УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

ДИПЛОМСКИ РАД

Детекција знакова и праћење трака при кретању аутономног возила у саобраћају

Кандидат:

Јован Славујевић

Ментор:

проф. др Рајс Владимир

Нови Сад, 2020. год.

УНИВЕРЗИТЕТ У НОВОМ САДУ ●**ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:		
Идентификациони број, **ИБР**:		
Тип документација, **ТД**:		Монографска публикација
Тип записа, **ТЗ**:		Текстуални штампани документ
Врста рада, **ВР**:		Дипломски рад
Аутор, **АУ**:		Јован Славујевић
Ментор, **МН**:		проф. др Владимир Рајс
Наслов рада, **НР**:		Детекција знакова и праћење трака при кретању аутономног возила у саобраћају
Језик публикације, **ЈП**:		Српски
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Република Србија
Уже географско подручје, **УГП**:		Аутономна Покрајина Војводина
Година, **ГО**:		2020.
Издавач, **ИЗ**:		Ауторски репринт
Место и адреса, **МА**:		21000 Нови Сад, Факултет техничких наука, Трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/страна/цитата/табела/слика/графика/прилога)		10 / 42 / 10 / 1 / 55 / 8 / 0
Научна област, **НО**:		Аутоматика
Научна дисциплина, **НД**:		Рачунарске комуникације
Предметна одредница/Кључне речи, **ПО**:		Аутомотив
УДК		
Чува се, **ЧУ**:		Библиотека Факултета техничких наука, Трг Доситеја Обрадовића 6, Нови Сад
Важна напомена, **ВН**		
Извод, **ИЗ**:		У пројекту је симулиран рад аутономног возила коришћењем *ROS* платформе, односно његових нодова. Нодови врше пријем снимка са предње камере аутомобила, слање снимка „фрејм по фрејм“ на обраду, слање података о детекцији ка *ECU*, затим визуализацију обрађеног снимка, као и надзирање рада целокупног система. Уколико неки од нодова престане да ради, специјални *Watchdog* нод треба поново да га покрене.
Датум прихватања теме, **ДП**:		Март 2020.
Датум одбране, **ДО**:		10.03.2020
Чланови комисије, **КО**:	Председник:	доц. др Бабковић Калман
	Члан:	доц. др Бркић Миодраг — Потпис ментора
	Ментор:	доц. др Рајс Владимир

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, after which the data is sent to the *ECU*. 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:	PhD Babković Kalman	
	Member:	PhD Brkić Miodrag	Menthor's sign
	Member, Mentor	PhD Rajs Vladimir	

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:			
Студент	Јован Славујевић	Број индекса:	ЕЕ 239/2015
Предмет:	Електроника		
Ментор:	доц. др Владимир Рајс		

ТЕМА ДИПОМСКОГ РАДА:

Детекција знакова и праћење трака при кретању аутономног возила у саобраћају

ТЕКСТ ЗАДАТКА:

1. Интегрисати *ROS* окружење у своју машину, направити јединствени *ROS* пакет и своје радно окружење.
2. Направити нод задужен за пријем снимка и слање фрејмова даље на обраду.
3. Реализовати детекцију *STOP* знака.
4. Реализовати детекцију ограничења.
5. Реализовати детекцију и праћење линија саобраћајне траке.
6. Направити нодове задужене за сву задату детекцију, који шаљу фрејм даље на визуализацију, као и обрађене податке на *ECU*.
7. Направити *Watchdog* нод задужен за контролу рада целог система.
8. Тестирати рад система преко *Unit* тестова.

Шеф катедре:	Ментор рада:
др. Дамњановић Мирјана	др. Рајс Владимир

Примерак за: o - Студента; o - Ментора; o - Катедру; o - Студентску службу факултета;

Садржај

Листа скраћеница

Списак скраћеница на енглеском језику

ROS	Robot Operating System – Роботски оперативни систем
ECU	Electronic Control Unit – Електронска контролна јединица
CPU	Central Processor Unit – Централна процесорска јединица
OCR	Optical Character Reading – Оптичко читање карактера
STL	Standard Template Libraries – Стандардна библиотека образаца
ROI	Region of Interest – Регион од интереса
CNN	Convolutional Neural Networks – Конволуционе неуронске мреже
GUI	Graphical User Interface – Графички кориснички интерфејс
OOP	Object Oriented Principle – Објектно оријентисани принцип
PIMPL	Pointer to Implementation – Од показивача до имплементације
FPS	Frames per Second – Фрејмови у секунди
ASCII	American Standard Code for Information Interchange – Амерички стандардни код за размену информација
QA	Quality Assurance – Осигурање квалитета
GPU	Graphics Processing Unit – Графичка процесорска јединица
RCNN	Region Convolution Neural Network – Регионална конволуциона неуронска мрежа

1. УВОД

Као што је познато у светским оквирима, тенденције у даљем технолошком развоју и развитку у свим гранама индустрије јесте развој роботике и саме вештачке интелигенције. С обзиром да се већ годинама уназад напредовало у развитку роботике дошло се и до комерцијализације и употребе појединих вештачких интелигентних машина у људском окружењу. Једна од кључних идеја и примена робота у људским животима јесте употреба роботизованих возила тј. аутономних возила у саобраћају. Евидентан раст кретања људи превозним средствима, последично повећани ризик од саобраћајних незгода, повећан број потребних ресурса за све већи број возила, повећане саобраћајне гужве, те знатно повећано време проведено у саобраћају итд. су нека од кључних питања саобраћаја данашњице. [1]

Слика 1. Аутономно возило

Задатак дипломског рада јесте реализација софтвера који ће омогућити аутономном возилу да самостално препознаје саобраћајне знаке и да прати саобраћајне траке како бисе могло самостално кретати кроз саобраћај. Ово је реализовано помоћу *ROS*[2] платформе, обрада снимка је рађена помоћу *OpenCV* библиотеке, софтвер је писан користећи програмски језик C++, по C++ 17 стандардима. Детекција знакова рађена је помоћу конволуционих неуронских мрежа, што спада под Дубоко машинско учење (енг. *Deep machine learning*), а поступак је заправо тренирање модела којима јак процесор (*CPU*) омогућава процесуирање великог броја позитивних и негативних слика које омогућавају моделу да разазна објекат.

2. *ROS*

ROS[2] (енг. *Robot Operating System*, Роботски оперативни систем) је програмско окружење које пружа алатке и програмске библиотеке које омогућавају кориснику да направи апликацију за робота. До скоро се стриктно могао користити у *Linux,* тачније *Ubuntu* оперативном систему, док су новије верзије (*ROS Melodic*), коришћене у овом пројекту, компатибилне и са *Windows* оперативним системом.

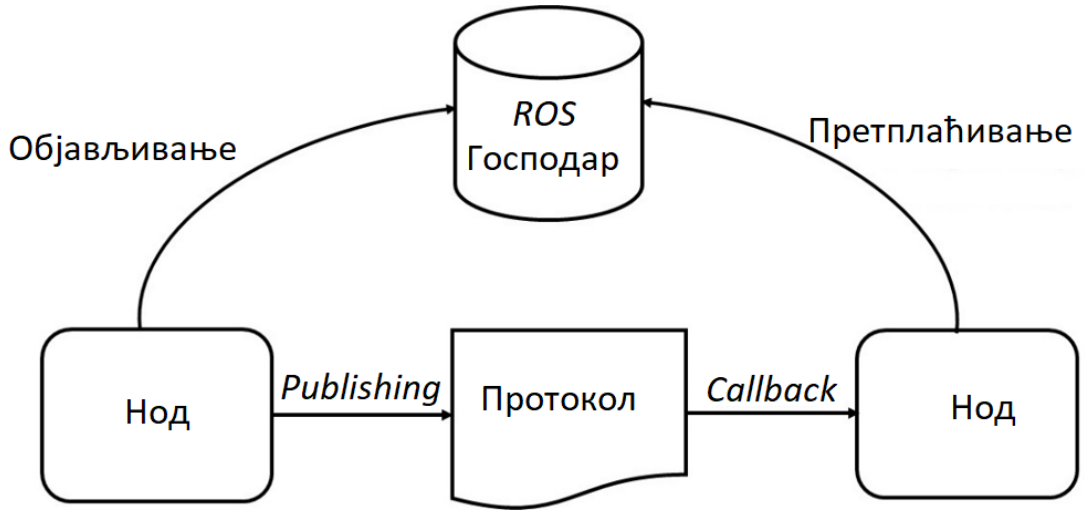
Слика 2. Лого ROS Melodic-а

ROS је оперативни систем отвореног кода (енг. *Open source*) у ширем смислу те речи, што значи да поседује отворене библиотеке којима се лако приступа, али пре свега, бесплатне су. Програм који се извршава може се писати у *C/C++* и/или *Python* програмском језику. Односно, различити нодови могу бити написани на различитим програмским језицима и да заједно чине један систем. *ROS* окружење омогућава лако и једноставно *build*-овање извршних фајлова командом *catkin make*, а нод уникатног назива из специфичног програмског пакета се може покренути командом *rosrun*, уз помоћ *GNOME* терминала. Поред њих, постоји још команди за рад у овом окружењу, као нпр. за брзо приступање директоријумима, за излиставање покренутих нодова, за графичку визуализацију итд. [3]

2.1 Нод

Нодови су делови софтвера који могу да реализују различите задатке у зависности од тога на који су начин програмирани. Нодови могу да процесуирају податке са бројних сензора, као и да задају даље извршавање команди неким актуаторима, али и да међусобно комуницирају. Та комуникација међу нодовима може да се извршава преко два различита протокола, а то су: Гласник-претплатник (енг. *Publisher-Subscriber*) и Клијент-Сервер.

Први протокол подразумева такав вид комуникације где један нод може нешто да публикује (енг. *publish*) на један протокол (енг. *topic*), а други нод се може претплатити (енг. *subscribe*) на тај протокол и преузети објављене податке. Нод ради у служби самоактуализације, где ће ослушкивати податке на које се претплатио и реаговати у складу са евентуалним изменама, односно извршава окидање (енг. *Callback*) први свакој новој публикованој поруци. Порука може да садржи различите типове података појединачно, као што је целобројна вредност (*integer*), децимални број (*float*), те низ карактера (*string*) ,па чак и слика или звук. Такође, порука се може генерисати и по наруџбини (енг. *Custom message*), те може да комбинује све ове типове. Треба нагласити да овај протокол функционише по класичном „господар-слуга“ (енг. *Master-Slave*) принципу, где само један нод може да објављује поруку у једном тренутку, а остали нодови моду да је примају, при чему број претплатних нодова није ограничен. Други протокол (Клијент-Сервер) функционише тако да нод који игра улогу клијента шаље податке ка серверу, а на основу њих сервер нод враћа повратну информацију ка њему.[4]

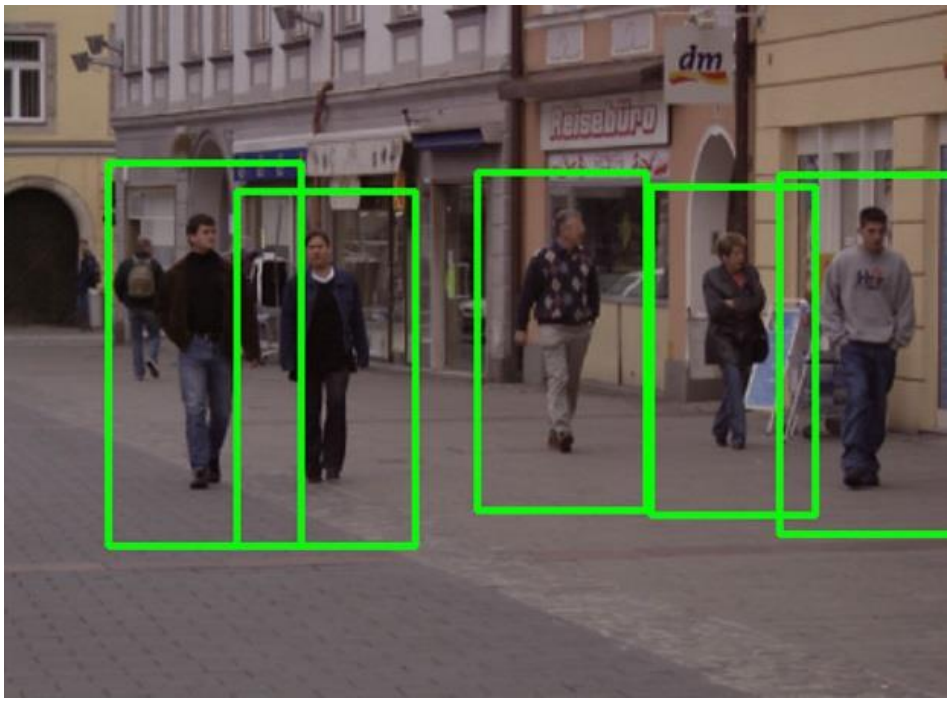


Слика 3. Блок дијаграм Гласник-Претплатник протокола

3. АЛАТИ ЗА ДЕТЕКЦИЈУ

3.1 *OpenCV*

OpenCV[5] представља колекцију библиотека писаних у *C/C++*, *Python* и *Java* програмском језику које се првенствено користе за компјутерску визију у реалном времену. Првобитно ју је развила компанија *Intel*, а данас је доступна као софтвер отвореног кода. Неке од примена ових библиотека укључују: сегментацију, праћење покрета, препознавање лица, препознавање гестова, померање камера унутар *3D* простора (енг. *Egomotion*), али и препознавање објеката помоћу тренираних неуронских мрежа.



Слика 4. Пример детекције објеката помоћу OpenCV платформе

3.2 Платформа за оптичко препознавање карактера - *Tesseract*

Tesseract[6] је платформа за оптичко препознавање карактера (енг. *Optical Character Recognition, OCR*) која се може користити за разне оперативне системе. У питању је бесплатан софтвер чији је развој спонзорисан од стране *Google*-а још од 2006. Сматра се једном од најтачнијих *OCR* платформи које су отвореног кода. Већина кода написана је у програмском језику *C*, а остатак у *C++*, с тим што је после цео код преписан да би се макар могао компајлирати *C++* компајлером. *OCR* представља електронску или механичку конверзију слика куцаног, ручно писаног или штампаног текста у машински енкодован текст, било са скенираног документа, фотографије документа или фотографије природног окружења са знакова, билборда и слично. *OCR* је значајно поље истраживања за препознавање образаца, вештачку интелигенцију и компјутерску визију.

У овом пројекту, *OCR* се конкетно користи за очитавање ограничења брзине или речи *STOP* са *STOP* знака.

Слика 5. Лого Tesseract OCR

4. ДЕТЕКЦИЈА ЗАДАТИХ ОБЈЕКАТА

У склопу израде овог дипломског рада траажено је да израђени програм детектује поједине објекте у саобраћају, конкретно линије саобраћајних трака унутар којих се возило налази, *STOP* знакове и знакове ограничења брзине.

4.1 Детекција *STOP* знака

STOP знак је један од специфичних саобраћајних знакова. У питању је, дакле, осмоугаона табла на којој је белим словима исписано *STOP* са црвеном подлогом. Постоје различите могућности и приступи како би се могла одрадити детекција *STOP* знака. Пре свега, најједноставнији приступ био би препознати знак на основу његове боје и облика. Црвена боја је доминантна и лако опазива, те би први корак у обради слике био сегментације црвене прављењем бинарне маске која за праг (енг. *threshold*) узима нијансе црвене, док остали пиксели имају вредност логичке нуле.

Слика 6. Сегментација црвене боје

Добијена маска се филтрира, како би се уклонили шумови, замагљивањем (енг. *blurring*) преко Гаусове расподеле. Након тога *OpenCV* пружа могућност да се сакупе пиксели у оквиру структуре *cv::Point* као низ тачака. У *C++* програмском језику најпогоднија за овај поступак је употреба *STL*[7] вектора, који се понашају као динамички алоцирани низови, те није потребно претходно дефинисати опсег низа. Одабир контура

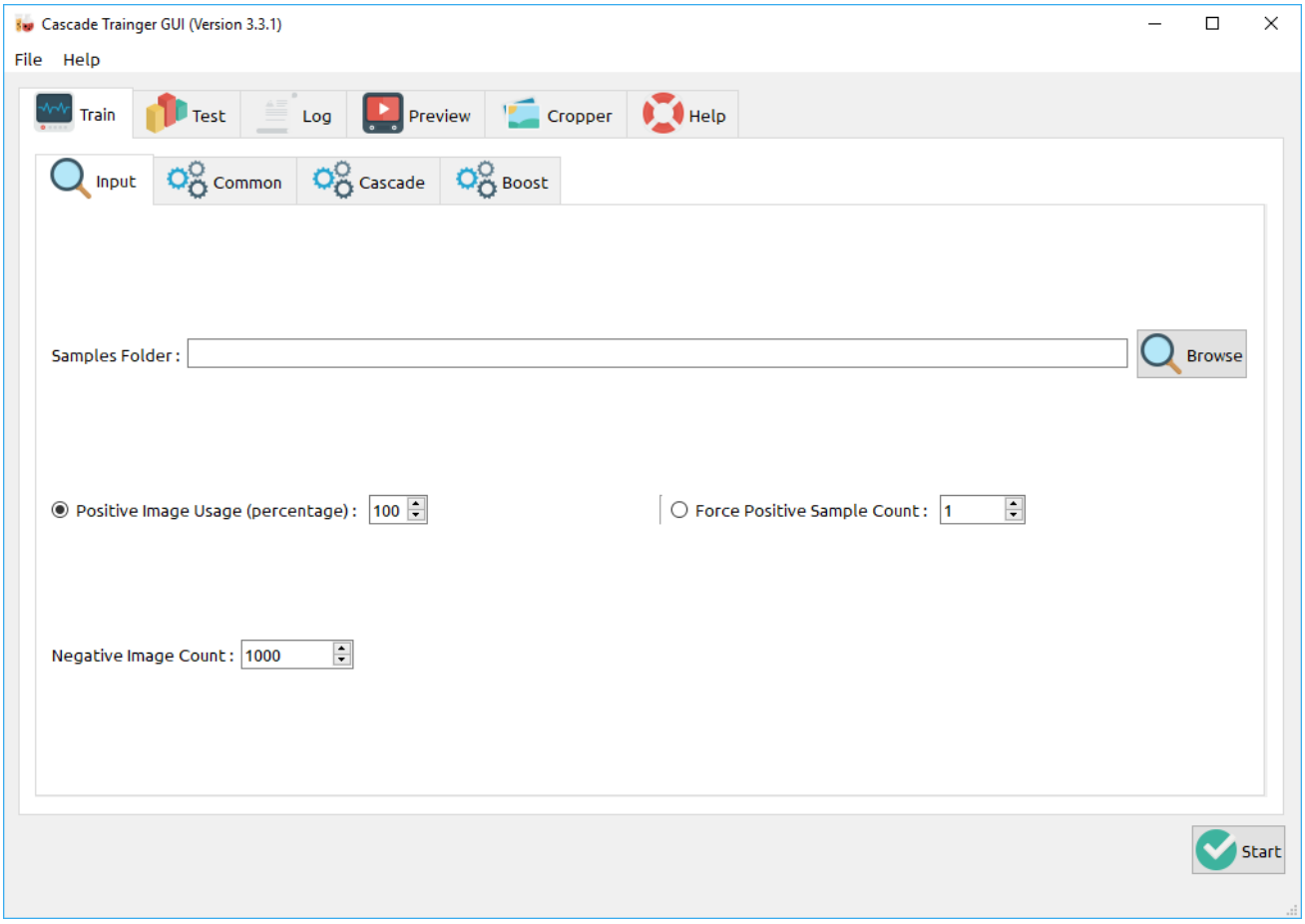
од интереса се може вршити на основу површине, као и облика контуре. Контура мора да се поседује осам страница и да буде правилног облика, односно да су све странице једнаке и сви углови једнаки. Након таквог одабира, маркирањем преосталих контура добија се визуелна детекција знака.

Слика 7. Blur-овање маске и маркирање одабране контуре

Поставља се питање да ли је ово добар начин за детекцију. Овај поступак је добар за детекцију статичких објеката, али показује извесне недостатке када се ради о детекцији при покрету и у реалном времену из више разлога. Осим што се може десити да камера због своје позиције на возилу и перспективе у односу на знак никада не види правилни осмоугао, такође, теоретски би се могло догодити да поред знака у саобраћају сусретне и још неки правилни осмоугаони објекат црвене боје (нпр. реклама на билборду), или да због услова осветљења камера забележи фрејм где пиксели на знаку неће проћи бинарну маску. Ови проблеми захтевају другачији приступ.

Последњих година за решавање проблема детекције користе се неуронске мреже, тачније конволуционе неуронске мреже (енг. *Convolutional Neural Network, CNN*). *OpenCV* омогућава тренирање неуронских мрежа на једноставан начин. Потребно је обезбедити слике објекта од интереса, у овом случају *STOP* знака, који се називају позитиви, као и негатива, а то је велики број слика у којима се не налази објекат од интереса. Потребно

је направити табеле оба поддиректоријума где се налазе имена слика, а за позитиве мора бити назначен регион од интереса (енг. *Region of Interest, ROI*), односно почетне координате правоугаоника у којима се налази објекат, као и ширина и дужина тог правоугаоника. Постоји могућност да се унутар истог позитива нађе више објеката од интереса, у ком случају се процедура понавља за сваки од објеката као нови ред у колони табеле. Поступак тренирања је значајно олакшан захваљујући *Windows* апликацији *Cascade Trainer GUI*. У питању је програм који поседује графички интерфејс за исецање објеката унутар слика или снимака, који при покретању тренинга сам генерише све потребне фајлове, након чега започиње тренинг. Тренинг може да буде готов за неколико сати, али може да траје и неколико дана, у зависности од броја слика које се стављају на располагање мрежи, али и од перформанси машине на којој се тренинг врши.

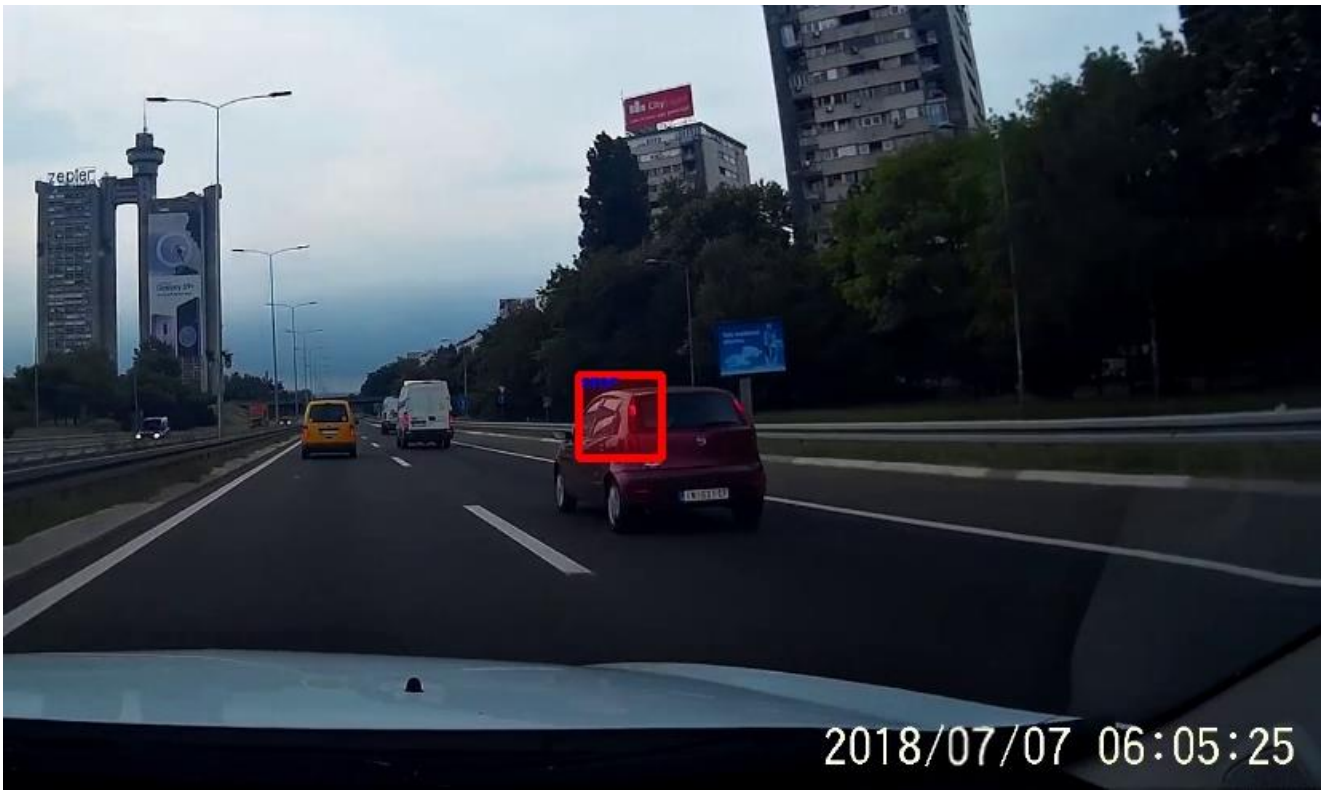


Слика 8. Cascade Trainer GUI програм

За детекцију *STOP* знака у овом пројекту кориштен је сет слика немачких саобраћајних знакова са интернет странице Института за Неуроинформатику Савезне Републике Немачке. У питању је бесплатан сет слика специјализовано намењених за ту

сврху. Коришћено је око 600 позитивних слика *STOP* знака и око 8000 негатива, а тренинг је трајао преко 10 часова. *OpenCV* за обраду користи *CPU* рачунара, а након завршетка тренинга програм коначно креира фајл са *.xml* екстензијом. Од свих генерисаних фајлова овај фајл је једини значајан фајл. Фајл се укључује у пројекат помоћу *OpenCV* структуре *cv::CascadeClassifier*, тако што се при инстанцирању објекта њему као аргумент проследи апсолутна путања локације на којој се фајл налази. Уколико је фајл исправно генерисан, након његовог укључивања у пројекат, инстанца ове структуре може даље несметано да обавља детекцију објеката на снимку. Координате сваког објекта којег неуронска мрежа сматра објектом од интереса бивају похрањене у вектор *ROI*, што се после може искористити за визуелно маркирање објекта на фрејму.

Уколико се неуронска мрежа користи као таква без претходне обраде у виду смерница које би јој наговестиле на шта да се фокусира, долази до прегршт проблема. Неуронске мреже су још увек у повоју и као такве нису идеалне, те долази до грешака у детекцији, које и даље нису у потпуности објашњене. Такође, долази до драстичног успоравања у раду система када неуронска мрежа обрађује целу слику односно фрејм.



Слика 9. Пример погрешне детелцоке STOP знака

Због ових проблема неопходно је да се пре рада *CNN*-а одради предпроцесуирање фрејма. Много бољи резултати се постижу комбиновањем двеју наведених метода. Опсег претраге сужава се прво уз помоћ бинарне маске, а затим следи одабирање контура где су критеријуми за одабирање обим и површина објекта, како би се избегла евентуална детекција објеката који по овим параметрима не одговарају саобраћајним знацима, већ могу бити потпуно насумични објекти на слици. Тек онда се вектори ових контура прослеђују неуронској мрежи на детекцију. Овим се значајно повећава тачност детектовања и повећавају се перформансе програма, те не долази до успоравања због преоптерећења процесора. Међутим, и даље може доћи до грешке у детекцији при чему се детектују објекти који су прошли одабир контуре, али нису *STOP* знак нити по габаритима могу одговарати *STOP* знаку (нпр. велики црвени камион, штоп светло возила и сл.). Ово је последица пре свега непостојања универзалне норме за параметар одабирања, јер се објекти током кретања возила снимају из различитих углова. Због овога, пожељно је направити још једно предпроцесуирање у виду бинарне маске, која додатно сужава опсег детекције исецајући делове фрејма који нису од интереса, јер се у том региону никада неће наћи *STOP* знак. Такав регион се пре свега односи на небо и на објекте који су удаљени од саобраћајнице. Више речи о овоме биће у поглављу где је објашњена детекција ограничења брзине.

После додатног предпроцесуирања и детекције класификатора, као гарант исправне детекције *STOP* знака, уводи се *OCR.* Увођење *OCR*-а изискује додатно процесуирање. Задатак *OCR*-а јесте да на свим објектима од интереса који су прошли претходни поступак детектује бар три од четири слова речи *STOP*. Оптичко препознавање карактера има изузетно високу успешност код читања карактера скенираног или усликаног текстуалног документа, односно светлог платна са црним словима, али се проблем усложњава када треба детектовати карактере на фрејму видео снимка због непостојања одговарајућег контраста између позадине и написаног текста, али и појаве контура које својим обликом могу заварати *OCR* да помисли да се ради о словима иако то није случај. Један пример за ово јесте осмоугаона контура *STOP* знака. Дакле, неопходно је детектовати место на којем се налази текст пре него што се уради оптичко препознавање карактера, или слику обрадити тако да се на препознавање пошаљу само одговарајући делови фрејма.

Слика 10. Знак STOP пре и након процесуирања

Комбиновањем наведених приступа остварује се задовољавајућа тачност при детектовању објеката од интереса, у овом случају *STOP* знака.

Слика 11. Коначна детекција STOP знакова

4.2 Детекција и препознавање ограничења

Поред *STOP* знакова, један од значајних елемената у реализацији овог задатка била је детекција знакова ограничења брзине. У почетку је реализована само детекција знакова са ограничењем од 80km/h, да би затим доградњивањем *OCR*-а ово било проширено на све знаке ограничења брзине који су законом прописани у Републици Србији.

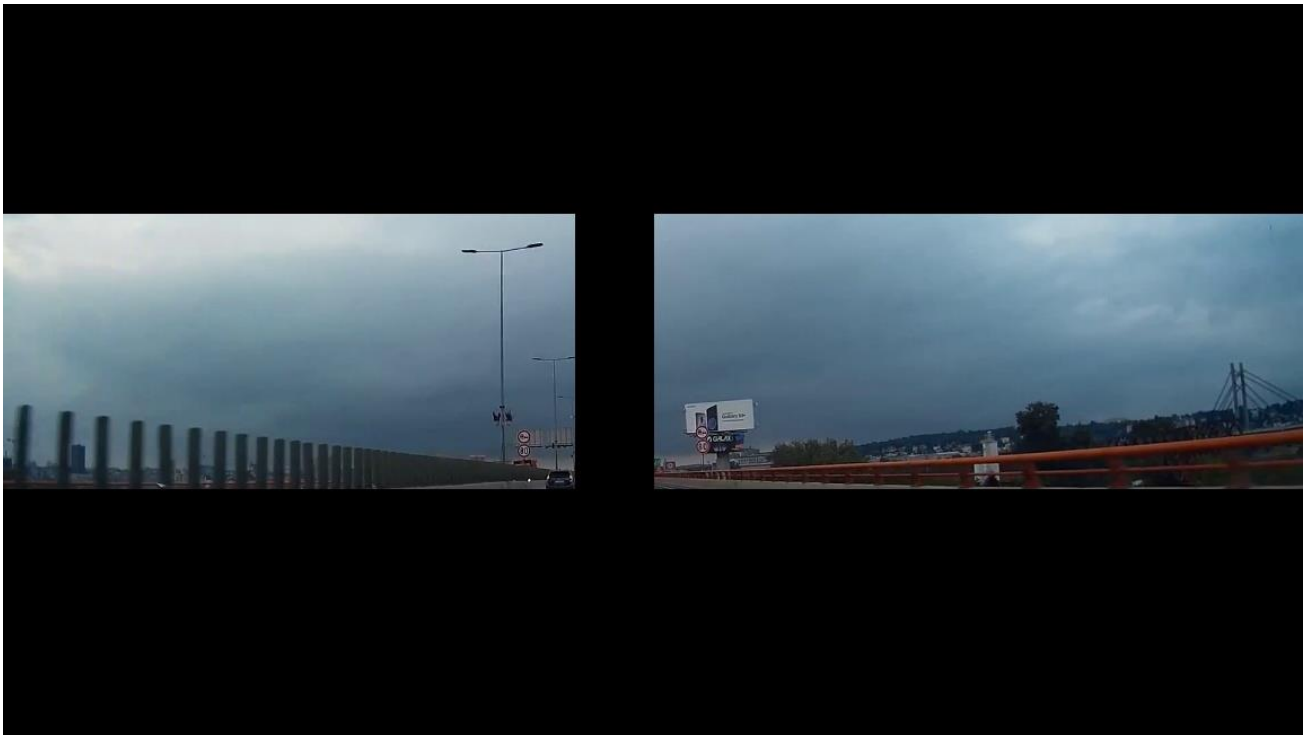
Првобитни покушај да се детектује знак ограничења брзине састојао се из примене бинарних маски, сличних онима које су коришћене за детекцију *STOP* знака, које издвајају црвену боју, карактеристичну за обод знакова ограничења брзине. За разлику од *STOP* знака, у којем се користио *cv::findContours* који ће покупити све контуре, у оквиру ове детекције тражене су искључиво кружне контуре које се виде у бинарној маски, што се извршава захваљујући *cv::HoughCircles*. Кружне контуре складиште се у векторима тачака (*cv::Point*). Ова процедура детектоваће било који кружни облик који преостане након примене бинарне маске која издваја црвене контуре. Резултат овакве процедуре биће следећи:

Слика 12. Погрешна детекција круга који није на знаку за ограничење брзине

Овакав поступак не даје очекиване резултате због великог броја детекција знакова који својим обликом и контурама подсећају на знакове ограничења брзине, на пример знак забране претицања који такође има облик круга са црвеним ободом. Овај проблем је очевидан, за разлику од детекције *STOP* знака, где због своје специфичности оваква аномалија не би дошла до изражаја, због тога што *STOP* знак има јединствене контуре које нису својствене осталим саобраћајним знацимиа. Како знаци ограничења брзине припадају скупу знакова изричитих наредби, они свој облик и боју деле са многим другим саобраћајним знацима, те овај поступак не би био довољно добар у пракси. Поред овог очигледног недостатка, као и код *STOP* знака, може доћи до тога да у реалним условима саобраћаја сензор који врши детекцију окружења (камера) неће забележити савршен круг јер га снима из различитих углова који зависе од њене позиције у аутомобилу који се креће кроз саобраћај.

Имајући све ово у виду, било је потребно предложити ново решење које би елиминисало уочене мањкавости. Поступак детекције ограничења принципијелно се своди на исти поступак као при детекцији *STOP* знака, где кључну улогу играју класификатори (енг. *Classifier*).

Поступак који није детаљније описан при детекцији *STOP* знака је коришћен у оба случаја, а поред сегментације црвене боје, подразумева и креирање маске која игнорише одређене регионе фрејма који нису од интереса. Наиме, саобраћајни знаци које је потребно детектовати се по природи налазе само у одређеном делу фрејмова које камера бележи. Да би се олакшао процес детекције, неопходно је сузити простор на којем се трага за објектима од интереса на онај простор где се ти објекти заправо и могу налазити, уместо на цео фрејм где се могу наћи разноразни други објекти који по свом изгледу или контурама подсећају на саобраћајне знакове, али то нужно не мора бити случај.

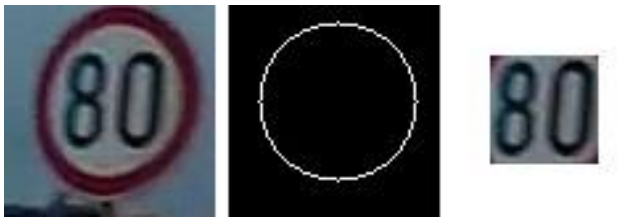


Слика 13. Приказ региона од интереса

Као и у случају *STOP* знака, сегментација црвене боје представља следећи поступак, а затим следује одабирање контура унутар бинарне слике које се даље прослеђују ка класификатору (*cv::Classifier*). Тренинг сет класификатора морао је садржати велику количину знакова ограничења различитих вредности, док су негативи поред све околине коју имају морали да садрже и известан број негатива који су сличног облика и боје. Ово значи да је у негативу садржано мноштво знакова ограничења који су морали бити црвене боје обода и беле боје унутрашњости, као на пример знак за ограничење међуосовинског оптерећења, за ограничење висине, знак за забрану претицања, знак за забрану скретања и сл. Овај тренинг, који је такође вршен захваљујући истом програмском интерфејсу, се извршавао нешто дуже. Готов класификатор који је прошао кроз свих 20 корака (енг. *stage*) добијен је за преко два дана тренирања неуронске мреже. Отежавајућа околност у овом случају била је да би тренинг потрајао и по 10 сати, затим детектовао неку грешку, па је проблем било неопходно уклонити, а цео поступак радити испочетка.

Слика 14. Погрешна детекција знака ограничења брзине преко класификатора

Као и у случају код *STOP* знака и даље долази до погрешне детекције, што је последица несавршености неуронских мрежа. У поступак решавања морао се поново придодати *OCR* као доказ да унутар детектоване површине постоје бројеви који репрезентују ограничење брзине. Поступак поново није могао бити директан, што значи да након детекције помоћу класификатора исти тај исечак није могао бити послат на *OCR* већ се морало радити предпроцесуирање како би исечак имао чисто црно бели текст.



Слика 15. Процес припреме исечка за OCR

Након очитавања текста, исти се складишти у *string* који може бити из *C++* стандардне библиотеке (*std::string*) или обичан низ карактера (*char[]* или *const char**). Иако није детаљније описан у претходном поглављу, овај поступак се користи и током детекције *STOP* знака, те ће му овде бити посвећена већа пажња. Наиме, очитавање *string*-а са слике понекад може резултовати у низу карактера који поред бројева који

сигнализирају дозвољену брзину садржи и неке нежељене карактере, па се у том случају ради компарација *string*-а, односно тражи се користан део *string*-а у очитаном *string*-у, па ако дође до подударности, сматра се да је детектовано и успешно очитано ограничење брзине. Овај податак прослеђује се даље ка потенцијалном *ECU* аутономног возила (у овом случају симулаторском ноду који игра ту улогу).

Слика 16. Изглед коначне детекције знака ограничења брзине

4.3 Детекција линија саобраћајне траке

За разлику од до сада наведених поступака чија је сврха била детекција саобраћајних знакова, у овом случају било је потребно детектовати беле линије исцртане на путу како би возило знало у којој се саобраћајној траци налази. Овакав поступак тражи потпуно другачији приступ. Првенствена разлика се налази у томе да се детекција линија не може вршити уз помоћ *OCR*-а и класификатора. Потребно је константно пратити две линије које ограничавају траку којом се возило креће. Конзистентност праћења линија је веома битна у току вожње. Праћење линије возилу говори колико треба евентуално да скрене како би услед закривљења пута остало у својој саобраћајној траци.

Слика 17. Линије на путу

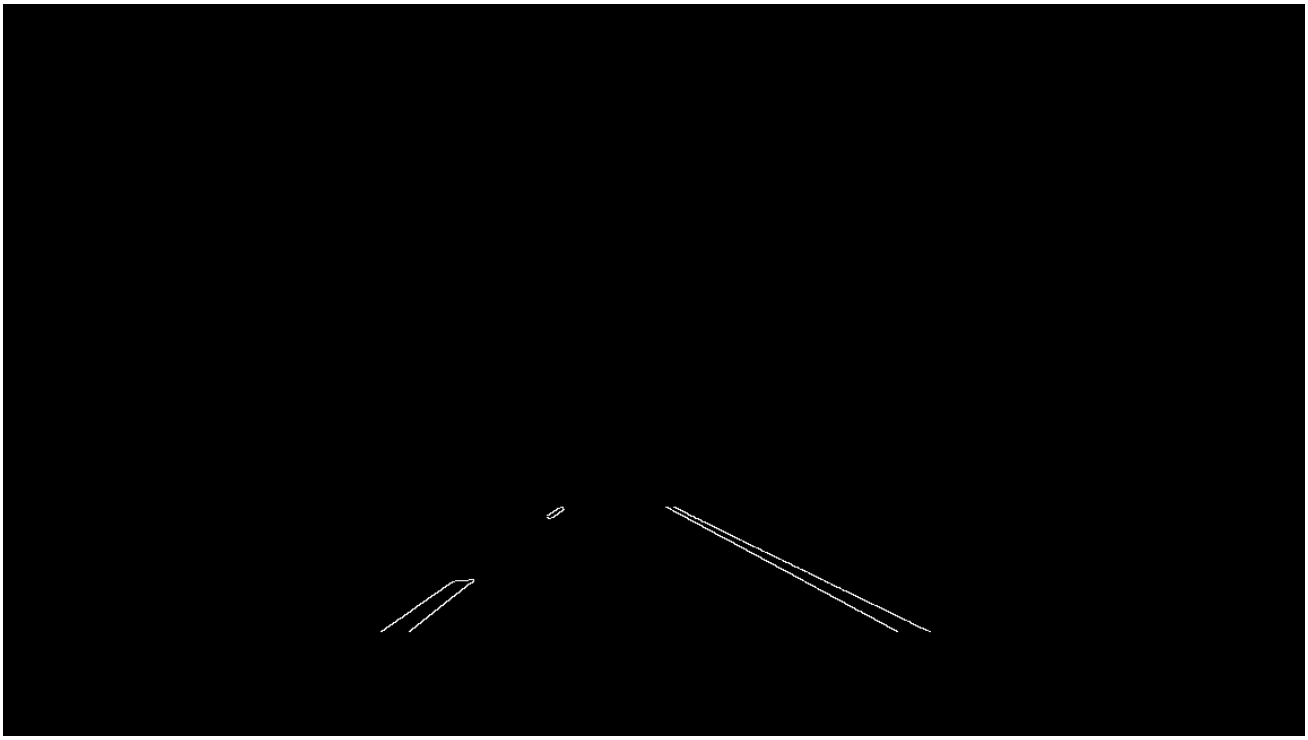
Поставља се питање како овом проблему приступити. Анализа почиње констатацијом да линије могу бити беле боје или жуте, уколико се ради о траци за јавни градски саобраћајни превоз или о привременим тракама услед радова на путу. Линије се могу појавити на тачно одређеном месту на слици (фрејму) које пре свега одлучује позиција камере у односу на возило. Линије могу бити испрекидане, пуне, па чак их не мора ни бити, што возило треба да детектује.

Први поступак могао би бити сегментација боје, у овом случају боје од интереса су бела и жута. Како је тешко направити бинарну маску која би радила савршену сегментацију само те две боје, односно тих линија, од тог поступка се одустало.

Први корак заправо је био ограничавање фрејма на регион од интереса. Овај регион је изузетно специфичан, варира од случаја до случаја, те маска умногоме зависи од позиције камере унутар возила. Да би тако нешто постало генеричко, а не унапред дефинисано, неопходно је поред саме детекције линија направити и својеврсну калибрацију камере како би се она сама оријентисала и фокусирала на прави регион од значаја. Овај поступак није имплементиран у систем, првенствено јер није био задатак пројекта. Дакле, референтна тачка, координате региона од интереса, а тиме и облик контуре и њена површина на фрејму, морају бити унапред дефинисани. Регион од значаја представљао је само део пута на којем се возило налази.

Слика 18. Регион од интереса детекције линија

Након тога, слику је требало процесуирати пре детектовања линије. Поступак трансформације који је примењен је такође из *OpenCV*-а. У питању је *cv::CannyEdge*. Ова функција замагљену (енг. *blur*) слику претвања у слику у којој се виде само ивице објеката.



Слика 19. Canny Edge трансформација слике

Следећи корак је сакупљање детектованих линија у вектор вектора тачака (низ линија). Сакупљање линија слично је сакупљању кругова у детекцији ограничења, с тим што се прикупљање кругова ради преко функције *cv::HoughCircles*, док се сакупљање линија ради уз помоћ *cv::HoughLines*. Сакупљене линије се процесуирају одабирањем линија које припадају левој и десној (подела у две групе вектора) страни региона од интереса. Ова два подрегиона даље бирају које од сакупљених линија могу представљати линију траке пута тако што се рачуна нагиб сваке од линија, који мора да буде изнад 0.7 по апсолутној вредности, где 1 представља савршену вертикалну линију. Овим путем се избегава детекција хоризонталних линија које би се из различитих разлога могле наћи на путу (нпр. разграничење два смера раскрснице код семафора).

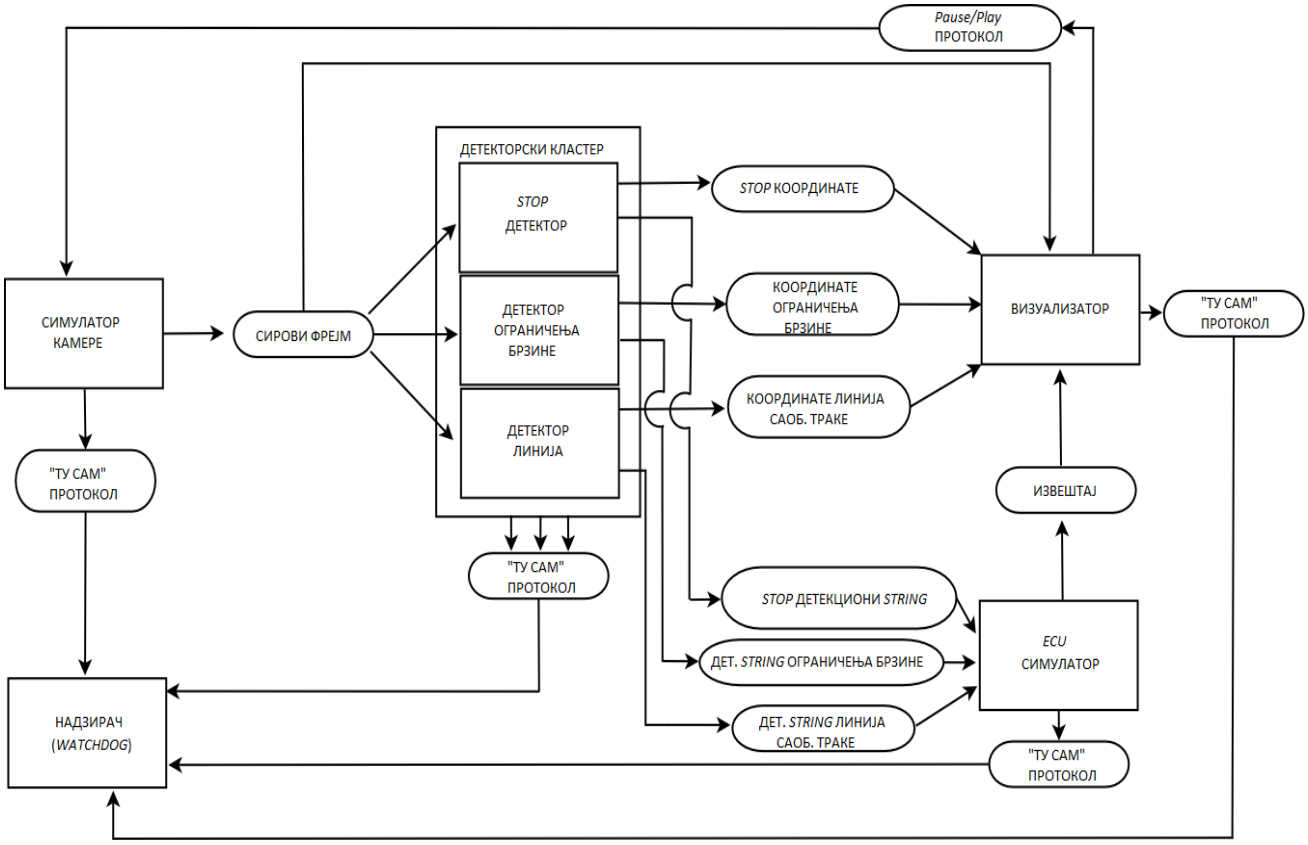
Овако одабране линије даље омогућују апроксимацију коначних јединствених линија са леве и десне стране, које одређују саобраћајну траку. Како је линија некада испрекидана са барем једне од страна, може доћи до нагле промене нагиба апроксимиране линије, што је вид грешке који се јавља при овом случају. Међутим, поред потенцијалног унапређења, могу се употребити различити софтверски алати како би се решио проблем који настаје када је линија испрекидана. Нпр. направити алгоритам очекивања, где уколико се између фрејмова појави један фрејм у којем нагиб нагло

одскочи, долази до игнорисања истог фрејма, те ће возило наставити несметано да се креће напред.

Слика 20. Коначна визуализација саобраћајних трака

5. ИНТЕГРАЦИЈА ЦЕЛОГ СИСТЕМА

Након што су побројани и објашњени индивидуални процеси који се одвијају током рада система, потребно је приказати како су ови, али и други процеси који се одвијају у позадини, повезани и успешно интегрисани у заједнички систем. Цео систем се састоји из седам нодова, од тога три чине један кластер (енг. *Detection Cluster*, нодови за детекцију), чији рад је допуњен преосталим нодовима: Симулатор камере (енг. *Camera Simulator*), *ECU* симулатор, Визуализатор (енг. *Visualizer*) и Надзирач (енг. *Watchdog*[1]).



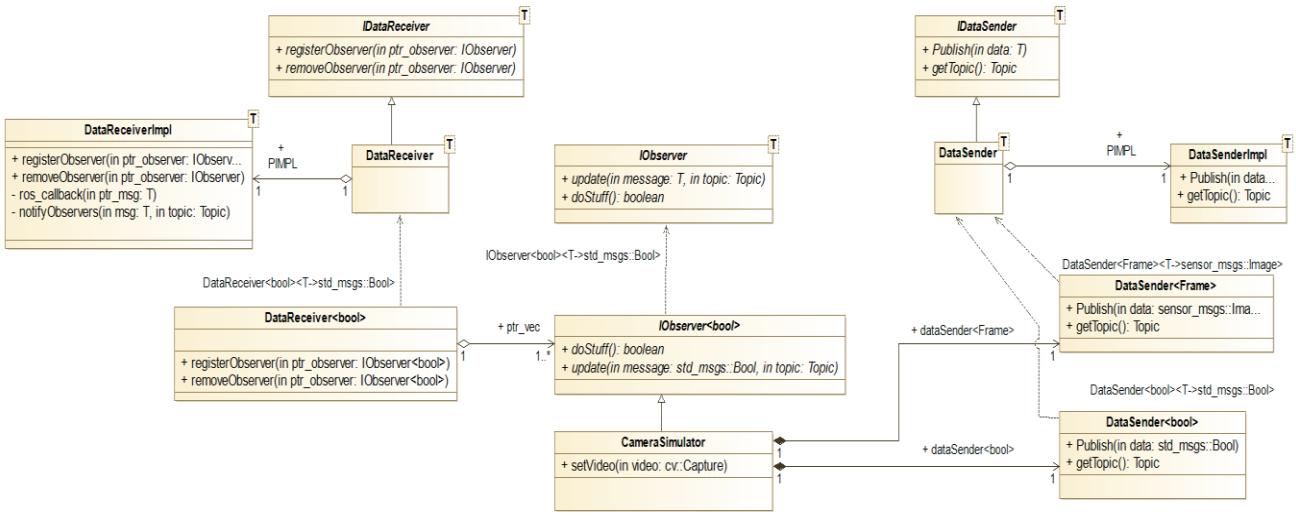
Слика 21. Релације између нодова

[1] Буквалан превод назива *Watchdog* био би „пас чувар“, међутим употребљен је термин „надзирач“ јер се ради оноду који надзире рад целог система и реагује уколико дође до неправилности

5.1 Симулатор камере

Овај нод је, како му само име говори, задужен за симулирање рада камере која је у реалним условима причвршћена на предњи део аутомобила. Симулације се обавља тако што се ноду уместо директног видео преноса (енг. *Video stream*) вожње прослеђује снимак који је забележила камера у реалним условима саобраћаја. Нод се покреће као и било који програм на *Linux*-у, с тим што му се као аргумент прслеђује и апсолутна путања на којој се налази видео. *String* путање виде се прослеђује *OpenCV* структури *cv::VideoCapture* чија инстанца је задужена да видео шаље ка следећем ноду "фрејм по фрејм" при свакој итерацији (циклусу).

ROS платформа је компатибилна са *OpenCV*-јем, те је публиковање фрејма на протокол обављено уз помоћ *cv_bridge*-а, који представља конвертор *OpenCV* инстанце фрејма (*cv::Mat*) у стандардну *ROS* поруку *sensor_msgs.* Апликација (нод) је такође задужена за проверу да ли је видео запис на наведеној адреси, као и да ли је одговарајућег формата (било који видео формат).



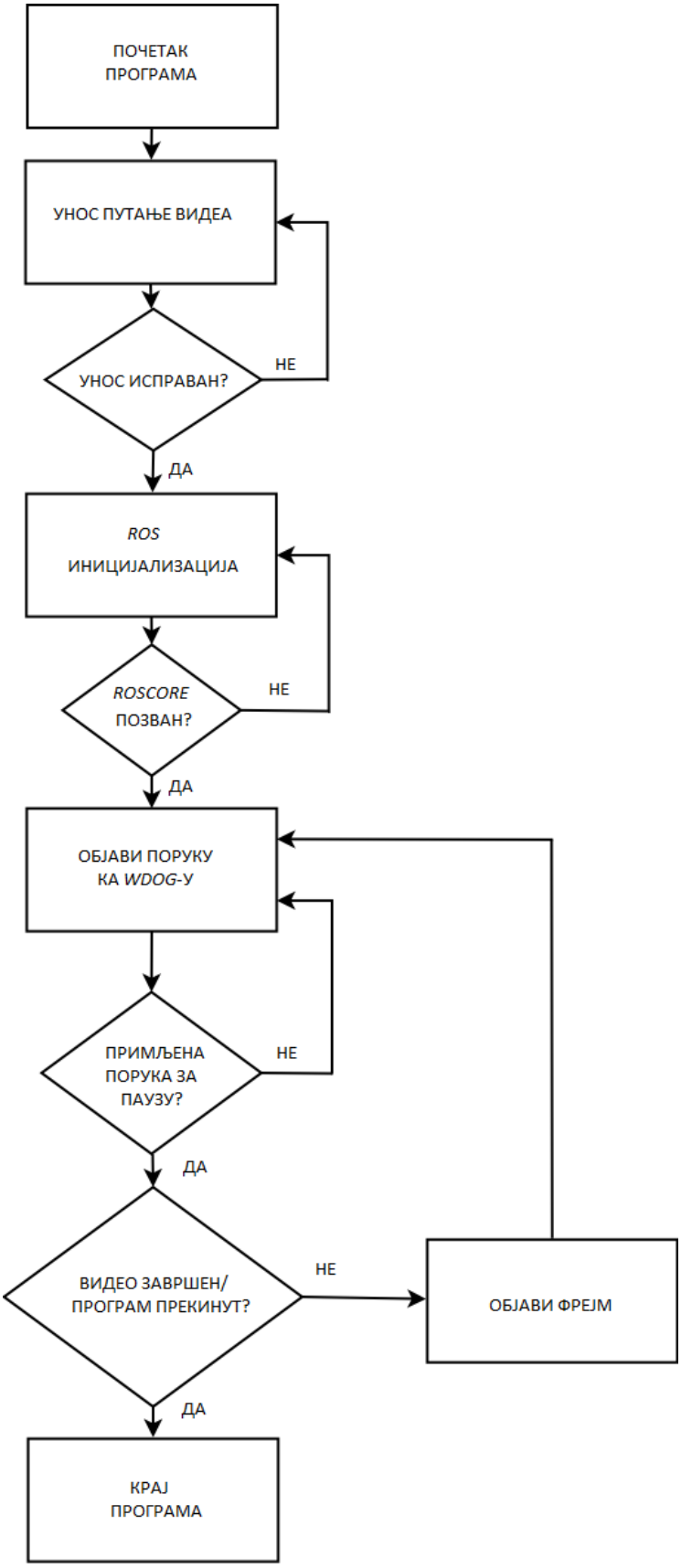
Слика 22. Класни дијаграм нода Симуалтора камере

Овај нод функционише користећи класе: *Camera Simulator*, *Sender* и *Receiver*. Класа *Camera Simulator* наслеђује *IObserver* класу која представља интерфејс апстрактног типа који прима једну или више порука. У питању је образац (енг. *template*) класа која нема унапред декларисан тип променљиве која јој се прослеђује, што заправо омогућава њено функционисање са различитим спектром улазних променљивих, уместо да се пише посебна класа за сваки тип. Ово је изузетно корисна могућност у објектно оријентисаној

имплементацији система. Класа *Camera Simulator* у овом случају наслеђује само *bool* тип *IObserver*-а, јер прима поруку само од визуализатора. Ова класа повезана је помоћу већ постојећег и веома познатог посматрачког шаблона (енг. *Observer Pattern*) са *Receiver* класом.

Receiver је класа је задужена за пријем поруке од визуализаторског нода и унутар себе има имплементирану *ROS* архитектуру задужену за пријем поруке, што је замаскирано преко *PIMPL Idiom*-а. Ово је још једна у низу изузетно корисних могућности у *OOP*, где зависност креираног система од платформе на којој је реализован не долази до изражаја, јер се уз минималне промене може омогућити рад система на некој другој платформи. Цела логика остаје готово нетакнута, а неопходно је променити два *source* фајла. Поступак је такав да је унутар ове класе декларисана још једна класа унутар *header* фајла, коју види само главна класа, целокупна дефиниција поткласе је написана у *.cpp* фајлу, а потклaсом се управља помоћу показивача на њу, члана главне класе. Дакле, *Receiver* прима поруку помоћу поткласе, а помоћу *Observer Pattern*-а ажурира промене у *Camera Simulator*-у.

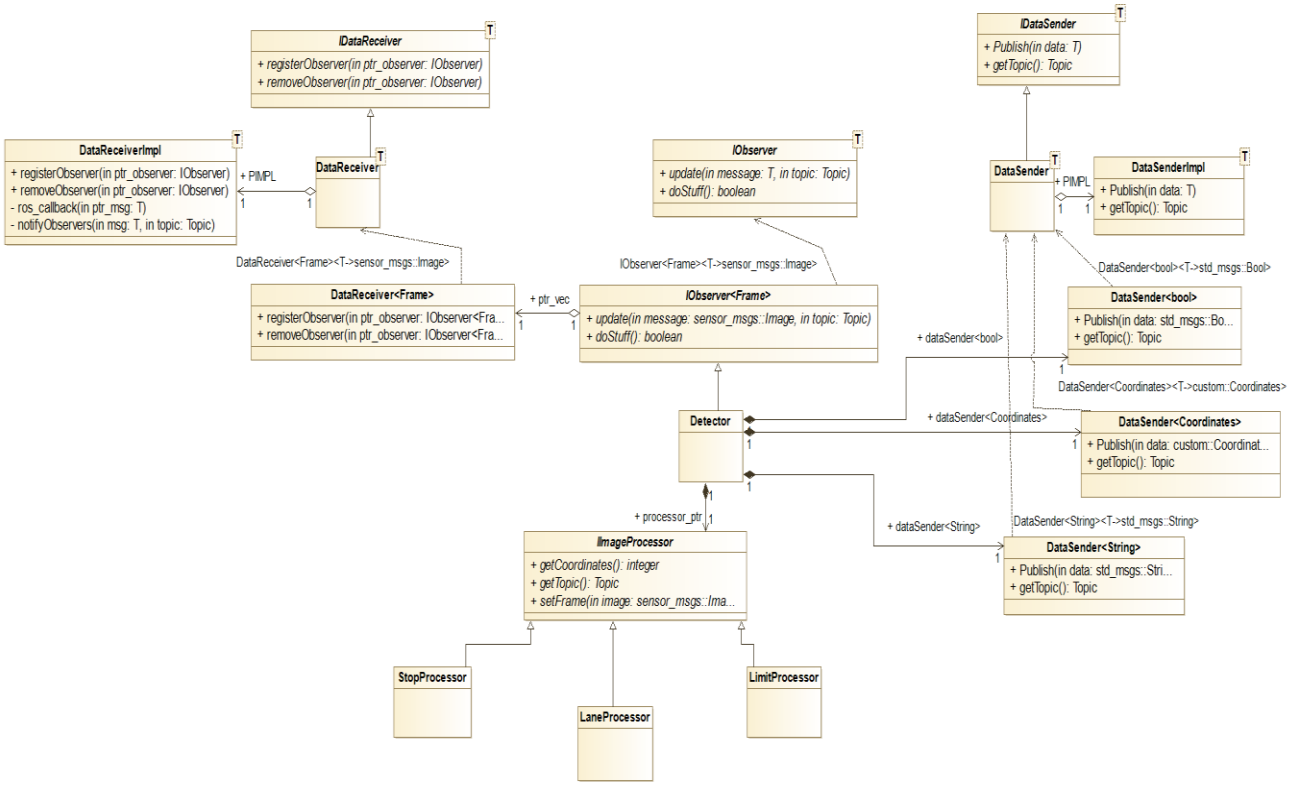
Sender је класа задужена за публиковање жељених података ка другим нодовима. Као и претходне две класе, у питању је *template* класа, док је *ROS* платформа замаскирана *PIMPL*-ом и у овом случају. Имплементирана је као композиција *Camera Simulator*-а, који је користи у две сврхе (два засебна члана), где је први задужен за слање *bool* поруке ка *Watchdog* ноду, а други је задужен за публиковање самог фрејма.



Слика 23. Алгоритам рада нода Симулатора камере

5.2 Кластер детектора

Кластер се састоји из три нода који се због сличног начина рада и имплементације могу описати у оквиру истог поглавља. Принципски, овај део задатка је могао бити реализован у оквиру једног нода који би секвенцијално обављао све операције детекције, међутим овај начин реализације би значајно успорио рад система, пре свега због *OCR*-а и класификатора, те је ради боље оптимизације рада система одлучено да се овај део система издели на три подсистема (нода) који ће паралелно обављати задате функције. Ова три нода поседују идентичну архитектуру, а разлику представља класа за обраду слике, као и прослеђивање параметара у виду протокола на који се обрађени подаци даље шаљу.



Слика 24. Класни дијаграм Детектор нодова

Кластер чине нодови за детекцију *STOP* знака, ограничења, као и линија саобраћајне траке. Сва три нода поседују следеће класе: *Receiver*, *Sender*, *Detector*, као и *Image Processor*. *Receiver* и *Sender*, су већ описани у склопу претходног нода, те им овде неће бити посвећена додатна пажња, јер је њихова улога у оквиру овог нода иста.

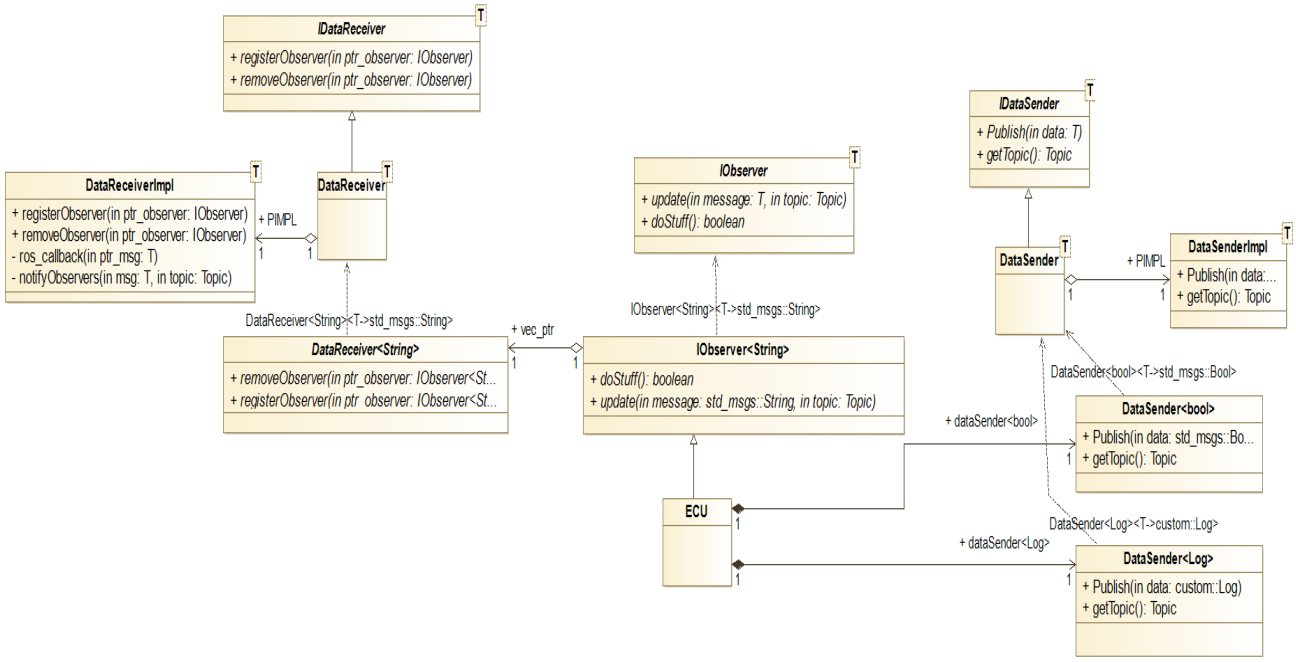
Receiver прима фрејм и прослеђује *IObserver*-u (у овом случају га наслеђује *Detector*), док један *Sender* шаље поруку ка ноду задуженом за визуализацију, други ка *ECU*, а трећи ка *Watchdog*-у. *Watchdog* и у овом случају прима *bool* поруку, *ECU* прима *string* детекције (да ли је *STOP* детектован, кретање у односу на линије на путу, као и очитано ограничење брзине), међутим за визуализацију детектованих објеката и линија шаље се сет координата на којима се исти налазе на слици (фрејму). Овакав тип поруке не спада у стандардну *ROS* поруку, већ је у питању дизајнирана порука (*custom*) коју је могуће креирати у *ROS-y*. Ова порука поседује 5 променљивих, све су типа *integer* и редом представљају: величину низа – број сет координата који се шаље, док остале представљају две тачке, односно њихове *x* и *y* координате. Свака од ове четири координате је заправо низ *integer*-а (*STL* вектора). Предност коришћења *STL* вектора у овом случају јесте то што није потребно унапред знати дужину низа, већ се врши динамичка алокација меморије.

Detector класа је задужена за пријем и слање порука, тачније за прослеђивање, док је *Image Processor* класа имплементирана као њена зависност (енг. *Dependency Injection*). *Detector* класа која добија сиров фрејм који прослеђује својој зависности (*Image Processor*) која на фрејму извршава детекцију објеката од интереса, те враћа координате маркираног објекта, али и закључак детекције у виду *stringa* који иде на *ECU* нод. *Detector* добијене променљиве пакује у *ROS* поруке и публикује. *Image Processor* представља интерфејс класу апстрактног типа коју наслеђују *Stop, Lane* и *Limit Processor*. Као и у претходном ноду, класе које наслеђују апстрактни интерфејс морају да дефинишу (енг. *Override*) методе које има апстрактна класа, а у њиховој декларацији има придодат назив *virtual* и изједначене су са нулом. Овај принцип (*Dependency Injection*) одликује динамички полиморфизам, који представља суштину *OOP*, где се уз помоћ декларисаног интерфејса као члана неке класе може користити било која класа која га наслеђује.

Слика 25. Алгоритам рада Детектор нодова

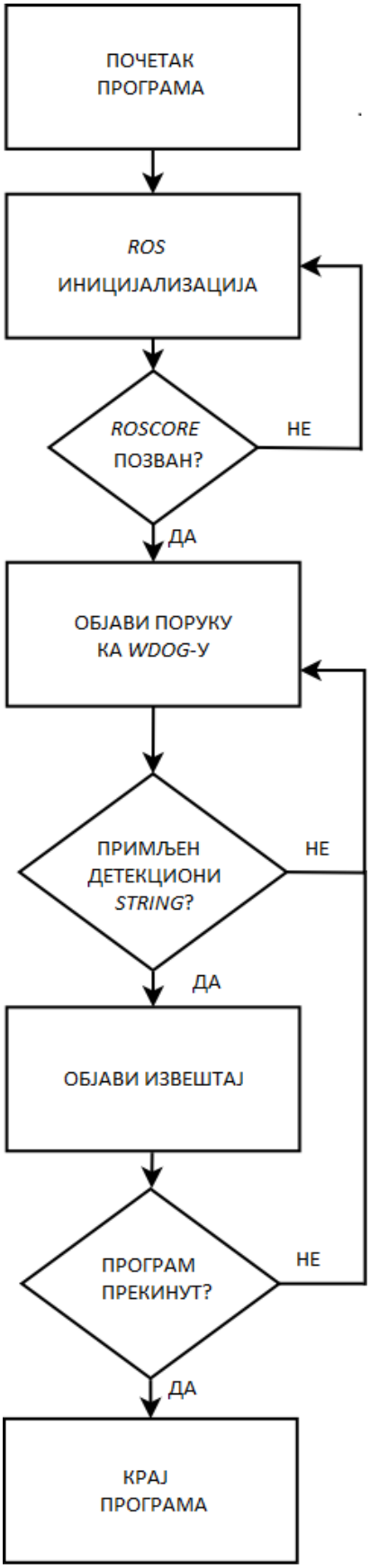
5.3 *ECU* Симулатор

У правом аутономном возилу *ECU* би имао много значајнију улогу јер не би само управљао целокупном сензориком и скупљањем података о околини возила, већ би био одговоран и за рад актуатора који би управљали смером и брзином возила, што су области саме за себе и нису покривене у оквиру овог пројекта. Стога овај нод има само симболичну улогу, а то је да на основу примљених резултата детекције шаље информације ка визуализатору за исписивање.



Слика 26. Класни дијаграм ECU симулатора

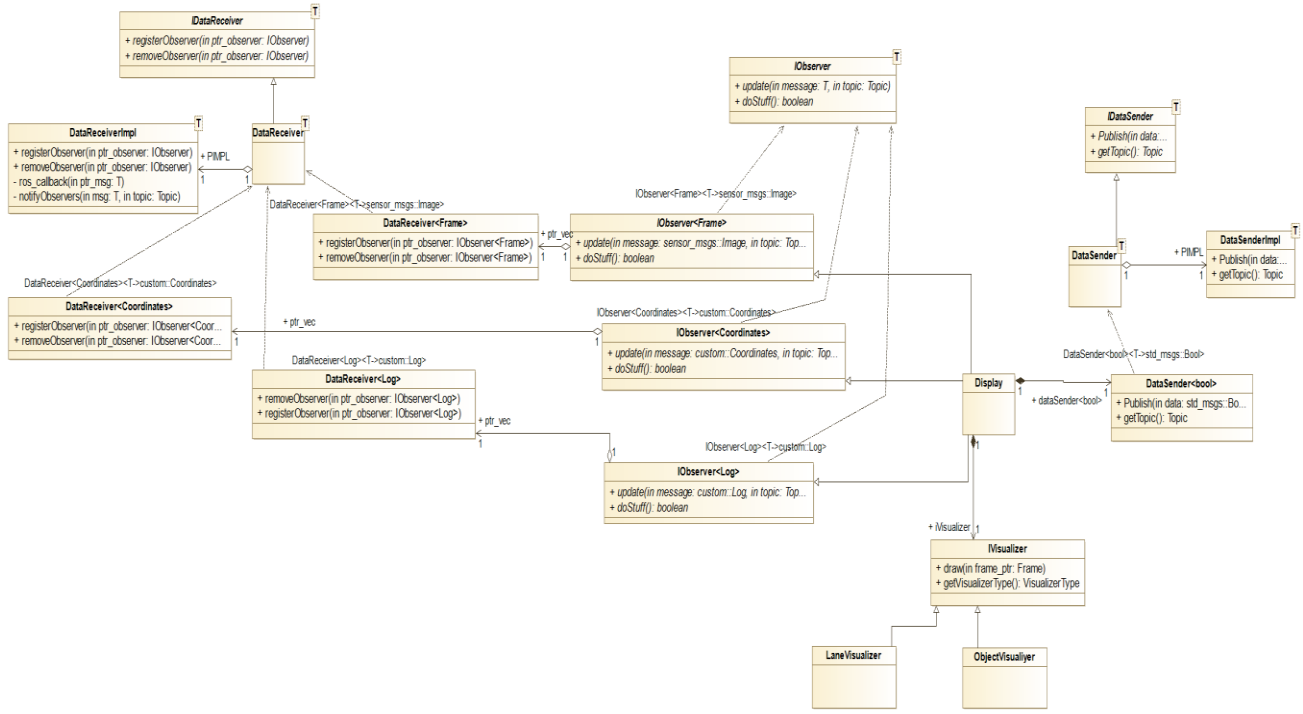
Класе које поседује овај нод су: *ECU* (која налеђује *IObserver*), *Receiver* и *Sender* ка *Watchdog-y* и визуализатору. Ка визуализатору се и овог пута прослеђује *custom* порука која се састоји из *bool* променљиве намењене *STOP* знаку, *integer* променљиве за ограничење и *stringa* за кретање. Назив генерисане поруке је *ECU Log*.



Слика 27. Алгоритам рада ECU симуалтора

5.4 Визуализатор

Главно задужење овог нода јесте преузимање сировог фрејма, на којем ће после обраде бити исцртани правоугаоници на месту детектованих саобраћајних знакова и линије на месту хоризонталне сигнализације на путу. Правоугаоници се цртају тако што му се прослеђују *x* и *y* координате два дијагонална наспрамна темена правоугаоника, док се што се тиче линија на путу, црта линија између две прослеђене тачке.



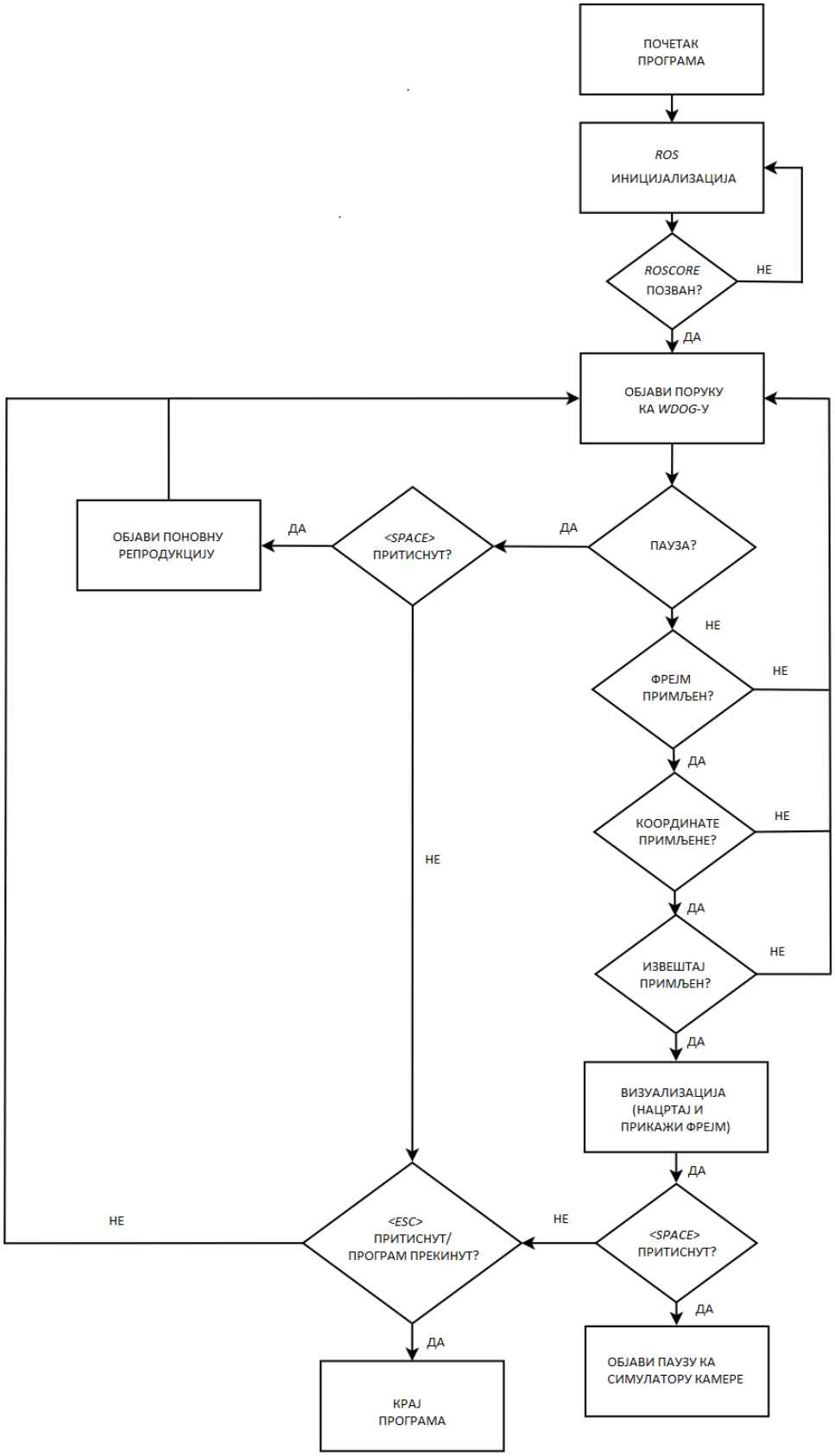
Слика 28. Класни дијаграм Визуализатора

Нод у оквиру своје архитектуре поседује *Display* класу, која наслеђује *IObserver* интерфејс и прима координате, сирови фрејм, као и извештај са *ECU*-а поново помоћу *Reciever*-а. Ова класа шаље *bool* поруку ка симулатору камере, а та порука представља паузирање или наставак репродукције видео снимка, те такође извештава и *Watchdog* да је присутан.

Сама екранизација обрађеног снимка извршава се посредством *OpenCV* функције *cv::imshow* која отвара нови прозор и приказује фрејм, а уз помоћ *cv::waitKey* одређује се *fps* ток снимка. Последња функција при отвореном прозору дисплеја притиском дугмета тастатуре враћа *char* (*ASCII*) вредност карактера. У прављењу једноставне апликације довољно је одредити *ASCII* вредност која наизменично поставља вредност *fps* на 0 или неку другу вредност како би се видео паузирао или наставила његова репродукција. У

овом случају то није могуће јер би се паузирао приказ, али се не би паузирало слање и обрада фрејмова. Неопходно је ставити до знања симулатору камере да престане са слањем фрејмова. То се постиже слањем поруке од стране визуализатора при притиску <*space*> тастера.

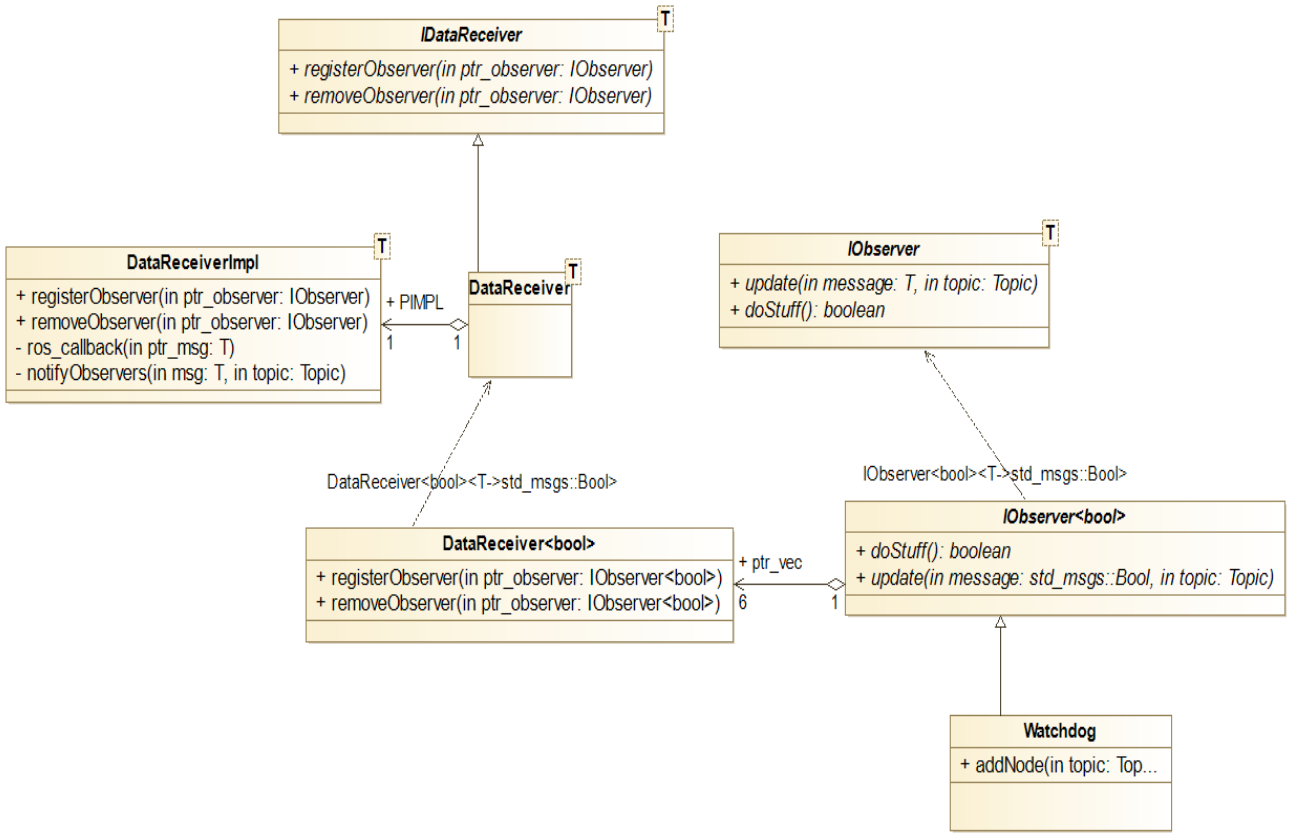
Примљени фрејм и координате се прослеђују *IVisualizer* класи која је интерфејс класа задужена за исцртавање детектованог и асоцијацијом је додељена *Display* класи. За исцртавање се такође користе функције из *OpenCV* библиотеке. Након што је исцртавање извршено преостаје приказ обрађеног фрејма.



Слика 29. Алгоритам рада Визуализатора

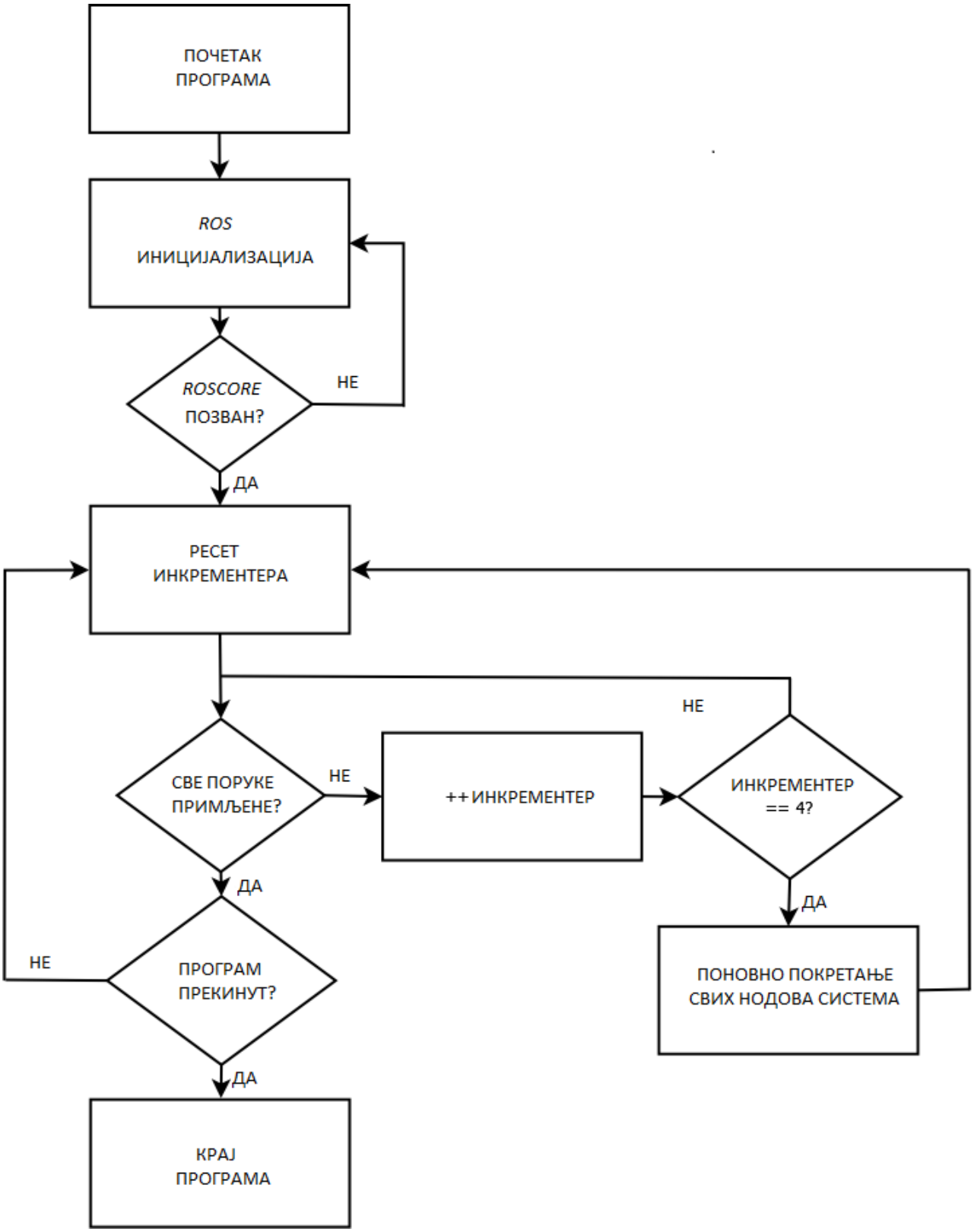
5.5 Надзирач (*Watchdog*)

Овај нод је добио име по псу чувару који надгледа цео систем (енг. *Watchdog*) и реагује уколико дође до дисконтинуације у раду неког од нодова. Наиме, сви нодови у оквиру система шаљу *bool* поруке овом ноду чиме му јављају да су и даље пристуни и да обављају своју функцију. Поруке послате овом ноду при свакој итерацији постављају *bool* чланове његове класе (*Watchdog*) на *true*, а уколико су све *bool* вредности логичке јединице (енг. *true*) нод закључује да систем у овој итерацији исправно и правовремено функционише те вредности поново ресетује на логичку нулу (енг. *false*). Пошто сваки од нодова ради на различитој учестаности толеранција *Watchdog*-а је максимум од четири итерације у којима све примљене поруке нису *true*. Уколико пак и након четврте узастопне итерације ово није случај, овај нод ресетује цео систем, тачније сваки нод убија (искључује), а затим га поново покреће. Убијање и поновно покретање нодова извршава се помоћу *Linux*-ове *system* функције која прослеђени *string* директно уноси у терминал.



Слика 30. Класни дијаграм Watchdog нода

Поред главне класе (*Watchdog*), овај нод поседује и *Reciever* класе које логично служе да му проследе неопходне поруке са сваког нода.



Слика 31. Алгоритам рада Watchdog нода

6.ТЕСТИРАЊЕ СИСТЕМА

У *Automotive* индустрији једна од круцијалних грана јесте *QA* (енг. *Quality Assurance*, Осигурање квалитета) као крунски показатељ исправног рада система. Овај сегмент представља процес тестирања којим се осигурава исправно функционисање готово сваке линије кода. Сваки програмер отпочиње писање кода са јасном замисли о томе шта тачно писани код треба да извршава али готово неминовно долази до одступања које је потребно у току тестирања уклонити. Важно је напоменути да постоји разлика између симулационих тестова и физичких тестова перформанси возила.

У *C++* програмском језику тестирање се извршава помоћу интеграционих тестова познатијих као *Unity Tests.* Ови тестови се могу поделити на *GTest* и *GMock*. *GTest* је задужен за тестирање понашања целокупних и појединачних функција система или класе, док је *GMock* задужен за тестирање зависности класе. У тзв. *mock*-овању *Mock* класа наслеђује интерфејс зависности главне класе и генерише *mock*-новане методе. Тестирање се своди на постављање параметара и очекивање исхода у складу са задатим параметрима. Уколико су сва очекивања испуњена, може се закључити да је тест имао успешан исход.

Целокупни пројекат прошао је кроз интеграционо тестирање.

7. ЗАКЉУЧАК

Коначно тестирање показало је да су сви елементи система успешно интегрисани у једну синхронизовану целину која тражени задатак обавља са високим процентом тачности. Обезбеђен је поуздан рад свих нодова, па чак и метода која би их реиницијализовала у случају да не јављају да могу да испуне свој задатак. Прослеђивањем различитих снимака написаном програму установљено је да се сви задати елементи система успешно реализују. Детектују се *STOP* знаци, знаци ограничења брзине, која год да је брзина на њима написана, а такође је остварено праћење линија које омогућава аутомобилу да увек остане у својој траци.

Једна од предности представљене реализације система јесте и коришћење паметних показивача (енг. *Smart Pointer*) који се увек сами деалоцирају што спречава евентуално нагомилавање смећа на динамичкој меморији (енг. *heap*). Пре свега, смањена је вероватноћа да ће *developer* направити такву грешку.

Тачност детекције би се могла појачати уколико би у система била интегрисана нека од модерних платформи као што је *Tensorflow*, *YOLO* или *RCNN*. Међутим, свака од ових алатки изискује хардвер изузетно високих перформанси, а првенствено употребу *GPU* картица. Неке од ових библиотека су искључиве и по питању типа картице, те *Tensorflow* ради са *nVidia* грфичким картицама што повећава трошкове реализације пројекта.

Такође, тачност детекције могуће је унапредити преко постојећих класификатора употребом већег тренинг сета слика. Ово изискује знатно већи број позитива, а самим тим и знатно већи број негатива, чије би тренирање могло потрајати и по неколико недеља. Важно је нагласити да процес тренирања модела постоји и код горенаведених алтернативних приступа.

Још један начин побољшања перформанси рада система био би укључивање више тредова (енг. *Multithreading*). Овај поступак није коришћен унутар овог пројекта због *ROS* платформе. Наиме, примећено је да имплементацијом *thread*-ова и *mutex*-а долази до проблема при раду *ROS*-а.

8. РЕФЕРЕНЦЕ

[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*, приступљено: јануар 2020.

[2] http://wiki.ros.org/ROS/Introduction - “*What is ROS? ROS Introduction*”, Званична *ROS* документациона страница, приступљено: јануар 2020.

[3] http://wiki.ros.org/ROS/CommandLineTools - “*ROS Command-line tools*”, Званична *ROS* документациона страница, приступљено: јануар 2020.

[4] http://wiki.ros.org/Nodes - “*ROS Nodes*”, Званична *ROS* документациона страница, приступљено: јануар 2020.

[5] https://en.wikipedia.org/wiki/OpenCV - “*OpenCV - from Wikipedia, the free encyclopedia*”, *Wikipedia* чланак, приступљено: јануар 2020.

[6] https://github.com/tesseract-ocr/tessdoc - “*Tesseract User Manual* “, Званични *github* репозиторијум, приступљено: јануар 2020.

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