Detekcija linija voznih traka na slici pomoću FPGA

Martin, Martin

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

Sveučilišni studij

DETEKCIJA LINIJA VOZNIH TRAKA NA SLICI POMOĆU FPGA

Diplomski rad

Martin Martin

Osijek, 2020.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 28.09.2020.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Martin Martin
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-998R, 21.09.2019.
OIB studenta:	28901326120
Mentor:	Izv.prof.dr.sc. Ratko Grbić
Sumentor:	Leon Šneler
Sumentor iz tvrtke:	Dario Pović
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Marijan Herceg
Član Povjerenstva 1:	Izv.prof.dr.sc. Ratko Grbić
Član Povjerenstva 2:	Izv. prof. dr. sc. Mario Vranješ
Naslov diplomskog rada:	Detekcija linija voznih traka na slici pomoću FPGA
Znanstvena grana rada:	**Arhitektura računalnih sustava (zn. polje računarstvo)**
Zadatak diplomskog rada:	Kako bi vozilo postalo autonomno, jedan od zadataka koji mora samostalno obavljati je i zadržavati se u vlastitoj voznoj traci i prestrojavati se kada za to postoje potreba i mogućnost. Kako bi vozilo moglo obavljati te funkcije, mora moći prepoznati linije voznih traka na cesti. U sklopu ovog diplomskog rada potrebno je izraditi algoritam za detekciju linija voznih traka na cesti ispred vozila, koristeći pritom FPGA za digitalnu obradu slike s kamere koja se nalazi na prednjem dijelu vozila. Algoritam mora pravilno detektirati položaj linija voznih traka na cesti. Nakon razvoja algoritma u Matlabu, potrebno je izraditi digitalni dizajn u VHDL jeziku koji opisuje navedeni algoritam, a zatim izvršiti njegovu sintezu i optimizaciju u ZYNQ SoC FPGA. Nakon
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	28.09.2020.

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

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

IZJAVA O ORIGINALNOSTI RADA

Osijek, 06.10.2020.

Ime i prezime studenta:	Martin Martin
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-998R, 21.09.2019.
Turnitin podudaranje [%]:	2

Ovom izjavom izjavljujem da je rad pod nazivom**: Detekcija linija voznih traka na slici pomoću FPGA**

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

i sumentora Leon Šneler

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

Sadržaj

1. UVOD

Kako bi vozilo postalo autonomno, jedan od zadataka koji mora samostalno obavljati je zadržavati se u vlastitoj voznoj traci i prestrojavati se kada za to postoji potreba i mogućnost. Detekcija voznih traka je dio jednog od mnogih podsustava korištenih unutar sustava za pomoć vozaču (engl. *Advanced Driver – Assistance Systems* – ADAS) kao što su sustavi upozorenja o odlasku s prometne trake (engl. *Lane Departure Warning System* – LDWS) i sustavi zadržavanja u prometnoj traci (engl. *Lane Keeping Assist System* - LKAS). Potonji podsustav se sastoji od detekcije žutih i bijelih linija na cesti te u slučaju prelaska preko jedne od njih obavještava se vozač s nekim od vizualnih ili zvučnih signala. Detekcija voznih traka je dosta zahtjevan proces u kojem je vrijeme potrebno za detekciju od iznimne važnosti. Iz tog razloga kako bi se rasteretili računalni sustavi, izrađuju se sustavi posebne namjene, kao što su sustavi programibilnih polja logičkih sklopova (engl. *Field Programmable Gate Array* – FPGA), a time se postižu i veće brzine detekcije.

Korištene metode za postizanje autonomnosti vozila mogu se svrstati u dvije skupine: klasične metode iz područja računalnog vida i moderne metode koje se temelje na strojnom učenju. Iako strojno učenje ima široko područje mogućnosti, klasične metode su u nekim područjima i dalje zastupljenije zbog niske kompleksnosti zadatka. Detekcija voznih linija je odličan primjer odabira klasične metode iz razloga što ne zahtijeva pronalazak kompleksnih oblika sa slike nego detekciju pravca što je znatno jednostavniji zadatak. U ovom radu korištena je Hough-ova transformacija za detekciju voznih linija što pripada jednoj od klasičnih metoda računalnog vida.

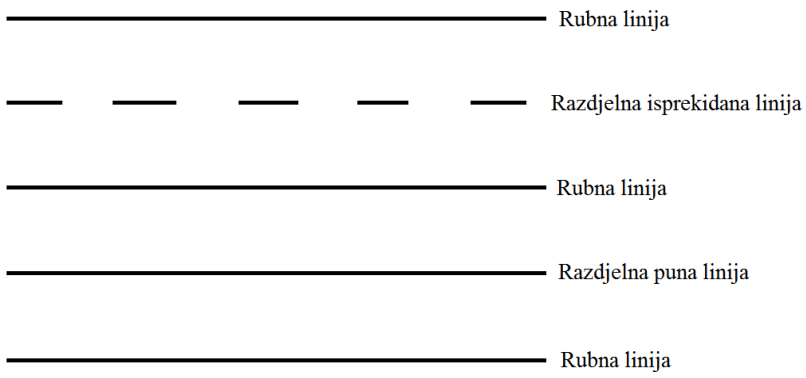
U okviru ovog diplomskog rada realizirana je detekcija voznih traka sa slike koristeći Zynq-7000 sustav na čipu (engl. *System on Chip* - SoC). Predložen je algoritam koji detektira uzdužne kolničke linije, a temelji se na klasičnim metodama obrade slike iz područja računalnog vida. Nakon izrade dizajna algoritma izvršena je sinteza i optimizacija, a nakon toga implementacija sintetiziranog dizajna na FPGA. Uz uspješnu implementaciju sustava na FPGA izvršena je verifikacija predloženog rješenja te mjerenja vezana za brzinu obrade slike i postotak uspješno detektiranih voznih linija. Mjerenja su provedena nad slikama različite rezolucije, a postignuti rezultati su uspoređeni s Matlab implementacijom predloženog algoritma.

Drugo poglavlje ovog diplomskog rada daje pregled postojećih radova usko vezanih uz problem detekcije voznih linija. U trećem poglavlju se navodi sva tehnologija i teorijska podloga korištena prilikom izrade te samo rješenje sustava. U četvrtom poglavlju napravljena je verifikacija

predloženog rješenja i odgovarajuća rasprava postignutih rezultata. Na kraju je dan zaključak rada te su navedene smjernice za danja poboljšanja predloženog rješenja.

2. DETEKCIJA VOZNIH TRAKA

Vozna traka je dio kolnika koji se nalazi između dviju voznih linija. Vozne linije na kolniku se najčešće pojavljuju kao žute ili bijele uzdužne linije koje mogu biti isprekidane ili pune. Prema [1] razlikujemo 3 vrste linija: rubna isprekidana linija ili puna razdjelna linija. Rubne linije su one linije koje se nalaze na krajnje desnoj ili krajnje lijevoj strani kolnika dok razdjelne linije predstavljaju linije koje razdjeljuju kolnik prema smjeru kretanja vozila. Primjer kolnika s označenim nazivima linija može se vidjeti na slici 2.1.



Sl. 2.1. Nazivi voznih linija na kolniku.

U cilju razvoja autonomne vožnje, na vozila se postavljaju razni senzori poput kamera, radara (engl. *RAdio Detection And Ranging*) i lidar-a (engl. *Light Detection And Ranging*) koji za zadatak imaju percipiranje stanja okoline oko vozila. Za detekciju voznih traka od iznimne važnosti je kamera montirana na prednjoj strani vozila, koja iz okoline prikuplja slike u neposrednoj blizini ispred vozila. Slike se zatim obrađuju s nekim od algoritama za detekciju vozne trake kojem je u cilju izdvojiti područja na kojima se nalaze vozne linije od onih na kojima se ne nalazi. Područje koje se nakon detekcije nalazi između najbliže lijeve i najbliže desne detektirane vozne linije predstavlja voznu traku kojom se vozilo kreće.

Prije nego što se započne obrada slike moguće je smanjiti područje obrade na način da se dio slike, za koji se pretpostavlja da ne sadrži vozne linije, uopće ne obrađuje. Područja koja sa sigurnošću možemo odbaciti su primjerice nebo, horizont i sl. Područja u kojima je moguće pronaći elemente slike koji odgovaraju voznim linijama nazivamo područja od interesa (engl. *Region Of Interest* – ROI).

Prilikom detekcije voznih traka u slici odnosno linija koje omeđuju voznu traku, na sliku se najprije primjenjuju neke od metoda obrade slike. Na slici se osim voznih traka i kolnika nalaze i

mnogi drugi objekti. Takvi objekti predstavljaju neželjene informacije koje nazivamo šumovima. Šumovi su prisutni gotovo na svakoj slici koja se obrađuje, a moguće ih je ukloniti ili reducirati s nekom od metoda korištenih u računalnom vidu. Dodatno se na sliku primjenjuju metode računanja gradijenta koje ističu rubove prisutnim objektima na slici. Na takvo obrađenu sliku primjenjuje se neki od algoritama za pronalazak pravca koji se najbolje podudara s rubovima te u konačnici pronađeni pravac smatramo voznom linijom. Pronalaskom pravaca koji odgovaraju voznim linijama na desnoj i lijevoj polovini slike možemo odrediti voznu traku kao područje unutar slike koje se nalazi između tih pravaca.

S obzirom da se ovakav sustav gotovo uvijek postavlja na vozila, od iznimne su važnosti efikasnost algoritma te brzina obrade video okvira. Efikasnost algoritma za detekciju obično se izražava uz pomoć dvije metrike, preciznost (engl. *precison*) i odziv (engl. *recall*). Brzina obrade video okvira podrazumijeva koliko slika sustav može obraditi i najčešće se izražava u milisekundama ili broju okvira u sekundi (engl. *Frames Per Second* – FPS).

2.1. Postojeći algoritmi detekcije voznih traka

Iako algoritmi za detekciju pravaca na slikama postoje još od sredine sedamdesetih godina dvadesetog stoljeća prema [2] prvi primjeri korištenja takvih algoritama u automobilskoj industriji kreću tek početkom 21. stoljeća. Neki od značajnijih radova predstavljeni su u ovom poglavlju, a većina ih se bavi istom ili sličnom problematikom.

U radu [3] predstavljen je pronalazak voznih traka u stvarnom vremenu korištenjem FPGA tehnologije i procesora za obradu digitalnog signala (engl. *Digital Signal Processor* – DSP). U radu su najprije predstavljena dva moguća načina rješavanja zadanog problema koja se klasificiraju kao geometrijski orijentiran pristup detekcije te metoda detekcije oblika. Geometrijski orijentiran pristup detekcije voznih traka bazira se na korištenju Hough-ove transformacije koja kao zadatak ima pronalazak pravca koji se najbolje podudara s rubovima objekata na obrađivanoj slici. Rečeno je kako je geometrijski orijentiran pristup vrlo efikasna metoda detekcije voznih traka osim u slučaju intenzivnih smetnji kao što su sjena i automobili, ali i taj problem je rješiv uz prethodnu primjenu Gauss-ovog filtera na sliku. Ipak postoji mana u ovoj metodi, a to je ako je boja voznih linija vrlo slična dominantnoj boji okoline u kojoj se nalazi. Metoda detekcije oblika temelji se na pronalasku objekata na slici te se nakon izdvajanja tih objekata analizira radi li se o pravcima ili ne. Od svih izdvojenih objekata algoritmom k srednjih vrijednosti (engl. *K-means algorithm*) se bira onaj objekt koji se najbolje podudara s pravcem te se taj objekt smatra voznom linijom. U radu se tvrdi kako ova metoda može otkriti linije i u kompleksnim okruženjima

neovisno o sjeni i drugim automobilima, ali mana joj je što ima poteškoća prilikom pronalaska isprekidane linije. Autori su se u radu odlučili na korištenje geometrijski orijentiranog pristupa detekcije voznih traka uz korištenje FPGA i DSP DM642. FPGA se koristi za izdvajanje ROI te kao video sučelje, a DM642 izvršava Hough-ovu transformaciju i prikazuje rezultate. Rješenje je testirano na tri seta podataka, jedan je predstavljao ceste Changsha, a druga dva Washington i Cordobe. Sva tri seta podataka imaju veličine video okvira 640x480 piksela. Tablicom 2.1 prikazani su rezultati dobiveni testiranjem točnosti sustava, a u tablici 2.2 prikazano je vrijeme izvršavanja algoritama. Rezultati su pokazali kako je rješenje točno i brzo te se izračunom došlo do podatka obrade do 25 FPS za video okvire veličine 640x480 piksela Jedan nedostatak je nepotpuna iskorištenost FPGA sustava. Veći dio zadatka pridodan je DSP sustavu, a bolja raspodijeljenost zadatka bi dovela do ubrzanja ovakvog sustava.

Tablica 2.1 Performanse sustava s obzirom na točnost detekcije.

Set podataka	*Changsha*	*Cordoba*	*Washington*
Broj testiranih okvira	200	250	232
Točno detektirane vozne linije (%)	95	94	88.4
Netočno detektirane vozne linije (%)	8	6	11.6

Tablica 2.2 Performanse sustava s obzirom na brzinu obrade video okvira.

Sustav	*FPGA*	*DSP (DM642)*	*Ukupno*
Vrijeme (ms)	0.9	40	41

U radu [4] je predstavljena detekcija voznih traka na dvije razine uz korištenje ugradbene FPGA platforme. Prva razina detekcije zasniva se na Hough-ovoj transformaciji s ograničenim resursima pri normalnim uvjetima, a u slučaju kada Hough-ova transformacija nije dovoljno precizna aktivira se druga razina detekcije. Druga razina detekcije se primjenjuje samo na područja s predefiniranim dimenzijama te s ograničenim brojem mogućnosti detekcije voznih linija. Glavna karakteristika ovog rada je omjer između točnosti, vremena izvršavanja i količini raspoloživih resursa na sustavu kako bi se postiglo rješenje izvedivo u stvarnom vremenu. Slično kao i u radu [3] najprije se odredio ROI koji je ovisio o položaju i kutu postavljene kamere. U ovom radu ROI određen je ručno te se kao takvo koristilo za svaku pojedinu sliku. Odvojeno područje se zatim pretvara u slike s sivim tonovima (engl. *grayscale*) nad kojom se zatim izvršava nekoliko operacija. S ciljem pripreme slike za detekciju voznih linija najprije se provodi Gauss-ov filter. Gauss-ov filter je niskopropusni filter s kojim je moguće ukloniti detalje odnosno reducirati količinu šuma na slici. Nakon toga slijedi Sobel-ov algoritam za računanje gradijenta slike i naglašavanje rubova objekata unutar slike, a na kraju se primjenjuje Hough-ova transformacija za

detekciju pravaca na slici. U slučajevima kada Hough-ova transformacija nije u mogućnosti u potpunosti detektirati vozne linije primijenjeno je detektiranje voznih linija koje radi na principu pamćenja pozicija prethodno pronađenih voznih linija. Slike su na obradu dolazile brzinom od 25 FPS-a, pa se s tom činjenicom moglo zaključiti kako se pozicija vozne linije pronađene u prethodnoj slici neće uvelike razlikovati o poziciji trenutne slike. Arhitektura korištena u radu sastoji se od KC705 pločice sa XC7K325T-2FFG900C FPGA čipom. FPGA se sastoji od 326080 logičkih ćelija, 840 DSP blokova te 16020 kB memorije s nasumičnim pristupom (engl. *Random Access Memory* – RAM). Simulacija i sinteza navedenog sustava izvršena je unutar Vivado programskog paketa, a upravljanje se preko proširivih naprednih protokola (engl. *Advanced eXtensible Interface* - AXI) izvršilo korištenjem C programskog jezika. Za testiranje korišteno je preko 5000 slika kvalitete 720p u različitim uvjetima vožnje kao što su: autoput, vožnja noću, tunel, magla itd. Postignuti su iznimno dobri rezultati od prosječno 92.92% točnosti uz prosječnu brzinu od 60 FPS. Rezultati su prikazani na tablici 2.3. Ispostavilo se da je rješenje pronalaska voznih linija na principu pamćenja prethodnih pozicija, povećalo točnosti pronalaska s 80% na 90% u idealnim uvjetima.

Tablica 2.3 Performanse sustava s obzirom na točnost detekcije i brzine obrade video okvira.

Set podataka	*autocesta*	*dan*	*noć*	*sijena mosta*	*sijena drveća*	*Tunel*	*magla*	*Gužva*	*Prosjek*
Broj testiranih okvira	480	380	600	791	600	1350	910	400	5511
Točno detektirane vozne linije (%)	94.5	97.3	87.96	96.6	89.8	97.6	97.8	81.8	92.92
FPS	57.3	60.4	58.2	55	59	67.3	67.3	55.5	60

U radu [5], koji osim što se bavi detekcijom voznih linija u stvarnom vremenu, pokušava se primijeniti i Kalman-ov filter za praćenje voznih traka tijekom kretanja vozila. Kao i u do sada već predstavljenim radovima, detekcija voznih linija implementira se korištenjem Hough-ove transformacije, a novost u ovom radu je upravo Kalman-ov filter. Kalman-ov filter je efikasna metoda procjene bazirana na matematičkim formulama. Omogućava procjenu i ispravak stanja sustava, a primjenjuje se za praćenje prethodno detektiranih voznih linija. Za implementaciju sustava korišteno je ALTERA DE2 razvojno okruženje s Cyclon II EP2C35F672C FPGA čipom. Slike se na sustav dovode s digitalnog fotoaparata Nikon COOLPIX P510 spojenog NTSC konektorom te se spremaju na statičnu memoriju s nasumičnim pristupom (engl. *Static Random Access Memory* – SRAM), a razvojno okruženje je u svrhu testiranja spojeno s računalom koristeći RS232 sučelje. Postignuta brzina signala takta sustava iznosi 115 MHz uz zauzeće od 45% resursa

FPGA čipa. Brzina obrade iznosi 25 FPS-a s kvalitetom od 720p, prikazani su rezultati obrade slika, ali u rezultatima nije navedena točnost sustava u postotcima.

Za razliku od prethodnih radova [3] koji se uglavnom bave implementacijom Hough-ove transformacije na sustavu za detektiranje voznih linija uz dodatne metode/filtre, rad [6] se bavi optimizacijom Hough-ove transformacije prilikom detektiranja pravaca. Hough-ova transformacija za svoju točnost u radu zahtijeva veliku količinu memorije i procesorske snage, a u praksi se često koristi kao alat za pronalazak pravaca sa slike što se pak dalje primjenjuje za pronalazak voznih traka, prometnih znakova i sl. Problem nastaje kada se takav algoritam postavlja u automobile na ugradbene računalne sustave koji imaju ograničenu procesorsku snagu i ograničenu količinu memorije. Rad predlaže uvođenje tablica s vrijednostima (engl. *Look Up Table* – LUT) zbog velikog ponavljanja identičnih matematičkih operacija. Dodatan prijedlog je smanjenja potreba za velikom količinom memorije tako da se dodatno smanjuju ROI, a samim time postiže i manji broj podataka potrebnih za obradu. U ovom radu za implementaciju korišten je Xilinx Virtex-5 FPGA čip na ML505 razvojnom okruženju te je u svrhu testiranja algoritma montirana kamera s komplementarnim poluvodičima od metal-oksida (engl. *Complementary Metal Oxide Semiconductor* – CMOS) na vjetrobransko staklo automobila. Slika koju je kamera dobivala bila je veličine 650x480 piksela te je s njom izvršeno nekoliko testova. Rezultati su pokazali kako je točnost algoritma između 90% i 97% u ovisnosti o mjestu na kojem se testiralo, a postignuta brzina iznosila je 95 FPS-a, što je znatno veći napredak od prethodnih radova. Tablicom 2.4 prikazani su rezultati dobiveni testiranjem.

Tablica 2.4 Performanse sustava s obzirom na točnost detekcije i brzine obrade video okvira.

Set podataka	*Autocesta dan*	*Autocesta noć*	*Gradska cesta dan*	*Ukupno*
Broj testiranih okvira	2820	1955	1994	6769
Točno detektirane vozne linije (%)	96.6	96.75	90.09	93.48
FPS	93.2	96.1	95.7	95.03

Rad [7] daje pregled nedavnih dostignuća u detekciji voznih traka i sustava upozorenja u slučaju prelaska preko voznih traka. Autor u radu se najprije spominje moguće smetnje prilikom obrade slike kao što su: oštećen kolnik, sijena, različita osvjetljenja, kiša, magla itd. Zatim daje pregled do sad korištenih metoda otklanjanja ili ublažavanja smetnji. Navedene su neke od često korištenih metoda kao što su Gauss-ov filter, filter konačnog odziva impulsa (engl. *Finite Impulse Response* - FIR) itd., a spomenuta je i manje korištena metoda: funkcija komadnog linearnog istezanja, (engl. *Piecewise Linear Stretching Function* – PLSF) koja povećava kontrast pri tamnijim slikama tako da se poveća širina obrađivane slike. Kod ROI je rečeno kako je dovoljno

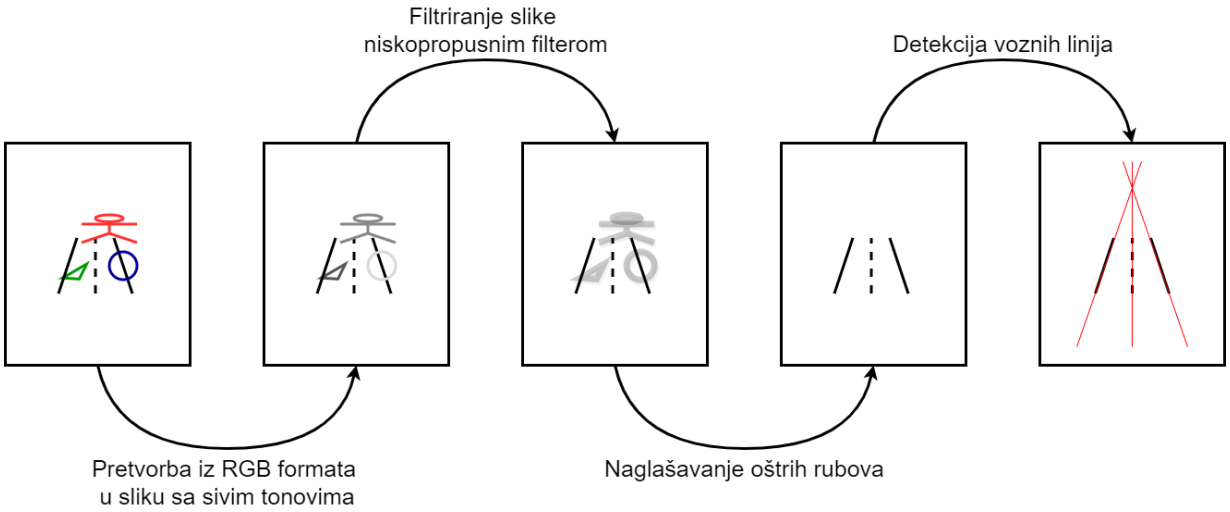
primjenjivati algoritam na donji dio slike jer je to područje gdje se mogu pronaći vozne linije. Algoritam za detekciju linija podijeljen je u dvije skupine: geometrijski orijentiran pristup detekcije te metoda detekcije oblika. Metode koje su navedene u radu su: Hough-ova transformacija, maksimalno stabilna ekstremna regija (engl. *Maximally Stable Extremal Region* - MSER), algoritam grupiranja (engl.– *Density Based Spatial Clustering of Applications with Noise* – DBSCAN) itd. Budući da su se u ovom radu najviše navodile tehnike i načini pristupa rješavanja ovakve vrste problematike, nije se izrađivalo vlastito rješenje nego je dana usporedba rješenja prijašnjih radova s obzirom na točnost pronalaska voznih linija.

Zadnji predstavljeni rad [7] uvelike je pomogao u donošenju odluke oko odabira vlastitog pristupa rješavanja ove problematike jer se u njemu mogla vidjeti usporedba do sad korištenih pristupa zadanom problemu. Stoga, vlastito rješenje detekcije vozne trake temelji se na klasičnim metodama računalnog vida za obradu slike i detekciju pravaca sa slike. Dalje u nastavku rada detaljnije je objašnjena svaka metoda pojedinačno te je predstavljen dizajn vlastitog rješenja.

3. ALGORITAM DETEKCIJE VOZNIH TRAKA I IZRADA VLASTITOG RJEŠENJA

3.1. Algoritam za detekciju voznih traka

Od primitka slike s kamere pa do detekcije voznih linija potrebno je izvršiti nekoliko postupaka obrade slike. Najprije je potrebo primljenu sliku iz RGB formata pretvoriti u sliku sa sivim tonovima, a zatim takvu sliku filtrirati niskopropusnim filtrom radi redukcije šuma na slici. Na filtriranu sliku primjenjuje se nekih od filtera za naglašavanje rubova te na kraju algoritam detekcije pravaca. Tijek obrade slike od primitka pa do detekcije voznih linija prikazan je na slici 3.1.



Sl. 3.1. Tijek obrade slike.

Kako bi detekcija voznih linija sa slika u ovom radu bila moguća, potrebno je dizajnirati sklopovsko rješenje temeljeno na nekim od metoda obrade slike. Dizajn predloženog rješenja je osmišljen te izrađen uz pomoć već postojećih radova [3], [6] i [7] te uz teorijsku podlogu korištenih metoda. Najprije se ulazna slika filtrira niskopropusnim Gauss-ovim filterom, zatim se naglašavaju oštri rubovi pomoću Sobel i Canny operatora, a detekcija voznih linija je realizirana koristeći Hough-ovu transformaciju. Važno je naglasiti da se metode obrade slike i metode detekcije pravaca ne izvršavaju na cijeloj slici nego samo na ROI dijelu slike. U ovom radu ROI se odnosio na donju polovicu slike iz razloga što se samo na toj polovici mogu naći vozne linije. Detalji o korištenim metodama objašnjeni su u nastavku.

3.1.1. Gauss-ov filter

Kako se detekcija voznih traka izvršava na slikama dobivenim iz okoline, ne može se sa sigurnošću znati da će se na slici nalaziti samo cesta, vozne trake i ništa drugo. Česti je slučaj da se osim željenih komponenti na slici nalaze i mnogi drugi detalji. Takve detalje na slici nazivamo šumovima i prilikom detekcije voznih traka u cilj je otkloniti ih ili reducirati kako bi se postigla što efikasnija detekcija voznih linija odnosno trake. Reduciranje takvih detalja u ovom radu postignuto je koristeći Gauss-ov filter. Gauss-ov filter je niskopropusni filter koji reducira količinu detalja na slikama na način da svaki element slike zamjeni sa srednjom vrijednošću susjednih elemenata. Na slici 3.2. dan je primjer primjene Gauss-ovog filtera na tipičnu sliku koja se dobiva s kamere montirane na prednjoj strani vozila [8].

a) b)

Sl. 3.2. a) izvorna slika, b) slika nakon primjene Gauss-ovog filtera.

Gauss-ovo filtriranje računa se jednadžbom

$$G(x,y) = \frac{1}{2\pi\sigma^2} e^{\frac{x^2+y^2}{2\sigma^2}} \tag{3-1}$$

gdje x i y označavaju poziciju centralnog elementa slike po x i y osima, a σ je standardna devijacija Gauss-ove funkcije. Dobiveni rezultat $G(x, y)$ predstavlja vrijednost elementa gdje vrijednost 0 predstavlja bijelu boju, a svaki veći broj predstavlja tamniju boju do maksimalne vrijednosti 255 koja predstavlja crnu boju. Budući da je slika skup diskretnih elemenata potrebno je aproksimirati Gauss-ov filter za računanje diskretnih funkcija. Gauss-ov filter može se aproksimirati matricom na nekoliko načina, a aproksimacija korištena u ovom radu prikazana je matricom (3-2).

$$G = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{3-2}$$

Element slike na koordinatama (*x, y*) sa njegovih osam susjednih elemenata čini matricu veličine 3x3, množenjem takve matrice s aproksimiranom Gauss-ovom matricom *G* dobije se filtrirana vrijednost za taj element. Ponavljajući postupak za sve elemente slike rezultat je filtrirana slika Gauss-ovim filterom.

3.1.2. Sobel-ov operator

Nakon što je sa slike reducirana količina detalja Gauss-ovim filterom, Sobel-ovim operatorom se računaju gradijenti slike te naglašavaju rubovi svih objekata prisutnih na slici. Sobel-ov operator radi na način da se za svaki element slike računa njegov gradijent intenziteta. Slika obrađena na ovakav način će zadržati sve one elemente na kojima dolazi do velikih promjena u odnosu na susjedne elemente, a to se najčešće nalazi na rubovima objekata na slici. Primjer slike obrađene Sobel-ovim operatorom prikazana je na slici 3.3.

a) b)

Sl. 3.3. a) slika nakon primjene Gauss-ovog filtera, b) slika nakon primjene Sobel-ovog operatora.

Sobel-ov operator definiran je dvjema matricama M_x i M_y veličine 3x3 (3-3) s kojima se dobivaju gradijenti u smjeru *x* i *y* osi.

$$M_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}, M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{3-3}$$

Ukupan gradijent elementa (*x, y*) računa se prema (3-4)

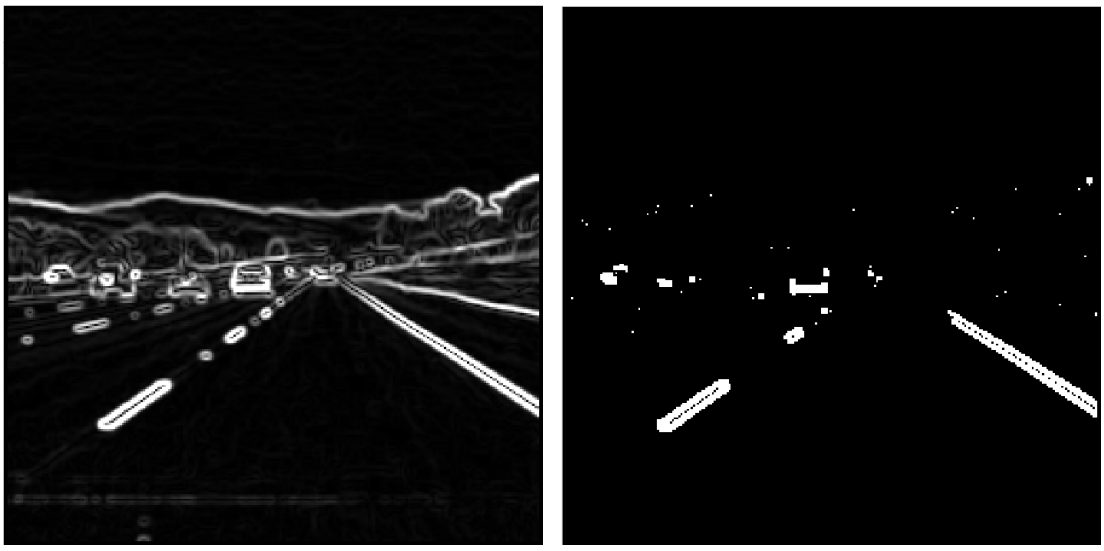
$$\begin{aligned} G_x &= M_x * Slika_x \\ G_y &= M_y * Slika_y \\ G(x,y) &= \sqrt{G_x^2 + G_y^2} \end{aligned} \tag{3-4}$$

a moguće je izračunati i smjer gradijenta prema (3-5):

$$\theta(x,y) = tan^{-1}\left(\frac{G_y}{G_x}\right) \tag{3-5}$$

3.1.3. Canny-ev operator

Sobel-ovim operatorom postiglo se detektiranje rubova svih objekata prisutnih na slici, a kako bi se ti rubovi izoštrili te se dodatno uklonio šum, na sliku se primjenjuje Canny-ev operator. Canny-ev operator se sastoji od dva dijela: najprije se detektirani rubovi stanjuju, a zatim se primjenjuje postupak otklanjanja šumova s dvije granice (engl. *Double Threshold*) [9]. Stanjivanje se obavlja na način da se za svaki element izvrši ispitivanje nalazi li se element na samom rubu uz pomoć smjera gradijenta θ (dobivenog iz Sobel-ovog operatora). Nakon stanjivanja, rubovi su izoštreni i „tanki“, ali postoji mogućnost da su neki od neželjenih detalja ostali na tako filtriranoj slici pa se stoga primjenjuje postupak otklanjanja šumova s dvije granice. Postupak ima u cilju ukloniti sve one gradijente koji su vrlo niske vrijednosti, a zadržati gradijente s visokim vrijednostima. Granice za visoke i niske vrijednosti su proizvoljne te se određuju po potrebi. Ako se neka vrijednost nađe između granica, ispituje se postoji li neki susjedni element koji ima vrijednost iznad visoke granice te u slučaju ako postoji takav element, element između granica se zadržava, a u protivnom odbacuje. Ovim postupkom se slika u sivim tonovima pretvara u binarnu



a) b)

Sl. 3.4. a) slika nakon primjene Sobel-ovog operatora, b) konačni rezultat primjene Canny-evog detektora rubova.

sliku. Binarna slika je slika na kojoj svaki piksel može poprimiti samo dvije vrijednosti 0 ili 255 odnosno crnu ili bijelu boju. Na slici 3.4. ilustrirana je primjena Canny operatora na ulaznu sliku koja je prethodno obrađena Sobel-ovim operatorom.

3.1.4. Houghova transformacija

Primjenom Canny-evog operatora završena je predobrada slike te kao rezultat je dobivena binarna slika s istaknutim rubovima. Kako su na binarnoj slici sve vrijednosti 0 ili 255 te kako sve vrijednosti 0 predstavljaju rubove nekih od objekata, na slici je sada moguće rubove tih objekata opisati nekim od geometrijskih oblika. Iz razloga što je zadatak ovog rada pronaći vozne linije koje se u blizini vozila najbolje mogu usporediti s pravcem, opisan je postupak pronalaska pravaca koji se najbolje poklapaju s voznim linijama koristeći se metodom Hough-ova transformacija.

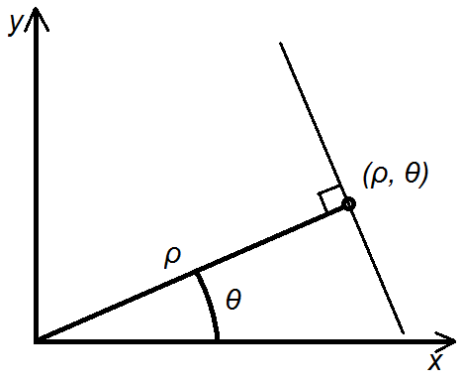
Hough-ova transformacija za pronalazak pravaca je jednostavna, ali memorijski zahtjevna metoda detekcije. Generalno pravac možemo opisati izrazom (3-6):

$$y = ax + b, \tag{3-6}$$

ali primjena ove formule predstavlja problem prilikom opisivanja vertikalnih pravaca iz razloga što parametar a može postići beskonačne vrijednosti. Iz tog razloga za potrebe Hough-ove transformacije, pravac se definira prema:

$$\rho(x, y) = x\cos\theta + y\sin\theta, \qquad (0 < \theta < \pi) \tag{3-7}$$

što predstavlja pravac u polarnom koordinatnom sustavu gdje se za svaku koordinatu x i y i proizvoljni kut smjera pravca θ može odrediti ρ. Grafički prikaz pravca u polarnom koordinatnom sustavu s pripadnim oznakama dan je na slici 3.5. Za izvršavanje Hough-ove transformacije



Sl. 3.5. Grafički odnos pravca i uređenog para *(ρ, θ)*.

potrebno je osigurati akumulator u obliku matrice čije se dimenzije se mogu izračunati prema:

$$\rho_{max} = \sqrt{{x_{max}}^2 + {y_{max}}^2}$$
$$\theta_{max} = \pi = 180° \rightarrow \theta_{max} = 180 \quad (3\text{-}8)$$

gdje ρ_{max} predstavlja maksimalnu udaljenost od ishodišta do najdaljeg elementa slike, a θ_{max} predstavlja broj vrijednosti kutova koje pravac može poprimiti. Tako kreirana matrica inicijalno je popunjena s nulama, a tijekom izvođenja algoritma vrijednosti matrice se povećavaju za jedan. Za svaki element slike s istaknutim rubovima (ako je vrijednost tog elementa 0) potrebno je prema (3-7) izračunati ρ za svaki od θ kutova te element akumulatora na mjestu (ρ, θ) povećati za 1. Vrijednosti popunjene matrice predstavljaju koliko puta pravac s vrijednostima (ρ, θ) odgovara elementima slike sa istaknutim rubovima. Stoga, na kraju se odabiru elementi akumulatora s najvećim vrijednostima kao rezultantni pravci. Za potrebe ovog projekta pronalaze se dva maksimuma matrice, jedan koji odgovara za lijevu i jedna za desnu voznu liniju.

3.2. Implementacija predloženog rješenja Matlab-u

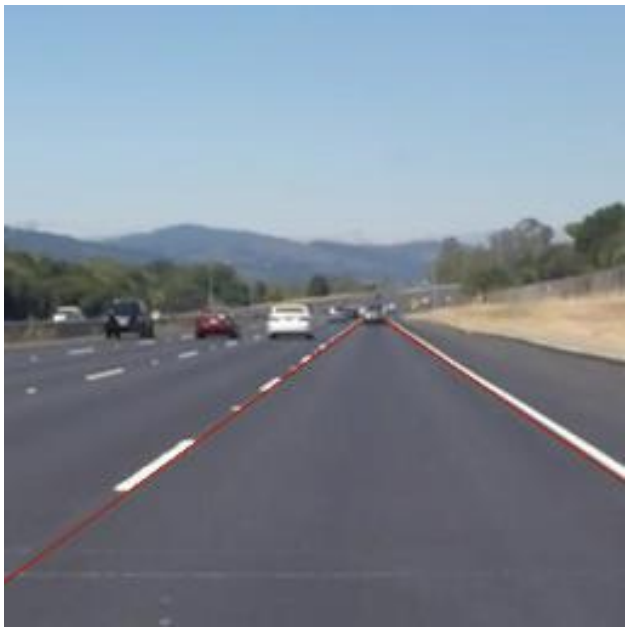
Kako bi se bolje razumjela problematika ovog zadatka te kako bi se izradilo sklopovsko rješenje sa što učinkovitijim dizajnom u pogledu paralelnog izvođenja sustava i brzine izvršavanja, najprije je rješenje ovog problema implementirano koristeći Matlab. Matlab je programski jezik koji se koristi za rješavanje raznih matematičkih i tehničkih problema. Koristeći brojne programske pakete, Matlab pokriva gotovo sva inženjerska područja u što se ubraja obrada slike i obrada digitalnog signala. Korištenjem ugrađenih Matlab funkcija može se postići rješenje ovakvog problema u svega nekoliko linija koda. Ovo rješenje je kasnije korišteno prilikom usporedbe s vlastitim izrađenim u FPGA u svrhu provedbe testova i analize sustava. Na slici 3.6. dana je implementacija algoritma predstavljenog uz korištenje ugrađenih Matlab funkcija. Iako ovaj kod daje iznimno dobre rezultate, njime se ne postiže razumijevanje principa rada korištenih metoda te se ne postiže napredak u razvoju paralelnog izvođenja sklopovskog rješenja. Za potrebe razumijevanja principa rada korištenih metoda izrađeno je vlastito rješenje detekcije voznih linija koristeći Matlab te je taj kod dan u prilogu P.1

Linija ***Kod***

```
1:      I = imread('road.png'); %Učitavanje slike
2:      Ibw = rgb2gray(I); %pretvorba iz RGB u sliku sa sivim tonovima
        BW = edge(Ibw, 'canny'); %izvršavanje predobrade (Gauss-ov filter te
3:      Sobel i Canny operator)
4:      [H, T, R]=hough(BW2); %Hough-ova transformacija
```

Sl. 3.6. Primjer kôda detekcije voznih linija korištenjem ugrađenih Matlab funkcija

Detekcija voznih linija koristeći Matlab započinje učitavanjem RGB (engl. *Red Green Blue* – crvena zelena plava) slike u memoriju sustava te se zatim slika pretvara u sliku sa sivim tonovima (engl. *grayscale*). Koristeći (3-1), slika u sivim tonovima se najprije piksel po piksel, filtrira Gauss-ovim filterom, a zatim se za svaki piksel slike primjenjuje Sobel-ov operator koristeći (3-4) i (3-5) čime se dobiva slika s gradijentima. Na tako obrađenu sliku se primjenjuje Canny-ev operator za naglašavanje rubova slike koristeći se metodom otklanjanja šuma s dvije granice. Pronalaženje pravaca koji se podudaraju s rubovima tako obrađene slike postiže se primjenom Hough-ove transformacije te se odvajaju dva najbolje podudarana pravca koji predstavljaju vozne linije (jedan na lijevoj polovici slike, a drugi na desnoj). Pravci se u konačnici crtaju na slici radi vizualizacije rješenja, a primjer pronađenih voznih linija označenih crvenim pravcima može se vidjeti na slici 3.7.



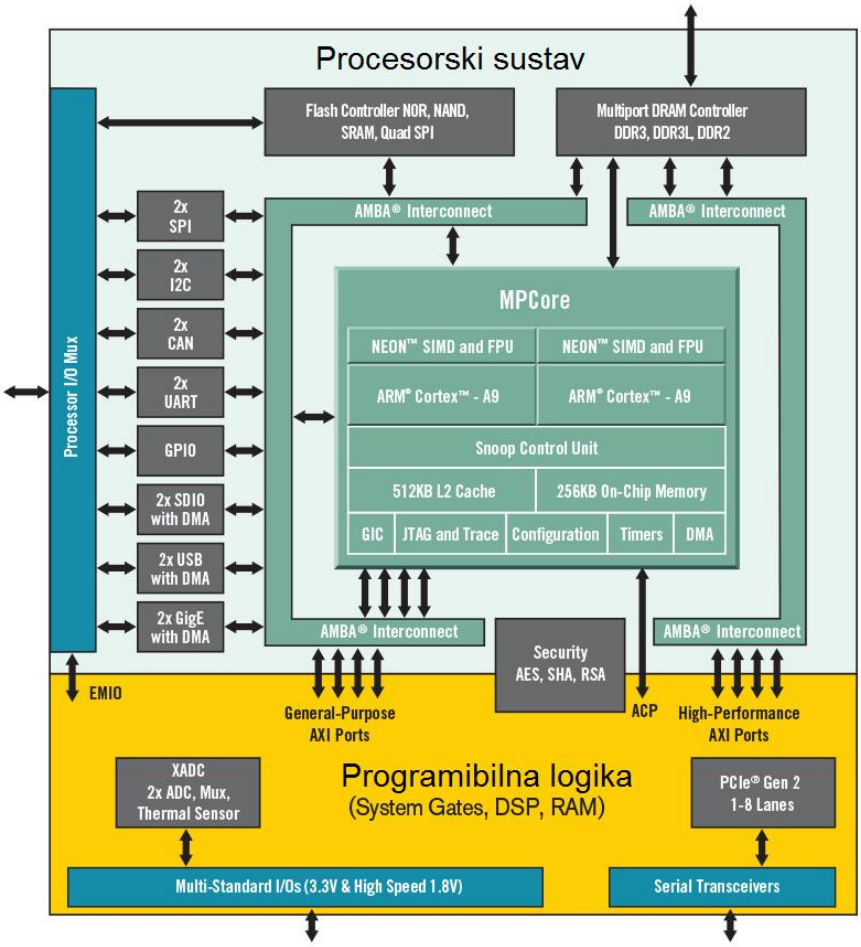
Sl. 3.7. Primjer detekcije voznih linija Matlab implementacijom

3.3. Korištene tehnologije

3.3.1. Zynq-7000

Budući da obrada videa zahtjeva u obradu velike količine podataka u što je moguće kraćem vremenu, procesori opće namjene često ne uspijevaju ispuniti zadatak u zadanom vremenu. U ovakvim situacijama pogodna tehnologija je upravo FPGA.

Za izradu ovog rada korišten je Zynq-7000 SoC koji se nalazi na Zybo razvojnom okruženju. Zynq-7000 je čip čija se arhitektura sastoji od procesorskog sustava i programabilne logike (FPGA) te sučelja koje ih međusobno povezuje. Jezgra procesorskog sustava je dvojezgreni ARM Cortex A9 procesor te se blok dijagram njegove arhitekture može vidjeti na slici 3.8. Uloga procesorskog sustava na ovom SoC-u je procesiranje aplikacija (engl. *Application Processor Unit* - APU), komunikacija s programabilnom logikom, upravljanje memorijom i druge. Programabilna logika je zasnovana prema Artix-7 čipu koja je zajedno sa procesorom spojena u jedan čip, odnosno zajedno čine Zynq-7000 SoC. Sučelje koje povezuje procesorski sustav i programabilnu logiku naziva se ulazno izlazna grupa AXI. Zadaća AXI sučelja je prijenos podataka, a pri tome je nužno postići visoke frekvencije i performanse. Dva su osnovna načina prijenosa podataka unutar AXI sučelja, a to su prijenos na razini bajta i kontinuirani prijenos (engl. *Burst mode*) [10].

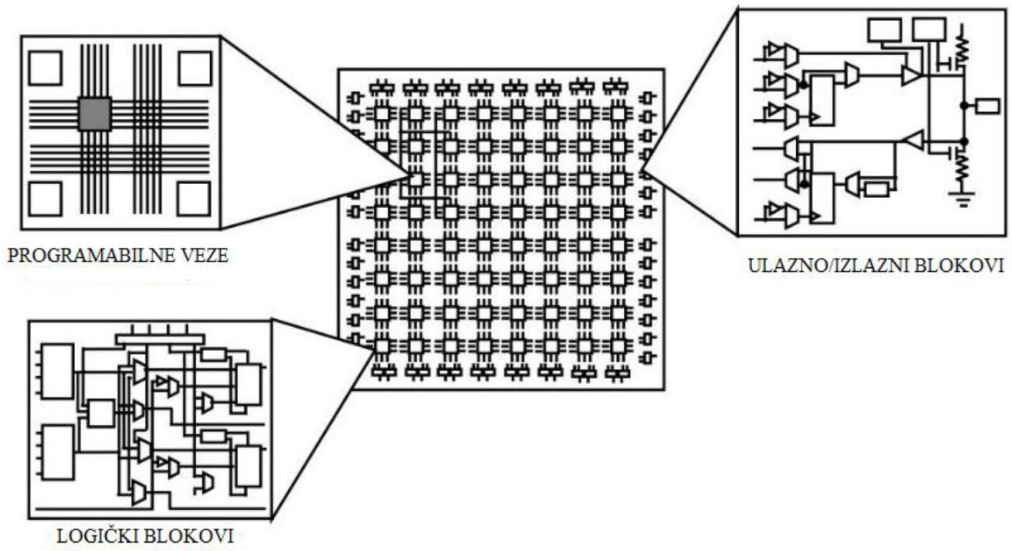


Sl. 3.8. Dijagram Zynq-7000 arhitekture [10]

3.3.2. FPGA

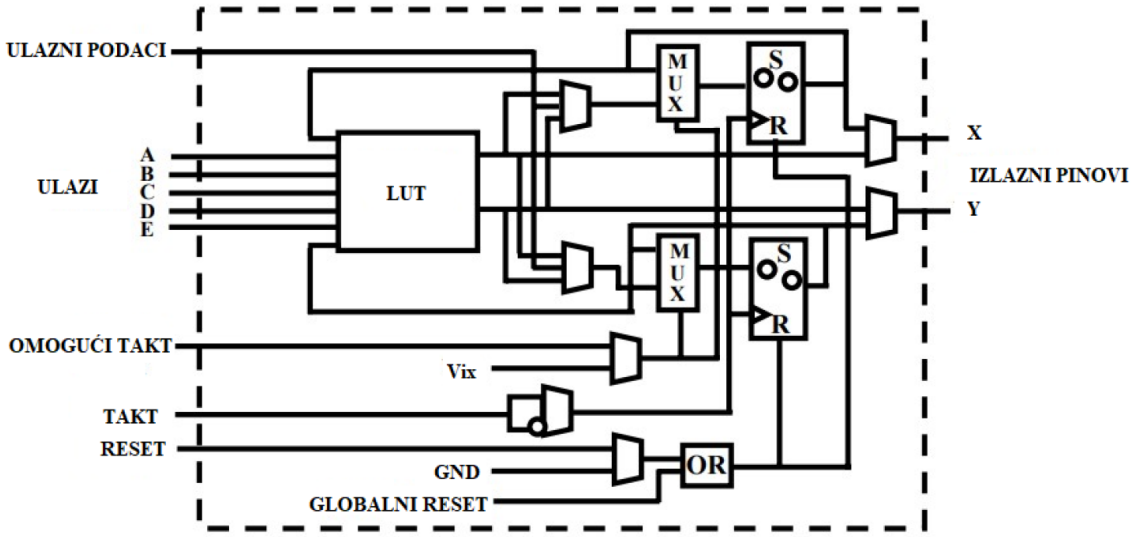
FPGA općenito je integrirani krug koji se sastoji od nekoliko međusobno spojenih programabilnih logičkih blokova (engl. *Configurable Logic Block* - CLB). Svaki navedeni logički blok moguće je zasebno konfigurirati u svrhu obavljanja neke od brojnih logičkih funkcionalnosti. Osim CLB blokova unutar FPGA nalaze se periferni ulazno/izlazni krugovi koji su povezani programabilnim vezama. Struktura internog dijela FPGA prikazana je na slici 3.9. Konfiguriranje FPGA moguće je izvršiti nekim od postojećih jezika za opisivanje sklopovlja poput VHDL-a i Verilog-a. Prilikom konfiguriranja FPGA korisnikov kod se prevodi u fizičko sklopovlje, odnosno spajaju se unutarnji vodovi koji na taj način stvaraju digitalnu sklopovsku logiku.

Logički blokovi unutar FPGA mogu biti implementirani kao sekvencijalni ili kombinacijski logički sklopovi. Također povezivanjem više CLB-a stvara se kompleksna digitalna logika. Unutrašnjost jednog CLB-a prikazana je na slici 3.10. [11]. Sa slike je vidljiv LUT koji se koristi isključivo za odabir konfiguracije za pojedini CLB blok, a primjerice za različite vrijednosti ulaza CLB-a na izlazu će se ostvarivati različite funkcionalnosti. CLB istovremeno može imati



Sl. 3.9. Interna struktura FPGA [10]

samo jednu funkcionalnost, a to može biti primjerice: multiplekser, tranzistorski parovi ili neka od osnovnih kombinacijskih vrata.



Sl. 3.10. Unutrašnjost CLB-a.

Osim mnogobrojnih CLB-a uz FPGA se često nalazi i blokovska memorija s nasumičnim pristupom (engl. *Block Random Access Memory* - BRAM) te nekoliko DSP blokova koje je također moguće konfigurirati prema potrebama projekta. BRAM je sinkrona memorija odnosno upisi i ispisi se izvršavaju sinkrono sa signalom takta. Istovremeno je moguće pristupati BRAM-u sa više različitih sučelja, a proizvoljan je odabir duljine i dubine vektora, odnosno broju bitova „riječi“ jednog podatka i broj takvih podataka. DSP blok korišten uz FPGA sadrži nekoliko DSP mikroprocesora posebne namjene. DSP se razlikuje od procesora opće namjene po tome što sadrži specifičan set instrukcija koji služe za obavljanje ponavljajućih ili često korištenih matematičkih operacija.

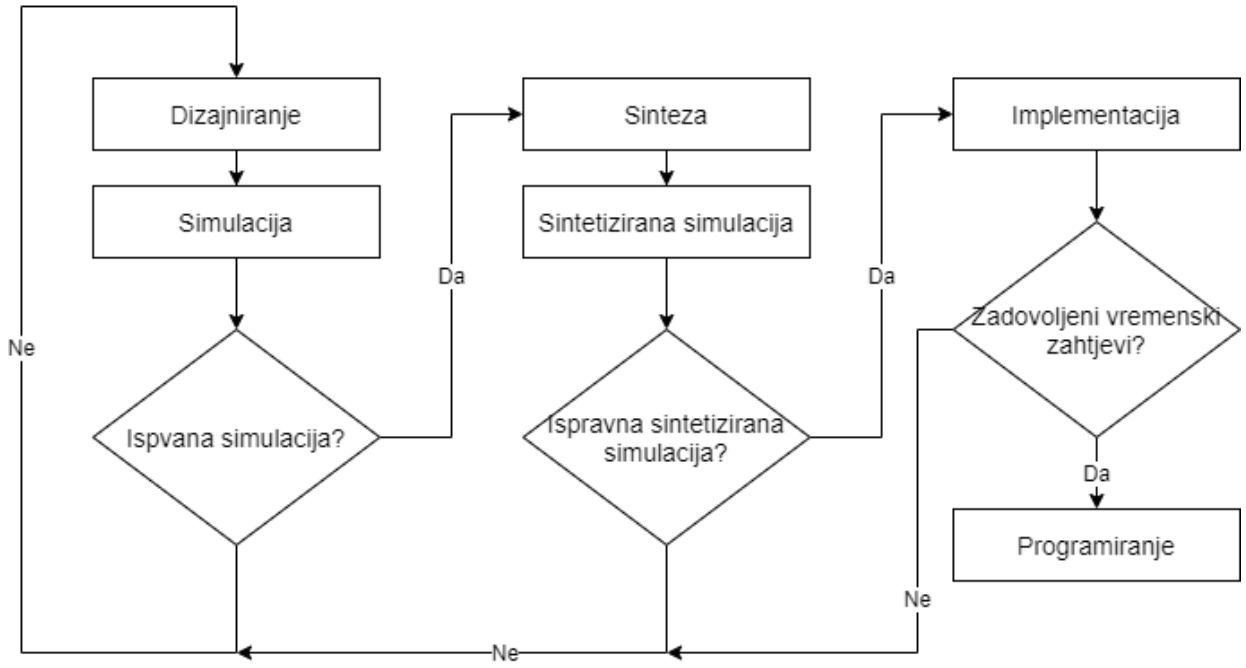
3.3.3. VHDL

U sklopu ovog rada korišten je jezik za opis sklopovlja iznimno brzih integriranih krugova (engl. *Very High Speed Integrated Circuit Hardware Description Language* – VHSIC-HDL, skraćeno VHDL). Osim što se koristi za opisivanje sklopovlja VHDL se može koristiti i za pisanje simulacijskih datoteka za testiranje sklopovlja unutar simulacije [12].

Datoteka koja sadrži VHDL-om strukturirani kod zove se modul te se on sastoji od entiteta i arhitekture. Unutar entiteta navode se svi željeni ulazi i izlazi tog modula, a unutar arhitekture opisuje se funkcionalnost tog modula. Entitet se može zamisliti kao omotač arhitekture koji skriva način na koji se taj modul izvršava, ali dozvoljava njegovo korištenje putem njegovih ulaza i izlaza.

Unutar arhitekture moguće je korištenje drugih već postojećih modula na način da pomoću signala poveže s ulazima i izlazima željenog entiteta. Signal je medij kojim se prenose podaci unutar arhitekture.

Sama VHDL datoteka nije dovoljna za konfiguriranje FPGA sustava, već je potrebno izvršiti sintezu sustava. Dijagram toka izrade dizajna FPGA sustava prikazan je na slici 3.11. Sinteza je izgradnja digitalnog dizajna na osnovu VHDL datoteke. Dizajn se stvara korištenjem programabilnih blokova, DSP-ova LUT-ova i drugih dijelova FPGA sustava. Takav sintetizirani dizajn moguće je simulirati te na taj način ispitati njegovu funkcionalnost, no njegova ispravnost ne jamči da takav dizajn može biti konfiguriran na FPGA sustav. Prije izvršavanja implementacije potrebno je mapirati sve korištene ulaze i izlaze entiteta korištenog modula sa ulazima i izlazima dostupnim na korištenoj arhitekturi. Pokretanjem implementacije ispituje se odgovara li korištena arhitektura željenom dizajnu, odnosno moguće li je dizajn fizički isprogramirati na FPGA čip. Ukoliko je dizajn složen, mogu se pojaviti dodatni problemi kao što su nedovoljna količina CLB-ova, DSP-ova ili LUT-ova, nemogućnost izvršavanja zadatka u zadanoj frekvenciji itd. Ako su svi koraci uspješno izvršeni te su zadovoljeni svi preduvjeti kreira se u konačnici binarna datoteka koja programira opisani dizajn na FPGA sustav.



Sl. 3.11. Dijagram toka izrade FPGA sustava.

3.3.4. Vivado

Kako je navedeni postupak sinteze, simulacije i implementacije kompleksan i zahtjevan postupak korišteno je Vivado grafičko razvojno okruženje koje služi upravo razvoju digitalnih dizajna. Vivado razvojno okruženje sadrži skup automatiziranih operacija koje se koriste za konfiguraciju i generiranje podrške razvojnih okruženja (engl. *Board Support Package* - BSP). Omogućava automatizirani postupak sinteze i implementacije digitalnih dizajna te sadrži i simulator za testiranje kreiranih dizajna. Postupak sinteze koristi sintezu na visokoj razini (engl. *High-level sintesis* - HSL) odnosno programski kod koji prevodi VHDL modul i kreira digitalni dizajn.

3.4. Dizajn rješenja

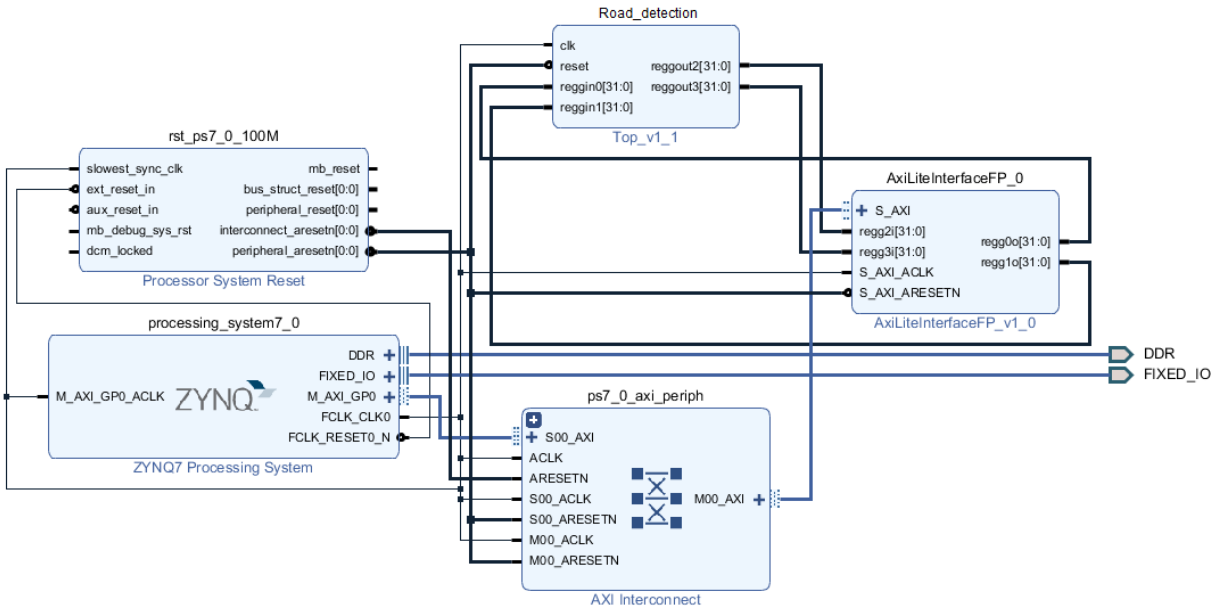
Kako bi se izvršila obrada slike potrebno ju je formatirati iz RGB formata (engl. *Red Green Blue*) u format slike sa sivim tonovima. Radi smanjenja kompleksnosti zadatka, ovaj korak se ne izvodi na sustavu nego se na ulaz dovodi slika prethodno formatirana u sliku sa sivim tonovima. Za razliku od rješenja koristeći Matlab, dizajnirani algoritam za FPGA nema mogućnost učitavanja cijele slike u memoriju sustava (zbog nedovoljne količine memorije na FPGA-u). Stoga se formatirana slika se najprije dijeli u makro blokove. Makro blokovi su veličine 64x64 piksela te se na ulaz sustava redom dovode na obradu sve dok posljednji makro blok nije obrađen. Nakon što se jedan makro blok pohrani u memoriju sustava, započinje dodatna predobrada metodama kao što su Gauss-ovo filtriranje, Sobel-ov operator i Canny-ev operator. Primjenom ovih navedenih metoda dobiva se binarni makro blok koji ima istaknute sve objekte koji se na njemu nalaze. Takav binarni makro blok se šalje na ulaz Hough-ove transformacije koja postupkom glasanja kao rezultat vraća akumulator jednadžbi pravaca. Broj glasova jednog pravca iz tog akumulatora predstavlja broj podudaranja rubova objekta s tim pravcem na trenutno obrađivanom makro bloku. Od svih pronađenih pravaca odvajaju se oni sa najviše osvojenih glasova, odnosno odvaja se jedan pravac s lijeve i jedan s desne polovice slike. Dva odvojena pravca predstavljaju vozne linije s lijeve i desne strane kolnika te donosi se zaključak da se područje koje se nalazi između njih naziva vozna traka po kojoj se vozilo kreće.

Obrada od slike u sivim tonovima pa do pronalaska parametara na temelju kojih se mogu iščitati vozne linije u potpunosti je dizajniran za FPGA sustav, a za prijenos slike u BRAM, upravljanje te iščitavanje rezultata brine se procesorski sustav. Slika 3.12. prikazuje na koji je način preko AXI protokola ostvarena komunikacija između procesorskog sustava i programabilne logike odnosno dizajniranog algoritma za detekciju voznih traka. Jedan od dva ulazna registra je

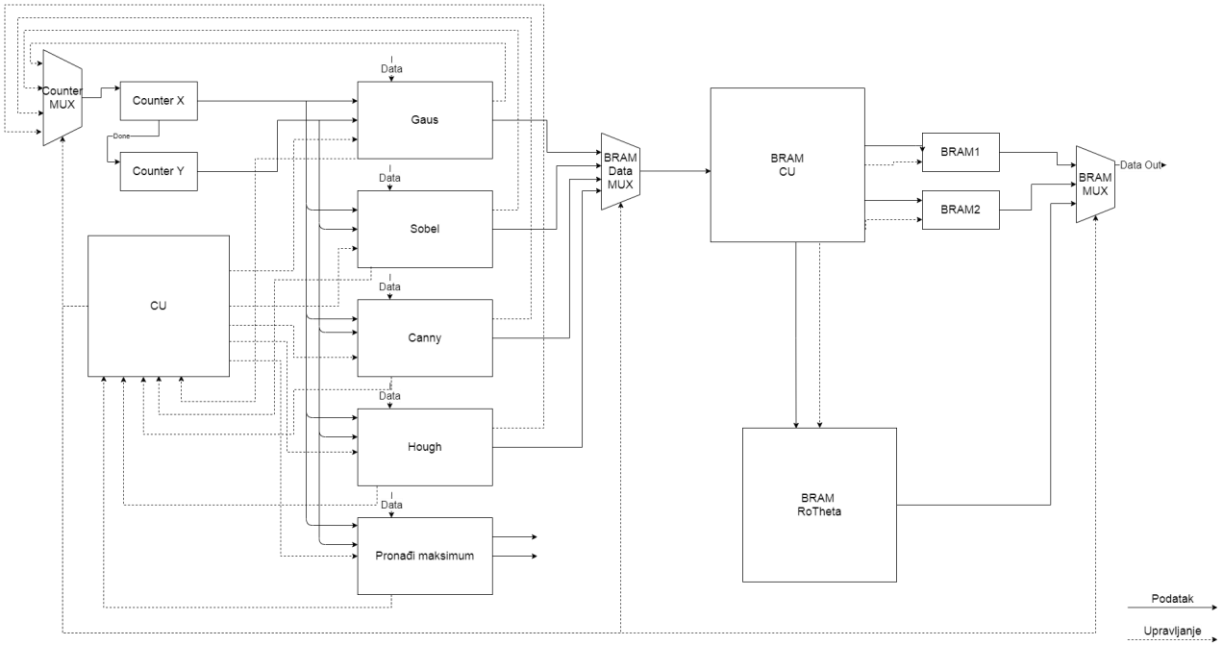
upravljački te ima mogućnost pokretanja, zaustavljanja i ponovnog pokretanja sustava, dok drugi služi kao unos podataka u BRAM. Izlazni registri nakon obrade vraćaju ρ i θ vrijednosti za dva pravca koja se najbolje podudaraju s voznim linijama na slici. Na slici 3.13. prikazan je dijagram rješenja dizajniranog algoritma za detekciju voznih traka. Jedan blok na slici predstavlja jedan dizajnirani VHDL modul. Svi blokovi zajedno čine jednu cjelinu te su povezani u viši modul hijerarhije. Blokove možemo svrstati u nekoliko cjelina:

- Upravljačke jedinice (engl. *Control Unit* - CU) – jedna za upravljanje cijelim sustavom, a duga se brine za upis i ispis podataka u BRAM memoriju
- Izvršne jedinice – prilikom uključivanja izvršavaju određen zadatak. Pod izvršne jedinice spadaju blokovi: Gaus, Sobel, Canny, Hough i pronađi maksimum
- Multiplekseri
- Brojači
- BRAM

U narednim potpoglavljima detaljnije je objašnjena svaka od ovih cjelina.



Sl. 3.12. Procesorski sustav, programibilni sustav te AXI veza između njih.



Sl. 3.13. Procesorski sustav, programibilni sustav te AXI veza između njih.

3.4.1. Upravljačke jedinice

Upravljačka jedinica općenito unutar procesora opće namjene ima izravnu kontrolu nad ostalim dijelovima procesora, odnosno odlučuje kada će se pojedina komponenta tog sustava uključiti/isključiti [13]. Isto tako slijedeći taj primjer, za potrebe ovog projekta dizajnirane su dvije upravljačke jedinice, jedna za upravljanje s upisom i ispisom u BRAM, a druga, za kontrolu cijelog sustava. Obje upravljačke jedinice kreirane su modelom automata sa konačnim brojem stanja (engl. *Finite-State Machine* - FSM) [14].

Svaka izvršna jedinica ovog projekta tokom svog izvođenja ima potrebu pristupiti različitim BRAM-ovima za pohranu i dohvaćanje podataka. Kako ne bi svaka izvršna jedinica zasebno morala brinuti o upisu i ispisu, osmišljena je upravljačka jedinica za BRAM koja se brinula upravo o upisu i ispisu za svaku od izvršnih jedinica. Na ovaj način smanjilo se pisanje redundantnog koda te su se kodovi izvršnih jedinica uvelike pojednostavljeni.

Upravljačka jedinica na razini cijelog sustava se brine o pravilnom uključivanju i isključivanju pojedinih blokova kako bi nastao koordinirani rad sustava. Određivanje kada je potrebno uključiti/isključiti pojedini blok definirano je završnim signalima koje je svaka od izvršnih jedinica zasebno šalje te na taj način signalizira da je izvršen traženi zadatak. Redoslijed izvršavanja blokova je unaprijed definiran. Najprije se izvršava Gauss-ov blok, pa zatim Sobel-ov i Canny-jev za naglašavanje rubova te na kraju blok Hough-ove transformacije za glasanje. Blok

za pronalazak i vraćanje vrijednosti odnosno pronalaska dvije maksimalne vrijednosti iz ρ i θ matrice direktno je upravljan s procesorskim sustavom te se po potrebi može uključiti.

3.4.2. Izvršne jedinice

Izvršne jedinice, kao što sama riječ sugerira izvršavaju određeni zadatak. Ukupno je u sustavu 5 blokova izvršnih jedinica: Gauss, Sobel, Canny, Hough i „pronađi maksimum“. Svaka od ovih jedinica ima signal uključivanja kojim se započinje njezino izvođenje te završni signal kojim se signalizira da je zadatak uspješno izvršen. Također na ulazu svakog izvršnog bloka nalazi se brojač s kojim se određuje adresa za pristup BRAM-u, a svaki blok zasebno se brine o inkrementiranju tog brojača.

Blokovi Gauss, Sobel i Canny imaju sličan zadatak:

1. dohvati podatak ili podatke
2. izračunaj ili donesi zaključak na temelju dohvaćenih podataka
3. spremi podatak.

Koliko podataka te iz kojeg BRAM-a je potrebno dohvatiti odnosno spremiti podatke brine se svaki blok zasebno unutar sebe.

Gauss blok najprije dohvaća podatak te prema (3-1) računa izlaznu vrijednost i sprema ju nazad u BRAM. Sobel blok najprije dohvaća trenutno obrađivani element te nakon njega dohvaća 8 susjednih elemenata kako bi se kreirala 3x3 matrica. Zatim se kreirana matrica množi s matricama (3-3), koristeći (3-4) te se rezultat sprema na odgovarajuće mjesto nazad u BRAM. Ovim postupkom u BRAM se sprema makro blok s izračunatim gradijentima slike.

Canny operator prema 3.1.3. se sastoji od dva dijela: stanjivanja rubova te postupka otklanjanja šumova s dvije granice (engl. *Double Threshold*). U ovom radu se primjenjuje samo postupak otklanjanja šumova s dvije granice iz razloga što stanjivanje rubova zahtjeva velik broj uvjetnih grananja, a to nije prikladno za FPGA sustav. Izostavljanje postupka stanjivanja rubova neće uvelike utjecati na konačan rezultat iz razloga što ovaj postupak smanjuje količinu piksela koji omeđuju objekt sa slike. Canny blok najprije dohvaća podatak te na temelju vrijednosti podatka i dviju granica kod metode za otklanjanje šumova zaključuje se hoće li podatak ostati u BRAM-u ili ga se izjednačava s 0 odnosno briše iz BRAM-a.

Blok Hough u kojem se izvršavalo glasanje na temelju Hough-ove transformacije ima nešto složeniji algoritam. Najprije je potrebno dohvatiti iz BRAM-a vrijednost elementa sa slike te zatim ako je dohvaćena vrijednost veća od 0 potrebno je inkrementirati vrijednost (ρ, θ) matrice. Kako

bi se uopće postiglo inkrementiranje potrebno je izračunati prema (3-7) vrijednost ρ za svaki kut θ te se prema izračunatim (ρ, θ) na odgovarajućim mjestima u memoriji vrijednost povećava za 1. Taj postupak je potrebno ponavljati za sve piksele slike s istaknutim rubovima koji imaju vrijednost 0. Kao što je rečeno Hough-ova transformacija je poprilično memorijski zahtjevan proces, pogotovo ako se radi o slikama visoke rezolucije. Kako bi smanjili memorijske potrebe u ovom projektu se za detekciju voznih linija, nisu se koristile sve vrijednosti θ. Slike koje dolaze na obradu ovog sustava dolaze s prednje kamere automobila, pa je bilo moguće zaključiti da će se vozne linije koje je potrebno pronaći nalaziti između 20° i 70° za desnu polovicu slike te 110° i 160° za lijevu polovicu slike. S ovime je postignuta 80 puta manja memorijska zahtjevnost.

Posljednji izvršni blok za pronalazak maksimalne vrijednosti je jednostavno na poziv procesorskog sustava prošao kroz sve vrijednost (ρ, θ) matrice te pronašao dvije najveće vrijednosti. Jedna vrijednost je odgovarala za kutove između 20° i 70°, a druga vrijednost za kutove između 110° i 160°. Vrijednosti ρ i θ se šalju natrag na procesorski sustav radi iščitavanja te usporedbu s nekim od postojećih rješenja.

3.4.3. Ostali blokovi

Multiplekser je kombinacijski logički sklop koji se sastoji od podatkovnog ulaza, upravljačkog ulaza te podatkovnog izlaza. S obzirom na upravljački ulaz odabire se jedan podatak s podatkovnog ulaza te ga prikazuje na izlazu [15]. U ovom radu korišteno je nekoliko multipleksera iz razloga što više izvršnih jedinica treba pristupiti nekom drugom bloku, a to je fizički nemoguće ostvariti.

Za potrebe ovog projekta također je izrađen je brojač sa proizvoljnom duljinom brojanja, mogućnosti ponovnog pokretanja te pauziranja. Brojač je u ovom projektu najčešće služi kao spremnik adrese za pristup BRAM-u. Osim navedenih ulaza i izlaza postoji i dodatan izlaz koji indicira zadnju vrijednost brojanja prije ponovnog pokretanja. Ovaj izlaz omogućava spajanje više brojača u jedno što se u projektu koristi za pamćenje x i y vrijednosti trenutnog elementa slike.

Kako je kompleksnost ovog zadatka iznimno velika bilo je potrebno negdje pohranjivati sve međurezultate. Međurezultati su pohranjeni u BRAM te od fizički dostupno 256 kB BRAM-a na SoC-u iskorišteno je 145 kB. Četiri BRAM-a veličine 4096 B bila su potrebna za pohranu obrađivanog dijela slike kod izvršnih jedinica. Iako se činilo da su četiri BRAM-a potrebna kako bi se pokrili zahtjevi svake od izvršnih jedinica, ipak se iskoristila činjenica da se neke od vrijednosti nakon obrade mogu prepisati novima. Na taj način se svelo na dva BRAM-a od 4096 B te dok bi se jedan punio, drugi bi se istovremeno praznio, a nakon toga je došlo do izmjene u

redoslijedu korištenja. Ostatak memorije pridodan je *(ρ, θ)* matrici te se izračunom došlo da je maksimalna veličina obradive slike veličine 512x512 piksela.

Osnovne biblioteke VHDL-a podržavaju samo osnovne matematičke operacije kao što su plus, minus puta i podijeljeno. U matematičkim formulama prema (3-4), (3-5) i (3-7) korištene su i neke od operacija koje nisu dostupne u osnovnim bibliotekama, pa se u takvim slučajevima, po potrebi koristi algoritam za koordinirano rotacijsko digitalno računanje (engl. *Coordinate Rotational Digital Computer* - CORDIC) [16].

3.5. Implementacija sustava

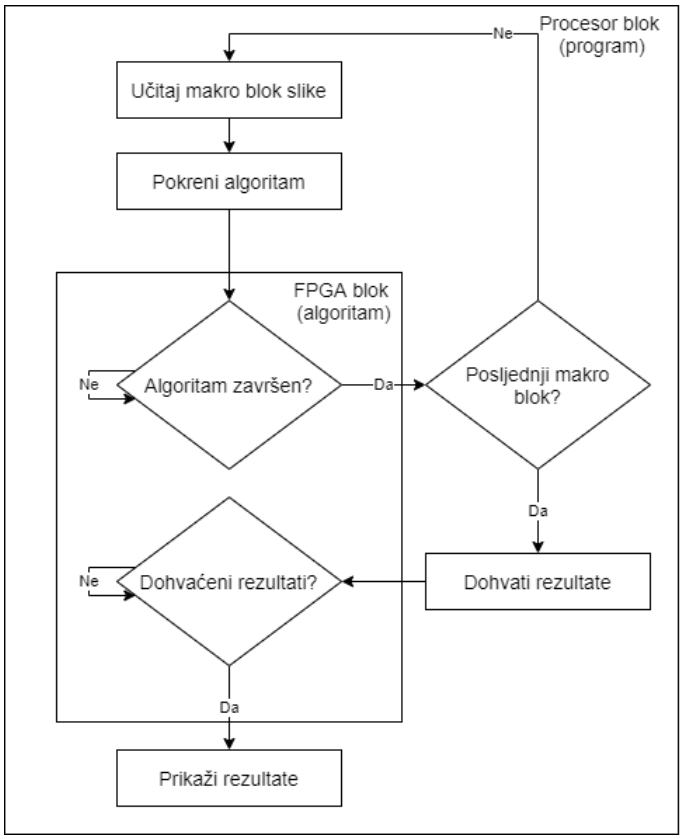
Nakon što je dizajn dovršen, sustav se sintetizira pomoću automatske funkcionalnosti za sintezu unutar Vivado razvojnog okruženja. Rezultat sintetiziranja sustava VHDL koda vraća dizajn načinjen od logičkih sklopova koje je fizički moguće implementirati. Takav dizajn se zatim unutar Vivada pokreće na automatsku implementaciju sustava koja sintetizirani dizajn pokušava smjestiti na željeno sklopovlje odnosno u ovom slučaju Zynq-7000. Postignuta brzina izvršavanja algoritma na Zynq-7000 iznosi 8 MHz. Rezultat implementacije prikazan je tablicom 3.1. Tablica prikazuje koliko je resursa zauzeto na sklopovlju. Iz navedene tablice možemo vidjeti kako BRAM zauzima oko 50% no povećavanjem rezolucije potreba za BRAM-om eksponencijalno raste te je iz tog razloga maksimalna veličina slike ograničena na rezoluciju 512x512. Osim podataka navedenih u tablici iz implementacije možemo iščitati informacije o temperaturi spojeva, snazi SoC-a itd. Ako je implementacija uspješno izvršena te su zadovoljeni svi preduvjeti željenog sklopovlja (Zynq-7000) moguće je generirati kod za programiranje sklopovlja.

Tablica 3.1. Zauzeće resursa sklopovlja dizajniranog sustava.

Resursi	*Dostupno*	*Iskorišteno*	*Iskorišteno [%]*
LUT	17600	3698	21.0
LUTRAM	6000	62	1.03
BRAM	60	34	55.67
DSP	80	9	3.13

Pokretanje sustava izvršava se na razvojnoj programskoj okolini (engl. *Software development kit* - SDK) sa prethodno dodanim kodom za programiranje sklopovlja. Izrađen je program u kojem je najprije potrebno pomoću odlagališta memorije (engl. *memory dump*) potrebno učitati sliku u RAM Zybo razvojnog sustava. Sliku je potrebno učitati u „.bin“ formatu te nakon učitavanja pokrenuti program. Program automatski šalje dijelove slike blok po blok na sustav te ga nakon svakog poslanog bloka pokreće kako bi se izvršila obrada. Blokovi se na sustavu spremaju u BRAM te se iz BRAM-a koristeći algoritam obrađuje blok. Nakon obrađenog bloka sustav vraća

signal kako je završio s obradom te čeka na ulazu novi blok. Program šalje idući blok te se ovaj niz operacija ponavlja sve dok se svi blokovi ne obrade. Kada sustav pošalje signal da je uspješno obradio posljednji blok, program novom naredbom zatraži od sustava rezultate obrade. Rezultati pristignu u obliku uređenog para (ρ, θ) te je pomoću toga prema (3-6) moguće kreirati pravac. Pravac se zatim na osobnom računalu prikazuje na obrađivanoj slici radi vizualne procjene točnosti rezultata. Na slici 3.14. prikazan je dijagram toka programa za učitavanje slika odnosno blokova u BRAM i pokretanje algoritma.



Sl. 3.14. Dijagram toka programa za učitavanje slike i pokretanje algoritma.

4. VERIFIKACIJA IMPLEMENTIRANOG RJEŠENJA NA FPGA

Nakon uspješne implementacije sustava za detekciju voznih linija, koja je detaljno opisana u prethodnom poglavlju, na sustavu su provedena mjerenja s obzirom na efikasnost detekcija linija na kolniku te s obzirom na brzinu izvođenja predloženog algoritma. Sukladno tome izvršene su usporedbe s implementiranim rješenjem koristeći ugrađene Matlab funkcije. Rezolucije slika na kojima su se provodila mjerenja iznosile su 64x64, 256x256 i 512x256, 512x512. Testni uzorci sastojali su se od slika cesta sa voznim linijama, a neke od slika su sadržavale dodatne smetnje poput vozila, oštećenog kolnika itd. Sustav je testiran nad skupom podataka od 100 slika za svaku od spomenutih rezolucija. Stavke koje su bile testirane odnosile su se na brzinu izvršavanja algoritma, brzinu obrade slike s učitavanjem u BRAM te usporedbu rezultata detektiranih voznih linija s obzirom na gotove funkcije Matlab razvojnog okruženja radi utvrđivanja točnosti algoritma. Za procjenu efikasnosti detekcije linija predloženim algoritmom korištene su sljedeće tvrdnje: točno detektirana linija (engl. *true positive* – TP), detektirana linija koja ne postoji (engl. *false positive* – FP) te nije detektirana postojeća linija (engl. *false negative* – FN). Iz ovoga slijede metrike (4-1):

$$preciznost = \frac{TP}{TP + FP}\ [\%]$$
$$odziv = \frac{TP}{TP + FN}\ [\%] \tag{4-1}$$

Budući da za testne slike nisu postojale označene linije (engl. *ground truth*), za svaku sliku je napravljena subjektivna procjena TP, FP i FN.

4.1. Mjerenje efikasnosti detekcije voznih linija

U nastavku su ilustrirani rezultati koji se postižu primjenom implementiranog algoritma za detekciju voznih linija na FPGA i rezultati koji se postižu na temelju implementacije u Matlab-u, na nekoliko testnih slika različite rezolucije. U slijedećim testovima na slikama crveni pravci predstavljaju pronađene vozne linije koristeći Matlab-a, dok plavi pravci označavaju pronađene vozne linije predloženim algoritmom implementiranim na FPGA.

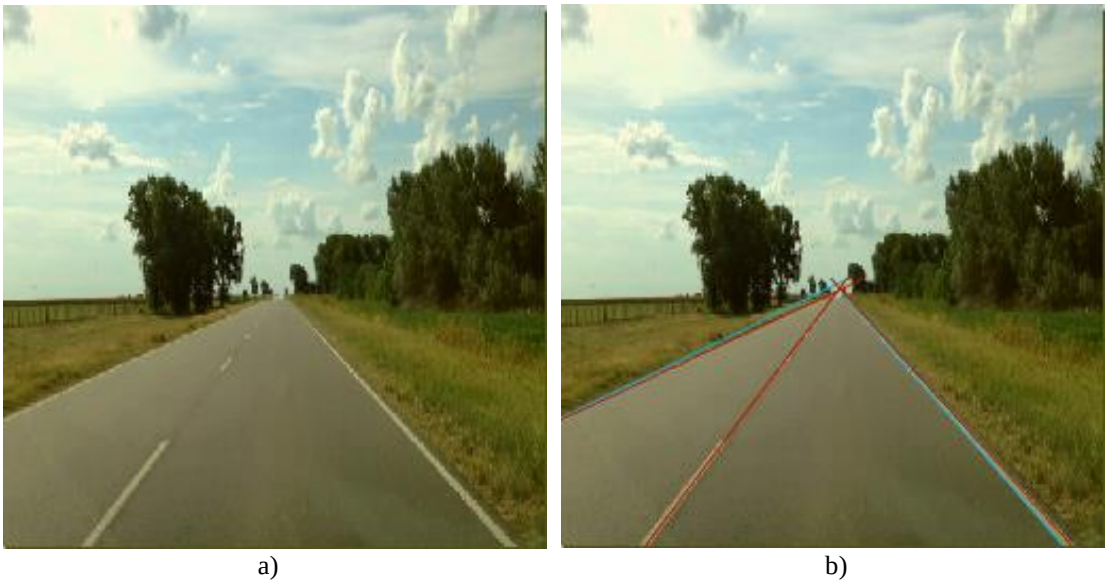
Prvi test obavljen je na slici koja prikazuje cestu sa vidljivim voznim linijama. Slika je veličine jednog makro bloka koji se dovodi na ulaz sustava (64x64) te se rezultat algoritma može vidjeti na slici 4.1. Možemo vidjeti kako algoritam i pri ovako malim rezolucijama može prepoznati voznu liniju sa slike. Male razlike u odstupanju algoritma implementiranog na FPGA sustavu i algoritma korištenog u Matlab-u nastaju zbog kvantizacije odnosno prilikom zaokruživanja

brojeva na cjelobrojne vrijednosti. Ovaj primjer je dan kao reprezentativni primjer makro bloka s kojim algoritam radi. Dalje u nastavku, izvedeni su testovi na slikama većih rezolucija na kojima se jasnije vidi razlika između algoritma implementiranog na FPGA i Matlab implementacije.

a) b)

Sl. 4.1. a) ulazna slika 64x64 b) slika sa označenim pronađenim voznim linijama.

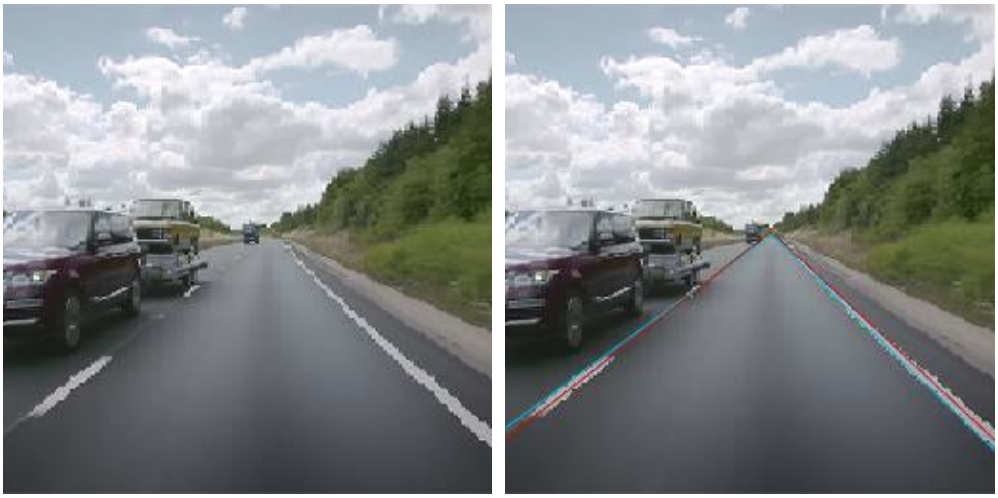
Drugi test izvršen je na slici rezolucije 256x256 piksela. Slika se sastoji od ceste s vidljivim voznim linijama bez ikakvih smetnji. Rezultat obje implementacije može se vidjeti na slici 4.2. Algoritam uspješno detektira vozne linije te se može vidjeti kako je odstupanje od postojećih rješenja zanemarivo. Potrebno je napomenuti kako je implementacija ovog rada pronašla jednu voznu liniju manje.



a) b)

Sl. 4.2. a) ulazna slika 256x256 b) slika sa označenim pronađenim voznim linijama.

Treći test izvršen je na slici iste rezolucije kao i slika iz prethodnog testa, ali osim voznih linija na slici se nalaze smetnje. Smetnje su predstavljene kao vozila koja se kreću u suprotnoj voznoj traci kolnika. Rezultat algoritma može se vidjeti na slici 4.3. U ovom testu odstupanje od postojećeg rješenja je nešto veće, ali i dalje zanemarivo.



a) b)

Sl. 4.3. a) ulazna slika 256x256 b) slika sa označenim pronađenim voznim linijama.

Četvrti test izvršen je na slici rezolucije 512x256 piksela. Slika se sastoji od ceste s vidljivim voznim linijama bez ikakvih smetnji. Rezultat implementacije može se vidjeti na slici 4.4. Implementacija uspješno detektira dvije od tri postojeće vozne linije u odnosu na Matlab implementaciju koja detektira sve tri.

Posljednji peti test izvršen je na slici rezolucije 512x256 piksela. Slika se sastoji od ceste s vidljivim voznim linijama tijekom prometne gužve. Rezultat algoritma može se vidjeti na slici 4.5. Implementacija na ovoj slici nije uspjela ispravno detektirati vozne linije za razliku od implementacije na Matlab-u, a glavni razlog tome je pogreška kvantizacije i razlike Canny operatora objašnjene u potpoglavlju 3.4.2. Također mogući razlog je puno veći broj smetnji na slici što rezultira detekciju drugih objekata prisutnih na slici.

a)

b)

Sl. 4.4. a) ulazna slika 512x256 b) slika sa označenim pronađenim voznim linijama.

a)

b)

Sl. 4.5. a) ulazna slika 512x256 b) slika sa označenim pronađenim voznim linijama.

Nad istim skupom podataka ispitana je i efikasnost detekcije linija pomoću metrika (4-1). Tablicom 4.1. prikazani su dobiveni rezultati s obzirom na rezoluciju slike. Iz tablice se može vidjeti kako povećanjem rezolucije rastu preciznost i odziv. Treba naglasiti kako veliki utjecaj na pogreške doprinosi to što je unaprijed predefinirano da algoritam izbaci dva maksimalna rješenja bez obzira na broj prisutnih linija na slici. Efikasnost algoritma na svim rezolucijama prikazana je tablicom 4.2. Također proveden je test predloženog algoritma s obzirom na detekciju vozne trake odnosno dvije vozne linije koje čine voznu traku kojom se vozilo kreće. Tablicom 4.3. prikazani postignuti rezultati algoritma s obzirom na detekciju vozne trake. Ovom tablicom prikazan je pronalazak vozne trake što je ujedno i zadatak ovog rada. Usporedbom tablica 4.2. i 4.3. može se zaključiti kako se u slučaju detekcije odziv sustava značajno povećao te možemo zaključiti kako predložena implementacija na FPGA sustavu prilično dobro otkriva obje linije odnosno voznu traku.

Tablica 4.1. Efikasnost detekcije linije FPGA implementacije

Rezolucija	*TP*	*FP*	*FN*	*Preciznost [%]*	*Odziv [%]*
64x64	164	36	40	82	80.39
256x256	173	27	31	86.5	84.8
512x256	175	25	29	87.5	85.78
522x512	183	17	21	91.5	89.71

Tablica 4.2. Efikasnost algoritma s obzirom na sve prisutne vozne linije

	TP	*FP*	*FN*	*Preciznost [%]*	*Odziv [%]*
Detekcija u slučaju proizvoljnog broja voznih linija	695	105	121	86.88	85.17

Tablica 4.3. Efikasnost algoritma s obzirom na detekciju vozne trake

	TP	*FP*	*FN*	*Preciznost [%]*	*Odziv [%]*
Detekcija u slučaju prisutne točno dvije vozne linije (vozna traka)	695	105	70	86.88	90.84

4.2. Mjerenje performansi sustava

Osim testova točnosti, izvršeni su testovi brzine izvršavanja algoritma te brzine izvršavanja sustava (izvršavanje algoritma uz uračunato vrijeme učitavanja slike u BRAM). Tablica 4.4. prikazuje ovisnost vremena izvršavanja algoritma detekcije linija na FPGA s obzirom na rezoluciju. Iz tablice se može vidjeti kako povećanjem rezolucije raste i vrijeme izvođenja

algoritma, a razlog tome je što povećanjem rezolucije povećava se matrica Hough-ove transformacije pa samim time i broj izvršavanja operacija spremanja vrijednosti u matricu i pretraživanja vrijednosti iz nje. Prosječno vrijeme izvršavanja algoritma uz maksimalnu rezoluciju iznosi 3,3 sekunde uz standardnu devijaciju od 32,4 milisekunde. Ovo vrijeme izvršavanja algoritma ne može zadovoljiti detekciju voznih linija u stvarnom vremenu iz razloga što čak i pri malim brzinama od npr. 5 m/s, algoritam bi detekciju izvršio nakon 15 prijeđenih metara, a za to vrijeme vozna traka se može značajno promijeniti. Pri najmanjoj rezoluciji (64x64) algoritam može postići brzinu od 19 FPS te bi se takav sustav mogao primijeniti u stvarnom vremenu, ali zbog niske rezolucije vjerodostojnost rezultata bi bila upitna.

Tablica 4.4. Vrijeme izvođenja algoritma

	Rezolucija			
	64x64	*256x256*	*512x256*	*512x512*
Prosječno vrijeme [ms]	51	831	1606	3303,5
Standardna devijacija [ms]	0.99	4.93	8.14	17.38
Najduže vrijeme [ms]	52	841	1621	3341

Izvršavanje algoritma nije jedino vrijeme koje utječe na brzinu sustava, dodatan utjecaj ima učitavanje makro blokova u BRAM. Tablicom 4.5. prikazana je brzina izvođenja detekcije voznih linija s uračunatim vremenom učitavanja makro blokova u BRAM. Usporedbom ove tablice s tablicom 4.4. možemo zaključiti kako učitavanje slike u BRAM nije zanemarivo, ali isto tako nije ni najveći razlog zbog kojeg implementirani sustav ima ovakva vremena izvođenja. Unaprjeđenje sustava primarno se treba odnositi na raspoređivanje koraka algoritma, a mogući su i napredci u dodatnoj paralelizaciji sustava i preciznijem određivanju ROI.

Tablica 4.5. Vrijeme izvođenja algoritma s učitavanjem u BRAM

	Rezolucija			
	64x64	*256x256*	*512x256*	*512x512*
Prosječno vrijeme [ms]	82.5	911.5	1809	3634
Standardna devijacija [ms]	2.26	7.21	12.22	21.32
Najduže vrijeme [ms]	86	926	1830	3668

5. ZAKLJUČAK

Detekcija voznih traka, odnosno voznih linija na kolniku važan je segment u automobilskoj industriji čijim se rješavanjem približavamo ostvarivanju autonomnosti vozila. Ispravna detekcija voznih linija pridonosi detektiranju vozne trake kojom se vozilo kreće te detekciji voznih traka ostalih vozila koja sudjeluju u prometu. Ovim sustavom omogućava se detekcija prelaska iz jedne vozne trake u drugu, a samim time obavještavaju se i drugi sustavi automobila koji mogu pravovremeno reagirati u slučaju opasnosti i izbjeći potencijalnu nesreću.

U okviru rada uspješno je na temelju postojećih radova predloženo vlastito rješenje te izrađena implementacija u Matlab-u. Prema Matlab implementaciji dizajniran je i implementiran sustav detekcije vozne trake na Zybo razvojnom okruženju. Sustav ima mogućnost detektiranja vozne trake sa slika rezolucije od 64x64 do 512x512 piksela. Vrijeme potrebno sustavu da se vozna linija detektira pri maksimalnoj rezoluciji iznosi 3.6 sekundi, a samo izvođenje algoritma iznosi 3.3 sekundi. Izvođenje algoritma pri stvarnom vremenu moguće je postići samo pri rezoluciji od 64x64 piksela gdje postiže 19 FPS-a. Analizom rezultata provedenih tesnim slikama ustanovilo se da sustav detektira voznu traku s preciznošću od 86.88 % i odzivom od 90.84%.

Rezultat točnosti sustava upućuje na prilično dobre performanse sustava, ali moguća su poboljšanja u vidu vremena izvršavanja algoritma. Jedno od mogućih rješenja bila bi optimizacija sustava te poboljšanja u odabiru ROI. Osim brzine, moguća su poboljšanja i na povećanju rezolucije obradivih slika, to bi se također moglo poboljšati usavršavanjem odabira ROI ili pak implementiranjem dizajna na sustav s većom količinom BRAM-a.

LITERATURA

[1] „Uzdužne oznake“, *Prometna Signalizacija*. https://www.prometna-signalizacija.com/horizontalna-signalizacija/uzduzne-oznake/ (pristupljeno ruj. 23, 2020).

[2] „Lane departure warning system“, *Wikipedia*. ruj. 21, 2020, Pristupljeno: ruj. 23, 2020. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Lane_departure_warning_system&oldid=979630276.

[3] „A Real-Time System for Lane Detection Based on FPGA and DSP | SpringerLink“. https://link.springer.com/article/10.1007/s11220-016-0133-8 (pristupljeno ruj. 23, 2020).

[4] S. Malmir i M. Shalchian, „Design and FPGA implementation of dual-stage lane detection, based on Hough transform and localized stripe features“, *Microprocess. Microsyst.*, sv. 64, str. 12–22, velj. 2019, doi: 10.1016/j.micpro.2018.10.003.

[5] „FPGA based real-time lane detection and tracking implementation - IEEE Conference Publication“. https://ieeexplore.ieee.org/abstract/document/7519587 (pristupljeno ruj. 23, 2020).

[6] I. El Hajjouji, S. Mars, Z. Asrih, i A. El Mourabit, „A novel FPGA implementation of Hough Transform for straight lane detection“, *Eng. Sci. Technol. Int. J.*, sv. 23, izd. 2, str. 274–280, tra. 2020, doi: 10.1016/j.jestch.2019.05.008.

[7] S. P. Narote, P. N. Bhujbal, A. S. Narote, i D. M. Dhane, „A review of recent advances in lane detection and departure warning system“, *Pattern Recognit.*, sv. 73, str. 216–234, sij. 2018, doi: 10.1016/j.patcog.2017.08.014.

[8] M. Forko, „NEFOTOREALISTIČNA TEHNIKA PRIKAZA VIDEO ZAPISA“, str. 41, lipanj 2009.

[9] „Canny edge detector“, *Wikipedia*. srp. 22, 2020, Pristupljeno: ruj. 15, 2020. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Canny_edge_detector&oldid=968976334.

[10] „Zynq-7000 SoC“, *Xilinx*. https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html (pristupljeno ruj. 23, 2020).

[11] I. Valek, „Implementacija JPEG kodera zasnovanog na FPGA“, master’s thesis, Josip Juraj Strossmayer University of Osijek. Faculty of Electrical Engineering, Computer Science and Information Technology Osijek. Department of Communications. Chair of Multimedia Systems and Digital Television., 2018.

[12] „vhdl.pdf“. Pristupljeno: ruj. 23, 2020. [Na internetu]. Dostupno na: http://www.cs.ucr.edu/~ehwang/courses/cs120b/vhdl.pdf.

[13] „Upravljačka jedinica“, *Wikipedija*. velj. 17, 2020, Pristupljeno: ruj. 23, 2020. [Na internetu]. Dostupno na: https://hr.wikipedia.org/w/index.php?title=Upravlja%C4%8Dka_jedinica&oldid=5459572.

[14] „Finite-state machine“, *Wikipedia*. kol. 11, 2020, Pristupljeno: ruj. 15, 2020. [Na internetu]. Dostupno na: https://en.wikipedia.org/w/index.php?title=Finite-state_machine&oldid=972361694.

[15] „Multipleksori“. https://www.fpz.unizg.hr/hgold/ES/DE/multipleksori.htm (pristupljeno ruj. 14, 2020).

[16] „pg105-cordic.pdf“. Pristupljeno: ruj. 15, 2020. [Na internetu]. Dostupno na: https://www.xilinx.com/support/documentation/ip_documentation/cordic/v6_0/pg105-cordic.pdf.

SAŽETAK

U radu je opisana izrada i implementacija detekcije voznih traka uz korištenje FPGA tehnologije. Najprije dan pregled nekoliko postojećih rješenja detekcije vozne trake. Spomenuti radovi navode metode korištene za izradu sličnih sustava te FPGA kao prikladan sustav za rješavanje ovakve vrste problema. Nakon toga je u radu dan prijedlog algoritma za detekciju voznih traka te je isti implementiran u Matlab-u. Prema implementaciji iz Matlab-a dizajnirano je sklopovsko rješenje za detekciju vozne trake koristeći VHDL. Dizajn je sintetiziran i implementiran na Zynq-7000 SoC-u. Na procesorskom sustavu se izvršava upravljanje, unos slike i prikazivanje rješenja, dok se na programabilnom dijelu izvršava dizajnirani algoritam. Izvršena je verifikacija implementiranog dizajna te su na sustavu provedena mjerenja s obzirom na efikasnost detekcija linija na kolniku te s obzirom na brzinu izvođenja predloženog algoritma.

Ključne riječi: FPGA, VHDL, detekcija vozne trake, Zynq.

ABSTRACT

The paper describes the development and implementation of lane detection using FPGA technology. First, an overview of several existing lane detection solutions. The mentioned papers list the methods used for makeing similar or the same FPGA systems as a suitable system for solving these types of problems. After that, the paper presents a proposal for an algorithm for detecting lanes, which is implemented in Matlab. According to the implementation from Matlab, a hardware solution for detecting driving tracks using VHDL was designed. The design was synthesized and implemented on a Zynq-7000 SoC. On the processor system, control, image input and solution display are performed, while on the programmable part, the designed algorithm is executed. Verification of the implemented design was performed and measurements were performed on the system with regard to the efficiency of line detection on the pavement and with regard to the speed of execution of the proposed algorithm.

Key words: FPGA, VHDL, lane detection, Zynq.

ŽIVOTOPIS

Martin Martin rođen je 6. prosinca 1994. godine u Zagrebu. Završio je osnovnu školu Julija Benešića u Iloku, nakon čega upisuje Tehničku školu Nikole Tesle u Vukovaru, smjer tehničar za računarstvo. Nakon završetka srednjoškolskog obrazovanja upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Sudjelovao je na natjecanju iz informatike na Elektrijadi u Budvi 2017. godine te na natjecanju STEM Games u Poreču 2018. godine. Na istom fakultetu upisuje diplomski studij, smjer računalno inženjerstvo. U suradnji s Institutom RT-RK Osijek, izrađuje diplomski rad kroz koji se usavršava u području FPGA tehnologije.

PRILOG

Prilog P.1 Vlastito rješenje algoritma detekcije voznih linija koristeći Matlab

```
I = imread('road.png');
IbwFull = rgb2gray(I);
Ibw = zeros(64, 64);

GxMatrix = [1 0 -1; 2 0 -2; 1 0 -1];
GyMatrix = [1 2 1; 0 0 0; -1 -2 -1];
GausMatrix = [1 2 1; 2 4 2; 1 2 1];

thetah = -89:1:90;
roh = zeros(1, size(thetah,2));
offset = (round(sqrt(size(IbwFull,1)^2 + size(IbwFull,2)^2)));
rothetah = zeros(offset*2+1,size(thetah,2)+1);

for imagePartX = 0 : 3
    for imagePartY = 0 : 3
        Ibw = IbwFull((64 * imagePartX)+1:(64 * (imagePartX+1)),(64 *
imagePartY)+1:(64 * (imagePartY+1)));
        Gx = zeros(size(Ibw, 1), size(Ibw, 2));
        Gy = zeros(size(Ibw, 1), size(Ibw, 2));
        G = zeros(size(Ibw, 1), size(Ibw, 2));
        Gaus = zeros(size(Ibw, 1), size(Ibw, 2));
        Theta = zeros(size(Ibw, 1), size(Ibw, 2));

        for c = 1:(size(Ibw, 1))
            for r = 1:(size(Ibw, 2))
                if c == 1 || r == 1 || c == size(Ibw, 1) || r == size(Ibw, 2)
                    Gaus(r, c) =Gaus(r, c);
                else
                    for cMini = c-1:c+1
                        for rMini = r-1:r+1
                            Gaus(r, c) = Gaus(r, c) + (double(Ibw(rMini,
cMini)) * GausMatrix(rMini-r+2, cMini-c+2));
                        end
                    end
                    Gaus(r, c) = Gaus(r, c)/16;
                end

            end
        end
        for c = 2:(size(Ibw, 1)-1)
            for r = 2:(size(Ibw, 2)-1)

                for cMini = c-1:c+1
                    for rMini = r-1:r+1
                        Gx(r, c) = Gx(r, c) + (double(Gaus(rMini, cMini)) *
GxMatrix(rMini-r+2, cMini-c+2));
                        Gy(r, c) = Gy(r, c) + (double(Gaus(rMini, cMini)) *
GyMatrix(rMini-r+2, cMini-c+2));
                    end
                end
                G(r, c) = sqrt(((Gx(r, c))^2) + ((Gy(r, c))^2));
                G(r, c) = round(G(r, c)*(255/1141));
                Theta(r, c) = atan((Gy(r, c))/(Gx(r, c))) * (180/pi);
                Theta(r, c) = mod(Theta(r, c), 180);

                if (Theta(r, c) <= 22.5) || (Theta(r, c) >=157.5)
                    Theta(r, c) = 0;
```

```
                elseif (Theta(r, c) <= 67.5)
                    Theta(r, c) = 1;
                elseif (Theta(r, c) <= 112.5)
                    Theta(r, c) = 2;
                elseif (Theta(r, c) <= 157.5)
                    Theta(r, c) = 3;
                end
            end
        end

        THHigh = 68;
        THLow = 22;

        for c = 2:(size(Ibw, 1)-1)
            for r = 2:(size(Ibw, 2)-1)
                if (G(r, c) < THLow)
                    G(r, c) = 0;
                elseif G(r, c) < THHigh && G(r, c) > THLow
                    edge = 0;
                    for cMini = c-1:c+1
                        for rMini = r-1:r+1
                           if G(rMini, cMini) > THHigh
                               edge = 1;
                               break
                           end
                        end
                    end
                    if(edge == 0)
                        G(r, c) = 0;
                    end
                end
            end
        end
        for c = 2:(size(Ibw, 1)-1)
            for r = 2:(size(Ibw, 2)-1)
                if (G(r, c) > 0)
                    for thetafor = 25:3:75
                        a=round((r+(64*imagePartX))*cosd(thetafor) +
(c+(64*imagePartY))*sind(thetafor));
                        b=thetafor;
                        roh(thetafor+90) = round((r*(imagePartX-
1))*cosd(thetafor) + (c*(imagePartY-1))*sind(thetafor));
                        rothetah( a+offset, b+90) = rothetah( a+offset, b+90)
+1 ;
                    end
                    for thetafor = -75:3:-25
                        a=round((r+(64*imagePartX))*cosd(thetafor) +
(c+(64*imagePartY))*sind(thetafor));
                        b=thetafor;
                        roh(thetafor+90) = round((r*(imagePartX-
1))*cosd(thetafor) + (c*(imagePartY-1))*sind(thetafor));
                        rothetah( a+offset, b+90) = rothetah( a+offset, b+90)
+1 ;
                    end
                end
            end
        end
    end
end

figure;
```

```
imshow(IbwFull);
hold on;
max_points = 0;
x = 1:256;
y = 1:256;
for c = 1:(size(rothetah, 1)-1)
    for r = 1:(size(rothetah, 2)-1)
        if (rothetah(c,r)>80)
            romax = c;
            thetamax = r;
            max_points = rothetah(c,r);
            for i = 1:size(IbwFull, 1)
                y(i)= (c -offset - x(i)*cosd(r-90))/sind(r-90);
            end
            plot(y,x);
        end
    end
end
```