Detekcija krugova na slikama dobivenim s kamere u automobilu koristeći razvojnu ADAS platformu

Brekalo, Matej

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni studij

Detekcija krugova na slikama dobivenim s kamere u automobilu koristeći razvojnu ADAS platformu

Diplomski rad

Matej Brekalo

Osijek, 2021.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 03.12.2021.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime studenta:	Matej Brekalo
Studij, smjer:	Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije
Mat. br. studenta, godina upisa:	D-31ARK, 06.10.2019.
OIB studenta:	67726084991
Mentor:	Izv.prof.dr.sc. Ratko Grbić
Sumentor:	
Sumentor iz tvrtke:	Matteo Brisinello
Predsjednik Povjerenstva:	Izv. prof. dr. sc. Mario Vranješ
Član Povjerenstva 1:	Izv.prof.dr.sc. Ratko Grbić
Član Povjerenstva 2:	Izv. prof. dr. sc. Marijan Herceg
Naslov diplomskog rada:	Detekcija krugova na slikama dobivenim s kamere u automobilu koristeći razvojnu ADAS platformu
Znanstvena grana rada:	**Umjetna inteligencija (zn. polje računarstvo)**
Zadatak diplomskog rada:	Jedan od zadataka naprednih sustava za pomoć vozaču u vožnji (engl. Advanced Driver Assistance System - ADAS) je detektirati i klasificirati različite tipove objekata u sceni snimljenoj kamerom postavljenom u automobilu. Često je jedna od značajki koja se koristi pri detekciji i klasifikaciji objekata sami oblik objekta. Neki od objekata od interesa koji se žele detektirati i klasificirati su okrugli, kao npr. semafori i okrugli prometni znakovi. U sklopu ovog diplomskog rada potrebno je implementirati dvije različite funkcije za detekciju krugova, na temelju Houghove transformacije i RANSAC-a. Rješenja je potrebno implementirati na osobnom računalu i na ugradbenoj računalnoj platformi za izvršavanje ADAS algoritama. Potrebno je provesti optimizaciju rješenja i raspoređivanje određenih podzadataka na različite procesore ADAS razvojne platforme. Uspješnost implementacije potrebno je najprije testirati na umjetno generiranim slikama koje sadrže krugove, a potom testiranje provesti na realnim slikama iz prometa, snimljenim automotiv kamerom. Pri testiranju, osim uspješnosti rada rješenja potrebo je izmjeriti i brzinu izvođenja te memorijski otisak. Tema rezervirana za: Matej Brekalo Sumentor iz tvrtke: Matteo, Brisinello (Institut RT-RK Osijek d.o.o.)
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)

Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene mentora:	03.12.2021.

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, 08.12.2021.

Ime i prezime studenta:	Matej Brekalo
Studij:	Diplomski sveučilišni studij Automobilsko računarstvo i komunikacije
Mat. br. studenta, godina upisa:	D-31ARK, 06.10.2019.
Turnitin podudaranje [%]:	6

Ovom izjavom izjavljujem da je rad pod nazivom**: Detekcija krugova na slikama dobivenim s kamere u automobilu koristeći razvojnu ADAS platformu**

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

i sumentora

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

Potpis studenta:

SADRŽAJ

1. UVOD

Velik broj prometnih nesreća uzrokovan je isključivo ili u velikoj mjeri ljudskom pogreškom. Te nesreće se mogu potpuno izbjeći ili barem smanjiti njihove posljedice upotrebom naprednih sustava za pomoć vozaču (engl. *Advanced Driver Assistance Systems* - ADAS). Uloga ADAS-a je vožnju učiniti lakšom i udobnijom kao i spriječiti smrtne ishode i ozljede u prometu tako da smanji broj prometnih nesreća i utjecaj onih koje se ne mogu izbjeći.

Upozorenje za napuštanje vozne trake, sustav za detekciju ograničenja brzine i upozoravanje, detekcija mrtvog kuta ili automatsko kočenje u nuždi samo su neki od sustava koji se sve više ugrađuju u moderna vozila. Kako bi ovi sustavi funkcionirali vozilo mora biti opremljeno odgovarajućim senzorima, snažnom računalnom podrškom i aktuatorima. Neki od senzora koji se koriste za prikupljanje tih podataka su senzori brzine, akceleracije, promjene putanje, kuta upravljanja, ultrazvučni senzori, RADAR (engl. *Radio Detection and Ranging*), LiDAR (engl. *Light Detection and Ranging*) i video kamere. Svi navedeni senzori imaju različite radne karakteristike. Radne karakteristike koje se uzimaju u obzir kada se ocjenjuju performanse senzora su koliko on dobro prikuplja podatke pomoću kojih se vrši detekcija objekata, klasifikacija objekata, procjena udaljenosti, detekcija rubova, praćenje voznog traka te ostale karakteristike kao što su doseg vidljivosti, performanse po lošem vremenu i performanse pri lošem osvjetljenju. Najrobusniji sustav bi se dobio kombinacijom svih navedenih senzora. Međutim zbog visoke cijene nekih senzora to se izbjegava kod vozila niže i srednje klase, jer bi krajnja cijena vozila bila previsoka.

Senzor koji se najviše koristi u ADAS aplikacijama je kamera. ADAS se oslanja na kamere kako bi točno detektirao i klasificirao druga vozila, pješake, prepreke, prometne znakove, linije po cesti i drugo. Informacije dobivene pomoću kamere mogu se brzo obraditi računalnom podrškom, nakon čega se može automatski reagirati na tijek vožnje sustavom za kočenje ili zakretanjem volana. Iako kamera ima loših strana (rad u uvjetima lošeg osvjetljenja, kiše, magle i sl.) danas je postala osnovni senzor u ADAS aplikacijama zbog toga što su relativno jeftine i pružaju veliku količinu informacija o okolini vozila na kojeg su montirane. Sve više novih vozila ima ugrađene kamere u retrovizorima kojima se lako može pratiti stanje prometa ispred vozila.

Jedan od zadataka ADAS-a je detektirati i klasificirati različite tipove objekata na slici dobivenoj pomoću kamere montirane na prednjoj strani vozila. Često je jedna od značajki koja se koristi pri detekciji i klasifikaciji objekata sami oblik objekta. Tako u prometu ima mnogo objekata od interesa koji su okrugli kao što su znakovi ili semafori. Ako se nešto želi detektirati na slici, u

ovom slučaju krug, potrebno je smanjiti broj informacija na slici što se postiže detekcijom rubova. Detekcija rubova se odnosi na proces identifikacije i lociranja naglih diskontinuiteta na slici. Tim postupkom uvelike se smanjuje broj detalja na slici i ostaju samo rubovi koji očuvaju bit slike i na temelju kojih je moguće određenim metodama iz računalnog vida detektirati položaj krugova u slici.

U okviru ovog rada razvijeno je rješenje koje detektira krugove u slikama dobivenim s kamere montirane na prednjoj strani vozila. Rješenje se temelji na metodama iz područja računalnog vida. Najprije je razvijen koncept rješenja na osobnom računalu u *Python* programskom jeziku, a nakon toga je rješenje razvijeno u C programskom jeziku također na osobnom računalu. Razvijeno rješenje je zatim implementirano i evaluirano na slikama s kamere montirane na prednjoj strani vozila na ADAS razvojnoj platformi.

Rad se sastoji od četiri poglavlja. U drugom poglavlju detaljnije je opisan problem detekcije krugova te je predstavljeno nekoliko postojećih rješenja ovog problema kao i njihove prednosti i nedostaci. U trećem poglavlju prikazano je vlastito rješenje za detekciju krugova na slikama iz prometa kao i metode, tehnike i poduzeti koraci kojima je ovaj problem riješen. Četvrto poglavlje bavi se rezultatima testiranja predloženog rješenja. Na kraju rada dan je zaključak i korištena literatura.

2. PREGLED RJEŠENJA ZA DETEKCIJU KRUGOVA U ADAS APLIKACIJAMA

2.1. Problem detekcije krugova

Jedan od zadataka naprednih sustava za pomoć vozaču u vožnji je detektirati i klasificirati različite tipove objekata u sceni snimljenoj kamerom postavljenoj u automobilu. Pod pojmom detekcija podrazumijeva se određivanje točne pozicije objekta na slici koja se označava graničnim pravokutnikom (engl. *Bounding Box*), a pod pojmom klasifikacije podrazumijeva se pridruživanje objekta klasi kojoj on pripada. Često je jedna od značajki koja se koristi pri detekciji i klasifikaciji objekata sami oblik objekta. Neki od objekata od interesa koji se žele detektirati i klasificirati su okrugli, kao npr. semafori i okrugli prometni znakovi. Prometni znakovi sadrže korisne informacije koje mogu obavijestiti sustav i vozača o nadolazećoj promjeni stanja na cesti, ograničenjima, zabranama, upozorenjima i ostalim korisnim informacijama. Ove informacije su kodirane u znakovima svojim oblikom, bojom i slikom. Svi okrugli znakovi spadaju u skupinu znakova izričitih naredbi koji mogu ukazivati na zabranu prometa u nekom od smjerova, na zabranu prometa za neka vozila, na ograničenje brzine, na obavezan smjer kretanja itd. Semafor je još jedan važan uređaj u prometu za signalizaciju i regulaciju prometa čija svjetla imaju oblik kruga.

Pogrešna procjena ili neuočavanje znaka u prometu može direktno ili indirektno utjecati na vjerojatnost prometne nesreće. Zbog gužve u prometu ili loših vremenskih uvjeta čovjek veoma lako može pogrešno procijeniti znak ili ga može uopće ne primijetiti. Kod takvih situacija, ako u vozilu postoje sustavi za detekciju prometnih znakova, sustav može kompenzirati pogrešku vozača i obavijestiti ga o prisutnosti znaka, što u konačnici vožnju čini sigurnijom i lakšom. Interes prema sustavima za detekciju i klasifikaciju objekata je porastao u zadnje vrijeme i to zbog potencijalne primjene u više aplikacija. Primjena sustava za detekciju znakova je široka, pa tako možemo detektirati prisutnost znakova u raznim uvjetima i na raznim mjestima kao što su gradovi ili autoceste. Postoji niz izazova koji stoje ispred ovakvog jednog sustava, jer performanse i mogućnosti tog sustava uvelike ovise o uvjetima okoline i vidljivosti samih znakova ili semafora u prometu. Loše osvjetljenje i loši vremenski uvjeti jedni su od problema koji se mogu često pojaviti. Postoji još niz drugih parametara koji mogu utjecati na točnost detektiranja znaka na slici. Neki od tih parametara su loš položaj znaka, djelomična okluzija znaka, oštećeni znak itd.

Primjeri kružnih objekata u prometu, odnosno semafora i prometnih znakova, prikazani su slikom 2.1. Prikazani znakovi pružaju informacije o upozorenjima, ograničenjima i zabranama.

Slika 2.1. Primjeri kružnih oblika u prometu

Detekcija krugova je problem koji se može riješiti upotrebom metoda temeljenim na klasičnom računalnom vidu ili upotrebom metoda temeljenim na strojnom učenju. Neke od metoda koje su temeljene na klasičnom računalnom vidu a mogu se koristiti za detekciju krugova su *Hough*-ova transformacija i *Random Sample Consensus* (RANSAC). Kod pristupa metodama temeljenim na strojnom učenju, mogu se koristiti konvolucijske neuronske mreže. U nastavku ovog poglavlja dan je kratki pregled metoda za detekciju i klasifikaciju prometnih znakova.

2.2. Pregled postojećih rješenja za detekciju krugova

U radu [1] predstavljena je nova arhitektura hardvera za izvođenje generalizirane *Hough*-ove transformacije (engl. *Generalized Hough Transform - GHT*). Arhitektura se može konfigurirati kako bi se omogućio kompromis između performansi, točnosti i korištenja hardvera. Predložena arhitektura implementirana je na povoljnom *Zynq-7000 FPGA*-u (engl. *Field-Programmable Gate Array*) i testirana u dvije praktične primjene, detekciji brana na slikama iz zraka i detekciji prometnih znakova. Lanac obrade slike sastoji se od modula za detekciju rubova temeljenu na *Canny* algoritmu. Slijedi niz od jednog ili više *GHT* blokova. R tablica, koja služi za pohranjivanje glasova, pohranjena je u memorijskim ćelijama na čipu (engl. *BlockRAM*). Budući

da su memorijske ćelije obično s dvostrukim ulazom, svaka R tablica može se dijeliti s dvije *GHT* ćelije. Lanac obrade slike kreiran je korištenjem *ASTERICS* okvira, koji također sadrži implementaciju *Canny* detektora rubova. Modul detektora rubova proširen je *CORDIC* modulom za generiranje preciznih smjerova gradijenta. Svaka *GHT* ćelija sadrži memoriju birača i potrebnu logiku za samostalno izvođenje potpunog *GHT*-a. Međutim, ako je prisutno više *GHT* ćelija, one se mogu proizvoljno konfigurirati da paraleliziraju izračune na mnogo različitih načina i na kraju da kombiniraju svoje R tablice u jednu veliku virtualnu memoriju. Na primjer, jedna *GHT* ćelija može biti konfigurirana za obradu samo svih kutova rotacije između 0 i 180 stupnjeva, a druga za obradu svih kutova između 180 i 360 stupnjeva. To će otprilike zauzeti polovicu računskog napora po ćeliji, a svaka njihova memorija konačno će sadržavati podatke od polovice prostora parametara. Nakon završetka procesa glasovanja, kompletan sadržaj memorije može se očitati i analizirati. Logika maksimalne vrijednosti može se koristiti za pronalaženje spremnika s najvećim brojem glasova. Predstavljeni *GHT* dizajn testiran je na slikama koje sadrže prometne znakove. Rezultati testiranja prikazani su tablicom 2.1. Za svaki od tri prometna znaka, tablica navodi broj unosa R tablice, broj dijelova slike referentne vrijednosti koji sadrže znak, te minimalni, prosječni i maksimalni broj rubnih točaka po slici koju isporučuje *Canny* modul. Redak "vrijeme" prikazuje minimalno, prosječno i maksimalno vrijeme obrade, koje nikada ne prelazi 28 ms, što znači da se slike rezolucije 640x480 mogu obraditi brzinom većom od 35 okvira po sekundi. Zadnji redak označava postotak slika na kojima je prometni znak otkriven u ispravnom položaju. Položaj detekcije određen je jednostavnom maksimalnom pretragom u akumulacijskom polju.

Tablica 2.1. Rezultati testiranih slika [1].

Znak	*"ograničenje brzine"*	*"pješački prijelaz"*	*"stop"*
Broj unosa u R tablicu	488	531	441
Broj slika sa znakom	60	10	21
Broj rubova (min/avg/max)	6.5k/23k/59k	7.7k/33k/64k	12k/23k/42k
Vrijeme [ms] (min/avg/max)	6.16/10.4/21.0	6.74/16.1/28.0	7.29/10.5/17.2
Stopa detekcije [%]	50.00%	70.00%	66.67%

Predstavljeno rješenje nadmašuje postojeće pristupe u ovoj primjeni, jer su pokrivene 4 dimenzije u *Hough*-ovoj transformaciji na vrlo efikasan način. Testiranjem se pokazalo da je arhitektura sposobna nositi se s oba zadatka (detekcija brana i prometnih znakova) uz postizanje brzine dovoljne za rad u stvarnom vremenu. Nedostatak rješenja je relativno niska stopa detekcije prometnih znakova.

Rad [2] donosi izazove koji se odnose na otkrivanje prometnih znakova u stvarnom vremenu. Prikazan je pregled postojećih metoda detekcije kao što su detekcija temeljena na boji, na obliku i na strojnom učenju. Također raspravlja o podudarnosti značajki i algoritmima strojnog učenja koji se koriste u fazi prepoznavanja prometnih znakova. U radu se navode neki izazovi koji su uključeni u detekciju i klasifikaciju prometnih znakova kao što su varijacije osvjetljenja ovisno o dijelu dana i vremenskim uvjetima, boja prisutna na znakovima blijedi s vremenom, geometrijska izobličenja zbog oštećenja, varijacije u mjerilu kako se vozilo približava prometnom znaku, ostali slični oblici na slici, okluzija drugim objektima i sl. Faza detekcije prometnih znakova uključuje identificiranje regije koja sadrži prometne znakove u okviru slike ili videa. Budući da su prometni znakovi sastavljeni od specifičnih boja i oblika, mogu se otkriti korištenjem informacija o boji i obliku. Sljedeći odjeljak u radu pruža kratak pregled postojećih metoda detekcije na temelju boja, oblika, boja i oblika i drugih pristupa. U radu se uspoređuje stopa točnosti detekcije pri korištenju RGB, HSV i HSI sustava boja kod slučaju detekcije prometnih znakova pomoću boje. U slučaju korištenja RGB sustava boja stopa detekcije iznosi 88.75%, u slučaju gdje se koristi HSV sustav boja točnost detekcije iznosi 95%, korištenjem HSI sustava boja postiže se točnost od 91%. Iz navedenog se može zaključiti da je najveća točnost detekcije pri korištenju HSV sustava boja. U radu je obrađena i detekcija prometnih znakova na temelju oblika. Za okrugle i trokutaste znakove korištena je *Hough*-ova transformacija, dok se za detekciju oktogona koristila transformacija radijalne simetrije. Uspješnost detekcije krugova i trokuta iznosi 97.3%, dok je uspješnost detekcije oktogona nešto niža i iznosi 94.3%. Osim detekcije u radu se spominje i klasifikacija prometnih znakova pomoću algoritama temeljenih na izdvajanju značajki i korištenjem algoritama strojnog učenja. U radu su uspoređena dva algoritma za izvlačenje značajki: BRISK (engl. *Binary Scalable Key point*) i nešto sporiji SURF (engl. *Speeded Up Robust Features*). U radu je komentirano i nekoliko pristupa detekciji pomoću strojnog učenja. Korištenjem umjetne neuronske mreže (engl. *Artificial Neural Network* - ANN) postignuta je točnost detekcije od 94.7%. Dubokom neuronskom mrežom postigla je se točnost detekcije od 98.83%. Korišteno je i još nekoliko tehnika detekcije prometnih znakova kao što su nadzirano učenje, deskriptori značajki, GA (engl. *Genetic Algorithm*), itd.

U radu [3] opisan je algoritam za detekciju i praćenje prometnih znakova u slikama dobivenim s kamerom montiranoj na prednjoj strani vozila. Kao dodatnu informaciju, algoritam koristi informaciju o brzini vozila što omogućuje predviđanje ne samo prisutnosti objekta, već i njegove veličine i lokacije. Tako se povećava točnost detekcije, uz zanemarivo povećanje računske složenosti. Prepoznavanje prometnih znakova se odvija u tri koraka: priprema slike, detekcija i

klasifikacija. Detekcija se temelji na informacijama o boji dobivenim iz slika. Stoga priprema slike počinje s postavljanjem praga boje i smanjenjem šuma. Zatim se za lokaliziranje znakova na slikama koristi modifikacija generalizirane *Hough*-ove transformacije. Za provjeru prisutnosti znaka provodi se postupak praćenja na temelju brzine vozila. Konačno, otkriveno područje se klasificira. Filtriranje slike po boji izvedeno je prebacivanjem slike iz RBG sustava boja u HSV sustav boja. Time se postigla mogućnost lakšeg postavljanja gornjeg i donjeg praga filtera boje. Ovo je potrebno zato što dio znaka nekada može biti tamniji ili svjetliji zbog utjecaja sunca ili sjene od drugih objekata. Sljedeći korak u radu je smanjivanje šuma u slici. To se postiglo korištenjem algoritma za smanjenje šuma koji se temelji na detekciji i retuširanju točkastih odsjaja na reprodukcijama umjetničkih djela. Za detekciju ovih odsjaja korišten je algoritam kliznih prozora. Glavna prednost ovakvog algoritma je da se parametri mogu postaviti tako da će se ukloniti samo šum nalik točkama. Nakon filtriranja slijedi detekcija prometnih znakova implementirana modificiranom generaliziranom *Hough*-ovom transformacijom (GHT). Glavna razlika od originalnog GHT-a je u korištenju drugačijeg akumulatorskog prostora i izbjegavanju konstrukcije R tablice. Nakon primjene posebnog trokutastog predloška na binarnu sliku, točka s maksimalnom vrijednošću središnja je točka traženog objekta. U ovom radu objašnjena je detekcija samo trokutastih oblika. Međutim opisani algoritam može se koristiti i za detekciju kvadratnih i okruglih prometnih znakova. Za praćenje i predikciju položaja znaka iskorištena je poznata brzina vozila dobivena s putnog računala ili GPS senzora. Ako je poznata brzina vozila i brzina kojom kamera generira slike lako se može izračunati udaljenost znaka od vozila. Razvijeni algoritam testiran je na video okvirima dobivenim na ulicama grada Samare pomoću kamere ugrađene u automobil. Za procjenjivanje točnosti algoritma za detekciju i praćenje, korištena je javno dostupna baza podataka prometnih znakova naziva Njemačko mjerilo za otkrivanje prometnih znakova (engl. *German Traffic Sign Detection Benchmark* - GTSDB) koja sadrži više od 50000 slika. Prilikom testiranja razvijenih algoritama korišteno je 9987 slika koje sadrže prometne znakove traženog oblika i crvenih kontura. Eksperimenti su pokazali 97,3% ispravno otkrivenih i prepoznatih prometnih znakova zabrane i opasnosti. Prednosti ovog algoritma su što koristi specijalni filter za uklanjanje šuma sa slike što skraćuje vrijeme trajanje algoritma i pomaže u ispravnoj detekciji prometnog znaka. Algoritam može raditi u stvarnom vremenu. Nedostatak rješenja je što radi samo za trokutaste oblike prometnih znakova.

U radu [4] predlaže se novi algoritam za otkrivanje i prepoznavanje okruglih prometnih znakova. Prvo se koristi segmentacija slike po boji i analiza povezane domene za detekciju područja od interesa (engl. *Region of Interest* - ROI), a zatim se koristi poboljšana dvostruka

Hough-ova transformacija kako bi se zasebno odredio centar kruga i radijus prometnog znaka. Za prepoznavanje se koristi Ponderirana Hausdorffova udaljenost (engl. *Weighted Hausdorff Distance*) jer je dokazana visoka točnost podudaranja, otpornost na šum i okluziju te je prikladna za poseban slučaj kineskih prometnih znakova. Za segmentaciju slike po boji koristi se HSI sustav boja jer on može biti invarijantan na promjenu svjetline. Nakon segmentacije uklonjena je većina slike koja nije iste boje kao znak. Na slici će također ostati i dijelovi slike koji su iste boje kao znak, a ne pripadaju znaku. Nakon filtriranja šumova iz slike slijede dvije iteracije *Hough*-ove transformacije. Prva *Hough*-ova transformacija se koristi za izračunavanje najvjerojatnijih središta kružnice. U drugoj iteraciji *Hough*-ove transformacije se prema jednadžbi kružnice izračunava udaljenost između točaka koje su pohranjene u fazi detekcije središta kruga i središta kružnice. Ako udaljenost zadovoljava raspon radijusa [r, s] koji se temelji na stvarnoj veličini prometnog znaka, akumulator radijusa dodaje 1. Nakon crtanja histograma akumulatora u idealnom slučaju, krugovi će se prikazati kao oštri lokalni maksimumi u histogramu radijusa. Nakon detekcije prometnog znaka, ROI se izdvaja iz izvorne slike. Zatim postoje dva koraka u prepoznavanju prometnih znakova, jedan je predobrada ROI, a drugi precizni postupak mjerenja podudaranja. U fazi mjerenja podudaranja, prilagodljiva Hausdorffova udaljenost na temelju težina sličnosti primjenjuje se za mjerenje udaljenosti između ROI-a i referentnih slika. Rješenje je testirano na 337 realnih slika koje sadrže prometni znak. Točnost detekcije znakova, ovisno o vrsti znaka, kreće se između 79.2% i 92.9%. U usporedbi s konvencionalnim metodama, predloženi algoritam ima manje memorijske zahtjeve, manju vremensku složenost te dobru robusnost s obzirom na nagib i okluziju.

U radu [5] korištene su metode bazirane na *Hough*-ovoj transformaciji u svrhu detekcije prometnih znakova. Predstavljeno je nekoliko koraka predobrade i podešavanja slika kako bi se povećao postotak točnih detekcija. Uspostavljena je nova metoda za detekciju boja prometnih znakova kao i novi algoritam sličan *Hough*-ovoj transformaciji za detekciju kružnih i trokutastih oblika. U radu je predstavljeno nekoliko metoda za detekciju okruglih i trokutastih prometnih znakova kao što su RPD (engl. *Regular Polygon Detector*), RSD (engl. *Radial Symmetry Detector*), VBT (engl. *Vertex Bisector Transform*), BCT (engl. *Bilateral Chinese Transform*), STVUT (engl. *Single Target Vote for Upright Triangles*), STVUE (engl. *Single Target Vote for Upright Ellipses*). RPD, STVUT i VBT se koriste za detekciju trokutastih znakova, a RSD, STVUE i BCT za detekciju okruglih znakova. STVUT i STVUE nisu ograničeni na detekciju pravilnih oblika trokuta i kruga već mogu detektirati i njihove istegnute oblike, odnosno jednakokračne trokute i elipse. U radu je također predloženo sedam metoda predobrade koji

transformiraju dobivenu sliku u boji u gradijentnu sliku koja svakom elementu slike dodjeljuje veličinu i orijentaciju. Te metode su GMT (engl. *Gradient Magnitude Threshold*), CG (engl. *Colour Gradient*), LCG (engl. *Learned Colour Gradient*), CT (engl. *Learned Colour Threshold*), LCSG (engl. *Learned Colourwise Segmentation Gradient*), EO (engl. *Expected Orientations*) i LCGCV (engl. *Learned Colour Gradient with Constant Vote*). Za testiranje rješenja korišteno je 847 slika koje sadrže 251 znak. Tablica 2.2. prikazuje točnost detekcije za svaku kombinaciju metoda detekcije i predobrade slike. Prednost ovog rješenja je smanjeni broj nepotrebnih glasova u akumulacijskom polju što smanjuje potrebnu memoriju za njihovo spremanje. Nedostatak je prilično niska stopa detekcije znakova.

Tablica 2.2. Postignuti rezultati detekcije prometnih znakova [5].

Znak	RPD	RSD	VBT	BCT	STVUT	STVUE
GMT	11%	65%	15%	44%	26%	62%
CG	19%	91%	15%	47%	52%	68%
LCG	52%	90%	74%	86%	78%	91%
CT	63%	56%	63%	45%	70%	72%
LCSG	11%	47%	7%	17%	52%	43%
EO	48%	-	81%	-	81%	-
LCGCV	41%	90%	70%	83%	74%	84%

3. PREDLOŽENO RJEŠENJE ZA DETEKCIJU KRUGOVA U ADAS APLIKACIJAMA

U ovom poglavlju prvo je predstavljena teorijska podloga koja podupire predloženo rješenje za detekciju krugova. Zatim je dan opis predloženog rješenja za detekciju krugova u ADAS aplikacijama koje se temelji na poznatim tehnikama iz područja računalnog vida. Krajnji cilj je implementacija rješenja na razvojnu platformu, međutim zbog lakšeg razvoja prvo je razvijen i testiran koncept rješenja na osobnom računalu u *Python* programskom jeziku. Zatim je slijedilo razvijanje rješenja u programskom okruženju *Visual Studio* korištenjem C programskog jezika također na osobnom računalu. Potom se implementirano rješenje na PC-u testiralo na setu slika iz prometa. U konačnici je rješenje prilagođeno kako bi moglo raditi i biti testirano na ADAS razvojnoj platformi.

Proces detekcije krugova na slici pomoću standardnih metoda u okviru računalnog vida može se podijeliti na nekoliko koraka. Ulaz u algoritam su slike iz prometa dobivene s kamere montirane na prednjoj strani vozila. Zatim je potrebno smanjiti količinu detalja na slici i ukloniti šum kako bi sljedeći korak dao bolje rezultate. U ovom rješenju odabran je medijan filter za filtriranje slike jer je pokazao najbolje rezultate. Nakon filtriranja slike tok obrade slike se dijeli u dvije grane obrade. U jednoj grani obrade se na slici detektiraju rubovi pomoću *Canny* operatora nakon čega nastaje binarna slika. U drugoj grani obrade se slika filtrira po bojama koje su od interesa u prometu, kao što je na primjer crvena ili plava jer su okrugli znakovi uglavnom crvene i plave boje. Nakon filtriranja po boji dobiva se također binarna slika kao i u prvoj grani obrade. Kako ne bi došlo do gubitka podataka nad binarnom slikom u grani obrade potrebno je izvršiti proces dilatacije. Kao posljednji korak pripreme slike, dvije dobivene binarne slike se spajaju u jednu pomoću logičkog operatora I (engl. *AND*) na bazi elementa slike, tj. pojedini element slike će ostati prikazan kao rub samo ako se u obje slike na tom mjestu nalazio rub. Nastala binarna slika predstavlja ulaz u algoritam za detekciju krugova. U ovom radu korištena su dva algoritma za detekciju krugova, *Hough*-ova transformacija i RANSAC algoritam.

3.1. Teorijska podloga predloženog rješenja

Ovo potpoglavlje daje teorijsku podlogu koja je potrebna za razumijevanje rada rješenja. Prvo su dani detalji o medijan filteru koji je korišten za smanjenje šuma na slikama. Slijedi opis HSV sustava boja koji je korišten za filtriranje slike po boji i *Canny* detektora rubova. Nakon toga objašnjena je *Hough*-ova transformacija i RANSAC algoritam za detekciju krugova.

3.1.1. Medijan filter

Filtriranje slike pripada u postupak predobrade digitalne slike, a radi se za potrebe poboljšanja kvalitete slike. Filtriranje slike može se provoditi u prostornoj i frekvencijskoj domeni. U prostornoj domeni obrada se odnosi na samu ravninu (plohu) slike, a metode obrade slike su zasnovane na izravnoj manipulaciji na elementima slike. Za filtriranje slike u prostornoj domeni najčešće se koristi konvolucijska maska koja se primjenjuje na grupu elemenata slike. Konvolucija je operacija susjedstva u kojoj je svaki izlazni element slike ponderirani zbroj susjednih ulaznih elemenata slike. Koeficijenti su grupirani u matricu, nazvanu konvolucijska maska. Filtere dalje možemo podijeliti na niskopropusne i visokopropusne. Kod niskopropusnih filtera se prigušuju visoke prostorne frekvencije što rezultira smanjenim detaljima i manje uočljivim rubovima u izlaznoj slici. Visokopropusni filteri se koriste za izoštravanje slike, jer naglašavaju rubove u rezultantnoj slici. Medijan filter spada u skupinu niskopropusnih filtera i predstavlja posebnu vrstu filtriranja, gdje se kao rezultat uzima medijan vrijednosti svih piksela u M x N okolini piksela koji se trenutno obrađuje. Ovaj filter učinkovito uklanja *"salt and pepper"* tip šuma. Slika 3.1. prikazuje učinkovitost medijan filtera na ranije spomenutim šumovima. Slika 3.1. a) prikazuje sliku sa šumom, dok slika 3.1. b) prikazuje sliku s uklonjenim šumom pomoću medijan filtera. Ovaj niskopropusni filter je odabran zato što dobro uklanja šumove koji se javljaju u slici kod loših uvjeta snimanja kao što su niska razina osvjetljenja, magla, kiša i sl. Nedostatak običnog niskopropusnog filtera je što loše uklanja navedene šumove u slici i zbog toga nije korišten.

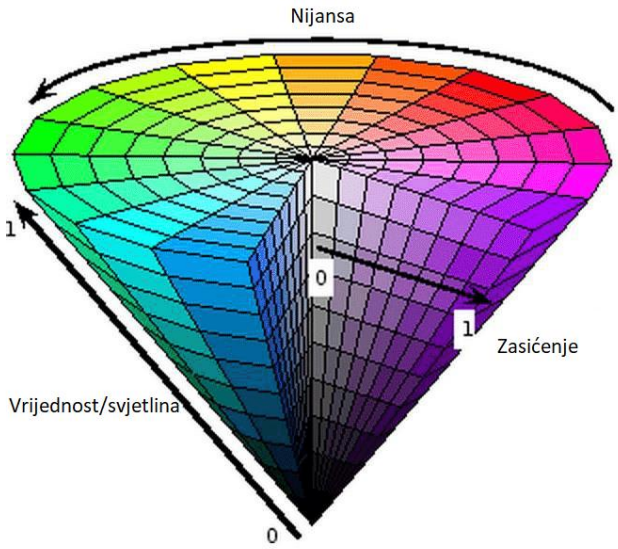
a) b)

Slika 3.1. Primjena medijan filtera na "*salt and pepper*" tipu šuma a) originalna slika, b) filtrirana slika

3.1.2. HSV sustav boja i filtriranje po boji

HSV sustav boja opisuje boje (ton ili nijansu) u smislu njihove sjene (zasićenost ili količina sive boje) i njihove vrijednosti svjetline. Nijansa (engl. *Hue*) je dominantna boja promatrana od strane čovjeka. Zasićenje (engl. *Saturation*) predstavlja količinu bijele svjetlosti u nijansi. Vrijednost (engl. *Value*) jednostavno predstavlja intenzitet boje od 0 do 100 posto, gdje je 0 potpuno crno, a 100 najsvjetlija nijansa boje. Kao što se može vidjeti na slici 3.2., HSV sustav boja se može prikazati kao stožac, gdje kut predstavlja nijansu boje, pri čemu crvena boja kreće od kuta 0°, zatim se boja mijenja u zelenu na 120°, u plavu na 240° i na kraju opet završava u crvenoj boji na kutu 360°. Udaljenost između centra i ruba stošca predstavlja zasićenost. Zasićenje prema vanjskom rubu stošca raste i boja postaje sve izraženija. Centralna vertikalna os stošca predstavlja vrijednost ili svjetlinu, na dnu stošca boja je najtamnija odnosno crna, a kako se vrijednost povećava, povećava se i svjetlina boje [6].

Ovaj sustav boja je odabran zbog lakše manipulacije s nijansom boje. Tako se jednostavno može odabrati jedna boja i sve njezine nijanse svjetline postavljanjem gornjeg i donjeg praga. Ovo je potrebno zato što dio prometnog znaka nekada može biti tamniji ili svjetliji zbog utjecaja sunca ili sjene od drugih objekata. Filtriranje po boji u HSV sustavu boja se radi tako da se odabere gornji i donji prag H, V i S vrijednosti za boju koja se želi filtrirati. Boja koja se želi filtrirati definira se pragovima u H. Zatim se svaki element uspoređuje s vrijednostima praga i ako se element nalazi u okviru praga zapisuje se binarna jedinica na to mjesto u slici, u suprotnom upisuje se 0. Izlaz filtera je binarna slika koja ima vrijednosti veće od 0 samo na mjestima gdje se nalazila boja koja je bila u okviru postavljenih pragova.



Slika 3.2. Vizualni prikaz HSV sustava boja [7].

3.1.3. Canny detektor rubova

Canny je operator detekcije rubova koji koristi višestupanjski algoritam za otkrivanje širokog raspona rubova na slikama. Kao ulaz prima sliku sivih tonova, a kao izlaz proizvodi binarnu sliku s istaknutim rubovima. Prije svega, ulazna slika se filtrira niskopropusnim Gaussovim filterom. Zatim se primjenjuje jednostavan 2-D operator prve derivacije na filtriranu sliku kako bi se istaknula područja slike s visokim prvim prostornim derivacijama. Algoritam zatim prati ta područja i postavlja na nulu sve elemente slike koji imaju vrijednost nižu od maksimalne za to područje kako bi se dobila tanka linija u izlazu, proces poznat kao potiskivanje ne-maksimalnih vrijednosti. Za taj postupak potrebno je imati gornji i donji prag koji će odlučivati koji se elementi slike postavljaju na nulu, a koji ne. Praćenje ruba počinje propuštanjem svih vrijednosti koje su iznad gornjeg praga. Zatim se svi elementi koji su ispod vrijednosti donjeg praga postavljaju na 0. Vrijednosti koje se nalaze između ta dva praga se dodatno provjeravaju tako da se provjeravaju okolni element i to tako da ako se pokraj tog ruba nalazi drugi rub, on će se također smatrati rubom. Postavljanje pragova je izuzetno važno jer previsoko postavljen prag može propustiti važne informacije. S druge strane, prenizak prag lažno će identificirati nebitne informacije kao važne, npr. šum. Teško je odrediti jedinstvenu vrijednost praga koji dobro funkcionira na cijelom setu slika za dani problem, zbog toga je potrebno eksperimentalnim putem odrediti granice pragova.

3.1.4. *Hough*-ova transformacija

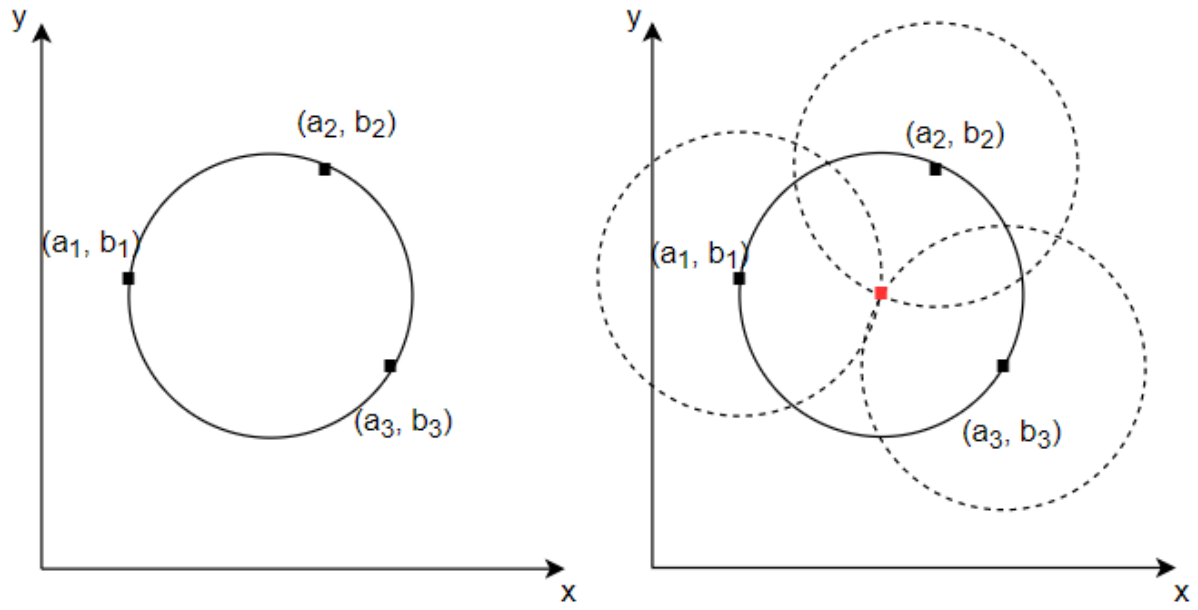
Hough-ova transformacija je tehnika koja se koristi za izoliranje obilježja određenog oblika unutar slike. Budući da zahtjeva da se željene značajke navedu u nekom parametarskom obliku, klasična *Hough*-ova transformacija se najčešće koristi za otkrivanje pravilnih krivulja poput linija, krugova, elipsa, itd. Općenita *Hough*-ova transformacija se može primijeniti u aplikacijama u kojima jednostavan analitički opis značajki nije moguć. Glavna prednost *Hough*-ove transformacije je u tome što je tolerantna na praznine u opisima krivulja i što je u određenoj mjeri otporna na šum u slici.

Ulaz u *Hough*-ovu transformaciju je slika s istaknutim rubovima, stoga je prvi korak korištenje nekog detektora rubova. Zatim je potrebno odrediti odgovarajuću jednadžbu koja će se koristiti ovisno o tome koji oblik se želi detektirati na slici. Ako se žele detektirati krugovi potrebno je iz izraza (3-1), gdje je t parametarska varijabla iznosa 0 do 2π, a i b koordinate središta kruga i r polumjer kruga, definirati jednadžbu kružnice koja omeđuje krug.

$$x = a + r\,cos\,t$$
$$y = b + r\,sin\,t \tag{3-1}$$

$$(x - p)^2 + (y - q)^2 = r^2 \tag{3-2}$$

Definicijom dolazimo do izraza (3-2), gdje su p i q koordinate središta kružnice $S(p, q)$, a r polumjer kružnice. Iz izraza se vidi su za opis jedne kružnice potrebne tri varijable, pozicija centra na x osi, pozicija centra na y osi i polumjer kružnice. Sljedeći korak je transformacija slike istaknutih rubova u parametarski prostor koji je predstavljen akumulacijskim poljem. U *Hough*-ovom prostoru jedna kružnica s tri parametra je predstavljena jednom točkom. Pošto postoje tri parametra za kružnicu, akumulacijsko polje mora biti trodimenzionalno. Preslikavanje parametara oblika u akumulacijsko polje ostvaruje se postupkom glasanja. Najjednostavniji postupak glasanja je povećavanje vrijednosti u akumulacijskom polju na mjestu čije su koordinate jednake paru (ili n-torci) parametara traženog oblika [8]. Proces prijelaza u *Hough*-ov prostor prikazan je slikom 3.3. Na slici 3.3. a) prikazan je izraženi rub u obliku kružnice, a na slici 3.3. b) je prikazana jedna dvodimenzionalna razina akumulacijskog polja za određeni radijus r. Za svaki element 3.3. a) slike, upisuju se glasovi u akumulacijsko polje na mjesta prikazana isprekidanom linijom na 3.3. b) slici. Na slici 3.3. b) se može vidjeti da se glasovi nakupljaju najviše u centru kružnice prikazane crvenom točkom. U zadnjem koraku potrebno je detektirati lokalne maksimume u akumulacijskom polju. Mjesta u akumulacijskom polju gdje postoje lokalni maksimumi predstavljaju potencijalnu lokaciju centra kružnice [9].



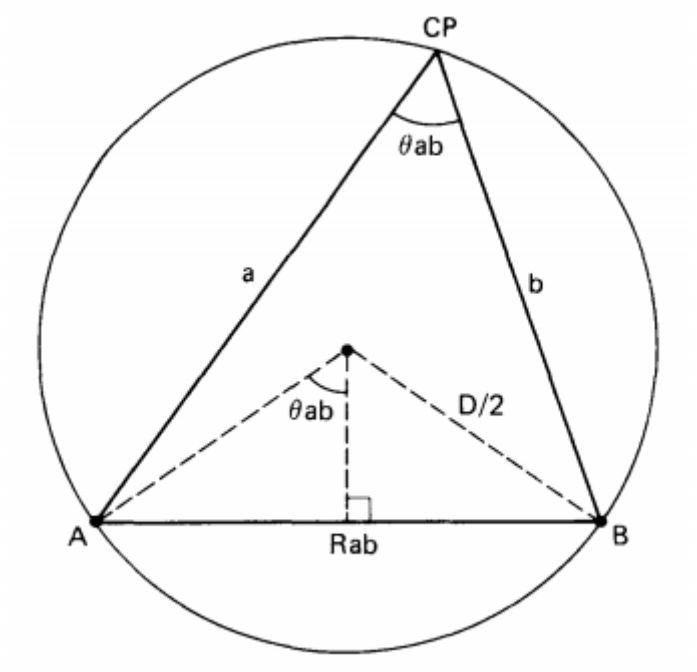
a) b)

Slika 3.3. Proces upisivanja glasova u akumulacijsko polje a) izraženi rub u obliku kružnice, b) jedna dvodimenzionalna razina akumulacijskog polja za određeni radijus r

3.1.5. *Random Sample Consensus* (RANSAC)

Analiza scene i znanost općenito, bavi se interpretacijom podataka u smislu skupa unaprijed definiranih modela. Konceptualno, interpretacija uključuje dvije različite aktivnosti. Prvo, postoji problem pronalaženja najboljeg podudaranja između podataka i jednog od dostupnih modela, ovo predstavlja problem klasifikacije. Drugo, postoji problem izračunavanja najboljih vrijednosti za slobodne parametre odabranog modela, ovo predstavlja problem procjene parametara. Ta dva parametra su najčešće ovisna, pa tako za rješavanje problema klasifikacije često je potrebno rješenje problema procjene parametara. Klasične metode za procjenu parametara, kao što je metoda najmanjih kvadrata, izvode procjenu parametara modela na temelju svih predstavljenih podataka. Te metode nemaju mehanizme za detekciju i otklanjanje podataka koji uvelike odstupaju od ostatka podataka pa se tako unosi velika pogreška u krajnji model. Zbog velikog broja *outlier*-a u podacima koji se razmatraju u ovom radu, takve metode nisu prikladne [10].

RANSAC je iterativna metoda za procjenu parametara matematičkog modela iz skupa promatranih podataka. Glavna razlika između RANSAC metode i klasičnih metoda je u količini podataka koji se uzimaju za procjena parametara modela na temelju podataka. Umjesto da koristi što je više moguće podataka za dobivanje rješenja i eliminiranja točaka koje jako odstupaju od ostalih, RANSAC koristi što je moguće manji početni skup podataka. Na primjer, ako se želi smjestiti kružnica u skup podataka, RANSAC odabire tri točke jer su tri točke dovoljne za jednoznačno određivanje kruga. Zatim se iz te tri točke računa centar kruga i njegov radijus kako je prikazano slikom 3.4. Slika prikazuje geometrijski prikaz kako se iz tri točke u prostoru može doći do parametara kružnice koja prolazi kroz te točke, gdje su odabrane točke označene s A, B i C, a radijus kružnice s D/2. Potrebno je riješiti sustav od tri jednadžbe kako bi se dobila jednadžba kružnice. Dobivenu kružnicu je zatim potrebno dodatno evaluirati odnosno procijeniti koliko dobro opisuje dane podatke.

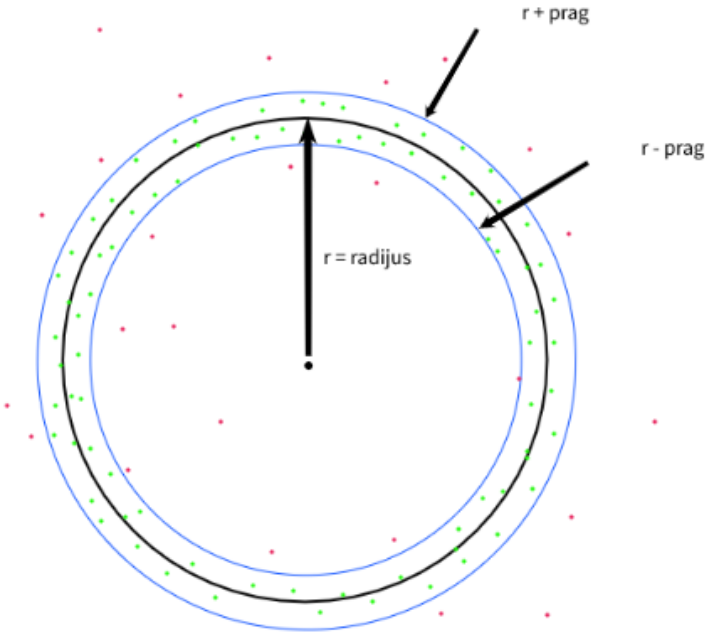


Slika 3.4. Određivanje parametara kruga iz tri točke

Procjena koliko se dobivena kružnica podudara s danim podacima odvija se tako da se broje točke čija je udaljenost od kružnice ispod nekog unaprijed zadanog praga. Pomoću izraza (3-3), gdje je $d(A, B)$ udaljenost između točaka A = (x_1, y_1) i B = (x_2, y_2), računa se udaljenost svake točke od središta kružnice, poznavajući polumjer i zadani prag lako se dolazi do zaključka da li točka pripada kružnici ili ne. Točke čija je udaljenost u zadanom pragu nazivaju se *inlier*-i, a točke koje su izvan praga *outlier*-i.

$$d(A,B) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}, \tag{3-3}$$

Osim praga pomoću kojeg se određuje koji je podatak *inlier*, a koji *outlier* postoji i prag koji određuje minimalni broj *inlier*-a za konkretni radijus da bi model bio valjan. Ako je broj *inlier*-a iznad zadanog praga smatra se da u podacima postoji kružnica. Ako je broj ispod tog praga model se odbacuje i uzimaju se tri nove točke. U ovom rješenju prag koji određuje minimalni broj *inlier*-a da bi se model smatrao kružnicom određen je eksperimentalnim putem za sve radijuse. Na slici 3.5. prikazan je postupak određivanja podataka koji pripadaju modelu koji se smješta. Crnom bojom je prikazana procijenjena kružnica za dane podatke, a dvije plave kružnice predstavljaju granice koje određuju da li se točka broji kao podatak koji pripada tom modelu (*inlier*). Zelenom bojom označeni su *inlier*-i dok su *outlier*-i prikazani crvenom.



Slika 3.5. Postupak određivanja *inlier*-a i *outlier*-a u RANSAC algoritmu

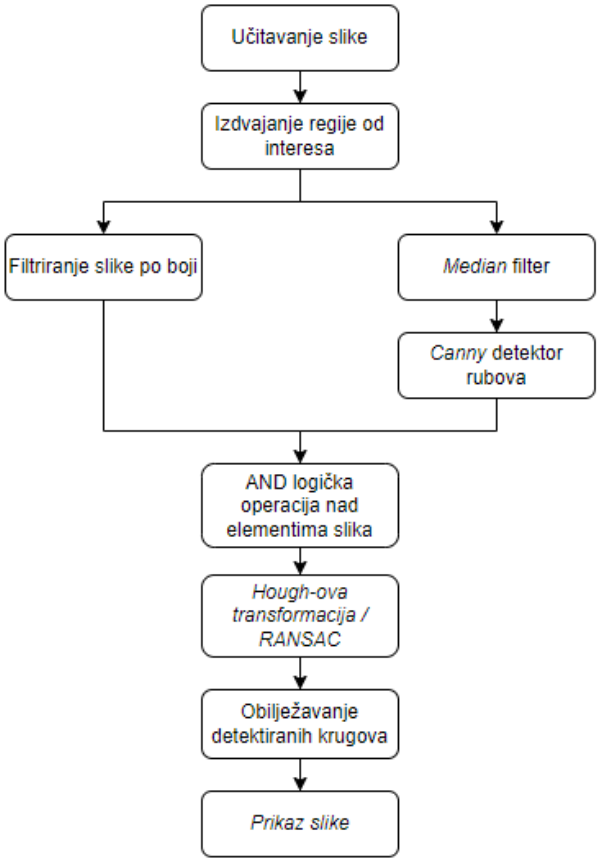
Kao što je ranije spomenuto RANSAC je iterativna metoda i broj iteracija će ovisiti o veličini skupa podataka, o broju točaka potrebnih za jednoznačno određivanje modela koji se smješta u podatke i o željenoj vjerojatnosti uspješnog pronalaska modela. Izrazom (3-4), gdje je k broj iteracija, p željena vjerojatnost uspješnog pronalaska modela, w omjer *inlier*-a i *outlier*-a i n broj točaka potrebnih za jednoznačno određivanje modela, moguće je izračunati potreban broj iteracija za dane parametre.

$$k = \frac{\log(1-p)}{\log(1-w^n)} \tag{3-4}$$

3.2. Koncept rješenja za detekciju krugova

Razvoj programskog rješenja za ovaj algoritam izvodio se u dva razvojna okruženja i dva programska jezika. Prvi prototip rješenja razvijen je u razvojnom okruženju *Visual Studio Code* u programskom jeziku *Python*, a drugi u programskom okruženju *Visual Studio* u C programskom jeziku. Za rješenje u *Python*-u korištene su funkcije iz biblioteke *OpenCV* koja je otvorenog koda i sadrži velik broj programskih funkcija za primjenu u računalnom vidu u stvarnom vremenu [11]. Funkcije iz ove biblioteke omogućile se izradu prvog koncepta rješenja u vrlo kratkom vremenu. Pošto je detekcija krugova izvedena s dva algoritma, postoje dva odvojena programa. Na slici 3.6. dan je prijedlog algoritma za detekciju krugova. Prvi korak je učitavanje slike. Učitana slika se

dalje obrađuje u dvije grane obrade. U lijevoj grani obrade slika se filtrira po boji kako bi se izbacili dijelovi slike koji ne sadrže korisne informacije. Propuštaju se boje koje se pojavljuju na okruglim objektima u prometu, kao što je crvena i plava na prometnim znakovima. U desnoj grani obrade se uklanjaju detalji i šum medijan filterom. Nakon toga se slika propušta kroz *Canny* visokopropusni filter čime se dobiva slika s istaknutim rubovima. Slike iz lijeve i desne grane obrade spajaju se u jednu pomoću logičkog operatora I. U jednom programu na pripremljenoj slici se detektiraju krugovi pomoću *Hough*-ove transformacije, a u drugom pomoću RANSAC algoritma. U konačnici se detektirani krugovi omeđuju graničnim pravokutnicima na slici s istaknutim rubovima.



Slika 3.6. Blok dijagram algoritma za detekciju krugova pomoću *Hough*-ove transformacije ili RANSAC algoritma

3.3. Implementacija predloženog rješenja na osobnom računalu u Python programskom jeziku

Slika 3.7. prikazuje kod u kojem se slika prvo učitava pomoću funkcije `imread()`. ROI se primjenjuje tako da se u ostatku programa obrađuje samo desna strana slike. Obrada slike se dalje dijeli u dvije grane obrade. U desnoj grani obrade slici se prvo uklanjaju detalji i šum pomoću

funkcije `medianBlur()`, kojoj se kao parametar predaje slika koja se filtrira i veličina kernela filtera. Kernel ili maska je matrica koja se koristi za zamućenje, izoštravanje, utiskivanje, detekciju rubova i još mnogo toga. Za slike u rezoluciji 1280x720 korišten je filter kernela veličine 11x11, za slike u rezoluciji 960x540 kernel veličine 10x10, a za slike u rezoluciji 640x360 kernel veličine 9x9. Zatim se provodi detekcija rubova na slici pomoću funkcije `Canny()` koja koristi *Canny* detektor rubova, a kao ulazne parametre prima filtriranu sliku te gornji i donji prag. U funkciji `Canny()` vrijednost praga može iznositi od 0 do 255. Vrijednosti praga koje su pokazale najbolje rezultate za odabrani set slika su 100 za donji prag i 200 za gornji. U lijevoj grani obrade, slika se filtrira po boji kako bi se uklonili dijelovi slike koji nisu od interesa. Prije filtriranja po boji potrebno je odrediti granice svih boja koje se žele propustiti na slici. Po *HSV* sustavu boja donja granica crvene boje za nijansu je 0, a gornja 20. Za plavu boju donja granica je 105, a gornja 135. Navedene granice boja su utvrđene eksperimentalno. Obje maske se zasebno primjenjuju na sliku funkcijom `inRange()` koje primaju prethodno definirane granice za svaku boju. Zatim se slike dobivene iz lijeve i desne grane obrade spajaju u jednu binarnu sliku logičkim operatorom ILI (engl. *OR*) pomoću funkcije `bitwise_or()`. Kako se ne bi izgubili dijelovi slike u procesu spajanja rezultata obrade lijeve i desne grane algoritma, potrebno je izvršiti proces dilatacije na binarnoj slici dobivenoj od filtriranja po boji. Proces dilatacije sve elemente slike na binarnoj slici proširuje kernelom koji se predaje funkciji `dilate()`, što je veći kernel elementi slike će se više proširiti. Sada su slike iz obje grane obrade spremne za spajanje u jednu binarnu sliku. Spajanje slike se izvršava pomoću I logičkog operatora pomoću funkcije `bitwise_and()`. Ta funkcija od dvije slike stvara jednu novu sliku tako da uzima svaki element od obje slike i na njih primjenjuje logički operator I. Ako su oba elementa različita od nule na to mjesto se upisuje jedinica u novoj slici. Tim postupkom u konačnoj slici ostaju samo elementi slike koji postoje u obje binarne slike. Lokacije rubova slike se spremaju u tekstualnu datoteku kako bi se olakšalo razvijanje rješenja u C programskom jeziku.

Linija ***Kod***

```
1:          img = cv.imread(image_string)
2:
3:          med = cv.medianBlur(img, 11)
4:          canny = cv.Canny(med, 100, 200)
5:
6:          hsv = cv.cvtColor(img, cv.COLOR_BGR2HSV)
7:          lower_blue = np.array([105, 140, 30], np.uint8)
8:          upper_blue = np.array([135, 255, 255], np.uint8)
9:
10:         lower_red = np.array([0, 140, 30], np.uint8)
11:         upper_red = np.array([20, 255, 255], np.uint8)
12:
```

```
13:       mask_blue = cv.inRange(hsv, lower_blue, upper_blue)
14:       mask_red = cv.inRange(hsv, lower_red, upper_red)
15:
16:       mask = cv.bitwise_or(mask_blue,mask_red)
17:       kernel = np.ones((11,11),np.uint8)
18:       mask = cv.dilate(mask, kernel)
19:
20:       con = cv.bitwise_and(mask, canny)
21:       for y in range(0, con.shape[0]):
22:           for x in range(0, con.shape[1]):
23:               if (con[y, x] != 0):
24:                   x_data.append(x)
25:                   y_data.append(y)
26:                   file.writelines(str(x) + '\n' + str(y) + '\n')
27:                   j+=1
28:       j = 0
29:       file.close()
```

Slika 3.7. Kôd pripreme slike u *Python* programskom jeziku

U jednom programu pripremljena slika se predaje funkciji `HoughCircles()` kojoj se predaju parametri kao što su omjer veličine slike i akumulacijskog polja, minimalna udaljenost između dva kruga koja se žele detektirati te minimalni i maksimalni polumjer kruga koji se želi detektirati. Detektirani krugovi se omeđuju pravokutnicima na binarnoj slici koja se u konačnici prikazuje pomoću funkcije `imshow()`.

U drugoj verziji programa krugovi se detektiraju pomoću RANSAC algoritma. Slika 3.6. prikazuje blok dijagram toka programa. Tok je isti kao u slučaju gdje je za detekciju krugova korištena *Hough*-ova transformacija, samo što se u ovom slučaju za detekciju krugova koristi RANSAC algoritam. RANSAC algoritam se izvodi pozivom funkcije `execute_ransac()`. Funkciji se predaju podaci o *x* i *y* koordinatama detektiranih rubova, broj iteracija, te prag koji određuje koliko *inlier*-i smiju biti udaljeni od modela kruga u mjeri jednog elementa slike.

Izrada koncepta programa za detekciju krugova u programskom jeziku *Python* dala je dobar uvid u to kako bi vlastito rješenje trebalo funkcionirati. Sljedeći korak je razvoj rješenja na osobnom računalu u programskom jeziku C.

3.4. Implementacija predloženog rješenja na osobnom računalu u C programskom jeziku

U ovom potpoglavlju dani su detalji algoritma za detekciju krugova u programskom okruženju *Visual Studio* u C programskom jeziku. Izrađena su dva rješenja gdje se u jednom krugovi detektiraju pomoću *Hough*-ove transformacije, a u drugom pomoću RANSAC algoritma. U oba rješenja koriste se rezultati pripreme slike od rješenja u *Python* programskom jeziku. Nakon

potrebne obrade slike u programskom jeziku *Python*, slijedi daljnji razvoj rješenja u C programskom jeziku. Umjesto učitavanja slike, u program se učitava tekstualna datoteka koja sadrži lokacije detektiranih rubova na slici u obliku koordinata *x, y* pri čemu je ishodište definirano u gornjem lijevom uglu. Jedina razlika između rješenja u ta dva programska jezika je u tome što su algoritmi za detekciju krugova (*Hough* i RANSAC) razdvojeni na nekoliko funkcija.

3.4.1. Detekcija krugova pomoću *Hough*-ove transformacije

Za učitavanje lokacija točaka iz datoteke koristi se funkcija `getCannyPointsLocation()` čija je deklaracija prikazana slikom 3.8. Funkcija za argument `fileName` prima ime datoteke iz koje će čitati koordinate detektiranih rubova. Povratni tip funkcije je pokazivač na prvi element polja tipa `Point`. `Point` je struktura koja sadrži informacije o *x* i *y* koordinati točke. Funkcija radi tako da prvo izbroji koliko linija sadrži tekstualna datoteka kako bi se odredio ukupan broj točaka. Zatim se alocira potrebna memorija u koju se upisuju podaci o točkama i u konačnici funkcija vraća pokazivač na tu memorijsku lokaciju.

Linija ***Kod***

```
1:      Point* getCannyPointsLocation(const char* fileName);
```

Slika 3.8. Deklaracija funkcije za dohvaćanje lokacija rubova iz tekstualne datoteke

Nakon dohvaćanja informacija o lokacijama rubova poziva se funkcija `executeHoughCircles()` čiji je prototip prikazan slikom 3.9. Funkcija kao argumente prima polje točaka `cannyPoints` koje predstavlja koordinate elemenata slike koji odgovaraju detektiranim rubovima na slici. Pošto krugovi na slikama najčešće nisu savršeno okrugli i potpuni, uveden je prag tolerancije koji je ovdje nazvan `threshold` i kreće se između 0 i 100, gdje je 0 maksimalna tolerancija i na toj vrijednosti bi bilo detektirano mnogo lažnih krugova, dok 100 na drugu stranu predstavlja najmanju toleranciju i na toj vrijednosti bi se detektirali samo potpuni i savršeno okrugli krugovi. Taj argument zapravo predstavlja varijablu koja se dijeli sa 100 i zatim množi s vrijednošću glasova koje bi imao savršeno okrugli i potpuni krug, nakon toga se ta vrijednost uspoređuje s vrijednostima u akumulacijskom polju i po tome zaključuje da li je to mjesto u polju kandidat za krug. Argument `minDistanceBetweenDetectedCircleCenters` određuje koliki je minimalno dozvoljeni razmak između centara detektiranih krugova. Kako bi se ubrzao proces pronalaženja krugova postoje argumenti `minRadius` i `maxRadius` koji određuju minimalni i maksimalni radijus koji se pretražuje. Ovo znatno ubrzava proces jer se traže samo krugovi u tom rasponu, a ne svi mogući koji mogu postojati na slici koja se obrađuje. Funkcija

započinje pozivom funkcije `createAccumulator()` koja alocira memoriju potrebnu za spremanje glasova u akumulacijsko polje. Zatim slijedi pozivanje funkcije `fillAccumulatorForRadii()` koja prolazi kroz detektirane rubove i puni akumulacijsko polje s glasovima. Nakon što se akumulacijsko polje napuni s glasovima slijedi traženje lokalnih maksimuma, odnosno mjesta u akumulacijskom polju gdje postoji najviše glasova. Ako postoji mjesto u akumulacijskom polju čija vrijednost prijelazi vrijednost praga na tom mjestu se nalazi kandidat za krug. Zbog samog oblika akumulacijskoj polja po lokaciji lokalnog maksimuma se može odrediti položaj i polumjer kruga na slici. Kada se odrede parametri kruga poziva se funkcija `circleAssembler()` kojoj se predaju parametri kruga nakon čega funkcija vraća strukturu tipa `Circle`. Taj krug se sprema u polje detektiranih krugova i algoritam nastavlja sve dok ne prođe kroz cijelo akumulacijsko polje. Struktura tipa `Circle` sadrži podatke o lokaciji središta kruga i o radijusu kruga.

Linija ***Kod***

```
1:      Circle* executeHoughCircles(Point* cannyPoints,
2:              int threshold,
3:              int minDistanceBetweenDetectedCircleCenters,
4:              int minRadius,
5:              int maxRadius);
```

Slika 3.9. Deklaracija funkcije za detekciju krugova pomoću *Hough*-ove transformacije

Funkcija `createAccumulator()`, čiji je prototip prikazan slikom 3.10., služi za alokaciju memorije za akumulacijsko polje. Argument `imageHeight` predstavlja visinu slike, a argument `imageWidth` širinu slike na kojoj se želi detektirati krug. Treći argument funkcije `numberOfRadii` određuje se razlikom između najvećeg i najmanjeg polumjera koji se pretražuje. Zadatak ove funkcije je alokacija trodimenzionalne matrice koja predstavlja akumulacijsko polje za *Hough*-ovu transformaciju.

Linija ***Kod***

```
1:      int*** createAccumulator(int imageHeight,
2:              int imageWidth,
3:              int numberOfRadii);
```

Slika 3.10. Deklaracija funkcije za alokaciju memorije za akumulacijsko polje

Slika 3.12. prikazuje deklaraciju funkcije `fillAccumulatorForRadii()` koja za argumente prima adresu akumulacijskog polja `accumulator`. Pošto se akumulacijsko polje puni s glasovima na temelju detektiranih rubova, funkcija prima i adresu `cannyPoints` u kojoj su

spremljene lokacije detektiranih rubova. Osim navedenih parametara postoje još dva argumenta koja su jako važna za punjenje akumulacijskog polja glasovima, a to su `minRadius` i `maxRadius`. Oni predstavljaju raspon radijusa krugova koji se žele detektirati. Funkcija za svaki element slike na kojem je detektiran rub upisuje glasove u akumulacijsko polje i to za svaki polumjer zasebno. Za svaki polumjer poziva se funkcija `drawCircleInMatrix()` koja u jednu zasebnu, pomoćnu matricu, upisuje glasove na mjesta odgovarajuća za taj radijus pomoću funkcije prikazane slikom 3.13. Veličina matrice ovisi o polumjeru kružnice za koju je potrebno upisati glasove. Primjer takve jedne matrice s glasovima za krug polumjera 10 prikazan je slikom 3.11. Pošto se radi o kružnici polumjera 10, dimenzije matrice moraju biti 21x21 kako bi se u matricu mogli upisati glasovi za tu veličinu kruga. Lokacije glasova u pomoćnoj matrici su računate tako da se za svaki element matrice 21x21 računa udaljenost od centra iste matrice. Centar ove matrice je u ovom slučaju na lokaciji [10,10]. Ako je zaokružena vrijednost udaljenosti ista polumjeru, na to mjesto se upisuje glas. Razlog zbog kojeg se prvo računaju lokacije glasova u pomoćnu matricu i nakon toga se prepisuju u glavno akumulacijsko polje, je da se izbjegne računanje lokacija glasova za svaki detektirani rub. Ovom metodom je potrebno samo jednom izračunati njihove lokacije za svaki polumjer. Sljedeći korak je kopiranje glasova iz pomoćne matrice koju puni funkcija `drawCircleInMatrix()` u glavno akumulacijsko polje. Svaki detektirani rub na slici ima svoju *x* i *y* koordinatu. Lokacija svakog ruba će se u akumulacijskom polju odraziti kao centar kruga na čijem se radijusu upisuju glasovi.

0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	1	1	0	0	0	0	0	0	0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0

Slika 3.11. Primjer popunjene pomoćne matrice za krug polumjera 10 pomoću funkcije `drawCircleInMatrix()`

Linija ***Kod***

```
1:        void fillAccumulatorForRadii(int*** accumulator,
2:               Point* cannyPoints,
3:               int minRadius,
4:               int maxRadius);
```

Slika 3.12. Deklaracija funkcije za upisivanje glasova u glavno akumulacijsko polje

Ranije spomenuta funkcija `drawCircleInMatrix()` služi za računanje lokacija glasova koje upisuje u pomoćnu matricu. Za argumente funkcija prima adresu pomoćne matrice `circleInMatrix` i veličinu radijusa za koje računa lokacije glasova `radius`. Kako bi pomoćna matrica zauzimala što manje mjesta i da bi pristup elementima bio lakši, glasovi se ne upisuju u matricu već se njihove lokacije zapisuju u polje točaka tipa `Point`. Funkcija za povratnu informaciju vraća broj elemenata u polju.

Linija ***Kod***

```
1:        int drawCircleInMatrix(Point* circleInMatrix, int radius);
```

Slika 3.13. Deklaracija funkcije za upisivanje glasova u pomoćnu matricu

3.4.2. Detekcija krugova pomoću RANSAC algoritma

Za učitavanje lokacija točaka iz datoteke koristi se funkcija `getCannyPointsLocation()` čija je deklaracija prikazana slikom 3.8. Nakon učitavanja lokacija elemenata slike slijedi pozivanje funkcije za detekciju krugova `executeRansac()`, čija je deklaracija prikazana slikom 3.14. Funkcija krugove detektira pomoću RANSAC algoritma koji je predstavljen u potpoglavlju 3.1.5. Prvi argument koji funkcija prima su lokacije detektiranih rubova `points`. Temeljna karakteristika RANSAC algoritma je iterativno pokušavanje procjene parametara modela na temelju svih podataka. Iz tog razloga funkciji je potrebno predati željeni broj ponavljanja algoritma, `iterations`. Pošto krugovi na slikama nisu savršeno okrugli, a često nisu ni kompletni, funkciji je potrebno predati dva argumenta koji će postaviti dva praga. Argument *accuracy* određuje prag koji pomaže u određivanju što je *inlier*, a što je *outlier*. Kao što je ranije spomenuto krugovi ne moraju biti kompletni kao što ni manji i veći krug neće imati jednak broj *inlier*-a, iz tog razloga postoji argument `threshold` koji predstavlja adaptivni prag, ovisan o polumjeru kruga, koji povećava ili smanjuje broj potrebnih *inlier*-a da bi se model kruga smatrao kandidatom. Kako bi se odredile optimalne vrijednosti ovog argumenta, napravljene su testne slike s krugovima poznatih polumjera. Nakon toga slijedi brojanje koliko svaki krug sadrži točaka. Kako bi se isti argument mogao primijeniti za sve veličine krugova, za svaki krug zasebno njegov broj

točaka je podijeljen s njegovim polumjerom. Kao rezultat se dobio broj točaka po polumjeru kruga, koji se na testnim krugovima kretao između 5 i 6 točaka po jedinici polumjera. Po dobivenim rezultatima može se zaključiti da bi se argument `threshold` trebao postaviti nešto ispod navedenih vrijednosti. Premali argument `threshold` rezultirat će s mnogo lažnih krugova, na drugu stranu, prevelik argument `threshold` rezultirat će detektiranjem samo savršeno okruglih i potpunih krugova. Kako bi se izbjeglo detektiranje krugova na istim mjestima funkcija prima argument `minDistanceBetweenDetectedCricleCenters` koji određuje koliki je minimalni razmak između centara detektiranih krugova. Argumenti `minRadius` i `maxRadius` određuju raspon polumjera krugova. Funkcija `executeRansac()` se sastoji od petlje koja se ponavlja željeni broj puta, prema argumentu `iterations`. U toj petlji za svako ponavljanje se nasumično odabiru tri točke. Odabiru se tri točke jer je to dovoljan broj točaka da se jednoznačno odrede parametri kruga. Odabrane točke predaju se funkciji `makeCircle()` koja izračunava parametre kruga i vraća ih kao strukturu `Circle` u glavnu petlju. U petlji se provjerava da li već postoji detektirani krug s istim parametrima, i ako ne postoji nastavlja se daljnja obrada, u suprotnom se ide u novu iteraciju. Kandidat kruga predaje se funkciji `evaluateCircle()` koja vraća broj *inlier*-a. Ako broj *inlier*-a prijelazi postavljeni prag kandidat se sprema kao detektirani krug.

Linija ***Kod***

```
1:        Circle* executeRansac(Point* points,
2:               int iterations,
3:               int minRadius,
4:               int accuracy,
5:               double threshold,
6:               int minDistanceBetweenDetectedCircleCenters,
7:               int minRadius,
8:               int maxRadius);
```

Slika 3.14. Deklaracija funkcije za detekciju krugova pomoću RANSAC algoritma

Slika 3.15. prikazuje prototip funkcije `makeCircle()` koja služi za izračun parametara kruga na temelju tri predane točke. Funkcija za argument prima polje od tri točke `points`. Tri točke se upisuju u tri jednadžbe kružnice i slijedi rješavanje tri jednadžbe s tri nepoznanice. Slika 3.16. prikazuje kod kojim je riješen ovaj matematički problem.

Linija ***Kod***

```
1:        Circle makeCircle(Point* points);
```

Slika 3.15. Deklaracija funkcije koja iz polja predanih točaka izračunava parametre kruga koji prolaze kroz te točke

Linija ***Kod***

```
1:        int x12 = threePoints[0].x - threePoints[1].x;
2:        int x13 = threePoints[0].x - threePoints[2].x;
3:
4:        int y12 = threePoints[0].y - threePoints[1].y;
5:        int y13 = threePoints[0].y - threePoints[2].y;
6:
7:        int y31 = threePoints[2].y - threePoints[0].y;
8:        int y21 = threePoints[1].y - threePoints[0].y;
9:
10:       int x31 = threePoints[2].x - threePoints[0].x;
11:       int x21 = threePoints[1].x - threePoints[0].x;
12:
13:       float sx13 = pow(threePoints[0].x, 2) - pow(threePoints[2].x, 2);
14:       float sy13 = pow(threePoints[0].y, 2) - pow(threePoints[2].y, 2);
15:       float sx21 = pow(threePoints[1].x, 2) - pow(threePoints[0].x, 2);
16:       float sy21 = pow(threePoints[1].y, 2) - pow(threePoints[0].y, 2);
17:
18:       float f = ((sx13) * (x12)+(sy13) * (x12)+(sx21) * (x13)+(sy21) *
17:            (x13)) / 2 * ((y31) * (x12)-(y21) * (x13));
18:       float g = ((sx13) * (y12)+(sy13) * (y12)+(sx21) * (y13)+(sy21) *
19:            (y13)) / 2 * ((x31) * (y12)-(x21) * (y13));
20:
21:       float c = -(threePoints[0].x * threePoints[0].x) - (threePoints[0].y
22:            * threePoints[0].y) - 2 * g * threePoints[0].x - 2 * f *
23:            threePoints[0].y;
24:
25:       circle.centerX = (int)(round(-g));
26:       circle.centerY = (int)(round(-f));
27:
28:       circle.radius = (int)round(sqrt(circle.centerX * circle.centerX +
29:       circle.centerY * circle.centerY - c));
```

Slika 3.16. Računanje parametara kružnice koja prolazi kroz tri zadane točke

Funkcija `evaluateCircle()` čiji je prototip prikazan slikom 3.17. računa broj *inlier*-a za zadani krug i zadane točke. Funkcija za argument prima krug `circle` za koji se želi izračunati broj *inlier*-a. Argument `points` predstavlja skup točaka iz kojih se provjerava da li je točka *inlier* ili *outlier* kao što je objašnjeno u 3.1.5. Zbroj svih *inlier*-a vraća se kao podatak u glavni program.

Linija ***Kod***

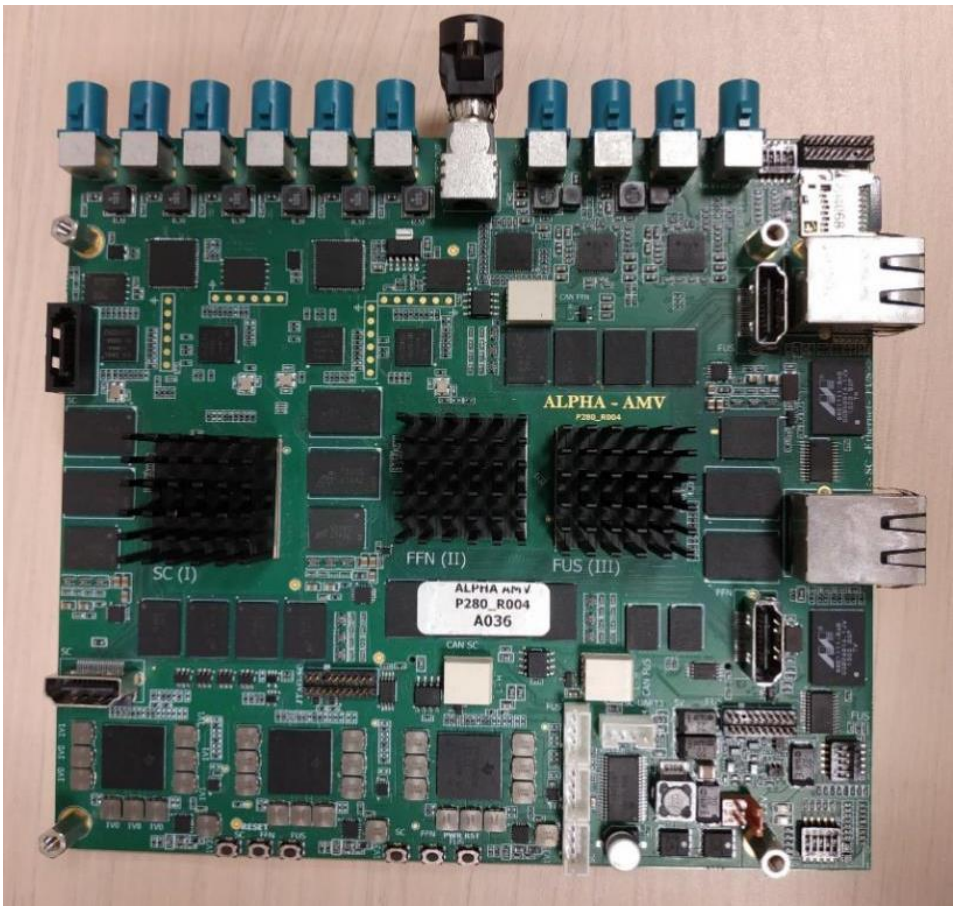
```
1:        int evaluateCircle(Circle circle, Point* points, int accuracy);
```

Slika 3.17. Deklaracija funkcije koja računa broj *inlier*-a za predani krug

Programski kod rješenja na osobnom računalu u C programskom jeziku za metode detekcije krugova pomoću *Hough*-ove transformacije i pomoću RANSAC algoritma nalazi se u elektroničkom prilogu P.3.1.

3.5. Opis korištene ADAS razvojne platforme

ADAS *Alpha* razvojna platforma pripada skupini ugradbenih računalnih sustava koji su namijenjeni za ugradnju u vozila. Ova razvojna platforma, prikazana slikom 3.18., sastoji se od tri različita *TDA2xx* sustava na čipu (engl. *System on Chip*) - (SoC): *SC* SoC, *FFN* SoC, *FUS* SoC [12]. *SC* se uglavnom koristi za prikaz okoline vozila, ili za bilo koju upotrebu koja zahtijeva veći broj kamera. Podržava do 6 širokokutnih kamera u isto vrijeme i kamere se moraju koristiti u parovima. Ovaj SoC podržava i Ethernet konekciju s drugim uređajima putem kojih može slati i/ili primati podatke. *FFN* SoC namijenjen je za uobičajene primjene kao što su prednji ili stražnji pogled iz vozila. Može raditi s bilo kojim brojem kamera, ali podržava maksimalno 4 uskokutne kamere. Ne podržava razmjenu podataka putem Ethernet komunikacije. *FUS* SoC je namijenjen za potrebe komunikacije između SoC-ova i rasterećenja. Ne podržava korištenje kamera, ali Ethernet komunikacija se može koristiti. Svi navedeni SoC-evi imaju [12]: 2 x A15 procesora, 2 x M4 procesora, 2 x DSP procesora, 4 x EVE procesora, 1.5 GB RAM memorije, Micro SD utor za karticu, HDMI utor, UART utor, JTAG utor, *Boot mode switch* – koristi se za odabir SoC-a, Ethernet konekciju s drugim SoC-evima.



Slika 3.18. ADAS *Alpha* razvojna ploča

Korištenje svih funkcija razvojne platforme i razvijanje algoritama omogućeno je višeprocesorskom platformom za razvoj softvera pod nazivom *VisionSDK*. To softversko okruženje omogućuje korisnicima stvaranje različitih tokova podataka za ADAS primjene kao što je snimanje videa, predobrada videa, algoritmi za analizu videa i prikaz video zapisa. *VisionSDK* temelji se na okviru nazvanom "*Links and Chains*", a korisničko aplikacijsko sučelje naziva se "*Link API*". Poveznica (engl. *Link*) je osnovni korak procesiranja u protoku video podataka. Poveznica se sastoji od niti (engl. *thread*) operacijskog sustava povezane s okvirom poruke. Budući da se svaka veza izvodi kao zasebna nit, poveznice se mogu izvoditi paralelno jedna drugoj. Poveznica implementira određeno sučelje koje omogućuje ostalim poveznicama izravnu razmjenu video okvira i/ili *bit stream*-ova, stoga nije potrebna intervencija korisničke aplikacije između razmjene video okvira. Aplikacijsko sučelje omogućuje korisniku stvaranje, upravljanje i povezivanje poveznica. Stvorena veza između poveznica naziva se lanac (engl. *chain*). Lanac se stvara na procesoru označenom kao *Host CPU*, koji je u slučaju korištenja *TDA2xx* obitelji ADAS SoC-a, *IPU-M4-0*. Svaka poveznica može imati ulaz koji predstavlja jedan ili više redova (engl. *queue*) koji se sastoje od jednog ili više kanala. Slično tome, svaka poveznica može imati i izlaz koji predstavlja jedan ili više redova sastavljenih od jednog ili više kanala. Na slici 3.19. prikazan je primjer jedne poveznice [12].

U *VisionSDK* potrebno je dodati algoritme koje će slučajevi upotrebe (engl. *Use case*) koristiti. Pod pojmom slučaj upotrebe, misli se na aplikaciju koja sadrži poveznice i odgovarajuće veze s ciljem obrade ulaznih video podataka i stvaranja nekog izlaza. Izlaz može, na primjer, biti prikazivanje obrađene slike s kamere na zaslon. Slučajevi upotrebe mogu biti različite aplikacije, poput pogleda iz vozila sprijeda, detekcija vozne trake i upozoravanje vozača i sl. Slučajevi korištenja izrađeni su od raznih poveznica koje se zajedno čine jedan veliki lanac.

Slika 3.19. Primjer poveznice [12]

3.6. Implementacija razvijenog rješenja na ADAS razvojnu platformu

U ovom potpoglavlju objašnjena je implementacija rješenja na ADAS platformu. Kako bi rješenje moglo raditi na razvojnoj platformi potrebno je u *VisionSDK* dodati novi slučaj upotrebe za rješenje iz potpoglavlja 3.4.1 i 3.4.2.

3.6.1. Izrada slučaja upotrebe i algoritma za detekciju krugova pomoću *Hough*-ove transformacije

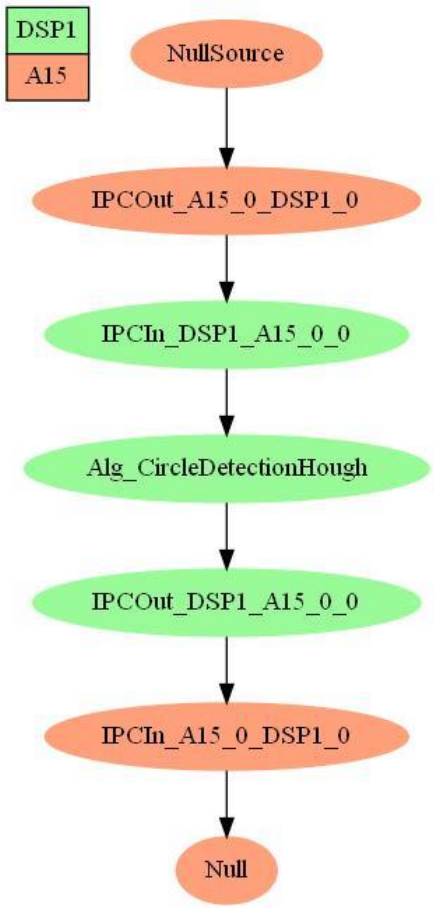
Prvi korak u izgradnji novog slučaja upotrebe je pisanje datoteke definicije (engl. *Usecase Definition File*). Definicijska datoteka slučaja upotrebe je tekstualna datoteka koja definira cijeli lanac slučaja upotrebe. Definicijska datoteka za slučaj upotrebe detekcije krugova pomoću *Hough*-ove transformacije prikazana je slikom 3.20.

```
UseCase: chains_circleDetectionHoughNull

NullSource (A15) -> Alg_CircleDetectionHough (DSP1) -> Null (A15)
```

Slika 3.20. Definicijska datoteka za slučaj upotrebe detekcije krugova pomoću Hough-ove transformacije

Prva linija definira naziv slučaja upotrebe koji bi trebao biti identičan nazivu koji se koristi u svim datotekama za taj slučaj upotrebe. Druga linija je glavni lanac poveznica korištenih u tom slučaju upotrebe. Veza *NullSource* koristi se za slanje video podataka s računala na razvojnu platformu. Kada se koristi *NullSource*, neobrađeni podaci šalju se preko Ethernet komunikacije, a zatim se podaci mogu poslati dalje kao redoviti jednokanalni red. Važno je napomenuti da se *NullSource* mora izvoditi na *A15* procesoru. Sljedeća poveznica, *Alg_CircleDetectionHough* koja predstavlja algoritamsku poveznicu, omogućuje pokretanje proizvoljnog koda unutar poveznice. U ovom slučaju pokreće se algoritam za detekciju krugova pomoću *Hough*-ove transformacije na procesoru DSP1. Zadnja poveznica *Null* služi za slanje video podataka Ethernet komunikacijom na računalo. Ovo može biti korisno ako se žele pohraniti rezultati na osobnom računalu za potrebe naknadne evaluacije efikasnosti predloženog algoritma. *Null* poveznica ima jednokanalni red za ulaz. Kao i *NullSource*, *Null* poveznica se mora pokretati na *A15* procesoru. Nakon pisanja definicijske datoteke slijedi generiranje slučaja upotrebe. Slučajevi upotrebe generiraju se pomoću priloženog alata *Usecase generator tool*. Alat uzima tekstualnu datoteku definicije slučaja upotrebe i generira određene datoteke. Alat generira datoteku koja predstavlja sliku slučaja gdje se lako uočava na kojem procesoru se izvršava pojedina poveznica i kako su međusobno povezane. Slika 3.21. prikazuje sliku generiranu za slučaj detekcije krugova pomoću *Hough*-ove transformacije. Osim

slike generira se i tekstualna definicija slike koja generalno ne sadrži informacije bitne programeru. Generiraju se i još dvije veoma važne datoteke *chains_circleDetectionHough_priv.c* i *chains_circleDetectionHough_priv.h*. To su datoteke čiji sadržaj ovisi o definicijskoj datoteci, a sadrže funkcije za inicijalizaciju i deinicijalizaciju cijelog sustava poveznica. Osim datoteka koje je stvorio generator, u direktoriju slučaja upotrebe se nalazi još nekoliko datoteka. Datoteka *cfg.mk*, definira koji će procesori i algoritmi biti uključeni u slučaj upotrebe. Zatim, u datoteci *SRC_FILES.MK* moraju biti definirane **.c* datoteke koje će biti uključene u izgradnju slike pri pokretanju *gmake* naredbe. Datoteka *chains_circleDetectionHough.c*, sadrži definicije svih korištenih poveznica u lancu.



Slika 3.21. Slika generirana *Usecase generator tool* alatom za slučaj detekcije krugova pomoću *Hough*-ove transformacije

Osim izrade novog slučaja upotrebe, u *VisionSDK* potrebno je dodati ranije spomenuti algoritam *Alg_CircleDetectionHough* kojeg poziva slučaj upotrebe. Kako bi se dodao novi algoritam potrebno je napraviti kopiju bilo kojeg algoritma iz *VisionSDK*. Kopiju algoritma potrebno je preurediti kako bi radila funkcionalnost koju želimo, u ovom slučaju detekciju krugova. Kao što je ranije spomenuto između poveznica u lancu razmjenjuju se video podaci.

Pomoću *NullSource* poveznice s računala se šalju binarne slike s istaknutim rubovima koje lancem dolaze do *Alg_CircleDetectionHough* poveznice. U algoritmu se prvo provjerava da li je slika u odgovarajućem podatkovnom formatu. Sve slike koje se obrađuju ovim algoritmom moraju imati format *SYSTEM_DF_YUV422I_YUYV*. Jedina razlika od rješenja u C programskom jeziku implementiranog na osobnom računalu je način na koji se dolazi do lokacija detektiranih rubova. Na razvojnoj platformi je jednostavno pristupati elementima slike, pa se iz tog razloga direktno sa slike uzimaju *x* i *y* koordinate istaknutih rubova. Na razvojnoj platformi su napravljene dvije nove funkcije za zapisivanje lokacija, pa tako umjesto funkcije `getCannyPointsLocation()` koja je na osobnom računalu dohvaćala lokacije rubova, ovdje postoje funkcije `countPoints()` i `fillPoints()`. Funkcija `countPoints()` prolazi kroz sve elemente slike i broji rubove na slici. Po tom broju se alocira memorija potrebna za spremanje lokacija svih elemenata slike koji pripadaju rubovima. Funkcija `fillPoints()` ponovno prolazi kroz cijelu sliku i zapisuje lokacije rubova u ranije alociranu memoriju.

3.6.2. Izrada slučaja upotrebe i algoritma za detekciju krugova pomoću RANSAC algoritma

Za rješenje detekcije krugova koji se temelji na RANSAC algoritmu potrebno je dodati novi slučaj upotrebe jer se koristi drugi algoritam. Prvo je potrebno napisati definicijsku datoteku iz koje će generator slučaja upotrebe generirati datoteke koje nedostaju. Definicijska datoteka korištena za slučaj upotrebe detekcije krugova pomoću RANSAC algoritma prikazana je slikom 3.22.

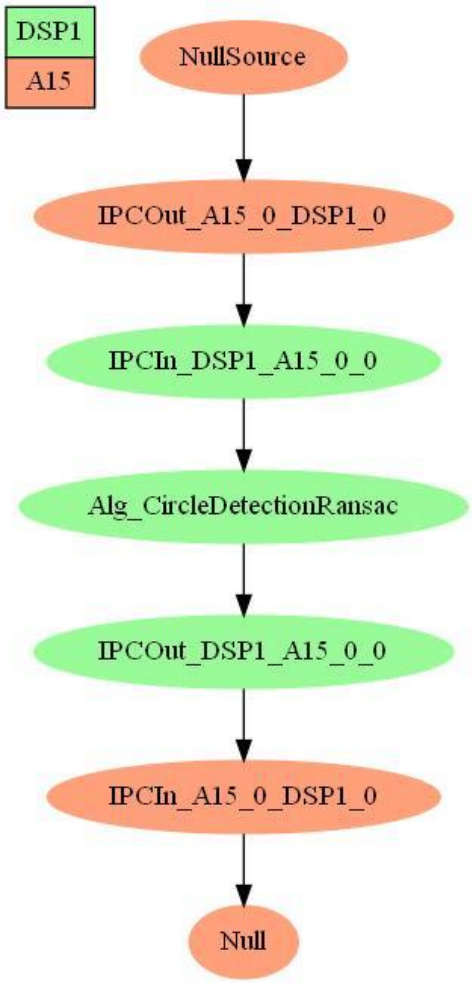
```
UseCase: chains_circleDetectionRansac

NullSource (A15) -> Alg_CircleDetectionRansac (DSP1) -> Null (A15)
```

Slika 3.22. Definicijska datoteka za slučaj upotrebe detekcije krugova pomoću RANSAC algoritma

Na temelju definicijske datoteke, *Usecase generator tool* stvara datoteke koje nedostaju kao i sliku 3.23. koja prikazuje tok podataka kroz poveznice u lancu. Isto kao i u prethodnom rješenju korištene su poveznice *NullSource* i *Null* za slanje i primanje slike. Jedina razlika je što se ovdje koristi druga poveznica algoritma koja za detekciju krugova koristi RANSAC algoritam, *Alg_CircleDetectionRansac*. Slanje i primanje slike obavlja procesor *A15* dok se algoritam pokreće na DSP1 procesoru.

Kao i s prethodnim algoritmom u *VisionSDK* je potrebno dodati novi algoritam *Alg_CircleDetectionRansac* koji poziva slučaj upotrebe. Algoritam je identičan onom razvijenom

na osobnom računalu, osim što se lokacije detektiranih rubova dohvaćaju iz slike kao i u detekciji krugova pomoću *Hough*-ove transformacije.



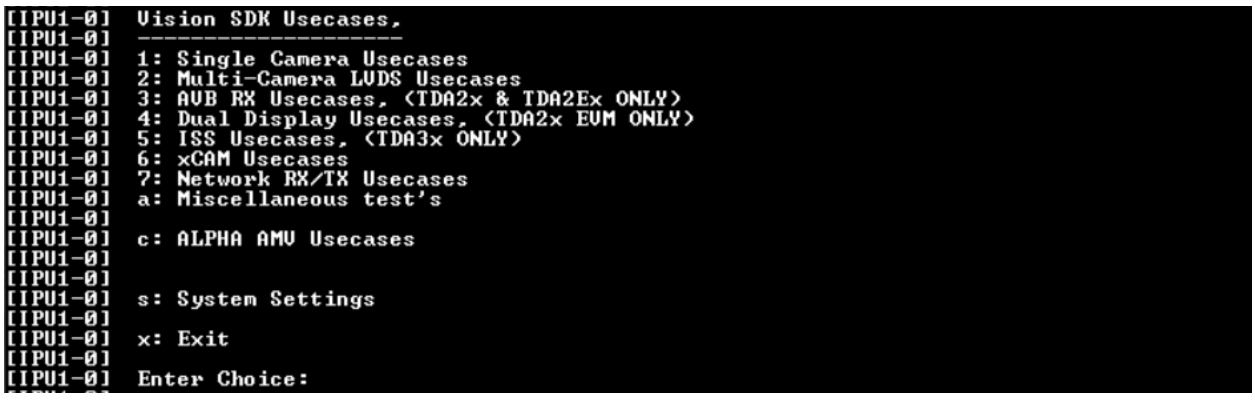
Slika 3.23. Slika generirana *Usecase generator tool* alatom za slučaj detekcije krugova pomoću RANSAC algoritma

3.7. Način pokretanja programskog rješenja na ADAS razvojnoj platformi

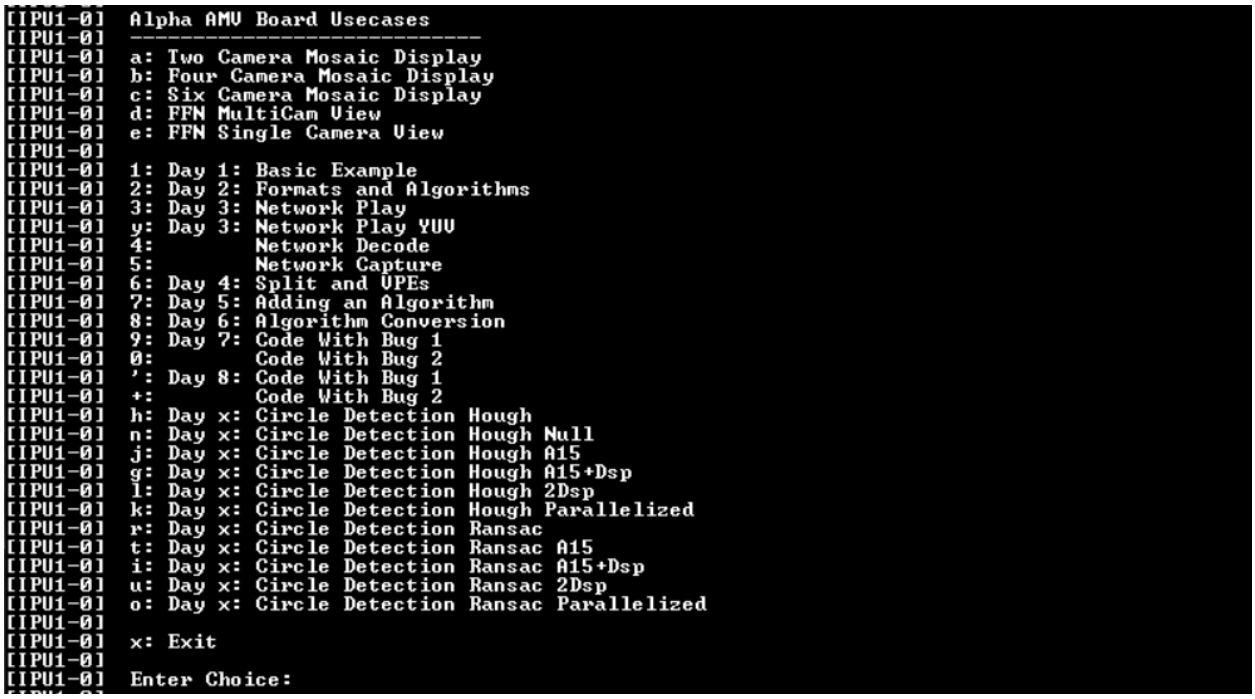
Nakon implementiranih slučaja upotrebe i algoritama slijedi izgradnja slike koja će se pokretati s memorijske kartice i omogućiti odabir i pokretanje izgrađenih slučajeva upotrebe. *VisionSDK* predstavlja jedan kompleksan sustav koji je sastavljen od više komponenti. To znači da se proces izgradnje mora izvoditi u nekoliko koraka. Kada se izvodi izgradnja sustava, važno je pozicionirati komandnu liniju u direktorij *<VSDK>\vision_sdk*. Prva naredba koja se poziva, *gmake -s -j depend*, obrađuje i spaja sve **.mk* datoteke. Ta naredba se poziva pri svakoj promjeni SoC-a i pri dodavanju novih slučaja upotrebe i algoritama. Sljedeća naredba koja se pokreće, *gmake -s -j sbl_sd*, generira *MLO* datoteku koja se mora pohraniti na memorijsku karticu. Ova naredba se pokreće samo kada se mijenja SoC. Treća naredba, *gmake -s -j*, je "glavna" naredba

koja se poziva. Traje najviše vremena i izvodi stvarni proces izgradnje sustava. Ako postoje greške u samom kodu, one će se pojaviti prilikom izvođenja ove naredbe. Zadnja naredba koja se izvodi, *gmake -s -j appimage*, služi za generiranje *Appimage* datoteke koju je zajedno s datotekom *MLO* potrebno spremiti na memorijsku karticu. Slično kao i prethodna naredba, mora se pokrenuti pri svakoj izgradnji [12].

Razvojna platforma komunicira s računalom pomoću serijske komunikacije, stoga je potrebno koristi *USB* na *UART* adapter za njihovo spajanje. Ovisno o tome koji SoC se koristi, adapter mora biti priključen u odgovarajući *UART* priključak na ploču. Za komunikaciju računala i razvojne platforme koristi se program *Tera Term*. U programu je potrebno postaviti serijsku komunikaciju s razvojnom platformom tako da se odabere u kojem utoru na računalu se nalazi adapter te kojom brzinom prijenosa će se odvijati komunikacija. Nakon što su se datoteke *Appimage* i *MLO* spremile na memorijsku karticu, ista se priključuje u odgovarajući priključak na ploči. Ploča se priključuje na napajanje i *VisionSDK* se pokreće. Kada se sustav pokrene, potrebno je odabrati koji slučaj upotrebe se želi pokrenuti. Izbor slučaja upotrebe prikazan je slikom 3.24., gdje se vidi izbor kategorija slučaja upotrebe. Potrebno je pokrenuti *ALPHA AMV Usecases* slučaj upotrebe odabirom tipke *c* na tipkovnici. Nakon toga će se prikazati slučajevi upotrebe koji su dodani, a to su slučajevi upotrebe za detekciju krugova pomoću *Hough*-ove transformacije i pomoću RANSAC algoritma. Kao što se može vidjeti na slici 3.25., za svaki algoritam postoji nekoliko slučaja upotrebe zbog potrebe pokretanja algoritma na raznim kombinacijama procesora.



Slika 3.24. *VisionSDK* slučajevi upotrebe



Slika 3.25. *Alpha AMV Board Usecases* slučajevi upotrebe

Nakon pokretanja željenog algoritma, započinje slanje video okvira putem Ethernet komunikacije i obrada odgovarajućim algoritmom za detekciju krugova.

4. TESTIRANJE RADA PREDLOŽENOG RJEŠENJA ZA DETEKCIJU KRUGOVA

U ovom poglavlju dani su rezultati testiranja razvijenog rješenja za detekciju krugova pomoću *Hough*-ove transformacije i RANSAC algoritma. Testiranje rješenja na ADAS razvojnoj platformi je provedeno uz različite kombinacije korištenih procesora na razvojnoj platformi. Za testiranje korišten je skup od 20 slika s kamere montirane na prednjoj strani vozila. Skup slika je preuzet iz baze podataka korištene u radu [13]. Originalne slike .jpg formata u rezoluciji 1628x1236 skalirane su na rezoluciju 1280x972 nakon čega je odrezan donji dio slike kako bi se postigla standardna rezolucija od 1280x720. Rješenje je testirano i na slikama manje rezolucije tako da su slike rezolucije 1280x720 dodatno skalirane na rezoluciju 960x540 i na 640x360. Odabrane su slike na kojima su objekti od interesa na desnoj polovici slike zbog lakše primjene ROI.

Testiranje je izvršeno slanjem slika u obliku video okvira s računala na razvojnu platformu. Detekcija se smatra točnom (engl. *True Positive* - TP) ako je detektirani krug u potpunosti unutar graničnog pravokutnika. Promašenom detekcijom (engl. *False Negative* - FN) se smatra kada algoritam ne pronađe krug koji postoji na slici. Detekcija je pogrešna (engl. *False Positive* – FP) kada algoritam pronađe krug koji ne postoji na slici. Pri izvršavanju algoritma prikupljani su podaci o TP, FN i FP detekcijama i brzini izvođenja. Brzina izvođenja mjerena je tako da se oduzela početna vremenska oznaka pokretanja algoritma od krajnje. Na temelju prikupljenih podataka o TP, FN i FP detekcijama izračunata je preciznost (engl. *accuracy*) i odziv (engl. *recall*). Preciznost predstavlja udio točno detektiranih krugova u skupu pozitivno detektiranih primjera, izražena je u postotcima, a računa se formulom (4-1) [14]. Odziv predstavlja udio točno detektiranih krugova u skupu svih pozitivnih primjera, izražen je u postotcima, a računa se formulom (4-2) [14].

$$P = \frac{TP}{TP + FP} \times 100 \tag{4-1}$$

$$R = \frac{TP}{TP + FN} \times 100 \tag{4-2}$$

4.1. Rezultati detekcije krugova na testnim slikama

U ovom potpoglavlju predstavit će se rezultati detekcije krugova na testnim slikama. Budući da nisu provedene nikakve aproksimacije korištenih algoritama sve tri implementacije (Python programski jezik, C programski jezik na osobnom računalu i C programski jezik na ADAS razvojnoj platformi) rješenja postižu jednake rezultate u pogledu performansi točnosti detekcije krugova. Tablica 4.1. prikazuje preciznost i odziv rješenja temeljenog na *Hough*-ovoj transformaciji i na RANSAC algoritmu na sve tri promatrane rezolucije. Prikazani rezultati predstavljaju preciznost i odziv rješenja na cijelom skupu slika, tj. u izračunu su se koristile zbrojene TP, FP i FN vrijednosti od svih 20 slika. Iz rezultata je vidljivo da preciznost i odziv ne opadaju znatno smanjivanjem rezolucije. To znači da se može koristiti manja rezolucija što je važno sa stajališta obrade i upotrebe u stvarnom vremenu. Iz tablice se može zaključiti da rješenje pomoću *Hough*-ove transformacije u usporedbi s rješenjem pomoću RANSAC algoritma ima veću preciznost i odziv na svim rezolucijama. Rješenje pomoću *Hough*-ove transformacije u testiranju nije imalo ni jednu FP detekciju dok je rješenje pomoću RANSAC algoritma imalo do maksimalno 4 FP detekcije na 20 slika. To se događa zbog toga što se kod RANSAC algoritma koristio manji prag koji određuje minimalni broj *inlier*-a za zadani polumjer da bi se model smatrao krugom jer algoritam rijetko odabere tri točke čija će kružnica točno prolaziti kroz sredinu elemenata slike u slici istaknutih rubova koji predstavljaju kružnicu. U najvećem broju slučaja algoritam odabere tri točke čija će kružnica malo odstupati od stvarnog položaja kružnice i zbog toga dolazi do gubitka *inlier*-a. Kod *Hough*-ove transformacije to nije slučaj jer se evaluiraju svi elementi slike istaknutih rubova i kod njega ne dolazi do gubitka glasova.

Tablica 4.1. Preciznost i odziv oba rješenja na različitim rezolucijama

Rezolucija slike	*Hough*-ova transformacija		RANSAC	
	Preciznost	Odziv	Preciznost	Odziv
1280x720	100%	96.3%	94.04%	80.74%
960x540	100%	100%	99.2%	85.93%
540x360	100%	85.19%	89.49%	81.48%

Slika 4.1. a) prikazuje dva primjera ulaznih slika, a slika 4.1. b) dva primjera izlaznih slika na kojima su označeni točno detektirani krugovi.

a) b)

Slika 4.1. Primjeri slika s uspješno detektiranim krugovima a) ulazna slika, b) izlazna slika sa zeleno označenim točno detektiranim krugovima

Slika 4.2. a) prikazuje dva primjera ulaznih slika, a slika 4.2. b) dva primjera izlaznih slika na kojima nisu uspješno detektirani krugovi.

a) b)

Slika 4.2. Primjeri slika s neuspješnim detekcijama krugova a) ulazna slika, b) izlazna slika bez detektiranih krugova

Slika 4.3. a) prikazuje dva primjera ulaznih slika, a slika 4.3. b) dva primjera izlaznih slika na kojima su označeni pogrešno detektirani krugovi.

a) b)

Slika 4.3. Primjeri slika s pogrešnim detekcijama krugova a) ulazna slika, b) izlazna slika sa zeleno označenim pogrešno detektiranim krugovima

Sve slike na kojima je provedena detekcija krugova nalaze se u elektroničkom prilogu P.4.1. priloženom uz ovaj rad.

4.2. Optimizacija predloženog rješenja i mjerenje vremena izvođenja

Nakon prikupljenih referentnih podataka o brzini izvođenja rješenja u C programskom jeziku na osobnom računalu slijedila je optimizacija koda. U rješenju u C programskom jeziku na osobnom računalu gdje se koristila *Hough*-ova transformacija, najveće unaprjeđenje performansi u pogledu vremena izvođenja algoritma dobilo se optimizacijom upisa glasova u akumulacijsko polje. U prvom su se rješenju računale pozicije glasova za svaki polumjer i za svaki element slike posebno. U optimiziranom rješenju su se pozicije glasova računale samo jednom za svaki polumjer. Nakon toga su se glasovi samo prepisivali iz pomoćne matricu u glavno akumulacijsko polje za svaki element binarne slike. U rješenju u C programskom jeziku na osobnom računalu gdje se koristio RANSAC algoritam najznačajnije unaprjeđenje brzine izvođenja algoritma ostvarilo se ograničavanjem područja u kojem se provjerava broj *inlier*-a. U drugom su se rješenju za nasumično odabran model kružnice provjeravali svi elementi binarne slike dok su se u

optimiziranom rješenju provjeravali samo oni elementi koji su unutar zamišljenog pravokutnika koji omeđuje model kružnice koji se provjerava.

Tablica 4.2. prikazuje brzine izvođenja rješenja prije i poslije optimizacije koda za oba rješenja na osobnom računalu. U drugom i četvrtom stupcu prikazano je vrijeme izvođenja algoritma prije optimizacije izraženo u sekundama. U trećem i petom stupcu prikazano je vrijeme izvođenja poslije optimizacije izraženo u milisekundama.

Tablica 4.2. Brzine izvođenja rješenja na osobnom računalu prije i poslije optimizacije

Broj slike	*Hough*-ova transformacija		RANSAC	
	Prije optimizacije [s]	Poslije optimizacije [ms]	Prije optimizacije [s]	Poslije optimizacije [ms]
1	151.47	1.34	65.21	0.333
2	84.2	1.12	102.34	0.037
3	219.73	1.49	49.28	0.937
4	250.91	1.6	78.91	1.523
5	213.85	1.46	124.38	1.17
6	335.54	1.98	94.83	0.862
7	212.44	1.49	89.41	0.478
8	169.18	1.29	67.85	0.301
9	182.47	1.32	134.97	0.409
10	151.82	1.22	73.41	0.362
11	232.41	1.56	97.27	0.402
12	473.12	2.37	106.6	1.387
13	190.9	1.38	101.13	0.551
14	161.89	1.25	85.71	0.291
15	157.92	1.28	98.82	0.286
16	239.71	1.56	83.57	0.414
17	215.26	1.44	116.58	0.513
18	109.26	1.07	97.2	0.185
19	127.59	1.14	93.86	0.383
20	136.25	1.15	79.1	0.576
Srednje vrijeme	200.8	1.43	92.02	0.57

Za usporedbu brzine izvođenja rješenja na osobnom računalu korištene su slike rezolucije 1280x720. Iz tablice se može vidjeti da se optimizacijom koda izvođenje rješenja na osobnom računalu značajno ubrzalo. Prosječno vrijeme trajanja algoritma prije i poslije optimizacije koda rješenja pomoću *Hough*-ove transformacije je dvostruko sporije od rješenja pomoću RANSAC algoritma. Razlog tome je čisto veća složenost *Hough*-ove transformacije u odnosu na RANSAC algoritam. RANSAC algoritam daje rješenje s određenom vjerojatnošću u odnosu na broj zadanih iteracija. Ako se odabere veći broj iteracija vjerojatnost pronalaska kruga se povećava, obrnuto smanjivanjem broja iteracija vjerojatnost pronalaska kruga se smanjuje. Broj optimalnog broja iteracija se računa formulom (3-4) u kojoj se pri računanju koristi željena vjerojatnost pronalaska

modela. U ovom radu se koristio broj iteracija koji se dobio na slici čiji krug ima najveću razliku *inlier*-a i *outlier*-a, a izračunati broj iteracija iznosi 30000.

Na osobnom računalu se radilo na optimizaciji koda. Na ADAS razvojnoj platformi provedene su dodatne optimizacije u pogledu paralelizacije poslova. Za paralelizaciju poslova korišteni su DSP1, DSP2 i A15 procesori s ADAS razvojne platforme. Prvo je rješenje testirano samo na jednom DSP procesoru, zatim na jednom A15 procesoru. Nakon toga se radila raspodjela posla na više procesora. Prvo se posao rasporedio na A15 i DSP procesor, zatim na DSP1 i DSP2 procesor i u konačnici na sva tri odnosno A15, DSP1 i DSP2. Kod rješenja gdje se koristila *Hough*-ova transformacija raspodjela se napravila tako da se na svakom procesoru izvodio algoritam koji traži krugove u određenom rasponu polumjera. Tako se na primjer kod korištenja A15, DSP1 i DSP2 procesora posao raspodijelio tako da su se na procesoru A15 pretraživali polumjeri od 10 do 98 elemenata slike, na procesoru DSP1 od 98 do 104 elemenata slike i na DSP2 procesoru od 104 do 110 elemenata slike. U rješenju gdje se koristio RANSAC algoritam posao se rasporedio tako da se broj iteracija podijelio na sve procesore. Kod slučaja gdje su korišteni procesori A15, DSP1 i DSP2 posao se rasporedio tako da je algoritam na A15 procesoru radio 28000 iteracija, na DSP1 procesoru 1000 iteracija i na DSP2 procesoru 1000 iteracija. Osim paralelizacije poslova na sve slike primijenjeno je područje interesa. Pošto se svi znakovi nalaze na desnoj strani slike za područje interesa je uzeta desna polovica slika. Osim rezolucije 1280x720 korištene su i rezolucije 960x540 i 640x360, kao kod testiranja efikasnosti detekcije.

U naredne dvije tablice, koje prikazuju performanse rješenja u pogledu brzine izvođenja, prvi stupac prikazuje rezoluciju slike, dok ostali stupci prikazuju prosječno vrijeme izvođenja rješenja na 20 slika u ranije navedenim kombinacijama procesora. Iz rezultata se može zaključiti da DSP procesor radi oko 7 puta sporije od A15 procesora. Najkraće vrijeme izvršavanja rješenja ima kombinacija A15, DSP1 i DSP2 procesora. Kod rješenja gdje se koristila *Hough*-ova transformacija vrijeme obrade je opadalo smanjenjem rezolucije slike što je očekivano jer postoji manji broj elemenata slike koji pripadaju istaknutim rubovima na slici i manje su dimenzije akumulacijskog polja. Međutim kod rješenja gdje se koristio RANSAC algoritam vrijeme trajanja algoritma je poraslo kod primjene manjih rezolucija slika. Uzrok ovog problema nije otkriven tijekom izrade rada jer bi vrijeme izvođenja trebalo opadati smanjenjem broja elemenata slike koji pripadaju istaknutim rubovima. Tablica 4.3. prikazuje prosječno vrijeme izvođenja rješenja gdje se koristila *Hough*-ova transformacija, a tablica 4.4. prikazuje prosječno vrijeme izvođenja rješenja gdje se koristio RANSAC algoritam. Dodavanjem procesora vrijeme izvođenja algoritma opada. Ali to ovisi o tome koji se procesor dodaje jer je A15 procesor oko 7 puta brži od DSP procesora.

Stoga su se najbolji rezultati postizali kada je u kombinaciji korišten A15 procesor. Zbog toga što se brzina obrade podataka na procesorima razlikuje, trebalo je pravilno rasporediti posao po procesorima. Tako se A15 procesoru uvijek dodjeljivalo 7 puta više posla nego DSP procesoru. Na primjer ako je kod RANSAC algoritma trebalo izvršiti 8000 iteracija, procesoru A15 bi se dodijelilo 7000 iteracija, a procesoru DSP 1000 iteracija.

Cilj razvijenih rješenja je rad na ugradbenoj platformi u automobilu uz zahtjev rada u stvarnom vremenu. Da bi sustav detekcije krugova radio u stvarnom vremenu treba postići brzinu od oko 10 FPS-a (engl. *Frames per second*). Rješenje gdje je korištena *Hough*-ova transformacija na ugradbenoj platformi najveća postignuta brzina je 1.22 FPS-a i to za rezoluciju slike 640x360 korištenjem A15, DSP1 i DSP2 procesora. Kod rješenja gdje se koristio RANSAC algoritam na ugradbenoj platformi najveća postignuta brzina je 2.1 FPS-a na rezoluciji od 1280x720 korištenjem samo A15 procesora. Ni jedno rješenje nije zadovoljilo uvjete za rad u stvarnom vremenu na ugradbenoj platformi. Rješenje na osobnom računalu gdje je korištena *Hough*-ova transformacija postiže brzinu od 700 FPS-a na rezoluciji 1280x720. Rješenje gdje je korišten RANSAC algoritam postiže brzinu od 1754 FPS-a također na rezoluciji 1280x720. Rješenja testirana na osobnom računalu zadovoljavaju uvijete za rad u stvarnom vremenu. Rješenje na osobnom računalu radi s punom većom brzinom zbog veće računalne moći osobnog računala. Iz rezultata se može zaključiti da je potrebna daljnja optimizacija rješenja kako bi sustav zadovoljio zahtjeve za rad u stvarnom vremenu na osobnom računalu.

Tablica 4.3. Prosječno vrijeme [ms] izvođenja rješenja detekcije krugova koje se temelji na *Hough*-ovoj transformaciji

Kombinacija procesora / Rezolucija slike	DSP	A15	A15 + DSP	DSP1 + DSP2	A15 + DSP1 + DSP2
1280x720	15748	2213	2082	8394	1878
960x540	8375	1483	1353	4648	1207
640x360	6138	1033	907	3530	819

Tablica 4.4. Prosječno vrijeme [ms] izvođenja rješenja detekcije krugova koje se temelji na RANSAC algoritmu

Kombinacija procesora / Rezolucija slike	DSP	A15	A15 + DSP	DSP1 + DSP2	A15 + DSP1 + DSP2
1280x720	9434	477	571	5291	606
960x540	11838	603	667	6873	741
640x360	19377	776	803	9034	890

5. ZAKLJUČAK

Okrugli objekti u prometu kao što su semafori i prometni znakovi vozačima pružaju informacije o uvjetima na cesti i mogućim opasnostima, kako bi vožnju učinili lakšom i sigurnijom. Prometni znakovi su izrađeni tako da se njihova boja i oblik bitno razlikuje od prirodnog okruženja kako bi ih vozači mogli lako uočiti i prepoznati. U radu su predložena dva rješenja za detekciju okruglih objekata u prometu koja koriste metode iz klasičnog računalnog vida. Rješenje koje koristi *Hough*-ovu transformaciju za detekciju okruglih objekata daje veću točnost detekcije u usporedbi s rješenjem koje koristi RANSAC algoritam, ali je nešto sporije. Prilikom evaluacije rješenje pomoću *Hough*-ove transformacije nije imalo ni jedan pogrešno detektirani krug na 20 slika, dok je rješenje pomoću RANSAC algoritma imalo 4 u najgorem slučaju. Osim evaluacije predloženog rješenja s obzirom na efikasnost detekcije, provedena je i evaluacija s obzirom na vrijeme izvođenja. Najveće unaprjeđenje performansi u pogledu vremena izvođenja postignuto je optimizacijom petlji u programu koje se ponavljaju puno puta. Daljnja optimizacija na razvojnoj platformi je još više poboljšala brzinu izvođenja algoritma raspoređivanjem posla na procesore razvojne platforme te smanjivanjem rezolucije slika.

LITERATURA

[1] G. Kiefer, M. Vahl, J. Sarcher, i M. Schaeferling, „A configurable architecture for the generalized hough transform applied to the analysis of huge aerial images and to traffic sign detection“, u *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, stu. 2016, str. 1–7. doi: 10.1109/ReConFig.2016.7857143.

[2] M. Swathi i K. V. Suresh, „Automatic traffic sign detection and recognition: A review“, u *2017 International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET)*, velj. 2017, str. 1–6. doi: 10.1109/ICAMMAET.2017.8186650.

[3] P. Yakimov i V. Fursov, „Traffic Signs Detection and tracking using modified Hough transform“, u *2015 12th International Joint Conference on e-Business and Telecommunications (ICETE)*, srp. 2015, sv. 05, str. 22–28.

[4] S. Huang, C. Gao, S. Meng, Q. Li, C. Chen, i C. Zhang, „Circular road sign detection and recognition based on hough transform“, u *2012 5th International Congress on Image and Signal Processing*, lis. 2012, str. 1214–1218. doi: 10.1109/CISP.2012.6469709.

[5] S. Houben, „A single target voting scheme for traffic sign detection“, u *2011 IEEE Intelligent Vehicles Symposium (IV)*, lip. 2011, str. 124–129. doi: 10.1109/IVS.2011.5940429.

[6] Department of Computer Science, Lady Doak College, Madurai, D. Hema, Dr. S. Kannan, i Department of Computer Applications, School of Information Technology, Madurai Kamaraj University, Madurai, „Interactive Color Image Segmentation using HSV Color Space“, *Sci. Technol. J.*, sv. 7, izd. 1, str. 37–41, sij. 2019, doi: 10.22232/stj.2019.07.01.05.

[7] K. Erdogan i N. Yilmaz, „Shifting Colors to Overcome not Realizing Objects Problem due to Color Vision Deficiency“, str. 5.

[8] S. Tomic, „Sveučilište u Zagrebu“, str. 26.

[9] Jun, „[CV] 6. Structure Extraction with Hough Transform (line, circle)“, *Medium*, stu. 24, 2020. https://medium.com/jun-devpblog/cv-6-structure-extraction-with-hough-transform-line-circle-aaf8be62f169 (pristupljeno srp. 12, 2021).

[10] R. C. B. Martin A. Fischler, „Random Sample Consensus: A Paradigm for Model Fitting with Apphcatlons to Image Analysis and Automated Cartography“, sv. 24, lip. 1981.

[11] „Home“, *OpenCV*. https://opencv.org/ (pristupljeno stu. 18, 2021).

[12] D. Vajak, „Alpha Board and VisionSDK - Reference Guide“, str. 91.

[13] M. Mathias, R. Timofte, R. Benenson, i L. Van Gool, „Traffic sign recognition — How far are we from the solution?“, u *The 2013 International Joint Conference on Neural Networks (IJCNN)*, Dallas, TX, USA, kol. 2013, str. 1–8. doi: 10.1109/IJCNN.2013.6707049.

[14] „How the Compute Accuracy For Object Detection tool works—ArcGIS Pro | Documentation“. https://pro.arcgis.com/en/pro-app/latest/tool-reference/image-analyst/how-compute-accuracy-for-object-detection-works.htm (pristupljeno stu. 28, 2021).

SAŽETAK

Ovaj rad se bavi metodama za detekciju krugova na slikama snimljenim kamerom montiranoj na prednjoj strani vozila, te implementacijom metoda na ugradbenu ADAS razvojnu platformu. Prvo je dana teorijska podloga za metode koje se koriste u ovom radu. Metode koje se u radu koriste za detekciju krugova su *Hough*-ova transformacija i RANSAC algoritam. U sljedećem koraku je izrađen koncept rješenja u *Python* programskom jeziku na osobnom računalu. Nakon toga rješenje je implementirao i optimizirano u C programskom jeziku na osobnom računalu. Optimizirano rješenje je zatim implementirano na ADAS razvojnu platformu također u C programskom jeziku. Na razvojnoj platformi je slijedila daljnja optimizacija rješenja u pogledu pravilne raspodjele poslova na dostupne procesore razvojne platforme. Nakon optimizacije rješenja na ugradbenoj platformi uspoređene su performanse dvije metode. Rezultati su pokazali da rješenje pomoću *Hough*-ove transformacije bolje obavlja zadatak detekcije krugova na slikama, ali je nešto sporije od onog gdje je za detekciju korišten RANSAC algoritam.

Ključne riječi: *detekcija krugova, ADAS, VisionSDK, Hough-ova transformacija, RANSAC*

CIRCLE DETECTION ON IMAGES OBTAINED FROM A CAR CAMERA USING THE ADAS DEVELOPMENT PLATFORM

ABSTRACT

This paper deals with methods for detecting circles in images captured by a camera mounted on the front of the vehicle and implementing methods on the built-in ADAS development platform. First, the theoretical basis for the methods used in this paper are given. The methods used in this paper to detect circles are the Hough transformation and the RANSAC algorithm. In the next step, the concept of the solution in the Python programming language on a personal computer is developed. After that, the solution was implemented and optimized in the C programming language on a personal computer. The optimized solution was then implemented on the ADAS development platform also in the C programming language. The solution was further optimized on development platform in terms of the proper distribution of work on the available processors of the development platform. After optimizing the solution on the development platform, the performance of the two methods was compared. The results showed that the solution using the Hough transformation performs the task of detecting circles in images better, but is somewhat slower than where the RANSAC algorithm was used for detection.

Key words: *circle detection, ADAS, VisionSDK, Hough transform, RANSAC*

ŽIVOTOPIS

Matej Brekalo rođen je 26. Kolovoza 1997. godine u Požegi. Od 2004. do 2008. pohađa Područnu školu Vidovci, zatim od 2008. do 2012. pohađa Osnovnu školu Antuna Kanižlića Požega. Godine 2012. upisuje Tehničku školu Požega, smjer Elektroenergetika, koju završava 2016. godine polaganjem državne mature. Iste godine upisuje preddiplomski sveučilišni studij Elektrotehnika na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku. Godine 2017. završava prvu godinu preddiplomskog sveučilišnog studija Elektrotehnike nakon čega mijenja studij i upisuje drugu godinu na preddiplomskom sveučilišnom studiju Računarstva. Godine 2019. završava preddiplomski sveučilišni studij i na istom fakultetu upisuje diplomski sveučilišni studij Automobilsko računarstvo i komunikacije.

_____

Potpis autora

PRILOZI

P.3.1. Programski kod rješenja na osobnom računalu u C programskom jeziku za metode detekcije krugova pomoću *Hough*-ove transformacije i pomoću RANSAC algoritma

P.4.1. Slike na kojima je provedena detekcija krugova (elektronički prilog)