosa i ravna, šira, uža itd. Nadalje, linije koje označavaju granice parkirnog mjesta mogu biti različitih boja poput plavih, žutih i bijelih, a uz to se javlja i problem što su neke od linija zbog utjecaja vremena izblijedjele. Osim toga, s obzirom da se radi o jednostavnim oznakama na kolniku, ne postoje nekakve složene karakteristične značajke po kojima bi ih se moglo otkriti (npr. kao što je to postojanje očiju, nosa i usta za slučaj detekcije ljudskog lica). Drugi problem je klasifikacija parkirnog mjesta. Nakon što se parking detektira, potrebno je utvrditi da li je slobodan ili zauzet i pri tome je moguće koristiti jedan od dva pristupa. U prvom se pristupu koristi fuzija informacija s različitih tipova senzora za klasifikaciju parkirnog mjesta. Takav pristup je znatno složeniji, ali i manje podložan pogreškama. Kod drugog pristupa koristi se samo jedan senzor (npr. radar ili kamera), čime je ovaj pristup jednostavniji, ali posljedično najčešće i podložniji pogreškama. U ovom radu se za klasifikaciju parkirnih mjesta koristi samo jedan senzor (RGB kamera). Naposljetku se javlja problem izvođenja manevra samog parkiranja, gdje se uz praćenje parkirnog mjesta moraju zadavati odgovarajuće naredbe aktuatorima, kako bi se manevar uspješno izvršio.

Iz navedenog se vidi kako je za uspješno autonomno parkiranje vozila potrebno uspješno riješiti niz složenih problema.

Cilj je ovog diplomskog rada proučiti načine detekcije parkirnog mjesta isključivo obradom slike dobivene s kamere postavljene na bočnoj strani vozila, bez korištenja drugih tipova senzora na vozilu. Nakon osmišljavanja i implementacije algoritma za detekciju parkirnog mjesta te određivanja je li ono slobodno ili nije, treba osmisliti algoritam koji će uparkirati vozilo na detektirano slobodno parkirno mjesto vožnjom prema naprijed i vožnjom prema nazad. Algoritam treba biti prilagodljiv uvjetima na okolnim parkirnim mjestima i treba raditi za obje strane vozila (tj. vozilo se treba moći uparkirati na parkinge detektirane i s lijevu i s desne strane vozila). Kako bi se testirali rezultati kompletnog razvijenog rješenja, isti je potrebno testirati u prikladnom simulatoru.

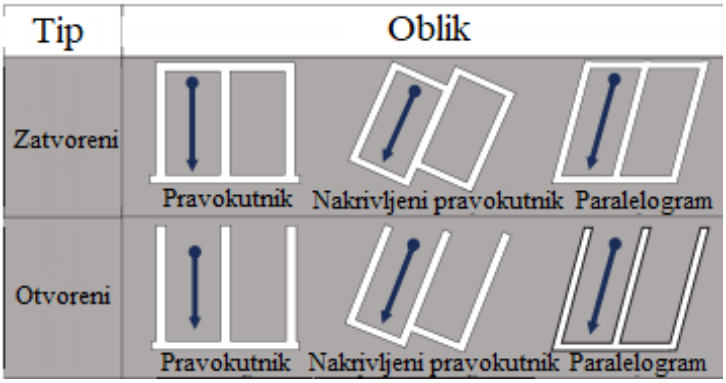
U nastavku rada, u drugom poglavlju opisana su postojeća rješenja za detekciju i klasifikaciju parkirnih mjesta, kao i načini manevriranja vozilima. Analizirani su načini rada pojedinih rješenja i identificirane su prednosti i potencijalni nedostatci postojećih rješenja. U trećem poglavlju opisan je proces izrade vlastitog rješenja za autonomno parkiranje vozila i detaljno je objašnjen način rada algoritma. Nakon toga, u četvrtom poglavlju predstavljeni su rezultati testiranja algoritma autonomnog parkiranja, a u posljednjem poglavlju rada izneseni su zaključci.

2. PROBLEM AUTONOMNOG PARKIRANJA VOZILA

Autonomno parkiranje, iako se na prvu može činiti kao jednostavan problem, zapravo je vrlo kompleksan zadatak kojeg je potrebno podijeliti na više pod-zadataka. Jedan od zadataka je detektirati prostor na slici dobivenoj s kamere koji se može klasificirati kao parkirno mjesto. Drugi zadatak predstavlja donošenje ispravne odluke o zauzeću parkinga, tj. algoritam mora donijeti odluku je li parkirno mjesto slobodno ili zauzeto. Naposljetku je potrebno izvršiti sam manevar parkiranja u detektirano slobodno parkirno mjesto.

Kako je ova tema poprilično složena, ne postoji velik broj javno dostupnih radova koji obrađuju sve tri radnje autonomnog parkiranja, tj. i detekciju, i klasifikaciju, i manevar parkiranja vozila. Stoga su u ovom poglavlju obrađeni radovi koji se bave pojedinom od navedenih radnji autonomnog parkiranja, tako da su obrađeni radovi koji se tiču detekcije parkinga, radovi koji se tiču klasifikacije parkinga i radovi koji obrađuju samo izvođenje manevra parkiranja vozila.

U prvom od radova koji je ovdje analiziran [1] i koji se bavi detekcijom parkirnog mjesta, predlaže se da se parkirna mjesta grupiraju prema tipu i obliku. Tako znanstvenici u radu predlažu dva tipa parkirnih mjesta (otvorena i zatvorena) te tri osnovna oblika parkirnih mjesta (pravokutnik, paralelogram i nakrivljeni pravokutnik) (Slika 2.1.).



Sl. 2.1. *Tipovi i oblici parkirnih mjesta prema [1]*

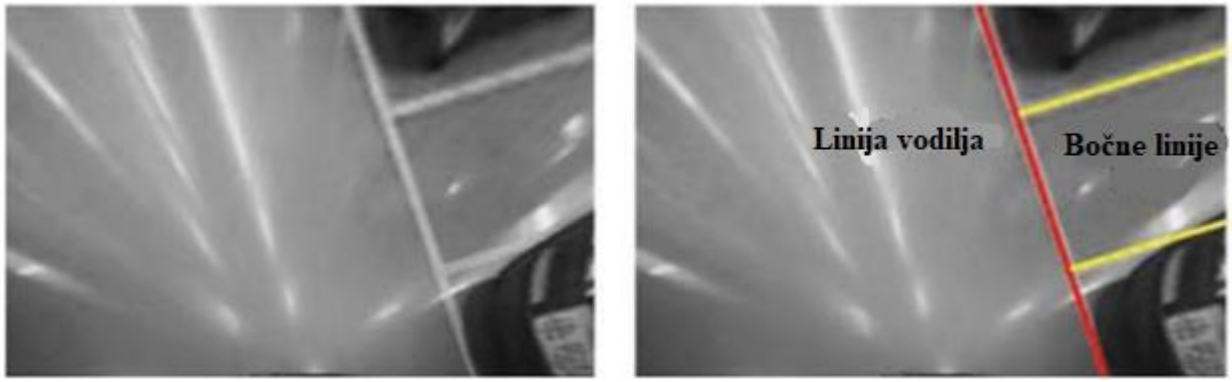
Poznavanjem tipa i oblika parkirnog mjesta određuje se točka ulaza u parkirno mjesto te krajnja točka tog istog parkirnog mjesta. Koristi ovakvog pristupa su mogućnost detekcije raznih oblika parkinga. Nakon što se detektiraju kandidati za linije parkinga, traže se točke presjeka linija (ili krajnje točke linija ako je parkirno mjesto otvorenog tipa). Kada se utvrde njihove lokacije određuje se njihov smjer. Iz tih značajki algoritam zaključuje o kojem tipu i obliku parkinga se radi. Algoritam je testiran na 4109 slika (rezolucije 330x110 elemenata slike) dobivenih snimkom

iz stvarnog vozila, na način da se vozilo kretalo uz parkirna mjesta, a kamera je bila pozicionirana sa strane vozila i snimala. Prilikom testiranja utvrđeno je da ova metoda radi ispravno u 96% slučajeva. Kako bi se algoritam dalje unaprijedio potrebno je dodati detekciju zauzetosti parkinga i algoritam praćenja parkinga.

U radu [2] predlaže se detekcija parkirnih mjesta pomoću fuzije dvaju pristupa detekcije parkirnih mjesta. Prvi je detekcija pomoću ultrazvučnih senzora koji pomoću dostupnih informacija o brzini vozila i podataka sa senzora definiraju da li je parkirno mjesto zauzeto ili nije. Mana ovog pristupa je ta što se ne detektiraju linije parkirnog mjesta, nego se oslanja isključivo na ultrazvučni senzor. Drugi pristup je detekcija linija u slikama dobivenim s kamere pomoću RANSAC (engl. *RANdom SAmple Consensus)* algoritma. RANSAC je iterativna metoda procjene parametara unaprijed definiranog matematičkog modela na temelju podataka u kojima su prisutne stršeće vrijednosti. Kombinacijom ovih dviju metoda dobiva se robusnija detekcija koja ne ovisi o razini osvjetljenja scene. Algoritam je testiran na 265 slika rezolucije 360x480 elemenata slike snimljenih u podzemnim garažama. Dobiveni rezultati su pokazali kako detekcija parkirnih mjesta radi znatno bolje fuzijom dvaju pristupa, uz točnost detekcije 97.4%. Mana ovog algoritma je ta što je osjetljiv na refleksije svjetlosti od podloge te u pojedinim situacijama nije bio u mogućnosti ispravno detektirati oznake parkirnog mjesta.

Autori u radu [3] predlažu korištenje algoritma koji procjenjuje položaj parkinga u odnosu na položaj vozila. Za to se koriste kamerom koja je postavljena na stražnjoj strani vozila. Kamera snima parkirna mjesta, a korisnik nakon zaustavljanja odabere na koje od njih se želi parkirati. Nakon odabira parkirnog mjesta na koje se želi parkirati, postavlja se ROI (engl. *Region Of Interest*) nad tim područjem i vrši se transformacija perspektive u pogled odozgor. Nakon toga se detektiraju linije pomoću Houghove transformacije. Linija s najviše točaka odziva u Houghovom prostoru uzima se kao dominantna linija i ona predstavlja liniju „vodilju“ (Slika 2.2.). Linije okomite na nju su bočne linije parkinga.

Kako bi se detektirao položaj parkirnog mjesta u odnosu na vozilo, koristi se NCC (engl. *Normalized Cross Correlation*) metoda, koja, poznajući dimenzije i oblik parkinga, traži kut s maksimalnim brojem točaka preklapanja s predloškom (Slika 2.3.). Algoritam je testiran za slučajeve kada su tri parkirna mjesta slobodna i kada su dva od tri parkirna mjesta zauzeta te se došlo do zaključka da algoritam radi stabilno u oba slučaja. Mana ovog pristupa je ta što algoritam radi samo za pravokutne zatvorene parkinge i za parkiranje unatrag.



a) b)

Sl. 2.2. *Detekcija linije vodilje (a) Prikaz odabranog parkinga iz ptičje perspektive (b) Prikaz linije vodilje (crvena) i bočnih linija parkinga (žute) [3]*

Sl. 2.3. *Procjena položaja parkin ga pomoću rotiranja predloška [3]*

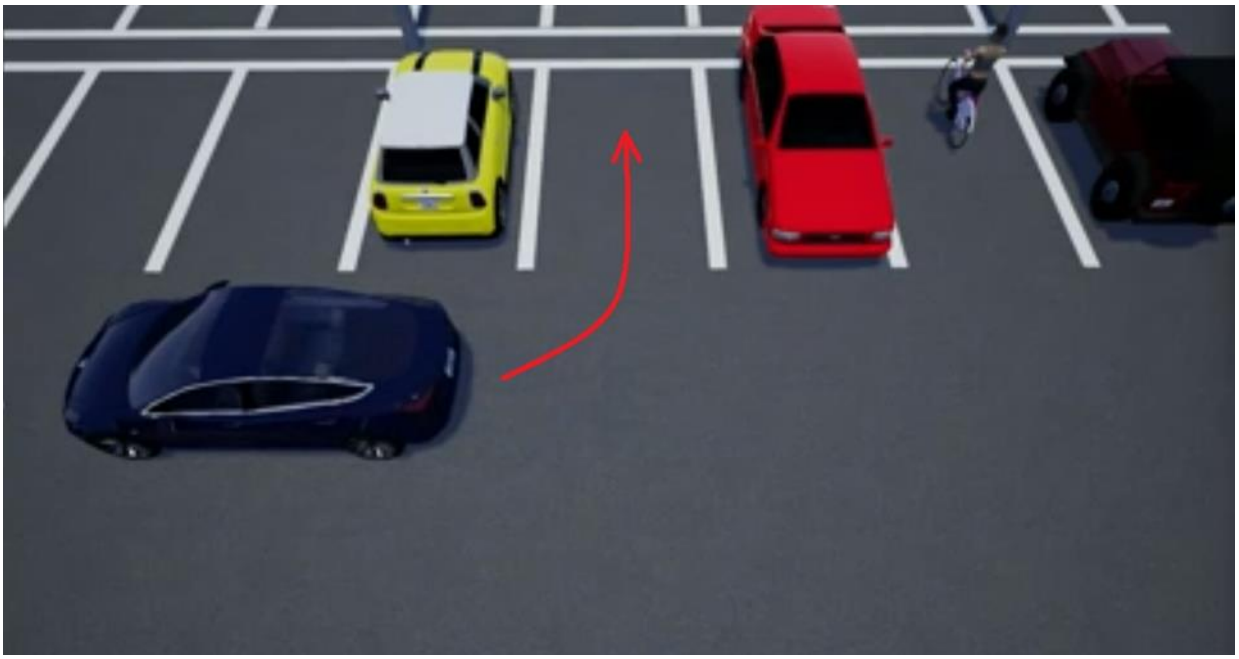
Sve do sada spomenute metode detekcije parkirnog mjesta zasnivaju se na modelima detekcije rubova ili modelu detekcije značajki. Znanstvenici u radu [4] opisuju model detekcije zauzetosti parkinga korištenjem dubokog učenja. Oni u svom radu predlažu korištenje duboke konvolucijske neuronske mreže (engl. *Counvolution Neural Network*) i binarnog SVM (engl. *Support Vector Machine*) klasifikatora. Klasifikator je treniran i testiran na značajkama koje je duboka konvolucijska mreža naučila iz javno dostupnog skupa podataka (PKlot). Konvolucijska mreža se sastoji od pet konvolucijskih slojeva i tri potpuno povezana sloja s po 4096, 4096 i 1000 neurona, a na kraju mreže se nalazi *softmax* klasifikator. Mreža je trenirana za različite razine osvjetljenja i za različite vremenske uvjete, te prilikom testiranja pokazuje uspješnost klasifikacije parkirnih mjesta od 99.7% za trening set i 96.7% za novi set ulaznih (testnih) slika. Ograničenje ovog

pristupa je to što je potrebno unaprijediti algoritam za rad po noći. Također, sjene ili razne solarne refleksije predstavljaju problem u klasifikaciji parkinga.

U sljedećim radovima koji će ovdje biti analizirani obrađen je problem klasifikacije parkirnog mjesta te u njima sama detekcija istog nije predmet istraživanja. Za razliku od prethodnih radova, u radovima [5] i [6] razmatra se samo zauzetost parkinga, a ne i detekcija parkirnih mjesta. Iako se u navedenim radovima kamera pomoću koje se vrši klasifikacija parkinga ne nalazi na vozilu, oni su obrađeni iz razloga što se metode klasifikacije parkinga potencijalno mogu primijeniti u vlastitom algoritmu pomoći pri parkiranju vozila. Naime, u radu [5] predlaže se da se u garažama umjesto senzora na svakom od parkirnih mjesta koriste fiksne kamere postavljene na zidu garaže koje bi detektirale da li su parkirna mjesta slobodna ili ne. Za početak rada algoritma potrebno je ručno označiti parkirna mjesta. Korisnik označava parkirna mjesta na način da mišem označi prostor na slici dobivenoj s kamere između linija svih vidljivih parkirnih mjesta, nakon čega se vrši detekcija rubova, kontura i detekcija prednjeg plana i pozadine unutar označenih područja. Kombinacijom triju navedenih metoda odlučuje se o zauzetosti svakog označenog parkirnog mjesta. Metoda je testirana za dnevne uvjete na uličnim parkiralištima i za uvjete u garažama te se pokazala iznimno pouzdanom za dane uvjete. U radu [6] predlaže se da se umjesto detektiranja linija i kontura koriste segmentacija slike i lokalni binarni uzorci. Polazi se od ideje da prazno parkirno mjesto nema tako raznoliku kompoziciju boja i tekstura kao mjesto na kojem se nalazi vozilo. Algoritam korišten za segmentaciju slike je grupiranje pomoću Mean shift algoritma. Zbog redundancije je dodan i LBP (engl. *Local binary pattern*) algoritam, koji karakterizira lokalnu teksturu prostorne strukture slike. Kako bi se istrenirao LPB klasifikator korišteno je 1439 slika praznog parkinga. Prilikom testiranja na parkirnim mjestima u garažama i na otvorenom prostoru, pokazalo se da algoritam radi s točnošću od 97%. Mana ovog algoritma je ta što ne može s velikom sigurnošću utvrditi zauzetost parkinga na parkirnim mjestima koja se ne vide u potpunosti zbog okluzije. Na primjer, kada vozilo parkirano na nekom parkingu prekriva dio susjednog slobodnog parkinga, algoritam ga označava kao zauzetog iako je slobodan.

U do sada spomenutim radovima fokus je bio na detekciji i klasifikaciji parkirnih mjesta, ali se pažnja nije posvećivala samom manevru parkiranja. U radu [7] autori predlažu parkiranje unatrag okomito u pravokutno parkirno mjesto (Slika 2.4.), tako da se upravljanje vozilom izvede iz jednog manevra s kutom zakretanja prednjih kotača manjim od maksimalnog kuta kotača. Koristi ovakvog pristupa su pojednostavljenje parkiranja unatrag i geometrijsko planiranje putanje bez kolizije, tj. prilikom parkiranja se u odnos uzimaju veličina parkirnog mjesta i veličina vozila.

Ograničenja ovakvog pristupa su ta da se vozilo može parkirati samo unatrag okomito te susjedna vozila (ukoliko postoje) moraju biti simetrično parkirana s obje strane parkirnog mjesta (Slika 2.4.), tj. ne smiju ulaziti unutar okvira parkirnog mjesta gdje se vozilo mora parkirati jer nema ugrađen sustav protiv kolizije. Prilikom testiranja u simulacijama i na stvarnom vozilu ustanovljeno je da vozilo uspješno savladava zadatak parkiranja.



Sl. 2.4. *Primjer okomitog pravokutnog parkirnog mjesta i simetrično parkiranih vozila*

U radu [8] predloženo je autonomno parkiranje vozila unatrag okomito, koje se izvodi iz maksimalno tri manevra. Za razliku od trenutno postojećih rješenja autonomnog parkiranja koja koriste metodu planiranja putanje pomoću lokalnih *klotoida* (engl. *Local Clothoid Planner*), a koja omogućava proračun samo jedne putanje za koju je potrebno više manevara, u ovom radu autori predlažu korištenje metode koja pomoću Reeds-Shepp metode računa najkraći mogući put vozila do parkirnog mjesta iz maksimalno 3 manevra. Predloženo je korištenje fuzije sonarnih senzora na vozilu te RGB kamera koje se nalaze na svakoj strani vozila, kako bi se detektirali slobodni parkinzi i potencijalne prepreke. Nakon što se detektiraju prepreke, računa se niz potencijalnih putanja za koje je potrebno 3 ili manje manevara kako bi se izvele. Broj potrebnih manevara direktno ovisi o širini ceste (R_w), širini i duljini vozila koje se želi parkirati (W_v, L_{vt}), udaljenosti bližeg boka vozila do parkirnog mjesta (D_s) i širini samog parkirnog mjesta (P_w). U Tablici 2.1. dana je ovisnost broja potrebnih manevara i navedenih parametara kako bi se izvelo parkiranje vozila.

Tablica 2.1. *Ovisnost broja manevara o kinematičkim ograničenjima [8]*

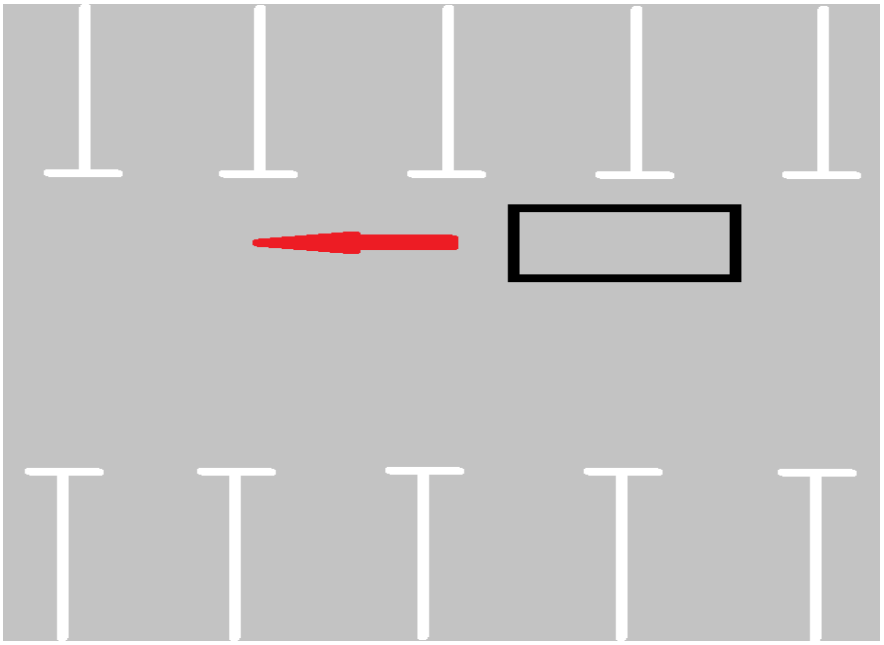
Ime	Kinematička ograničenja	Broj manevara
Široko	$R_w > 3\ W_v$ && $D_s > 1.5\ P_w$ && $P_w > (Wv + 1.2(m))$	1
Normalno	$R_w > 3\ W_v$ && $D_s > P_w$ && $(W_v + 1.2(m)) > P_w > (W_v + 0.4(m))$	2
Usko	$R_w > W_v$ && $(W_v + 0.4(m)) > D_s > 1.5\ P_w$ && $P_w > (W_v + 1.2$ $(m))$	3

Rješenje je testirano i evaluirano od strane Panasonic Automotiv odjela na više od 1000 različitih scenarija parkiranja pomoću MATLAB simulacija, uz uspješnost od 95%. Prednosti ovakvog pristupa su te što se planiranje putanje kretanja vozila može početi računati već kada se kamera postavljena na vozilu nalazi na polovini parkirnog mjesta na koje će se parkirati. Time se dobiva dodatno vrijeme za proračune putanja. Također ovaj algoritam proračunava više mogućih putanja od kojih se odabire jedna s najmanjim brojem manevara na osnovu parametara okoline (tablica 2.1). Nedostaci ovog rješenja su ti što trenutno ne postoji mogućnost parkiranja ukoso, dok je parkiranje bočno moguće, ali još nije razvijeno u sklopu predloženog rješenja.

3. ALGORITAM ZA DETEKCIJU SLOBODNOG PARKIRNOG MJESTA I AUTONOMNO PARKIRANJE VOZILA

Zadatak ovog rada bio je osmisliti i implementirati računalni algoritam za detekciju parkirnog mjesta i klasifikaciju istoga (je li slobodno ili nije). Uz to je nužno osmisliti algoritam pomoću kojega se izvodi manevar parkiranja vozila na prethodno detektirano slobodno parkirno mjesto. Potrebno je pomoću obrade okvira iz videa dobivenog s kamere postavljene na bočnoj strani vozila detektirati parkirno mjesto i provjeriti je li slobodno za parkiranje. Potom, ako je slobodno, treba uparkirati vozilo na slobodno mjesto. Algoritam mora detektirati parkinge koji se nalaze bočno od vozila, na lijevoj i na desnoj strani. Nadalje, algoritam treba raditi za parkirna mjesta koja se nalaze okomito u odnosu na smjer kretanja vozila, a njihov oblik je pravokutan. Skica slučaja za koji algoritam treba raditi dan je na slici 3.1, a vrijedi za parkinge s obiju strana ceste. Na slici se vidi smjer kretanja (crvena strelica) vozila (crni pravokutnik) u odnosu na parkirna mjesta (označena bijelom bojom). Također, na slici se vidi kako se vozilo kreće desnom stranom kolnika. Ujedno, potrebno je osmisliti više manevara parkiranja kako bi se pokrile sve specifične situacije. Za parkiranje vožnjom unatrag bit će napisan po jedan manevar za svaku stranu jer se vozilo parkira na isti način neovisno o zauzeću susjednih parkirnih mjesta. Za parkiranje vožnjom unaprijed biti će osmišljena dva tipa manevra za svaku stranu, jedan za slučaj kada se vozilo mora parkirati između dva postojeća vozila na parkingu i jedan za slučaj kada se vozilo mora parkirati na parkirno mjesto čija su susjedna parkirna mjesta slobodna. Dakle, ukupno je osmišljeno 6 manevara čime se osigurava uspješno parkiranje i za lijevu i za desnu stranu. Skica koja prikazuje gdje se i kako vozilo mora parkirati nalazi se u dijelu 3.2.3. Osmišljeni manevri moraju ispravno uparkirati vozilo za parkinge detektirane s lijeve i desne strane vozila.

Iako vrlo kompleksan, kompletan algoritam autonomnog parkiranja vozila može se razložiti u niz jednostavnih operacija obrade slike i matematičkih proračuna koji između ostalog na izlazu daju informaciju o detekciji parkinga i njegovoj klasifikaciji. Nadalje, ukoliko se zahtijeva parkiranje unaprijed, algoritam u ovisnosti o zauzetosti susjednih parkirnih mjesta mora odlučiti koji manevar od dva dostupna će koristiti, ovisno o tome da li se vozilo parkira na parkirno mjesto koje se nalazi između dva druga vozila ili se parkira na parkirno mjesto čije je susjedno parkirno mjesto slobodno. U nastavku ovog poglavlja prvo su opisani razvojno okruženje, način podešavanja potrebnih biblioteka, a potom je detaljno raspisan rad algoritma za autonomno parkiranje vozila s dodanim popratnim sadržajem za lakše razumijevanje.



Sl. 3.1. *Skica položaja vozila u odnosu na parkirna mjesta prilikom detekcije i klasifikacije istih s obje strane ceste*

3.1. Podešavanje radnog okruženja

Algoritam za detekciju parkinga je napisan u Python programskom jeziku, a pri tome su korištene razne dostupne biblioteke, poput OpenCV i NumPy. Algoritam je razvijan na Ubuntu 16.04 LTS operacijskom sustavu. Prilikom razvoja i testiranja korišten je CARLA Simulator verzije 0.9.6.

Python je programski jezik visoke razine, omogućuje vrlo brz i efikasan razvoj aplikacija, no nedostatak mu je to da se programi pisani u Python jeziku izvršavaju nešto sporije od nižih prevoditeljskih jezika poput C/C++. Za razliku od C/C++ programskih jezika, Python je interpreterski jezik. U Python programskom jeziku koriste se uvlačenja kako bi se razlikovali programski blokovi [9].

OpenCV (engl. *Open Source Computer Vision Library*) je biblioteka otvorenog koda (engl. *open source*) koja se koristi za računalni vid i strojno učenje. Izrađena je kako bi se stvorila standardna infrastruktura za primjenu u algoritmima strojnog učenja i računalnog vida, a sastoji se od preko 2500 optimiziranih algoritama. Neki od njih su algoritmi za prepoznavanje lica, identifikaciju objekata, praćenje objekata, detekciju linija, detekciju rubova itd. Koristi se u mnogim kompleksnim zadacima poput spajanja slika u *Google streetview* aplikaciji, u detekciji utapanja u bazenima, u navigaciji robota, a odgovorna je i za napredak na području autonomne

vožnje u autoindustriji. Postoje implementacije za C++, Python, Java i MATLAB programske jezike [10]. Upute za instalaciju OpenCV biblioteke na Linux OS-u nalaze se na [11].

NumPy je biblioteka otvorenog koda koja se koristi u Python programskom jeziku, a omogućuje korištenje raznih matematičkih funkcija visoke razine kao i podršku za rad s višedimenzionalnim nizovima i matricama. NumPy biblioteka nudi alate za rad s matricama i pri tome osigurava visoke performanse [12]. Upute za instalaciju NumPy biblioteke nalaze se na [13].

CARLA (engl. *CAR Learning to Act*) je simulator otvorenog koda koji je izrađen kako bi se koristio kao modularan i fleksibilan API *(*engl. *Application Programming Interface*) za rješavanje niza problema autonomne vožnje. Samim time što je otvorenog koda, CARLA omogućuje vrlo lako pristupanje i prilagodbu koda za specifične scenarije razvoja autonomne vožnje. CARLA je zasnovan na Unreal Engine-u, te koristi OpenDrive 1.4 standard za definiranje cestovnih i gradskih postavki/logike. Sastoji se od skalabilne klijent-server arhitekture. Server je odgovoran za sve zadatke koji su povezani sa simulacijom, poput prikaza senzora, proračuna i simulacije zakona fizike, ažuriranja stanja, pomaka sudionika u prometu itd. Klijenti se sastoje od modula koji kontroliraju logiku sudionika koji se nalaze u „sceni“. Također, klijent se koristi za postavljanje uvjeta scene poput biranja vremena, mijenjanja mape grada itd. Kako bi se omogućila interakcija klijenta i servera, potrebno je koristiti CARLA API sloj koji se nalazi između sva navedena sloja [14]. Uputstva za instalaciju CARLA simulatora nalaze se na [14].

3.2. Razvoj programskog rješenja za autonomno parkiranje

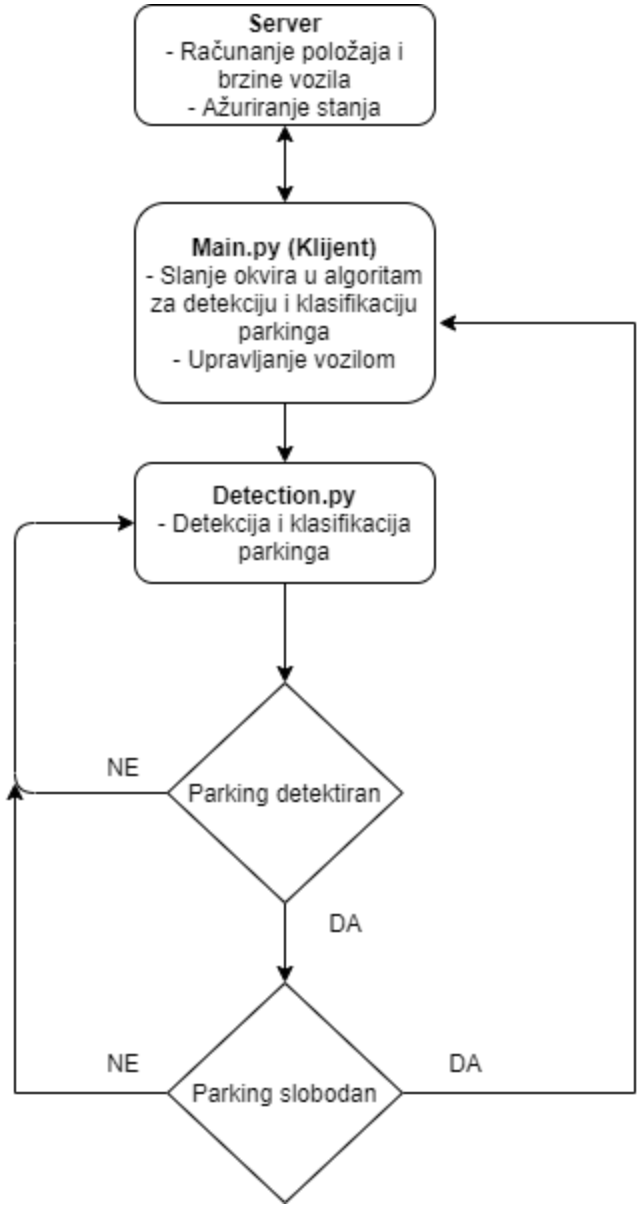
Programsko rješenje za autonomno parkiranje sastoji se od niza algoritama koji se slijedno izvršavaju. Ti algoritmi su redom: algoritam za detekciju parkirnog mjesta, algoritam za klasifikaciju parkirnog mjesta i algoritam za manevriranje vozilom. Programsko rješenje funkcionira na način da se za niz ulaznih video okvira s kamere postavljene na bočnoj strani vozila na izlazu dobiva informacija o detektiranom parkingu i njegovoj zauzetosti. Ulaz u algoritam detekcije parkirnog mjesta je video okvir (ili niz okvira), na kojemu se nakon obrade i klasifikacije iscrtavaju parkirna mjesta, i to crvenom bojom se označavaju zauzeta parkirna mjesta, a zelenom slobodna. Nakon detekcije i klasifikacije pokreće se algoritam manevriranja vozilom kako bi se ono parkiralo u prethodno detektirano slobodno parkirno mjesto.

Skripta za obradu pristiglih okvira zove se *Detection.py*. Ona sadrži funkcije koje se koriste za predprocesiranje okvira, detekciju rubova, detekciju linija, filtriranje, itd., a sve u svrhu

detekcije i klasifikacije parkirnih mjesta. Nadalje, za potrebe izvođenja radnje vezane za algoritam samog autonomnog parkiranja kreirana je još jedna skripta s nazivom Main.py. Main.py prima video okvire sa servera i prosljeđuje ih u Detection.py, ali i vrši upravljanje vozilom nakon što se pronađe slobodan parking. Ta skripta služi kao veza između CARLA servera i algoritma za detekciju i klasifikaciju parkirnog mjesta (*Detection.py*). *Main.py* skripta se ponaša se kao klijent u radu CARLA simulatora. Pomoću te skripte kreira se vozilo, kreira se senzor (RGB kamera postavljena na bočnoj strani vozila), postavljaju se parametri senzora i upravlja se tim istim vozilom. CARLA simulator je dizajniran kao klijent–server sustav. Prilikom zadavanja naredbe za pomak vozila u *Main.py* skripti (klijent), informacije se prosljeđuju CARLA serveru gdje se računa brzina vozila, pomak itd. i ažurira se nova lokacija vozila. Dakle, CARLA server cijelo vrijeme ažurira scenu na kojoj se nalazi vozilo, a klijent šalje naredbe. Pri tome senzor prima podatke od servera. Slike dobivene s RGB kamere prosljeđuju se u *Detection.py* skriptu (ovdje se *Deteciton.py* može zamisliti kao funkcija koja se poziva iz *Main.py* skripte), gdje se dobiveni okviri obrađuju. Kao povratna vrijednost iz skripte *Detection.py* u skriptu *Main.py* dobiva se informacija o zauzetosti parkinga na temelju koje se vrši manevriranje vozila u simulatoru pomoću *Main.py* skripte. Na slici 3.2. može se vidjeti dijagram toka podataka između CARLA servera, CARLA klijenta (*Main.py*) i algoritma detekcije i klasifikacije parkirnog mjesta.

3.2.1. Carla klijent algoritam – dohvaćanje videa s kamere na vozilu iz simlatora

U ovom dijelu detaljno je opisana *Main.py* skripta koja se ponaša kao CARLA klijent algoritam. Kako se CARLA simulator koristio prilikom testiranja uspješnosti parkiranja, ali i prilikom razvoja jednog dijela algoritma detekcije parkinga, bilo je potrebno napisati CARLA klijent algoritam (*Main.py*) koji se ponašao kao spona između CARLA servera i algoritma detekcije i klasifikacije parkinga. Sve funkcije za pristup serveru i manipulaciju sadržaja nalaze se u *carla* biblioteci. Prilikom pokretanja *Main.py* skripte potrebno se spojiti na stranu servera. Bitno je napomenuti da se prvo mora pokrenuti server, a tek onda klijent. Nakon uspješnog spajanja, klijent dohvaća mapu/svijet, a potom se kreiraju svi glumci (engl. *actor*) iz baze dostupnih nacrta (engl. *blueprint*). Svi senzori, vozila, znakovi, pješaci i ostali sudionici prometa mogu se naći u dostupnoj CARLA bazi. Vozilo korišteno za razvoj algoritma je automobil Tesla model 3, a može ga se naći u bazi pod nazivom *„model3“*. CARLA nudi nekoliko tipova percepcije okoline pomoću kamere. Neki od njih su prikazani na slici 3.3 [15]. U razvoju algoritma korištena je RGB kamera (ulazna slika u algoritam bila je slika kao što je prikazana na slici 3.3.(a)). Taj senzor se nalazi u bazi pod nazivom *„sensor.camera.rgb“*, a postavljen je na obje strane vozila u području bočnih retrovizora.

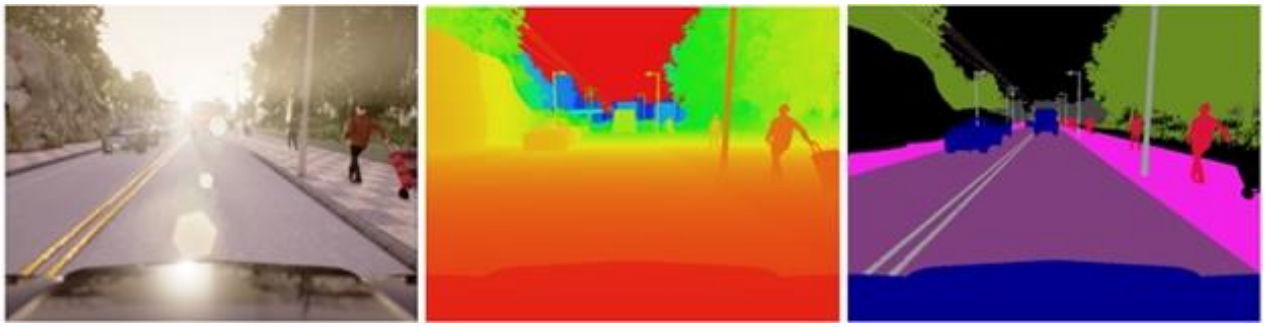


Sl. 3.2. *Dijagram toka podataka između CARLA servera, CARLA klijenta (Main.py) i Detection.py skripte za detekciju i klasifikaciju parkirnog mjesta*

Rezolucija korištenih kamera je 1280x720 elemenata slike, dok je horizontalno vidno polje (engl. *Field of View*) kamere 95° za lijevu stranu vozila i 110° za desnu stranu vozila. Koriste se različita vidna polja jer se vozilo kreće desnom stranom ceste, kao što je prikazano na slici 3.1. Samim time nalazi se bliže desnom parkingu nego lijevom. Kako bi se dobile približno jednake slike s obiju kamera, bilo je potrebno koristiti različita horizontalna vidna polja. Kamere se nalaze pod kutom od 20^{o} prema horizontalnoj osi. Skica položaja kamere može se vidjeti na slici 3.4.

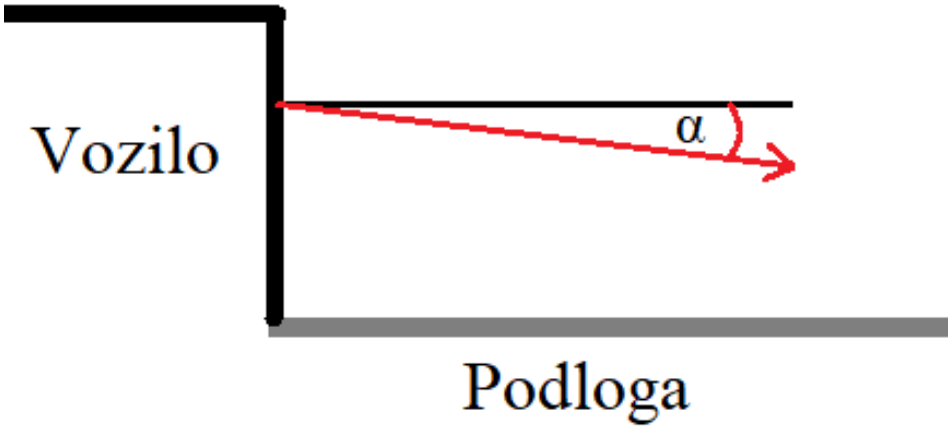
Kamera je prikazana crvenom strelicom koja se nalazi pod kutom α (20^{o}) u odnosu na horizontalnu os.

Ostali parametri kamere su predefinirani od strane simulatora i može ih se vidjeti tablici 3.1. Programski kod napisan u Python programskom jeziku koji se koristio kao CALRA klijent može se naći u prilogu P.3.1. pod nazivom *Main.py*.



a) b) c)

Sl. 3.3. *Prikaz dostupnih percepcija okoline pomoću kamere: (a) RGB kamera, (b)procjena dubine slike pomoću kamere, (c) semantička segmentacija slike pomoću kamere [15]*



Sl. 3.4. *Skica položaja kamere na vozilu u odnosu na horizontalnu os (strelica ukazuje gdje je kamera usmjerena)*

Tablica 3.1. *Parametri kamere korišteni u CARLA simulatoru*

Naziv Parametra	Tip podatka	Vrijednost	Opis
`sensor_tick`	float	0.0	Broj sekundi u simulatoru između dviju snimki senzora (za vrijednost 0.0 broj snimki senzora jednak je maksimalnom broju okvira koje server može poslati)
`image_size_x`	int	1280	Širina slike iskazana u broju elemenata slike
`image_size_y`	int	720	Visina slike iskazana u broju elemenata slike
`gamma`	float	2.2	Ciljana *gamma* vrijednost kamere
`fov`	float	95.0/110.0	Horizontalno vidno polje iskazano u stupnjevima
`shutter_speed`	float	60.0	Vrijeme koje je senzor kamere otvoren iskazano u sekundama, a računa se kao (1.0 / *Shutter speed*) s
`iso`	float	1200.0	Osjetljivost senzora kamere na svjetlost (Najčešći raspon 100 – 6400). Veći broj znači da je senzor osjetljiviji na svjetlost.
`fstop`	float	1.4	Otvor objektiva iskazan kao omjer žarišne duljine objektiva i promjera otvora objektiva

Kako bi se dobile slike sa kamere (okviri), potrebno je pozvati funkciju [14]:

sensor.listen(lambda data: do_something(data)

Na taj način se pretplaćuje na podatke koji se generiraju u senzoru, a svaki puta kada je generiran novi podatak poziva se funkcija *„do_something()“*, gdje se taj podatak obrađuje. Primjer podataka dobivenih sa senzora može se vidjeti na Slikama 3.5. za lijevu stranu vozila i 3.6. za desnu stranu. Na tim slikama se također vidi kako vidno polje lijeve kamere obuhvaća nešto više parkirnih mjesta u kadru za razliku od onoga desne kamere.

Sl. 3.5. *Prikaz okvira dobivenog s kamere postavljene na lijevoj strani vozila*

Sl. 3.6. *Prikaz okvira dobivenog s kamere postavljene na desnoj strani vozila*

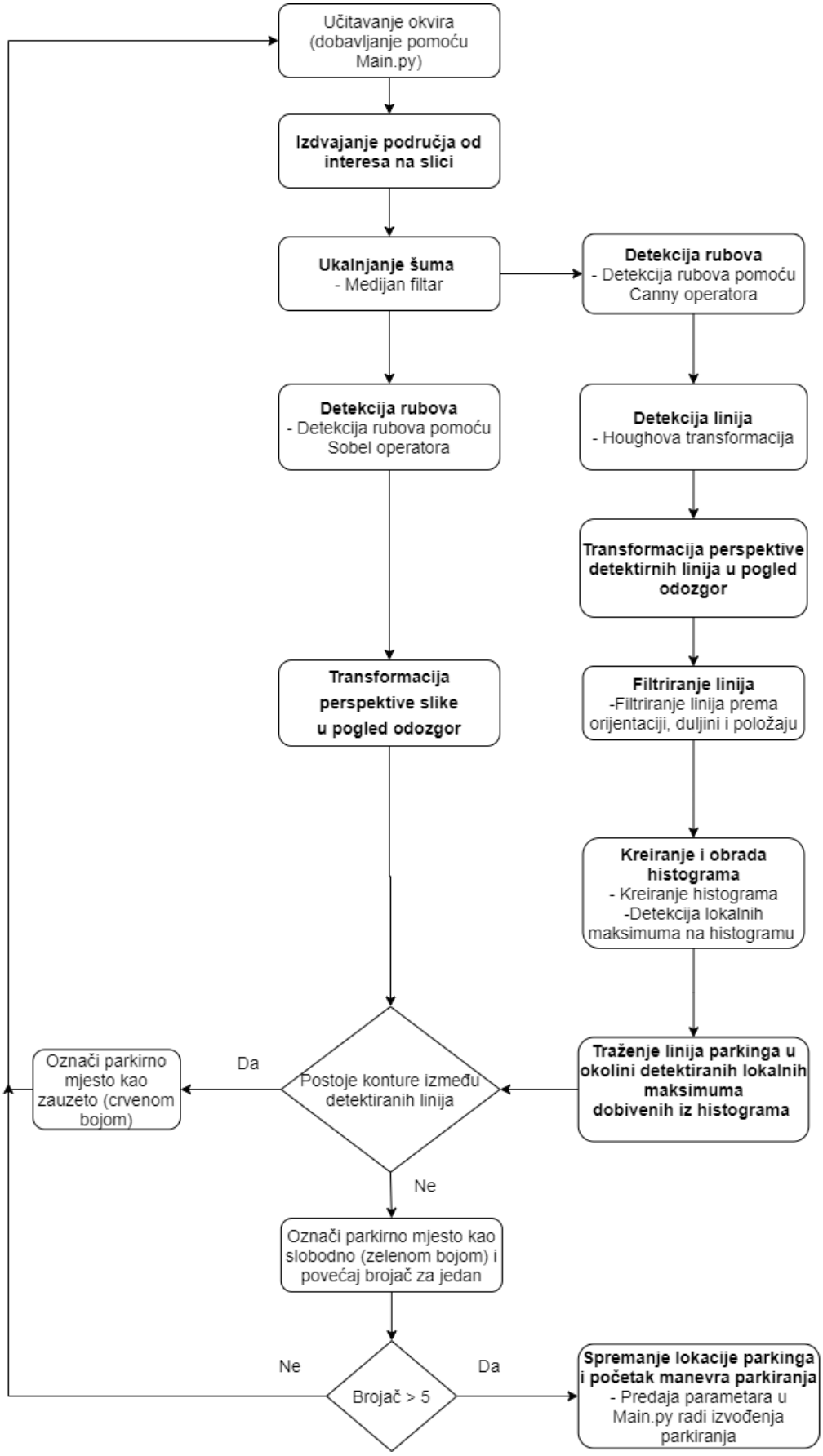
Nakon što se iz niza slika dobivenih s kamere detektira slobodno parkirno mjesto na koje se automobil treba parkirati (a o tome su detalji dani u dijelu 3.2.2.), senzor se isključi kako bi se smanjili zahtjevi za računalnim resursima i kako bi se smanjila pogreška prilikom izvođenja manevra parkiranja te ubrzalo izvođenje ostatka simulacije, o čemu će biti više govora u četvrtom

poglavlju. Naredba za isključivanje senzora je *„sensor.destroy()“*. Kada je senzor isključen, ne poziva se *„do_something()“* funkcija i time dolazi do prekida detekcije parkirnih mjesta. Kada se završi detekcija i isključi senzor, slijedi manevar parkiranja koji je kasnije opisan u dijelu 3.2.3.

3.2.2. Algoritam za detekciju i klasifikaciju parkirnog mjesta

Kompletan algoritam detekcije i klasifikacije parkirnog mjesta sastoji se od niza koraka koji slijedno obrađuju okvire pristigle s kamere vozila. Kao izlaz iz algoritma dobiva se informacija o detekciji i zauzetosti parkinga te obrađena slika na kojoj su označena parkirna mjesta ovisno o njihovoj klasifikaciji. Na slici 3.7. može se vidjeti kako cjelokupni proces izgleda prilikom kretanja vozila uz detekciju parkinga u stvarnom vremenu. Na Slici 3.8. prikazan je dijagram toka algoritma za detekciju i klasifikaciju parkinga. U ovom dijelu, nakon dijagrama toka algoritma detekcije i klasifikacije parkirnog mjesta, bit će detaljno opisan svaki od koraka prikazanih na dijagramu toka, te će za određene dijelove koda biti dane slike koje se dobiju kao rezultat obrade u pojedinim dijelovima koda.

Sl. 3.7. *Prikaz detekcije parkinga u stvarnom vremenu*



Sl. 3.8. *Dijagram toka algoritma za detekciju i klasifikaciju parkirnog mjesta*

Učitavanje okvira i definiranje područja od interesa na slici

Rad algoritma započinje primanjem okvira dobivenog s kamere u CARLA klijentu koji je spojen na prethodno kreirani CARLA server. Okvir kojega algoritam prima je u RGB formatu boja i ima rezoluciju 1280x720 elemenata slike, no dio dobivenog okvira je irelevantan za detekciju parkinga. Pošto je poznat položaj kamere u automobilu, poznato je i na kojem dijelu slike se mogu pojaviti parkinzi od interesa. Polazi se od pretpostavke da se parkirna mjesta nalaze na donjoj polovici slike te se zbog toga dio gornje polovice slike može zanemariti. Kao ROI (engl. *Region of Interest*) uzima se dio ulazne slike koji sadrži donje dvije trećine slike te cijelu širinu slike. Gornja trećina slike se oboja u crno kako bi se uklonio potencijalni šum u detekciji parkinga. Okvir nakon izdvajanja ROI-ja i dalje ima iste dimenzije 1280x720x3, ali su sada gornjoj trećini obrisane sve boje, rubovi, značajke. Na slici 3.9. može se vidjeti kako izgleda ulazni okvir nakon određivanja ROI-ja.

Sl. 3.9. *Ulazni okvir nakon izdvajanja ROI-ja*

Uklanjanje šuma pomoću medijan filtriranja

Medijan filtar je dio pred-obrade slike, uklanja visokofrekvencijski šum iz slike, dok zadržava bitne karakteristike rubova. Medijan filtar je nisko propusni filtar gdje se kao rezultat uzima medijan vrijednosti svih elemenata u *N*x*N* okolini elementa slike [16]. Pokazao se kao bolji izbor od Gaussovog filtra za primjenu detekcije parkinga. Također, medijan filtar veličine 5x5 se pokazao optimalnim za korištenje u ovom algoritmu jer zadržava bitne elemente linija parkinga, dok uklanja neke od rubova susjednih vozila. Do rezultata se došlo empirijski tj.

eksperimentalnom metodom, na način da se uz nepromijenjene ostale dijelove algoritma mijenjao samo tip filtra i njegova veličina. Prilikom testiranja isprobani su Gaussov filtar veličine 5x5 i 9x9 i medijan filtar veličine 3x3, 5x5, 7x7 i 9x9. Na slici 3.10. može se vidjeti okvir nakon medijan filtriranja. Na njemu se vidi kako su linije na parkingu u pozadini zamućene (parking iza promatranog parkinga), što je dobro jer se smanjuje vjerojatnost detekcije tih linija susjednog parkinga prilikom Houghove transformacije. Također, razne refleksije koje se mogu vidjeti na automobilima su također zamućene što smanjuje vjerojatnost pogrešno detektiranih linija (engl. *False Positive)* Medijan filtriranje se nalazi u OpenCV biblioteci i poziva se funkcijom *cv2.meidianBlur().*

Sl. 3.10. *ROI sa slike 3.9 nakon medijan filtriranja*

Detekcija rubova

Proces detekcije rubova predstavlja proces identifikacije i lociranja naglih diskontinuiteta na slici, a vrši se pomoću 2D filtara koji su osjetljivi na velike promjene gradijenata intenziteta elemenata slike [16]. U ovom algoritmu koriste se Sobelov operator i Cannyjev operator. Slika dobivena pomoću Sobelovog filtriranja koristi se za detekciju kontura na slici, a slika dobivena pomoću Cannyjevog filtriranja koristi se za detekciju linija. Nakon primjene Sobelovog operatora dodatno se provodi postupak reskaliranja rubova, te se uvode donji i gornji prag za detekciju rubova, koji iznose 50 i 200. Naime, vrijednosti nakon Sobelovog filtriranja u obrađenoj slici mogu biti u rasponu 0-255. No najjači rubovi ne moraju imati iznos 255 ako nisu toliko izraženi. Zbog toga se provodi postupak reskaliranja na sljedeći način. Na slici na kojoj su prethodno

detektirani rubovi pomoću Sobelovog operatora traži se najjači rub te se svi rubovi skaliraju (normiraju) u odnosu na njega. Ukoliko je vrijednost najjačeg ruba ipak 255, onda zapravo nema skaliranja i svi rubovi ostaju isti. No ukoliko je vrijednost najjačeg ruba manja od 255, vrši se reskaliranje svih rubova na puni opseg i to prema formuli:

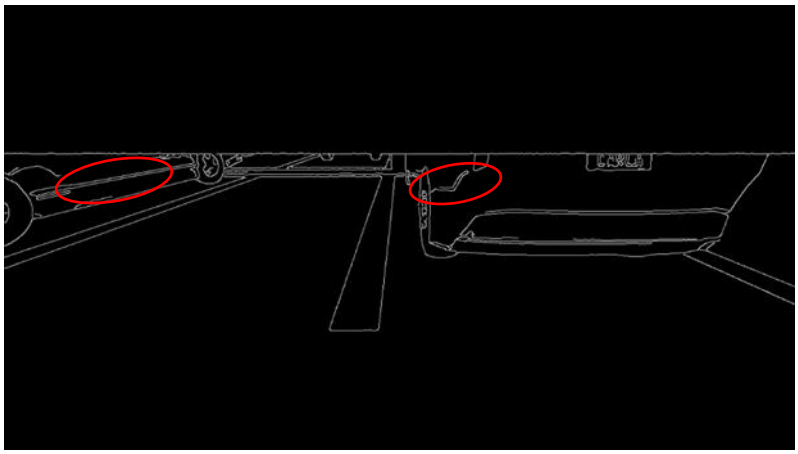
$$255 * Rub/Najjači\ rub. \tag{3-1}$$

Rub je svaka vrijednost na slici nakon Sobelovog filtriranja koja je veća od 0. *Najjači rub* je maksimalna vrijednost ruba koja se pojavljuje u slici nakon Sobelovog filtriranja. Postupkom reskaliranja zapravo se osigurava da su izlazne vrijednosti za rubove uvijek u rasponu 0-255, što omogućava naknadno korištenje jedinstvenih pragova. Kada se završi reskaliranje rubova, vrši se filtriranje rubova prema jačini ruba. Pragovi određuju koje će se vrijednosti rubova zanemariti i to na sljedeći način. Odbacuju se sve vrijednosti manje od donjeg praga (koji iznosi 50) i sve vrijednosti veće od gornjeg praga (koji iznosi 200). Dodavanjem gornjeg praga eliminirani su nagli diskontinuiteti na slici koji su se ponašali kao šum. Kako se sa slike dobivene Sobelovim filtriranjem detektiraju konture koje će služiti za provjeru zauzetosti parkinga, cilj je bio eliminirati sve dodatne konture koje bi mogle utjecati na klasifikaciju parkinga. Kada se na ulaznu sliku primijene Sobelov operator i reskaliranje, vrijednosti elemenata tako obrađene slike spremaju se u novu matricu koja se naziva matrica rubova (horizontalnih ili vertikalnih). U tu matricu se spremaju one vrijednosti koje su veće od donjeg praga za detekciju i koje su manje od gornjeg praga za detekciju. Svim ostalim vrijednostima se dodjeljuje vrijednost 0 i kao takve ih se sprema u matrice. Kako bi se izvršila detekcija rubova pomoću Sobelovog operatora poziva se funkcija *cv2.Sobel()*. Na slici 3.11. su prikazani Sobelovi operatori za detekciju vertikalnih i horizontalnih rubova. Nakon što se kroz sliku *I* prođe pomoću dolje prikazanih operatora i nakon što se zadrže rubovi koji su unutar definiranih vrijednosti detekcije, radi se logička operacija ILI između matrice horizontalnih i matrice vertikalnih rubova. Na mjestima na kojima je jedna (ili obje) od vrijednosti veća od 0, upisuje se 1 u izlaznu matricu.

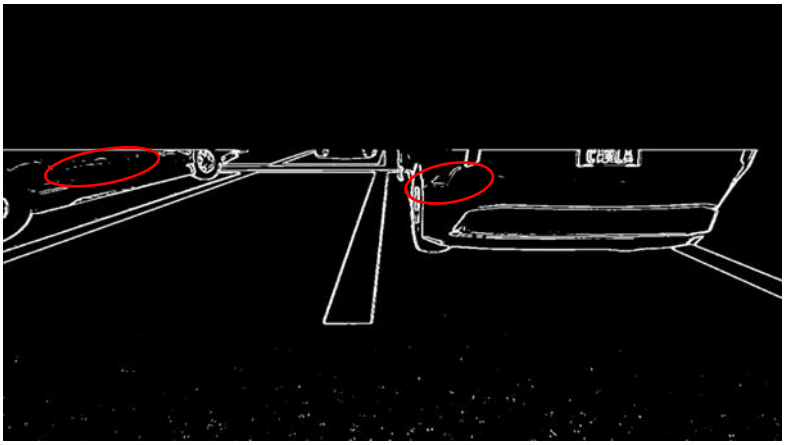
$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * I \qquad G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * I$$

Sl. 3.11. *Sobelov operator za detekciju vertikalnih i horizontalnih rubova*

Ulaz u Cannyjev operator je također ROI slika dobivena nakon medijan filtriranja, na koju se prvo primjenjuje Gaussov filtar za smanjenje šuma, a potom se primjenom metode gradijenata formiraju četiri matrice horizontalnih, vertikalnih i dijagonalnih rubova. Potom se radi mapa položaja rubova i njihove orijentacije. Cannyjev operator koristi gornji (210) i donji prag (60) detekcije rubova. Kod Cannyja svi rubovi na slici veći od gornjeg praga se zadržavaju, svi rubovi manji od donjeg praga se eliminiraju. Svi elementi slike koji se nalaze u okolini rubova većih od gornjeg praga se zadržavaju ukoliko imaju odgovarajući smjer i ukoliko je njihova vrijednost između gornjeg i donjeg praga. Iz navedenog se može primijetiti da se osim medijan filtriranja koristi još jedan oblik niskopropusnog filtriranja i to u vidu Gaussovog filtriranja koji je sastavni dio Cannyjevog operatora, kako bi se dodatno smanjio utjecaj šuma na ulaznoj slici. Slike s detektiranim rubovima se mogu vidjeti na Slikama 3.12. i 3.13. Također, iz dviju navedenih slika vidi se kako okvir obrađen Sobelovim operatorom ima naglašene horizontalne i vertikalne linije, uz nešto više šuma, dok Cannyjev operator bolje detektira dijagonalne rubove (primjer je na slici 3.13 označen crvenom bojom).



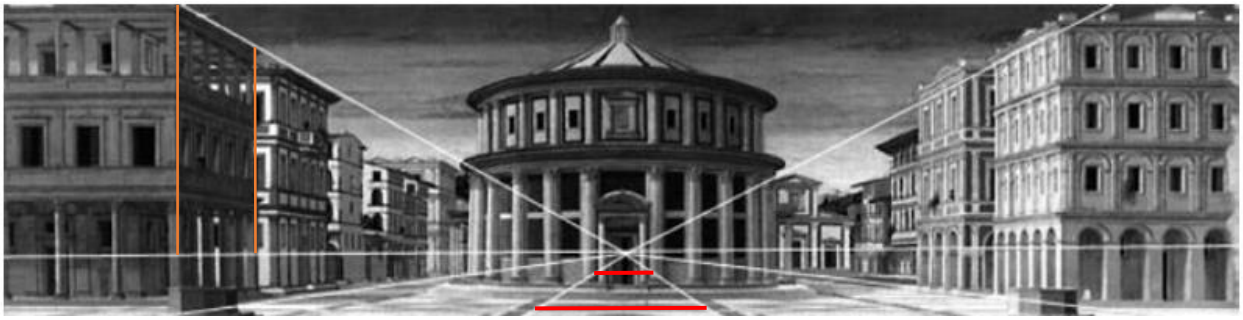
Sl. 3.12. *Izlazni okvir nakon primjene Cannyjevog operatora*



Sl. 3.13. *Izlazni okvir nakon primjene Sobelovog operatora*

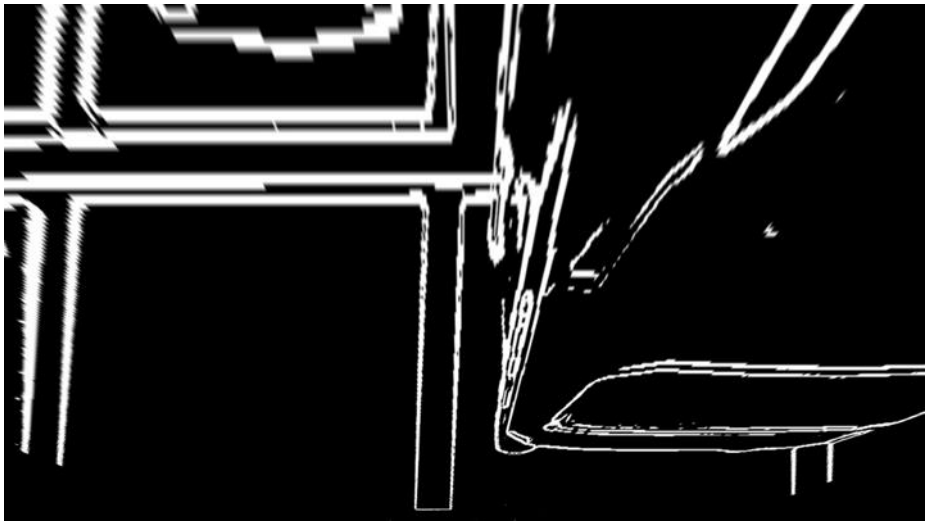
Transformacija perspektive slike s rubovima detektiranim Sobelovim operatorom

Perspektiva označava način prikazivanja trodimenzionalnih objekata na dvodimenzionalnoj plohi slike. Ljudski vid, kao i kamera, funkcionira na takav način. Prilikom obrade slike dolazi se do problema linearne perspektive, koji govori da se dva predmeta jednake veličine udaljavanjem od motrišta linearno smanjuju razmjerno s udaljenošću (Sl. 3.14.) Na slici se vidi da vertikalni pravci pri tome ostaju vertikalni (označeni narančasto), horizontalni ostaju horizontalni (označeni crveno), a slike uzdužnih pravaca postaju dijagonale koje se sijeku u jednoj ili više točaka (bijeli pravci) [17]. Oznake ruba parkinga predstavljaju uzdužne pravce, koji iako su u prirodi paralelni, na ulaznoj slici se međusobno nalaze pod određenim kutom zbog perspektive iz koje su snimane. Kako bi se ispravno detektirali položaji linija, te duljine i širine parkirnih mjesta, potrebno je transformirati perspektivu primljene slike. Perspektiva se transformira u pogled odozgor (engl. *bird's eye perspective*). Takvim pristupom postiže se da linije izgledaju približno paralelno.



Sl. 3.14 *Prikaz linearne perspektive [17]*

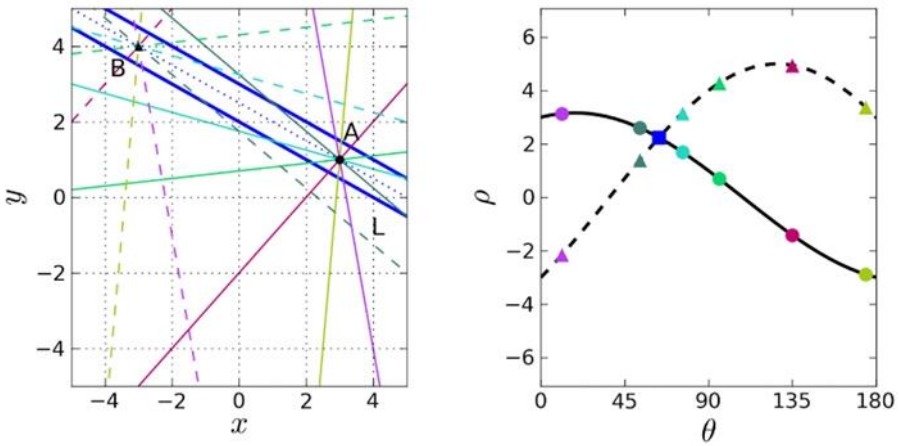
Sve potrebne funkcije za transformaciju perspektive mogu se pronaći unutar OpenCV biblioteke. Korištene funkcije za transformaciju perspektive su *cv2.getPerspectiveTransform() i cv2.warpPerspective()*. Pomoću funkcije *cv2.getPerspectiveTransform(src, dst)* dobiva se transformacijska matrica **M**, a kako bi se izračunala transformacijska matrica potrebno je odrediti *src* i *dst* koordinate slike. *Src* koordinate su koordinate četiriju točaka na izvornoj slici kojima se omeđuje prostor koji se želi transformirati, dok *dst* koordinate predstavljaju koordinate odgovarajućih četiriju točaka na kojima će se nalaziti točke s koordinatama sadržanim u *src* u transformiranoj slici. Poznajući transformacijsku matricu **M**, ulaznu sliku i njezine dimenzije, pomoću funkcije *cv2.warpPerspective()* dobiva se transformirana slika koja je iste rezolucije kao i ulazna slika. U ovom algoritmu se transformira perspektiva slike dobivene nakon detekcije rubova pomoću Sobelovog operatora. Na slici 3.15. može se vidjeti pogled na parkirna mjesta iz ptičje perspektive.



Sl. 3.15. *Izlazni okvir nakon primjene Sobelovog operatora i transformacije perspektive*

Detekcija linija i transformacija perspektive linija

Segmentacija slike je postupak podjele slike u povezane grupe elemenata slike. Detekcija linija predstavlja oblik segmentacije slike prema linearnim strukturama. Metoda korištena za detekciju linija je Houghova transformacija [18]. Detekcija linija pomoću Houghove transformacije se zasniva na tome da se pravac iz *x-y* koordinatnog sustava može preslikati u jednu točku u θ - ρ koordinatnom sustavu (Slika 3.16.) Ako se preslikavanje provede za sve pravce (tzv. familiju pravaca) koji prolaze kroz točke A i B dobit će se dvije sinusoide u θ - ρ koordinatnom sustavu. Točka presjeka u θ - ρ koordinatnom sustavu označava pravac iz *x-y* koordinatnog sustava koji prolazi kroz točke A i B u tom sustavu.



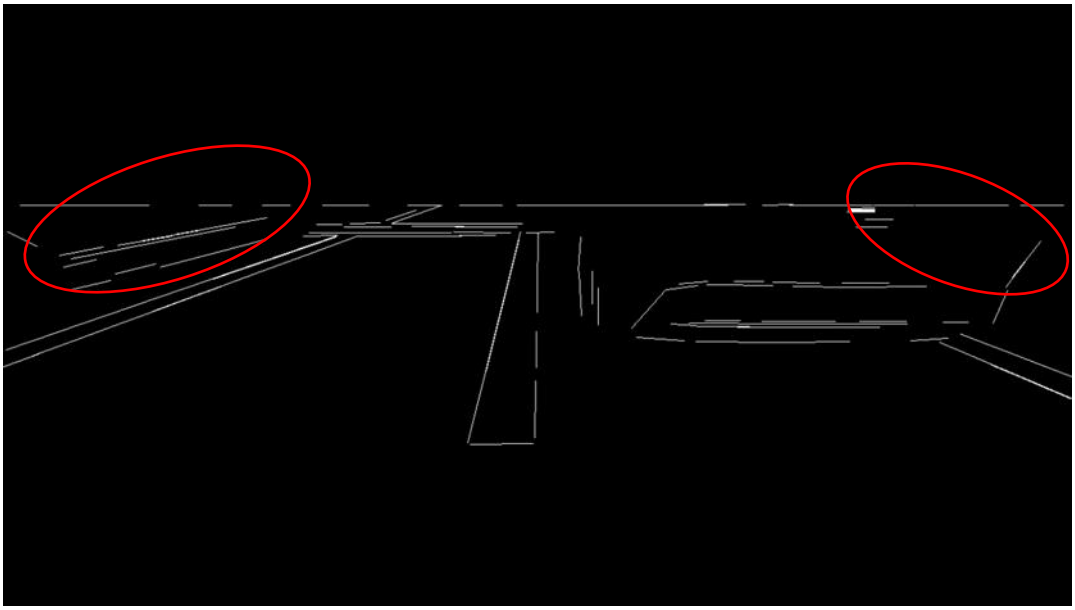
a) b)

Sl. 3.16. (a) *Prikaz grupa pravaca koji prolaze kroz točke A i B u x-y koordinatnom sustavu (b) grupe istih pravaca preslikane u θ - ρ koordinatni sustav Houghovom transformacijom [19]*

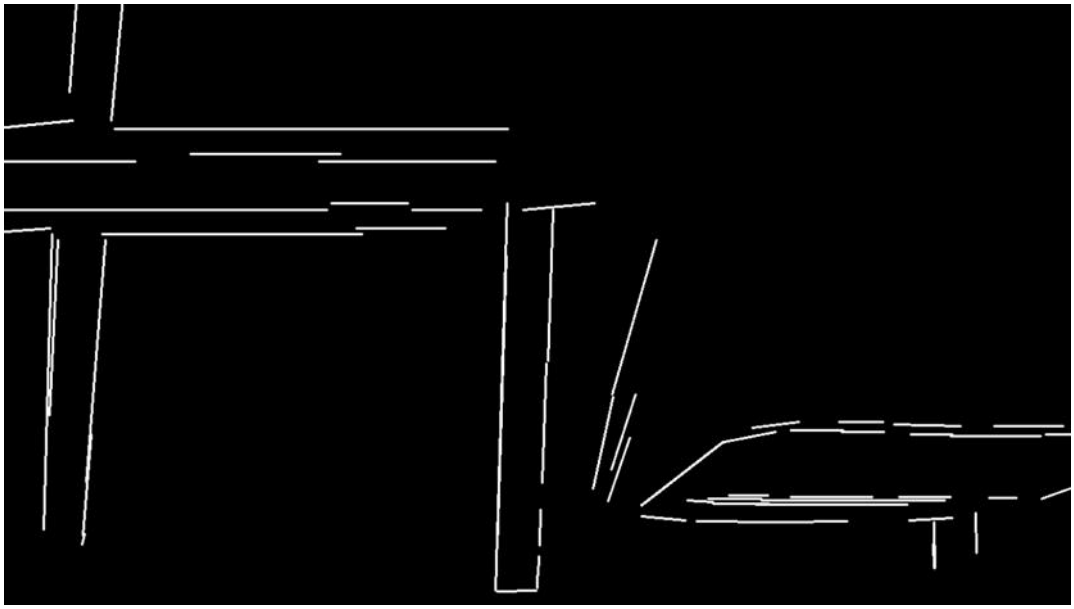
U ovom algoritmu, Houghova transformacija se primjenjuje na sliku na kojoj su se prethodno detektirali rubovi pomoću Cannyjevog detektora. Kako bi se detektirale linije, poziva se funkcija *lines=cv.HoughLinesP(image, rho, theta, threshold[, lines[, minLineLength[, maxLineGap]]])*. Parametri funkcije su:

- *image* - slika na kojoj se vrši detekcija linija, a to je slika na kojoj su prethodno detektirani rubovi pomoću Canny detektora,
- *rho* – korak za ρ – iznosi 1 element slike,
- *theta* – korak za θ – iznosi 1°.
- *threshold* – akumulatorsko polje, svaka oznaka na binarnoj slici koja dobije više od 100 glasova proglašava se linijom
- *minLineLength* – minimalna duljina linije - iznosi 150 elemenata slike,
- *maxLineGap* – maksimalna dopuštena udaljenost između dviju točaka na istoj liniji - iznosi 100 elemenata slike.

Kako bi se neki niz točaka proglasio linijom mora zadovoljavati gore navedene uvjete, a to je da mora postojati više od 100 točaka koje se nalaze na toj liniji, njezina duljina mora biti minimalno 150 elemenata slike, te maksimalna udaljenost između dviju susjednih točaka ne smije biti veća od 100 elemenata slike. Korišteni parametri su dobiveni empirijski i to na način da se uz nepromijenjene parametre *image, rho i theta* mijenjala kombinacija parametara *threshold, minLineLength i maxLineGap.* Smanjenjem navedenih parametara povećava se broj linija koje se detektiraju, a koje nisu relevantne. Parametri su se redom povećavali uz promatranje izlazne slike dok se nije došlo do zadovoljavajućeg rezultata. Izlazna slika nakon Houghove transformacije je prikazana na slici 3.17. Bitno je za napomenuti da slici prije Houghove transformacije nije promijenjena perspektiva. Nakon Houghove transformacije, samo za dobivene linije (Sl. 3.17.) vrši se transformacija perspektive pomoću prethodno izračunate transformacijske matrice **M**. Ovaj korak je potreban kako bi se linije lakše filtrirale prema orijentaciji i duljini. Također, transformacijom perspektive samo detektiranih linija, a ne cijele slike, štedi se određena količina računalnih resursa. Na slici 3.18. vide se linije nakon transformacije perspektive. Linije parkinga su od kosih postale gotovo vertikalne, dok se dio slike koji je irelevantan da detekciju parkinga više ne nalazi na slici (označeno crvenim elipsama na Sl. 3.17.).



Sl. 3.17. *Prikaz linija nakon Houghove transformacije*

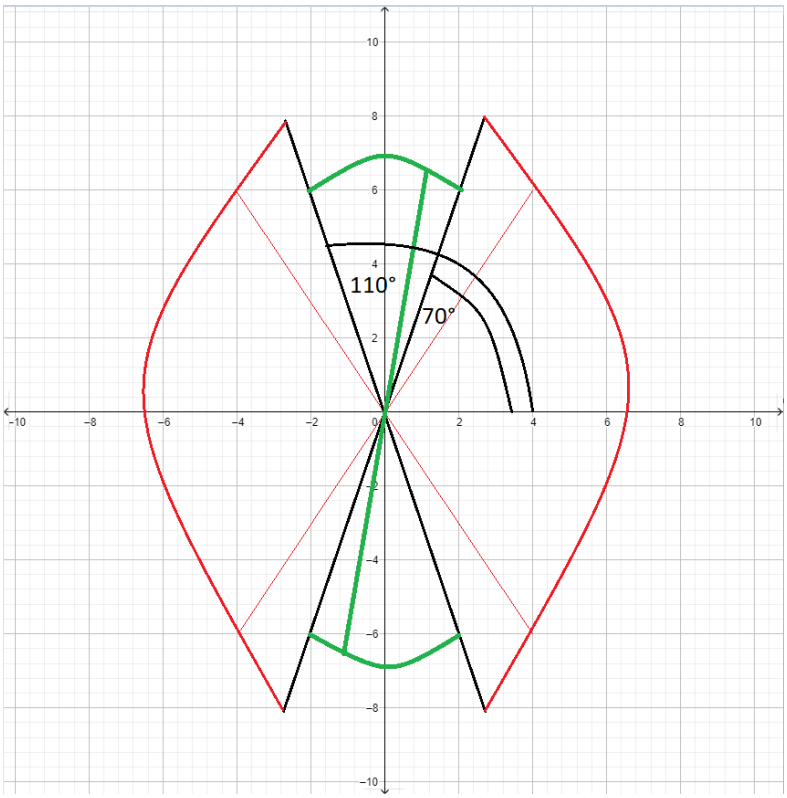


Sl. 3.18. *Prikaz linija nakon Houghove transformacije i transformacije perspektive linija*

Filtriranje linija

Nakon Houghove transformacije dobiva se skup linija, od kojih je dio irelevantan za detekciju parkinga, kao što se može vidjeti na slici 3.18. Kako bi se što točnije odredile koordinate parkirnih mjesta, potrebno je filtrirati linije po određenim kriterijima. U konkretnom algoritmu linije se filtriraju prema trima kriterijima. Prvo se odbacuju sve linije čiji se i početci i krajevi nalaze u gornjoj polovici slike, jer se polazi od pretpostavke da se parkirna mjesta od interesa

nalaze niže u okviru te da se prostiru maksimalno do dvije trećine visine slike (zbog samog poznatog položaja kamere u vozilu). Drugi kriterij je filtriranje linija po orijentaciji, s obzirom na to da se pretpostavlja da će se linije parkinga nakon transformacije perspektive nalaziti pod kutom od 90° u odnosu na horizontalnu os. Odbacuju se sve linije koje imaju kut manji od 70° kao i sve linije koje imaju kut veći od 110°, kao što se može vidjeti na skici (Slika 3.19.) Linije koje su unutar područja označenog crvenim lukovima se odbacuju, dok se linije unutar zelenih lukova zadržavaju.



Sl. 3.19. *Prikaz linija koje se odbacuju (crvene) i linija koje se zadržavaju (zelena)*

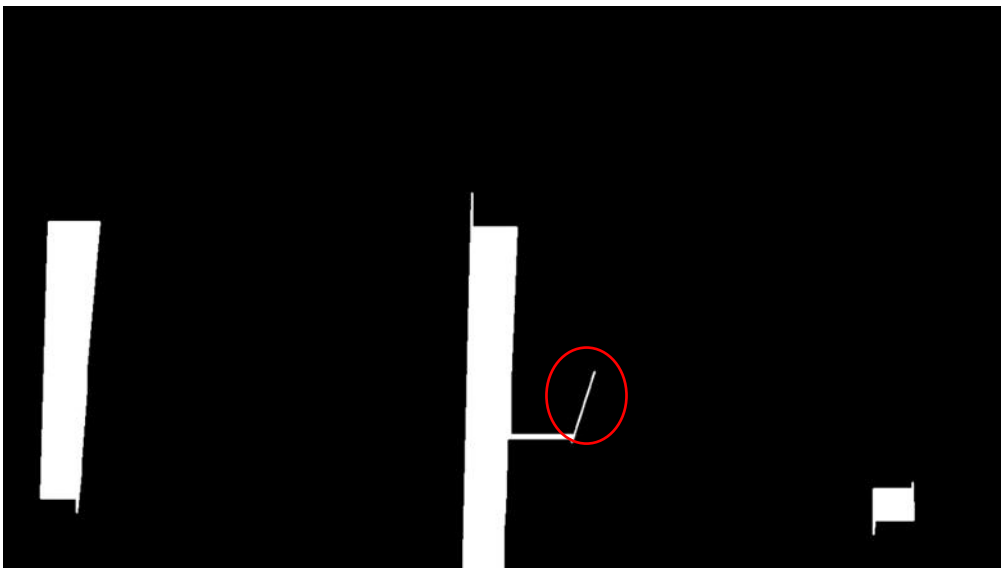
Treći kriterij je filtriranje linija po duljini. Sve linije koje su manje od 15 elemenata slike se odbacuju. Do tog broja se došlo eksperimentalnom metodom. Naime, na vozilu i u okolini vozila postoji mnogo linija koje su u našem slučaju šum. Promatranjem niza slika nakon detekcije linija, došlo se do zaključa da je većina tih linija u rasponu 5 – 20 elemenata slike. Postavljanjem filtra na nešto veću razinu, poput 20, prilikom filtriranja su odbacivane kraće linije koje su bile sastavni dio linija parkinga. S druge strane, smanjenjem tog broja na manje od 15 i dalje se odbacuju neke od linija koje su dio linija parkinga, ali u znatno manjoj mjeri, dok se istovremeno linije koje predstavljaju šum dovoljno dobro eliminiraju.

Iz svega navedenog može se zaključiti da će nakon ovih filtriranja linija ostati samo vertikalne dugačke linije koje se nalaze u donjem dijelu transformirane slike. Te linije su prikazane na slici 3.20.

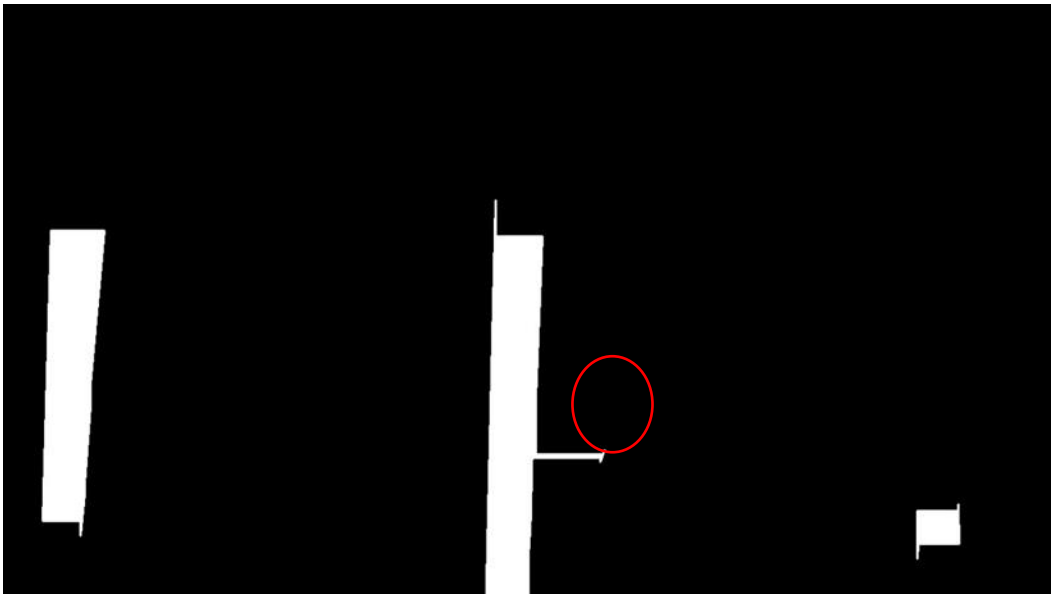
Sl. 3.20. *Prikaz linija dobivenih Houghovom transformacijom preostalih nakon filtriranja*

Budući da i među filtriranim linijama može biti onih koje zadovoljavaju sve uvjete filtriranja, a da pritom ipak nisu dio linija parkirnog mjesta (označena na slici 3.20.), poseže se za još jednom vrstom filtriranja koje se naziva morfološko filtriranje. Prilikom morfološkog filtriranja koristi se korelacija susjednih elemenata slike, dakle dvije paralelne (dovoljno bliske) linije će se prilikom morfološkog filtriranja ispuniti tj. zatvoriti, dok će se nekakve linije koje su dovoljno udaljene od oznaka parkinga i koje nemaju susjednih linija oko sebe obrisati. Morfološko filtriranje dio je post-procesnih metoda. U ovom algoritmu koristi se funkcija morfološkog filtriranja *cv2.morphologyEx()* dostupna u OpenCV biblioteci. Slika se dva puta morfološki filtrira. Prvi puta se koristi maska koja zatvara susjedna područja sa sličnim karakteristikama, a drugi puta se koristi maska koja uklanja šum. Kod korištenja filtra zatvaranja maska je veličine 80x80 elemenata slike. Do broja se došlo eksperimentalnim putem, testirano je pomoću maski manje veličine (60x60 i 70x70), ali one u nekim slučajevima nisu zatvarale površinu između dviju linije koje se karakteriziraju kao oznake parkinga. Zbog toga se poseglo za nešto većim filtrom kako bi se osiguralo da se svaka oznaka parkinga u potpunosti zatvori. Prilikom korištenja morfološkog filtra polazi se od pretpostavke da se svaka od traka parkirnog mjesta sastoji od dviju linija koje su vrlo bliske i slične po svom kutu i duljini, a njihova širina je između 60 i 70 elemenata slike. Do tih brojeva se došlo na način da su promatrane izlazne slike za različite udaljenosti vozila

od parkinga i jednostavnim brojanjem se utvrdilo da se širina linija parkirnog mjesta kreće između ta dva navedena broja za različite položaje oznaka parkinga na slici i za različite udaljenosti vozila od parkinga. Nakon što se spoje sve takve linije (Slika 3.21.), koristi se maska koja otvara (briše) sve linije koje su dimenzijama manje od 40x40 elemenata slike. Također, kao i za prethodne maske do ovog broja se došlo eksperimentalnim putem. Prilikom korištenja nešto većih maski u nekim slučajevima su brisane oznake parkinga, tako da se odabrao najveći filtar prilikom čijeg korištenja nije dolazilo do brisanja oznaka parkinga. Tim postupkom se brišu sve linije koje su pretanke da bi bile oznake parkinga, i sve linije koje su izolirane na slici. Slika 3.22. prikazuje izlazni okvir nakon drugog morfološkog filtriranja.



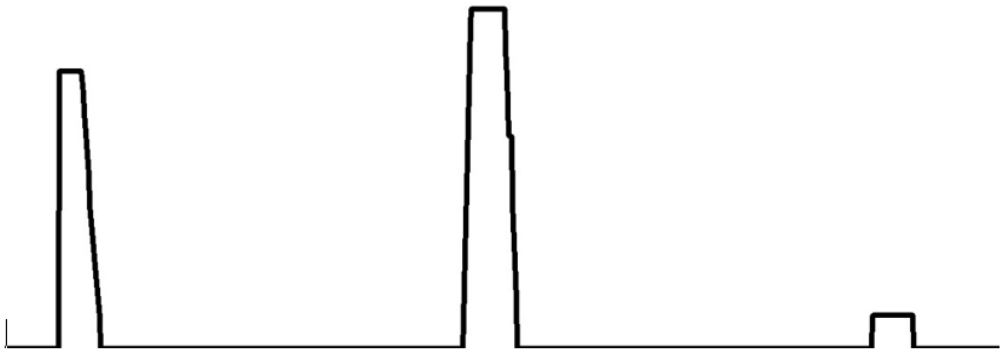
Sl. 3.21. *Izlazni okvir nakon morfološkog filtriranja zatvaranja*



Sl. 3.22. *Izlazni okvir nakon drugog morfološkog filtriranja*

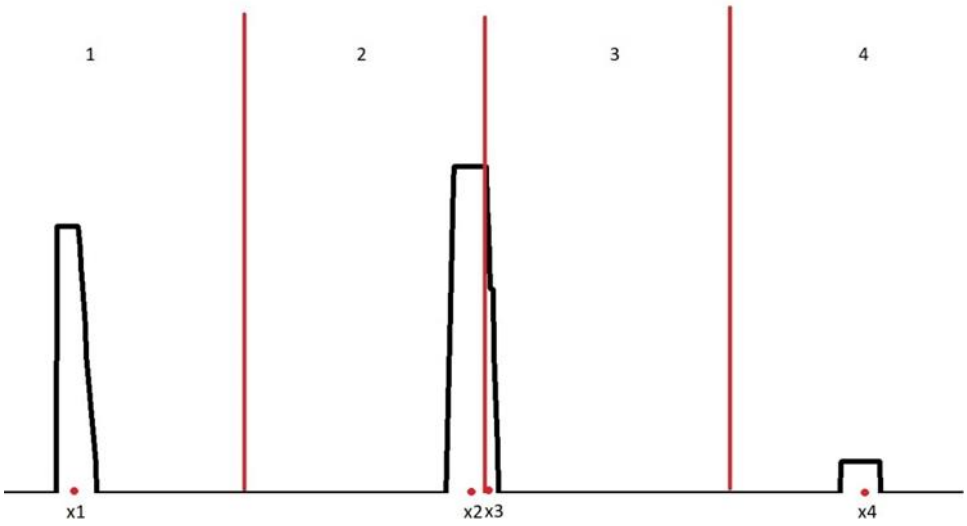
Kreiranje i obrada histograma

Kako bi se odredile *x* koordinate oznaka parkirnih mjesta potrebno je kreirati histogram koji prikazuje distribuciju bijelih točaka sa slike 3.22. Te bijele točke predstavljaju potencijalne oznake parkirnih mjesta. Histogram predstavlja grafički prikaz distribucije podataka po jednoj od osi. U konkretnom slučaju radi se o podatcima iz binarne slike dobivene filtriranjem linija. Ulazna slika ima dvije dimenzije, *x i y*, gdje *x* predstavlja broj stupaca, a *y* broj redaka. Histogram se kreira na način da se kreće s lijeva na desno i za svaki stupac se zbraja koliki je broj elemenata slike koji predstavljaju bijele točke u tom stupcu. Na slici 3.22. se vidi izgled binarne slike dobivene filtriranjem linija, a na slici 3.23. nalazi se njezin histogram. Iz priloženog histograma se vidi da su maksimumi histograma tamo gdje se nalaze linije parkinga. Prilikom kreiranja histograma zanemaruje se prvih deset stupaca i posljednjih deset stupaca te se njihova vrijednost postavlja na nulu jer se prilikom korištenja maske zatvaranja (maskClose) spajaju sva područja koja se nalaze blizu krajeva slike. Također se vidi, da ukoliko postoji bilo kakav šum, njegova amplituda na histogramu je zanemariva u odnosu na amplitudu linija parkinga. Može se reći kako je histogram u ovom slučaju poslužio kao još jedna vrsta post-obrade, tj. filtriranja.



Sl. 3.23. *Histogram okvira nakon morfološkog filtriranja*

Za *x* koordinate oznaka parkinga uzimaju se točke na kojima se nalaze lokalni maksimumi iz prethodno dobivenog histograma, i to na način da se histogram podijeli u četiri područja jednake veličine (svako je veliko 320 elemenata slike) te se na svakom od tih područja traži lokalni maksimum. Lokalni maksimumi se traže u četirima područjima, tj. traže se maksimalno četiri potencijalne *x* koordinate za oznake parkirnog mjesta iz razloga što se pretpostavlja da se u kadru mogu nalaziti maksimalno tri parkirna mjesta koja su omeđena pomoću četiriju linija. Kako bi se zaključilo da li su dobivene *x* koordinate uistinu koordinate oznaka parkirnih mjesta, provjeravaju se dodatni uvjeti koji su niže opisani. Primjer odabira *x* koordinata za svako od četiriju područja prikazan je na skici (Sl.3.24.).



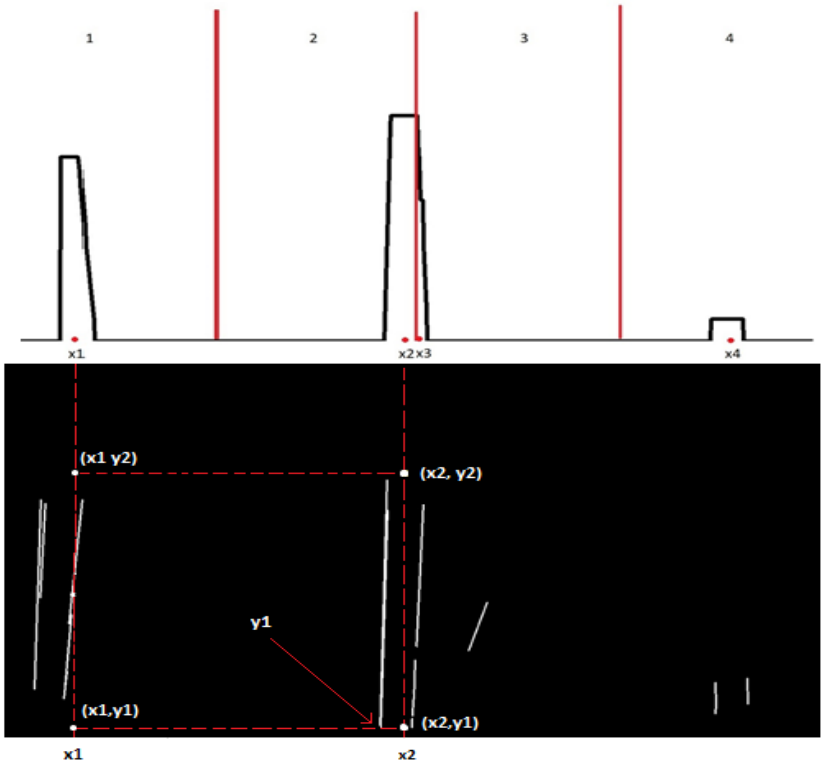
Sl. 3.24. *Podjela histograma u četiri jednaka područja i odabir lokalnih maksimuma*

Traženje linija parkinga u okolini detektiranih lokalnih maksimuma dobivenih iz histograma

Kako bi se sa sigurnošću moglo tvrditi da su *x* – koordinate dobivene iz histograma zaista koordinate oznaka parkirnih mjesta, provjeravaju se posljednja dva uvjeta. Prvi je da širina parkinga, tj. udaljenost između dviju oznake parkirnog mjesta mora biti određenih dimenzija. U konkretnom slučaju ona iznosi minimalno 420 elemenata slike, a maksimalno 600 elemenata slike. Do ovih brojeva se došlo empirijski proučavanjem različitih slučajeva parkirnih mjesta. Promatranjem izlaznih okvira utvrdilo se da širina parkinga prilikom kretanja vozila varira unutar gore navedenih vrijednosti. Drugi uvjet koji mora biti zadovoljen je taj da u okolini *x* koordinata (lijevo-desno) dobivenih iz histograma, moraju postojati prethodno detektirane linije. Kako je ranije u radu utvrđeno da je oznaka (linija) parkirnog mjesta široka između 60 i 70 elemenata slike, za očekivati je da će se u okolini lijevo ili desno $\pm$35 elemenata slike od *x* koordinata dobivenih iz histograma nalaziti linije koje su prethodno detektirane pomoću Houghove transformacije, a koje omeđuju oznake parkirnog mjesta. Na slici 3.25 vidljivo je kako u okolini koordinata *x1, x2* (gledajući samo *x* os) postoje detektirane linije (one postoje i za koordinate *x3* i *x4* iz histograma sa slike 3.24, ali će ovdje radi jednostavnosti fokus biti samo na koordinatama *x1 i x2*). Na slici se također vidi da se linija koja počinje najniže nalazi u okolini koordinate *x2* i da počinje od koordinate *y1* (gledajući *y* os) i ona predstavlja konačnu *y* koordinatu početka parkinga. Gornja *y* koordinata (*y2*), tj. koordinata kraja parkinga, dobiva se na način da se proračunava iz *y1* koordinate, uz poznate parametre kamere i pretpostavku da duljina okomitog parkirnog mjesta

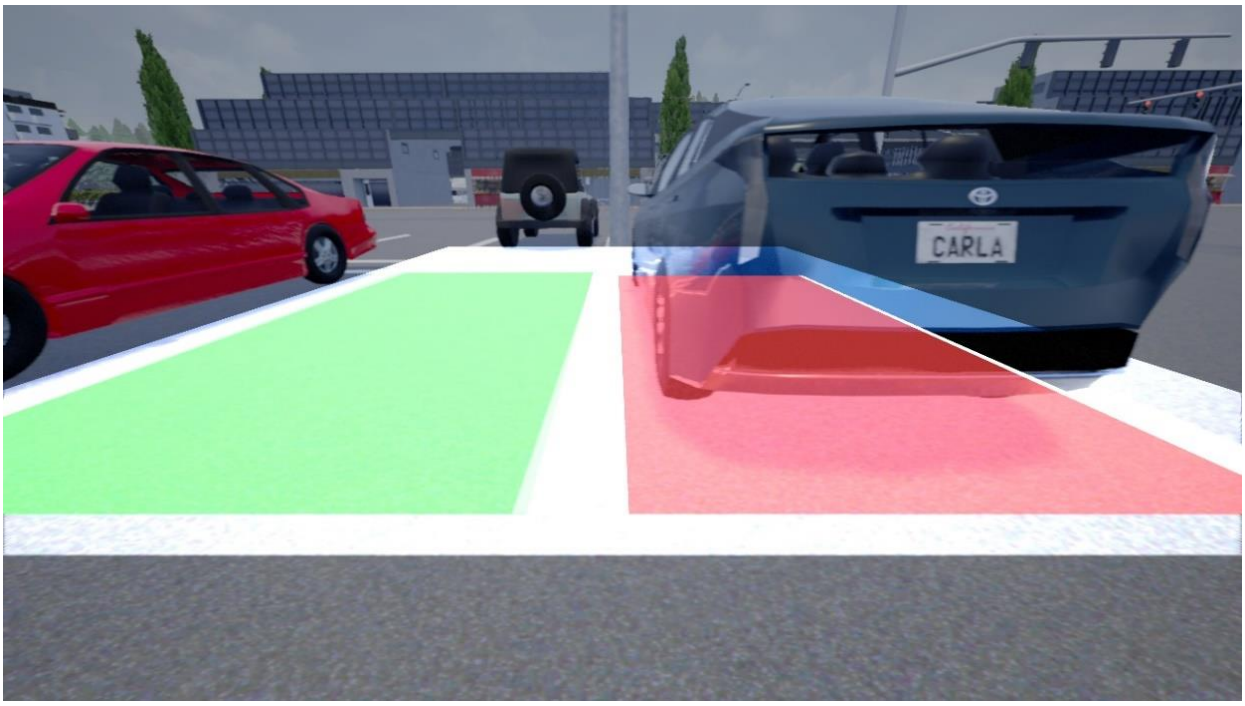
iznosi 5 m. Nakon što se izračuna *y2* koordinata, poznate su sve točke koje omeđuju parkirno mjesto (pravokutnik omeđen bijelim točkama na slici 3.25.). Tako dobivene koordinate se spremaju u spremnike (engl. *buffer*) kako bi se izgladile eventualne pogreške prilikom detekcije linija parkinga u sljedećem okviru. Nakon što se obrade koordinate *x1* i *x2*, isti proces se odvija i za koordinate *x3* i *x4*. Koordinata *x3* će biti odbačena kao kandidat jer se nalazi preblizu koordinati *x2*, dok će koordinata *x4* sa koordinatama *x2, y1* i *y2* tvoriti četiri točke drugog parkinga na slici.

Nakon što su poznate sve četiri točke koje omeđuju parkirno mjesto (Slika 3.25.), detektiraju se konture kako bi se utvrdilo da li je prostor između parkirnih linija slobodan ili zauzet (klasifikacija detektiranog parkirnog mjesta). Za detekciju kontura koristi se slika dobivena nakon detekcije rubova pomoću Sobelovog operatora na koju je primijenjena transformacija perspektive (Sl. 3.15.). Unutar zadanih koordinata traže se konture i broje elementi slike tih kontura. OpenCV funkcija za traženje kontura je *cv2.findContours()*. Nakon detekcije svih kontura vrši se podjela kontura na „velike“ i „male“. Kako bi se parking proglasio zauzetim potrebno je postojanje barem jedne velike konture (tj. konture koja sadrži više od 100 elemenata slike), ili više manjih kontura duljine od 35 do 100 elemenata slike, čiji ukupni broj elemenata slike prelazi brojku 100. U suprotnom se parking proglašava slobodnim. Zauzeti parking se boja crvenom bojom, a slobodni zelenom bojom.



Slika 3.25. *Određivanje koordinata parkinga*

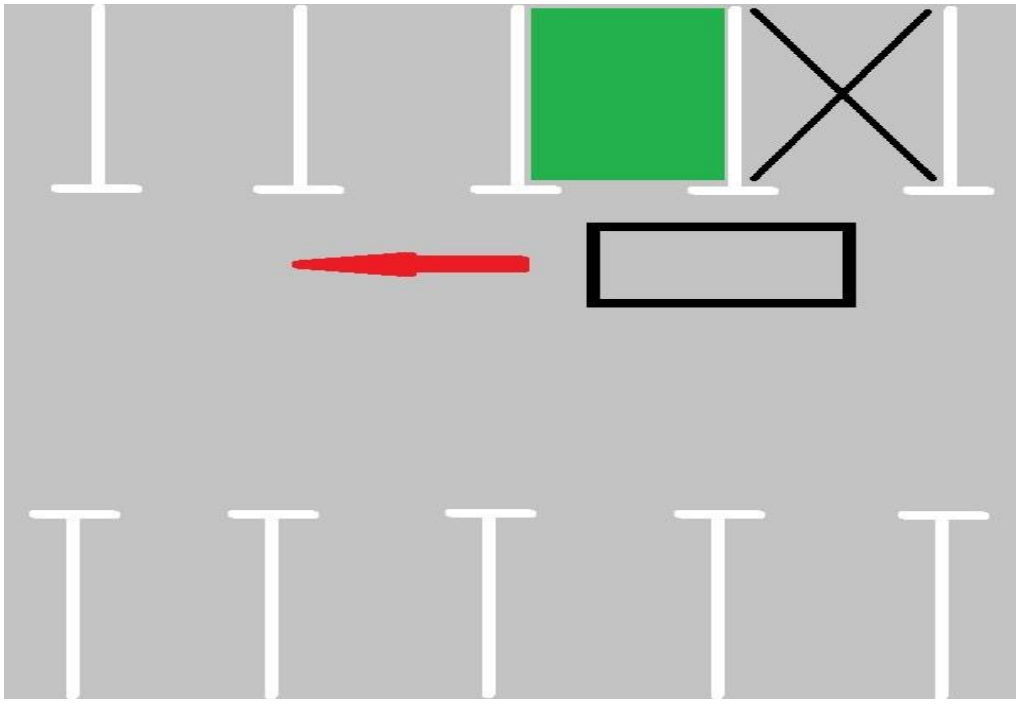
Da bi se pokrenuo manevar parkiranja, potrebno je minimalno 5 klasifikacija istog slobodnog parkinga u 5 uzastopnih okvira u nizu. Slika 3.26. prikazuje detektirane i klasificirane parkinge.



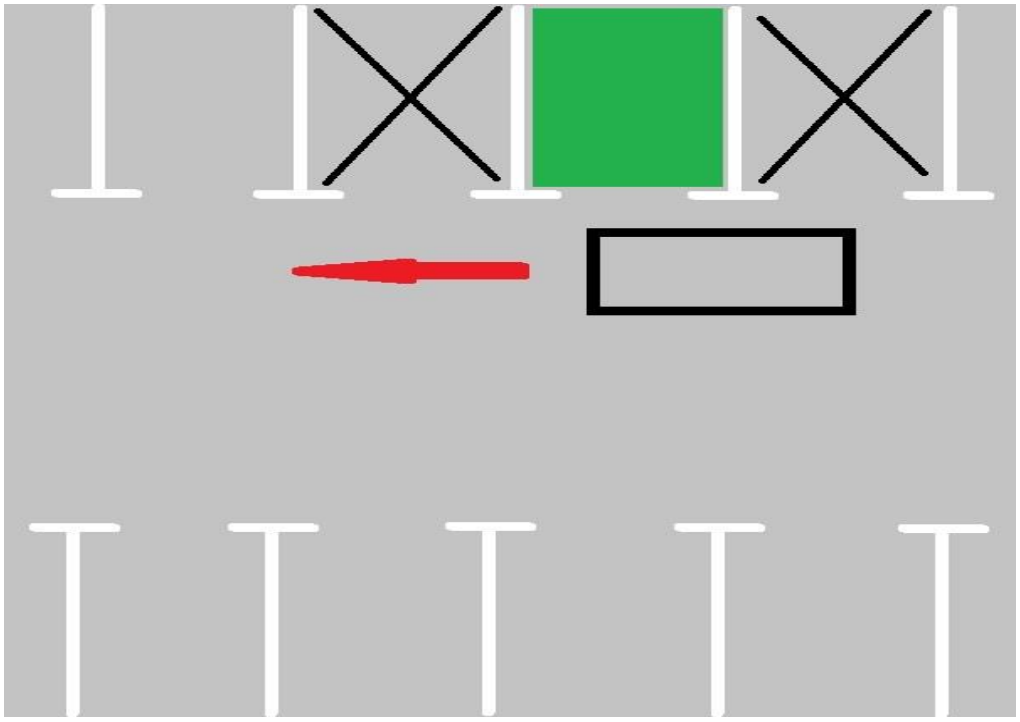
Sl. 3.26. *Izlaz iz algoritma detekcije i klasifikacije parkirnog mjesta*

3.2.3. Algoritam manevriranja vozilom kako bi se ono parkiralo

Kada se sa sigurnošću utvrdi da je određeno parkirno mjesto slobodno, potrebno je uparkirati vozilo na isto. Parkiranje vozila je složen zadatak koji uključuje fuziju raznih senzora poput GPS-a, radara, ultrazvučnih senzora, kamera itd., uz složene algoritme praćenja parkinga i procjene udaljenosti od susjednih vozila, te algoritma zakretanja kotača u ovisnosti o trenutnom položaju vozila. Kako ovakav pristup premašuje okvire ovog rada, poseglo se za nešto jednostavnijim načinom parkiranja. Naime, kako bi se zadatak pojednostavio kreirane su gotove sekvence parkiranja, po tri za svaku stranu vozila. Od toga su dvije sekvence za parkiranje unaprijed. Jedna se koristi kada je parking koji slijedi iza detektiranog slobodnog parkinga također slobodan (Slika 3.27., na njoj X u parkirnom mjestu znači da je zauzet), a druga kada se vozilo mora uparkirati na mjesto kraj kojega postoje parkirana vozila te je potreban složeniji manevar (Slika 3.28.). Za parkiranje unatrag postoji samo jedan manevar neovisno o postojanju vozila na susjednim parkirnim mjestima.



Sl. 3.27. *Skica parkinga gdje je parkirno mjesto koje se nalazi odmah nakon detektiranog slobodnog parkirnog mjesta također slobodno*

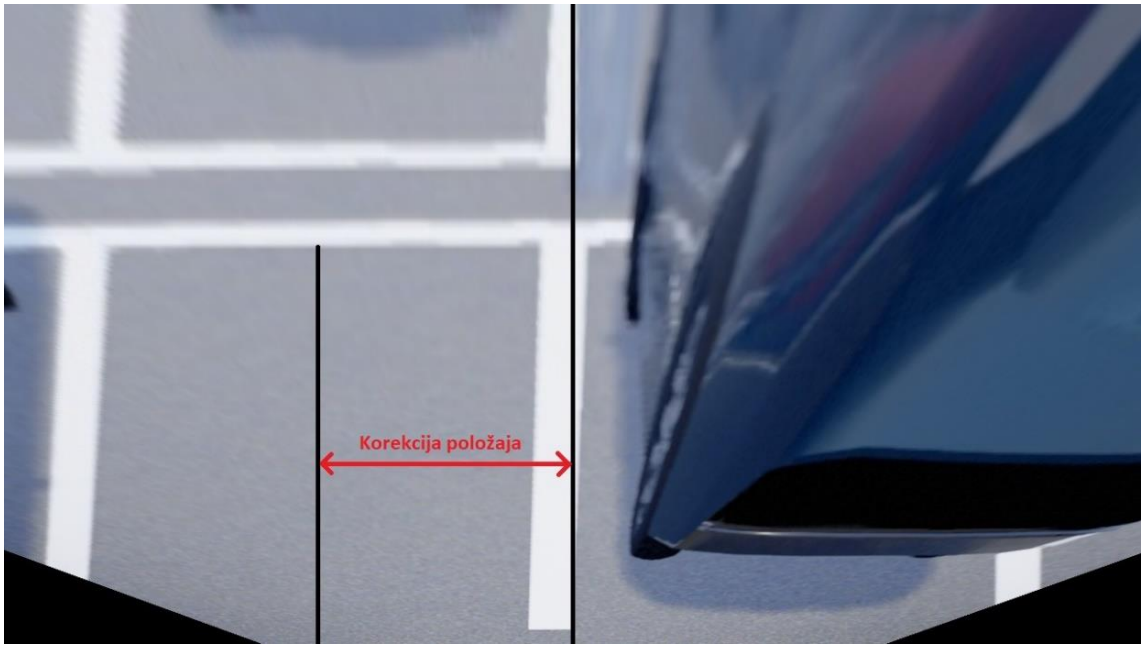


Sl. 3.28. *Skica parkinga gdje je parkirno mjesto koje se nalazi odmah nakon detektiranog slobodnog parkirnog mjesta zauzeto*

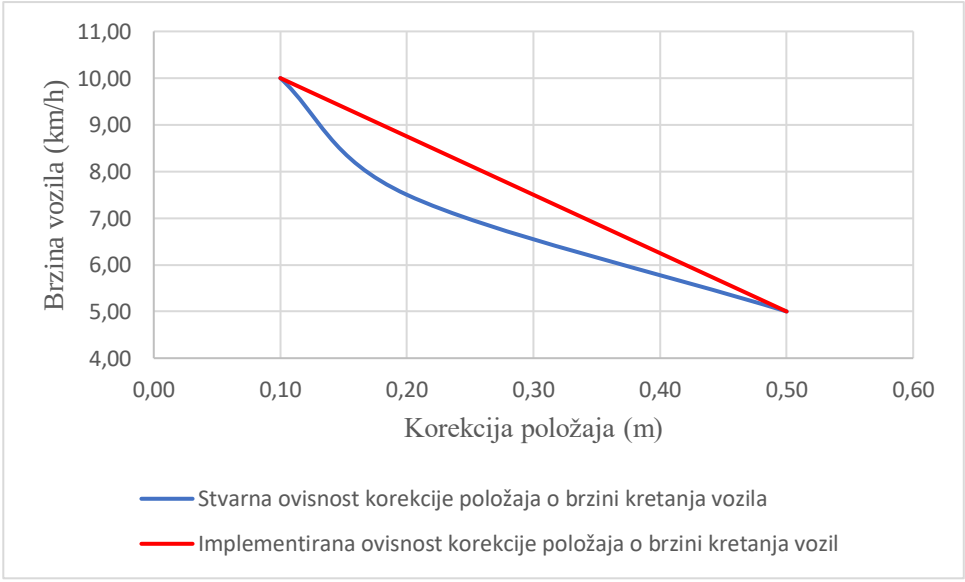
Detektiranje parkinga i manevri parkiranja izvedeni su u CARLA simulatoru. Kada se detektira i klasificira slobodno parkirno mjesto (nakon 5 klasifikacija parkinga kao slobodnog u 5 uzastopnih okvira), algoritam pamti trenutnu lokaciju vozila na kojoj je do toga došlo (koordinate položaja vozila u tom trenutku). Prilikom razvoja rješenja primijećeno je da za iste uvjete okoline i istu brzinu kretanja vozila, algoritam detektira i klasificira isto slobodno parkirno mjesto u nešto drugačijim vremenskim trenucima (tj. koordinate položaja vozila u trenutku klasifikacije istog slobodnog parkirnog mjesta nisu uvijek identične za iste uvjete okoline i vožnje). Zbog toga je bilo potrebno tu činjenicu uzeti u obzir, a to je učinjeno na način da se na položaj vozila zabilježen u trenutku klasifikacije parkinga kao slobodnog, dodaje određena korekcija. Na posljednjoj slici detekcije i klasifikacije parkinga promatra se koliko je središte promatranog parkinga (kada se parking promatra duž *x* osi) udaljeno od središta samog okvira (crni pravci na slici 3.29.). Iz poznatih parametara kamere izračuna se kolika je ta udaljenost duž *x* osi i ta se udaljenost u metrima dodaje kao korekcija položaja (označena crvenom bojom) na *x* koordinatu položaj vozila zabilježenu u trenutku klasifikacije parkinga kao slobodnog.

Prilikom razvoja rješenja utvrđeno je da vozilo ne parkira dobro (između linija) za različite brzine kretanja ako se koristi samo korekcija položaja zbog trenutka klasifikacije. Stoga se uvela i korekcija položaja zbog brzine kretanja vozila. Razlog zbog kojeg se morala uvesti je taj što vozilo prelazi određeni put za vrijeme obrade svakog od okvira (prilikom detekcije i klasifikacije parkinga), ali je taj put znatno veći za veće brzine te se unosi dodatna greška prilikom spremanja lokacije (kada se detektira i klasificira slobodno parkirno mjesto). Korekcija položaja zbog brzine kretanja vozila se ne mijenja linearno s povećanjem brzine, ali radi jednostavnosti je u ovom radu linearizirana i to na način da je određen pravac koji prolazi kroz dvije točke definirane dvjema krajnjim brzinama (5km/h i 10km/h) pa se stoga unosi pogreška prilikom parkiranja pri brzini od 7.5km/h. Na slici 3.30. dana je skica linearizacije korekcije položaja ovisno o brzini kretanja vozila. Na *y* osi je prikazana brzina kretanja vozila, a na *x* osi korekcija položaja zbog brzine kretanja vozila. Plava krivulja prikazuje stvarnu ovisnost korekcije položaja zbog brzine kretanja vozila, a crveni pravac prikazuje kako je korekcija položaja izračunata i implementiran u algoritmu.

Shodno prethodno opisanom, na lokaciju na kojoj je klasificirano slobodno parkirno mjesto dodaju se korekcija položaja zbog trenutka klasifikacije parkinga i korekcija položaja zbog brzine kretanja vozila. Tako korigirana lokacija predstavlja lokaciju duž *x* osi na kojoj će vozilo početi skretati prema parkingu kada se bude parkiralo unaprijed.



Sl. 3.29. *Korekcija položaja vozila na kojoj će započeti kasnije skretanje prema detektiranom slobodnom parkingu u odnosu na njegov položaj prilikom klasifikacije parkirnog mjesta kao slobodnog*

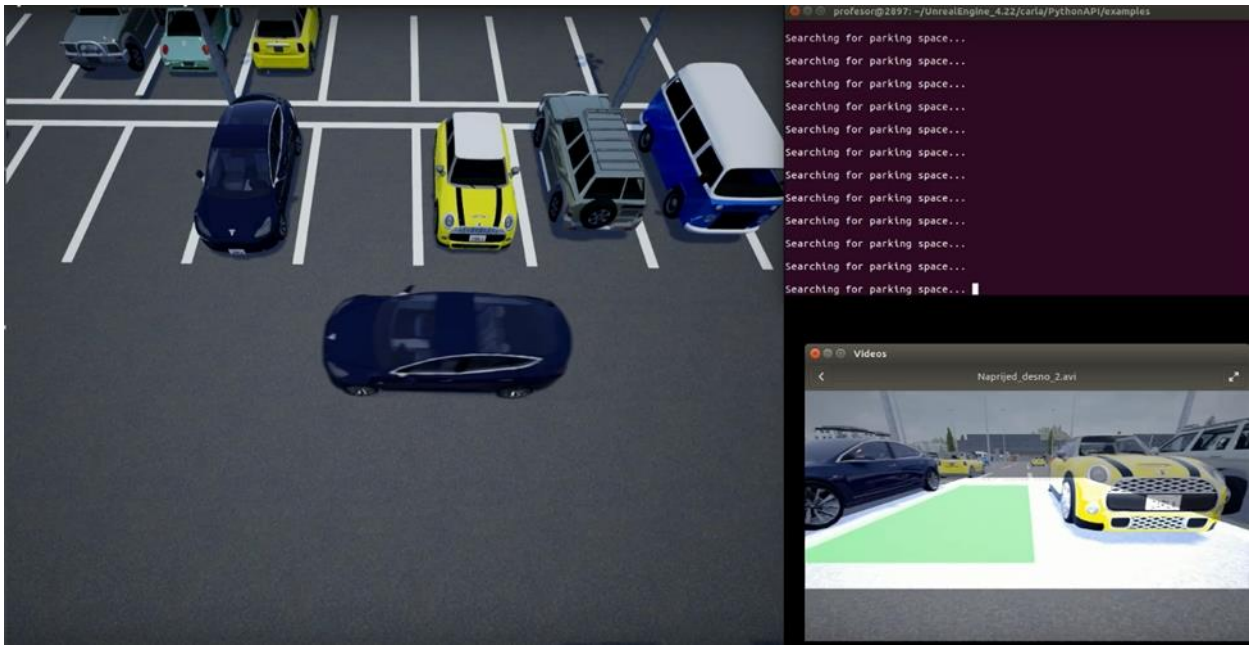


Sl. 3.30. *Stvarna ovisnost korekcije položaja o brzini kretanja vozila te aproksimirana ovisnost implementirana u predloženom algoritmu autonomnog parkiranja*

Za naredbe manevriranja vozilom koristi se funkcija implementirana u CARLA simulatoru. Primjer funkcije: *vehicle.apply_control(carla.VehicleControl(throttle,brake, steer, reverse))*. Funkcija se poziva u više navrata uz zadane parametre i predefiniranu vremensku

odgodu između poziva kako bi se izvršio željeni manevar. Vremenska odgoda se koristi kao dio manevriranja jer se svaki dio manevra izvršava specifično dugo vremena, npr. automobil ide ravno dvije sekunde, zatim skreće jednu sekundu pa potom staje. Nakon što se izvede cijela sekvenca funkcija za kontrolu vozila, isto će se nalaziti parkirano na prethodno detektiranom parkirnom mjestu. U prilogu P.3.2. nalaze se video materijali na kojima se mogu vidjeti implementirani različiti manevri za različite scenarije i različite brzine vozila. Detekcija parkirnog mjesta i pripadni manevar parkiranja bit će prikazani kroz niz slika.

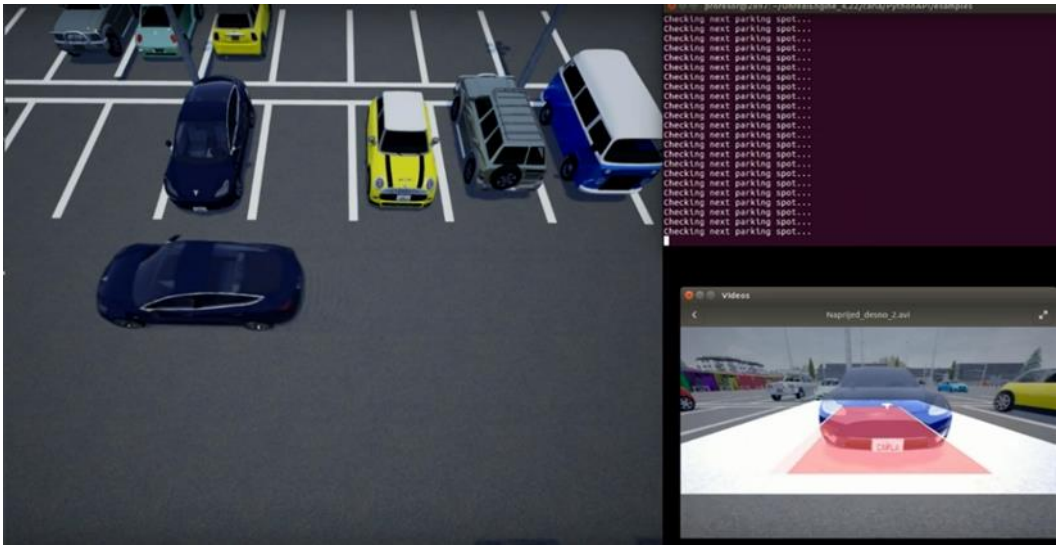
Nakon što se vozilo i njemu pripadni senzor stvore na mapi u simulatoru, poziva se funkcija za pokretanje vozila *vehicle.apply_control(carla.VehicleControl(throttle=0.3,brake =0.0, steer=0.0, reverse=False))*. Ovom funkcijom zadaje se naredba vozilu da se kreće ravno brzinom 7.5kmh. Parametri funkcije *throttle* i *brake* poprimaju vrijednosti [0,1], parametar *steer* poprima vrijednosti [-1, 1], a *reverse* 0 ili 1. Osim pokretanja vozila potrebno je u uključiti rad senzora za detekciju parkinga pomoću funkcije *sensor.listen(lambda data: process_img(data))*. Pozivanjem ove funkcije slike snimljene kamerom šalju se na obradu, gdje se parkirno mjesto detektira i klasificira. Na slici 3.31. vidi se kako se vozilo kreće uz desni parking te traži slobodna parkirna mjesta.



Sl. 3.31. *Primjer iz simulatora - vozilo traži slobodni parking*

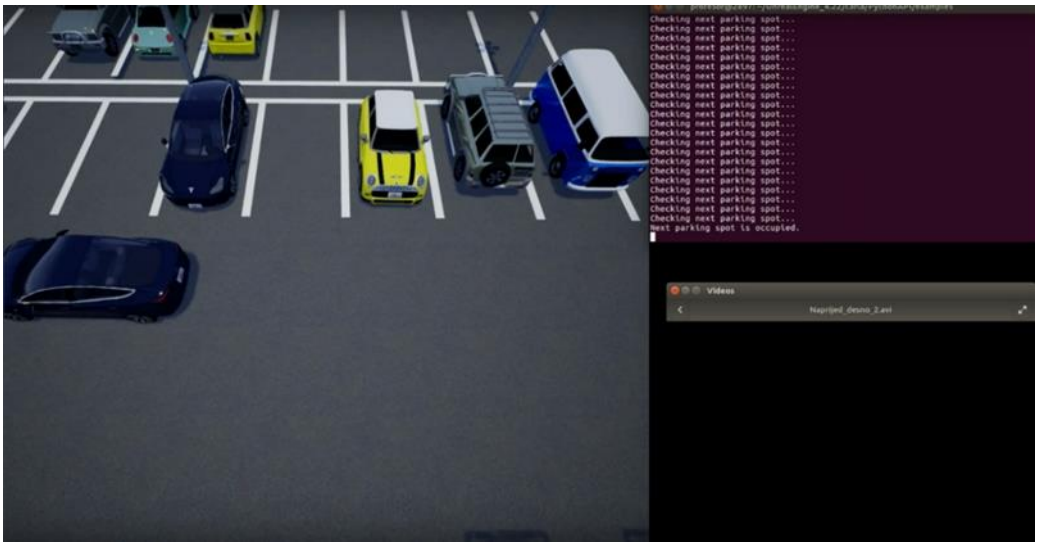
Kada se detektira slobodno parkirno mjesto (zelena površina u donjem desnom rubu slike 3.31.) pamti se lokacija vozila na mapi u tom trenutku. To je lokacija do koje će se vozilo pomaknuti (uz dodane korekcije položaja) u jednom trenutku kada se bude parkiralo (detaljnije ispod). Osim toga

provjerava se i susjedno parkirno mjesto te se ovisno o njegovoj zauzetosti odlučuje koji manevar od dva moguća će se koristiti za parkiranje unaprijed, na slici 3.32. se ono nalazi nakon (lijevo od) detektiranog parkinga i može se vidjeti kako je klasificirano kao zauzeto u donjem desnom uglu slike.



Sl. 3.32. *Primjer iz simulatora - provjera zauzetosti susjednog parkirnog mjesta*

Kada se susjedno parkirno mjesto klasificira, vozilo se zaustavlja i senzor se isključuje, što se može vidjeti na sljedećoj slici 3.33. (pravokutnik gdje su bila označena parkirna mjesta je sada crn) Naredba za zaustavljanje vozila je *vehicle.apply_control (carla.VehicleControl(throttle=0.0, brake =0.1, steer=0.0, reverse = False))*, dok je naredba za isključivanje senzora *sensor.destroy()*.



Sl. 3.33. *Primjer iz simulatora - vozilo se zaustavlja i isključuje se senzor za detekciju i klasifikaciju parkirnog mjesta*

Kako bi se vozilo parkiralo između dva susjedna vozila unaprijed na prethodno detektirani parking, potrebno ga je vratiti unatrag i pri tome pomaknuti u lijevu stranu ceste kako bi imalo dovoljan kut za izvođenje manevra. Na slici 3.34. se nalazi skup naredbi pomoću kojih se vozilo prestrojava u lijevu traku i pomiče unatrag. Sve naredbe za izvođenje manevara su dobivene empirijski manevriranjem vozila u samom CARLA simulatoru.

```
vehicle.apply_control(carla.VehicleControl(throttle=0.0,brake =0.9, steer=0, reverse = False))
time.sleep(1)
vehicle.apply_control(carla.VehicleControl(throttle=0.5,brake =0.0, steer=-0.4, reverse = True))
time.sleep(2.7)
vehicle.apply_control(carla.VehicleControl(throttle=0.5,brake =0.0, steer=0.4, reverse = True))
time.sleep(1.4)
```

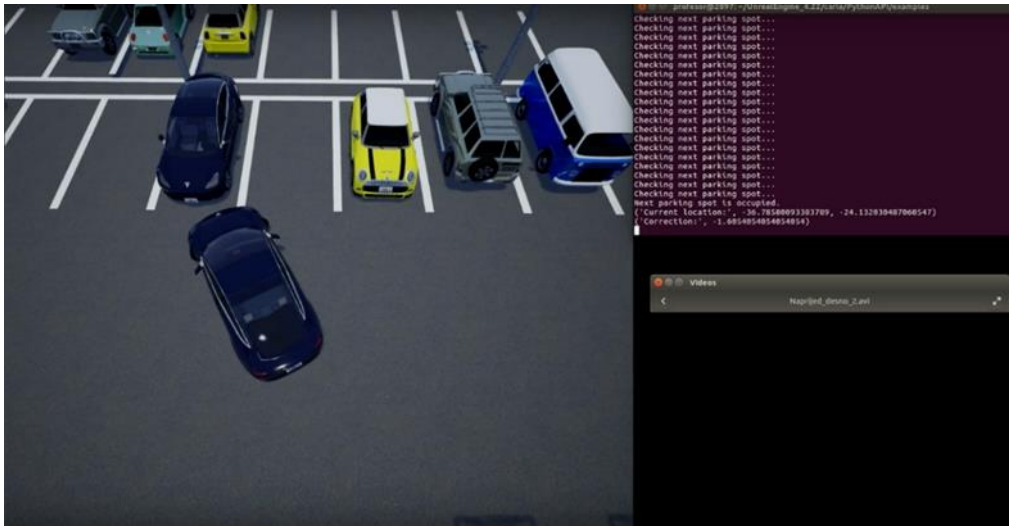
Sl. 3.34. *Naredbe za prestrojavanje vozila i pomak unatrag*

Kada se vozilo prestroji u susjednu traku i zaustavi, ponovno se počinje kretati prema naprijed, na lokaciju na kojoj će skrenuti u parking. Kada vozilo dođe na tu lokaciju, poziva se niz funkcija za manevriranje vozila kako bi se ono uspješno uparkiralo. Na slici 3.35. se mogu vidjeti naredbe za parkiranje vozila, dok se na Slikama 3.36. i 3.37. i 3.38. vidi proces parkiranja vozila

```
vehicle.apply_control(carla.VehicleControl(throttle=0.5,brake =0.0, steer=0.8, reverse = False))
time.sleep(1.6)
vehicle.apply_control(carla.VehicleControl(throttle=0.0,brake =1, steer=0.8, reverse = False))
time.sleep(2)
vehicle.apply_control(carla.VehicleControl(throttle=0.5,brake =0.0, steer=0.0, reverse = True))
time.sleep(0.7)
vehicle.apply_control(carla.VehicleControl(throttle=0.5,brake =0.0, steer=-0.6, reverse = True))
time.sleep(1.02)
vehicle.apply_control(carla.VehicleControl(throttle=0.4,brake =0.0, steer=0.0, reverse = False))
time.sleep(2.6)
vehicle.apply_control(carla.VehicleControl(throttle=0.5,brake =0.0, steer=0.3, reverse = False))
time.sleep(0.7)
vehicle.apply_control(carla.VehicleControl(throttle=0.5,brake =0.0, steer=0.0, reverse = False))
time.sleep(0.3)
vehicle.apply_control(carla.VehicleControl(throttle=0.0,brake =0.9, steer=0.0, reverse = False))
```

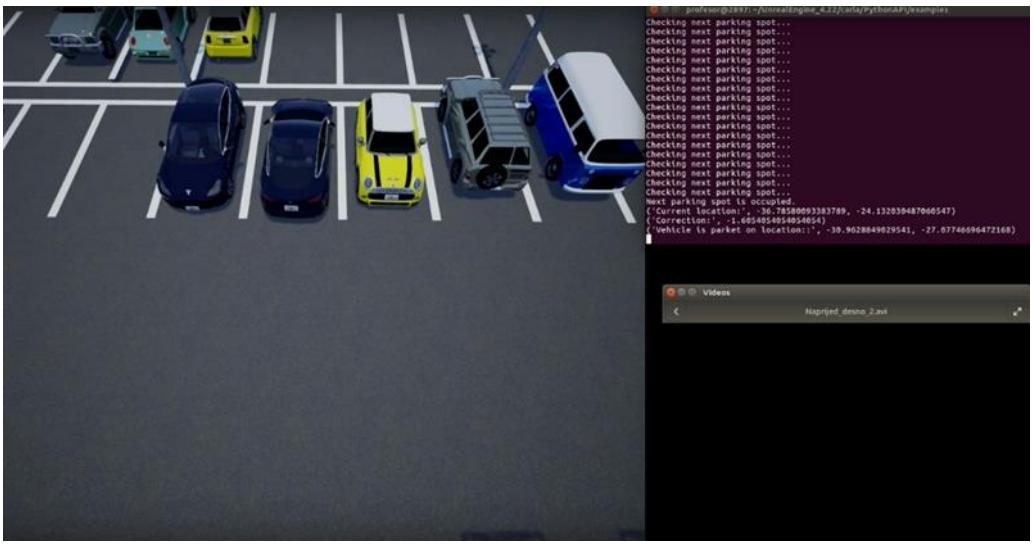
Sl. 3.35. *Naredbe za manevar parkiranja između dva susjedna vozila*

Nakon što vozilo dođe do prethodno spremljene lokacije detekcije parkinga, ono počinje skretati, no s obzirom na to da se na susjedom parkingu nalazi drugo vozilo ne može se uparkirati iz jednog manevra. Zato se vozilo nakon prvog skretanja (Sl. 3.36) vraća unatrag (Sl. 3.37.) kako bi se dobio dovoljan kut da vozilo može skrenuti u parkirno mjesto bez kolizije sa susjednim vozilima. Na slici 3.38 se potom vidi kako je manevar uspješno izvršen.



Sl. 3.36. *Primjer iz simulatora - vozilo skreće prema parkingu*

Sl. 3.37. *Primjer iz simulatora - vozilo se vraća unatrag kako se ne bi sudarilo sa susjednim vozilom*



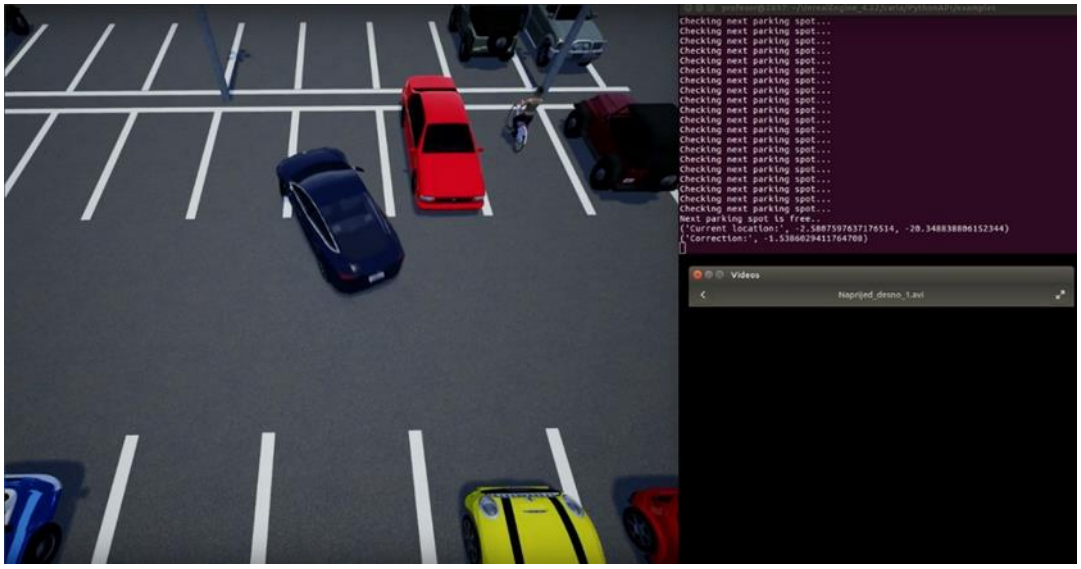
Sl. 3.38. *Primjer iz simulatora - vozilo se uspješno uparkiralo*

Dio cjelokupnog postupka automatskog parkiranja je za sve slučajeve do zaustavljanja vozila nakon detekcije i klasifikacije parkirnog mjesta jednak. Zato će taj dio biti izostavljen u narednim slikama te će se prikazati samo manevar parkiranja unaprijed kada nema parkiranog automobila nakon (lijevo od) detektiranog parkirnog mjesta i manevar parkiranja unatrag. Kompletan programski kod kao i naredbe parkiranja vozila mogu se vidjeti u skripti *Main.py* koja se nalazi u prilogu P.3.1.

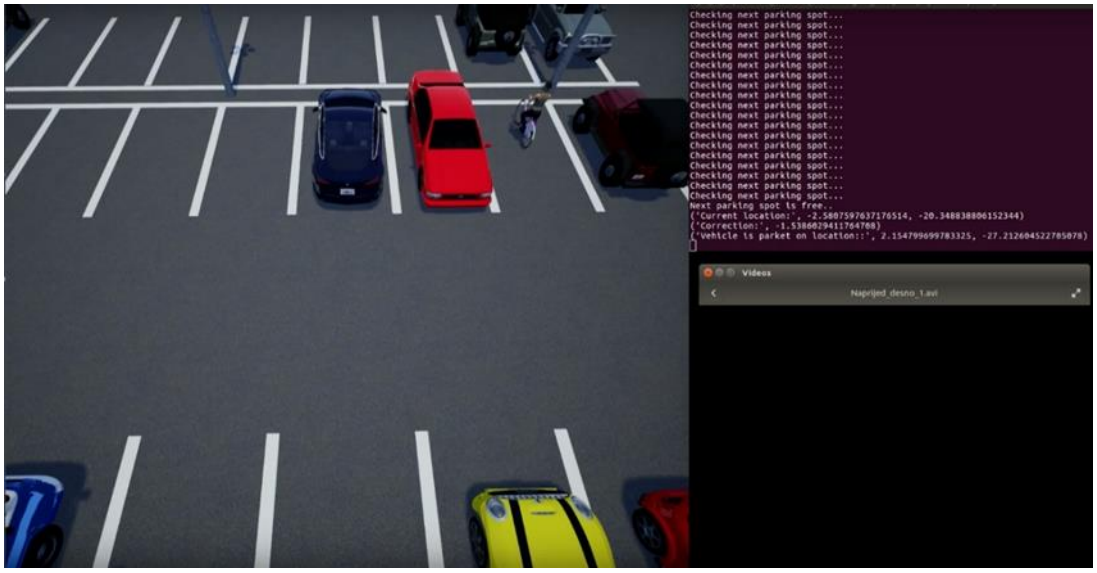
Prilikom parkiranja unaprijed kada je susjedno parkirno mjesto slobodno, izvršava se jednostavniji manevar parkiranja. Na slici 3.39. se vidi kako je susjedni parking klasificiran kao slobodan.

Sl. 3.39. *Primjer iz simulatora - vozilo traži slobodan parking i provjerava njemu susjedni parking*

Nakon što je susjedni parking klasificiran, vozilo se kao i u prethodnom slučaju vreća unatrag te se pri tome pomiče u lijevu stranu ceste kako bi imalo dovoljan kut za manevriranje. Kada se pomakne unatrag, vozilo se kreće prema naprijed do lokacije (uz dodane korekcije položaja) na kojoj je bio detektiran parking. Kada dođe do nje pokreće parkiranje unaprijed iz jednog manevra (Sl.3.40). Za razliku od prethodnog slučaja, ovdje se vozilo ne vreća unatrag nakon što skrene, nego mu je dozvoljeno gaziti susjedno parkirno mjesto pošto je ono klasificirano kao slobodno. Samim time se uspijeva parkirati iz jednog manevra. Na slici 3.40. se vidi vozilo prilikom skretanja na parkirno mjesto a na slici 3.41. se vidi da je uspješno parkirano nakon izvođenja manevra.

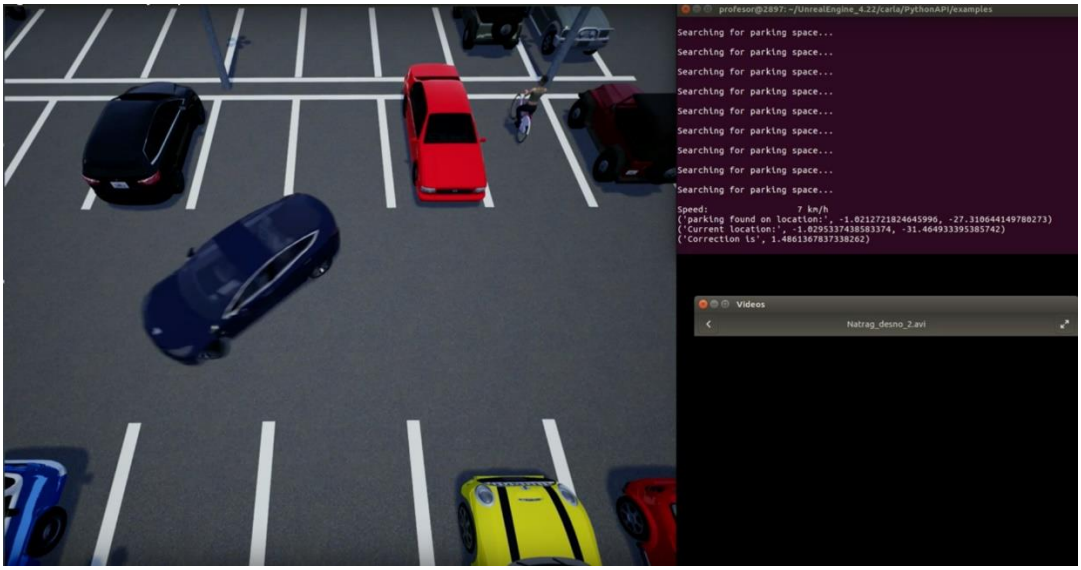


Sl. 3.40. *Primjer iz simulatora - Parkiranje vozila unaprijed iz jednog manevra*



Sl. 3.41. *Primjer iz simulatora - vozilo se uspješno uparkiralo iz jednog manevra*

Za parkiranje unatrag je osmišljen samo jedan manevar iz razloga što se vozilo može uspješno parkirati na slobodni parking neovisno o postojanju vozila na susjednim parkirnim mjestima. Nakon što se detektira slobodno parkirno mjesto, vozilo se pomakne za približno dvije širine parkinga od mjesta detekcije unaprijed. Ta udaljenost iznosi 4.3 metra kako bi vozilo imalo dovoljan kut za manevar. Širina parkinga je izmjerena u CARLA simulatoru i iznosi 2.2 metra. Nakon tog pomaka vozilo se zaustavlja i pokreće manevar parkiranja koji se može vidjeti na slici 3.42. Sve naredbe manevriranja se mogu vidjeti u prilogu P.3.1. u skripti *Main.py*



Sl. 3.42. *Vozilo izvodi manevar parkiranja unatrag*

Na slici 3.42. se također vidi da nema opasnosti od kolizije s vozilom koje može biti parkirano na susjednom parkirnom mjestu jer vozilo u dovoljnom luku obilazi to parkirno mjesto.

Za parkiranje ulijevo vrijedi sve što je do sada navedeno, uz iznimku da se za parkiranje unaprijed ne prestrojava u lijevu stranu ceste nego se vozilo samo vrati unatrag jer već ima dovoljan luk za izvođenje manevra. Za parkiranje unatrag se vozilo prestroji u lijevu stranu ceste kako ne bi udarilo automobile parkirane na desnoj strani. Programski kod u kojemu se nalaze svi manevri za lijevu i desnu stranu kao i kod za detekciju i klasifikaciju parkinga nalazi se u prilogu P.3.1. Nadalje, sve video sekvence koje demonstriraju način rada algoritma se nalaze u prilogu P.3.2.

3.3. Način pokretanja vlastitog rješenja za pomoć pri parkiranju vozila

Kako bi se pokrenulo vlastito rješenje za parkiranje vozila potrebno je prije svega instalirati CARLA simulator na Linux 16.04 LTS operacijski sustav te instalirati potrebne biblioteke za ispravan rad programa. Linkovi s uputama za instalaciju svih potrebnih biblioteka i CARLE mogu se pronaći podpoglavlju 3.1. Nakon što se sve uspješno instalira potrebno je pokrenuti CARLA simulator. Kako bi se pokrenuo simulator potrebno se pozicionirati u datoteku gdje se nalazi skripta za pokretanje CARLA simulatora. U Linux terminal je potrebno upisati:

cd UnrealEngine_4.22/carla/Dist/CARLA-Shipping_0.9.6-4-ga91f0da-dir/LinuxNoEditor/

i potom pokrenuti skriptu pomoću naredbe

./CarlaUE4.sh /Game/Carla/Maps/Town03 -windowed -ResX=320 -ResY=240 -benchmark -fps=10

Nakon te naredbe će se pokrenuti simulator i pri tome će se učitati neka zadana mapa na kojoj nema parkinga. Kako bi se stvorila mapa na kojoj postoje parkirna mjesta, potrebno je otvoriti novi Linux terminal i pozicionirati se pomoću naredbe

cd UnrealEngine_4.22/carla/PythonApi/util.

Tu se pokreće nova skripta pomoću koje se mijenja mapa. Naredba za pokretanje skripte je

python config.py -m Town05.

Nakon što se promijeni mapa potrebno se pozicionirati pomoću naredbe

cd UnrealEngine_4.22/carla/PythonAPI/examples/

i u prethodno otvorenu mapu pod nazivom *examples* kopirati sve datoteke iz priloga P.3.1. U toj mapi se nakon kopiranja nalazi *Main.py* skripta kao i sve ostale datoteke potrebne za rad te skripte. Za pokretanje skripte dovoljno je u terminal upisati

python Main.py -strana -manevar

i program automatskog parkiranja će se početi izvoditi. Na mjesto *-strana* se upisuje „*-strana=L*“ ukoliko se žele pretraživati parkinzi s lijeve strane „*-strana=D*“ za desnu stranu, dok se umjesto *-manevar* unosi „*-manevar=Nap*“ za parkiranje unaprijed i „*-manevar=Naz*“ za parkiranje unatrag. U *Main.py* skripti je već predefinirano gdje će se vozilo stvoriti na mapi (vozilo će biti stvoreno na postojećem parkingu)

spawn_point=carla.Transform(carla.Location(-1,-16,5),carla.Rotation(0,-90,0)).

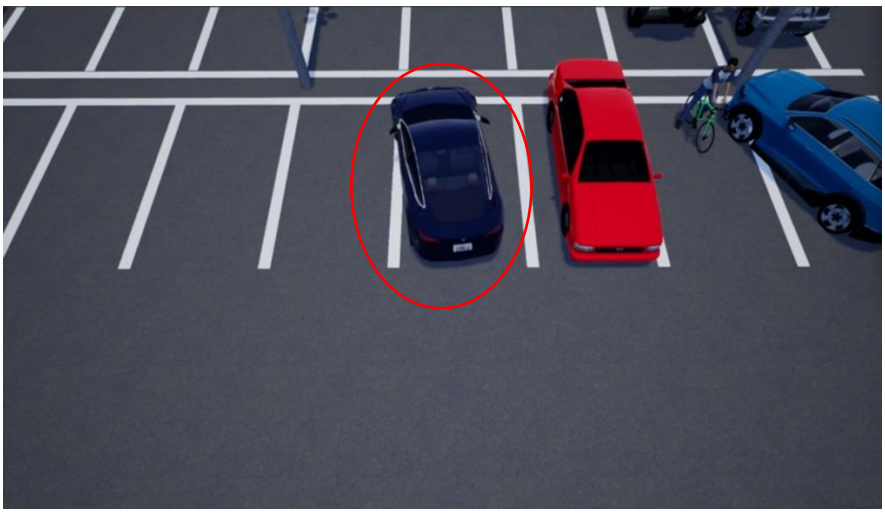
Promjenu lokacije stvaranja vozila moguće je izvesti promjenom parametara u danoj funkciji. Nakon što se automobil stvori na mapi on se automatski počinje kretati po mapi i tražiti parkirno mjesto. Po zadanim postavkama tražiti će se parkirno mjesto te će se izvesti željeni manevar.

4. EVALUACIJA PERFORMANSI PREDLOŽENOG ALGORITMA ZA AUTONOMNO PARKIRANJE

U ovom poglavlju opisan je proces verifikacije ispravnosti rada izrađenog rješenja za autonomno parkiranje i načinjena je evaluacija njegova rada za različite testne scenarije unutar CARLA simulatora. Opisani su rezultati testiranja za detekciju parkinga, njegovu klasifikaciju i parkiranje vozila. Također su opisani rezultati testiranja za različite scenarije kreirane u simulatoru, gdje je vozilo moralo ispravno reagirati na dane uvjete te izvršiti ispravan manevar parkiranja.

Algoritam za pomoć pri parkiranju testiran je na računalu s operacijskim sustavom Ubuntu 16.04 LTS, Grafičkom karticom Nvidia GTX 1060 6Gb, 16 GB RAM-a i procesorom Intel Core i7 8700 3.20 Ghz.

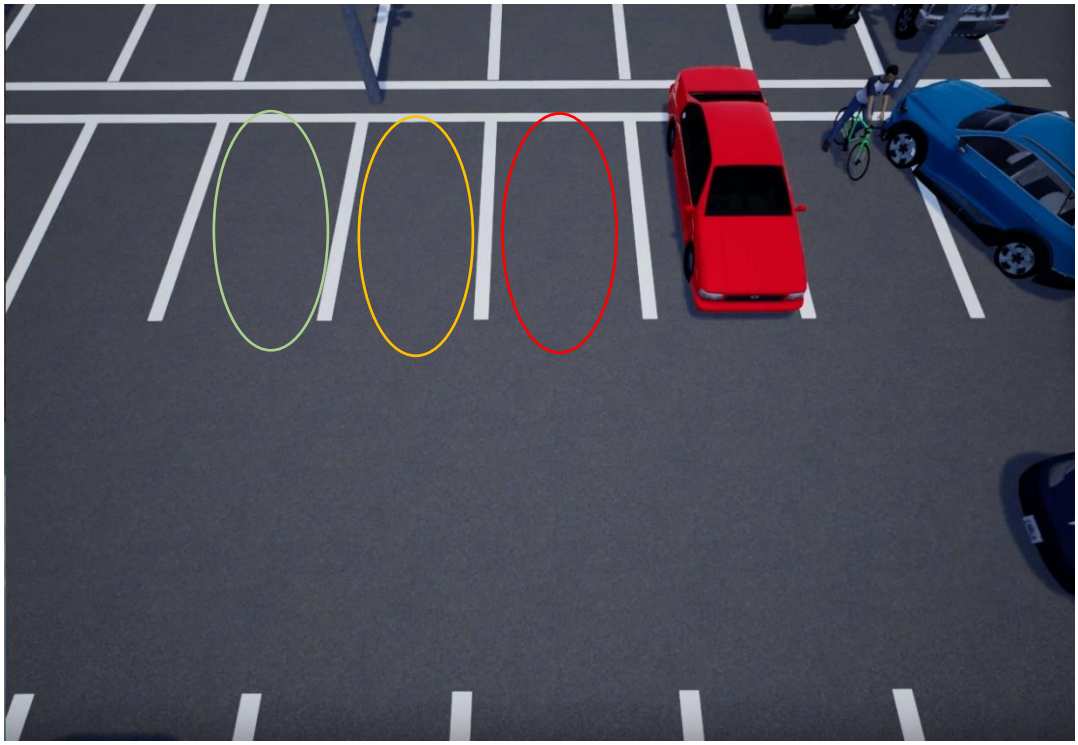
Stopa uspješnosti rada rješenja procjenjuje se na način da se računaju određeni statistički pokazatelji uspješnosti parkiranja. Kako bi se vozilo uspješno parkiralo, potrebno je prethodno imati uspješnu detekciju parkinga, njegovu klasifikaciju i uspješno izvesti sam manevar parkiranja. Ako algoritam zakaže u jednom od navedenih koraka, parkiranje će biti neuspješno. Parkiranje se smatra uspješnim kada se vozilo uparkira između linija detektiranog slobodnog parkirnog mjesta i pri tome ne gazi iste. Svi drugi slučajevi smatraju se neuspješnim. Primjer uspješnog parkiranja vidi se na slici 3.41. (crni automobil koji se nalazi lijevo od crvenog), dok se primjer neuspješnog parkiranja može vidjeti na slici 4.1., gdje prikazano vozilo gazi lijevu oznaku parkirnog mjesta na koje se parkiralo.



Sl. 4.1. *Primjer neuspješno parkiranog vozila*

4.1. Opis skupa testnih scenarija u simulatoru i rezultati testiranja algoritma autonomnog parkiranja

U simulatoru su kreirana tri različita scenarija, gdje se promatraju prva tri parkirna mjesta, počevši ih brojati od prvog slobodnog koje dolazi kad se vozilo kreće (crvena elipsa na slici 4.2) Prvi scenarij se naziva *Slobodno Slobodno Slobodno* (SSS), drugi je *Slobodno Zauzeto Slobodno* (SZS) i treći *Slobodno Slobodno Zauzeto* (SSZ), a nazivi su složeni prema zauzetosti triju parkirnih mjesta označenih na slici gledajući s desna na lijevo (crvena elipsa, žuta elipsa, zelena elipsa na slici 4.2). Za svaki od triju scenarija zauzetosti parkirnih mjesta (SSS, SZS, SSZ) kreirane su sekvence u kojima se vozilo kretalo trima različitim brzinama, tj. 5km/h, 7.5km/h i 10km/h. Sve je to ponovljeno za parkiranje vozila za dvije strane parkinga (lijevu i desnu). Na slici 4.2. prikazana su i označena crvenom, žutom i zelenom elipsom parkirna mjesta koja se promatraju prilikom testiranja rada programskog rješenja za desnu stranu parkinga. Ta tri parkirna mjesta se promatraju jer vozilo do njih uspije postići u simulatoru željenu brzinu kretanja (ranije ne uspije).



Sl. 4.2. *Promatrana parkirna mjesta za parkiranje s desne strane ceste*

U prvom scenariju su sva tri parkirna mjesta slobodna (SSS), u drugom je parkirno mjesto označeno crvenom bojom slobodno, parkirno mjesto označeno žutom bojom je zauzeto, a parkirno mjesto označeno zelenom bojom je također slobodno (SZS), dok je u trećem scenariju parkirno

mjesto označeno zelenom bojom zauzeto dok su mjesta označena crvenom i žutom bojom slobodna (SSZ). Za testiranje parkiranja unaprijed koriste se sva tri scenarija, dok se za testiranje unatrag koriste SSZ i SZS scenariji jer se vozilo parkira na isti način neovisno o zauzeću susjednih parkirnih mjesta. Svi scenariji su kreirani za iste vremenske uvjete, a to je sunčano vrijeme uz dobro osvjetljenje. Za lijevu stranu se promatraju parkirna mjesta koja se nalaze nasuprot označenih parkirnih mjesta s desne strane. Važno je za napomenuti da su promatrana pakirna mjesta u svim testnim slučajevima uspješno detektirana i klasificirana, tj. da je predloženo rješenje te zadatke (detekcija i klasifikacija parkirnog mjesta) obavljalo sa 100% točnosti.

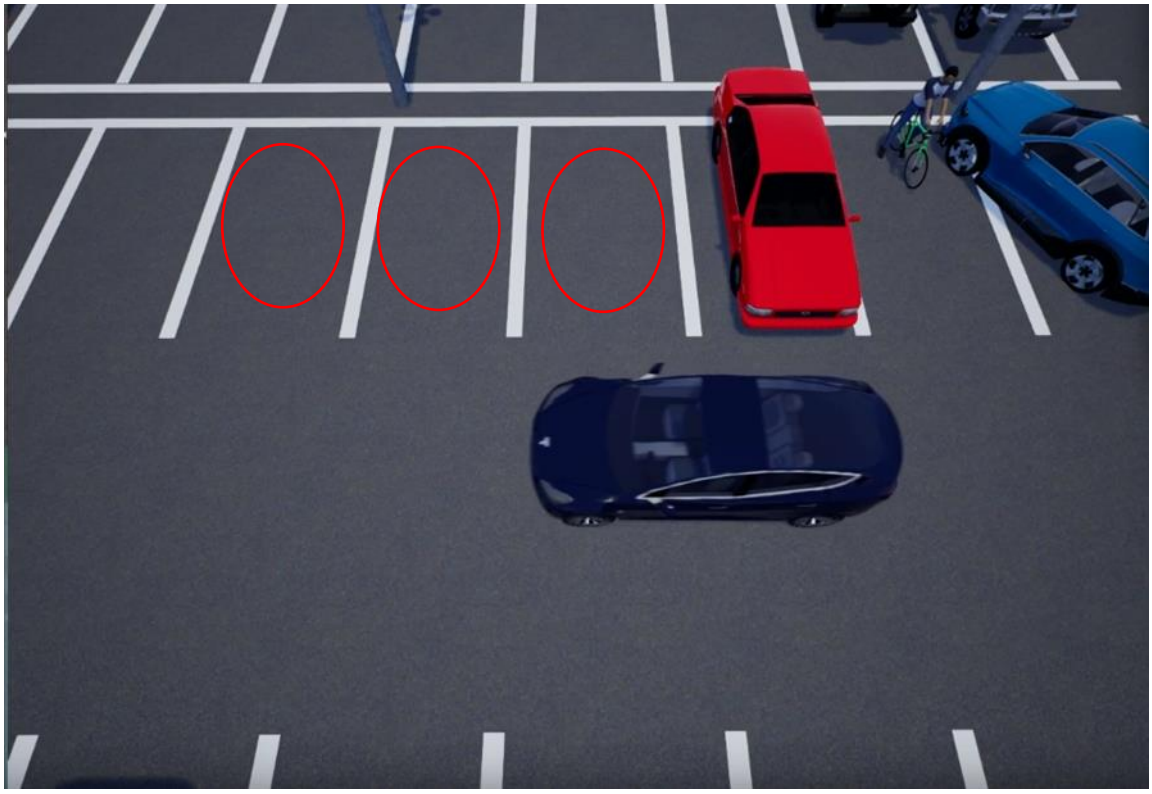
Prilikom postavljanja brzine vozila unutar simulatora, korištene su gotove funkcije u kojima se umjesto definiranja brzine određuje pritisak papučice gasa. Tako za testni model vozila pod nazivom „Tesla Model 3“ pritisak papučice gasa od 0.27 odgovara brzini 5 km/h, 0.3 brzini od 7.5 km/h i 0.36 brzini od 10 km/h. Video sekvence s testnim scenarijima i rezultatima rada algoritma mogu se pogledati u elektroničkom prilogu P.4.1. danom na DVD-u priloženom uz ovaj rad. Predložak za imenovanje video sekvenci je sljedeći:
„Smjer_Strana_Scnearij_Brzina_br.Testa.mp4“

- *Smjer* poprima naziv „*Nap“* ukoliko se parkira unaprijed ili „*Naz“* za parkiranje unatrag
- *Strana* se označava sa slovom *„D“* za parkiranje udesno ili sa slovom *„L“* za parkiranje ulijevo
- *Scenarij* se označava s jednim od prethodno navedenih, tj. SSS, SSZ ili SZS
- *Brzina* može biti 5kmh, 7kmh ili 10kmh
- *Br.Testa* poprima vrijednost od 1 do 10

Konkretan primjer bi izgledao: *„Nap_D_SSS_5kmh_8.mp4“*. Dakle ovdje se radi o testnom slučaju gdje se vozilo parkira unaprijed udesno, kada su sva tri parkirna mjesta slobodna, a radi se o osmom testu za brzinu kretanja od 5km/h. Za brzine od 7.5km/h koristi se naziv 7km/h radi jednostavnosti imenovanja sekvenci.

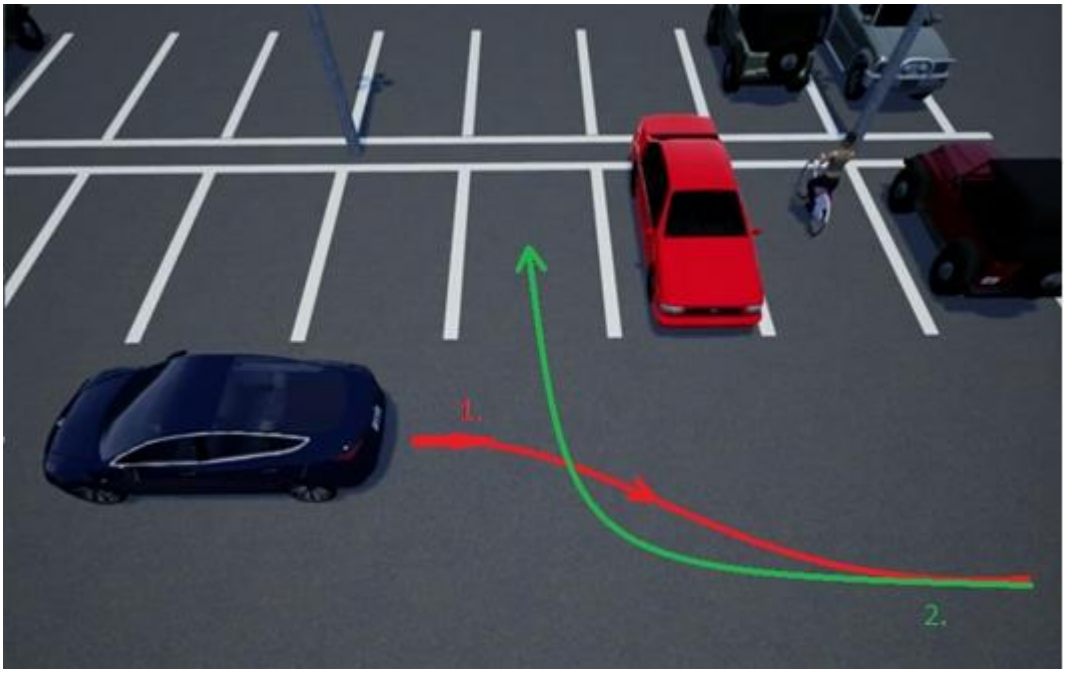
4.1.1. Scenarij Naprijed_Desno_SSS

Vozilo kreće iz stanja mirovanja i ubrzava dok ne postigne zadanu brzinu kretanja po parkiralištu, kojom se nakon toga kreće jednoliko. Smjer kretanja vozila je okomit na položaj parkirnih mjesta kao što se vidi na slici 4.3.



Sl. 4.3. *Položaj vozila u odnosu na položaj parkirnih mjesta*

Sva tri promatrana parkirna mjesta (označena crvenim elipsama na slici 4.3) su u ovom scenariju slobodna. Testno vozilo detektira parkirna mjesta od trenutka kada se počelo kretati sve do trenutka pozitivne detekcije slobodnog parkirnog mjesta, kao što je prethodno opisano u dijelu 3.2.2, nakon čega se senzor (RGB kamera) isključuje i započinje manevar parkiranja. U prvom testnom scenariju se nakon detekcije prvog slobodnog parkirnog mjesta provjerava zauzetost susjednog parkirnog mjesta (srednjeg od tri označena na slici 4.3.). Ukoliko je i ono slobodno, vozilo se zaustavlja te se vraća unatrag i pri tome se udaljava od parkirnog mjesta (prestrojava se u lijevu voznu traku) kako bi imalo dovoljan kut za parkiranje (detaljnije opisano u dijelu 3.2.3., a prikazano je na slici 4.4). Nakon što se vozilo prestroji, kreće se jednolikom brzinom od 4 km/h prema parkirnom mjestu gdje se, na prethodno zabilježenoj lokaciji, izvodi parkiranje udesno iz jednog manevra (detaljnije pojašnjeno u dijelu 3.2.3). Testiranje za ovaj scenarij za parkiranje unaprijed je izvedeno deset puta za svaku od prethodno navedenih brzina. U tablici 4.1. dani su rezultati provedenih testova za ovaj scenarij za sve tri postavljene brzine kretanja vozila.



Sl. 4.4. *Putanja vozila prilikom parkiranja unaprijed iz jednog manevra*

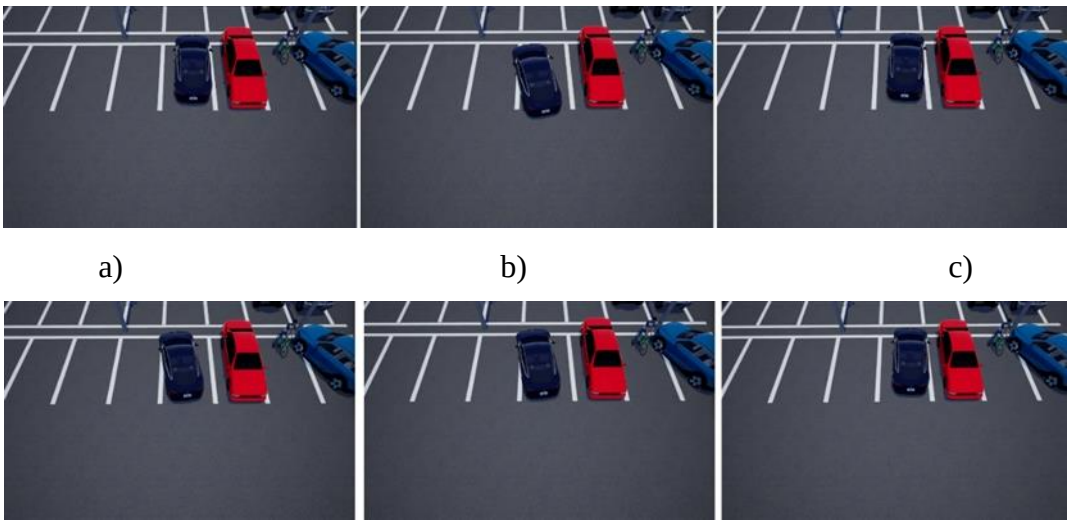
Tablica 4.1. *Rezultati testiranja algoritma autonomnog parkiranja za scenarij Naprijed_Desno_SSS za brzine kretanja vozila 5km/h, 7.5km/h i 10km/h*

Brzina kretanja vozila u simulatoru	**Broj uspješnih parkiranja**	**Broj neuspješnih parkiranja**	**Naziv video sekvence neuspješnog parkiranja (P.4.1.)**	**Postotak uspješnosti parkiranja**
5km/h	9	1	Nap_D_SSS_5kmh_8.mp4	90.00%
7.5km/h	6	4	Nap_D_SSS_7kmh_2.mp4 Nap_D_SSS_7kmh_5.mp4 Nap_D_SSS_7kmh_6.mp4 Nap_D_SSS_7kmh_9.mp4	60.00%
10km/h	9	1	Nap_D_SSS_10kmh_3.mp4	90.00%
Sve brzine zajedno	24	6	-	80.00%

Iz dobivenih rezultata se vidi da se testno vozilo parkira nešto bolje za brzine kretanja od 5 km/h i 10 km/h. Za brzinu od 7.5 km/h testno vozilo ima najmanji postotak uspješnih parkiranja. Analizom provedenih testiranja utvrđeno je da je razlog taj što se prilikom procjene korekcije položaja zbog brzine kretanja vozila objašnjene u dijelu 3.2.3. unijela pogreška za brzinu od 7.5km/h. Problem prilikom parkiranja vozila je taj je što varijacije od par centimetara značajno utječu na ishod parkiranja. Također, korekcija položaja nije jednaka za sve manevre, tako da se

unose dodatne varijacije za svaki od njih. Korekcije položaja za neke od manevara su bolje podešene nego za druge. Uz to sve, vozilo varira prilikom parkiranja zbog nekih drugih kašnjenja koji se javljaju u sustavu. Primijećeno je da se za identične uvjete (ista brzina kretanja, ista okolina vozila) vozilo svaki puta drugačije parkira. Razlog tomu je manjak računalnih resursa, jer se na istom računalu pokreću CARLA server, nekoliko CARLA klijenata, program za snimanje pozadine (Kazam) i sam algoritam autonomnog parkiranja, što zaokupljuje veliki dio radne memorije i procesorskih resursa. Zbog toga vozilo ne detektira parkinge u istom trenutku (u odnosu na trenutak u kojem je započeo kretanje) za iste testne slučajeve, niti se *sleep()* funkcije koje se koriste za izvršavanje manevra parkiranja odvijaju jednako dugo. Na primjer *sleep(2)* funkcija „uspavljuje“ izvođenje algoritma parkiranja na 2 sekunde, no zbog prevelike opterećenosti računala ona se nekad izvodi 2.1 sekundu a nekad 2.01 sekundu i to unosi pogrešku u manevar parkiranja. Dakle kada bi se parkiralo bez korekcija položaja, vozilo bi se i dalje dobro parkiralo za neke slučajeve, a za neke ne. Uz ovakav način rada varijacije u ishodima parkiranja se dodatno smanjuju te je postotak uspješnih parkiranja nešto veći.

Sveukupno su zabilježena 24 uspješna parkiranja, što je 80% ukupnog broja parkiranja. U ovom scenariju testno vozilo prilikom neuspješnih parkiranja gazi desnu liniju tri puta (Slika 4.5. (a), (c), (f)), lijevu liniju gazi dva puta (Sl. 4.5. (d),(e) te siječe lijevu liniju i ulazi u susjedni parking jednom (Sl. 4.5. (b)).



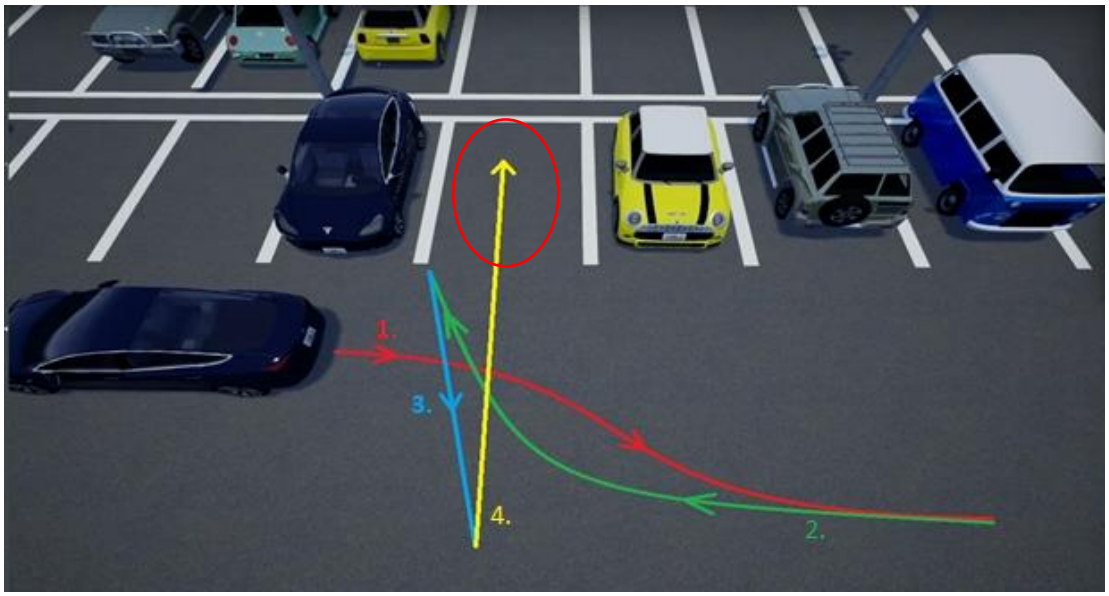
a) b) c)

d) e) f)

Sl. 4.5. *Prikaz neuspješnih pokušaja parkiranja za testni scenarij Naprijed_ Desno_SSS (a) Nap_D_SSS_5kmh_8.mp4, (b) Nap_D_SSS_7kmh_2.mp4, (c) Nap_D_SSS_7kmh_5.mp4, (d) Nap_D_SSS_7kmh_6.mp4 (e) Nap_D_SSS_7kmh_9.mp4 (f) Nap_D_SSS_10kmh_3.mp4*

4.1.2. Scenarij Naprijed_Desno_SZS

U drugom testnom scenariju srednje parkirno mjesto od tri promatrana je zauzeto i testno vozilo se mora parkirati između dva već postojeća vozila na susjednim zauzetim parkirnim mjestima (Slika 4.6.). Nakon što se detektira slobodno parkirno mjesto provjerava se njemu susjedno, ono koje se nalazi u smjeru kretanja vozila. Kada ga se klasificira kao zauzeto, testno vozilo se zaustavlja i počinje se kretati unatrag te se prestrojava u lijevu traku. Nakon toga se vozilo kreće jednolikom brzinom prema lokaciji parkirnog mjesta i izvodi složeniji manevar parkiranja između dva susjedna pomoću više različitih pokreta. Putanja parkiranja vozila je prikazana na slici 4.6. U tablici 4.2. dani su rezultati provedenih testova za ovaj scenarij za sve tri postavljene brzine kretanja vozila.



Sl. 4.6. *Putanja vozila prilikom parkiranja unaprijed između dva vozila*

Tablica 4.2. *Rezultati testiranja algoritma autonomnog parkiranja za scenarij Naprijed_Desno_SZS za brzine kretanja vozila 5km/h, 7.5km/h i 10km/h*

Brzina kretanja vozila u simulatoru	**Broj uspješnih parkiranja**	**Broj neuspješnih parkiranja**	**Naziv video sekvence neuspješnog parkiranja (P.4.1.)**	**Postotak uspješnosti parkiranja**
5km/h	9	1	Nap_D_SZS_5kmh_3.mp4	90.00%
7.5km/h	9	1	Nap_D_SZS_7kmh_6.mp4	90.00%
10km/h	10	0	-	100.00%
Sve brzine zajedno	28	2	-	93.33%

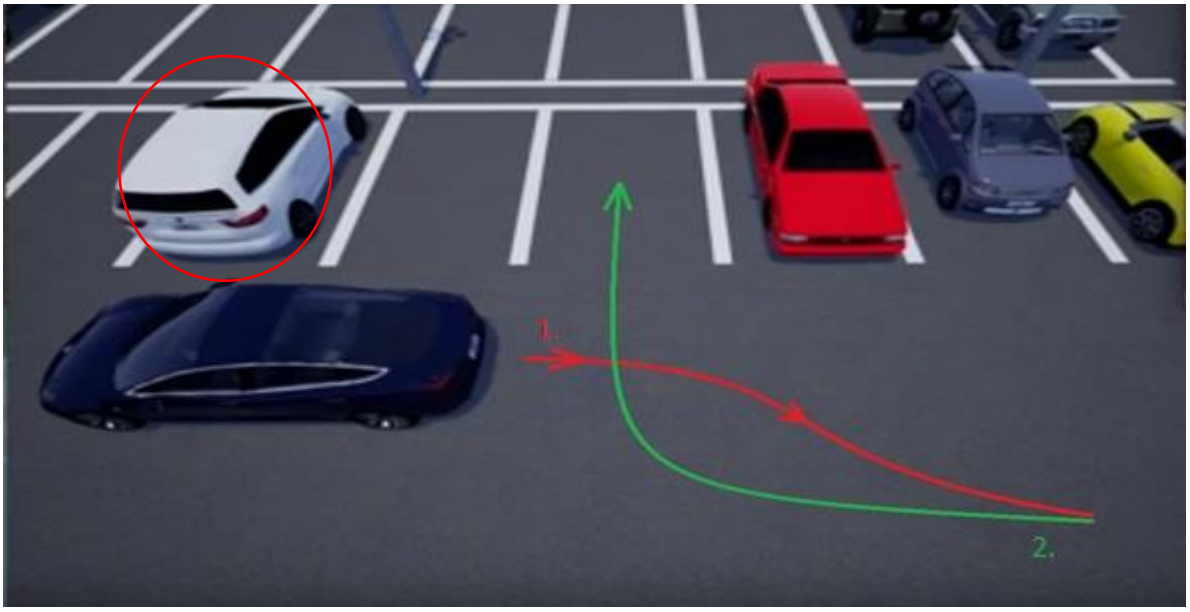
U ovom scenariju testno vozilo je imalo samo dva neuspješna parkiranja. Pritom je u jednom slučaju gazilo lijevu oznaku parkirnog mjesta, a u drugom desnu oznaku parkirnog mjesta (Sl. 4.7.). Razlog manjeg broja neuspješnih parkiranja u odnosu na prethodni scenarij potencijalno leži u tome što se vozilo nakon početnog skretanja prema parkingu vraća unatrag. Samim time si stvara dodatan prostor za izvršavanje manevra i ulazi u parkirno mjesto poprilično ravno, tako da nema potrebe za korekcijama položaja vozila (upravljača).

a) b)

Sl. 4.7. *Prikaz neuspješnih pokušaja parkiranja za testni scenarij Naprijed_Desno_SZS (a) Nap_D_SZS_5kmh_3.mp4, (b) Nap_D_SZS_7kmh_6.mp4*

4.1.3. Scenarij Naprijed_Desno_SSZ

U trećem testnom scenariju treće parkirno mjesto od tri promatrana (ono koje je najdalje od testnog vozila) je zauzeto i testno vozilo se mora parkirati na prvo slobodno parkirno mjesto. Cilj ovog testa je pokazati kako vozilo parkirano na skroz lijevom parkirnom mjestu od tri promatrana (označeno na Sl. 4.8.) ne utječe na odabir manevra za parkiranje vozila. Nakon što se detektira slobodno parkirno mjesto, provjerava se njemu susjedno, ono koje se nalazi u smjeru kretanja vozila. Kada ga se klasificira kao slobodnog testno vozilo se zaustavlja i počinje kretati unatrag te se prestrojava u lijevu traku. Nakon toga vozilo se kreće jednolikom brzinom prema lokaciji parkirnog mjesta i izvodi manevar parkiranja. Skica na kojoj se može vidjeti način parkiranja vozila je na slici 4.8. U tablici 4.3. dani su rezultati provedenih testova za ovaj scenarij za sve tri postavljene brzine kretanja vozila.

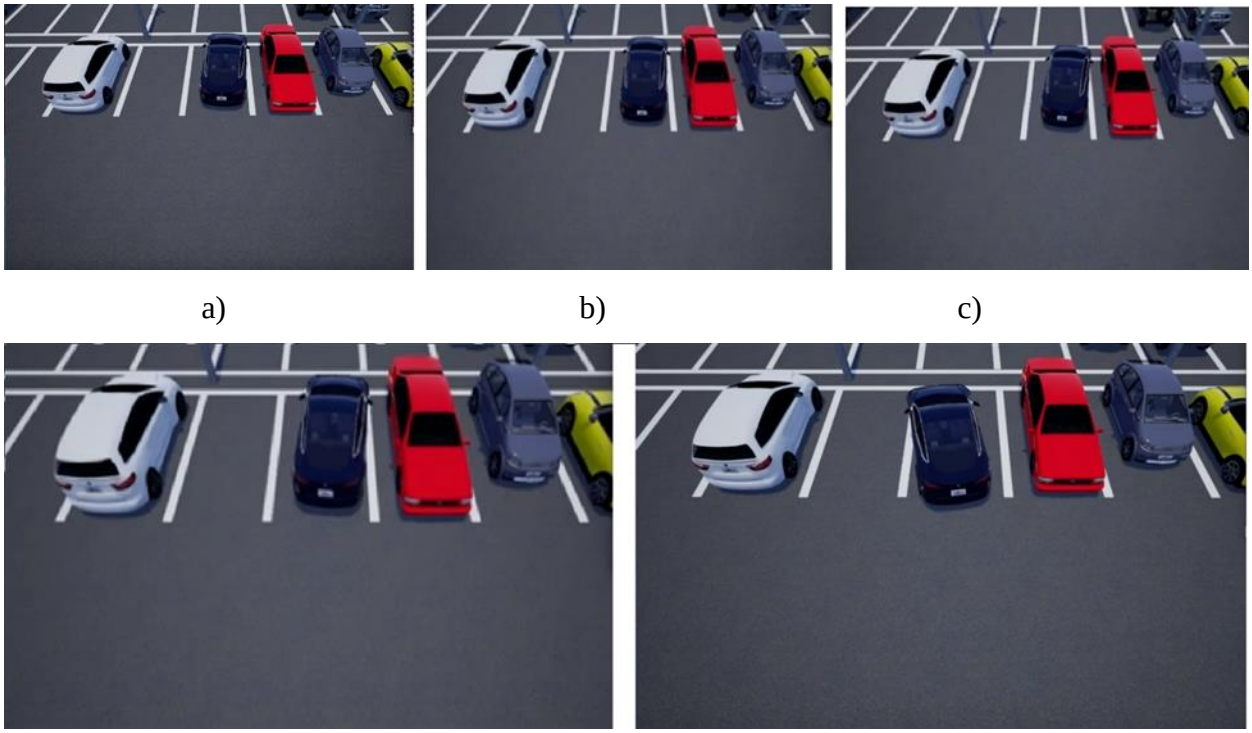


Sl. 4.8. *Putanja vozila prilikom parkiranja unaprijed iz jednog manevra kada postoji još jedno vozilo parkirano na jednom od 3 promatrana parkinga*

Tablica 4.3. *Rezultati testiranja algoritma autonomnog parkiranja za scenarij Naprijed_Desno_SSZ za brzine kretanja vozila 5km/h, 7.5km/h i 10km/h*

Brzina kretanja vozila u simulatoru	**Broj uspješnih parkiranja**	**Broj neuspješnih parkiranja**	**Naziv video sekvence neuspješnog parkiranja (P.4.1.)**	**Postotak uspješnosti parkiranja**
5km/h	8	2	Nap_D_SSZ_5kmh_2.mp4 Nap_D_SZZ_5kmh_5.mp4	80.00%
7.5km/h	9	1	Nap_D_SZZ_7kmh_6.mp4	90.00%
10km/h	8	2	Nap_D_SZZ_10kmh_1.mp4 Nap_D_SZZ_10kmh_3.mp4	80.00%
Sve brzine zajedno	25	5	-	83.33%

Iz rezultata se vidi da je prosječna uspješnost parkiranja 83.33%. Pri tome je bilo ukupno 5 neuspješnih parkiranja od kojih vozilo gazi desnu liniju tri puta (Sl. 4.9. (a) (b) (d)), jednom siječe lijevu liniju i ulazi u susjedni lijevi parking (Sl. 4.9. (e)) i jednom siječe desnu liniju i ulazi u susjedni desni parking (Sl. 4.9. (c)). Iz navedenih slika se može zaključiti da se manevar parkiranja za ovaj scenarij pokreće nešto ranije nego bi trebao i zbog toga vozilo siječe desnu liniju češće nego lijevu.



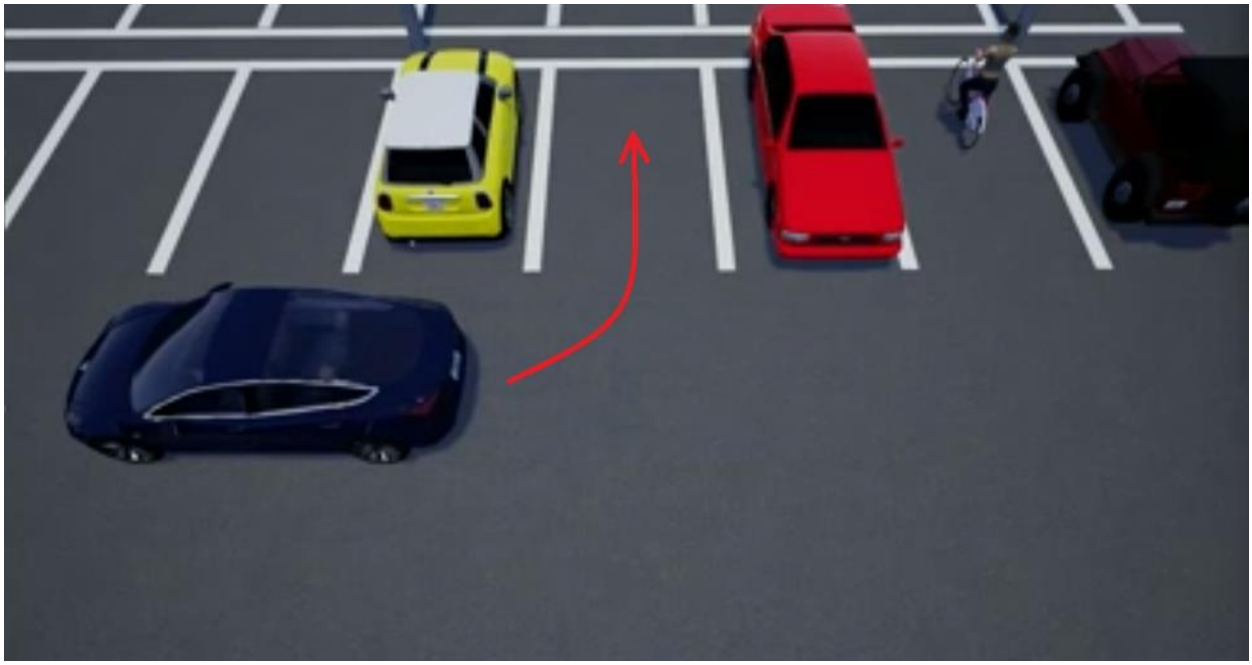
a) b) c)

d) e)

Sl. 4.9. *Prikaz neuspješnih pokušaja parkiranja za testni scenarij Naprijed_Desno_SSZ (a) Nap_D_SSZ_5kmh_2.mp4, (b) Nap_D_SSZ_5kmh_5.mp4, (c) Nap_D_SSZ_7kmh_6.mp4, (d) Nap_D_SSZ_10kmh_1.mp4, (e) Nap_D_SSZ_10kmh_3.mp4*

4.1.4. Scenarij Natrag_Desno_SZS

U ovom testnom scenariju srednje parkirno mjesto od tri promatrana je zauzeto i testno vozilo se mora parkirati između dva već postojeća vozila na susjednim zauzetim parkirnim mjestima (Slika 4.10.). Nakon što je parking detektiran i klasificiran kao slobodan, senzor se isključuje i pokreće se manevar parkiranja unatrag koji je prikazan na slici 4.10. Prilikom parkiranja unatrag ne provjerava se susjedno parkirno mjesto jer vozilo ima dovoljno prostora za izvođenje manevra. U tablici 4.4. dani su rezultati provedenih testova za ovaj scenarij za sve tri postavljene brzine kretanja vozila.



Sl. 4.10. *Putanja vozila prilikom parkiranja desno unatrag iz jednog manevra*

Tablica 4.4. *Rezultati testiranja algoritma autonomnog parkiranja za scenarij Natrag_Desno_SZS za brzine kretanja vozila 5km/h, 7.5km/h i 10km/h*

Brzina kretanja vozila u simulatoru	**Broj uspješnih parkiranja**	**Broj neuspješnih parkiranja**	**Naziv video sekvence neuspješnog parkiranja (P.4.1.)**	**Postotak uspješnosti parkiranja**
5km/h	10	0	-	100.00%
7.5km/h	10	0	-	100.00%
10km/h	10	0	-	100.00%
Sve brzine zajedno	30	0	-	100.00%

U ovom testnom scenariju je uspješnost parkiranja 100% bez ijednog pogrešnog parkiranja. Razlog velike točnosti parkiranja je taj što se parkiranje unatrag izvodi iz jednog manevra te nema dodatnih pozicioniranja vozila nakon detekcije parkinga (za razliku od parkiranja unaprijed gdje se vozilo mora pomaknuti za unatrag i u lijevo kako bi se moglo parkirati u određeno parkirno mjesto)

4.1.5. Scenarij Natrag_Desno_SSZ

U ovom testnom scenariju treće parkirno mjesto od tri promatrana (ono koje je najdalje od testnog vozila) je zauzeto (Slika 4.8.) i testno vozilo se mora parkirati na prvo slobodno parkirno mjesto. Kada se parkirno mjesto detektira i klasificira kao slobodno, testno vozilo usporava do proračunate lokacije gdje se u postupnosti zaustavi i nakon toga izvede manevar parkiranja unatrag na isti način kao što je prikazano na slici 4.10. Jedina je razlika u tome što je susjedno parkirno mjesto slobodno. U tablici 4.5. dani su rezultati provedenih testova za ovaj scenarij za sve tri postavljene brzine kretanja vozila.

Tablica 4.5. *Rezultati testiranja algoritma autonomnog parkiranja za scenarij Natrag_Desno_SSZ za brzine kretanja vozila 5km/h, 7.5km/h i 10km/h*

Brzina kretanja vozila u simulatoru	**Broj uspješnih parkiranja**	**Broj neuspješnih parkiranja**	**Naziv video sekvence neuspješnog parkiranja (P.4.1.)**	**Postotak uspješnosti parkiranja**
5km/h	8	2	Naz_D_SSZ_5kmh_4.mp4, Naz_D_SZZ_5kmh_5.mp4	80.00%
7.5km/h	9	1	Naz_D_SZZ_7kmh_8.mp4	90.00%
10km/h	8	2	Naz_D_SZZ_10kmh_3.mp4, Naz_D_SZZ_10kmh_7.mp4	80.00%
Sve brzine zajedno	25	5	-	83.33%

U ovom testnom slučaju zabilježeno je 25 uspješnih parkiranja i 5 neuspješnih. Od 5 neuspješnih u četiri slučaja vozilo gazi desnu liniju (Sl. 4.11. (a), (b,) (c), (d)), a u jednom slučaju lijevu liniju Sl. 4.11. (e). Zaključak je da se manevar parkiranja pokreće prerano, dakle vozilo ne pređe dovoljan put (prema naprijed) nakon detektiranja parkinga prije nego što započne manevar parkiranja unatrag tj. lokacija na kojoj se počinje skretati u parking loše određena (korekcija položaja vozila nije dobro izračunata za ovaj testni slučaj).

a) b) c)

d) e)

Sl. 4.11. *Prikaz neuspješnih pokušaja parkiranja za testni scenarij Natrag_Desno_SSZ (a) Naz_D_SSZ_5kmh_4.mp4, (b) Naz_D_SZZ_5kmh_5.mp4, (c) Naz_D_SZZ_7kmh_8.mp4, (d) Naz_D_SZZ_10kmh_3.mp4, (e) Naz_D_SZZ_10kmh_7.mp4*

U tablici 4.6. dani su skupni rezultati provedenih testova za parkiranja udesno za tri razmatrane brzine kretanja vozila. Iz tablice se može vidjeti da se vozilo parkira s iznimno visokim postotkom uspješnosti od 88%. Također, iz tablice svih scenarija za parkiranje u desno zajedno, može se vidjeti da vozilo podjednako dobro parkira za svaku od testnih brzina.

Tablica 4.6. *Skupni rezultati testiranja algoritma autonomnog parkiranja udesno*

Brzina kretanja vozila u simulatoru	**Broj uspješnih parkiranja**	**Broj neuspješnih parkiranja**	**Postotak uspješnosti parkiranja**
5km/h	44	6	88.00%
7.5km/h	43	7	86.00%
10km/h	45	5	90.00%
Sve brzine zajedno	132	18	88.00%

Svi do sada navedeni scenariji su identični i za parkiranje u lijevu stranu, tako da će se u nastavku samo prikazati tablice rezultata za svaki od slučajeva i prokomentirati dobiveni rezultati.

4.1.6. Scenarij Naprijed_Lijevo_SSS

U tablici 4.7. dani su rezultati provedenih testova za ovaj scenarij za sve tri postavljene brzine kretanja vozila.

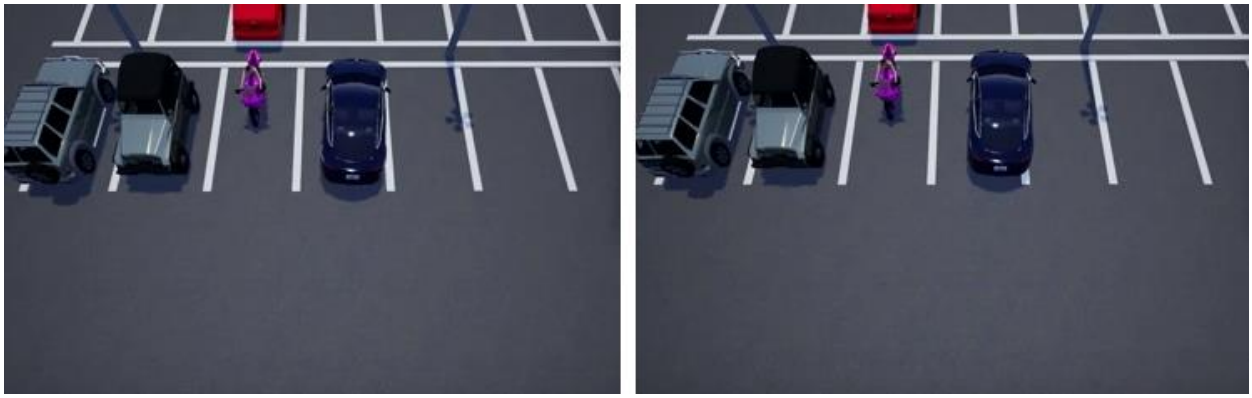
Tab 4.7. *Rezultati testiranja algoritma autonomnog parkiranja za scenarij Naprijed_Lijevo_SSS za brzine kretanja vozila 5km/h, 7.5km/h i 10km/h*

Brzina kretanja vozila u simulatoru	**Broj uspješnih parkiranja**	**Broj neuspješnih parkiranja**	**Naziv video sekvence neuspješnog parkiranja (P.4.1.)**	**Postotak uspješnosti parkiranja**
5km/h	10	0	-	100.00%
7.5km/h	8	2	Nap_L_SSS_7kmh_4.mp4, Nap_L_SSS_7kmh_5.mp4	80.00%
10km/h	8	2	Nap_L_SSS_10kmh_7.mp4, Nap_L_SSS_10kmh_9.mp4	80.00%
Sve brzine zajedno	26	4	-	86.67%

Za ovaj testni slučaj zabilježeno je 26 uspješnih parkiranja i 4 neuspješna. Prilikom neuspješnih parkiranja vozilo je tri puta gazilo desnu liniju i ju je jednom sjeklo i ušlo u susjedni parking (Slika 4.12 (d)). Vozilo za brzine 7.5km/h i 10km/h gazi desnu traku - dakle prekasno je pokrenut manevar parkiranja. Iz navedenog se može zaključiti da je točna lokacija parkirnog mjesta, a samim time i lokacija na kojoj vozilo počinje skretati u parking nešto bolje određena za brzinu od 5km/h.

a)

b)



c) d)

Sl. 4.12. *Prikaz neuspješnih pokušaja parkiranja za testni scenarij Naprijed_Lijevo_SSS (a) Nap_L_SSS_7kmh_4.mp4, (b) Nap_L_SSS_7kmh_5.mp4, (c) Nap_L_SSS_10kmh_7.mp4, (d) Nap_L_SSS_10kmh_9.mp4*

4.1.7. Scenarij Naprijed_Lijevo_SZS

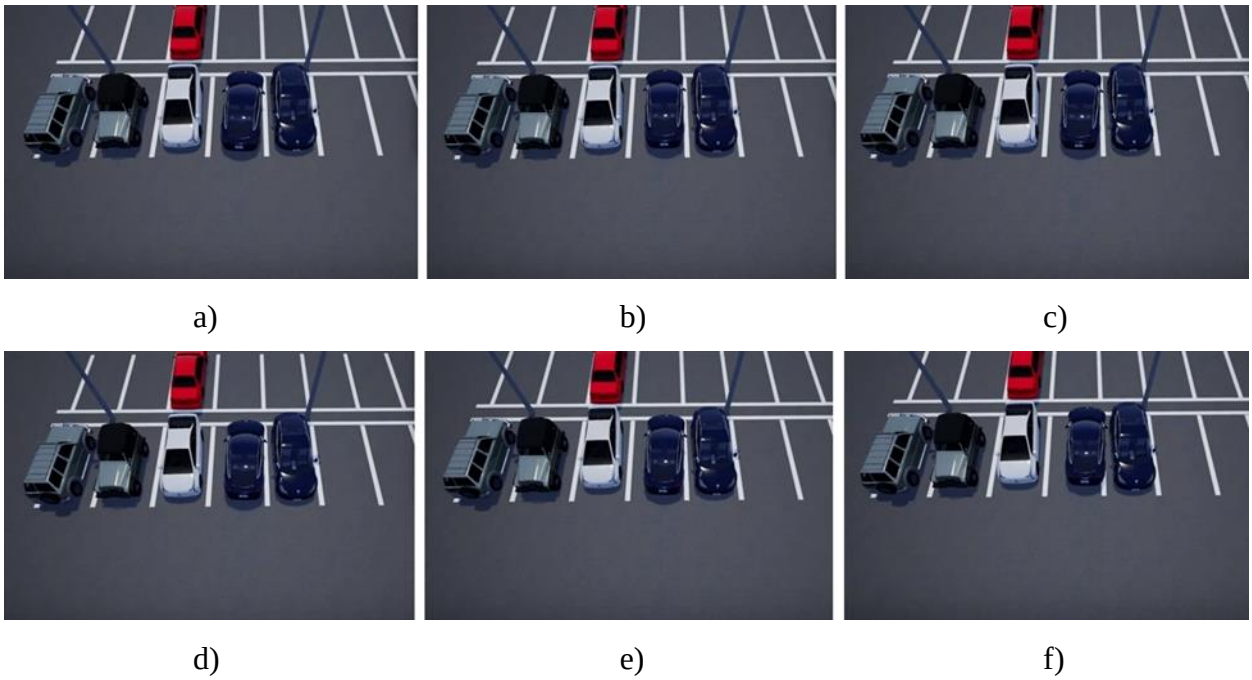
U tablici 4.8. dani su rezultati provedenih testova za ovaj scenarij za sve tri postavljene brzine kretanja vozila.

Tablica 4.8. *Rezultati testiranja algoritma autonomnog parkiranja za scenarij Naprijed_Lijevo_SZS za brzine kretanja vozila 5km/h, 7.5km/h i 10km/h*

Brzina kretanja vozila u simulatoru	**Broj uspješnih parkiranja**	**Broj neuspješnih parkiranja**	**Naziv video sekvence neuspješnog parkiranja (P.4.1.)**	**Postotak uspješnosti parkiranja**
5km/h	10	0	-	100.00%
7.5km/h	7	3	Nap_L_SZS_7kmh_1.mp4, Nap_L_SZS_7kmh_5mp4, Nap_L_SZS_7kmh_7.mp4	70.00%
10km/h	7	3	Nap_L_SZS_10kmh_1.mp4, Nap_L_SZS_10kmh_7.mp4, Nap_L_SZS_10kmh_8.mp4	70.00%
Sve brzine zajedno	24	6	-	80.00%

Prilikom parkiranja naprijed lijevo između dva vozila, vozilo se netočno parkira šest puta, od čega pet puta gazi desnu liniju i jednom siječe liniju susjednog parkinga i ulazi u susjedni desni parking

(Slika 4.13. (f)). Kao i u prethodnom slučaju, točka na kojoj počinje skretanje u parking prilikom parkiranja vozila je nešto bolje podešena za brzinu od 5km/h.



a) b) c)

d) e) f)

Sl. 4.13. *Prikaz neuspješnih pokušaja parkiranja za testni scenarij Naprijed_Lijevo_SZS (a) Nap_L_SZS_7kmh_1.mp4, (b) Nap_L_SZS_7kmh_5.mp4, (c) Nap_L_SZS_7kmh_7.mp4, (d) Nap_L_SZS_10kmh_1.mp4, (e) Nap_L_SZS_10kmh_7.mp4, (f) Nap_L_SZS_10kmh_8.mp4*

4.1.8. Scenarij Naprijed_Lijevo_SSZ

U tablici 4.9. dani su rezultati provedenih testova za ovaj scenarij za sve tri postavljene brzine kretanja vozila.

Tab 4.9. *Rezultati testiranja algoritma autonomnog parkiranja za scenarij Naprijed_Lijevo_SSZ za brzine kretanja vozila 5km/h, 7.5km/h i 10km/h*

Brzina kretanja vozila u simulatoru	**Broj uspješnih parkiranja**	**Broj neuspješnih parkiranja**	**Naziv video sekvence neuspješnog parkiranja (P.4.1.)**	**Postotak uspješnosti parkiranja**
5km/h	9	1	Nap_L_SSZ_5kmh_2.mp4	90.00%
7.5km/h	9	1	Nap_L_SSZ_7kmh_2.mp4	90.00%
10km/h	9	1	Nap_L_SSZ_10kmh_1.mp4	90.00%
Sve brzine zajedno	27	3	-	90.00%

Za parkiranje naprijed lijevo u trećem scenariju vozilo parkira netočno tri puta, od čega jednom gazi lijevu liniju (Slika 4.14. (a)) i dva puta desnu Sl. 4.14.(b), (c)). Potencijalni problem u ovom testnom slučaju su računalni resursi jer je vozilo dva puta prekasno pokrenulo manevar parkiranja, a jednom, iako je na vrijeme pokrenut manevar se predugo izvodio te je vozilo nagazilo lijevu liniju.

a) b) c)

Sl. 4.14. *Prikaz neuspješnih pokušaja parkiranja za testni slučaj Naprijed_Lijevo_SSZ (a) Nap_L_SSZ_5kmh_2.mp4, (b) Nap_L_SSZ_7kmh_2.mp4, (c) Nap_L_SSZ_10kmh_1.mp4*

4.1.9. Scenarij Natrag_Lijevo_SZS

U tablici 4.10. dani su rezultati provedenih testova za ovaj scenarij za sve tri postavljene brzine kretanja vozila.

Tab 4.10. *Rezultati testiranja algoritma autonomnog parkiranja za scenarij Natrag_Lijevo_SZS za brzine kretanja vozila 5km/h, 7.5km/h i 10km/h*

Brzina kretanja vozila u simulatoru	**Broj uspješnih parkiranja**	**Broj neuspješnih parkiranja**	**Naziv video sekvence neuspješnog parkiranja (P.4.1.)**	**Postotak uspješnosti parkiranja**
5km/h	10	0	-	100.00%
7.5km/h	10	0	-	100.00%
10km/h	10	0	-	100.00%
Sve brzine zajedno	30	0	-	100.00%

Za parkiranje unatrag lijevo između dva vozila postotak uspješnosti iznosi 100%. Kao i za ovakav slučaj na desnoj strani, radi se o jednostavnom manevru i vozilo je uspješno parkirano svaki puta.

4.1.10. Scenarij Natrag_Lijevo_SSZ

U tablici 4.11. dani su rezultati provedenih testova za ovaj scenarij za sve tri postavljene brzine kretanja vozila.

Tab 4.11. *Rezultati testiranja algoritma autonomnog parkiranja za scenarij Natrag_Lijevo_SSZ za brzine kretanja vozila 5km/h, 7.5km/h i 10km/h*

Brzina kretanja vozila u simulatoru	**Broj uspješnih parkiranja**	**Broj neuspješnih parkiranja**	**Naziv video sekvence neuspješnog parkiranja (P.4.1.)**	**Postotak uspješnosti parkiranja**
5km/h	9	1	Naz_L_SSZ_5kmh_5.mp4	90.00%
7.5km/h	9	1	Naz_L_SSZ_7kmh_4.mp4	90.00%
10km/h	9	1	Naz_L_SSZ_10kmh_3.mp4	90.00%
Sve brzine zajedno	27	3	-	100.00%

Za razliku od prethodnog scenarija, u ovom scenariju vozilo ima tri pogrešna parkiranja, i to po jedno za svako od testnih brzina. Pri tome je dva puta gazilo desnu liniju (sl 4.15. (b), (c)) i jednom lijevu liniju (Sl. 4.15. (a)). Ovdje je pretpostavka da je razlog loših parkiranja manjak računalnih resursa. Naime, u jednom slučaju se vozilo previše ukoso parkiralo i tako nagazilo lijevu liniju - dakle predug period je prošao od pokretanja manevra do završetka istog (vozilo je predugo bilo u fazi skretanja). U druga dva slučaja manevar se prekasno pokrenuo.

a) b) c)

Sl. 4.15. *Prikaz neuspješnih pokušaja parkiranja za testni scenarij Natrag_Lijevo_SSZ (a) Naz_L_SSZ_5kmh_5.mp4, (b) Naz_L_SSZ_7kmh_4.mp4, (c) Naz_L_SSZ_10kmh_3.mp4*

U tablici 4.12. dani su skupni rezultati provedenih testova za parkiranja ulijevo za tri razmatrane brzine kretanja vozila. Iz tablice se vidi kako vozilo najbolje parkira za najmanju

brzinu jer se u tom slučaju radi najmanja pogreška prilikom proračuna korekcije položaja vozila prije samog manevra parkiranja. Rezultati za brzine 7.5km/h i 10km/h su približno jednaki. Iz prethodno navedenih slika može se zaključiti da je razlog većine neuspješnih testova manjak računalnih resursa te pogreška prilikom proračuna korekcije položaja vozila.

Tablica 4.12. *Skupni rezultati testiranja algoritma autonomnog parkiranja ulijevo*

Brzina kretanja vozila u simulatoru	**Broj uspješnih parkiranja**	**Broj neuspješnih parkiranja**	**Postotak uspješnosti parkiranja**
5km/h	47	3	94.00%
7.5km/h	42	8	84.00%
10km/h	43	7	86.00%
Sve brzine zajedno	132	18	88.00%

U tablici 4.13. dani su skupni rezultati testiranja algoritma autonomnog parkiranja za sve testirane scenarije za obje strane ceste. Iz nje se može vidjeti da se vozilo značajno bolje parkira unatrag nego unaprijed. Razlog tomu je to što se parkiranje unatrag izvodi iz jednog manevra i vozilo se ne mora pomicati prije izvođenja manevra kako bi si stvorilo dovoljno prostora za sam manevar. Također prilikom parkiranja unaprijed se vidi najmanji postotak uspješnosti za brzinu od 7.5km/h. Razlog tome je pogreška pri računanju korekcije položaja vozila u ovisnosti o brzini vozila koja je ranije pojašnjena. Može se vidjeti i da za brzinu od 5km/h prilikom parkiranja unaprijed vozilo ima najviše uspješnih parkiranja. U većini netočnih parkiranja radi se o kombinaciji grešaka proračuna korekcija položaja vozila i o ograničenosti računalnih resursa. Za zaključiti je da se korekcija položaja vozila nešto bolje podesila za brzinu od 5km/h nego za ostale brzine.

Tablica 4.13. Kumulativni r*ezultati testiranja algoritma autonomnog parkiranja*

Brzina kretanja vozila u simulatoru	**Broj uspješnih parkiranja**		**Broj neuspješnih parkiranja**		**Postotak uspješnosti parkiranja**	
	Naprijed	**Natrag**	**Naprijed**	**Natrag**	**Naprijed**	**Natrag**
5km/h	54	37	6	3	90.00%	92.50%
7.5km/h	47	38	13	2	78.33%	95.00%
10km/h	51	37	9	3	85.00%	92.50%
Sve brzine zajedno	152	112	28	8	84.44%	93.33%
	264		36		88.00%	

4.2. Osvrt na dobivene rezultate

Nakon testiranja algoritma i predstavljanja rezultata zaključeno je da algoritam autonomnog parkiranja radi ispravno u 88% testnih slučajeva. Prilikom testiranja algoritma za autonomno parkiranje željeno parkirno mjesto je detektirano i klasificirano ispravno u 100% slučajeva, dok je izvođenje manevra parkiranja bilo uspješno u nešto manjem postotku (već spomenutih 88%). Jedan od razloga nešto lošijeg izvođenja manevra parkiranja u odnosu na detekciju i klasifikaciju je ograničenost resursa. Naime prilikom testiranja algoritma primijećeno je da se za identične uvjete scene za svaki od testnih slučajeva testno vozilo svaki puta parkira drugačije. Neki od potencijalnih problema su to što se na istom računalu pokreću CARLA server, nekoliko CARLA klijenata, program za snimanje pozadine (Kazam) i sam algoritam autonomnog parkiranja, što zaokupljuje veliki dio radne memorije i procesorskih resursa. Zbog toga vozilo ne detektira parkinge u istom trenutku (u odnosu na trenutak u kojem je započeo kretanje) za iste testne slučajeve, niti se *sleep()* funkcije koje se koriste za izvršavanje manevra parkiranja odvijaju jednako dugo. Na primjer *sleep(2)* funkcija „uspavljuje“ izvođenje algoritma parkiranja na 2 sekunde, no zbog prevelike opterećenosti računala ona se nekad izvodi 2.1 sekundu a nekad 2.01 sekundu i to unosi pogrešku u manevar parkiranja. Posljedica toga je da je prilikom manevara koji se sastoje od više pomaka automobila (ne parkiranje iz jednog manevra poput parkiranja unatrag) postotak pogrešnih parkiranja nešto veći. Dobar pokazatelj jednostavnog parkiranja je parkiranje unatrag, gdje se nakon detekcije vozilo pomiče za proračunatu vrijednost i pomoću jednog manevra se testni automobil parkira na detektirani parking. Postotak točnosti za sve radnje parkiranja unazad iznosi 93.33% (Tablica 4.13), dok je kumulativna točnost parkiranja za sve

manevre prema naprijed 84.44% (Tablica 4.13.). Razlog tome je naveden ranije. Sva parkiranja unatrag vrše se iz jednog manevra, dok se dio parkiranja unaprijed vrši iz više pomaka vozila. Također, primijećeno je da je za brzinu od 5 km/h najveći postotak točnih parkiranja unaprijed i iznosi 90%, a za brzinu od 7.5 km/h taj je postotak najmanji i iznosi 78,33%. Razlog ovakvih rezultata je taj što se prilikom računanja korekcije položaja vozila koja ovisi o brzini vozila unosi najveća pogreška za srednju brzinu jer se potrebna korekcija linearizirala radi jednostavnosti računanja iste.

5. ZAKLJUČAK

Tema ovog rada je problem autonomnog parkiranja vozila. Stvoren je vlastiti algoritam za detekciju i klasifikaciju parkirnih mjesta te za izvođenje potrebnih manevara kako bi se vozilo uspješno samostalno parkiralo. Algoritam kao ulaz prima sliku s kamere iz CARLA simulatora i detektira parkirna mjesta, a potom izvodi manevar parkiranja. Kompletan rad je napisan u Python programskom jeziku. Za obradu slike korištena je OpenCV biblioteka, a prilikom razvoja i testiranja korišten je CARLA simulator. Parkirna mjesta su se detektirala na način da su se tražile oznake istih uz cestu. Nakon što se utvrde lokacije parkinga, pokreće se drugi dio algoritma u kojemu se klasificira parkirno mjesto i to na način da se na slici, na prethodno detektiranim lokacijama parkinga, traže konture na temelju koji se odlučuje o tome da li je mjesto slobodno ili ne. U konačnici se, nakon što se utvrdi da je parkirno mjesto slobodno, proračunava korekcija položaja vozila u odnosu na trenutak detekcije i korekcija položaja vozila u ovisnosti o brzini vozila te se određuje do koje se lokacije vozilo mora kretati i koji manevar koji treba izvesti kako bi se uspješno parkiralo.

Prilikom testiranja utvrđeno je da algoritam uspijeva obraditi 10 slika u sekundi, što je dovoljno za rad sustava u stvarnom vremenu. Kada se koristi kamera koja snima više od 10 slika u sekundi, npr. 30, obrađivat će se svaka treća slika jer se scena i okolni uvjeti ne mijenjaju značajno između okvira za niske brzine kretanja do 10km/h. Brzina algoritma detekcije i klasifikacije parkirnog mjesta bi se mogla dodatno povećati kada bi se algoritam optimizirao i preveo u neki od nižih jezika, poput C-a ili kada bi se obrada slike vršila na okvirima manje rezolucije. Algoritam parkiranja radi uz točnost od 88.67%. Kako bi se postigao napredak u ovom segmentu, potrebno je dodati neke od metoda praćenja parkinga i adaptivnog zakretanja kotača u ovisnosti o položaju vozila u odnosu na parkirno mjesto. Ovaj algoritam je testiran u simulatoru, i samim time su uvjeti na korištenom parkiralištu iznimno povoljni za rad, tj. nema nikakvih šumova ili neočekivanih događaja. Da bi se ovaj algoritam mogao primijeniti i u realnom svijetu, potrebno ga je dodatno doraditi, ponajviše metodu klasifikacije parkirnog mjesta jer bi potencijalni šumovi poput smeća ili mrlja od ulja mogli utjecati na točnost klasifikacije.

LITERATURA

[1] Changmu Seo, Joongsik Kim, Yunhee Lee, Whoi Yul Kim, „Vision-based Approach in finding multiple parking stall entrance“, 2018

[2] Junzhao Liu, Mohamed Mohandes, Mohamed Deriche, „A Multi-Classifier Image Based Vacant Parking Detection System“ 2013

[3] Seung-Hyun Kim, Joong-Sik Kim, Whoi-Yul Kim, „A method of detecting parking slot in hough pace and pose estimation using rear view image for autonomous parking system“, 2016

[4] Debaditya Acharaya, Weilin Yan, Kourosh Khoshelham, „Real-time image-based parking occupancy detection using deep learning“, 2017

[5] Jae Kyu Suhr, Ho Gi Jung ,“Automatic Parking Space Detection and Tracking for Underground and Indoor Environments,“

[6] Wang Lixia, Jiang Dalin, „A method of Parking space detection based on image segmentation and LBP“ , 2012 Fourth International Conference on Multimedia Information Networking and Security

[7] Plamen Petrov, Fawzi Nashashibi, Member, Mohamed Marouf „Path Planning and Steering Control for an Automatic Perpendicular Parking Assist System“

[8] Saeid Sedighi, Duong-Van Nguyen, Klaus-Dieter Kuhnert, „Implementation of a Parking State Machine on Vision-Based Auto Parking Systems for Perpendicular Parking Scenarios“, 2019 6th International Conference on Control, Decision and Information Technologies (CoDIT’19)

[9] Python 3.8.2. documentation https://docs.python.org/3/index.html , [10.2.2020]

[10] About OpenCV https://opencv.org/about/ [10.2.2020]

[11] Install OpenCV-Python Ubunntu https://docs.opencv.org/3.4.1/d2/de6/tutorial_py_setup_in_ubuntu.html [10.2.2020]

[12] Numpy documentation https://numpy.org/doc/ [10.2.2020]

[13] SciPy installation guide https://scipy.org/install.html [10.2.2020]

[14] CARLA documentation https://carla.readthedocs.io/en/latest/ [10.2.2020]

[15] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, V. Koltun, CARLA: An Open Urban Driving Simulator, Proceedings of the 1st Annual Conference on Robot Learning, str. 1-16, 2017

[16] M. Vranješ, R. Grbić, predavanja iz kolegija Digitalna obrada slike i videa za autonomna vozila, studij Automobilsko računarstvo i komunikacije, ak.god. 2018/2019., „postupci za predobradu slike“ https://loomen.carnet.hr/pluginfile.php/2459745/mod_resource/content/1/3.%20Postupci%20za%20predobradu%20digitalne%20slike.pdf [14.2.2020]

[17] Dr. Sc. Snježana Barić, Luisa Trombetta Burić, prof., Katarina Sablić, „Linearna perspektiva i optičke iluzije“, 2015

[18] M. Vranješ, R. Grbić, predavanja iz kolegija Digitalna obrada slike i videa za autonomna vozila, studij Automobilsko računarstvo i komunikacije, ak.god. 2018/2019., „Segmentacija slike“ https://loomen.carnet.hr/pluginfile.php/2464119/mod_resource/content/1/3.%20Segmentacija%20slike.pdf [30.3.2020]

[19] Implementing Hough Transform http://sidekick.windforwings.com/2012/12/implementing-hough-transform.html [14.2.2020.]

[20] CARLA Linux build https://carla.readthedocs.io/en/latest/build_linux/ [30.3.2020]

SAŽETAK

U ovom radu razvijeno je programsko rješenje za autonomno parkiranje vozila. Algoritam obrađuje okvire dobivene s kamere postavljenje na bočnoj strani automobila, detektira i klasificira parkirna mjesta te potom parkira vozilo na prvo slobodno mjesto koje je pronašao. Kompletan kod za detekciju i klasifikaciju parkinga se sastoji od sekvence funkcija za obradu slike pomoću kojih se na posljetku dobiju informacije o lokaciji parkinga i njegovoj zauzetosti. Osim dijela za detekciju i klasifikaciju parkirnih mjesta, rješenje sadrži i dio vezan za manevriranje vozilom kako bi se ono parkiralo na detektirano slobodno mjesto. Algoritam je sposoban obrađivati deset okvira u sekundi, što omogućuje rad u stvarnom vremenu za tipične brzine kretanja po parkiralištu. Kompletan kod je napisan u Python programskom jeziku uz korištenje OpenCV biblioteke za obradu slike. Algoritam je testiran u CARLA simulatoru za tri različite brzine kretanja vozila 5km/h, 7.5km/h i 10km/h, za tri scenarija parkiranja unaprijed i dva scenarija parkiranja unatrag za lijevu i desnu stranu. Prilikom testiranja utvrđeno je da je algoritam u 100% slučajeva ispravno detektirao i klasificirao parkirna mjesta te da je ukupni postotak točnosti parkiranja nakon detekcije i klasifikacije 88.67%.

Ključne riječi: detekcija parkinga, detekcija linija, autonomno parkiranje, OpenCV, ADAS

SOFTWARE SOLUTION FOR VEHICLE PARK ASSISTANCE

ABSTRACT

In this work, an autonomous parking assistance software solution was developed. An algorithm detects and classifies parking spaces by processing frames received from the camera placed on the side of the ego vehicle and parks itself on the first detected free parking spot. Complete code is made of sequence of image processing functions which provide information on the location of the parking spot and its occupation. In addition to image processing, the algorithm consists of a set of predefined parking maneuvers, which are used to park vehicle on a vacant parking space. The algorithm is capable of processing ten frames per second, allowing real-time operation for typical speeds around parking lot. Complete algorithm is written in Python programming language using OpenCV image processing library. The algorithm was tested in CARLA simulator for three different speeds of 5km/h, 7.5km/h and 10km/h, for three forward parking scenarios and two reverse parking scenarios for parking on the left and right side of the parking lot. While testing, it was found that the algorithm correctly detected and classified parking spaces in 100% of cases, and overall parking accuracy after parking detection and classification was 88.67%.

Keywords: parking detection, line detection, autonomous parking, ADAS, OpenCV

ŽIVOTOPIS

Marko Kuprešak rođen je 30. svibnja 1995. godine u Vinkovcima, Republika Hrvatska. Od 2003. godine. pohađao je osnovnu školu Mate Lovraka u Županji. Nakon završene osnovne škole upisuje Prirodoslovno matematičku gimnaziju u Županji koju završava 2014. godine. Iste godine upisuje Preddiplomski sveučilišni studij elektrotehnike na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku na Sveučilištu J.J. Strossmayera. Nakon završenog preddiplomskog studija, 2017. godine upisuje diplomski studij Automobilskog računarstva i komunikacija u Osijeku.

Potpis:

_____

PRILOZI

P.3.1. – Mapa u kojoj se nalaze sve potrebne Python skritpte za uspješno pokretanje algoritma

P.3.2. – Mapa sa primjerima rada algoritma

P.4.1. – Mapa s testnim video sekvencama