UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA U NOVOM SADU

DIPLOMSKI RAD

Detekcija znakova i praćenje traka pri kretanju autonomnog vozila u saobraćaju

Kandidat:

Jovan Slavujević

Mentor:

Prof. dr Vladimir Rajs

Novi Sad, 2020.god.

UNIVERZITET U NOVOM SADU ●**FAKULTET TEHNIČKIH NAUKA**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KLJUČNA DOKUMENTACIJSKA INFORMACIJA

Redni broj, **RBR**:		
Identifikacioni broj, **IBR**:		
Tip dokumentacije, **TD**:		Monografska publikacija
Tip zapisa, **TZ**:		Tekstualni štampani dokument
Vrsta rada, **VR**:		Diplomski rad
Autor, **AU**:		Jovan Slavujević
Mentor, **MN**:		Prof. dr Vladimir Rajs
Naslov rada, **NR**:		Detekcija znakova i praćenje traka pri kretanju autonomnog vozila u saobraćaju
Jezik publikacije, **JP**:		Srpski
Jezik izvoda, **JI**:		Srpski
Zemlja publikovanja, **ZP**:		Republika Srbija
Uže geografsko područje, **UGP**:		Autonomna Pokrajina Vojvodina
Godina, **GO**:		2020.
Izdavač, **IZ**:		Autorski reprint
Mesto i adresa, **MA**:		21000 Novi Sad, Fakultet tehničkih nauka, Trg Dositeja Obradovića 6
Fizički opis rada, **FO**: (poglavlja/strana/citata/tabela/slika/grafika/priloga)		10 / 42 / 10 / 1 / 55 / 8 / 0
Naučna oblast, **NO**:		Automatika
Naučna disciplina, **ND**:		Računarske komunikacije
Predmetna odrednica/Ključne reči, **PO**:		Automotive
UDK		
Čuva se, **ČU**:		Biblioteka Fakulteta tehničkih nauka, Trg Dositeja Obradovića 6, Novi Sad
Važna napomena, **VN**		
Izvod, **IZ**:		U projektu je simuliran rad autonomnog vozila korišćenjem ROS platforme, odnosno njegovih nodova kao komunikacionih jedinica. Nodovi vrše prijem snimka sa prednje kamere automobila, slanje snimka – "frejm po frejm" na obradu, odnosno, pretprocesuiranje i procesuiranje, te slanje podataka ka ECU jedinici – da li je znak detektovan ili nije, kao i šta je detektovano, da li su trake detekovane i da li se vozilo treba prestrojavati, zatim vizualizaciju obrađenog snimka, kao i nadziranje rada celokupnog sistema. Ukoliko neki od nodova prestane da radi, specijalni "*Watchdog*" nod treba ponovo da ga pokrene.
Datum prihvatanja teme, **DP**:		Mart 2020.
Datum odbrane, **DO**:		10.03.2020
Članovi komisije, **KO**:	Predsednik:	
	Član:	Потпис ментора
	Mentor:	doc. dr Vladimir Rajs

UNIVERSITY OF NOVI SAD●**FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:		
Identification number, **INO**:		
Document type, **DT**:		Monographic publication
Type of record, **TR**:		Textual printed material
Contents code, **CC**:		Bachelor Thesis
Author, **AU**:		Jovan Slavujević
Mentor, **MN**:		PhD Vladimir Rajs
Title, **TI**:		Traffic sign detection and lane following during the movement of autonomous vehicle in traffic conditions
Language of text, **LT**:		Serbian
Language of abstract, **LA**:		English
Country of publication, **CP**:		Republic of Serbia
Locality of publication, **LP**:		Autonomous Province of Vojvodina
Publication year, **PY**:		2020.
Publisher, **PB**:		Author's reprint
Publication place, **PP**:		Faculty of Technical Sciences, Trg Dositeja Obradovića 6, 21000 Novi Sad
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)		10 / 42 / 10 / 1 / 55 / 8 / 0
Scientific field, **SF**:		Automation
Scientific discipline, **SD**:		Computer communications
Subject/Key words, **S/KW**:		Automotive
UC		
Holding data, **HD**:		Library of the Faculty of Technical Sciences, Novi Sad
Note, **N**:		
Abstract, **AB**:		In this project the operation of an autonomous vehicle was simulated using the ROS platform, more specifically its nodes as communication units. The nodes receive the front camera footage, they send the image frame by frame to be processed – preprocessing and processing – sfter which the data is sent to the ECU to be determined whether or not a traffic sign was detected, as well as what has been detected, wheter the traffic lanes were detected and does the vehicle need to change lanes. The process is then visualized, and the performace of the whole system is monitored. If any node ceases operation, a special *Watchdog* node is supposed to restart it.
Accepted by the Scientific Board on, **ASB**:		March 2020.
Defended on, **DE**:		10.03.2020
Defended Board, **DB**:	President:	
	Member:	
	Member, Mentor	PhD Vladimir Rajs

Menthor's sign

NA OSNOVU PODNETE PRIJAVE, PRILOŽENE DOKUMENTACIJE I ODREDBI STATUTA FAKULTETA IZDAJE SE ZADATAK ZA DIPLOMSKI RAD, SA SLEDEĆIM ELEMENTIMA:			
Student:	Jovan Slavujević	Broj indeksa:	EE 239/2015
Predmet:	Elektronika		
Mentor:	Doc. dr Vladimir Rajs		

TEMA DIPLOMSKOG RADA:

Detekcija znakova i praćenje traka pri kretanju autonomnog vozila u saobraćaju

TEKST ZADATKA :

1. Integrisati ROS okruženje u svoju mašinu, napraviti jedinstveni ROS paket i svoje radno okruženje.
2. Napraviti nod zadužen za prijem snimka i slanje frejmova dalje na obradu.
3. Realizovati detekciju stop znaka.
4. Realizovati detekciju i praćenje saobrajaćnih traka.
5. Realizovati detekciju ograničenja.
6. Napraviti nodove zadužene za svu zadatu detekciju, koji šalju frejm dalje na vizualizaciju, kao i obrađene podatke na ECU.
7. Napraviti "*Watchdog*" nod zadužen za kontrolu rada celog sistema.
8. Testirati rad sistema preko "*Unit*" testova.

Šef katedre:	Mentor rada:
dr Mirjana Damnjanović	dr Vladimir Rajs

Primerak za: o - Studenta; o - Mentora; o - Katedru; o - Studentsku službu fakulteta;

Sadržaj

Lista skraćenica

Spisak skraćenica na engleskom jeziku

ROS	Robot Operating System – Robotski operativni sistem
ECU	Electronic Control Unit – Elektronska kontrolna jedinica
CPU	Central Processor Unit – Centralna procesorska jedinica
OCR	Optical Character Reading – Optičko čitanje karaktera
STL	Standard Template Libraries – Standardna biblioteka obrazaca
ROI	Region of Interest – Region od interesa
CNN	Convolutional Neural Networks – Konvolucione neuronske mreže
GUI	Graphical User Interface – Grafički korisnički interfejs
OOP	Object Oriented Principle – Objektno orijentisani princip
PIMPL	Pointer to Implementation – Od pokazivača do implementacije
FPS	Frames per Second – Frejmovi u sekundi
ASCII	American Standard Code for Information Interchange – Američki standardni kod za razmenu informacija
QA	Quality Assurance – Osiguranje kvaliteta
GPU	Graphics Processing Unit – Grafička procesorska jedinica
RCNN	Region Convolution Neural Network – Regionalna konvoluciona neuronska mreža

1. Uvod

Kao što je poznato u svetskim okvirima, tendencije u daljem tehnološkom razvoju i razvitku u svim granama industrije jesu razvoj robotike i same veštačke inteligencije. S obzirom da se već godinama unazad napredovalo u razvitku robotike došlo se i do komercijalizacije i same upotrebe pojedinih veštački inteligentnih mašina u ljudskom okruženju. Jedna od ključnih ideja i primena robota u ljudskim životima jeste upotreba robotizovanih vozila tj. autonomnih vozila u saobraćaju. Evidentan rast kretanja ljudi prevoznim sredstvima, posledično povećani rizik od saobraćajnih nezgoda, povećan broj potrebnih resursa za sve veći broj vozila, povećane saobraćajne gužve, te znatno povećano vreme provedeno u saobraćaju itd. su neka od ključnih pitanja saobraćaja današnjice. [1]

Slika 1. Autonomno vozilo

Zadatak diplomskog rada jeste realizacija sotfvera koji će omogućiti autonomnom vozilu da samostalno prepoznaje saobraćajne znake i da prati saobraćajne trake kako bi se moglo samostalno kretati kroz saobraćaj. Ovo je realizovano pomoću *ROS*[2] platforme, sama obrada snimka je rađena pomoću *OpenCV* biblioteke, softver je pisan koristeći programski jezik C++, po C++ 17 standardima. Detekcija znakova rađena je pomoću konvolucionih neuronskih mreža, što spada pod Duboko mašinsko učenje (eng. *Deep Machine Learning*), a postupak je zapravo treniranje modela kojima jak procesor (*CPU*) omogućava procesuiranje velikog broja pozitivnih i negativnih slika koje omogućavaju modelu da razazna objekat.

2. ROS

ROS[2] (eng. *Robot Operating System*, robotski operativni sistem) je programsko okruženje koje pruža alatke i programske biblioteke koje omogućavaju korisniku da napravi aplikaciju za robota. Do skoro se striktno mogao koristiti u *Linux,* tačnije *Ubuntu* operativnom sistemu, dok su novije verzije (*ROS Melodic*), korišćene u ovom projektu, adaptabilne i u *Windows* operativnom sistemu.

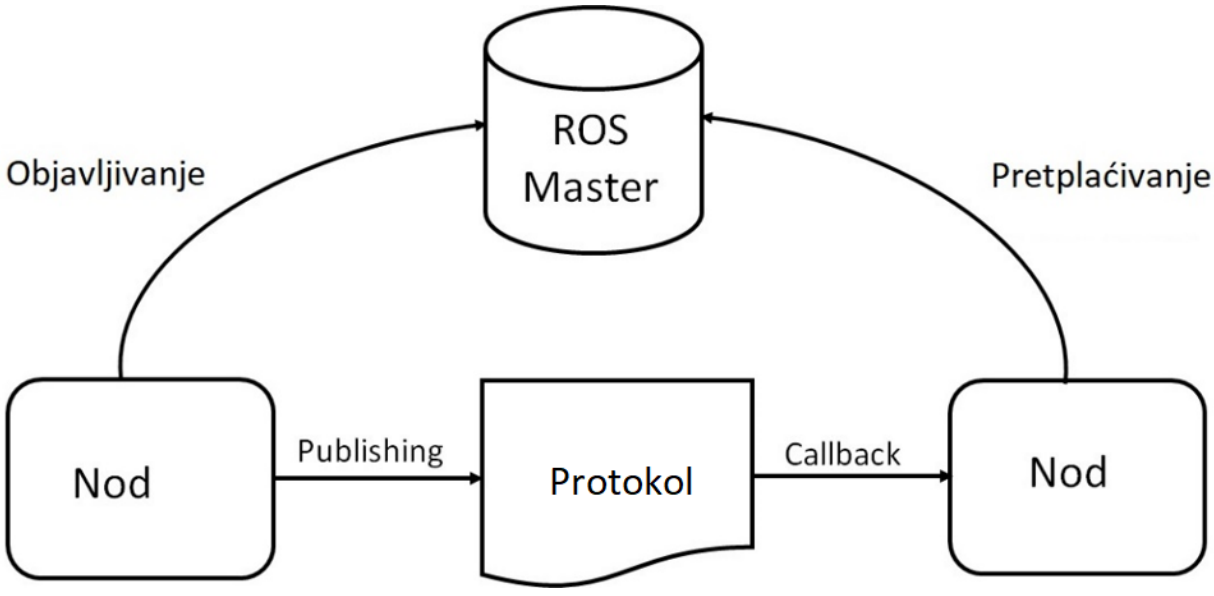
Slika 2. Logo ROS Melodic-a

ROS je operativni sistem otvorenog koda (eng. *Open Source*) u širem smislu te reči, što znači da poseduje otvorene biblioteke kojima se lako pristupa, ali pre svega, besplatne su. Program koji se izvršava može se pisati u C/C++ i/ili *Python* programskom jeziku. Odnosno, različiti nodovi mogu biti napisani na različitim programskim jezicima i da zajedno čine jedan sistem. *ROS* okruženje omogućava lako i jednostavno *build*-ovanje izvršnih fajlova komandom *catkin make*, a nod unikatnog naziva iz specifičnog programskog paketa se može pokrenuti komandom *rosrun*, uz pomoć *GNOME* terminala. Pored njih, postoji još komandi za rad u ovom okruženju, kao npr. za brzo pristupanje direktorijumima, za izlistavanje pokrenutih nodova, za grafičku vizualizaciju itd. [3]

2.1 Nod

Nodovi su delovi softvera koji mogu da realizuju različite zadatke u zavisnosti od toga na koji su način programirani. Nodovi mogu da procesuiraju podatke sa brojnih senzora, kao i da zadaju dalje izvršavanje komandi nekim aktuatorima, ali i da međusobno komuniciraju. Ta komunikacija među nodovima može da se izvršava na dva različita protokola; Glasnik-Pretplatnik (eng. *Publisher-Subscriber*) i Klijent-Server.

Prvi protokol podrazumeva takav vid komunikacije gde jedan nod može nešto da objavi (eng. *Publish*) na jedan protkol (eng. *Topic*), a drugi nod se može pretplatiti (eng. *Subscribe*) na taj protokol i preuzeti objavljene podatke. Nod radi u službi samoaktualizacije, gde će osluškivati podatke na koje se pretplatio i reagovati u skladu sa eventualnim izmenama, odnosno izvršava okidanje (eng. *Callback*) pri svakoj novoj publikovanoj poruci. Poruka može da sadrži različite tipove podataka pojedinačno, kao što je celobrojna vrednost (*integer*), decimalni broj (*float*), te niz karaktera (*string*), pa čak i slika ili zvuk. Takođe, poruka se može generisati i po narudžbini (eng. *Custom Message*), te može da kombinuje sve ove tipove. Treba naglasiti da ovaj protokol funkcioniše po klasičnom "gospodar-sluga" (eng. *Master-Slave*) principu, gde samo jedan nod može da objavljuje poruku u jednom trenutku, a ostali nodovi mogu da je primaju, pri čemu broj pretplatnih nodova nije ograničen. Drugi protokol (Klijent-Server) funkcioniše tako da nod koji igra ulogu klijenta šalje podatke ka serveru, a na osnovu njih server nod vraća povratnu informaciju ka njemu. [4]

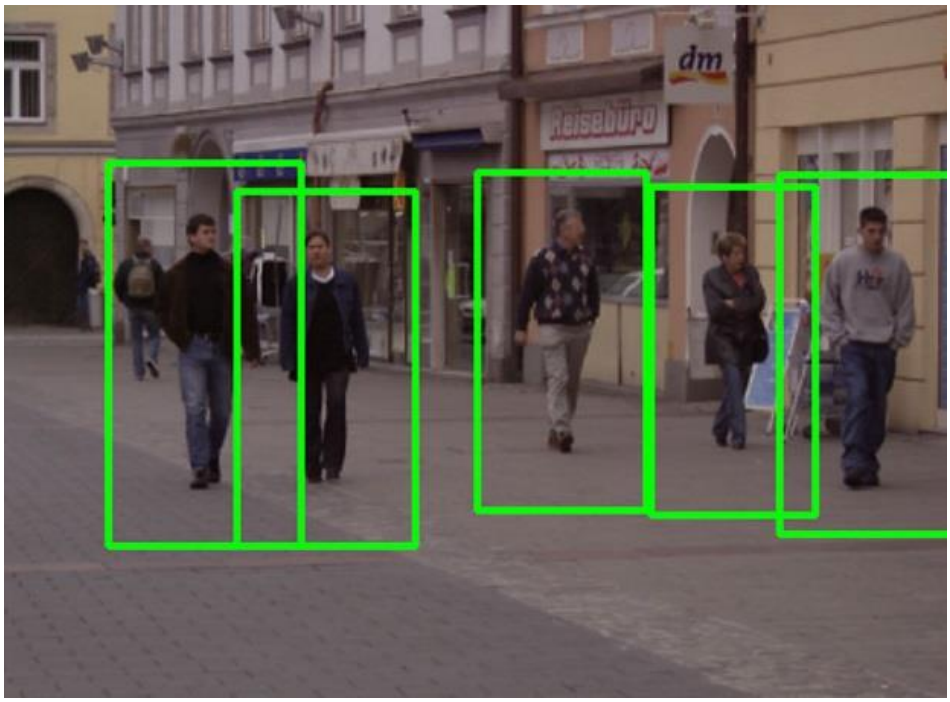


Slika 3. Blok dijagram Glasnik-Pretplatnik protokola

3. Alati za detekciju

3.1 OpenCV

OpenCV[5] predstavlja kolekciju biblioteka pisanih u C/C++, *Python* i Java programskom jeziku koje se prvenstveno koriste za kompjutersku viziju u realnom vremenu. Prvobitno ju je razvila kompanija *Intel*, a danas je dostupna kao softver otvorenog koda (eng. *Open source*). Neke od primena ovih biblioteka uključuju: segmentaciju, praćenje pokreta, prepoznavanje lica, prepoznavanje gestova, pomeranje kamera unutar 3D prostora (eng. *Egomotion*), ali i prepoznavanje objekata pomoću treniranih neuronskih mreža.



Slika 4. Primer detekcije objekata pomoću OpenCV platforme

3.2 Tesseract

Tesseract[6] je platforma za optičko prepoznavanje karaktera (eng. *Optical Character Recognition, OCR*) za razne operativne sisteme. U pitanju je besplatan softver čiji je razvoj sponzorisan od strane *Google*-a još od 2006. Smatra se jednim od najtačnijih *OCR* platformi koje su otvorenog koda (eng. *Open source*). Većina njegovog koda napisana je u programskom jeziku C, a ostatak u C++, s tim što je posle ceo kod prepisan da bi se makar mogao kompajlirati C++ kompajlerom. *OCR* predstavlja elektronsku ili mehaničku konverziju slika kucanog, ručno pisanog ili štampanog teksta u mašinski enkodovan tekst, bilo sa skeniranog dokumenta, fotografije dokumenta ili fotografije prirodnog okruženja sa znakovima bilbordima i slično. *OCR* je značajno polje istraživanja za prepoznavanje obrazaca, veštačku inteligenciju i kompjutersku viziju.

4. Detekcija zadatih objekata

U sklopu izrade ovog diplomskog rada traženo je da izrađeni program detektuje pojedine objekte u saobraćaju, konkretno linije saobraćajnih traka unutar kojih se vozilo nalazi, STOP znakove i znakove ograničenja brzine.

4.1 Detekcija STOP znaka

Stop znak je jedan od specifičnih saobraćajnih znakova. U pitanju je, dakle, osmougaona tabla na kojoj je belim slovima ispisano STOP sa crvenom podlogom. Postoje različite mogućnosti i pristupi kako bi se mogla odraditi detekcija STOP znaka. Pre svega, najjednostavniji pristup bio bi prepoznati znak na osnovu njegove boje i oblika. Crvena boja je dominantna i lako opaziva, te bi prvi korak u obradi slike bio segmentacija crvene pravljenjem binarne maske koja za prag (eng. *threshold*) uzima nijanse crvene, dok ostali pikseli imaju vrednost logičke nule.

Slika 5. Segmentacija crvene boje

Dobijenu maska se filtrira, kako bi se uklonili šumovi, zamagljivanjem (eng. *blurring*) preko Gausove raspodele. Nakon toga *OpenCV* pruža mogućnost da se sakupe pikseli u okviru strukture *cv::Point* kao niz tačaka. U C++ programskom jeziku. Najpogodnija za ovaj postupak je upotreba *STL* (eng. *Standard Template Libraries,* Standardne biblioteke obrazaca)[7] vektora, koji se ponašaju kao dinamički alocirani nizovi, te nije potrebno prethodno definisati

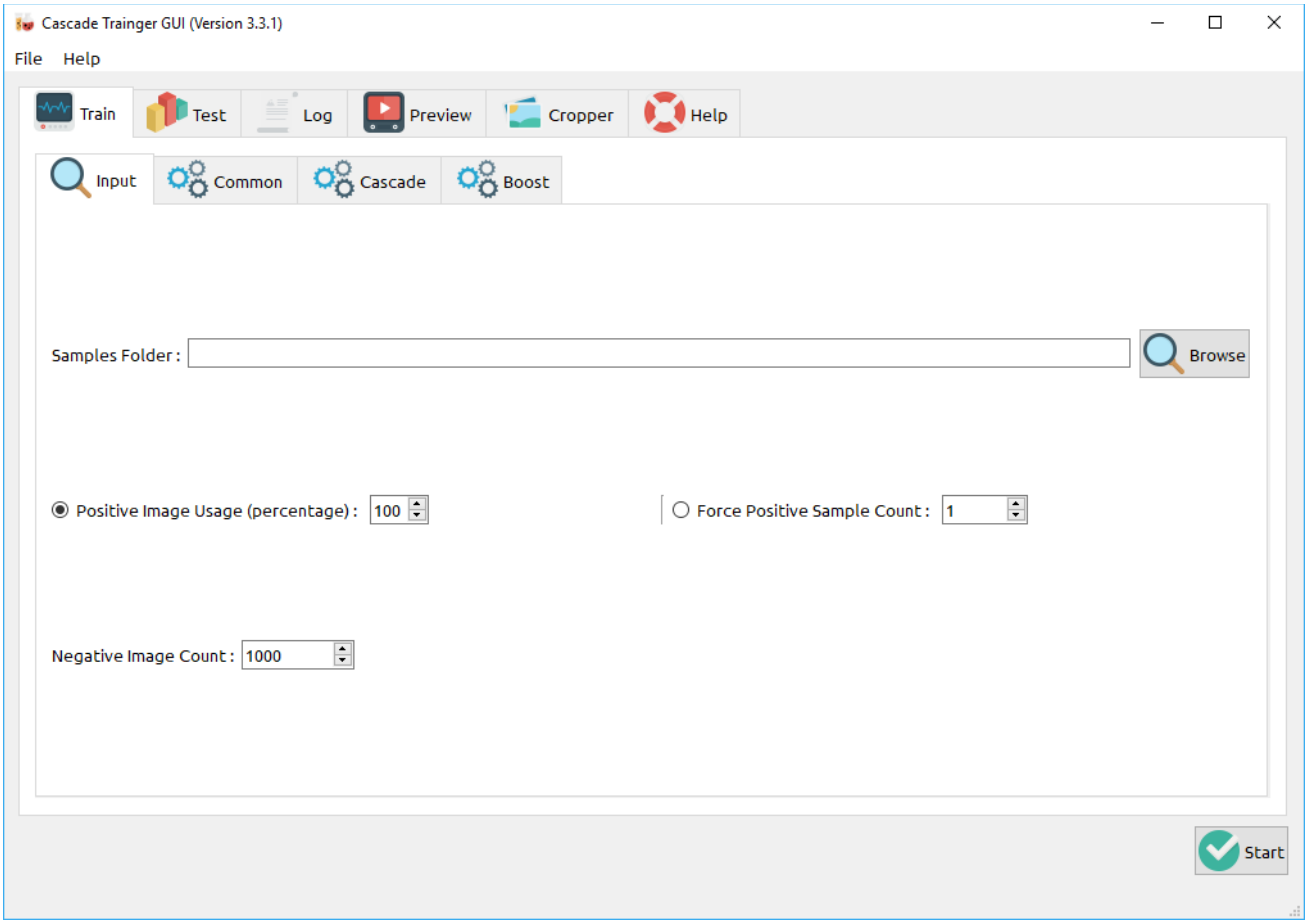
opseg niza. Odabir kontura od interesa se može vršiti na osnovu površine, kao i oblika konture. Kontura mora da poseduje osam stranica i da bude pravilnog oblika, odnosno da su sve stranice jednake i svi uglovi jednaki. Nakon takvog odabira, markiranjem preostalih kontura dobija se vizuelna detekcija znaka.

Slika 6. Blurovanje maske i markiranje odabrane konture

Postavlja se pitanje da li je ovo dobar način za detekciju. Ovaj postupak dobar je za detekciju statičkih objekata, ali pokazuje izvesne nedostatke kada se radi o detekciji pri pokretu i u realnom vremenu iz više razloga. Osim što se može desiti da kamera zbog svoje pozicije na vozilu i perspektive u odnosu na znak nikada ne vidi pravilni osmougao, takođe, teoretski bi se moglo dogoditi da pored znaka u saobraćaju susretne i još neki pravilni osmougaoni objekat crvene boje (npr. reklama na bilbordu), ili da zbog uslova osvetljenja kamera zabeleži frejm gde pikseli na znaku neće proći binarnu masku. Ovi problemi zahtevaju drugačiji pristup.

Poslednjih godina za rešavanje problema detekcije koriste se neruonske mreže, tačnije konvolucione neuronske mreže (eng. *Convolutional Neural Network, CNN*). *OpenCV* omogućava treniranje neuronskih mreža na jednostavan način. Potrebno je obezbediti slike objekta od interesa, u ovom slučaju STOP znaka, koji se nazivaju pozitivi, kao i negativa, a to je veliki broj slika u kojima se ne nalazi objekat od interesa. Potrebno je napraviti tabele oba poddirektorijuma gde se nalaze imena slika, a za positive mora biti naznačen region od interesa

(eng. *Region of Interest, ROI*), odnosno početne coordinate pravougaonika u kojima se nalazi objekat, kao i širina i dužina tog pravougaonika. Postoji mogućnost da se unutar istog pozitiva nađe više objekata od interesa, u kom slučaju se procedura ponavlja za svaki od objekata kao novi red u koloni tabele. Postupak treniranja je značajno olakšan zahvaljujući *Windows* aplikaciji *Cascade Trainer GUI*. U pitanju je program koji poseduje grafički interfejs za isecanje objekata unutar slika ili snimaka, koji pri pokretanju treninga sam generiše sve potrebne fajlove, nakon čega započinje trening. Trening može da bude gotov za nekoliko sati, ali može da traje i nekoliko dana, u zavisnosti od broja slika koje se stavljaju na raspolaganje mreži, ali i od performansi mašine na kojoj se trening vrši.

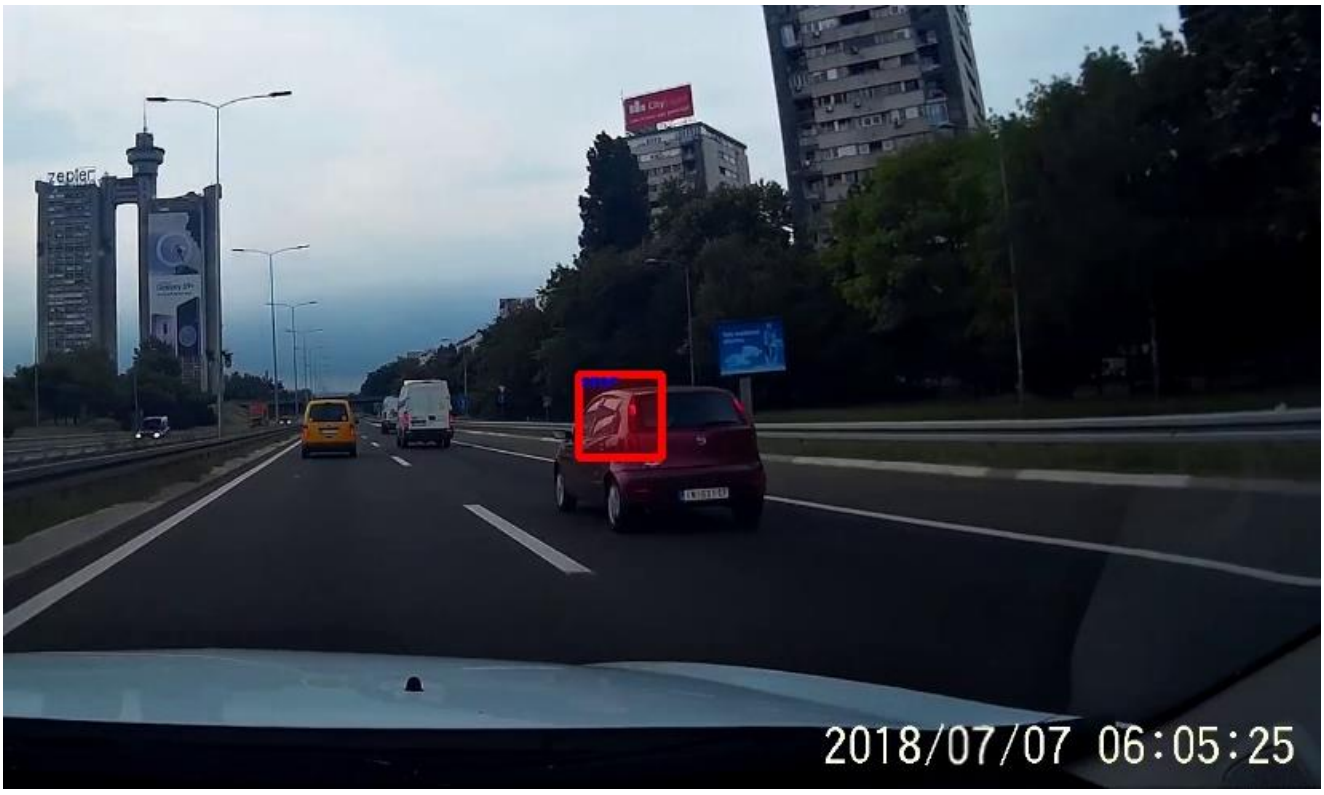


Slika 7. Cascade Trainer GUI programa

Za detekciju STOP znaka u ovom projektu korišten je set slika nemačkih saobraćajnih znakova sa internet stranice Institua za Neuroinformatiku Savezne Republike Nemačke. U pitanju je besplatan set slika specijalizovano namenjenih za tu svrhu. Korišćeno je oko 600 pozitivnih slika STOP znaka i oko 8000 negativa, a trening je trajao preko 10 časova. *OpenCV* za obradu koristi *CPU* računara, a nakon završetka treninga program konačno kreira fajl sa

.xml ekstenzijom. Od svih generisanih fajlova ovaj fajl je jedini značajan fajl. Fajl se uključuje u projekat pomoću *OpenCV* strukture *cv::CascadeClassifier*, tako što se pri instanciranju objekta njemu kao argument prosledi apsolutna putanja lokacije na kojoj se fajl nalazi. Ukoliko je fajl ispravno generisan, nakon njegovog uključivanja u projekat, instanca ove strukture može dalje nesmetano da obavlja detekciju objekata na snimku. Koordinate svakog objekta kojeg neuronska mreža smatra objektom od interesa bivaju pohranjene u vektor *ROI*, što se posle može iskoristiti za vizuelno markiranje objekta na frejmu.

Ukoliko se neuronska mreža koristi kao takva bez prethodne obrade u vidu smernica koje bi joj nagovestile na šta da se fokusira, dolazi do pregršt problema. Neuronske mreže su još uvek u povoju i kao takve nisu idealne, te dolazi do grešaka u detekciji, koje i dalje nisu u potpunosti objašnjene. Takođe, dolazi do drastičnog usporavanja u radu sistema kada neuronska mreža obrađuje celu sliku, odnosno frejm.



Slika 8. Primer pogrešne detekcije STOP znaka

Zbog ovih problema neophodno je da se pre rada *CNN*-a odradi predprocesuiranje frejma. Mnogo bolji rezultati se postižu kombinovanjem dveju navedenih metoda. Opseg pretrage sužava se prvo uz pomoć binarne maske, zatim sledi odabiranje kontura, gde su kriterijumi za odabiranje obim i površina objekta, kako bi se izbegla eventualna detekcija

objekata koji po ovim parametrima ne odgovaraju saobraćajnim znacima, već mogu biti potpuno nasumični objekti na slici. Tek onda se vektori ovih kontura prosleđuju neuronskoj mreži na detekciju. Ovim se značajno povećava tačnost detektovanja i povećavaju se performanse programa, te ne dolazi do usporavanja zbog preopterećenja procesora. Međutim, i dalje može doći do greške u detekciji pri čemu se detektuju objekti koji su prošli odabir konture, ali nisu STOP znak niti po gabaritima mogu odgovarati STOP znaku (npr. veliki crveni kamion, stop svetlo vozila i sl.). Ovo je posledica pre svega nepostojanja univerznalne norme za parametar odabiranja, jer se objekti tokom kretanja vozila snimaju iz različitih uglova. Zbog ovoga, poželjno je napraviti još jedno predprocesuiranje u vidu binarne maske, koja dodatno sužava opseg detekcije isecajući delove frejma koji nisu od interesa, jer se u tom regionu nikada neće naći STOP znak. Takav region se pre svega odnosi na nebo i na objekte koji su udaljeni od saobraćajnice. Više reči o ovome biće u poglavlju gde je objašnjena detekcija ograničenja brzine.

Posle dodatnog predprocesuiranja, kao garant ispravne detekcije STOP znaka, uvodi se *OCR.* Uvođenje *OCR*-a iziskuje dodatno postprocesuiranje. Zadatak *OCR*-a jeste da na svim objektima od interesa koji su prošli prethodni postupak detektuje bar tri od četiri slova reči STOP. Optičko prepoznavanje karaktera ima izuzetno visoku uspešnost kod čitanja karaktera skeniranog ili uslikanog teksutalnog dokumenta, odnosno svetlog platna sa crnim slovima, ali se problem usložnjava kada treba detektovati karaktere na frejmu video snimka zbog nepostojanja odgovarajućeg kontrasta između pozadine i napisanog teksta, ali i pojave kontura koje svojim oblikom mogu zavarati *OCR* da pomisli da se radi o slovima iako to nije slučaj. Jedan primer za ovo jeste osmougaona kontura STOP znaka. Dakle, neophodno je detektovati mesto na kojem se nalazi tekst pre nego što se uradi optičko prepoznavanje karaktera ili sliku obraditi tako da se na prepoznavanje pošalju samo odgovarajući delovi frejma.

Slika 9. STOP pre i nakon procesuiranja

Kombinovanjem navedenih pristupa ostvaruje se zadovoljavajuća tačnost pri detektovanju objekata od interesa, u ovom slučaju stop znaka.

Slika 10. Konačna detekcija STOP znakova

4.2 Detekcija i prepoznavanje ograničenja

Pored STOP znakova, jedan od značajnih elemenata u realizaciji ovog zadatka bila je detekcija znakova ograničenja brzine. U početku je realizovana samo detekcija znakova sa ograničenjem brzine od 80km/h, da bi zatim dograđivanjem *OCR*-a ovo bilo prošireno na sve znake ograničenja brzine koji su zakonom propisani u Republici Srbiji.

Prvobitni pokušaj da se detektuje znak ograničenja brzine sastojao se iz primene binarnih maski, sličnih onima koje su korišćene za detekciju STOP znaka, koje izdvajaju crvenu boju, karakterističnu za obod znakova ograničenja brzine. Za razliku od STOP znaka, u kojem se koristio *cv::findContours* koji će pokupiti sve konture, u okviru ove detekcije tražene su isključivo kružne konture koje se vide u binarnoj maski, što se izvršava zahvaljujući *cv::HoughCircles*. Kružne konture skladište se u vektorima pointa (*cv::Point*). Ova procedura detektovaće bilo koji kružni oblik koji preostane nakon primene binarne maske koja izdvaja crvene konture. Rezultat ovakve procedure biće sledeći:

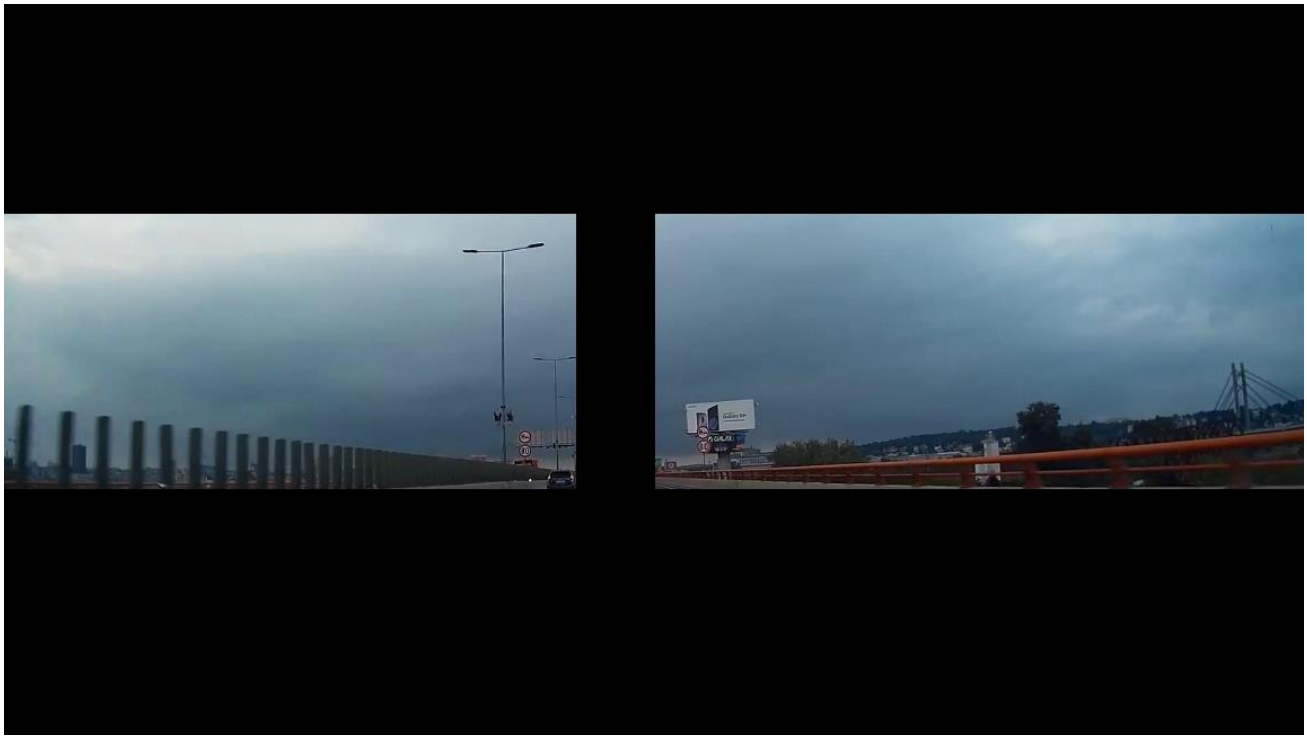
Slika 11. Pogrešna detekcija kruga koji nije ograničenje

Ovakav postupak ne daje očekivane rezultate zbog velikog broja detekcija znakova koji svojim oblikom i konturama podsećaju na znakove ograničenja brzine, na primer znak zabrane preticanja koji takođe ima oblik kruga sa crvenim obodom. Ovaj problem je očevidan, za razliku

od detekcije STOP znaka, gde zbog svoje specifičnosti ovakva anomalija ne bi došla do izražaja, zbog toga što STOP znak ima jedinstvene konture koje nisu svojstvene ostalim saobraćajnim znacima. Kako znaci ograničenja brzine pripadaju skupu znakova izričitih naredbi, oni svoj oblik i boju svoje konture dele sa mnogim drugim saobraćajnim znacima, te ovaj postupak ne bi bio dovoljno dobar u praksi. Pored ovog očiglednog nedostatka, kao i kod STOP znaka, može doći do toga da u realnim uslovima saobraćaja senzor koji vrši detekciju okruženja (kamera) neće zabeležiti savršen krug jer ga snima iz različitih uglova koji zavise od njene pozicije kao i pozicije automobila koji se kreće kroz saobraćaj.

Imajući sve ovo u vidu, bilo je porebno predložiti novo rešenje koje bi eliminisalo uočene manjkavosti. Postupak detekcije ograničenja principijelno se svodi na isti postupak kao pri detekciji STOP znaka, gde ključnu ulogu igraju klasifikatori (eng. *Classifier*).

Postupak koji nije detaljnije opisan pri detekciji STOP znaka je korišćen u oba slučaja, a pored segmentacije crvene boje, podrazumeva i kreiranje maske koja ignoriše određene regione frejma koji nisu od interesa. Naime, saobraćajni znaci koje je potrebno detektovati se po prirodi nalaze samo u određenom delu frejmova koje kamera zabeleži. Da bi se olakšao proces detekcije, neophodno je suziti prostor na kojem se traga za objektima od interesa na onaj prostor gde se ti objekti zapravo i mogu nalaziti, umesto na ceo frejm gde se mogu naći raznorazni drugi objekti koji po svom izgledu ili konturama podsećaju na saobraćajne znakove, ali to nužno ne mora biti slučaj.

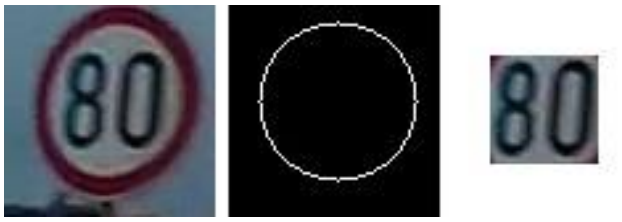


Slika 12. Prikaz regiona od interesa

Kao i u slučaju STOP znaka, segmentacija crvene boje predstavlja sledeći postupak, a zatim sleduje odabiranje kontura unutar binarne slike koje se dalje prosleđuju ka klasifikatoru (*cv::Classifier*). Trening set klasifikatora morao je sadržati veliku količinu znakova ograničenja različitih vrednosti, dok su negativi pored sve okoline koje imaju morali da sadrže i izvestan broj negativa koji su sličnog oblika i boje. Ovo znači da je u negativu sadržano mnoštvo znakova ograničenja koji su morali biti crvene boje oboda i bele boje unutrašnjosti, kao na primer znak za ograničenje međuosovinskog opterećenja, za ograničenje visine, znak za zabranu preticanja, znak za zabranu skretanja i sl. Ovaj trening, koji je takođe vršen zahvaljujući istom programskom interfejsu, se izvršavao nešto malo duže, naime gotov klasifikator koji je prošao kroz svih 20 koraka (eng. *stage*) dobijen je za preko dva dana treniranja neuronske mreže. Otežavajuća okolnost u ovom slučaju bila je da bi trening potrajao i po 10 sati, zatim detektovao neku grešku, pa je problem bilo neophodno ukloniti, a ceo postupak raditi ispočetka.

Slika 13. Pogrešna detekcija znaka ograničenja preko klasifikatora

Kao i slučaju kod STOP znaka i dalje dolazi do pogrešne detekcije, što je posledica nesavršenosti neuronskih mreža. U postupak rešavanja morao se ponovo pridodati *OCR* kao dokaz da unutar detektovane površine postoje brojevi koji reprezentuju ograničenje brzine. Postupak ponovo nije mogao biti direktan, što znači da nakon detekcije pomoću klasifikatora isti taj isečak nije mogao biti poslat na *OCR* već se moralo raditi predprocesuiranje kako bi isečak imao čisto crno beli tekst.



Slika 14. Proces pripreme isečka za OCR

Nakon očitavanja teksta, isti se skladišti u string koji može biti iz C++ standardne biblioteke (*std::string*) ili običan niz karaktera (*char[]* ili *const char**). Iako nije detaljnije opisan u prethodnom poglavlju, ovaj postupak se koristi i tokom detekcije STOP znaka, te će mu ovde biti posvećena veća pažnja. Naime, očitavanje stringa sa slike ponekad može rezultovati u nizu karaktera koji pored brojeva koji signaliziraju dozvoljenu brzinu sadrži i neke neželjene

karaktere pa se tom slučaju radi komparacija stringova, odnosno traži se koristan deo stringa u očitanom stringu, pa ako dođe do podudarnosti smatra se da je detektovano i uspešno očitano ograničenje brzine. Ovaj podatak dalje se prosleđuje ka potencijalnom *ECU* autonomnog vozila (u našem slučaju simulatorskog noda koji igra tu ulogu).

Slika 15. Izgled konačne detekcije ograničenja brzine

4.3 Detekcija linija saobraćajne trake

Za razliku od do sada navedenih postupaka čija je svrha bila detekcija saobraćajnih znakova, u ovom slučaju bilo je potrebno detektovati bele linije iscrtane na putu kako bi vozilo znalo u kojoj se saobraćajnoj traci nalazi. Ovakav postupak traži potpuno drugačiji pristup. Prvenstvena razlika se nalazi u tome da se detekciju linija ne može vršiti uz pomoć *OCR*-a i klasifikatora. Potrebno je konstantno pratiti dve linije koje ograničavaju traku kojom se vozilo kreće. Konzistentnost praćenja linija je veoma bitna u toku vožnje. Praćenje linije vozilu govori koliko treba eventualno da skrene kako bi usled zakrivljenja puta ostalo u svojoj saobraćajnoj traci.

Slika 16. Linije na putu

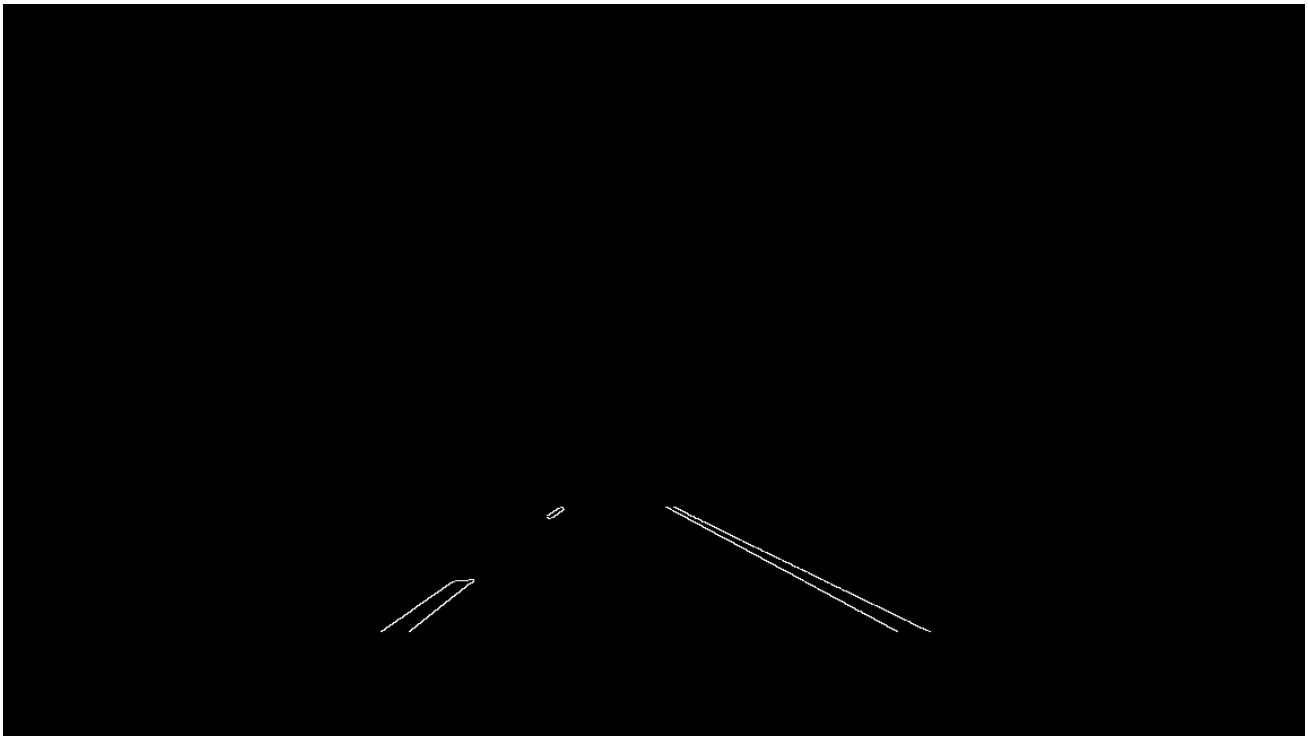
Postavlja se pitanje kako ovom problemu pristupiti. Analiza počinje konstatacijom da linije mogu biti bele boje ili žute, ukoliko se radi o traci za javni gradski saobraćajni prevoz ili o privremenim trakama usled radova na putu. Linije se mogu pojaviti na tačno određenom mestu na slici (frejmu) koju pre svega odlučuje pozicija kamere u odnosu na vozilo. Linije mogu biti isprekidane, pune, pa čak ih ne mora ni biti, što vozilo treba da detektuje.

Prvi postupak mogao bi biti segmentacija boje, u ovom slučaju boje od interesa su bela i žuta. Kako je teško napraviti binarnu masku koja bi radila savršenu segmentaciju samo te dve boje, odnosno tih linija, od tog postupka se odustalo.

Prvi korak zapravo je bio ograničavanje frejma na region od interesa. Ovaj region je izuzetno specifičan, varira od slučaja do slučaja, te maska umnogome zavisi od pozicije kamere unutar vozila. Da bi tako nešto postalo generično, a ne unapred definisano, neophodno je pored same detekcije linija napraviti i svojevrsnu kalibraciju kamere kako bi se ona sama orijentisala i fokusirala na pravi region od značaja. Ovaj postupak nije implementiran u sistem, prvenstveno jer nije bio zadatak projekta. Dakle, referentna tačka, koordinate regiona od interesa, a time i oblik konture i njena površina na frejmu, mora biti unapred definisana. Region od značaja predstavljao je samo deo puta na kojem se vozilo nalazi.

Slika 17. Region od interesa detekcije linija

Nakon toga, sliku je trebalo procesuirati pre samog detektovanja linije. Postupak transformacije koji je primenjen je takođe iz *OpenCV*-a. U pitanju je *cv::CannyEdge*. Ova funkcija zamagljenu (eng. *blur*) sliku pretvara u sliku u kojoj se vide samo ivice objekata.



Slika 18. Canny Edge transformacija slike

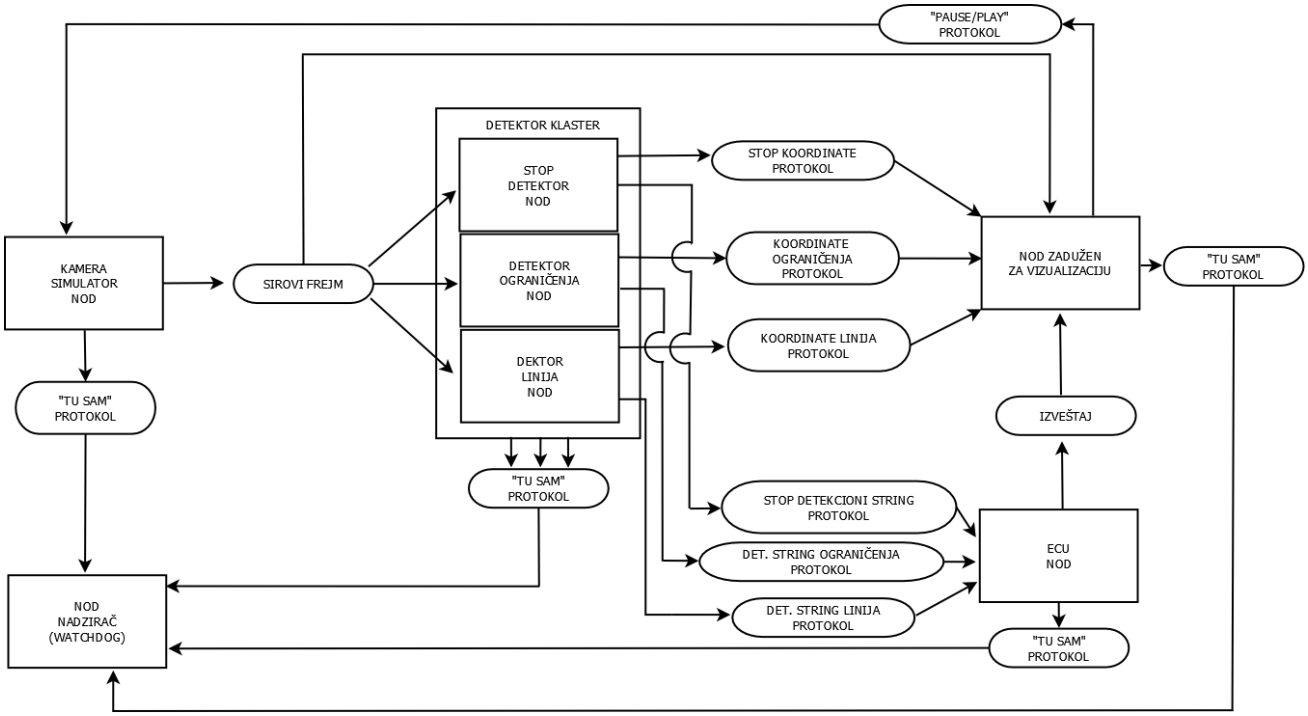
Sledeći korak je sakupljanje detektovanih linija u vektor vektora tačaka (niz linija). Sakupljanje linija slično je sakupljanju krugova u detekciji ograničenja, s tim što se prikupljanje krugova radi preko funkcije *cv::HoughCircles*, dok se sakupljanje linija radi uz pomoć *cv::HoughLines*. Sakupljene linije se procesuiraju odabiranjem linija koje pripadaju levoj i koje pripadaju desnoj (podela u dve grupe vektora) strani regiona od interesa. Ova dva podregiona dalje biraju koje od sakupljenih linija mogu predstavljati liniju trake puta tako što se računa nagib svake od linija, koji mora da bude iznad 0.7 po apsolutnoj vrednosti, gde 1 predstavlja savršeno vertikalnu liniju. Ovim putem se izbegava detekcija horizontalnih linija koje bi se iz različitih razloga mogle naći na putu (npr. razgraničenje dva smera raskrsnice kod semafora).

Ovako odabrane linije dalje omogućuju aproksimaciju konačnih jedinstvenih linija sa leve i desne strane, koje određuju saobraćajnu traku. Kako je linija nekada isprekidana sa jedne od strana, može doći do nagle promene nagiba aproksimirane linije, što je vid greške koji se javlja pri ovom slučaju. Međutim, pored potencijalnog unapređenja, mogu se upotrebiti različiti softverski alati kako bi se rešio problem koji nastaje kada je linija isprekidana. Npr. napraviti algoritam očekivanja, gde ukoliko se između frejmova pojavi jedan frejm u kojem nagib naglo odskoči, dolazi do ignorisanja istog frejma, te će vozilo nastaviti nesmetano da se kreće napred.

Slika 19. Konačna vizualizacija detekcije saobraćajnih traka

5. Integracija celog sistema

Nakon što su pobrojani i objašnjeni individualni procesi koji se odvijaju tokom rada sistema, potrebno je prikazati kako su ovi, ali i drugi procesi koji se odvijaju u pozadini, povezani i uspešno integrisani u zajednički sistem. Ceo sistem se sastoji iz sedam nodova, od toga tri noda čine jedan klaster (eng. *Detection Cluster*, nodovi za detekciju), čiji rad je dopunjen preostalim nodovima: Simulator Kamere (eng. *Camera Simulator*), *ECU* Simulator, Nod zadužen za vizualizaciju (eng. *Visualizer*) i Nadzirač (eng. *Watchdog*[1]).



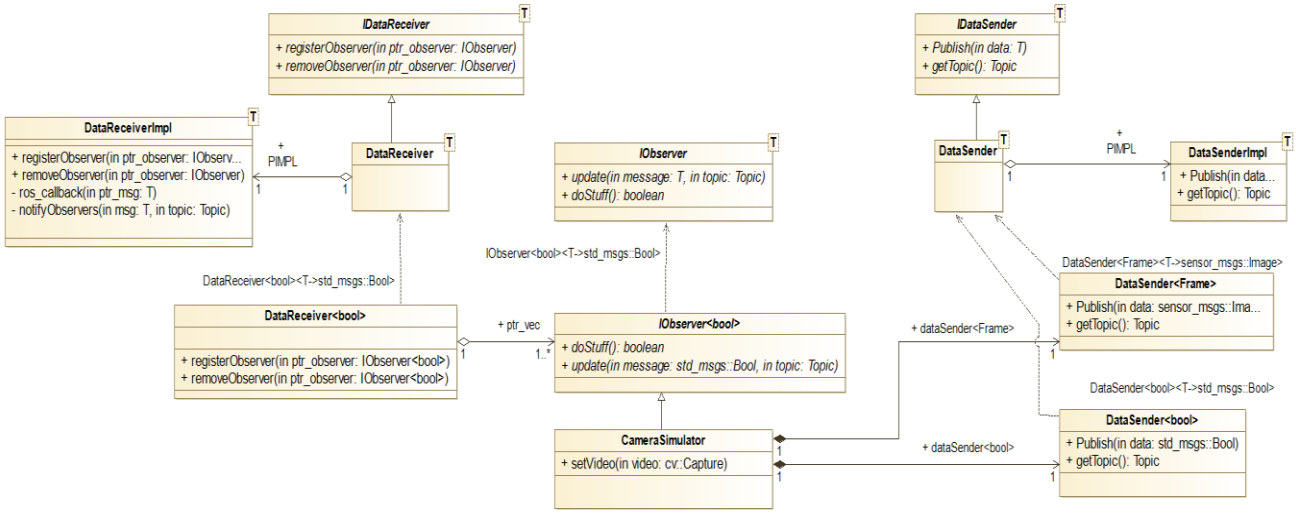
Slika 20. Relacije između nodova

[1] Bukvalan prevod naziva *Watchdog* bio bi „pas čuvar“, međutim upotrebljen je termin „nadzirač“ jer se radi o nodu koji nadzire rad celog sistema i reaguje ukoliko dođe do nepravilnosti

5.1 Simulator Kamere

Ovaj nod je, kako mu samo ime govori, zadužen za simuliranje rada kamere koja je u realnim uslovima pričvršćena na prednji deo automobila. Simulacija se obavlja tako što se nodu umesto direktnog video strima vožnje prosleđuje snimak koji je kamera zabeležila u realnim uslovima saobraćaja. Nod se pokreće kao i bilo koji program na *Linux*-u, s tim što mu se kao argument prosleđuje i apsolutna putanja na kojoj se nalazi video. String putanje videa se prosleđuje *OpenCV* strukturi *cv::Capture* čija instanca je zadužena da video šalje ka sledećem nodu “frejm po frejm” pri svakoj iteraciji (ciklusu).

ROS platforma je kompatibilna sa *OpenCV*-jem, te je publikovanje frejma na protokol obavljeno uz pomoć *cv_bridge*-a, koji predstavlja konvertor *OpenCV* instance frejma (*cv::Mat*) u standardnu *ROS* poruku *sensor_msgs.* Aplikacija (nod) je takođe zadužena za proveru da li je video zapis na navedenoj adresi, kao i da li je odgovarajućeg formata (bilo koji video format).



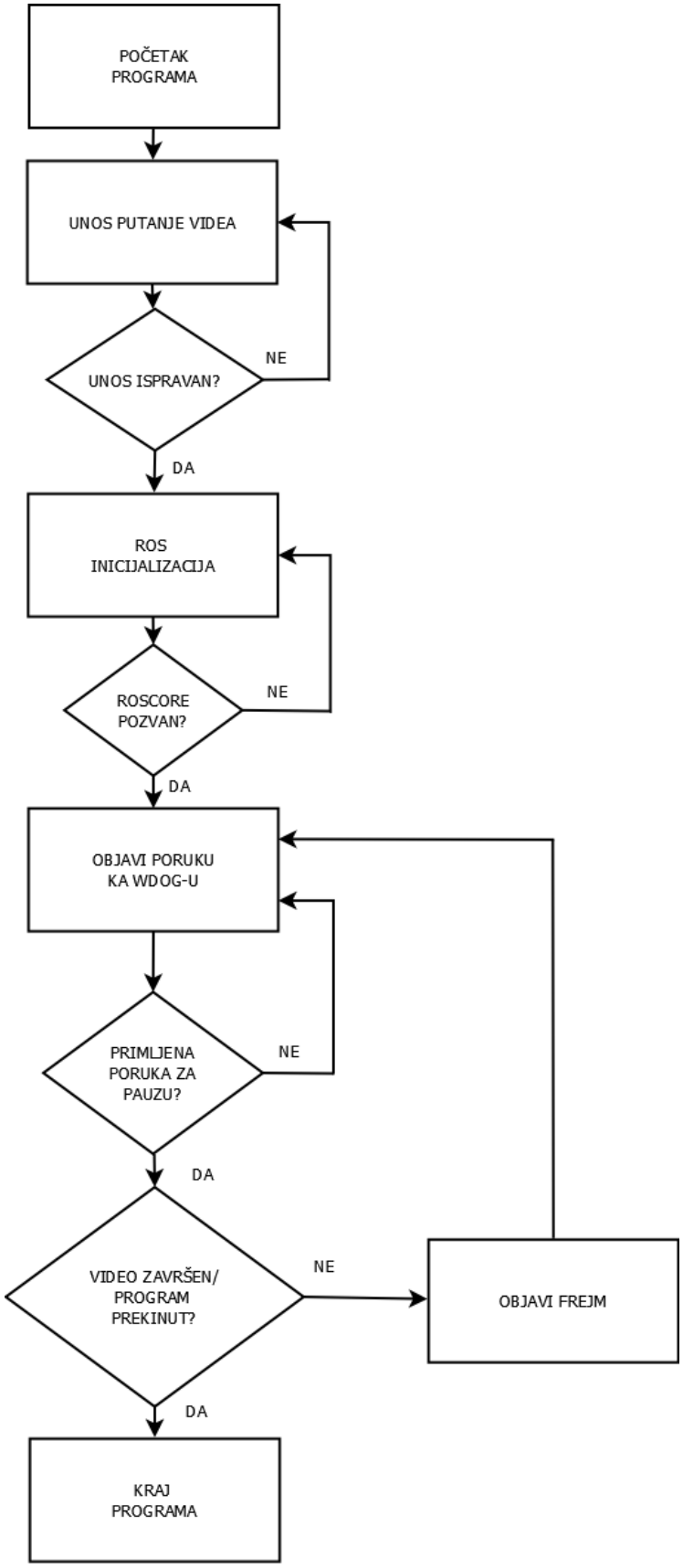
Slika 21. Klasni dijagram noda Simulatora Kamere

Ovaj nod funkcioniše koristeći klase: *Camera Simulator*, *Data Sender* i *Data Receiver*. Klasa *Camera Simulator* nasleđuje *IObserver* klasu koja predstavlja interfejs apstraktnog tipa koji prima jednu ili više poruka. U pitanju je obrazac (eng. *template*) klasa koja nema unapred deklarisan tip promenljive koja joj se prosleđuje, što zapravo omogućava njeno funkcionisanje sa različitim spektrom ulaznih promenljivih, umesto da se piše posebna klasa za svaki tip. Ovo je izuzetno korisna mogućnost u objektno orijentisanoj implementaciji sistema. Klasa *Camera Simulator* u ovom slučaju nasleđuje samo *bool* tip *Observer*-a, jer prima poruku samo od noda

za vizualizaciju. Ova klasa povezana je pomoću već postojećeg i veoma poznatog Posmatrač šablona (eng. *Observer Pattern*) sa *Data Receiver* klasom.

Data Receiver je zapravo zadužena za prijem poruke od noda zaduženog za Vizualizaciju i unutar sebe ima implementiranu *ROS* arhitekturu zaduženu za prijem poruke, što je zamaskirano preko *PIMPL Idiom*-a. Ovo je još jedna u nizu izuzetno korisnih mogućnosti u *OOP*, gde zavisnost kreiranog sistema od platforme na kojoj je realizovan ne dolazi do izražaja, jer se uz minimalne promene može omogućiti rad sistema na nekoj drugoj platformi. Cela logika ostaje gotovo netaknuta, a neophodno je promeniti dva *source* fajla. Postupak je takav da je unutar ove klase deklarisana još jedna klasa unutar *header* fajla, celokupna definicija potklase je napisana u *.cpp* fajlu, a potklasom se upravlja pomoću pokazivača na nju, člana glavne klase. Dakle, *Data Receiver* prima poruku pomoću potklase, a pomoću *Observer Patterna* ažurira promene u *Camera Simulator*-u.

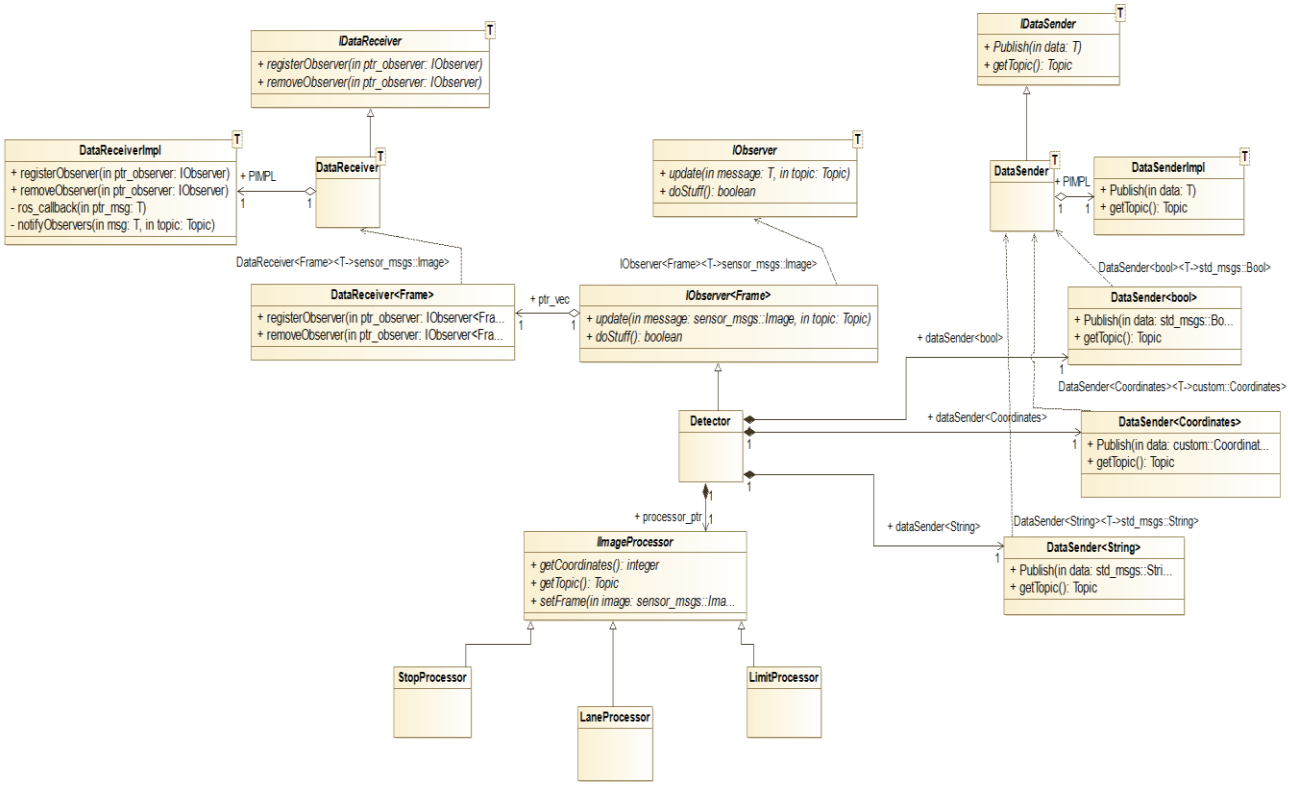
Data Sender je klasa zadužena za publikovanje željenih podataka ka drugim nodovima. Kao i prethodne dve klase, u pitanju je *template* klasa, dok je *ROS* platforma zamaskirana *PIMPL*-om i u ovom slučaju. Implementirana je kao kompozicija *Camera Simulator*-a, koji je koristi u dve svrhe (dva zasebna člana), gde je prvi zadužen za slanje *bool* poruke ka Nadziračkom (*Watchdog*) nodu, a drugi je zadužen za publikovanje samog frejma.



Slika 22. Algoritam rada noda Simulatora Kamere

5.2 Klaster Detektora

Klaster se sastoji iz tri noda koji se zbog sličnog načina rada i implementacije mogu opisati u sklopu istog poglavlja. Principski, ovaj deo zadatka je mogao biti realizovan u okviru jednog noda koji bi sekvencijalno obavljao sve operacije detekcije, međutim ovaj način realizacije bi značajno usporio rad sistema, pre svega zbog *OCR*-a i klasifikatora, te je radi bolje optimizacije rada sistema odlučeno da se ovaj deo sistema izdeli na tri podsistema (noda) koji će paralelno obavljati zadate funkcije. Ova tri noda poseduju identičnu arhitekturu, a razlika predstavlja klasa za obradu slike, kao i prosleđivanje parametara u vidu protokola na koji se obrađeni podaci dalje šalju.

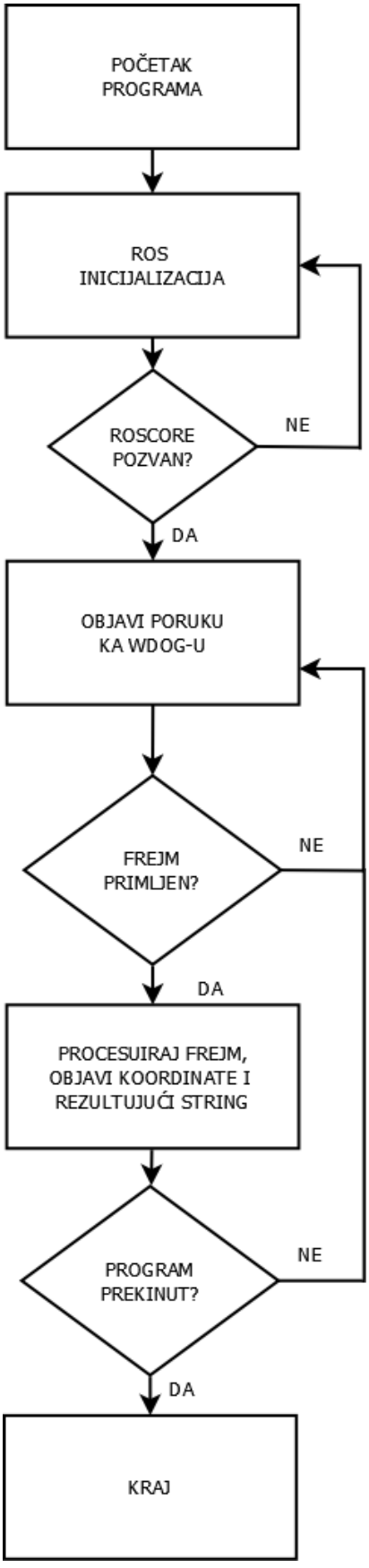


Slika 23. Klasni dijagram Detektor nodova

Klaster čine nodovi za detekciju STOP znaka, ograničenja, kao i linija saobraćajne trake. Sva tri noda poseduju sledeće klase: *Data Receiver*, *Data Sender*, *Detector*, kao i *Image Processor*. *Data Receiver* i *Data Sender*, su već opisani u sklopu prethodnog noda, te im ovde neće biti posvećena dodatna pažnja, jer je njihova uloga u okviru ovog noda ista.

Data Receiver, prima frejm i prosleđuje *Observer*-u (u ovom slučaju ga nasleđuje *Detector*), dok jedan *Data Sender* šalje poruke ka nodu zaduženom za vizualizaciju, drugi ka *ECU*, a treći ka *Watchdog*-u. *Watchdog* i u ovom slučaju prima *bool* poruku, *ECU* prima string detekcije (da li je STOP detektovan, kretanje u odnosu na linije na putu, kao i očitano ograničenje brzine), međutim za vizualizaciju detektovanih objekata i linija šalje se set koordinata na kojima se isti nalaze na slici (frejmu). Ovakav tip poruke ne spada u standardnu *ROS* poruku, već je u pitanju dizajnirana poruka (*custom*) koju je moguće kreirati u *ROS*-u. Ova poruka poseduje 5 promenljivih, sve su tipa *integer* i redom predstavljaju: veličinu niza – broj set koordinata koji se šalje, dok ostale predstavljaju dve tačke, odnosno njihove *x* i *y* koordinate. Svaka od ove četiri koordinate je zapravo niz *integer*-a (*STL* vektora). Prednost korišćenja *STL* vektora u ovom slučaju jeste to što nije potrebno unapred znati dužinu niza, već se vrši dinamička alokacija memorije.

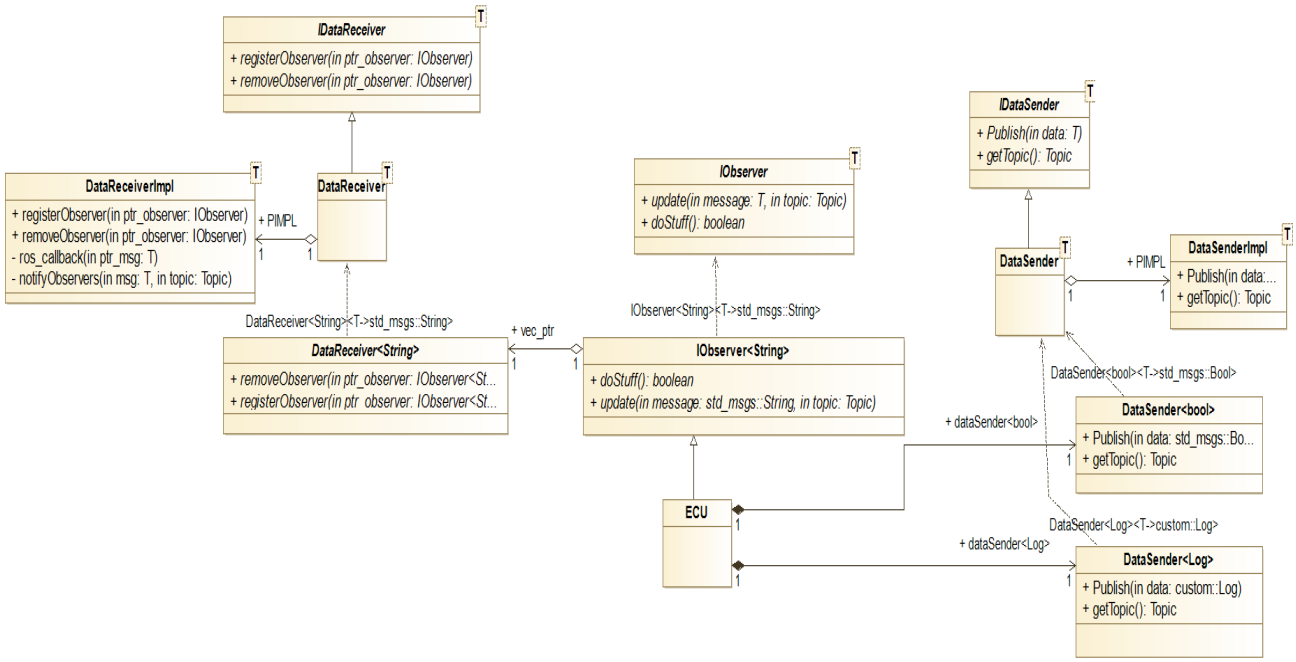
Detector klasa je zadužena za prijem i slanje podataka, dok je *Image Processor* klasa implementirana kao njena zavisnost (eng. *Dependency Injection*). *Detector* klasa dobija sirov frejm koji prosleđuje svojoj zavisnosti (*Image Processor*) koja na frejmu izvršava detekciju objekata od interesa, te vraća koordinate markiranog objekta, ali i zaključak detekcije u vidu stringa koji ide na *ECU* nod. *Detector* dobijene promenljive pakuje u *ROS* poruke i publikuje. *Image Processor* predstavlja interfejs klasu apstraktnog tipa koju nasleđuju *Stop, Lane* i *Limit Processor*. Kao i u prethodnom nodu, klase koje nasleđuju apstraktni interfejs moraju da poseduju metode koje ima apstraktna klasa, a u njihovoj deklaraciji ima pridodat naziv *virtual* i izjednačene su sa nulom. Ovaj princip (*Dependency Injection*) odlikuje dinamički polimorfizam, koji predstavlja suštinu *OOP*, gde se uz pomoć deklarisanog interfejsa kao člana neke klase može koristiti bilo koja klasa koja ga nasleđuje.



Slika 24. Algoritam rada Detektor nodova

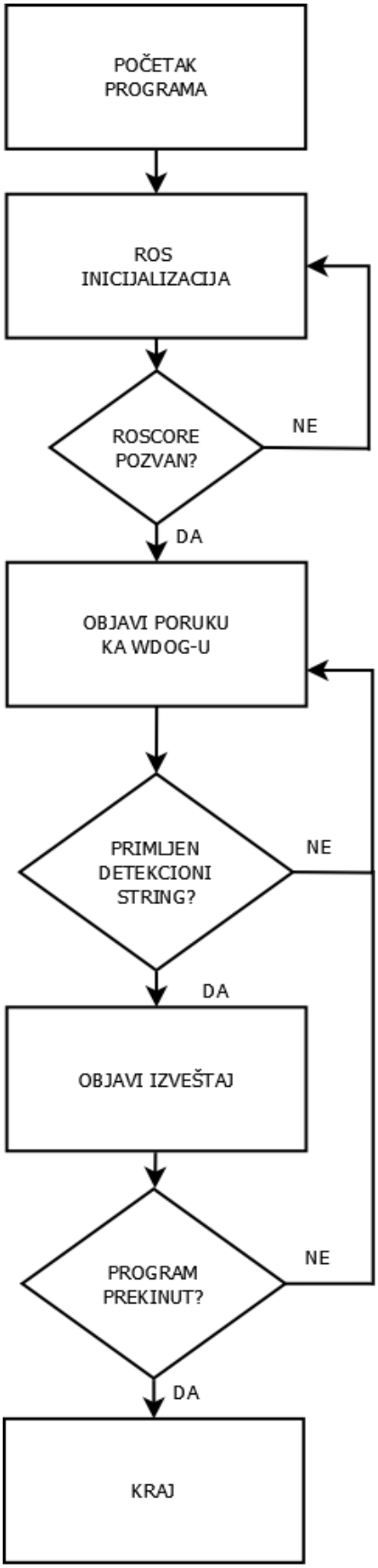
5.3 ECU Simulator

U pravom autonomnom vozilu *ECU* bi imao mnogo značajniju ulogu jer ne bi samo upravljao celokupnom senzorikom i skupljanjem podataka o okolini vozila, već bi bio odgovaran i za rad aktuatora koji bi upravljali smerom i brzinom vozila, što su oblasti same za sebe i nisu pokrivene u okviru ovog projekta. Stoga ovaj nod ima samo simboličnu ulogu, a to je da na osnovu primljenih rezultata detekcije šalje informacije ka *Visualizer*-u za ispisivanje.



Slika 25. Klasni dijagram ECU Simulatora

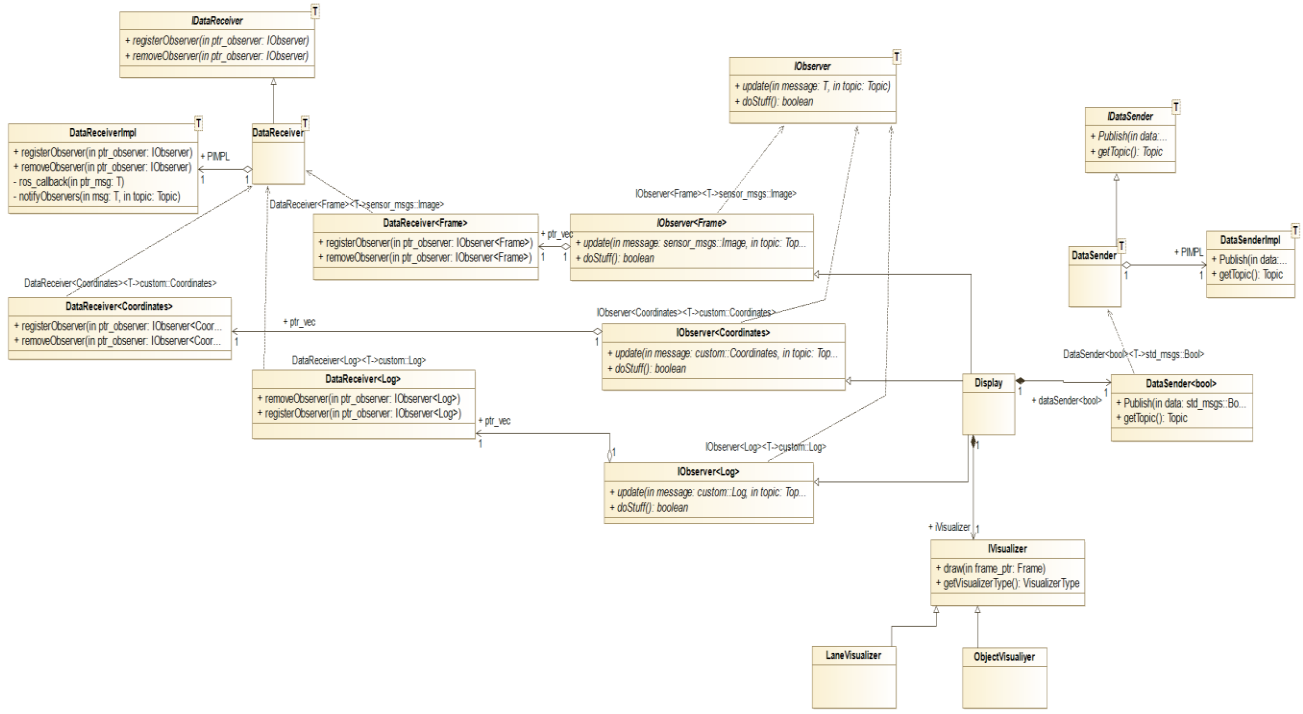
Klase koje poseduje ovaj nod su: *ECU* (koja nasleđuje *IObserver*), *Data Receiver* i *Data Sender* ka *Watchdog*-u i *Visualizer*-u. Ka *Visualizer*-u se i ovog puta prosleđuje *custom* poruka koja se sastoji iz *bool* promenljive namenjene STOP-u, *integer*-a za ograničenje i stringa za kretanje. Naziv generisane poruke je *ECU Log*.



Slika 26. Algoritam rada ECU Simulatora

5.4 Nod zadužen za vizualizaciju

Glavno zaduženje ovog noda jeste preuzimanje sirovog frejma, na kojem će posle obrade biti iscrtani pravougaonici na mestu detektovanih saobraćajnih znakova i linije na mestu horizontalne signalizacije na putu. Pravougaonici se crtaju tako što mu se prosleđuju *x* i *y* koordinate dva dijagonalno naspramna temena pravougaonika, dok se što se tiče linija na putu, crta linija između dve prosleđene tačke.



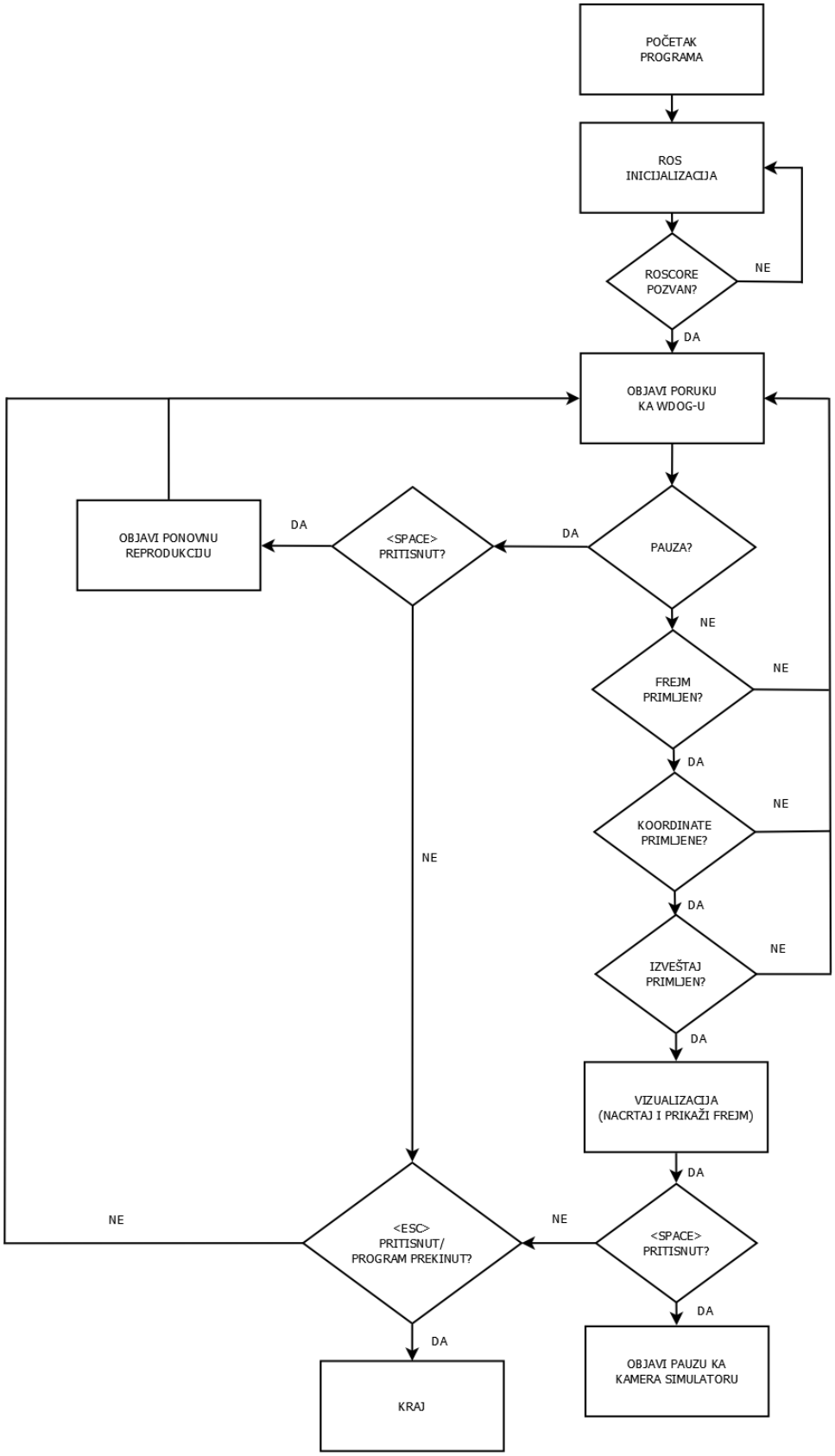
Slika 27. Klasni dijagram Vizualizacionog noda

Nod u okviru svoje arhitekture poseduje *Display* klasu, koja nasleđuje I*Observer* interfejs i prima koordinate, sirovi frejm, kao i izveštaj sa *ECU*-a ponovo pomoću *Data Reciever*-a. Ova klasa šalje *bool* poruku ka *Camera Simulator*-u što predstavlja pauziranje ili nastavak reprodukcije video snimka, a takođe izveštava i *Watchdog* da je prisutan.

Sama ekranizacija obrađenog snimka takođe se izvršava posredstvom *OpenCV* funkcije *cv::imshow* koja otvara novi prozor i prikazuje frejm, a uz pomoć *cv::waitKey* određuje se *fps* tok snimka. Poslednja funkcija pri otvorenom prozoru displeja pritiskom dugmeta tastature vraća *char* (*ASCII*) vrednost karaktera. U pravljenju jednostavne aplikacije dovoljno je odrediti *ASCII* vrednost koja naizmenično postavlja vrednost *fps* na 0 ili neku drugu vrednost kako bi se video pauzirao ili nastavila njegova reprodukcija. U ovom slučaju to nije moguće jer bi se pauzirao prikaz, ali se ne bi pauziralo slanje i obrada frejmova. Neophodno je staviti do znanja

Camera Simulator-u da prestane sa slanjem frejmova. To se postiže slanjem poruke od strane *Visualizer*-a pri pritisku <*space*> tastera.

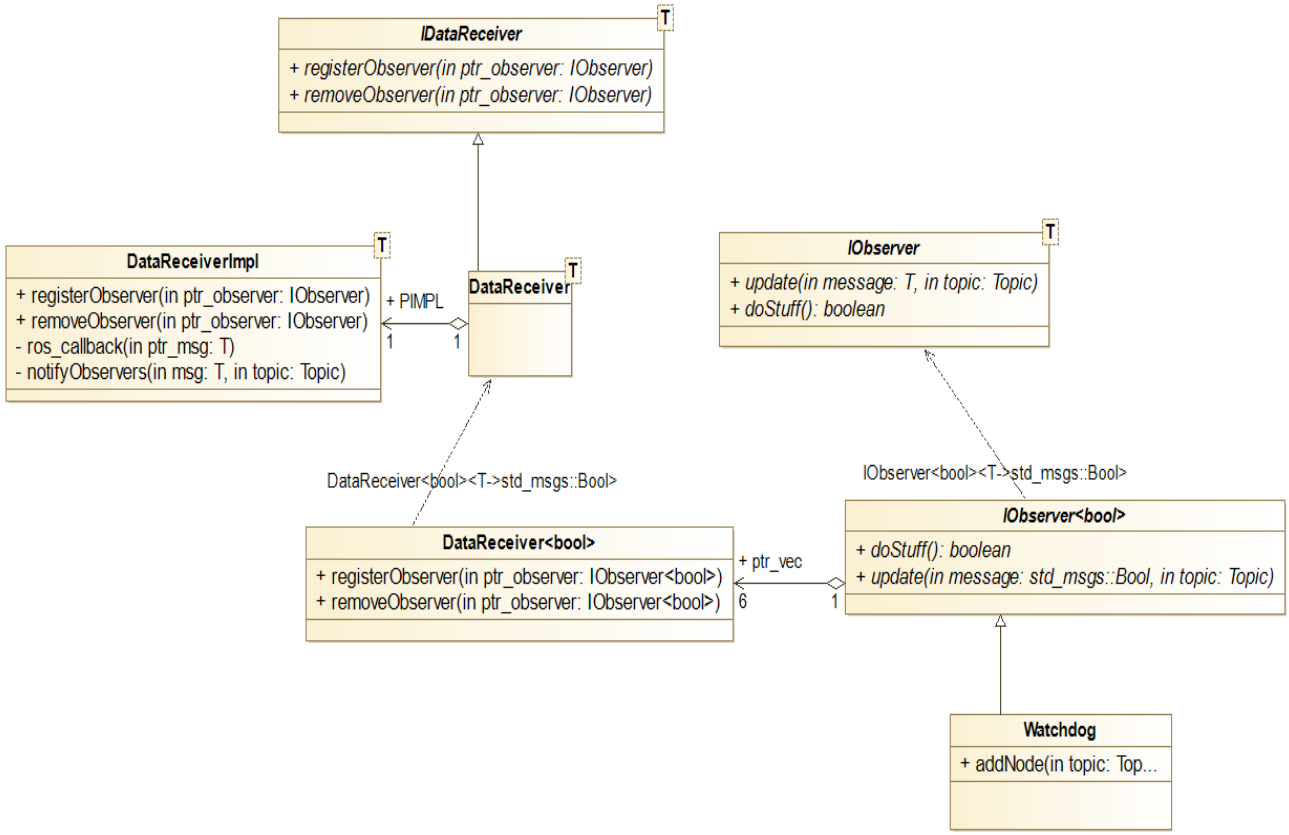
Primljeni frejm i koordinate se prosleđuju *IVisualizer* klasi koja je interfejs klasa zadužena za iscrtavanje detektovanog i asocijacijom je dodeljena *Display* klasi. Za iscrtavanje se takođe koriste funkcije iz *OpenCV* biblioteke. Nakon što je iscrtavanje izvršeno preostaje prikaz obrađenog frejma.



Slika 28. Algoritam rada Vizualizacionog noda

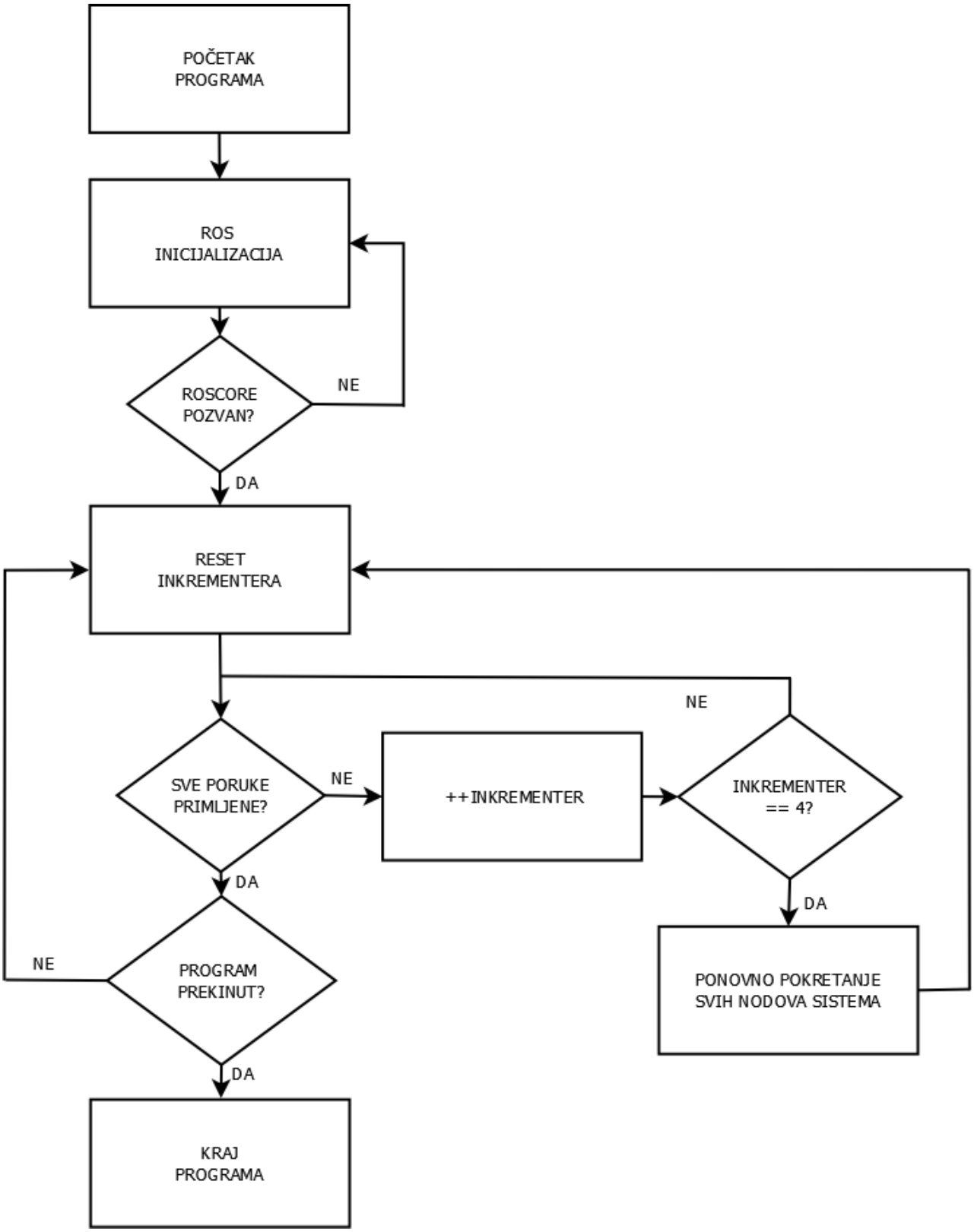
5.5 Nadzirač (*Watchdog*)

Ovaj nod je dobio ime po psu čuvaru koji nadgleda ceo system (eng. *Watchdog*) i reaguje ukoliko dođe do diskontinuacije u radu nekog od nodova. Naime, svi nodovi u okviru sistema šalju *bool* poruke ovom nodu čime mu javljaju da su i dalje prisutni i da obavljaju svoju funkciju. Poruke poslate ovom nodu pri svakoj iteraciji postavljaju *bool* članove njegove glavne klase (*Watchdog*) na *true*, a ukoliko su sve *bool* vrednosti logičke jedinice (eng. *true*) nod zaključuje da sistem u ovoj iteraciji ispravno i pravovremeno funkcioniše te vrednosti ponovo resetuje na logičku nulu (eng. *false*). Pošto svaki od nodova radi na različitoj učestanosti tolerancija *Watchdog*-a je maksimum od četiri iteracije u kojima sve primljene poruke nisu *true*. Ukoliko pak i nakon četvrte uzastopne iteracije ovo nije slučaj, ovaj nod resetuje ceo sistem, tačnije svaki nod ubija (isključuje), a zatim ga ponovo pokreće. Ubijanje i ponovno pokretanje nodova izvršava se pomoću *system* funkcije koja prosleđeni string direktno unosi u terminal.



Slika 29. Class Diagram Nadziračkog noda

Pored glavne klase (*Watchdog*), ovaj nod poseduje i *Data Reciever* koji logično služi da mu prosledi neophodne podatke.



Slika 30. Algoritam rada Nadziračkog noda

6. Testiranje sistema

U *Automotive* industriji jedna od krucijalnih grana jeste *QA* (eng. *Quality Assurance*, Osiguranje kvaliteta) kao krunski pokazatelj ispravnog rada sistema. Ovaj segment zapravo predstavlja proces testiranja kojim se osigurava ispravno funkcionisanje gotovo svake linije koda. Svaki programer otpočinje pisanje koda sa jasnom zamisli o tome šta tačno pisani kod treba da izvršava ali gotovo neminovno dolazi do odstupanja koje je potrebno u toku testiranja ukloniti. Važno je napomenuti da postoji razlika između simulacionih testova i fizičkih testova performansi vozila.

U C++ programskom jeziku testiranje se izvršava pomoću integracionih testova poznatijih kao *Unity Tests.* Ovi testovi mogu se podeliti na *GTest* i *GMock*. *GTest* je zadužen za testiranje ponašanja celokupnih i pojedinačnih funkcija sistema, dok je *GMock* zadužen za testiranje klase. U tzv. *mock*-ovanju test klasa nasleđuje klasu koja se testira i proverava njene metode. Samo testiranje svodi se na postavljanje parametara i očekivanja ishoda u skladu sa zadatim parametrima. Ukoliko su sva očekivanja ispunjena može se zaključiti da je test imao uspešan ishod.

Celokupni projekat prošao je kroz *unity* testiranje.

7. Zaključak

Konačno testiranje pokazalo je da su svi elementi sistema uspešno integrisani u jednu sinhronizovanu celinu koja traženi zadatak obavlja sa visokim procentom tačnosti. Obezbeđen je pouzdan rad svih nodova, pa čak i metoda koja bi ih reinicijalizovala u slučaju da ne javljaju da mogu da ispune svoj zadatak. Prosleđivanjem različitih snimaka napisanom programu ustanovljeno je da se svi zadati elementi sistema uspešno realizuju. Detektuju se STOP znaci, znaci ograničenja brzine, koja god da je brzina na njima napisana, a takođe je ostvareno praćenje linija koje omogućava automobile da uvek ostane u svojoj traci.

Jedna od prednosti predstavljene realizacije sistema jeste i korišćenje pametnih pokazivača (eng. *Smart Pointer*) koji se gotovo uvek sami dealociraju što sprečava eventualno nagomilavanje smeća na dinamičkoj memoriji (eng. *heap*). Pre svega, smanjena je verovatnoća da će *developer* napraviti takvu grešku.

Tačnost detekcije bi se mogla pojačati ukoliko bi u sistem bila integrisana neka od modernih platformi kao što je *Tensorflow*, *YOLO* ili *RCNN*. Međutim, svaka od ovih alatki iziskuje hardver izuzetno visokih performansi, a prvenstveno upotrebu *GPU* kartica. Neki od ovih biblioteka su isključive i po pitanju tipa kartice, te *Tensorflow* radi sa *nVidia* grafičkim karticama. što povećava troškove realizacije projekta.

Takođe, tačnost detekcije moguće je unaprediti preko postojećih klasifikatora upotrebom većeg trening seta slika. Ovo iziskuje znatno veći broj pozitiva, a samim tim i znatno većim brojem negative, čije bi treniranje moglo trajati i nekoliko nedelja. Važno je naglasiti da proces treniranja modela postoji i kod gorenavedenih alternativnih pristupa.

Još jedan način poboljšanja performansi rada sistema bio bi uključivanje više tredova (eng. *Multithreading*). Ovaj postupak nije korišćen unutar ovog projekta zbog *ROS* platforme. Naime, primećeno je da implementacijom *thread*-ova i *mutex*-a dolazi do problema pri radu *ROS*-a.

8. Reference

[1] https://www.researchgate.net/publication/305904140_Autonomous_Vehicles_Challenges_Opportunities_and_Future_Implications_for_Transportation_Policies - ”*Autonomous Vehicles: Challenges, Opportunities and Future Implications for Transportation Policies*”, *Saeed Asadi Bagloee*, pristupljeno: januar 2020.

[2] http://wiki.ros.org/ROS/Introduction - “*What is ROS? ROS Introduction*”, Zvanična *ROS* dokumentaciona stranica, pristupljeno: januar 2020.

[3] http://wiki.ros.org/ROS/CommandLineTools - “*ROS Command-line tools*”, Zvanična *ROS* dokumentaciona stranica, pristupljeno: januar 2020.

[4] http://wiki.ros.org/Nodes - “*ROS Nodes*”, Zvanična *ROS* dokumentaciona stranica, pristupljeno: januar 2020.

[5] https://en.wikipedia.org/wiki/OpenCV - “*OpenCV - from Wikipedia, the free encyclopedia*”, *Wikipedia* članak, pristupljeno: januar 2020.

[6] https://github.com/tesseract-ocr/tessdoc - “*Tesseract User Manual* “, Zvanični *github* repozitorijum, pristupljeno: januar 2020.

[7] https://www.geeksforgeeks.org/the-c-standard-template-library-stl/ - “*The C++ Standard Template Library (STL)”*, *Geeks-for-Geeks* članak, pristupljeno: januar 2020.