Inteligentno upravljanje prometnim raskrižjima zasnovano na računalnom vidu

Borovac, Edvin

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni diplomski studij Automobilsko računarstvo i komunikacije

INTELIGENTNO UPRAVLJANJE PROMETNIM RASKRIŽJIMA ZASNOVANO NA RAČUNALNOM VIDU

Diplomski rad

Edvin Borovac

Osijek, 2024.

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

Obrazac D1: Obrazac za ocjenu diplomskog rada na sveučilišnom diplomskom studiju

Ocjena diplomskog rada na sveučilišnom diplomskom studiju

Ime i prezime pristupnika:	Edvin Borovac
Studij, smjer:	Sveučilišni diplomski studij Automobilsko računarstvo i
Mat. br. pristupnika, god.	D-66 ARK, 07.10.2022.
JMBAG:	0165073002
Mentor:	izv. prof. dr. sc. Ratko Grbić
Sumentor:	
Sumentor iz tvrtke:	Zvonimir Kaprocki
Predsjednik Povjerenstva:	prof. dr. sc. Mario Vranješ
Član Povjerenstva 1:	izv. prof. dr. sc. Ratko Grbić
Član Povjerenstva 2:	prof. dr. sc. Marijan Herceg
Naslov diplomskog rada:	Inteligentno upravljanje prometnim raskrižjima zasnovano na računalnom vidu
Znanstvena grana diplomskog rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak diplomskog rada:	Prometne gužve svakim danom postaju sve veće zbog prenapučenosti gradova i sve većeg broja automobila. U gotovo svakom gradu postoje kritična raskrižja gdje svakodnevno dolazi do stvaranja gužve uzrokovane neoptimalnim radom semafora, koji je uzima u obzir opterećenje pojedine ceste u danom trenutku. U okviru ovog diplomskog rada potrebno je načiniti rješenje koje će regulirati rad semafora na raskrižju ovisno o broju vozila koja prometuju određenom prometnicom. Pametni algoritam upravljanja semaforima treba svoje odluke o načinu rada donositi

Datum ocjene pismenog dijela diplomskog rada od strane mentora:	20.09.2024.
Ocjena pismenog dijela diplomskog rada od strane mentora:	Izvrstan (5)
Datum obrane diplomskog rada:	30.09.2024.
Ocjena usmenog dijela diplomskog rada (obrane):	Izvrstan (5)
Ukupna ocjena diplomskog rada:	Izvrstan (5)
Datum potvrde mentora o predaji konačne verzije diplomskog rada čime je pristupnik završio sveučilišni diplomski studij:	30.09.2024.

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

IZJAVA O IZVORNOSTI RADA

Osijek, 30.09.2024.

Ime i prezime Pristupnika:	Edvin Borovac
Studij:	Sveučilišni diplomski studij Automobilsko računarstvo i komunikacije
Mat. br. Pristupnika, godina upisa:	D-66 ARK, 07.10.2022.
Turnitin podudaranje [%]:	2

Ovom izjavom izjavljujem da je rad pod nazivom**: Inteligentno upravljanje prometnim raskrižjima zasnovano na računalnom vidu**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ratko Grbić

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 pristupnika:

SADRŽAJ

1. UVOD

U današnjem svijetu, s brzim rastom urbanizacije i povećanjem broja vozila, prometne gužve postale su svakodnevni problem u mnogim gradovima. Prometne gužve uzrokuju nezadovoljstvo kod vozača zbog dugotrajnog čekanja u kolonama što dovodi do gubitka vremena i ekonomskih resursa. Također, prometne gužve loše utječu na okoliš zbog povećanja zagađenja zraka. Jedno od ključnih mjesta gdje se pojavljuju prometne gužve su prometna raskrižja na važnim prometnicama. Upravljanje prometom na raskrižjima često se temelji na fiksno zadanim ciklusima svjetlosnih prometnih znakova (u daljnjem tekstu bit će nazivani semaforima) koji ne uzimaju u obzir stvarno stanje prometa po pojedinim trakama određenog raskrižja što često rezultira nepreglednim gužvama. Stoga, razvijanje inteligentnog sustava za upravljanje prometnim raskrižjima postaje od izuzetne važnosti kako bi se unaprijedila protočnost prometa te smanjilo vrijeme čekanja vozača.

Problem kojim se bavi ovaj diplomski rad jest optimizacija upravljanja semaforima na prometnim raskrižjima. Tradicionalni semaforski sustavi koriste unaprijed definirane cikluse bazirane na tipičnoj gustoći prometa na tom raskrižju, međutim, promet se po određenim trakama često značajno mijenja u ovisnosti o dobu dana, vremenskim uvjetima ili nekom drugom nepredvidljivom faktoru, što može dovesti do nepotrebnog čekanja i zagušenja. U novije se vrijeme pojavljuje sve više rješenja za upravljanje semaforskim sustavom koje svoj rad temelje na prikupljanju informacija s različitih senzora poput kamera montiranih na stupove semafora ili nekih drugih senzora kao što su induktivni senzori postavljeni u kolnik ili infracrveni senzori za detekciju vozila, a sve s ciljem efikasnijeg upravljanja prometom raskrižja. Induktivne petlje za detekciju vozila su daleko najprecizniji način detekcije vozila zbog svoje sposobnosti da, neovisno o poziciji vozila u koloni na prometnom raskrižju, petlja detektira prisutnost vozila. Na temelju toga, računalni sustav može odrediti broj vozila i prikupljati informacije o protočnosti prometa. No, unatoč svojoj preciznosti, takvi senzori zahtjevaju značajniju prilagodbu infrastrukture prometnog raskrižja, što uvelike povećava cijenu za implementaciju sustava za inteligentno upravljanje semaforima. Također, uz svoju veliku cijenu, takvi senzori imaju poteškoće pri određivanju tipa vozila, kao što su vozila hitne službe u slučaju kada sustav za upravljanje semaforima treba takvim vozilima dati prioritet. Za razliku od induktivnih senzora, kamere, koje se najčešće montiraju na stupove semafora ili rasvjete, mogu detektirati veći spektar situacija i sudionika u prometu, ali kod njih može nastati problem pri preciznosti detekcije i

prebrojavanja vozila. Unatoč tome, upravo zbog cijene i jednostavnosti implementacije, danas se kamere pojavljuju kao jeftino rješenje koje pružaju veliku količinu informacija o sudionicima u prometu kao što su broj vozila, tipovi vozila, broj pješaka, detekciju prekšaja i sl.

Cilj ovog rada je razviti algoritam za inteligentno upravljanje prometnim raskrižjima koji bi smanjio prosječno vrijeme čekanja vozila i povećao protočnost prometa. Kako bi se riješio navedeni problem optimalnog upravljanja prometnim raskrižjem, potrebno je obuhvatiti nekoliko ključnih tema. Prva tema odnosi se prikupljanje informacija o trenutnom stanju prometa na odgovarajućem prometnom raskrižju, što uključuje korištenje kamera za prepoznavanje i prebrojavanje vozila na zasebnim prometnim trakama unutar raskrižja. Informacija o broju vozila u zasebnim prometnim trakama se u ovom slučaju temelji na detekciji vozila u digitalnoj slici dobivenoj s kamere. Validacija i testiranje ovakvog sustava u stvarnim uvjetima, odnosno na stvarnom raskrižju tijekom prometnih gužvi, predstavljala bi idealan pristup za procjenu njegove funkcionalnosti i učinkovitosti. Međutim, zbog praktičnih ograničenja i sigurnosnih razloga, nije bilo moguće provesti testiranje u stvarnom prometnom okruženju. Stoga je za simulaciju i evaluaciju sustava odabran CARLA simulator, koji pruža dovoljno realistične uvjete i fleksibilnost potrebnu za ovakvu vrstu istraživanja. Njegova glavna uloga je simulacija prometnog raskrižja, na kojem je moguće programski utjecati na količinu prometa u pojedinim prometnim trakama, te na taj način simulirati različite prometne scenarije na temelju kojih je testiran predloženi algoritam za inteligentno upravljanje prometnim semaforima. To je napravljeno tako da je na odabranom raskrižju unutar simulatora stvorena prometna gužva te su postavljene kamere koje snimaju svaku od traka i prebrojavaju vozila pomoću detektora za prepoznavanje i prebrojavanje vozila, čime se dobiva informacija o trenutnoj prometnoj situaciji na raskrižju. Treća ključna tema je optimizacija semaforskih ciklusa za promatrano prometno raskrižje, koja podrazumijeva razvijanje algoritma koji koristi prikupljene podatke o prometu za prilagodljivo upravljanje semaforima određenog raskrižja. Na temelju broja vozila u svakoj od traka na raskrižju i drugih parametara, kao što su vremena čekanja svake od traka, razvijen je sustav koji obrađuje te podatke i upravlja radom semafora na optimiziran način. U ovom radu napravljen je algoritam koji upravlja isključivo radom semafora zaduženih za kontrolu prometa vozila. Na kraju, evaluacija performansi razvijenog algoritma je također važna, a uključuje mjerenje učinkovitosti predloženog sustava u usporedbi s tradicionalnim načinom rada semafora na prometnim raskrižjima.

U drugom poglavlju ovog rada napravljen je detaljan pregled postojećih modernih tehnologija i predloženih rješenja za inteligentno upravljanje semaforima na prometnom raskrižju. Treće poglavlje obuhvaća postavljanje prometnog scenarija unutar CARLA simulatora, prikupljanje podataka s postavljenih kamera te implementaciju algoritma za detekciju i prebrojavanje vozila. Također, treće poglavlje opisuje način rada razvijenog algoritma za upravljanje semaforima na temelju ulaznih podataka o broju vozila u zasebnim trakama u raskrižju. Četvrto poglavlje sadrži rezultate i analizu rješenja na temelju usporedbe dobivenog rješenja sa standardnim načinom upravljanja semafora. To je napravljeno na temelju usporedbe vremena čekanja po vozilu za svaku zasebnu traku u prometnom raskrižju te na temelju prosječnog vremena čekanja za sva vozila koja su prošla kroz raskrižje u vremenu testiranja. Zadnje poglavlje sadrži zaključak u kojemu su sažeti rezultati te su obuhvaćene glavne poante rada uz ideje za unaprjeđivanje sustava razvijenog u ovom radu.

2. ANALIZA PROBLEMA I PREGLED POSTOJEĆIH RJEŠENJA ZA INTELIGENTNO UPRAVLJANJE SEMAFORIMA NA RASKRIŽJU

2.1. Problematika upravljanja semaforima u prometnim raskrižjima

Efikasno upravljanje prometom na raskrižjima opremljenim semaforima predstavlja ključni izazov u urbanim sredinama, gdje rastuća populacija i sve veći broj vozila stvaraju sve ozbiljnije prometne probleme. Raskrižja su mjesta na kojima se susreću i isprepliću različiti tokovi vozila. Takva mjesta često uzrokuju prometne gužve, povećanje emisije štetnih plinova i smanjenje učinkovitosti prometnih sustava. Tradicionalni način upravljanja semaforskim sustavom, koji se oslanjaju na unaprijed određene vremenske cikluse semafora, sve se više pokazuju neadekvatnima u suočavanju s promjenjivim i dinamičnim uvjetima prometa na raskrižjima [1]. U nastavku ovog poglavlja dan je uvid u ograničenja tradicionalnog pristupa upravljanja semaforima te u različite moderne tehnologije korištene za prilagodljivo upravljanje semaforskim ciklusima na prometnim raskrižjima.

2.1.1. Tradicionalni pristupi i njihova ograničenja

Tradicionalni sustavi upravljanja prometom na raskrižjima koriste unaprijed određene vremenske cikluse za semafore, koji ne uzimaju u obzir trenutne prometne uvjete na raskrižju. Ovi sustavi najčešće funkcioniraju prema fiksnim vremenskim intervalima koji su određeni prilikom projektiranja raskrižja, bez obzira na to koliko su relevantni u stvarnom vremenu [1]. Takav pristup može rezultirati situacijama u kojima semafor dopušta prolazak vozilima u traci na prometnom raskrižju u kojoj nema prometa, dok se preostalim trakama prometnog raskrižja vozila nepotrebno zadržavaju, čime se na taj način povećava vrijeme čekanja pojedinih vozila.

Ovakvo upravljanje, s fiksnim vremenskim ciklusima, može dovesti do niza problema, uključujući stvaranje prometnih gužvi, povećanu potrošnju goriva i emisiju CO_2 te smanjenje sigurnosti na cestama. Posebno je kritično što takvi sustavi nisu u stanju adekvatno odgovoriti na neočekivane situacije kao što su prometne nesreće ili nagle promjene u protoku prometa [2]. Također, nedostatak prilagodljivosti ovih sustava može uzrokovati dugotrajna zadržavanja vozila, što dodatno doprinosi povećanju onečišćenja zraka i negativno utječe na kvalitetu života u urbanim sredinama.

2.1.2. Izazovi u prilagodljivom upravljanju prometnim raskrižjima i inovativna rješenja

Promet na raskrižjima može biti vrlo dinamičan, s velikim varijacijama u broju vozila i njihovim smjerovima kretanja tijekom dana. U velikim gradovima, poput Toronta i New Yorka, prometne gužve mogu produljiti vrijeme putovanja i do 60%, što znatno utječe na svakodnevni život građana [2]. Smanjenje zagušenja i optimizacija protoka prometa su od ključnog značaja za održivi razvoj urbanih područja u budućnosti.

Jedan od ključnih izazova u prilagodljivom upravljanju prometom je prikupljanje točnih i pravovremenih podataka o prometu. Tradicionalne metode prikupljanja podataka, poput indukcijskih petlji i kamera, često nisu dovoljno brze ili precizne, što može dovesti do netočnih procjena i neadekvatnog upravljanja prometom. Napredak u tehnologijama poput bežičnih senzorskih mreža (WSN) i analitike velikih podataka obećava poboljšanja u ovom području, ali postoje i izazovi poput visoke cijene instalacije i održavanja takvih sustava tako da se u trenutnim rješenjima većinom radi kompromis između performansi i cijene sustava [3].

Moderni sustavi za upravljanje prometom često zahtijevaju integraciju različitih tehnoloških rješenja, uključujući IoT (engl. *Internet of Thing*) uređaje, senzore, kamere i sustave za predikciju. Međutim, integracija heterogenih sustava predstavlja značajan izazov zbog različitih standarda, protokola i operativnih zahtjeva. Ova raznolikost u tehnologijama često dovodi do problema u interoperabilnosti, gdje različiti sustavi ne mogu učinkovito komunicirati ili dijeliti podatke zbog nespojivih formata ili komunikacijskih protokola. To može rezultirati fragmentiranim sustavom upravljanja prometom koji ne koristi u potpunosti potencijal dostupnih podataka i tehnologija. Osim toga, nedostatak zajedničkih standarda za podatkovne formate i protokole može dovesti do poteškoća u skaliranju sustava. Svaka nova komponenta ili tehnologija koja se dodaje sustavu mora biti pažljivo prilagođena postojećoj infrastrukturi, što, većinom, značajno povećava troškove i složenost implementacije. Na primjer, sustavi temeljeni na starijim tehnologijama možda neće moći podržati napredne analitičke funkcije koje zahtijevaju moderne tehnologije poput umjetne inteligencije i strojnog učenja.

Osim tehničkih izazova, postoji i problem sigurnosti podataka. Različiti sustavi često koriste različite sigurnosne protokole, što može dovesti do ranjivosti u sustavu i potencijalno omogućiti neovlašteni pristup osjetljivim podacima o prometu. To zahtijeva dodatne mjere za osiguranje podataka, kao i

stalnu nadogradnju sigurnosnih standarda kako bi se osiguralo da su svi dijelovi sustava jednako zaštićeni.

Uspješna integracija heterogenih sustava zahtijeva ne samo tehničke vještine, već i temeljitu suradnju između različitih sektora, kao što su inženjering, IT i upravljanje prometom. Samo kroz koordinirani pristup moguće je razviti sustav koji učinkovito koristi sve dostupne tehnologije za optimizaciju upravljanja prometom u urbanim sredinama. To uključuje razvoj jedinstvenih standarda i protokola koji omogućuju različitim sustavima da nesmetano surađuju, kao i implementaciju fleksibilnih sustava koji se mogu prilagoditi budućim tehnološkim inovacijama.

S obzirom na sve izraženije probleme u upravljanju prometom, istraživači i inženjeri razvijaju nova rješenja koja uključuju upotrebu naprednih tehnologija poput IoT tehnologija, računalnog vida i algoritama strojnog učenja. Inteligentni semafori, opremljeni raznim senzorima i kamerama, mogu, u stvarnom vremenu, prikupljati podatke o gustoći prometa, brzini vozila i vremenu čekanja. Na temelju tih i sličnih podataka, ovi sustavi mogu dinamički prilagoditi vremenske cikluse semafora kako bi se optimizirao protok prometa na raskrižju.

Jedan od najčešćih primjera takvog sustava je kada sustavi temeljeni na analizi videozapisa i računalnom vidu mogu izračunati broj vozila u svakom smjeru te prema tim informacijama prilagoditi vrijeme trajanja zelenog svjetla. Na taj način se smanjuje vrijeme čekanja i povećava učinkovitost prometnog sustava. Osim toga, ovi sustavi mogu integrirati prioritete za hitna vozila, omogućujući im brži prolazak kroz raskrižje, što je ključno u hitnim situacijama [2].

2.2. Pregled postojećih tehnologija i algoritama za inteligentno upravljanje prometom na prometnim raskrižjima

S razvojem urbanih područja i povećanjem broja vozila na cestama, postalo je očito da tradicionalni sustavi upravljanja prometom na prometnim raskrižjima, temeljeni na fiksnim vremenskim ciklusima semafora, više nisu dovoljni za učinkovito rješavanje problema prometnih gužvi i optimizaciju protoka prometa. Kako bi se odgovorilo na ove izazove, razvijaju se i razvijene su različite tehnologije i algoritmi za prilagodljivo upravljanje prometom na prometnim raskrižjima koji koriste napredne metode za analizu podataka u stvarnom vremenu i prilagodbu semafora prema trenutnim uvjetima na cestama. Neka od takvih postojećih rješenja opisana su u nastavku ovog potpoglavlja.

Jedan od najznačajnijih pristupa pametnom upravljanju prometom je korištenje IoT tehnologija i računalnog vida. IoT tehnologije omogućuju povezivanje različitih senzora i uređaja u mrežu koja prikuplja podatke o prometu i pomoću algoritama ih obrađuje u stvarnom vremenu. Na primjer, kamere postavljene na raskrižjima mogu pomoću implementiranih algoritama za računalni vid analizirati gustoću prometa. To omogućuje dinamičku prilagodbu rada semafora u skladu s trenutnim uvjetima na cestama. Rad [1] se bavi temom inteligentnog upravljanja semaforima u pametnim gradovima koristeći spomenute tehnike obrade slike i videa te IoT-a. Glavna ideja tog rada je optimizacija vremena trajanja zelenog svjetla na semaforima na raskrižjima na osnovu trenutnog stanja prometa, odnosno na temelju gustoće vozila i broja vozila koja prolaze kroz raskrižje. Autori su se, prilikom izrade sustava za upravljanje semaforima na raskrižju, oslonili na upotrebu Raspberry Pi ploče i OpenCV alata za obradu slika i videa. Kao senzori su korištene kamere koje snimaju promet na različitim pravcima raskrižja, pritom omogućavajući analizu gustoće vozila i broja vozila. Predložena su dva različita algoritma za upravljanje semaforima. Prvi od njih se temelji na gustoći vozila. Taj model koristi slike prometa za analizu gustoće vozila na svakom pravcu raskrižja. Prvi korak je snimanje referentne slike za svaki pravac, koja prikazuje raskrižje bez vozila. Nakon toga, trenutna slika prometa se uspoređuje s referentnom slikom kako bi se odredila gustoća prometa. Na osnovu tih podataka, algoritam dinamički odlučuje o trajanju zelenog svjetla za svaku traku. Drugi algoritam je zasnovan na broju vozila u trakama na raskrižju. Taj algoritam koristi video snimke za prebrojavanje vozila koja prilaze raskrižju. Video snimci se analiziraju kako bi se utvrdio broj vozila koja prolaze kroz određeni pravac u određenom vremenskom intervalu. Na osnovu tih informacija, algoritam određuje koliko će dugo zeleno svjetlo biti uključeno za svaku od traka. Testiranje je izvršeno na Raspberry Pi jednokartičnom računalu koristeći OpenCV alat. Prvi algoritam je testiran na slikama koje su prikazivale četiri trake raskrižja, dok je drugi algoritam testiran na video snimcima. Vrijeme trajanja zelenog svjetla je određeno na osnovu postotka preklapanja slika i broja vozila. Rezultati testiranja pokazuju da predloženi algoritmi mogu značajno smanjiti gužve na raskrižjima jer omogućavaju prilagodljivo upravljanje semaforima na temelju trenutnog stanja u prometu. Na primjer, za jednu traku s većom gustoćom prometa i većim brojem vozila, trajanje zelenog svjetla je duže, što smanjuje nepotrebno zadržavanje na toj traci. Prednosti ovakvog sustava su prvenstveno to što je jeftin za implementaciju i može se proširivati. Također, ovakav sustav ne zahtjeva direktnu povezanost s vozilima, što dodatno smanjuje troškove instalacije u usporedbi s modelima slične namijene koji zahtjevaju specifičnu opremu u vozilima. Dodatno, ovakav jednostavan sustav pruža mogućnost

integracije s mobilnim uređajima i korištenje podataka u stvarnom vremenu što je gotovo uvijek uvjet za pomoćne sustave koji se koriste u prometu. Glavna mana ovakvog sustava je što postoji ograničenje u preciznosti prilikom detekcije vozila u prometu s obzirom da kamera ima ograničenu udaljenost na kojoj može prepoznati vozilo.

Sustavi temeljeni na računalnom vidu koriste napredne algoritme za prepoznavanje vozila, kao što je algoritam *You Only Look Once* (YOLO). Na temelju obrade slika pomoću YOLO detektora, omogućeno je precizno brojanje vozila u trakama na prometnom raskrižju. Ti podaci se potom koriste za optimizaciju vremena trajanja zelenog svjetla na raskrižju, čime se smanjuje nepotrebno čekanje i povećava učinkovitost prometnog sustava. Ovakvi sustavi su pokazali visok stupanj učinkovitosti, smanjujući prosječno vrijeme čekanja vozila na raskrižju za čak 50% u usporedbi s tradicionalnim sustavima. Jedan od takvih sustava opisan je u radu [2]. Osim upotrebe kamera i drugih kompatibilnih senzora, predložena je upotreba vozila opremljenih VANET (*Vehicular ad-hoc networks)* tehnologijom koja može pružiti podatke o položaju, brzini i vremenu čekanja vozila. U ovom radu razvijena su tri različita algoritma za upravljanje semaforima na raskrižju. Prvi algoritam, pod nazivom *Smart Traffic Light Scheduling based on Density* (STLSD), temelji se na gustoći prometa. Ovaj algoritam mjeri broj vozila u "zoni čekanja" za svaki smjer na raskrižju te prilagođava trajanje zelenog svjetla na semaforima tako da prioritetno propušta smjer s većom gustoćom prometa. Cilj je osigurati optimalan protok vozila kroz raskrižje, smanjujući zastoje u smjerovima s većim brojem vozila. Drugi algoritam, nazvan *Smart Traffic Light Scheduling based on Density and Delay Time* (STLSDT), uzima u obzir ne samo gustoću prometa već i vrijeme čekanja vozila. Ako određeno vozilo predugo čeka na prolazak, algoritam prilagođava trajanje zelenog svjetla na semaforu tako da smanji vrijeme čekanja, čak i ako to ponekad znači smanjenje ukupne efikasnosti protoka prometa. Ovaj algoritam omogućuje pravedniju distribuciju vremena zelenog svjetla, izbjegavajući situacije u kojima vozila dugo čekaju unatoč niskoj gustoći prometa. Treći algoritam, *Smart Traffic Light Scheduling for Emergency vehicles* (STLSDE), posebno je dizajniran za situacije u kojima hitna vozila trebaju prioritetni prolaz. Kada sustav detektira prisutnost hitnog vozila, algoritam automatski ignorira druge čimbenike poput gustoće prometa i vremena čekanja te osigurava da hitno vozilo ima slobodan prolaz kroz raskrižje. Ovaj algoritam doprinosi sigurnijem i bržem prolasku hitnih službi, što može biti ključno u kriznim situacijama. Spomenuti algoritmi su testirani pomoću simulacijske platforme *Isolated Intersection Simulator* (IIS) koja simulira promet na raskrižjima. Testiranja su obuhvatila različite scenarije, uključujući normalni promet u periodima kad je najveća gužva na prometnicama,

abnormalni promet koji nastaje zbog radova na cestama te situacije s hitnim vozilima. Rezultati simulacija pokazali su značajna poboljšanja u učinkovitosti prometa kada se koriste predloženi algoritmi u odnosu na tradicionalne metode. Konkretno, algoritam STLSDT je smanjio prosječno vrijeme čekanja na semaforu na 1 sekundu unutar korištene simulacije, u usporedbi sa 7 sekundi čekanja kod fiksnog ciklusa semafora. Glavna prednost ovakvog sustava u odnosu na prethodni sustav je što ima mogućnost prilagodbe načina rada u slučaju hitnih situacija kao što su pojava vozila hitnih službi ili vozila pod pratnjom koja imaju prioritet prolaska. Unatoč tome, performanse sustava u takvim hitnim slučajevima bit će smanjene za sve ostale sudionike u prometu.

U radu [4] je korišten inovativan pristup koji se temelji na povijesnim podacima o prometu u svrhu prilagodljivog upravljanja semaforima. Ova tehnologija koristi analizu prikupljenih podataka o prometu kroz različite periode kako bi predvidjela buduće prometne obrasce i na temelju toga optimizirala vrijeme trajanja zelenih i crvenih svjetala na semaforima. Ovakav sustav kontinuirano prikuplja podatke o prometnim tokovima tijekom različitih dana u tjednu, mjesecu ili godini. Podaci uključuju informacije poput gustoće prometa, prosječne brzine vozila, vremena najvećih opterećenja i drugih relevantnih metrika potrebnih za što bolje razumijevanje situacije u prometu. Analizom tih podataka identificiraju se obrasci i trendovi u svrhu predviđanja budućih prometnih stanja. Za prikupljanje podataka u stvarnom vremenu, sustav se može integrirati s različitim senzorima poput kamera, indukcijskih petlji, GPS uređaja i drugih IoT komponenti. Ova integracija omogućuje sustavu da kombinira povijesne podatke s aktualnim informacijama, pružajući sveobuhvatan pregled prometne situacije i omogućujući brze reakcije na nepredviđene događaje poput nesreća ili iznenadnih zagušenja. Ovakav sustava ima brojne prednosti. Prva od njih je poboljšana efikasnost prometa zbog dinamičkog prilagođavanja trajanja semaforskih stanja koje smanjuje vrijeme čekanja i poboljšava protok vozila kroz raskrižja. Nadalje, ovakav sustav pruža fleksibilnost i skalabilnost jer se može lako prilagoditi različitim urbanim sredinama i prometnim scenarijima te se može lako skalirati dodavanjem novih senzora ili izvora podataka. Ovakav sustav također ima mogućnost kontinuiranog poboljšanja korištenjem strojnog učenja, čime sustav može s vremenom i s povećanjem baze podataka postati sve precizniji i efikasniji. Glavni nedostatci ovakvog sustava najviše su temeljene na kompleksnosti sustava te ponašanje u nepredvidivim situacijama. Integracija različitih izvora podataka i tehnologija može biti tehnički veoma zahtjevna i skupa, posebno u urbanijim sredinama gdje se radi s većom količinom podataka. Također, ovakav sustav se teško nosi s nepredvidivim

situacijama kao što su prometne nesreće ili vremenske nepogode s obzirom da ne prikuplja podatke u stvarnom vremenu nego stvara uzorak na temelju prethodnih situacija.

Rad [5] se bavi problemom upravljanja prometom u urbanim sredinama, gdje se broj vozila eksponencijalno povećava. Ovaj rad predlaže sustav za prilagodljivo upravljanje prometom koji koristi metodu *Urban Traffic Control* (UTC) za optimizaciju prometa u stvarnom vremenu. Sustav uzima u obzir dinamičan prometni tok, s ciljem smanjenja gužvi, vremena putovanja i čekanja vozila na prometnim raskrižjima. Glavna ideja predloženog rješenja je integracija sustava inteligentnih semafora u postojeću prometnu mrežu, bez potrebe za značajnim infrastrukturnim promjenama. Sustav koristi multi-agentski pristup, gdje svaki prometni signal, vozilo i kontrolor raskrižja djeluju kao autonomni agenti koji prikupljaju i obrađuju podatke u stvarnom vremenu. Ti agenti komuniciraju međusobno kako bi donijeli odluke koje optimiziraju protok prometa i smanjuju čekanje na raskrižjima. Autori su u radu koristili kombinaciju senzora, među kojima su infracrveni senzori postavljeni na raskrižjima. Ovi senzori detektiraju vozila te na taj način broje vozila koja prolaze te mjere njihovu brzinu i duljinu. Cilj toga je procjena prometnog volumena. Sustav koristi podatke prikupljene ovim senzorima kako bi prilagodio trajanje faza zelenih svjetala na raskrižjima. Testiranje predloženog sustava provedeno je korištenjem simulacijskog modela u NetLogo okruženju, koji je obuhvatio mrežu od 25 međusobno povezanih raskrižja s ukupno 150 vozila. Testirano je nekoliko scenarija, uključujući tradicionalno upravljanje semaforima, upravljanje semaforima prema protoku prometa (engl. *lane flow*) i upravljanje prema kretanju bez interferencije (engl. *no interference movement*). Rezultati simulacije pokazali su da predloženi prilagodljivi sustav značajno smanjuje prosječno vrijeme čekanja vozila na semaforima. Konkretno, sustav je smanjio prosječno vrijeme čekanja za 25,98% u scenariju upravljanja prema protoku prometa i za 34,16% u scenariju upravljanja prema kretanju bez interferencije u usporedbi s tradicionalnim načinom upravljanja semaforima. Prednosti ovakvog sustava uključuju značajno smanjenje vremena čekanja i gužvi na raskrižjima te jednostavnost integracije sustava u postojeću infrastrukturu s obzirom da sustav ne zahtjeva velike promjene. Najveći nedostatak ovakvog sustava je što zahtijeva kontinuirano prikupljanje i obradu velikih količina podataka što može predstavljati izazov u implementaciji na velikom broju raskrižja. Također, kao i kod drugih sličnih sustava, učinkovitost može biti ograničena u ekstremnim uvjetima prometa.

U radu [6] za upravljanje prometom koristi se napredni sustav koji se temelji na kombinaciji IoT tehnologije i bežične komunikacijske mreže pod nazivom XMesh. Ovaj sustav koristi IoT uređaje za prikupljanje podataka o prometu u stvarnom vremenu i bežičnu mrežu XMesh za komunikaciju između semafora i središnjeg upravljačkog sustava. XMesh mreža omogućava pouzdanu i sigurnu komunikaciju s mogućnošću automatske rekonfiguracije u slučaju prekida veze ili drugih problema. Sustav koristi *ARM Cortex-M3* mikroupravljača za obradu podataka i upravljanje semaforima na raskrižjima, što omogućava fleksibilnu i dinamičku kontrolu prometa na temelju stvarnih uvjeta na cestama. Bežična komunikacija omogućava jednostavniju instalaciju i održavanje semafora, bez potrebe za postavljanjem potencijalno dugih kablova kroz cijeli grad. Sustav je prilagodljiv i može raditi na različitim vrstama raskrižja, od jednostavnih do složenih, te se može konfigurirati za različite situacije poput zatvaranja cesta ili hitnih intervencija. Sigurnosne značajke *XMesh* mreže uključuju višeslojnu enkripciju koja osigurava sigurnost podataka i zaštitu od neovlaštenog pristupa, što je ključno za sprječavanje mogućih napada na sustav. Najveći nedostatak ovakvog sustava predstavlja potencijalne smetnje u bežičnoj komunikaciji bez obzira na implementirane sigurnosne mehanizme.

3. RAZVOJ I IMPLEMENTACIJA INTELIGENTNOG SUSTAVA ZA UPRAVLJANJE SEMAFORIMA NA PROMETNOM RASKRIŽJU

U ovom poglavlju je opisan razvoj i implementacija inteligentnog sustava za upravljanje semaforima koji se temelji na detekciji vozila u prometnom raskrižju i algoritmu za prilagodljivo upravljanje semaforima. Poglavlje opisuje korišteno razvojno okruženje za realizaciju zadanog sustava te korištenje CARLA simulatora za stvaranje realističnog prometnog scenarija i postavljanje kamera na svaku traku prometnog raskrižja. Također, u ovom poglavlju se detaljno opisuju proces postavljanja scenarija prometne gužve na raskrižju i razrada mogućih situacija na raskrižju na temelju broja vozila u svakoj od traka te je na kraju detaljno opisana logika rada algoritma za prilagodljivo upravljanje semaforima na prometnom raskrižju.

3.1. Programsko okruženje i biblioteke

3.1.1. Programsko okruženje

Razvoj *Python* programa odvijao se uz pomoć *Notepad++*, besplatnog uređivača izvornog koda koji podržava više programskih jezika i radi u *MS Windows* okruženju [7]. Radno okruženje koristi *Python* verziju 3.8.19, osiguravajući da se sve *Python* skripte izvršavaju pomoću ove verzije jezika. Ta *Python* verzija odabrana je radi kompatibilnosti s potrebnim bibliotekama za rad u *CARLA* simulatoru. U radno okruženje su integrirani ključni alati za razvoj, uključujući upravitelj paketa *PIP*, koji omogućava jednostavno instaliranje dodatnih *Python* biblioteka.

Python interpretator nadopunjen je s nekoliko važnih paketa koji omogućavaju napredne funkcionalnosti. Među najvažnijim instaliranim paketima su *carla, threading, OpenCV, NumPy, pandas, torch, i cv2*, koji su instalirani putem *PIP*-a i koriste se za komunikaciju s *CARLA* serverom, obradu slike, rad s podacima, računalni vid i višenitni rad pri izvršavanju naredbi.

U ovom radu korišten je CARLA simulator verzije 0.9.15 za prikupljanje podataka za prikupljanje podataka potrebnih za razvoj rješenja i testiranje algoritama. Ova verzija pruža stabilnost i podršku za potrebne funkcionalnosti, uključujući rad s RGB kamerama i postavljanje prometnih scenarija. CARLA koristi klijent-poslužitelj arhitekturu, pri čemu se poslužitelj pokreće kroz zasebnu izvršnu datoteku, dok klijentske skripte omogućuju konfiguraciju i upravljanje vozilima te drugim objektima

u simulaciji [8]. Verzija 0.9.15 također omogućuje fleksibilnu kontrolu nad uvjetima okoliša i napredne funkcionalnosti senzora, što je ključno za prilagodljivo upravljanje prometom u ovom radu.

3.1.2. Korištene biblioteke

Carla je *Python* modul koji omogućava interakciju interpretera s CARLA simulatorom. Ovaj modul omogućava programerima upravljanjem različitim aspektima simulacije, uključujući stvaranje vozila, upravljanje njima, konfiguraciju senzorskih sklopova, kontrolu okolišnih uvjeta i interakciju s ostalim objektima unutar simulirane okoline. Ovaj modul je ključan za skriptiranje i automatizaciju simulacija u *CARLA* simulatoru.

Glob [9] je modul za pronalaženje putanja datoteka i direktorija na osnovu šablona (engl. *wildcard*). Koristi se za pretraživanje direktorija kako bi se pronašle datoteke koje odgovaraju određenom obrascu, npr. sve .txt datoteke u određenom direktoriju.

Os [10] je standardni *Python* modul za interakciju sa operacijskim sustavom. Koristi se za manipulaciju direktorijima i datotekama, rad sa putanjama, upravljanje okruženjem, izvršavanje sistemskih naredbi i još mnogo toga.

Sys [11] je modul za interakciju sa interpreterom *Python* programa. Omogućava pristup sistemskim parametrima i funkcijama, poput pristupa argumentima komandne linije, prekida programa, upravljanja izlazom i greškama, itd.

Time [12] je *Python* modul za rad sa vremenom. Koristi se za praćenje vremena, kašnjenja, mjerenje trajanja izvršavanja koda i druge vremenske operacije.

Cv2 [13] je glavni modul *OpenCV* biblioteke za obradu slike. Koristi se za računalni vid, obradu slika i video zapisa, kao što su detekcija objekata, obrada slika, prepoznavanje lica, itd.

Numpy [14] je biblioteka za rad sa nizovima (matricama) i numeričke operacije. Omogućava rad sa velikim, višedimenzionalnim nizovima i matricama, kao i velikim skupom matematičkih funkcija za operacije nad njima.

Pandas [15] je biblioteka za analizu podataka. Koristi se za manipulaciju i analizu podataka, omogućavajući rad s tablicama, serijama, vremenskim serijama, itd.

Ultralytics.YOLO [16] je dio *YOLO* modela za detekciju objekata razvijenog od strane *Ultralytics*-a. Koristi se za brzo i precizno prepoznavanje objekata u slikama i video zapisima.

Threading [17] je *Python* biblioteka za rad sa nitima. Omogućava paralelno izvršavanje koda unutar jednog procesa putem više niti (eng. *threads*), što je korisno za *I/O* operacije i ostale zadatke koji ne zahtijevaju puno *CPU* resursa.

Torch [18] je osnovni modul *PyTorch* biblioteke za duboko učenje. Koristi se za gradnju i treniranje neuronskih mreža, kao i za izvođenje operacija nad tenzorima.

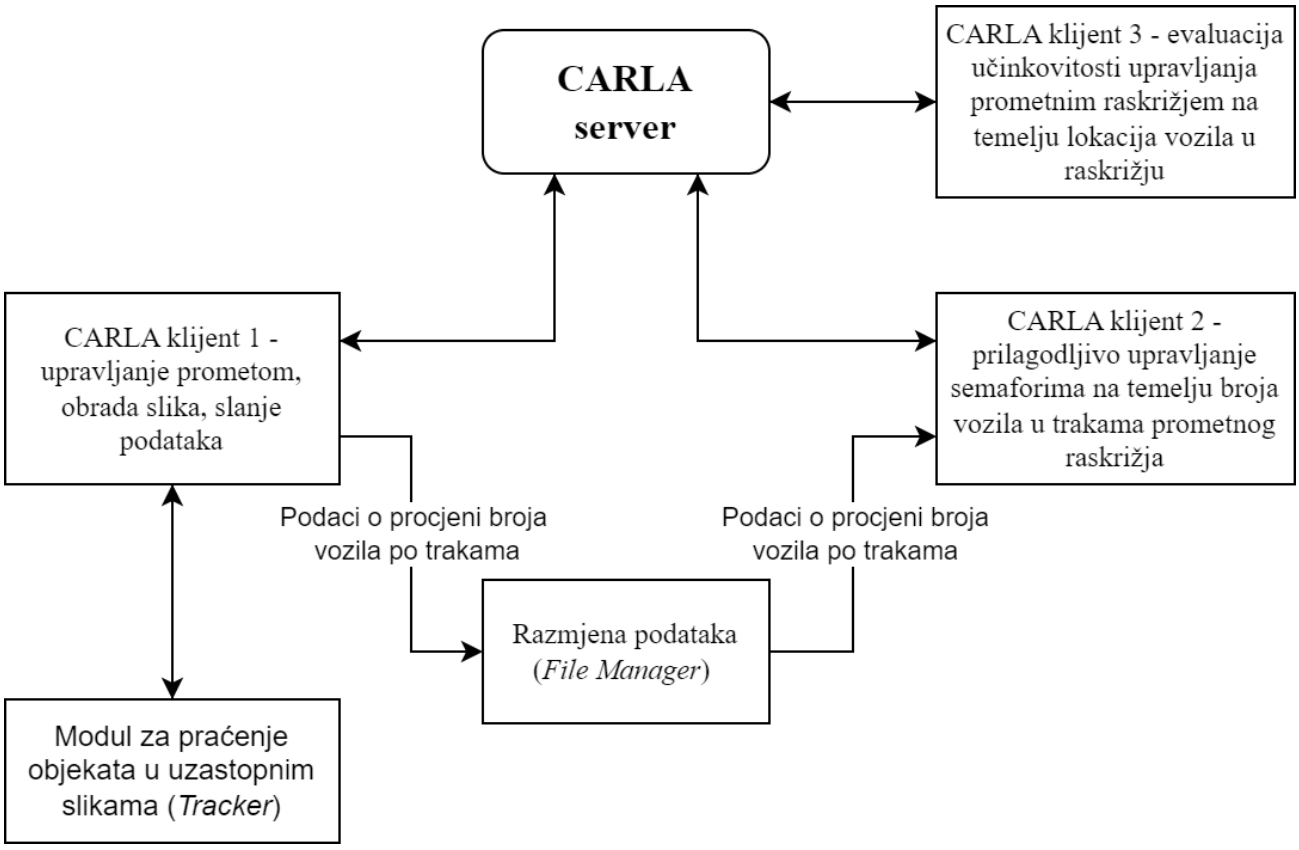
Torchvision.models [19] je dio *Torchvision* modula koji sadrži unaprijed trenirane modele za zadatke poput klasifikacije, segmentacije i detekcije objekata. Omogućava jednostavno korištenje i prilagođavanje popularnih arhitektura neuronskih mreža koje su već trenirane na velikim podatkovnim skupovima.

3.2. Arhitektura predloženog rješenja

Predloženo rješenje za sustav inteligentnog upravljanja prometnim raskrižjima zasnovano je detekciji i prebrojavanju vozila na trakama prometnog raskrižja koristeći računalni vid. To rješenje zahtijeva kontinuirano snimanje raskrižja te je osmišljeno da analizira digitalnu jednu po jednu sliku (okvir) video signala kamere. Pomoću informacija prikupljenih na taj način, sustav prilagodljivo upravlja semaforskim ciklusima na prometnom raskrižju. Razvijanje predloženog rješenja zasnovano je na CARLA simulatoru i implementaciji višestrukih klijenata koji međusobno komuniciraju kroz razmjenu podataka putem tekstualnih datoteka. Sustav se sastoji od jednog CARLA servera i tri klijenta, pri čemu svaki klijent ima specifičnu funkciju unutar sustava. Ovaj sustav omogućuje upravljanje prometom u realnom vremenu, prikupljanje podataka o prometu putem kamera, analizu prometa, prilagodbu semaforskih ciklusa na temelju aktualnog stanja u prometu te evaluaciju učinkovitosti upravljanja prometom putem razvijenog algoritma. Blok dijagram arhitekture predloženog rješenja je prikazan na slici 3.1.

U središtu arhitekture nalazi se CARLA server koji predstavlja temeljni dio sustava i omogućuje pokretanje simulacije prometa na raskrižju. CARLA server pruža virtualno okruženje za simulaciju urbanog prostora, stvaranje prometnih situacija te kontrolu nad sudionicima u prometu (vozila, pješaci, biciklisti). Server omogućava klijentima komunikaciju s virtualnim svijetom, uključujući

manipulaciju vozilima, kamerama i semaforima. Server je odgovoran za izvršavanje simulacije i pružanje podataka klijentima. Komunikacija između servera i klijenata odvija se putem specifičnih API poziva, a svaki klijent šalje i prima podatke o sudionicima u prometu na temelju kojih izvršava svoje zadatke. Server centralizira informacije i omogućuje suradnju više klijenata unutar jednog simulacijskog okruženja.



Slika 3.1. *Blok dijagram arhitekture inteligentnog sustava za upravljanje semaforima na prometnom raskrižju temeljeno na računalnom vidu*

CARLA klijent 1 preuzima ključnu ulogu u upravljanju prometom na raskrižju te služi kao glavni modul za prikupljanje i obradu podataka s RGB kamera. Njegove glavne funkcije uključuju uspostavljanje i kontrolu prometnih sudionika na raskrižju, što podrazumijeva generiranje vozila, postavljanje uvjeta prometa i sinkronizaciju sudionika u simulaciji. Klijent također postavlja kamere na raskrižju koje su pozicionirane tako da pokrivaju sve prometne trake, omogućujući prikupljanje slika u realnom vremenu. Nadalje, taj modul obrađuje slike dobivene s kamera koristeći tehnologije računalnog vida. U tom procesu detektira vozila na svakoj traci i prebrojava ih, pri čemu se koristi

napredna obrada slike za prepoznavanje objekata. Dodatno, ovaj modul uz pomoć modula za praćenje objekata u uzastopnim slikama omogućava da se svaki detektirani objekt prati na temelju njihovih centara. Glavna uloga tog modula je odrediti radi li se o istom objektu ili o novom detektiranom objektu između dvije uzastopne slike koje obrađuje detektor objekata. Nakon obrade slika i prikupljanja podataka o broju vozila po trakama, Klijent 1 stavlja te podatake na raspolaganje putem *File Managera*, a koje modul Klijent 2 čita po potrebi. Ovaj klijent predstavlja temeljni dio sustava, jer omogućuje detekciju i prebrojavanje vozila, što je ključno za prilagodljivo upravljanje semaforima.

Razmjena podataka (*File Manager*) služi kao središnji kanal za komunikaciju između Klijenta 1 i Klijenta 2. Klijent 1 nakon obrade slika pohranjuje podatke o broju vozila na prometnim trakama u tekstualne datoteke koje se potom koriste za prijenos podataka prema Klijentu 2. Ova arhitektura omogućuje fleksibilnu i jednostavnu razmjenu informacija između modula sustava. Tekstualne datoteke sadrže ključne informacije o broju vozila po trakama koje Klijent 2 koristi za prilagodljivo upravljanje semaforima. Ova razmjena podataka omogućuje sinkronizaciju između modula i osigurava da sustav ima sve potrebne podatke za učinkovito upravljanje prometom.

CARLA klijent 2 preuzima podatke o broju vozila koje Klijent 1 stavlja na raspolaganje te na temelju tih podataka upravlja radom semafora na raskrižju. Njegova glavna funkcija uključuje prilagodljivo upravljanje semaforima gdje Klijent 2 prilagođava način rada semafora na raskrižju ovisno o broju vozila na svakoj prometnoj traci. Na temelju podataka dobivenih od Klijenta 1, Klijent 2 izračunava optimalno vrijeme za svaku fazu semafora, čime se osigurava efikasniji protok prometa kroz raskrižje. Osim toga, Klijent 2 osigurava da se semafori sinkroniziraju prema prometnim uvjetima, smanjujući vrijeme čekanja i povećavajući protočnost prometa. Klijent 2 ima ključnu ulogu u optimizaciji prometa na raskrižju jer prilagođava rad semafora u skladu s promjenjivim prometnim uvjetima.

CARLA klijent 3 služi za evaluaciju učinkovitosti upravljanja prometom. Njegova funkcija je prikupljanje podataka o lokaciji vozila na raskrižju te analiza učinkovitosti prilagodljivog upravljanja semaforima. Klijent 3 omogućuje testiranje i usporedbu rezultata algoritma upravljanja semaforima u odnosu na tradicionalnu metodu upravljanja semaforima na raskrižju. Evaluacija sustava obavlja se na temelju podataka o vremenu čekanja vozila, broju vozila na raskrižju te ukupnom vremenu koje vozila u prosjeku provedu na raskrižju prije prolaska. Ovi podaci omogućuju procjenu koliko je sustav učinkovit u smanjenju gužvi i poboljšanju protočnosti prometa na raskrižju.

Modul za praćenje objekata u uzastopnim slikama (*Tracker*) je jednostavan, ali esencijalan modul za korištenje računalnog vida u okviru ovog rada. Ovaj modul se poziva prilikom svake obrade slike kako bi izračunao i zabilježio koordinate centra svakog detektiranog objekta u slici te provjerava jesu li ti objekti već detektirani usporedbom udaljenosti s trenutno pohranjenim centrima. U slučaju da je objekt već poznat, odnosno da je prvobitno bio detektiran u nekoj ranijoj slici, ažuriraju se koordinate njegovog centra i dodaje se u listu poznatih objekata. Ako je objekt tek detektiran, tada mu se dodjeljuje nova oznaka i dodaje ga se u listu poznatih objekata. Na kraju modula se, temeljeno na objektima detektiranim u trenutnom okviru, briše lista centara kako bi se uklonile oznake koji više nisu u upotrebi [20].

3.3. Odabir prikladnog detektora vozila

Detekcija objekata jedna je od ključnih komponenti u području računalnog vida i zbog toga ima široku primjenu u mnogim industrijama, uključujući autonomna vozila, nadzor, medicinsku dijagnostiku i robotsku navigaciju. Proces detekcije objekata uključuje identifikaciju i lokalizaciju objekata unutar slike ili video zapisa [1]. S razvojem dubokog učenja, metode detekcije objekata značajno su unaprijeđene, omogućujući visoku točnost i brzu obradu u stvarnom vremenu.

Za zadatak detekcije vozila iz slike dobivene putem RGB kamera u stvarnom vremenu u ovom radu odabran je YOLO (*You Only Look Once*) detektor objekata zbog svoje brzine i jednostavnosti implementacije te vrlo visoke točnosti i preciznosti. YOLO detektori objekata su revolucionarni u području računalnog vida zbog svoje sposobnosti da u stvarnom vremenu detektiraju objekte s visokim stupnjem točnosti [21]. Oni obrađuju cijelu sliku tijekom faza treniranja i detekcije, dajući predviđanja s jednim prolazom kroz mrežu. To znači da neuronska mreža procesira cijelu sliku odjednom i paralelno predviđa klase objekata i njihove koordinate unutar slike. Ovo se značajno razlikuje od klasičnih detektora objekata, poput R-CNN (*Region-based Convolutional Neural Networks*) koji prvo analiziraju dijelove slike pomoću selektivnog pretraživanja, a zatim za svaki dio posebno klasificiraju objekte. Takav proces zahtjeva više prolaza kroz mrežu što čini takav detektor objekata znatno sporijim od YOLO detektora.

YOLO detektori trenirani su na velikim skupovima podataka koji sadrže slike s raznim objektima, gdje su objekti unaprijed označeni pomoću graničnih pravokutnika (engl. *bounding boxes*) i pripadajućih klasa. Ovi skupovi podataka koriste se za učenje modela kako prepoznavati objekte u

različitim kontekstima, skalama i uvjetima osvjetljenja. Skup podataka na kojem su trenirani YOLO detektori su *COCO* (*Common Objects in Context*). COCO je skup podataka koji sadrži više od 300.000 slika s oko 80 različitih klasa objekata (npr. ljudi, vozila, životinje, namještaj). Svaka slika sadrži detaljne oznake objekata u njihovim prirodnim kontekstima, što pomaže u treniranju modela za prepoznavanje objekata u stvarnom svijetu. Kako su se verzije YOLO detektora razvijale, napravljena su poboljšanja u točnosti, brzini i sposobnosti detekcije malih objekata u raznim složenim okruženjima.

Osnovne verzije YOLO detektora su numerirane po generacijama od prvog YOLOv1 do trenutno najnovijeg YOLOv8 modela uz dodatne podverzije koje balansiraju između točnosti i brzine rada, ovisno o potrebama sustava.

Proces odabira odgovarajućeg modela za detekciju objekata u ovom radu temelji se na nekoliko ključnih faktora: točnosti detekcije, brzini obrade, prilagodljivosti stvarnim uvjetima prometa i jednostavnosti implementacije. Kako bi se postigli najbolji rezultati u optimizaciji upravljanja prometnim raskrižjima, analizirani su različiti YOLO modeli i uspoređeni na temelju ovih kriterija.

Konkretni podaci koji su bili uspoređivani prilikom odabira optimalne verzije YOLO detektora su prosječna preciznost (engl. *Average Precision - AP*), odziv (engl. *Recall*), prosječna preciznost na 50% presjeka (engl. *mean Average Precision at 50% Intersection over Union – mAP50*), prosjek AP na rasponu od 50% do 95% presjeka (*mAP50-95*) te sekunde po iteraciji.

Prije detaljnog objašnjenja svake korištene metrike za uspoređivanje verzija detektora, potrebno je objasniti terminologiju osnovnih parametara i koncepata koji se koriste za izračun tih metrika. Prvi od njih je *Intersection over Union* (IoU). To je metrički pokazatelj koji se koristi za kvantificiranje preciznosti predviđenih granica graničnih pravokutnika u odnosu na stvarne granice graničnih pravokutnika. IoU se izračunava kao omjer površine preklapanja između predviđenog graničnog pravokutnika i stvarnog graničnog pravokutnika prema ukupnoj površini koju obuhvaćaju oba granična pravokutnika. Drugo je izraz koji objedinjuje sve ispravno detektirane objekte u danoj slici, odnosno *True positives* (TP). Preciznije rečeno, to je slučaj kada detektor pronađe objekt u slici koji doista postoji i ispravno ga klasificira. Treći izraz koji ide u paru s TP je *False positives* (FP) koji predstavlja slučaj kada detektor prepozna određeni objekt na slici, ali objekt zapravo ne postoji u slici ili je pogrešno klasificiran. Treći izraz je takozvani *False negative* (FN) koji predstavlja slučaj kada detektor nije uspio prepoznati objekt koji postoji u slici.

Preciznost je metrički pokazatelj koji se koristi za razlikovanje ispravnih detekcija od pogrešnih. Definira se kao omjer TP i zbroja TP i FP. Visoka preciznost ukazuje na to da detektor ima nizak broj FP, što znači da je većina detekcija točna.

Odziv je metrički pokazatelj koji se koristi za mjerenje sposobnosti modela da pronađe sve relevantne instance objekata u skupu podataka. Definira se kao omjer TP i zbroja TP i FN. Visok odziv znači da model uspijeva detektirati većinu stvarnih objekata, ali može imati više lažnih pozitivnih rezultata. U kombinaciji s preciznošću, odziv pomaže u procjeni ukupne učinkovitosti detektora objekata [22].

Preciznost-odziv krivulja (engl. *Precision-Recall curve*) je korisna mjera uspješnosti klasifikacije, posebno u situacijama kada su klase vrlo neuravnotežene [23]. Preciznost-odziv krivulja prikazuje kompromis između preciznosti i odziva pri različitim pragovima detekcije. Visoko područje ispod krivulje znači da model postiže i visoku preciznost i visoki odziv. Visoka preciznost znači da detektor ima vrlo malo FP, dok visoki odziv znači da je model uspio detektirati većinu stvarnih objekata. Ako model ima visok odziv, ali nisku preciznost, to znači da vraća većinu relevantnih objekata, ali s mnogo FP. S druge strane, model s visokom preciznošću, ali niskim odzivom, detektira vrlo malo objekata, ali su gotovo svi ti objekti točno detektirani. Primjer sustava s visokim odzivom, ali niskom preciznošću bio bi sustav koji prepoznaje gotovo sve objekte u slici, ali s mnogo FP. Suprotno tome, sustav s visokom preciznošću, ali niskim odzivom prepoznaje vrlo malo objekata, ali gotovo svi predviđeni objekti su točni. Idealni sustav bi postigao i visoku preciznost i visok odziv, prepoznavajući većinu relevantnih objekata s vrlo malo pogrešnih predikcija.

Prosječna preciznost objedinjuje preciznost-odziv krivulju u jednu vrijednost. AP se računa kao površina ispod preciznost-odziv krivulje. Na taj način, prosječna preciznost integrira cijelu preciznost-odziv krivulju čime se dobiva jedinstvena mjera koja ocjenjuje ukupnu učinkovitost detektora.

mAP50 je metrički pokazatelj koji se koristi za mjerenje točnosti modela u detekciji objekata. Prosječna preciznost je prosjek preciznosti za različite razine odziva. mAP je prosjek AP vrijednosti za sve klase u skupu podataka. Oznaka 50 definira prag, odnosno potrebni iznos IoU između predviđenog i stvarnog objekta kako bi se detekcija smatrala ispravnom. mAP50 je važan pokazatelj performansi modela jer daje uvid u sposobnost modela da precizno locira i klasificira objekte unutar slike ili videozapisa. Viša vrijednost mAP50 ukazuje na bolje performanse detektora [24].

mAP50-95 je sveobuhvatniji metrički pokazatelj koji bolje ocjenjuje performanse modela u odnosu na mAP50 jer ocjenjuje točnost modela na širem rasponu IoU pragova. Predviđeni granični pravokutnik se procjenjuje na različitim pragovima IoU (npr. 50%, 55%, 60%, ..., 95%) u koracima od 5%. Za svaku klasu i za svaki prag IoU izračunava se AP. Prosječna vrijednost AP za sve pragove IoU i za sve klase daje mAP50-95. Viša vrijednost mAP50-95 ukazuje na bolje performanse detektora, a ovaj pokazatelj se često koristi za usporedbu performansi različitih modela i njihovih verzija.

Sekunde po iteraciji (engl. *seconds per iteration*) je metrički pokazatelj mjeri prosječno vrijeme koje je modelu potrebno za obradu jedne iteracije tijekom treniranja ili inferencije. *s/it* je ključan pokazatelj koliko brzo model može proći kroz jednu iteraciju treniranja. Niža vrijednost znači da se model trenira brže, što je korisno za razvojne cikluse i optimizaciju. Kada se koristi za inferenciju (predikciju), *s/it* pokazuje koliko brzo model može obraditi jednu sliku, što je važno za aplikacije u stvarnom vremenu kao što su autonomna vozila ili video nadzor. Niže vrijednosti *s/it* su poželjne jer ukazuju na bržu obradu, bilo tijekom treniranja ili inferencije, čime se omogućava efikasnija primjena modela u stvarnim scenarijima.

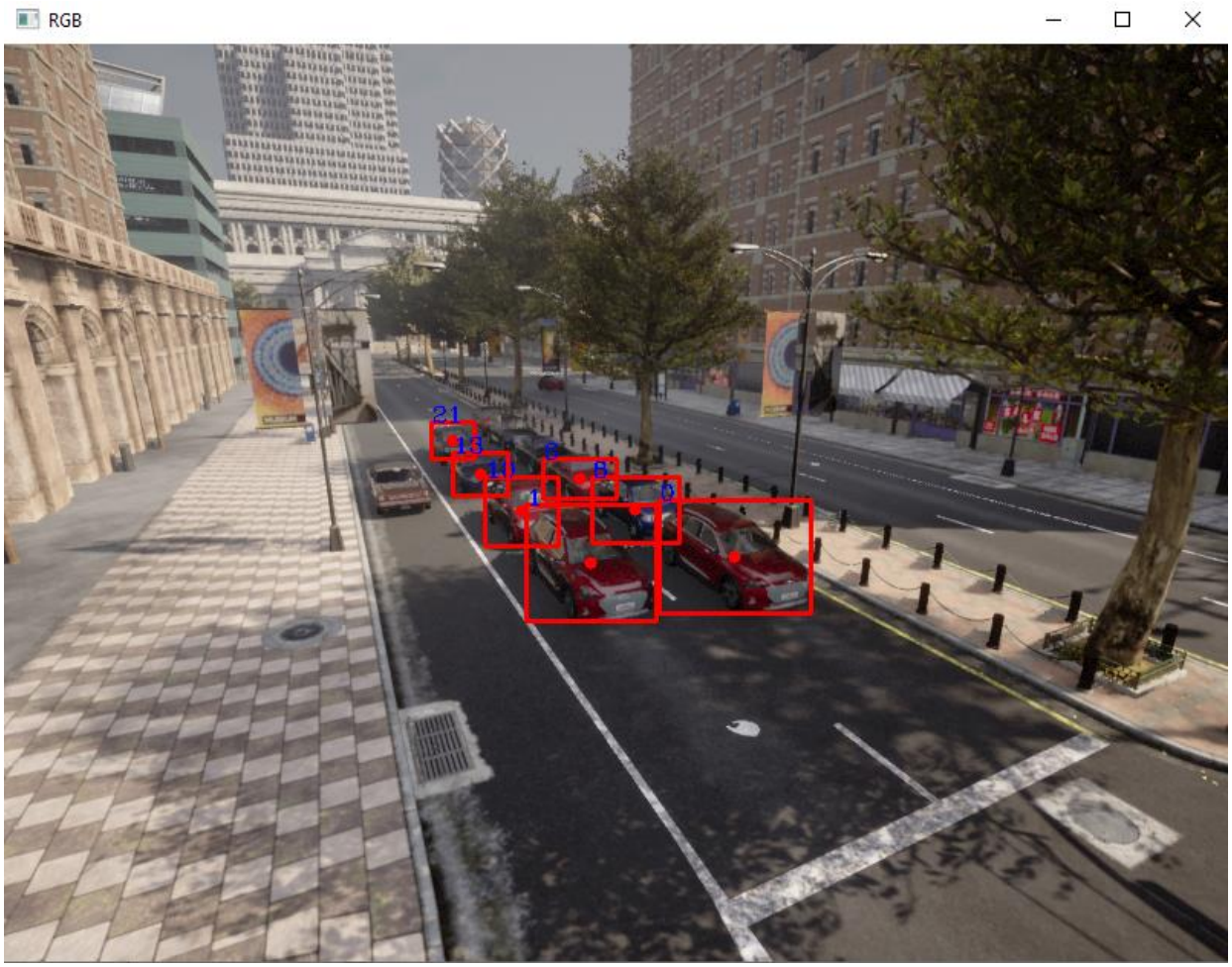
Na temelju testiranja i implementacije suvremenijih YOLO detektora, za konačnu usporedbu su izabrane verzije YOLOv5n, YOLOv5s, YOLOv8s, YOLOv8n s obzirom na svoja slična svojstva i jednostavnost implementacije. Te 4 verzije detektora objekata su dodatno istrenirane i evaluirane na temelju ručno označenih slika iz simulatora pomoću *LabelMe* alata [25] te su potom izlazne *JSON* datoteke bile parsirane i konvertirane u *.txt* datoteke koje su kompatibilne s tzv. *yolo* formatom. To je, unatoč tome što je YOLO detektor već unaprijed istreniran na slikama iz stvarnog svijeta, napravljeno kako bi detektor objekata bio što učinkovitiji za primjenu u ovom radu koji je isključivo implementiran u simulacijsko okruženje. Za potrebe dodatnog treniranja i evaluacije, iz CARLA simulatora prikupljeno je ukupno 190 različitih slika s različitim gustoćama prometa kako bi se obuhvatile sve moguće situacije s kojima se detektor može susresti u ovom radu. 140 od tih slika korišteno je za dodatno treniranje detektora, dok je 50 slika korišteno za evaluaciju. Prilikom označavanja slika pomoću spomenutog *LabelMe* alata, korištena je samo jedna klasa koja je označavala vozilo s obzirom da sustav za prepoznavanje i prebrojavanje vozila nije zadužen da raspoznaje različite tipove vozila na prometnicama, već samo da ih može prepoznati. Prikaz ručno označenih graničnih pravokutnika oko vozila na jednoj od slika prikupljenih iz CARLA simulatora prikazan na slici 3.2.

Slika 3.2. *Prikaz ručno označenih graničnih pravokutnika oko vozila na jednoj od slika prikupljenih iz CARLA simulatora*

Pregledom i usporedbom dobivenih rezultata evaluacije odabranih verzija YOLO detektora objekata, istaknuto je kako nema jedinstvenog rješenja za odabir prikladne verzije, odnosno potrebno je odabrati najbolji omjer brzine obrade slike i prosječne preciznosti kako bi se postiglo optimalno rješenje. U tablici 3.1. vidljivo je kako je YOLOv5n verzija uvjerljivo najbrža uz najlošije rezultate prosječne preciznosti, YOLOv8s verzija ima najveću prosječnu preciznost od svih odabranih modela, ali je uz to i najsporiji te je YOLOv8n drugi po brzini i drugi po prosječnoj preciznosti od 4 odabrane verzije. S obzirom na to da su i brzina i prosječna preciznost od velike važnosti za sustav detekcije vozila na raskrižju, odabrana je YOLOv8n verzija za implementaciju modula za detekciju vozila u prometnoj traci. Na slici 3.3. prikazan je primjer detekcije vozila u prometnoj traci pomoću implementiranog YOLOv8n detektora objekata. Prikaz slike s kamere i iscrtavanje graničnih pravokutnika je omogućen kroz *OpenCV* biblioteku. Na danom primjeru je vidljivo kako je većina vozila pravilno detektirana i označena crvenim graničnim pravokutnicima.

Tablica 3.1. *Prikaz i usporedba rezultata evaluacije odabranih YOLO verzija na vlastito prikupljenom podatkovnom skupu*

Model	Preciznost	Odziv	mAP50	mAP50-95	s/it
YOLOv5s	**0.941**	**0.903**	**0.955**	**0.611**	**4.4**
YOLOv5n	**0.943**	**0.900**	**0.944**	**0.586**	**2.35**
YOLOv8s	**0.945**	**0.932**	**0.976**	**0.694**	**5.71**
YOLOv8n	**0.974**	**0.922**	**0.975**	**0.677**	**3.11**

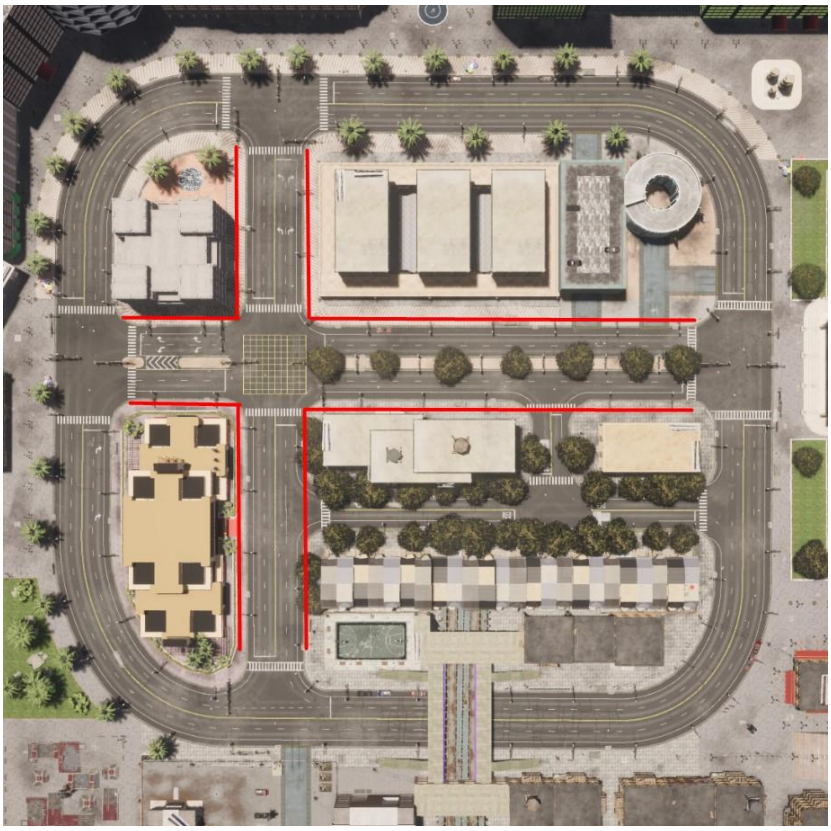


Slika 3.3. *Primjer detekcije vozila u prometnoj traci u slici dobivenoj s RGB kamere pomoću YOLOv8n detektora objekata u CARLA simulatoru*

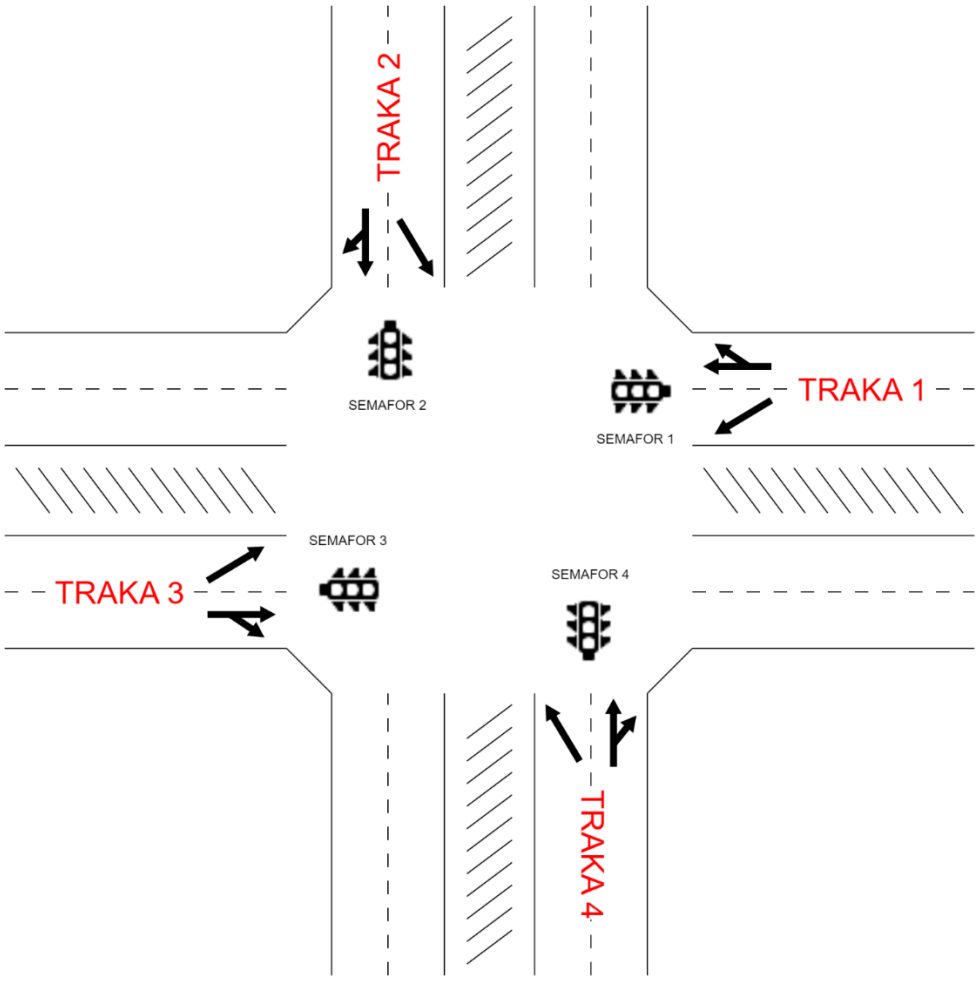
3.4. Simulacija prometnog raskrižja u okviru CARLA simulatora i integracija detektora vozila u CARLA okruženje

3.4.1. Analiza odabranog prometnog raskrižja i pripadajućih semafora

CARLA simulator dolazi s velikim izborom različitih gradova koji se mogu koristiti kao prostor za simulaciju. Za potrebe ovog rada, odabran je grad pod nazivom *Town10* za implementaciju i testiranje sustava za upravljanje semaforima na temelju računalnog vida. Prikaz odabranog grada iz ptičje perspektive s označenim odabranim raskrižjem je prikazan na slici 3.4. Skica odabranog raskrižja s označenim prilaznim trakama i ucrtanim semaforima i mogućim smjerovima kretanja je prikazana na slici 3.5. Vidljivo je da se raskrižje sastoji od 4 prilazne trake (Traka 1 – Traka 4) gdje se na svakoj traci vozila mogu grupirati u dvije kolone. Svaka od prometnih traka, neovisno o smjeru kretanja, ima samo po jedan semafor koji upravlja prometom cijele trake istovremeno.



Slika 3.4. *Prikaz grada Town10 iz ptičje perspektive s označenim korištenim raskrižjem [26]*



Slika 3.5. *Skica odabranog prometnog raskrižja unutar simulacijskog okruženja s označenim prometnim trakama i semaforima*

Semafori su, u početnim postavkama ovog raskrižja unutar CARLA simulatora, postavljeni na svoje osnovne periode. Taj način rada semafora se u daljnjem tekstu naziva tradicionalnim načinom rada, odnosno trajanje zelenog svjetla na svakom od semafora u tom načinu rada se naziva osnovnim periodom trajanja zelenog svjetla. Jedan semaforski ciklus dizajniran je tako da u bilo kojem trenutnku samo jedan semafor može biti postavljen na zeleno ili žuto svjetlo, dok su svi ostali semafori u istom trenutku postavljeni na crveno svjetlo. Tim principom se tijekom jednog ciklusa sukcesivno mijenjaju sva stanja na svim semaforima na raskrižju. Inicijalno se sva svjetla postave na crveno na fiksni period te se nakon toga uključuje zeleno svjetlo na fiksni period za aktivni semafor (semafor koji je idući na redu), dok su sva ostala svjetla i dalje crvena. Pri isteku fiksnog perioda za zeleno svjetlo se uključuje žuto svjetlo na semaforu na kojem je do tada bilo uključeno zeleno svjetlo te se po isteku fiksnog

perioda za žuto svjetlo uključuje crveno svjetlo na tom semaforu. Tada se ciklus ponovno vraća u fazu gdje su sva svjetla inicijalno postavljena na crveno svjetlo te se prelazi na upravljanje semaforom koji je idući na redu u kružnom ciklusu. Kronološki raspored semaforskih faza raspoređenih po semaforima na raskrižju u CARLA simulatoru za vrijeme tradicionalnog načina upravljanja semaforima prikazan je tablicom 3.2. U svakom stupcu napisano je trajanje te faze u sekundama te konkretno stanje za svaki od semafora u raskrižju u tom periodu (zeleno, žuto ili crveno svjetlo). Ukupno vrijeme trajanja jednog takvog kružnog semaforskog ciklusa je 43 sekunde.

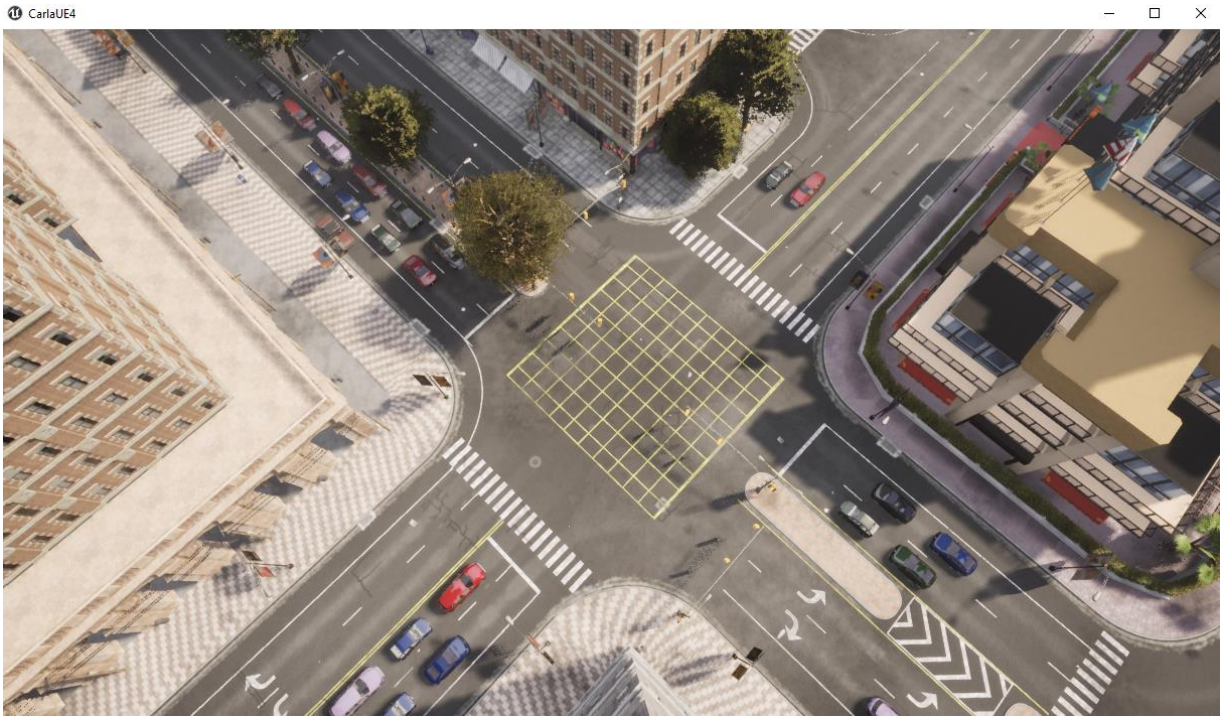
Tablica 3.2. *Kronološki raspored semaforskih faza raspoređenih po semaforima na odabranom raskrižju u CARLA simulatoru za vrijeme tradicionalnog načina upravljanja semaforima*

Trajanje faze	2,7s	5s	3,05s	2,7s	5s	3,05s	2,7s	5s	3,05s	2,7s	5s	3,05s
Semafor 1												
Semafor 2												
Semafor 3												
Semafor 4												

3.4.2. Postavljanje scenarija prometne gužve u CARLA simulatoru

Prvi korak u ovom procesu bio je kreiranje odgovarajućeg scenarija unutar CARLA simulatora koji bi što vjernije replicirao stvarne uvjete prometne gužve na raskrižju. Nakon pažljivog odabira odgovarajućeg grada unutar CARLA simulatora koji će se koristiti kao simulacijska okolina i odabira specifičnog raskrižja na kojem će se provoditi testiranje, bilo je potrebno identificirati sve moguće točke na mapi gdje se vozila mogu generirati. Za stvaranje prometa i upravljanje prometom unutar simulacije zadužen je *Python* modul označen kao Klijent 1 u potpoglavlju 3.2. Ovaj modul ima ključnu ulogu u komunikaciji sa serverom kako bi se prikupile i prikazale sve relevantne koordinate na mapi, omogućavajući generiranje vozila na točno određenim lokacijama unutar simulacije. Količina vozila stvorenih u simulaciji je određena programskim kodom ovisno o potrebama za testiranje specifične situacije. Na taj način je moguće efikasno testirati učinikovitost predloženog sustava. U svrhu što vjerodostojnije simulacije stvarnih prometnih uvjeta, osim vozila, navedeni modul također nasumično

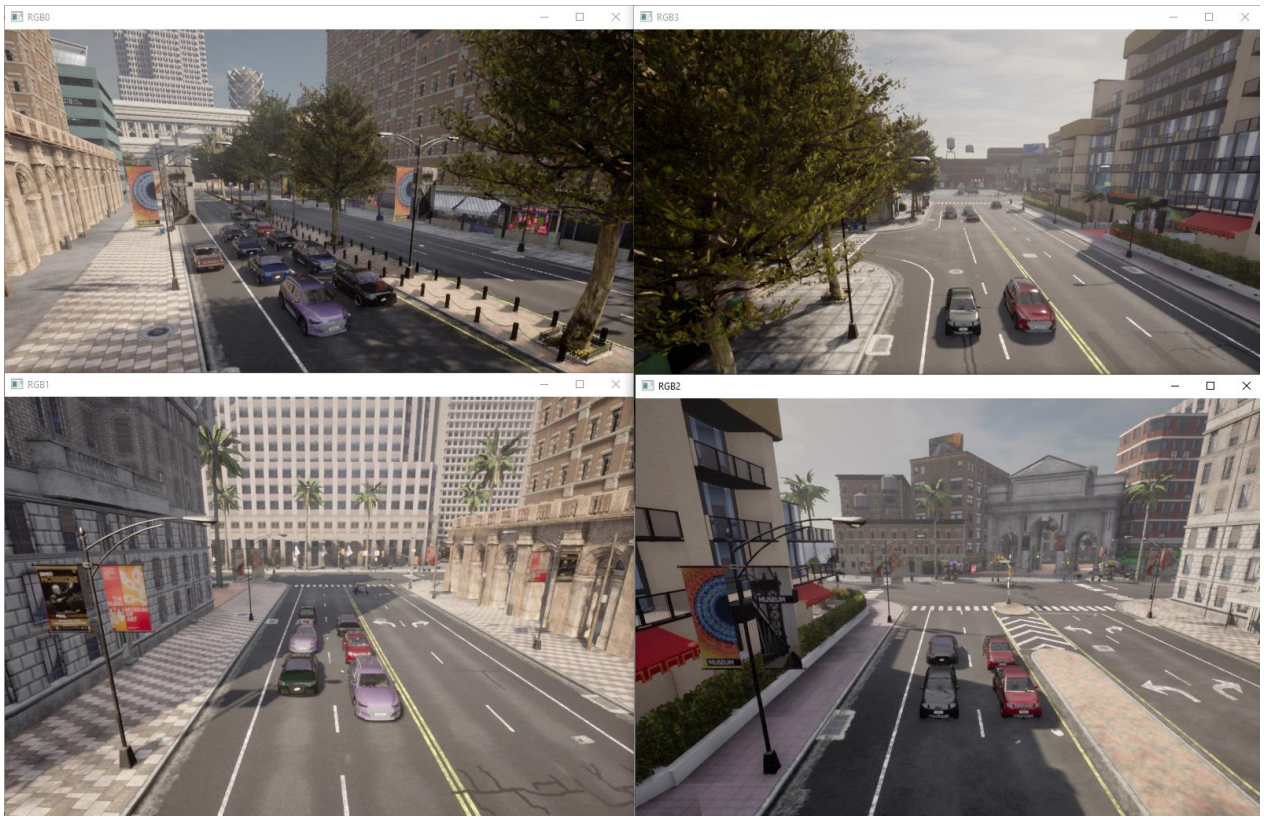
dodaje sporedne sudionike u prometu, poput pješaka i biciklista. Ovi dodatni sudionici pridonose složenosti simulacije, stvarajući izazovnije uvjete za detekciju i praćenje unutar sustava. Unutar tog modula su, u svrhu stvaranja prometa, prvo učitani svi 3D modeli vozila dostupni u bibiloteci vozila unutar CARLA simulatora. Nakon toga su ručno određene koordinate na kojima će biti stvorena vozila. Moguće koordinate stvaranja vozila su unaprijed određene kako ne bi dolazilo do kolizije prilikom stvaranja prometa. Nakon stvaranja, sva vozila su postavljena na autopilot kako bi sva vozila paralelno i samostalno slijedila prometna pravila kao što su poštivanje stanja na semaforima, praćenje ograničenja brzine i izbjegavanje sudara. U prilogu P.3.1. prikazan je dio programskog koda zadužen za određivanje koordinata na kojima će biti stvorena vozila te dio koda koji obavlja stvaranje vozila i postavljanje osnovnih postavki za sva stvorena vozila. Primjer stvorene prometne gužve unutar simulacije na promatranom raskrižju vidljiv je na slici 3.6.



Slika 3.6. *Prikaz odabranog raskrižja s vidljivom prometnom gužvom*

Nakon odabira odgovarajućeg raskrižja i stvaranja željenog broja vozila na raskrižju, sljedeći korak bio je pažljivo postavljanje RGB kamera unutar simulacijskog prostora u sklopu modula Klijent 1. RGB kamera je jedan od mnogih dostupnih senzora unutar biblioteke senzora koje nudi CARLA simulator. Osnovna rezolucija RGB kamera unutar CARLA simulatora postavljena je na 800x600

rezoluciju. Kamere su strateški pozicionirane kako bi snimale situaciju na svakoj od četiri prometne trake unutar raskrižja, što omogućuje sveobuhvatan nadzor prometa u stvarnom vremenu. Ove kamere igraju ključnu ulogu u sustavu jer omogućuju kontinuirano praćenje situacije na raskrižju i osiguravaju precizne podatke za daljnju obradu. U prilogu P.3.2. prikazan je dio programskog koda zadužen za određivanje koordinata za stvaranje svake kamere te za stvaranje kamera unutar simulacije. Svaka slika zabilježena kamerom prosljeđuje se na daljnju obradu putem YOLO detektora, koji izvršava detekciju vozila u svakoj slici zasebno. Na temelju dobivenih detekcija provodi se određivanje broja vozila u traci za analiziranu RGB sliku. Posebna pažnja posvećena je izboru lokacija za postavljanje kamera u CARLA simulatoru. Iako simulator omogućava potpunu slobodu u postavljanju kamera, odlučeno je da se kamere postave na visine i pozicije koje oponašaju stvarne uvjete, poput visine semafora. Ova strategija osigurala je da sustav bude što bliži stvarnom prometnom sustavu, čime je omogućeno da rezultati simulacije budu relevantni i primjenjivi u stvarnom svijetu. Prikaz slika s 4 stvorene *RGB* kamere koje snimaju svaku traku zasebno vidljiv je na slici 3.7.



Slika 3.7. *Prikaz slika u stvarnom vremenu s 4 RGB kamere na raskrižju*

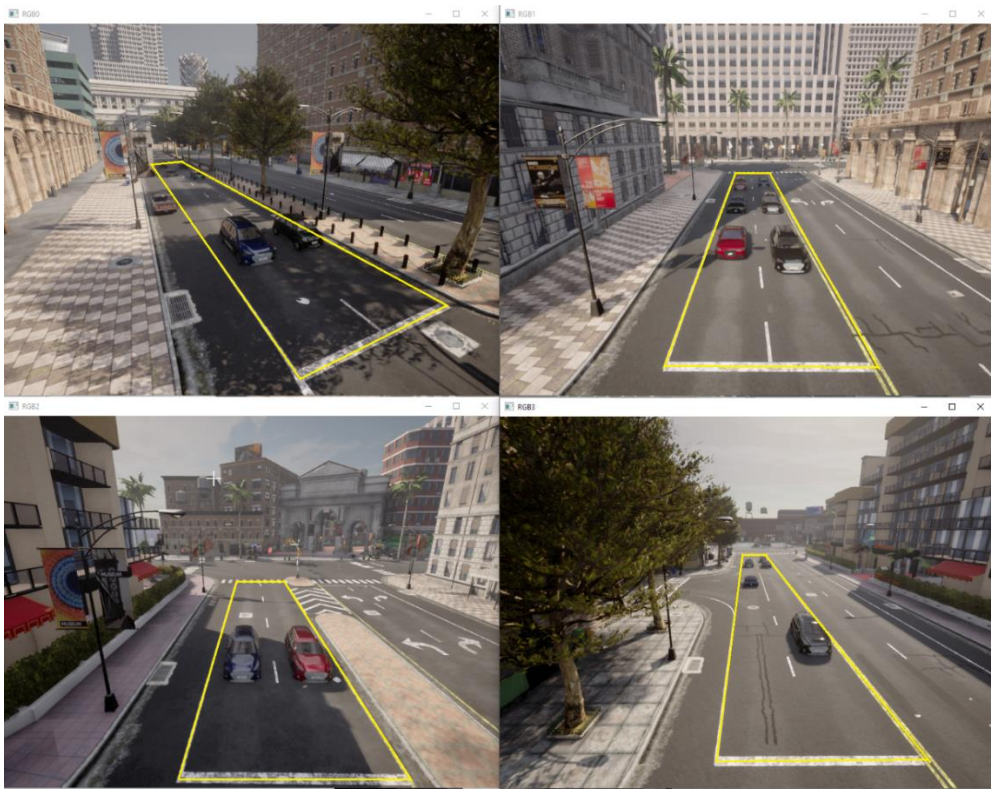
3.4.3. Implementacija detekcije vozila u okviru CARLA simulatora

Najzahtjevniji aspekt ovog sustava s hardverske strane je procesiranje svake slike i detekcija vozila u stvarnom vremenu. Slike s kamera se obrađuju pomoću funkcije unutar modula Klijent 1 prikazane u prilogu P.3.3. Ova funkcija je ključna za cijeli sustav jer omogućava praćenje količine prometa na raskrižju unutar *CARLA* simulatora pomoću RGB kamera i YOLO modela. U ovoj točki detaljno je opisan način rada funkcije koja je odgovorna za obradu slike i detekciju vozila.

Prvi korak, prije samog izvršavanja funkcije, uključuje učitavanje klase objekata koje YOLO model može prepoznati. Ovaj korak se ostvaruje otvaranjem datoteke *"coco.txt",* koja sadrži nazive svih klasa koje model može detektirati. Nazivi klasa se učitavaju u listu koja kasnije služi za povezivanje detektiranih objekata s njihovim odgovarajućim nazivima. Nakon toga, YOLO detektor objekata se učitava koristeći unaprijed trenirani model *yolov8n.pt*, a također se inicijalizira i objekt *Tracker*, koji prati kretanje detektiranih objekata kroz niz uzastopnih slika. Uz to, postavlja se lista *vehicle_counts* za pohranu broja vozila detektiranih u slikama sa svake kamere.

Kada se funkcija pozove, prvi zadatak je konvertiranje sirovih podataka iz kamere u format slike pogodan za obradu putem YOLO detektora i prikaz u *cv2* prozoru. Slika se zatim transformira iz *BGRA* formata u *BGR* format, koji je standardan za *OpenCV* biblioteku. Slika se također prilagođava na dimenzije pogodne za obradu (832x608), čime se osigurava brzina i efikasnost daljnje obrade. Nakon toga, slika se pretvara u tenzor kompatibilan s *PyTorch* bibliotekom, što je nužno za izvođenje detekcije pomoću *YOLO* modela. Proces uključuje normalizaciju slike dijeljenjem svakog elementa slike s vrijednošću 255.

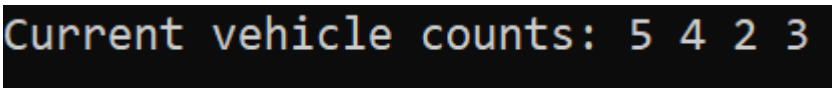
Sljedeći korak je definiranje područja od interesa (engl. *Region Of Intereset - ROI*) za svaku sliku koja se dobiva sa svake kamere, a podrazumijeva određivanje područja u slici koje odgovara području trake na prometnom raskrižju. Ovaj korak je od presudne važnosti, s obzirom na to da kamere snimaju cijelo prometno okruženje, uključujući i dijelove koji nisu relevantni za praćenje vozila. Definirane koordinate koriste se kasnije u procesu obrade kako bi se utvrdilo nalaze li se detektirana vozila unutar tih specifičnih područja. Kutovi poligona koji iscrtava i ograničava područje od interesa unutar *cv2* prozora određeni su pomoću koordinata elemenata slike. Ti kutovi predstavljaju kutove promatrane prometne trake unutar slike s RGB kamere. Primjer slika sa svih RGB kamera s iscrtanim područjima od interesa prikazane su na slici 3.8.



Slika 3.8. *Prikaz slika s RGB kamera koje snimaju trake raskrižja s iscrtanim područjima od interesa*

YOLO detektor objekata se koristi za predviđanje i detekciju objekata unutar slike. Rezultati detekcije uključuju koordinate pravokutnika koji okružuju svaki detektirani objekt, kao i *ID* klase koja odgovara prepoznatom objektu. Ovi podaci se pretvaraju u *pandas DataFrame* radi lakše manipulacije i daljnje obrade. Detektirani objekti se potom predaju u *Tracker*, koji prati njihovo kretanje kroz niz slika. Svakom vozilu dodjeljuje se jedinstveni *ID*, što omogućava točno brojanje vozila bez duplikata kroz više uzastopnih slika.

Na kraju, centar svakog detektiranog vozila se provjerava kako bi se utvrdilo nalazi li se unutar unaprijed definiranog područja interesa. Ako vozilo uđe u definiranu zonu, njegov *ID* se dodaje u skup, čime se osigurava da se svako vozilo broji samo jednom. Broj vozila detektiranih u svakoj traci pohranjuje se u odgovarajuću datoteku, što omogućava daljnju analizu i praćenje prometa. Osim toga, rezultati se prikazuju u konzoli, pružajući korisniku jasan uvid u trenutačno stanje prometa na raskrižju. Prikaz ispisa iz modula Klijent 1 u konzoli koji prikazuje brojeve vozila po trakama je prikazan na slici 3.9.



Slika 3.9. *Prikaz ispisa u konzoli koji prikazuje broj detektiranih vozila po trakama*

Na kraju funkcije, vraća se povratna vrijednost koja predstavlja jedan okvir obrađene slike, koji se potom prikazuje u prozoru putem *cv2* modula. Ovaj prikaz, kao što je ranije prikazano na slici 3.3., omogućava vizualnu verifikaciju detekcije vozila i nudi uvid u trenutnu situaciju na raskrižju.

Unutar CARLA simulatora, otkucaji (engl. *Ticks*) predstavljaju vremenske korake ili intervale unutar simulacijskog svijeta koji definiraju koliko često se simulacija ažurira. Svaki otkucaj označava prolazak jednog koraka simulacije tijekom kojeg se obavljaju svi procesi kao što su kretanje vozila, interakcija sudionika u prometu, prikupljanje podataka sa senzora, ažuriranje okoline i svi ostali procesi definirani u *Python* skripti koja je zadužena za upravljanje simulacijom. Svaki otkucaj u simulaciji definiran je pomoću *world.tick()* poziva funkcije koji se poziva na kraju petlje koja se izvodi unutar skripte. Otkucaji su direktno povezani s frekvencijom osvježavanja simulacije (engl. *Frames Per Second - FPS*). Veći broj otkucaja po sekundi znači bržu i fluidniju simulaciju, dok manji broj otkucaja po sekundi usporava simulaciju, ali može omogućiti detaljniju kontrolu i preciznije upravljanje senzorima i sudionicima prometa. Zbog visoke kompleksnosti obrade slike, detekcije vozila u stvarnom vremenu i upravljanja svim vozilima i sudionicima prometa, kao i zbog značajnih hardverskih zahtjeva potrebnih za pokretanje *CARLA* simulatora, cijela simulacija je morala biti usporena na približno 2 okvira u sekundi. Ova redukcija brzine simulacije bila je neophodna kako bi se osigurala ujednačena brzina rada simulacije i obrade podataka, omogućujući da se oba procesa odvijaju ravnomjernim tempom.

Ovakav princip simulacije omogućava detaljnu analizu i preciznu detekciju vozila, no predstavlja izazov u smislu stvarne primjene, jer je stvarna brzina prometa mnogo veća. Međutim, važno je napomenuti da ovaj problem ne bi bio prisutan u stvarnom okruženju kada bi se sustav implementirao na stvarno raskrižje jer bi tada svi računalni resursi bili usmjereni isključivo na obradu slike. U konačnici, iako je simulacija morala biti usporena, ovakav princip pruža vrijedne uvide i osigurala da sustav bude robustan, pouzdan i spreman za potencijalnu implementaciju u stvarnom prometnom okruženju.

Na kraju svake petlje koja se odvija unutar Klijent 1 modula se također pozivaju funkcije koje omogućuju stalno osluškivanje informacija s kamera putem metode *camera.listen*. Svaki put kada kamera uhvati novi okvir, podaci se šalju u prethodno opisanu funkciju za obradu slike i detekciju vozila. Ova funkcija, obrađujući svaki okvir, bilježi broj vozila u svakoj prometnoj traci. Broj vozila se zatim pohranjuje u zasebne *.txt* datoteke koje služe kao komunikacijski kanal između Klijenta 1 i Klijenta 2 kako je opisano u potpoglavlju 3.2. Na temelju tih podataka o broju vozila, modul Klijent 2 donosi odluke o prilagodbi semaforskih signala kako bi osigurao optimalan protok vozila kroz raskrižje.

3.5. Algoritam za prilagodljivo upravljanje semaforima

3.5.1. Scenariji prometnih situacija

S obzirom na to da su situacije na raskrižjima često vrlo dinamične i složene, bilo je nužno unaprijed osmisliti sve moguće scenarije koji se mogu dogoditi na raskrižju te ih objediniti u specifične slučajeve kojima program može učinkovito upravljati. Algoritam je konstruiran tako da prepoznaje trideset različitih slučajeva (engl. *CASE*), temeljenih na kombinacijama broja vozila u četiri prometne trake. Ovi slučajevi grupirani su u devet općih kategorija koje pokrivaju različite prometne situacije, od potpuno praznog raskrižja do stanja u kojem su sve trake preopterećene. Kao prag za specifično djelovanje na određenoj traci uzeta je granica od 4 vozila po traci jer je empirijski utvrđeno da taj broj vozila u prosjeku može proći unutar jednog osnovnog trajanja zelenog svjetla u CARLA simulatoru. Ta brojka može biti usklađena ovisno o parametrima raskrižja. Raspodjelu po grupama i opis navedenih slučajeva može se vidjeti u tablici 3.3.

Glavna prednost ovih opisanih slučajeva jest u tome što algoritam omogućuje dinamičko prilagođavanje u stvarnom vremenu što omogućuje da se prilikom svakog određivanja novog slučaja sustav brzo i učinkovito prilagodi trenutnim uvjetima na raskrižju. Na taj način sustav može optimalno upravljati prometom, smanjujući zastoje i poboljšavajući ukupnu protočnost prometa.

Tablica 3.3. *Raspodjela i opis predviđenih prometnih slučajeva na temelju broja vozila u trakama*

Grupa	Opis	Princip upravljanja
CASE 0	Sve trake u raskrižju imaju između 1 i 4 vozila.	Semafori rade u tradicionalnom načinu bez potrebe za intervencijom.
CASE 1-4	Tri trake imaju između 1 i 4 vozila, dok jedna traka ima 0 vozila.	Semafori rade u standardnom načinu uz preskakanje trake koja nema vozila.
CASE 5-10	Dvije trake imaju između 1 i 4 vozila, dok druge dvije imaju 0 vozila.	Semafori rade u standardnom ciklusu uz preskakanje traka koja nemaju vozila.
CASE 11-14	Jedna traka ima između 1 i 4 vozila, dok sve ostale imaju 0 vozila.	Uključeno zeleno svjetlo za traku s vozilima sve dok se ne pojavi vozilo na nekoj drugoj traci ili dok se trenutna traka ne isprazni.
CASE 15-18	Jedna traka ima više od 4 vozila, dok ostale trake imaju 4 ili manje vozila.	Algoritam daje prioritet traci s pojačanim prometom te uključuje zeleno svjetlo za tu traku na ponavljajući period. Ponavljajući period se završava kada se traka isprazni ili kada istekne dozvoljeno vrijeme čekanja u slučaju da ima vozila na drugim trakama.
CASE 19-24	Dvije trake imaju više od 4 vozila, dok ostale trake imaju 4 ili manje vozila.	Algoritam određuje prioritet između dvije trake s pojačanim prometom te uključuje zeleno svjetlo za tu traku na ponavljajući period. Ponavljajući period se završava kada se traka isprazni ili kada istekne dozvoljeno vrijeme čekanja.
CASE 25-28	Tri trake imaju više od 4 vozila, dok jedna traka ima 4 ili manje vozila.	Algoritam određuje prioritet između tri trake s pojačanim prometom te uključuje zeleno svjetlo za tu traku na ponavljajući period. Ponavljajući period se završava kada se traka isprazni ili kada istekne dozvoljeno vrijeme čekanja.
CASE 29	Sve trake imaju više od 4 vozila	Algoritam određuje prioritet između svih traka s pojačanim prometom te uključuje zeleno svjetlo za tu traku na ponavljajući period. Ponavljajući period se završava kada se traka isprazni ili kada istekne dozvoljeno vrijeme čekanja.
CASE 30	Sve trake imaju 0 vozila.	Svi semafori postavljeni na crveno svjetlo sve dok se ne pojavi vozilo u nekoj od traka

3.5.2. Računalni model razvijenog algoritma i način rada

Algoritam upravljanja semaforima unutar simulacije prometnog raskrižja strukturiran je kao stroj stanja (engl. *state machine*). Ovaj algoritam je sadržan u modulu Klijent 2. U ovom algoritmu

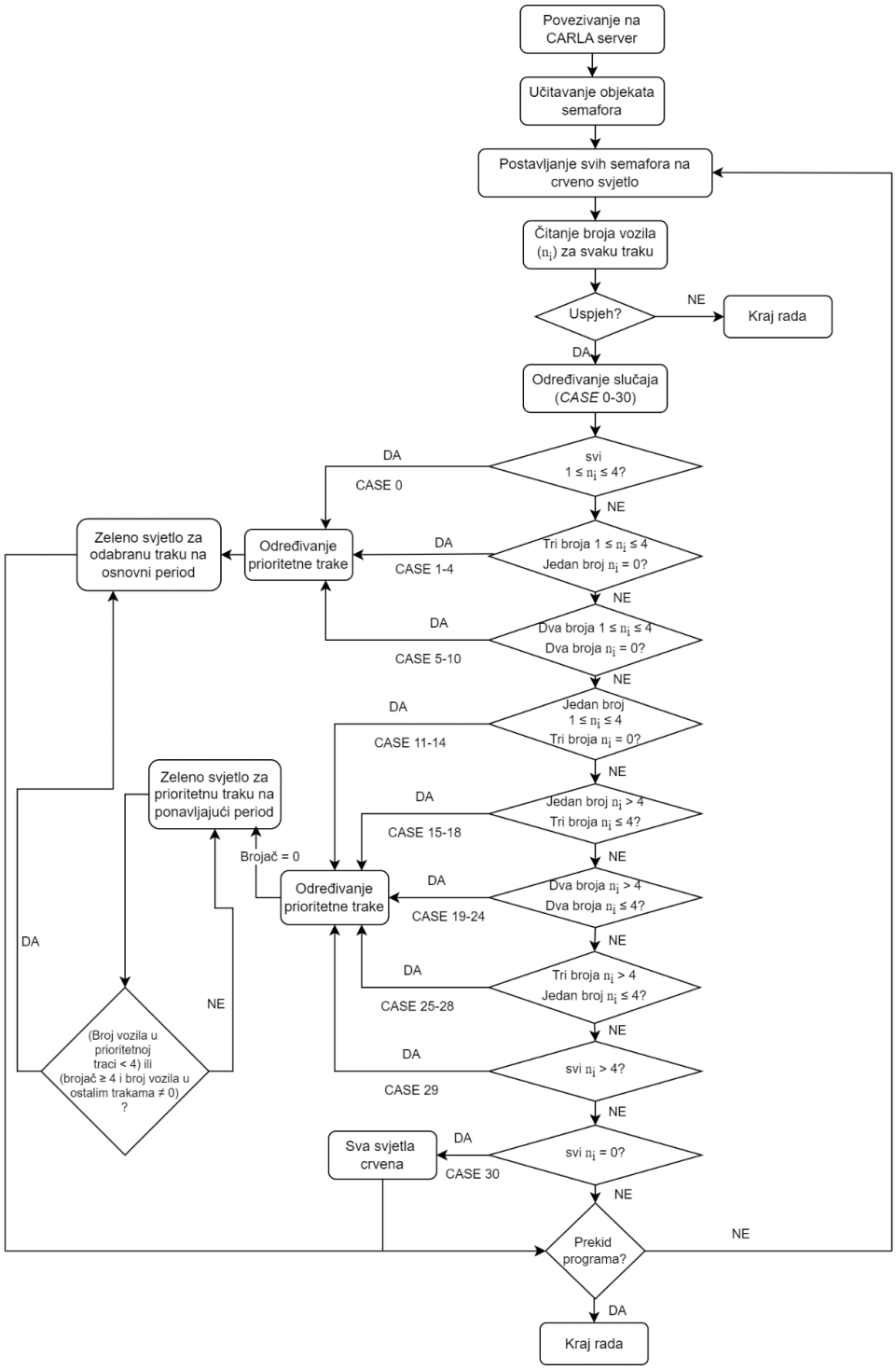
definirani su različiti načini upravljanja semaforima na temelju slučajeva određenih pomoću broja vozila u svakoj traci na prometnom raskrižju, prema tablici 3.3.. Algoritam se prilagođava prometnim uvjetima tako što kontinuirano očitava broj vozila u svakoj traci iz vanjskog izvora podataka te, na temelju tih informacija, prelazi u odgovarajuće stanje. Algoritam je napisan kao beskonačna petlja koja periodički donosi odluke o načinu upravljanja semaforima. Izlaz iz petlje moguć je samo u slučaju greške pri čitanju broja vozila u trakama ili u slučaju prekida programa od strane korisnika. Na ovaj način, algoritam djeluje kao deterministički stroj stanja, pri čemu su prijelazi između različitih stanja jasno definirani logikom uvjeta, osiguravajući da promet na raskrižju bude optimiziran u realnom vremenu. Dijagram toka predloženog algoritma za prilagodljivo upravljanje semaforima na prometnom raskrižju koji detaljno opisuje slijed događaja u algoritmu prikazan je na slici 3.10.

Modul se prvo povezuje na *CARLA* server čime uspostavlja vezu s virtualnim okruženjem. Potom prepoznaje i identificira semafore na temelju njihovih lokacija, nakon čega su svi semafori postavljeni u početno stanje, odnosno na svim semaforima je uključeno crveno svjetlo. To početno stanje simulira svako međustanje prije dodjeljivanja novog slučaja, odnosno nakon svakog odrađenog ciklusa, program se vraća u stanje gdje na svim semaforima promatranog prometnog raskrižja postavlja svjetla na crveno. Dio koda zadužen za učitavanje objekata semafora s promatranog raskrižja i postavljanje inicijalnog stanja prikazan je u prilogu P.3.4.

Nakon inicijalizacije, programskim kodom je definirana beskonačna petlja. Unutar te petlje se na učitava broj vozila na trakama prometnog raskrižja (u dijagramu toka označeno s n_i) i određuje se specifični prometni slučaj na temelju broja vozila u svakoj od traka prometnog raskrižja te se na temelju određenog slučaja upravlja semaforskim ciklusima. Prvi korak u svakom ciklusu upravljanja semaforskim stanjima, nakon postavljanja početnog stanja svih svjetala na crveno, je čitanje broja vozila na svakoj zasebnoj traci koje Klijent 1 stavlja na raspolaganje. Ovi podaci se učitavaju iz *.txt* datoteka koje služe kao komunikacijski kanali između modula za detekciju vozila i modula za upravljanje semaforima. Dio koda zadužen za učitavanje broja vozila u svakoj zasebnoj traci prikazan je u prilogu P.3.5.

Na temelju prikupljenih podataka, algoritam određuje trenutni slučaj (jedan od trideset mogućih slučajeva opisanih u točki 3.5.1.), tj. prepoznaje trenutno prometno stanje i odabire odgovarajuću strategiju upravljanja semaforima. Svaki scenarij ima definiran slijed događaja koji optimizira trajanje

zelenih i crvenih svjetala, čime se postiže bolji protok prometa. Dio koda zadužen za definiranje trenutnog prometnog slučaja na temelju broja vozila u svakoj od traka prikazan je u prilogu P.3.6.



Slika 3.10. *Dijagram toka algoritma za prilagodljivo upravljanje semaforima na prometnom raskrižju*

Kada je aktivan neki od slučajeva u kojima niti jedna traka nema više od 4 vozila, tada algoritam definira prioritetnu traku na temelju infomacije o posljednje aktivnoj traci u prethodnom ciklusu. Nakon definiranja prioritetne trake, prelazi u postavljanje zelenog svjetla za tu traku na osnovni period. Osnovni period je definiran kao trajanje jednog zelenog svjetla u tradicionalnom načinu rada. Prilikom uključivanja zelenog svjetla za zadanu traku, paralelno se uključuje crveno svjetlo za sve ostale trake u jednakom trajanju. Po završetku zelenog svjetla, uključuje se žuto svjetlo na zadani period te se potom postavljaju sva svjetla na semaforima na crveno svjetlo. U slučajevima kada su jedna ili više traka opterećene, algoritam odabire prioritetnu traku među opterećenim trakama te prelazi u postavljanje zelenog svjetla za tu traku na ponavljajući period. Jedan ponavljajući period traje kao pola jednog osnovnog perioda te se nakon svakog ponavljajućeg perioda provjerava broj vozila na svim trakama. Ponavljajući period se završava u slučaju ako je broj vozila u aktivnoj traci manji od 4 ili ako se ponavljajući period ponovi 4 puta, a da pritom na nekoj drugoj traci ima vozila. Tada algoritam prelazi u fazu gdje uključuje jedan završni osnovni period zelenog svjetla za aktivnu traku te potom uključuje žuto svjetlo i postavlja sve semafore na crveno svjetlo na kraju ciklusa. Dio koda zadužen za definiranje prioritetne trake na temelju definiranog stanja i za upravljanje stanjima na semaforu je prikazan u prilogu P.3.7.

4. EVALUACIJA RAZVIJENOG INTELIGENTNOG SUSTAVA ZA UPRAVLJANJE SEMAFORIMA NA PROMETNOM RASKRIŽJU

U ovom poglavlju detaljno su analizirane i evaluirane performanse razvijenog inteligentnog sustava za upravljanje semaforima na prometnom raskrižju, implementiranog unutar CARLA simulatora. Cilj evaluacije bio je utvrditi koliko učinkovito rješenje optimizira prometni tok, smanjuje vrijeme čekanja vozila te prilagođava trajanje semaforskih ciklusa stvarnim uvjetima na prometnim trakama. Za računanje i prikaz rezultata evalucije, zadužen je modul Klijent 3 opisan u potpoglavlju 3.2. Unutar tog modula, označene su koordinate kutova svih prometnih traka na prometnom raskrižju. Nakon toga, dohvaćaju se trenutne lokacije svih stvorenih vozila unutar simulacije. U trenutku kada se neko od vozila pojavi u nekoj od traka promatranog raskrižja tada počinje računanje vremena čekanja za to vozilo. Na taj način se istovremeno bilježi vrijeme čekanja za sva vozila koja čekaju u raskrižju. Po prolasku nekog od vozila se bilježi ukupno vrijeme čekanja za to vozilo i poveća se brojač koji označava ukupni broj vozila koja su prošla kroz raskrižje. Svako od zabilježenih vremena čekanja se zbraja u ukupni zbroj vremena čekanja vozila koja su prošla kroz raskrižje po određenoj traci i ukupni zbroj vremena čekanja svih vozila koja su prošla kroz raskrižje. Na kraju izvođenja simulacije se, na temelju ukupnog zbroja vremena čekanja za svaku traku zasebno i ukupnog zbroja vremena čekanja za sva vozila u raskrižju, računaju prosječna vremena čekanja po vozilu u zasebnoj traci i po vozilo u cijelom raskrižju. Evaluacija je provedena simulacijom različitih prometnih scenarija, od umjerenog prometa s manjim brojem vozila po traci, do scenarija s velikim zagušenjima i vozilima u svim trakama. Ukupno je simulirano šest različitih testnih prometnih scenarija koji su detaljnije opisani kod predstavljanja rezultata za svaki od scenarija.

Glavni cilj evaluacije bio je usporediti učinkovitost sustav za inteligentno upravljanje semaforima s tradicionalnim načinom upravljanja semaforima. Usporedba se temeljila na nekoliko ključnih parametara, kao što su prosječno vrijeme čekanja vozila po traci, prosječno vrijeme čekanja svih vozila koja su prošla kroz raskrižje te ukupnom broju vozila koja su prošla kroz pojedinu traku, odnosno ukupnom broju vozila koja su prošla kroz raskrižje tijekom simulacije. S obzirom na to da u prometu uvijek postoji mogućnost za nasumične događaje, svaka simulacija za određeni prometni scenarij pokrenuta je deset puta kako bi se smanjio utjecaj slučajnih varijacija. Svako testiranje trajalo je deset minuta, što omogućava dovoljan broj promjena stanja unutar simulacije za pouzdanu procjenu performansi sustava.

Budući da je simulacija u CARLA simulatoru usporena za faktor 16 u odnosu na stvarno vrijeme, sva izmjerena vremena tijekom testiranja skalirana su kako bi odražavala vrijednosti u stvarnim uvjetima. Konkretno, sva vremena čekanja vozila u simulaciji podijeljena su s faktorom 16 kako bi se dobile relevantne vrijednosti za stvarnu prometnu situaciju.

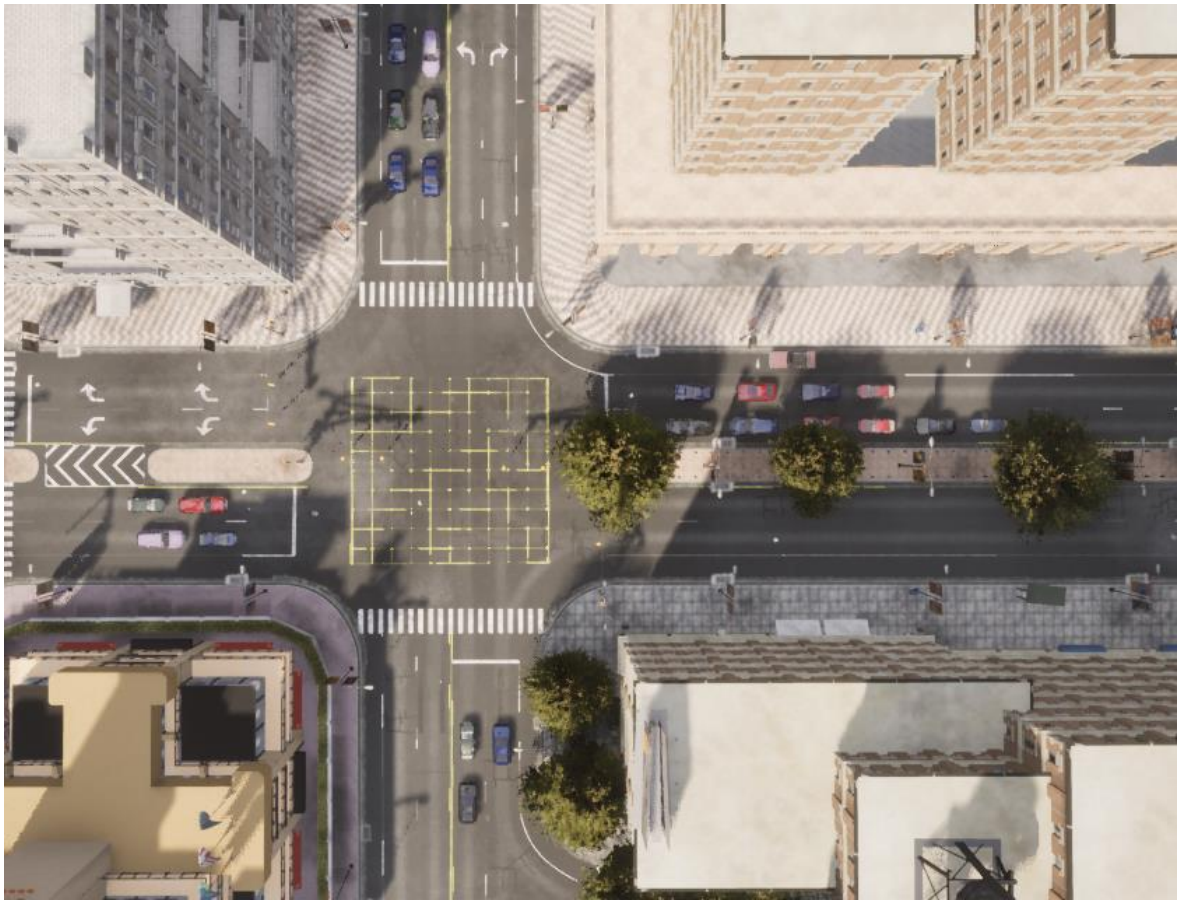
Podaci u redcima u tablicama s rezultatima označenima s „Traka“ prikazuju prosječne vrijednosti dobivene iz deset testova, gdje je svaki test proveden pod istim uvjetima. Ovi podaci odnose se na prosječno vrijeme čekanja po vozilu koje je prošlo kroz specifičnu traku, kao i na broj vozila koja su prošla kroz te trake. S druge strane, podaci u retku u tablicama s rezultatima označenom kao „Sva vozila“ predstavljaju ukupni prosjek vremena čekanja svih vozila koja su prošla kroz raskrižje, bez obzira na traku, kao i prosjek ukupnog broja vozila koja su prošla kroz raskrižje, također na temelju deset testova. Prosječna vremena čekanja i brojevi vozila koja su prošla kroz raskrižje za vrijeme tradicionalnog načina upravljanja semaforima dobiveni su za svaki testni prometni scenarij iz jednog testa s obzirom da je semaforski ciklus fiksan prilikom tradicionalnog načina upravljanja semaforima i ustanovljeno je da ne može doći do značajne promjene u vremenima čekanja.

Uz prosječna vremena čekanja i broj vozila, prikazana je i standardna devijacija koja je izračunata na temelju podataka iz deset testova za svako prosječno vrijeme čekanja. Standardna devijacija predstavlja statističku mjeru raspršenosti ili varijabilnosti podataka u odnosu na prosjek. Ova mjera omogućuje procjenu konzistentnosti rezultata, odnosno manja standardna devijacija u odnosu na prosječnu vrijednost sugerira da su rezultati ujednačeni i da je većina vrijednosti koncentrirana oko prosjeka. U kontekstu evaluacije sustava za inteligentno upravljanje semaforima, niska standardna devijacija je pokazatelj da algoritam postiže konstantne rezultate i da su varijacije u vremenu čekanja minimalne, što ukazuje na stabilan i pouzdan rad sustava.

4.1. Prvi testni prometni scenarij

Prvi testni prometni scenarij, na temelju kojeg je testiran sustav za inteligentno upravljanje semaforima, predstavlja situaciju u kojoj je na raskrižju stvorena maksimalna moguća prometna gužva, uzimajući u obzir tehničke kapacitete i ograničenja simulacijskog okruženja. Također, u simulacijskom okruženju je, osim na trake raskrižja, stvoreno mnogo vozila na okolnim trakama koje mogu voditi do odabranog raskrižja kako bi se promet konstantno obnaljvao. Prometno raskrižje

unutar simulacijskog okruženja sa stvorenim prvim testnim prometnim scenarijem prikazano je na slici 4.1.



Slika 4.1. *Prometno raskrižje unutar simulacijskog okruženja sa stvorenim prvim testnim prometnim scenarijem*

Ovaj scenarij osmišljen je kako bi se ispitala učinkovitost algoritma u uvjetima iznimno visokog prometnog opterećenja, pri čemu se očekuje da sustav prilagodljivo optimizira protok vozila kroz raskrižje unatoč znatnom povećanju broja vozila. U tablici 4.1. prikazani su rezultati za slučaj kada je korišten tradicionalni način upravljanja semaforima kao što je opisano u točki 3.4.1., dok su u tablici 4.2. prikazani rezultati kada je na temelju istog scenarija korišten razvijeni sustav za inteligentno upravljanje semaforima.

Tablica 4.1. *Prosječno vrijeme čekanja i broj vozila – prvi testni prometni scenarij – tradicionalni način upravljanja semaforima*

	Prosječno vrijeme čekanja (s)	Broj vozila
Traka 1	25,06	6
Traka 2	11,49	6
Traka 3	14,6	6
Traka 4	28,17	4
Sva vozila	19,07	22

Tablica 4.2. *Prosječno vrijeme čekanja, standardna devijacija i broj vozila na temelju 10 testova – prvi testni prometni scenarij – inteligentni način upravljanja semaforima*

	Prosječno vrijeme čekanja (s)	Standardna devijacija (s)	Broj vozila
Traka 1	20,86	2,07	9,4
Traka 2	8,71	2,14	7,7
Traka 3	22,04	3,15	6
Traka 4	8,62	2,52	3,1
Sva vozila	16,18	1,31	26,2

Tradicionalni način upravljanja semaforima pokazuje da su prosječna vremena čekanja vozila na pojedinim trakama različita, s Trakom 4 koja ima najveće prosječno vrijeme čekanja od 28,17 sekundi, dok Traka 2 ima najmanje vrijeme čekanja od 11,49 sekundi.

S druge strane, inteligentni sustav pokazuje značajne prednosti u odnosu na tradicionalni način upravljanja. Prosječno vrijeme čekanja vozila na Traci 1 smanjeno je s 25,06 sekundi na 20,86 sekundi, dok je na Traci 2 smanjeno s 11,49 na 8,71 sekundi. Također, inteligentni sustav omogućava veću protočnost vozila, što je vidljivo iz povećanja ukupnog broja vozila koja su prošla kroz raskrižje na 26,2 u odnosu na 22 vozila u tradicionalnom sustavu. Iako je na Traci 3 došlo do povećanja prosječnog vremena čekanja, s 14,6 sekundi na 22,04 sekundi, sustav je značajno smanjio vrijeme čekanja na Traci 4, s 28,17 sekundi na 8,62 sekunde, što ukazuje na bolje upravljanje prioritetima traka s većim prometnim opterećenjem.

Sveukupno, inteligentni način upravljanja semaforima, za ovaj scenarij, omogućio je smanjenje ukupnog prosječnog vremena čekanja za sva vozila, s 19,07 sekundi na 16,18 sekundi, uz povećanje broja vozila koja su prošla kroz raskrižje. Glavni razlog za takve rezultate je fleksibilnost razvijenog sustava za inteligentno upravljanje semaforima u situacijama kada u prometnim traka ima više od 4 vozila koliko u prosjeku može proći kroz jednu traku za vrijeme trajanja jednog osnovnog perioda

zelenog svjetla. Standardne devijacije ukazuju na relativno ujednačena vremena čekanja, s niskim varijacijama, što potvrđuje stabilnost sustava u uvjetima inteligentnog upravljanja prometom.

4.2. Drugi testni prometni scenarij

Drugi testni prometni scenarij, na temelju kojeg je testiran sustav za inteligentno upravljanje semaforima, predstavlja situaciju u kojoj je na raskrižju stvorena maksimalna moguća prometna gužva na tri trake prometnog raskrižja, dok je jedna traka ostavljena prazna. Prometno raskrižje unutar simulacijskog okruženja sa stvorenim drugim testnim prometnim scenarijem prikazano je na slici 4.2.

Slika 4.2. *Prometno raskrižje unutar simulacijskog okruženja sa stvorenim drugim testnim prometnim scenarijem*

Ovaj testni prometni scenarij osmišljen je kako bi se ispitala učinkovitost algoritma u uvjetima kada tradicionalni način uključuje zeleno svjetlo za traku bez vozila, za vrijeme dok druga vozila čekaju na ostalim trakama. U tablici 4.3. prikazani su rezultati za slučaj kada nije korišten sustav za inteligentno

upravljanje semaforima, dok su u tablici 4.4. prikazani rezultati kada je na temelju istog scenarija korišten razvijeni sustav za inteligentno upravljanje semaforima.

Tablica 4.3. *Prosječno vrijeme čekanja i broj vozila – drugi testni prometni scenarij – tradicionalni način upravljanja semaforima*

	Prosječno vrijeme čekanja (s)	Broj vozila
Traka 1	23,5	8
Traka 2	0	0
Traka 3	14,16	6
Traka 4	27,5	2
Sva vozila	20,43	16

Tablica 4.4. *Prosječno vrijeme čekanja, standardna devijacija i broj vozila na temelju 10 testova – drugi testni prometni scenarij – inteligentni način upravljanja semaforima*

	Prosječno vrijeme čekanja (s)	Standardna devijacija (s)	Broj vozila
Traka 1	17	1,16	12
Traka 2	0	0	0
Traka 3	4,49	0,31	6,9
Traka 4	18,40	0,82	3,4
Sva vozila	13,34	0,88	22,3

Kod tradicionalnog načina upravljanja semaforima na Traci 1 prosječno vrijeme čekanja iznosi 23,5 sekundi, dok je na Traci 4 zabilježeno čak 27,5 sekundi čekanja, unatoč relativno malom broju vozila (2 vozila). Ukupno, kroz raskrižje je prošlo 16 vozila, s prosječnim vremenom čekanja svih vozila od 20,43 sekunde.

Prilikom korištenja inteligentnog načina upravljanja semaforima, pri čemu algoritam detektira da na Traci 2 nema vozila i ne dodjeljuje joj zeleno svjetlo, upravljanje semaforima za preostale trake je optimizirano. Kao rezultat, na Traci 1 vrijeme čekanja je smanjeno na 17,01 sekundi, a na Traci 3 prosječno vrijeme čekanja značajno je smanjeno s 14,16 na samo 4,49 sekundi. Također, vrijeme čekanja na Traci 4 smanjeno je s 27,5 sekundi na 18,4 sekundi, što ukazuje na bolje upravljanje i raspodjelu vremena zelenog svjetla prema potrebama. Kao rezultat značajno smanjenog vremena čekanja, ukupan broj vozila koja su prošla kroz raskrižje za vrijeme simulacije povećan je sa 16 na 22,3 vozila, dok je prosječno vrijeme čekanja svih vozila smanjeno s 20,44 sekundi na 13,34 sekunde. U ovom testnom prometnom scenariju, sustav za inteligentno upravljanje semaforima pokazuje sposobnost efikasnijeg upravljanja semaforskim ciklusima u slučaju prometne gužve, kao i

preskakanje semaforskog ciklusa za traku u kojoj se ne nalaze vozila. Standardna devijacija za prosječna vremena čekanja u drugom scenariju pokazuje niže vrijednosti nego što je to bio slučaj kod prvog scenarija, što ukazuje na još stabilnije rezultate u slučaju kada jedna traka nema nijedno vozilo.

Ova usporedba jasno pokazuje prednosti inteligentnog upravljanja semaforima u uvjetima neujednačenog prometnog opterećenja. Sustav za inteligentno upravljanje semaforima smanjuje nepotrebno čekanje na trakama s velikim prometom i efikasno koristi vrijeme semafora na trakama koje su opterećene.

4.3. Treći testni prometni scenarij

Treći testni prometni scenarij predstavlja situaciju kada dvije trake prometnog raskrižja imaju najveću moguću gužvu, dok su preostale dvije trake potpuno prazne. Prometno raskrižje unutar simulacijskog okruženja sa stvorenim trećim testnim prometnim scenarijem prikazano je na slici 4.3.

Slika 4.3. *Prometno raskrižje unutar simulacijskog okruženja sa stvorenim trećim testnim prometnim scenarijem*

Ovaj testni prometni scenarij, slično kao i prethodni, treba pokazati efektivnost inteligentnog načina upravljanja semaforima u slučajevima kada nema potrebe za uključivanjem zelenog svjetla na dvije prometnice. U tablici 4.5. prikazani su rezultati za slučaj kada je korišten tradicionalni način upravljanja semaforima, dok su u tablici 4.6. prikazani rezultati kada je na temelju istog scenarija korišten razvijeni sustav za inteligentno upravljanje semaforima.

Tablica 4.5. *Prosječno vrijeme čekanja i broj vozila – treći testni prometni scenarij – tradicionalni način upravljanja semaforima*

	Prosječno vrijeme čekanja (s)	Broj vozila
Traka 1	25,68	6
Traka 2	0	0
Traka 3	0	0
Traka 4	18,92	5
Sva vozila	22,63	11

Tablica 4.6. *Prosječno vrijeme čekanja, standardna devijacija i broj vozila na temelju 10 testova – treći testni prometni scenarij – inteligentni način upravljanja semaforima*

	Prosječno vrijeme čekanja (s)	Standardna devijacija (s)	Broj vozila
Traka 1	12,72	0,2	12
Traka 2	0	0	0
Traka 3	0	0	0
Traka 4	21,63	1,9	5,2
Sva vozila	15,32	0,99	17,2

Usporedba rezultata iz trećeg testnog prometnog scenarija pokazuje značajne razlike između tradicionalnog i inteligentnog načina upravljanja semaforima. Kod rezultata za tradicionalni način, vidljivo je da se zeleno svjetlo dodjeljuje praznim trakama (Traka 2 i Traka 3), što dovodi do nepotrebnog gubitka vremena i povećanog čekanja na trakama s vozilima. Prosječno vrijeme čekanja na Traci 1 iznosi 25,68 sekundi, dok je ukupno prosječno vrijeme čekanja za sva vozila 22,63 sekunde. Ukupno, kroz raskrižje je prošlo 11 vozila.

S druge strane, rezultati se za inteligentni način upravljanja semaforima znatno razlikuju jer algoritam prepoznaje da na Traci 2 i Traci 3 nema vozila i ne dodjeljuje im zeleno svjetlo, čime optimizira protok prometa. Kao rezultat toga, prosječno vrijeme čekanja na Traci 1 smanjeno je na 12,72 sekundi, dok je ukupno prosječno vrijeme čekanja za sva vozila smanjeno na 15,32 sekunde. Broj vozila koja su prošla kroz raskrižje povećan je na 17,2, što ukazuje na bolju protočnost u inteligentnom sustavu.

4.4. Četvrti testni prometni scenarij

Četvrti testni prometni scenarij opisuje jednostavnu prometnu situaciju kada samo jedna traka na raskrižju ima vozila, dok su sve ostale trake potpuno prazne. Prometno raskrižje unutar simulacijskog okruženja sa stvorenim četvrtim testnim prometnim scenarijem prikazano je na slici 4.4.

Slika 4.4. *Prometno raskrižje unutar simulacijskog okruženja sa stvorenim četvrtim testnim prometnim scenarijem*

Slično kao i u prethodna dva testna prometna scenarija, cilj je prikazati sposobnost inteligentnog načina upravljanja semaforima da preskoči nepotrebne cikluse zelenog svjetla na praznim trakama. U tablici 4.7. prikazani su rezultati za četvrti scenarij za slučaj kada je korišten tradicionalni način upravljanja semaforima, dok su u tablici 4.8. prikazani rezultati kada je na temelju istog scenarija korišten razvijeni sustav za inteligentno upravljanje semaforima.

Tablica 4.7. *Prosječno vrijeme čekanja i broj vozila – četvrti testni prometni scenarij – tradicionalni način upravljanja semaforima*

	Prosječno vrijeme čekanja (s)	Broj vozila
Traka 1	25,8	7
Traka 2	0	0
Traka 3	0	0
Traka 4	0	0
Sva vozila	25,8	7

Tablica 4.8. *Prosječno vrijeme čekanja, standardna devijacija i broj vozila na temelju 10 testova – četvrti testni prometni scenarij – inteligentni način upravljanja semaforima*

	Prosječno vrijeme čekanja (s)	Standardna devijacija (s)	Broj vozila
Traka 1	12,62	0,2	12
Traka 2	0	0	0
Traka 3	0	0	0
Traka 4	0	0	0
Sva vozila	12,62	0,99	12

U četvrtom testnom prometnom scenariju, gdje samo jedna traka ima vozila dok su ostale trake prazne, jasno su prikazane razlike u učinkovitosti između tradicionalnog i inteligentnog načina upravljanja semaforima. Kada je primjenjen sustav za inteligentno upravljanje semaforima, prosječno vrijeme čekanja na Traci 1 smanjeno je na 12,62 sekunde u usporedbi s 25,8 sekundi kod tradicionalnog načina, dok je broj vozila koja su prošla kroz raskrižje povećan za 5 u odnosu na tradicionalni način rada. Ovdje je algoritam uspješno izbjegao nepotrebne cikluse zelenog svjetla na prometnim trakama bez vozila, čime je značajno smanjeno vrijeme čekanja i povećana protočnost prometa. Također, inteligentni način upravljanja semaforima je prepoznao kako Traka 1 ima više od 4 vozila te je držao uključeno zeleno svjetlo na toj traci sve dok se ona nije ispraznila. Standardna devijacija kod trake s vozilima je približna nuli, s obzirom da algoritam u ovakvom scenariju čitavo vrijeme drži uključeno zeleno na traci s vozilima, sve dok ju ne isprazni.

4.5. Peti testni prometni scenarij

U petom testnom prometnom scenariju predstavljen je slučaj u kojem je na svim prometnim trakama prisutan jednak broj vozila, pri čemu taj broj ostaje ispod praga koji bi zahtijevao specifične

intervencije inteligentnog sustava za upravljanje semaforima, odnosno maksimalno 4 vozila po traci. Prometno raskrižje unutar simulacijskog okruženja sa stvorenim petim testnim prometnim scenarijem prikazano je na slici 4.5.

Slika 4.5. *Prometno raskrižje unutar simulacijskog okruženja sa stvorenim petim testnim prometnim scenarijem*

Cilj ovog scenarija je pokazati da sustav za inteligentno upravljanje semaforima može pružiti jednako dobre, a potencijalno i bolje rezultate od tradicionalnog načina upravljanja semaforima, čak i u uvjetima gdje nije potrebna aktivna prilagodba sustava. U tablici 4.9. prikazani su rezultati za peti scenarij za kada se semaforima upravlja na tradicionalni način, dok su u tablici 4.10. prikazani rezultati kada je na temelju istog scenarija korišten razvijeni sustav za inteligentno upravljanje semaforima.

Tablica 4.9. *Prosječno vrijeme čekanja i broj vozila – peti testni prometni scenarij – tradicionalni način upravljanja semaforima*

	Prosječno vrijeme čekanja (s)	Broj vozila
Traka 1	19,84	4
Traka 2	12,33	4
Traka 3	18,84	4
Traka 4	21,8	4
Sva vozila	18,21	16

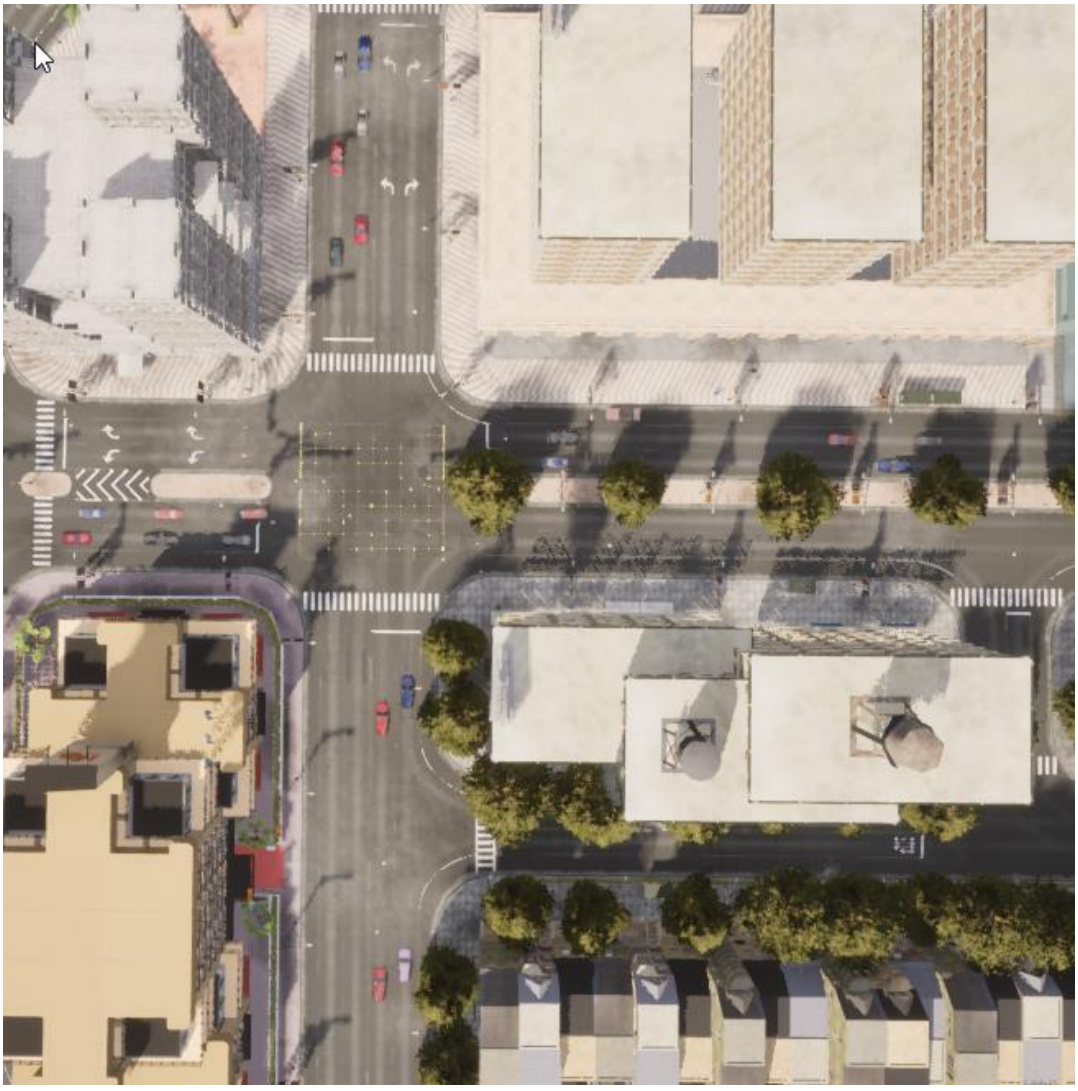
Tablica 4.10. *Prosječno vrijeme čekanja, standardna devijacija i broj vozila na temelju 10 testova – peti testni prometni scenarij – inteligentni način upravljanja semaforima*

	Prosječno vrijeme čekanja (s)	Standardna devijacija (s)	Broj vozila
Traka 1	19,82	0,18	4
Traka 2	12,15	0,15	4
Traka 3	19,24	0,25	4
Traka 4	21,13	0,3	4
Sva vozila	18,09	0,07	16

Usporedba rezultata iz petog testnog prometnog scenarija prikazuje vrlo slične performanse između tradicionalnog i inteligentnog načina upravljanja semaforima u uvjetima kada su sve prometne trake ravnomjerno opterećene, a broj vozila ne prelazi prag koji bi zahtijevao specifične intervencije inteligentnog sustava. Tablica 4.9, koja prikazuje rezultate za tradicionalni način upravljanja semaforima, pokazuje relativno ujednačena prosječna vremena čekanja na svim trakama. Najkraće prosječno vrijeme čekanja zabilježeno je na Traci 2, s 12,33 sekundi, dok je najduže zabilježeno na Traci 4, s 21,8 sekundi. Ukupno prosječno vrijeme čekanja za sva vozila iznosi 18,21 sekundu, a kroz raskrižje je prošlo ukupno 16 vozila. S druge strane, Tablica 4.10 prikazuje rezultate inteligentnog načina upravljanja semaforima, gdje su vremena čekanja također vrlo ujednačena. Na Traci 2, vrijeme čekanja je malo smanjeno na 12,15 sekundi, dok su na ostalim trakama razlike minimalne u odnosu na tradicionalni način. Ukupno prosječno vrijeme čekanja za sva vozila blago je smanjeno na 18,09 sekundi, uz zadržavanje istog broja vozila koja su prošla kroz raskrižje. Uz to, standardna devijacija u inteligentnom načinu upravljanja pokazuje vrlo nisku varijabilnost vremena čekanja između pojedinih simulacija, što ukazuje na stabilnost i konzistentnost algoritma.

4.6. Šesti testni prometni scenarij

Šesti, i ujedno posljednji, testni prometni scenarij na temelju kojeg je evaluiran rad sustava za inteligentno upravljanje semaforima predstavlja situaciju u kojoj je na svim trakama prisutan jednak broj vozila. Za razliku od prethodnog scenarija, u ovom slučaju broj vozila na svakoj traci premašuje prag koji zahtijeva specifičnu intervenciju sustava za inteligentno upravljanje semaforima. Konkretno, na svakoj traci je stvoreno 6 vozila. Prometno raskrižje unutar simulacijskog okruženja sa stvorenim šestim testnim prometnim scenarijem prikazano je na slici 4.6.



Slika 4.6. *Prometno raskrižje unutar simulacijskog okruženja sa stvorenim šestim testnim prometnim scenarijem*

Ovaj prometni scenarij prikazuje situaciju u kojoj tradicionalni način upravljanja semaforima ne može učinkovito riješiti prometno opterećenje, budući da je intervencija nužna za svaku traku kako bi se optimizirao protok prometa. U tablici 4.11. prikazani su rezultati za šesti scenarij za kada se semaforima upravlja na tradicionalni način, dok su u tablici 4.12. prikazani rezultati kada je na temelju istog scenarija korišten razvijeni sustav za inteligentno upravljanje semaforima.

Tablica 4.11. *Prosječno vrijeme čekanja i broj vozila – šesti testni prometni scenarij – tradicionalni način upravljanja semaforima*

	Prosječno vrijeme čekanja (s)	Broj vozila
Traka 1	25,53	6
Traka 2	12,37	4
Traka 3	13,77	3
Traka 4	28,23	3
Sva vozila	19,44	19

Tablica 4.12. *Prosječno vrijeme čekanja, standardna devijacija i broj vozila na temelju 10 testova – šesti testni prometni scenarij – inteligentni način upravljanja semaforima*

	Prosječno vrijeme čekanja (s)	Standardna devijacija (s)	Broj vozila
Traka 1	25,86	0,46	6
Traka 2	16,25	0,27	6
Traka 3	4,68	0,23	6
Traka 4	23,36	0,27	6
Sva vozila	17,54	0,15	24

Usporedba rezultata iz šestog testnog prometnog scenarija jasno pokazuje razliku u učinkovitosti između tradicionalnog i inteligentnog načina upravljanja semaforima u uvjetima kada je broj vozila na svim trakama iznad praga koji zahtijeva specifičnu intervenciju.

Kod rezultata za tradicionalni način upravljanja semaforima, vidljivo je da prosječna vremena čekanja variraju između traka, dok je najveće vrijeme čekanja zabilježeno na Traci 1 koja ima prosječno vrijeme čekanja od 25,53 sekundi.

Kod rezultata inteligentnog načina upravljanja semaforima, gdje algoritam efikasno prilagođava trajanje zelenog svjetla na temelju broja vozila na svakoj traci, vidljivo je značajno smanjenje vremena čekanja na određenim trakama. Traka 3 bilježi drastično smanjenje prosječnog vremena čekanja s 13,77 sekundi na samo 4,68 sekundi. Također, inteligentni sustav omogućio je veću protočnost, pa je kroz raskrižje za vrijeme simulacije prošlo 24 vozila u usporedbi s 19 prilikom tradicionalnog načina

upravljanja semaforima. Iako je vrijeme čekanja na Traci 1 ostalo približno isto za oba načina rada (25,86 sekundi), došlo je do značajnijeg smanjenja vremena čekanja na Traci 4, gdje je vrijeme smanjeno s 28,23 na 23,36 sekundi.

Zaključno, inteligentni način upravljanja semaforima smanjio je prosječno vrijeme čekanja za sva vozila s 19,44 na 17,54 sekunde, uz minimalnu standardnu devijaciju, što ukazuje na dosljedne rezultate kroz deset testova. Ovaj rezultat, kao i rezultati prethodnih scenarija, pokazuje da inteligentni sustav bolje raspodjeljuje prometna opterećenja što omogućuje veću protočnost i smanjenje čekanja na preopterećenim trakama, čime nadmašuje tradicionalni sustav upravljanja semaforima u uvjetima visoke prometne opterećenosti.

5. ZAKLJUČAK

U ovom diplomskom radu predloženo je rješenje za sustav inteligentnog upravljanja prometnim raskrižjima zasnovano na računalnom vidu. Sustav uspješno, pomoću kamera postavljenih na prometno raskrižje i YOLO detektora objekata, detektira prometnu gužvu prebrojavanjem vozila u koloni te na temelju prikupljenih informacija o broju vozila u svakoj od prometnih traka na raskrižju inteligentno upravlja radom semafora kako bi se smanjilo prosječno vrijeme čekanja vozila na raskrižju i povećala protočnost prometa. Razvoj, testiranje i evalucija predloženog rješenja napravljena je u CARLA simulatoru koji je pružio stvaranje realističnih prometnih situacija i stvaranje kamera za detekciju i prebrojavanje vozila u prometnim trakama na rasrkižju te testiranje razvijenog sustava za inteligentno upravljanje semaforima prometnog raskrižja.

Evaluacija razvijenog sustava provedena je kroz simulaciju različitih testnih prometnih scenarija. U svakom scenariju sustav je pokazao bolje rezultate od tradicionalnog pristupa, pogotovo u situacijama s većim prometnim gužvama. U svakom testnom prometnom scenariju, koristeći sustav za inteligentno upravljanje semaforima na prometnom raskrižju, smanjeno je prosječno vrijeme čekanja po vozilu. Razvijeni algoritam je najbolje pokazao svoju sposobnost inteligentnog upravljanja prometom u prometnim scenarijima kada su istovremeno neke trake raskrižja potpuno prazne, dok je na drugim trakama prometna gužva.

Međutim, postoje određena ograničenja. CARLA simulator, iako pruža realistične simulacije, ne može u potpunosti replicirati složenost stvarnih prometnih uvjeta. Dodatno testiranje u stvarnom okruženju bilo bi neophodno za daljnju validaciju i dodatni razvoj algoritma. Također, prilagodba sustava za različite vrste vozila i situacije poput hitnih slučajeva predstavlja potencijalni izazov za implementaciju sustav na prometna raskrižja u stvarnom okruženju.

Kao smjernice za budući rad, predlaže se optimizacija algoritma za specifične prometne uvjete, integracija dodatnih senzora poput pješačkih prijelaza te evaluacija sustava u stvarnim uvjetima. Razvoj algoritma za upravljanje semaforima koji bi uzimao u obzir i prioritetna vozila, poput vozila hitne pomoći, također je potencijalno područje za daljnji razvoj.

Općenito, razvijeni sustav pruža obećavajuće rezultate za optimizaciju upravljanja raskrižjima i može imati široku primjenu u urbanim sredinama gdje je upravljanje prometom ključno za smanjenje gužvi i zagađenja te poboljšanje protočnosti prometa.

LITERATURA

[1] M. Razavi, M. Hamidkhani, R. Sadeghi, Smart Traffic Light Scheduling in Smart City Using Image and Video Processing, Third International Conference on Internet of Things and Applications, University of Isfahan, Isfahan, Iran, 2019.

[2] J. Rezgui, M. Barri, R. Gayta, Smart Traffic Light Scheduling Algorithms, Laboratoire Recherche Informatique Maisonneuve, Montreal, Canada, 2019.

[3] R. Ravish, S. R. Swamy, INTELLIGENT TRAFFIC MANAGEMENT: A REVIEW OF CHALLENGES, SOLUTIONS, AND FUTURE PERSPECTIVES, Department of Computer Science and Engineering, PES University, Bangalore, India, 2021.

[4] K. M. A. Yousef, A. M. Shatnawi, M. Latayfeh, Intelligent Traffic Light Scheduling Technique Using Calendar-Based History Information, Future Generation Computer Systems, 2018.

[5] A. A.A. Alkhatiba, K. A. Mariaa, S. AlZu'bib, E. A. Mariaa, Smart Traffic Scheduling for Crowded Cities Road Networks, Computer Information System Department, Alzaytoonah University of Jordan, Amman, Jordan, 2021.

[6] L. F. P. Oliveira, L. T. Manera, P. D. G. da Luz, Development of a Smart Traffic Light Control System With Real-Time Monitoring, IEEE INTERNET OF THINGS JOURNAL, VOL. 8, NO. 5, 2021.

[7] What is Notepad++, dostupno na: https://notepad-plus-plus.org/ [31.8.2024.]

[8] CARLA, Open-source simulator for autonomous driving research, dostupno na: https://carla.org/ [31.8.2024.]

[9] Glob, dostupno na: https://docs.python.org/3/library/glob.html, [31.8.2024.]

[10] Os, dostupno na: https://docs.python.org/3/library/os.html, [31.8.2024.]

[11] Sys, dostupno na: https://docs.python.org/3/library/sys.html, [31.8.2024.]

[12] Time, dostupno na: https://docs.python.org/3/library/time.html, [31.8.2024.]

[13] OpenCV, dostupno na: https://pypi.org/project/opencv-python/, [31.8.2024.]

[14] NumPy, dostupno na: https://numpy.org/, [31.8.2024.]

[15] Pandas, dostupno na: https://pandas.pydata.org/ [31.8.2024.]

[16] Ultralytics, dostupno na: https://pypi.org/project/ultralytics/ [31.8.2024.]

[17] Threading, dostupno na: https://docs.python.org/3/library/threading.html [31.8.2024.]

[18] Torch, dostupno na: https://pypi.org/project/torch/ [31.8.2024.]

[19] Torchvision, dostupno na: https://pypi.org/project/torchvision/, [31.8.2024.]

[20] Izvorni kod *Tracker* modula, dostupno na: https://github.com/freedomwebtech/yolov8-opencv-win11/blob/main/tracker.py [18.9.2024.]

[21] J. R. Terven, D. M. Cordova-Esparza, A COMPREHENSIVE REVIEW OF YOLO ARCHITECTURES IN COMPUTER VISION: FROM YOLOV1 TO YOLOV8 AND YOLO-NAS, PUBLISHED AS A JOURNAL PAPER AT MACHINE LEARNING AND KNOWLEDGE EXTRACTION, 2024.

[22] A. F. Gad, Evaluating Deep Learning Models: The Confusion Matrix, Accuracy, Precision, and Recall, 2020., dostupno na: https://blog.paperspace.com/deep-learning-metrics-precision-recall-accuracy/ [27.6.2024]

[23] Precision-Recall, dostupno na: https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html [8.9.2024.]

[24] G. Jocher, R. Munawar, A. Vina, Performance Metrics Deep Dive, 2023., dostupno na: https://docs.ultralytics.com/guides/yolo-performance-metrics/#conclusion [27.6.2024]

[25] LabelMe: Image Polygonal Annotation with Python, dostupno na: https://github.com/labelmeai/labelme [28.6.2024.]

[26] Town 10, dostupno na: https://carla.readthedocs.io/en/latest/map_town10/ [8.9.2024.]

SAŽETAK

Glavni cilj ovog diplomskog rada bio je razviti sustav za inteligentno upravljanje semaforima koji smanjuje vrijeme čekanja vozila i povećava protočnost prometa na raskrižjima. Tradicionalni sustavi upravljanja semaforima često stvaraju nepotrebne zastoje u prometu. Predloženo je rješenje koje na temelju računalnog vida prilagodljivo upravlja semaforima. U radu je analizirano simulacijsko okruženje CARLA simulatora te su pojašnjeni osnovni principi stvaranja prometa i RGB kamera s kojima se snimala svaka traka prometnog raskrižja. U radu su detaljno prikazani razvojni koraci implementacije, uključujući postavljanje prometnih scenarija unutar simulatora, odabir prikladnog detektora objekata za detekciju vozila te integraciju detektora u simulacijsko okruženje. Podaci prikupljeni putem računalnog vida se obrađuju kako bi se prilagodila trajanja semaforskih ciklusa, čime se optimizira protok prometa. Sustav za inteligentno upravljanje semaforima na prometnom raskrižju prilagođava semaforske cikluse u stvarnom vremenu na temelju trenutnog broja vozila u svakoj traci. Evaluacija sustava obavljena je stvaranjem različitih prometnih scenarija na temelju kojih se usporedio rad sustava za inteligentno upravljanje semaforima s tradicionalnim načinom upravljanja semaforima. Rezultati evaluacije sustava pokazali su značajno smanjenje prosječnog vremena čekanja po vozilu te je povećana ukupna protočnost prometa na prometnom raskrižju.

Ključne riječi: algoritam, inteligentno upravljanje, prometno raskrižje, računalni vid, semafori

INTELLIGENT TRAFFIC INTERSECTION MANAGEMENT BASED ON COMPUTER VISION

ABSTRACT

The main goal of this thesis was to develop a system for intelligent traffic light management that reduces vehicle waiting time and increases traffic flow at intersections. Traditional traffic light management systems often cause unnecessary traffic jams. The proposed solution uses computer vision to adaptively control the traffic lights. The thesis analyzes the simulation environment of the CARLA simulator and explains the basic principles of traffic generation and the RGB cameras used to record each lane of the traffic intersection. The paper provides a detailed description of the development steps, including setting up traffic scenarios within the simulator, selecting an appropriate object detector for vehicle detection, and integrating the detector into the simulation environment. Data collected through computer vision is processed to adjust traffic light cycle durations, optimizing traffic flow. The intelligent traffic light management algorithm adjusts the traffic light cycles in real-time based on the current number of vehicles in each lane. The system evaluation was performed by creating different traffic scenarios to compare the performance of the intelligent traffic light management algorithm with the traditional traffic light management method. The evaluation results showed a significant reduction in the average vehicle waiting time and an overall increase in traffic flow at the intersection.

Keywords: algorithm, intelligent management, computer vision, intersection, traffic lights

PRILOZI

Prilog P.3.1: Dio programskog koda unutar Klijent 1 modula zadužen za definiranje koordinata za stvaranje vozila te stvaranje i konfiguracija vozila

```
spawn_points = [

            carla.Transform(carla.Location(x=-64.644844, y=24.471010, z=0.600000),
carla.Rotation(pitch=0.000000, yaw=0.159198, roll=0.000000)),

            carla.Transform(carla.Location(x=-67.254570, y=27.963758, z=0.600000),
carla.Rotation(pitch=0.000000, yaw=0.159198, roll=0.000000)),

            carla.Transform(carla.Location(x=-76.666107, y=24.471010, z=0.600000),
carla.Rotation(pitch=0.000000, yaw=0.159198, roll=0.000000)),

            carla.Transform(carla.Location(x=-79.275833, y=27.963758, z=0.600000),
carla.Rotation(pitch=0.000000, yaw=0.159198, roll=0.000000)),

            carla.Transform(carla.Location(x=-87.276062, y=24.441530, z=0.600000),
carla.Rotation(pitch=0.000000, yaw=0.159198, roll=0.000000)),

            carla.Transform(carla.Location(x=-89.885788, y=27.934278, z=0.600000),
carla.Rotation(pitch=0.000000, yaw=0.159198, roll=0.000000))

]

number_of_spawn_points = len(spawn_points)

SpawnActor = carla.command.SpawnActor

SetAutopilot = carla.command.SetAutopilot

FutureActor = carla.command.FutureActor


batch = []

hero = args.hero

for n, transform in enumerate(spawn_points):

      if n >= args.number_of_vehicles:

            break

      blueprint = random.choice(blueprints)

      if blueprint.has_attribute('color'):

            color = random.choice(blueprint.get_attribute('color').
recommended_values)

            blueprint.set_attribute('color', color)
```

```
        if blueprint.has_attribute('driver_id'):
            driver_id = random.choice(blueprint.get_attribute('driver_id').
recommended_values)
            blueprint.set_attribute('driver_id', driver_id)
        if hero:
            blueprint.set_attribute('role_name', 'hero')
            hero = False
        else:
            blueprint.set_attribute('role_name', 'autopilot')
        batch.append(SpawnActor(blueprint, transform).then(SetAutopilot(FutureActor,
True, traffic_manager.get_port())))


for response in client.apply_batch_sync(batch, synchronous_master):
        if response.error:
            logging.error(response.error)
        else:
            vehicles_list.append(response.actor_id)


if args.car_lights_on:
        all_vehicle_actors = world.get_actors(vehicles_list)
        for actor in all_vehicle_actors:
             traffic_manager.update_vehicle_lights(actor, True)
```

Prilog P.3.2: Dio programskog koda unutar Klijent 1 modula zadužen za definiranje koordinata za stvaranje RGB kamera te stvaranje i konfiguracija RGB kamera

```
camera_bp = world.get_blueprint_library().find('sensor.camera.rgb')

camera_init_trans0 = carla.Transform(carla.Location(x=-35.064191, y=7.963309,
z=6.254254), carla.Rotation(pitch=-16.132141, yaw=25, roll=0.000017))

camera0 = world.spawn_actor(camera_bp, camera_init_trans0)

camera_init_trans1 = carla.Transform(carla.Location(x=-51.352070, y=6.096909,
z=7.254254), carla.Rotation(pitch=-16.132141, yaw=270.282227, roll=0.000017))

camera1 = world.spawn_actor(camera_bp, camera_init_trans1)

camera_init_trans2 = carla.Transform(carla.Location(x=-57.236356, y=27.589287,
z=7.254254), carla.Rotation(pitch=-16.132141, yaw=180.282227, roll=0.000017))

camera2 = world.spawn_actor(camera_bp, camera_init_trans2)

camera_init_trans3 = carla.Transform(carla.Location(x=-40.930841, y=34.301954,
z=7.254254), carla.Rotation(pitch=-16.132141, yaw=90.282227, roll=0.000017))

camera3 = world.spawn_actor(camera_bp, camera_init_trans3)

image_w = camera_bp.get_attribute("image_size_x").as_int()

image_h = camera_bp.get_attribute("image_size_y").as_int()

print('',image_w, image_h)
```

Prilog P.3.3: Programski kod funkcije, unutar Klijent 1 modula, za obradu slike, detekciju vozila i prebrojavanje vozila pomoću YOLO detektora i spremanje broja vozila u tekstualne datoteke

```
my_file = open("coco.txt", "r")
data = my_file.read()
class_list = data.split("\n")
tracker = Tracker()

model = YOLO('yolov8n.pt')
vehicle_counts = [0,0,0,0]

def rgb_callback(image, data_dict, rb):
      array = np.frombuffer(image.raw_data, dtype=np.dtype("uint8"))
```

```
        array = np.reshape(array, (image.height, image.width, 4))
        frame = array[:, :, :3]  # BGR format for OpenCV
        frame = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)

        frame_resized = cv2.resize(frame, (832, 608))

        frame_tensor = torch.from_numpy(frame_resized).permute(2,0,1).float().
div(255.0).unsqueeze(0).to(device)

        img = np.reshape(np.copy(image.raw_data), (image.height, image.width, 4))
        img[:,:,3] = 255

        areas = {
             0: np.array([(738, 437), (478, 567), (232, 217), (280, 212)], np.int32),
             1: np.array([(605, 553), (268, 554), (379, 242), (429, 240)], np.int32),
             2: np.array([(656, 584), (288, 584), (375, 267), (452, 267)], np.int32),
             3: np.array([(684, 556), (349, 556), (390, 223), (428, 221)], np.int32)
        }

        area1 = areas[rb]
        tracker = Tracker()

        results = model.predict(frame_tensor)
        a = results[0].boxes.data
        px = pd.DataFrame(a.cpu()).astype("float")

        list=[]

        for index, row in px.iterrows():
             x1, y1, x2, y2 = int(row[0]), int(row[1]), int(row[2]), int(row[3])
              d = int(row[5])
              c = class_list[d]
              list.append([x1,y1,x2,y2])

        bbox_id=tracker.update(list)
        car_count_0 = set()

        frame_umat = cv2.UMat(frame)

        for bbox in bbox_id:
              x3,y3,x4,y4,id=bbox
              cx=int(x3+x4)//2
              cy=int(y3+y4)//2
              results=cv2.pointPolygonTest(np.array(area1,np.int32),((cx,cy)),False)
```

```
            if results >= 0:
                car_count_0.add(id)

      count=len(car_count_0)
      print('CAMERA %d: ' % rb)

      vehicle_counts[rb] = count
      file_name = f"camera_{rb}_count.txt"
      with open(file_name, 'w') as file:
            file.write(f"{count}\n")

      print(f"Current vehicle counts: {vehicle_counts[0]} {vehicle_counts[1]}
{vehicle_counts[2]} {vehicle_counts[3]}")

      img = cv2.resize(img, (800, 600))
      data_dict['rgb_image'] = img

      return frame_umat
```

Prilog P.3.4: Programski kod, sadržan u Klijent 2 modulu, zadužen za učitavanje objekata semafora te inicijalno postavljanje semafora na crveno svjetlo

---

```
def find_traffic_lights(traffic_lights):
    for traffic_light in traffic_lights:
        location = traffic_light.get_location()
        if location.x == -64.26419067382812 and location.y == 7.063309192657471:
            semafori[0] = traffic_light
            print('Semafor 0 pronadjen')
        if location.x == -62.35206985473633 and location.y == 20.196908950805664:
            semafori[1] = traffic_light
            print('Semafor 1 pronadjen')
        if location.x == -31.636356353759766 and location.y == 33.58928680419922:
            semafori[2] = traffic_light
```

```
            print('Semafor 2 pronadjen')
        if location.x == -31.93084144592285 and location.y == 20.30195426940918:
            semafori[3] = traffic_light
            print('Semafor 3 pronadjen')

    for index in range(4):
        set_initial_red(semafori[index], index)
    input("Pritisni Enter za nastavak...")
    print("Nastavak")
def set_initial_red(semafor, index):
    if semafor:
        semafor.set_state(carla.TrafficLightState.Red)
        semafor.set_red_time(20.0)
        print(f"Semafor {index} inicijalno postavljen na crveno")
    else:
        print(f"Greska {index}")
```

Prilog P.3.5: Programski kod, sadržan u Klijent 2 modulu, zadužen za učitavanje podataka o broju vozila za svaku traku iz tekstualnih datoteka

```
def read_vehicle_counts():
    vehicle_counts = []
    for i in range(4):
        try:
            with open(f'camera_{i}_count.txt','r') as file:
                count = int(file.readline().strip())
```

```
                vehicle_counts.append(count)
        except FileNotFoundError:
            vehicle_counts.append(0)
        except ValueError:
            vehicle_counts.append(0)
    return vehicle_counts
```

Prilog P.3.6: Programski kod, sadržan u Klijent 2 modulu, zadužen za određivanje trenutnog slučaja na temelju podataka o broju vozila na trakama

```
def define_state(vehicle_counts):
    global current_state
    if not isinstance(vehicle_counts, list) or len(vehicle_counts) != 4 or not all(isinstance(x, int) for x in vehicle_counts):
        raise ValueError("Funkcija očekuje listu od četiri cijela broja.\n")
    # CASE 30: Sve trake imaju 0 vozila, gori crveno sve dok ne dodje do promjene
    if all(v == 0 for v in vehicle_counts):
        current_state = 30
        print("CASE 30: Sve trake imaju 0 vozila, crveno svjetlo na svim semaforima.\n")
        return current_state
    # CASE 0: Defaultni mod rada
    if all(1 <= v <= 4 for v in vehicle_counts):
        current_state = 0
```

```
        print("CASE 0: Defaultni mod rada, svi semafori rade u defaultnom ciklusu.\n")
        return current_state
    # CASE 1-4: Defaultni mod rada bez određene trake
    for i in range(4):
        if vehicle_counts[i] == 0 and all(1 <= vehicle_counts[j] <= 4 for j in range(4) if j != i):
            current_state = i + 1
            print(f"CASE {current_state}: Defaultni mod rada bez trake {i}.\n")
            return current_state
    # CASE 5-10: Defaultni mod rada bez dvije trake
    for i in range(4):
        for j in range(i + 1, 4):
            if vehicle_counts[i] == 0 and vehicle_counts[j] == 0 and all(1 <= vehicle_counts[k] <= 4 for k in range(4) if k != i and k != j):
                current_state = 5 + (i * 2 + j - 1)
                print(f"CASE {current_state}: Defaultni mod rada bez traka {i} i {j}.\n")
                return current_state
    # CASE 11-14: Samo jedna traka ima vozila
    for i in range(4):
        if vehicle_counts[i] > 0 and all(vehicle_counts[j] == 0 for j in range(4) if j != i):
            current_state = 11 + i
            print(f"CASE {current_state}: Samo traka {i} ima vozila, ostalo je 0\n")
            return current_state
    # CASE 15-18: Samo jedna traka ima više od 4 vozila, ostale 0-4
```

```
    for i in range(4):
        if vehicle_counts[i] > 4 and all(0 <= vehicle_counts[j] <= 4 for j in range(4) if j != i):
            current_state = 15 + i
            print(f"CASE {current_state}: Traka {i} ima više od 4 vozila, ostale trake su ispod 4.\n")
            return current_state

    # CASE 19-24: Dvije trake imaju više od 4 vozila, ostale 0-4
    for i in range(4):
        for j in range(i + 1, 4):
            if vehicle_counts[i] > 4 and vehicle_counts[j] > 4 and all(0 <= vehicle_counts[k] <= 4 for k in range(4) if k != i and k != j):
                if i == 2 and j == 3:
                    current_state = 19
                if i == 1 and j == 3:
                    current_state = 20
                if i == 1 and j == 2:
                    current_state = 21
                if i == 0 and j == 3:
                    current_state = 22
                if i == 0 and j == 2:
                    current_state = 23
                if i == 0 and j == 1:
                    current_state = 24
```

```
                print(f"CASE {current_state}: Trake {i} i {j} imaju više od 4 vozila, ostale trake su ispod 4.\n")
                return current_state

    # CASE 25-28: Samo jedna traka ima 0-4 vozila, ostale više
    for i in range(4):
        if vehicle_counts[i] <= 4 and all(vehicle_counts[j] > 4 for j in range(4) if j != i):
            current_state = 25 + i
            print(f"CASE {current_state}: Traka {i} ima 0-4 vozila, ostale trake imaju više od 4.\n")
            return current_state

    # CASE 29: Sve trake imaju više od 4 vozila
    if all(v > 4 for v in vehicle_counts):
        current_state = 29
        print("CASE 29: Sve trake imaju više od 4 vozila, upravljanje prema prioritetima.\n")
        return current_state
    return None
```

Prilog P.3.7: Dio programskog, sadržan u Klijent 2 modulu, zadužen za definiranje prioritetne trake na temelju definiranog stanja i za upravljanje stanjima na semaforu

```
def manage_traffic_lights():

    global last_green

    while True:
        vehicle_counts = read_vehicle_counts()
        print(f"Current vehicle counts: {list(vehicle_counts)}")
        current_state = define_state(vehicle_counts)

        if current_state is not None:

            if current_state == 0:
                print("Upravljanje za CASE 0")
                next_green = (last_green + 1) % 4
                control_traffic_light(next_green)
                last_green = next_green

            elif 1 <= current_state <= 4:
                print(f"Upravljanje za CASE {current_state}")
                inactive_track = current_state - 1
                next_green = (last_green + 1) % 4
                if next_green == inactive_track:
                    next_green = (next_green + 1) % 4
                control_traffic_light(next_green)
                last_green = next_green

            elif 5 <= current_state <= 10:
```

```
        print(f"Upravljanje za CASE {current_state}")
        skip_lanes = {
            5: [0,1],
            6: [0,2],
            7: [0,3],
            8: [1,2],
            9: [1,3],
            10: [2,3]
        }

        next_green = (last_green + 1) % 4
        while next_green in skip_lanes[current_state]:
            next_green = (next_green + 1) % 4
        control_traffic_light(next_green)
        last_green = next_green

    elif 11 <= current_state <= 14:
        print(f"Upravljanje za CASE {current_state}")
        active_track = current_state - 11
        control_single_track(active_track, vehicle_counts)

    elif 15 <= current_state <= 18:
        print(f"Upravljanje za CASE {current_state}")
        active_track = current_state - 15
        control_single_track(active_track, vehicle_counts)

    elif 19 <= current_state <= 24:
        print(f"Upravljanje za CASE {current_state}")
        skip_lanes = {
            19: [0,1],
```

```
                20: [0,2],
                21: [0,3],
                22: [1,2],
                23: [1,3],
                24: [2,3]
            }
            next_green = (last_green + 1) % 4
            while next_green in skip_lanes[current_state]:
                next_green = (next_green + 1) % 4
            control_single_track(next_green, vehicle_counts)
            last_green = next_green

        elif 25 <= current_state <= 28:
            print(f"Upravljanje za CASE {current_state}")
            inactive_track = current_state - 25
            next_green = (last_green + 1) % 4
            if next_green == inactive_track:
                next_green = (next_green + 1) % 4
            control_single_track(next_green, vehicle_counts)
            last_green = next_green

        elif current_state == 29:
            print(f"Upravljanje za CASE {current_state}")
            next_green = (last_green + 1) % 4
            control_single_track(next_green, vehicle_counts)
            last_green = next_green

        elif current_state == 30:
            print("Upravljanje za CASE 30: svi semafori crveni, cekanje
vozila")
            all_red(24)
```

```
        start_time = time.time()
        while time.time() - start_time < 1:
            pass

def control_traffic_light(index)
    print(" ")
    #upravljanje svjetlom za odabrani semafor
    for i, semafor in enumerate(semafori):
        if i == index:
            semafor.set_state(carla.TrafficLightState.Green)
            semafor.set_green_time(80.0)
            print(f"UPALJENO ZELENO SVJETLO ZA SEMAFOR {index} NA DEFAULTNI
PERIOD")
        else:
            semafor.set_state(carla.TrafficLightState.Red)
            semafor.set_red_time(104) #80+24 - zeleno + zuto na aktivnom semaforu
            print(f"Upaljeno crveno svjetlo za semafor {i}")

    start_time = time.time()
    while time.time() - start_time < 80:
        pass  # Waiting for 80 seconds
    print("Sleep 80 sekundi")

    semafori[index].set_state(carla.TrafficLightState.Yellow)
    semafori[index].set_yellow_time(24)

    start_time = time.time()
    while time.time() - start_time < 24:
        pass  # Waiting for 24 seconds
    all_red(24)
```

```
def control_traffic_light_custom(index):
    print(" ")
    for i, semafor in enumerate(semafori):
        if i == index:
            semafor.set_state(carla.TrafficLightState.Green)
            semafor.set_green_time(40.0)
            print(f"UPALJENO ZELENO SVJETLO ZA SEMAFOR {index} NA PONAVLJAJUCI
PERIOD")
        else:
            semafor.set_state(carla.TrafficLightState.Red)
            semafor.set_red_time(40.0)
            print(f"Upaljeno crveno svjetlo za semafor {i}")

    start_time = time.time()
    while time.time() - start_time < 40:
        pass  # Waiting for 40 seconds

def control_single_track(index, vehicle_counts):
    repeat_counter = 0
    while vehicle_counts[index] > 4:

        if any(vehicle_counts[i] > 0 for i in range(4) if i != index) and
repeat_counter >= 4:
            break
        control_traffic_light_custom(index)
        vehicle_counts = read_vehicle_counts()
        repeat_counter += 1

    control_traffic_light(index)
```

```
def all_red(duration):
    for semafor in semafori:
        semafor.set_state(carla.TrafficLightState.Red)
        semafor.set_red_time(duration)

    print("Upaljeno crveno za sve semafore... Ceka se novi CASE\n")
    start_time = time.time()
    while time.time() - start_time < duration:
        pass  # Waiting for duration seconds
```