# Detekcija otklona položaja kamere i korekcija položaja kamere za primjenu u ADAS algoritmima

**Sabo, Željko**



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

**Sveučilišni studij**

# DETEKCIJA OTKLONA POLOŽAJA KAMERE I KOREKCIJA POLOŽAJA KAMERE ZA PRIMJENU U ADAS ALGORITMIMA

**Diplomski rad**

**Željko Sabo**

**Osijek, 2020.**

# SADRŽAJ

# 1. UVOD

Moderna vozila opremljena su brojnim sustavima za pomoć vozaču u vožnji (engl. *Advanced Driver Assistance System* - ADAS). Za uspješan rad, napredni sustavi za pomoć u vožnji koriste brojne senzore kao što su radar, LiDAR (engl. *Light Detetction and Ranging*), ultrazvučne i infracrvene senzore, kamere i sl. Napredni sustavi za pomoć u vožnji pomoću odgovarajuće softverske podrške obrađuju podatke koje prikupljaju od senzora, te djeluju na upravljački sustav vozila ili upozoravaju vozača pri upravljanju vozilom.

Cilj naprednih sustava za pomoć u vožnji je da jednog dana omoguće autonomnu vožnju vozila, odnosno da omoguće vozilima samostalno kretanje kroz promet bez potrebe za asistencijom vozača. Kako bi se došlo do te razine autonomne vožnje, potrebno je usavršiti brojne sustave pomoći koji se već koriste u vozilima današnjice kao što je na primjer sustav za detekciju pješaka, semafora, znakova, vozne trake i sl. Svi navedeni sustavi se oslanjaju na obradu slike. Video kamere su pričvršćene na vozila i snimaju voznu traku i okolinu vozne trake, a napredni sustav za pomoć u vožnji zatim obrađuje dobiveni video, okvir po okvir (engl. *frame by frame*).

Iako je kamera dobro pričvršćena na vozilo, moguće je da s vremenom dođe do određenog otklona položaja odnosno pomaka pojedine kamere. U idealnom slučaju nije došlo do pomaka kamere i napredni sustav za pomoć u vožnji će dobivenu sliku obraditi i s dobivene slike uspjeti dohvatiti sve potrebne informacije, no ako je došlo do otklona kamere dolazi do problema. Jedan od primjera je detekcija vozne trake. Algoritmi koji se koriste pri detekciji vozne trake u obzir uzimaju točan položaj kamere, na način da obrađuju samo dio slike (engl. *Region of Interest* – RoI). Ako bi na primjer došlo do otklona kamere za određeni kut u lijevu ili desnu stranu, dio slike koji se obrađuje bio bi zahvaćen nastalim otklonom, te algoritam za detekciju vozne trake ne bi radio ispravno.

Kako bi se navedena situacija izbjegla koriste se algoritmi za detekciju položaja i otklona kamere u naprednim sustavima vožnje. Jedan od pristupa rješavanja navedenog problema je pomoću matrice homografije [1]. Ideja je, proračunom matrice homografije pomoću jednakih parova ključnih točaka na referentnoj slici, te testnoj slici snimljenom pomoću kamere koja je otklonjena iz referentnog položaja izvršiti kompenzaciju nastalog otklona kamere. Dobivenom matricom homografije nastoji se tako kompenzirati otklon kamere tijekom obrade okvira koji se dobivaju s kamerom koja je otklonjena od referentnog (tvorničkog) položaja. Matrica homografije predstavlja matricu dimenzije 3x3 s kojom kad se pomnože $x$ i $y$ koordinate elemenata slike se dobiju nove koordinate za taj element slike pomoću kojih je moguće nadomjestiti pomak do kojeg je došlo.

Nakon obrade opisanim postupkom, u rezultantnoj slici pojavljuju se elementi slike koji nemaju odgovarajuće vrijednost pa ih je potrebno nadomjestiti.

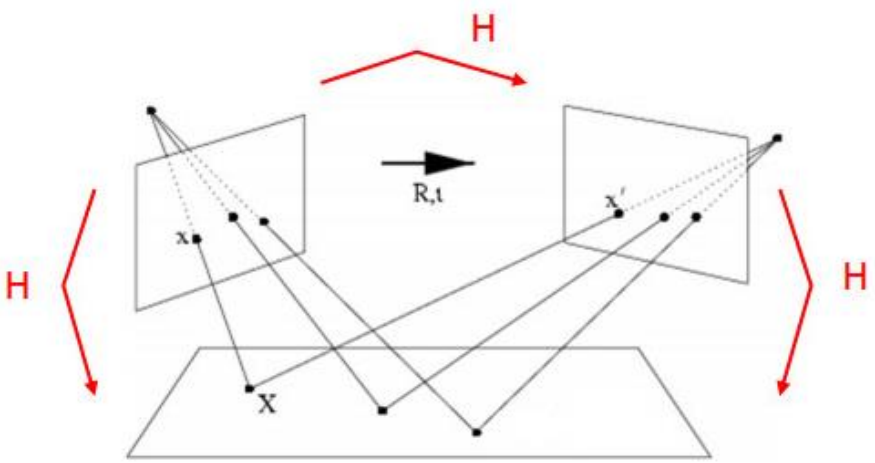
U ovom radu navedeni problem detekcije otklona položaja kamere je riješen pomoću matrice homografije i odgovarajućeg uzorka pomoću koje je razvijeno i testirano dobiveno programsko rješenje. Izvršena je implementacija programskog rješenja na osobnom računalu (engl. *Personal Computer* – PC) u *Python* [2] programskom jeziku uz pomoć *OpenCV* [3] i *NumPy* [4] biblioteke. Na temelju implementiranog rješenja na PC-u, izrađeno je rješenje navedenog problema za ADAS razvojnu ploču. Rješenje je implementirano u *C* programskom jeziku uz pomoć *VisionSDK* programskog okruženja.

Rad je strukturiran na sljedeći način. U drugom poglavlju dana je potrebna teorijska podloga, te se nalazi se pregled sličnih radova. Zatim je u trećem poglavlju predstavljena implementacija programskog rješenja kreiranog na osobnom računalu, te je predstavljeno programsko rješenje kreirano za ADAS razvojnu ploču. Četvrto poglavlje sadržava rezultate testiranja programskog rješenja implementiranog na osobnom računalu i programskog rješenja implementiranog na ADAS razvojnoj ploči, te se na kraju rada nalazi zaključak koji uokviruje sve navedeno i popis korištene literature.

# 2. PREGLED POSTOJEĆIH RJEŠENJA

Kao što je u prethodnom poglavlju spomenuto, iako su kamere prilikom proizvodnje vozila čvrsto postavljene na svoje položaje, moguće je da s vremenom dođe do određenog otklona njihova položaja, odnosno pomaka pojedine kamere. Isto tako prilikom proizvodnje vozila može se dogoditi da kamera nije pravilno pričvršćena ili da pravilno pričvršćivanje kamere nije moguće provesti u potpunosti. Također, vlasnik vozila može odlučiti naknadno ugraditi dodatne kamere za određenu primjenu koje se također zbog nesavršenosti ugradnje mogu s vremenom pomaknuti. Ako kamera odstupa od željenog položaja, potrebno je detektirati nastali otklon kamere, te ga na ispravan način kompenzirati.

Detekcija i kompenzacija otklona položaja kamere bazirana je na homografiji, koja opisuje projekcijsku geometriju dviju kamera i ravnine koju kamera snima, te omogućava preslikavanje točaka koje se nalaze unutar ravnine s jednog pogleda kamere na drugi. To je projekcijski odnos jer ovisi samo o presjeku ravnina s pravcima. Kraće rečeno, homografija opisuje transformaciju točaka između dvije ravnine. Slika 2.1. prikazuje ravnu površinu promatranu s dva različita položaja kamere. Točka *X* koja se nalazi na ravnoj površini je snimljena kamerom, te je na snimljenoj slici (okviru) preslikana u točku *x*. Nakon toga je napravljen pomak *R* unutar nekog vremena *t*, te se zatim kamera nalazi u položaju koji je od referentnog položaja otklonjen za *R*. Točka *x'* predstavlja točku *X* snimljenu s drugog položaja kamere, koji je od početnog položaja otklonjen za *R*.




Sl. 2.1. Ravna površina promatrana s dva različita položaja kamere.

Transformacija između dvije ravnine je opisana izrazom (2-1),

$$s \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \boldsymbol{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2-1}$$

gdje je *s* skalarna vrijednost, *x'* i *y'* novo proračunate koordinate za trenutni element slike koji se obrađuje. Matrica homografije je unutar izraza označena oznakom **H**, te su *x* i *y* koordinate trenutnog elementa slike koji se obrađuje.

## 2.1. POTREBNA TEORIJSKA PODLOGA ZA IZRADU PROGRAMSKOG RJEŠENJA ZA DETEKCIJA OTKLONA POLOŽAJA KAMERE I KOREKCIJU POLOŽAJA KAMERE

U ovom poglavlju predstavljena je teorijska podloga pomoću koje je implementirano programsko rješenje za detekciju otklona položaja kamere i korekciju položaja kamere na osobnom računalu i ADAS razvojnoj ploči.

### 2.1.1. Model i prostor boja

Model boja [5] i prostor boja [6] su dva pojma koji često dolaze zajedno, no što je zapravo model a što prostor boja?

Model boja predstavlja apstraktni matematički model koji omogućava opisivanje boja pomoću brojeva. Najčešće se za opisivanje boja pomoću apstraktnih matematičkih modela koriste tri do četiri komponente boje. Kada se apstraktni matematički model poveže s preciznim načinom tumačenja svake komponente kao rezultat se dobije set boja koji se naziva prostor boja.

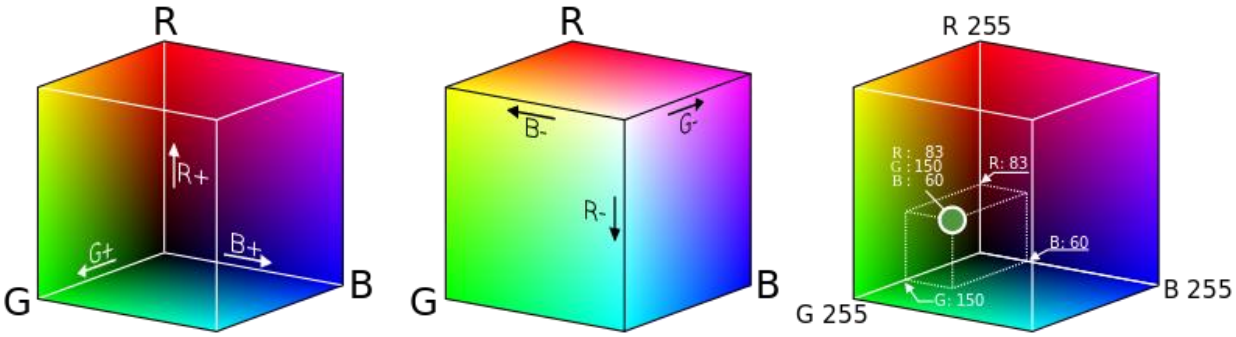
Prostor boja predstavlja specifičnu organizaciju boja koju ljudsko oko može vidjeti. Jedan od najpopularnijih prostora boja je RGB (*engl. Red, Green, Blue,* RGB) prostor boja [7].

### 2.1.2. RGB model boja

RGB model boja [8] predstavlja apstraktni matematički model koji omogućava opisivanje boja pomoću tri komponente boje, crvene (*R*), zelene (*G*) i plave (*B*) komponente boje. Svaka komponenta boje može poprimiti vrijednost u intervalu od 0 do 255. Ako je vrijednost komponente boje 0 tada nema doprinosa komponente boje, a ako je vrijednost komponente boje 255 tada se radi o maksimalnom doprinosu te komponente boje. Na primjer, ako želimo prikazati crvenu boju pomoću RGB modela boja, tada ćemo vrijednost *R* kanala postaviti na 255, a vrijednosti *G* i *B*

kanala na 0, te je time dobivena crvena boja, odnosno vrijednost crvene boje bi iznosila [255, 0, 0].

Sve boje osim crvene, zelene i plave boje u RGB modelu boja se mogu dobiti kombinacijom crvene, zelene i plave komponente boje. Kombinacijom vrijednosti *R* i *G* (crvenog i zelenog) kanala je moguće dobiti žutu boju, pa bi tako vrijednost žute boje bila [255, 255, 0]. Na slici 2.2. se nalazi kocka pomoću koje je prikazan RGB prostor boja.

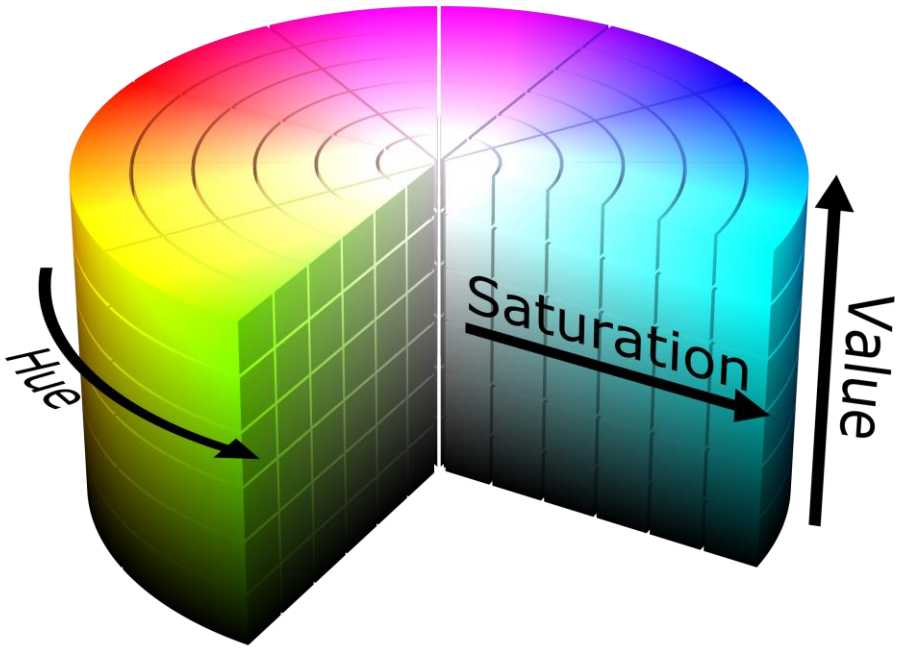



Sl. 2.2. RGB model boja.

### 2.1.3. HSV model boja

HSV (*engl. Hue, Saturation, Value* - HSV) model boja [9] je apstraktni matematički model boja koji se sastoji od tri komponente *H* (*engl. Hue*), *S* (*engl. Saturation*) i *V* (*engl. Value*), gdje *H* komponenta predstavlja nijansu boje, *S* komponenta predstavlja zasićenje boje, a *V* komponenta predstavlja svjetlinu boje. Kod HSV modela boja, *H* komponenta poprima vrijednosti u intervalu od 0 do 180 odnosno od 0 do 360 ako se radi o proširenom HSV modelu boja, dok su vrijednosti *S* i *V* kanala u intervalu od 0 do 255.

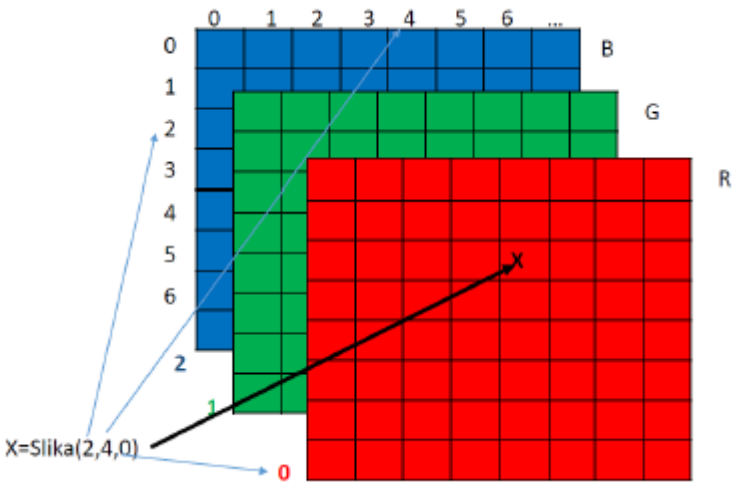
Definiranje boja u HSV modelu boja se znatno razlikuje od RGB modela boja. Kod RGB modela boja različite nijanse boje se dobiju pomoću različitih kombinacija crvene, zelene i plave boje, dok se kod HSV modela boja različite nijanse boje dobiju pomoću različitih vrijednosti *H* komponente. Primjerice. ako bi pomoću HSV modela boja želi prikazati crvena boja, vrijednost *H* komponente treba biti u intervalu od 0 do 20 ili u intervalu od 160 do 180 (ako se ne radi o proširenom HSV modelu boja), ili ako se radi o proširenom HSV modelu boja u intervalu od 0 do 40 ili od 320 do 360. Vrijednosti *S* i *V* komponente će najčešće imati vrijednost u intervalu od 80 do 120, ovisno o zasićenju i svjetlini boje. Crvena boja koja je u RGB modelu boja definirana kao [255, 0, 0] je u HSV modelu definirana kao [0, 100, 100]. Zapis boja pomoću HSV modela boja znatno olakšava filtriranje slike po boji, te se upravo iz tog razloga HSV model boja koristi u računalnom vidu. Na slici 2.3. se nalazi vizualna reprezentacija HSV modela boja.




Sl. 2.3. HSV model boja.

### 2.1.4. YUV model boja

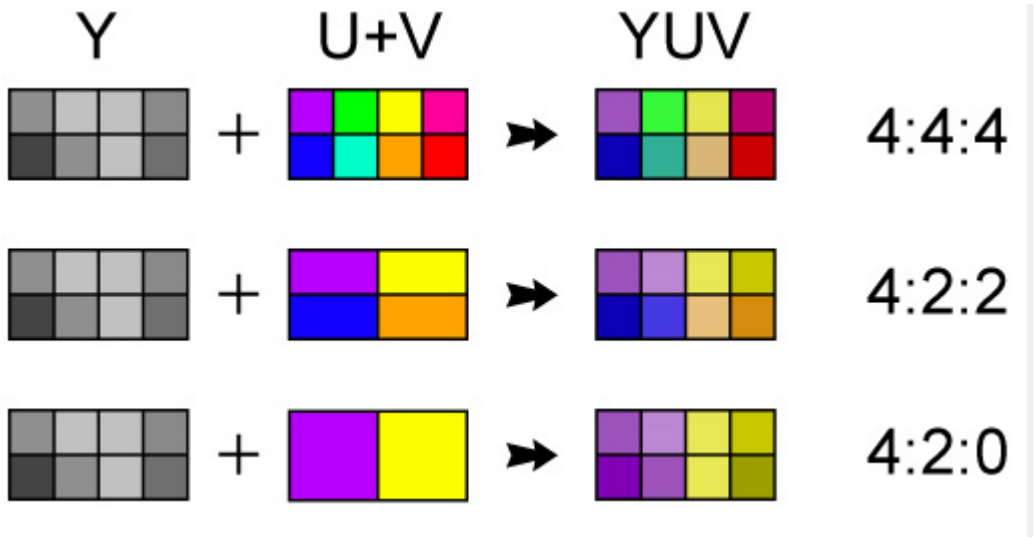
Kao što je prethodno u potpoglavlju 2.1.2. rečeno, RGB model boja se sastoji od tri kanala boje, *R*, *G* i *B*, odnosno crvenog, zelenog i plavog kanala. U računalu, RGB slika je pohranjena pomoću polja s tri dimenzije unutar kojeg se za svaki element RGB slike pohranjuju tri vrijednosti, po jedna za svaku komponentu boje. Na slici 2.4. se nalazi zapis RGB slike unutar polja s tri dimenzije.




Sl. 2.4. Zapis RGB slike pomoću polja s tri dimenzije.

U praksi, RGB slike su neprikladne za obradu podataka. Sve tri komponente boje kod RGB slike su podjednako važne i spremaju se u istoj rezoluciji. Na primjer, recimo da je potrebno potamniti RGB sliku koja se sastoji od 1280x720 elemenata slike jer je presvijetla. Kako bi se RGB slika potamnila potrebno je potamniti sva tri kanala boje, gdje se R, G i B kanali slike sastoje od 1280x720 elemenata slike. Vremenski je to jako zahtjevna operacija s obzirom na to da je potrebno tri puta proći kroz svaki element slike. Također, prilikom potamnjena RGB slike teško je reći koji kanal je potrebno potamniti više, a koji manje što RGB sliku čini nezgodnom za obradu. Kako bi se obrada slike ubrzala obično se koristi YUV [10] model boja, kojim se informacije o slici predstavljaju znatno efikasnije uzimajući u obzir ljudski vizualni sustav. YUV model boja se također kao i RGB model boja sastoji od tri komponente. *Y* komponenta sadrži informaciju o svjetlini, a *U* i *V* komponente informacije o boji. Brojni eksperimenti su pokazali da je ljudsko oko znatno osjetljivije na promjenu svjetline nego na promjenu boje. YUV model boja omogućava da se za svaki element slike pohrani potpuna količina informacije o svjetlini (*Y* kanal) i samo dio informacije o boji (*U* i *V* kanalima), odnosno da se informacija o boji ne pohranjuje za svaki element slike već za svaki drugi, četvrti i sl. Postupak gdje se pohranjuje sva informacija o svjetlini slike a samo dio informacije o boji se naziva poduzorkovanje boje u odnosu na svjetlinu.

Poduzorkovanjem boje u odnosu na svjetlinu odbacuje se dio informacije o boji za određene elemente slike, a slika osobi koja je promatra izgleda vizualno identično originalnoj slici kod koje su sve tri komponente boje zapisane bez gubitaka informacije. Na slici 2.5. su prikazana tri poduzorkovanja, 4:2:0, 4:2:2 i 4:4:4. Sva tri navedena poduzorkovanja pohranjuju potpuni sadržaj informacije o svjetlini za svaki element slike, ali ne i za boju. 4:4:4 poduzorkovanje pohranjuje potpuni sadržaj informacije o boji za svaki element slike, 4:2:2 poduzorkovanje pohranjuje za svaki drugi element slike, a 4:2:0 za svaki četvrti element slike.



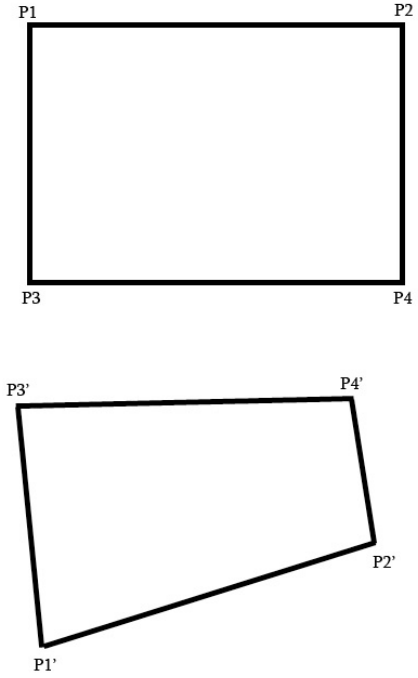

Sl. 2.5. Neki od načina poduzorkovanja boje u odnosu na svjetlinu.

### 2.1.5. Filtriranje slike po boji

Za proračun matrice homografije potrebno je riješiti sustav od osam jednadžbi s osam nepoznanica. Kako bi se sustav od osam jednadžbi s osam nepoznanica mogao postaviti, potrebno je s para slika, jedne referentne slike i jedne slike na kojoj je napravljen otklon, detektirati četiri točke na istim područjima slike. U ovom radu se navedene točke detektiraju pomoću filtriranja slike po boji koje se provodi pomoću HSV modela boja.

### 2.1.6. Matrica homografije

Matrica homografije predstavlja matricu dimenzije 3x3 koja opisuje preslikavanje slike točaka koje leže na ravnini scene s jednog pogleda kamere na drugi. To je projekcijski odnos jer ovisi samo o presjeku ravnina s pravcima. Na slici 2.6. dan je jednostavan primjer pravokutnika snimljenog iz dva različita položaja kamere. Gornji pravokutnik je snimljen kamerom koja se nalazi u referentnom položaju, dok je donji pravokutnik snimljen kamerom koja je otklonjena od referentnog položaja. Točke $P_1$, $P_2$, $P_3$ i $P4$ predstavljaju pozicije elemenata slike unutar referentne slike, dok točke $P'_1$, $P'_2$, $P'_3$, $P'_4$ predstavljaju iste elemente slike unutar slike snimljene nakon otklona kamere. Kako bi se matrica homografije mogla proračunati [11] potrebno je riješiti sustav dan izrazom (2-2).




Sl. 2.6. Primjer slike snimljen s dva različita položaja kamere.

$$\mathbf{PH} = 0 \tag{2-2}$$

Izraz (2-2) predstavlja sustav čijim rješavanjem se dobije matrica homografije **H**, dok matrica **P** predstavlja matricu dimenzije 9x9 koja se tvori pomoću matrice $\mathbf{p}_i$. Matrica $\mathbf{p}_i$ je dana u izrazu (2-3), te je dimenzije 2x9, a matrica **P** se tvori prema izrazu (2-4). Oznake $x_i$ i $y_i$ u izrazu (2-4), (2-6) predstavljaju koordinate točaka koje su odabrane u referentnoj slici, a oznake $x_i'$ i $y_i'$ koordinate istih elemenata slike u testnoj slici snimljene kamerom koja je otklonjena iz referentnog položaja

$$\mathbf{p_i} = \begin{bmatrix} -x_i & -y_i & -1 & 0 & 0 & 0 & x_i x'_i & y_i x'_i & x'_i \\ 0 & 0 & 0 & x_i & y_i & -1 & x_i y'_i & y_i y'_i & y'_i \end{bmatrix} \tag{2-3}$$

$$\mathbf{P} = \begin{bmatrix} \mathbf{p_1} \\ \mathbf{p_2} \\ \mathbf{p_3} \\ \mathbf{p_4} \end{bmatrix} \tag{2-4}$$

Nakon što su odabrani elementi slike unutar slike snimljene kamerom u referentnom položaju, te su pronađeni odgovarajući elementi unutar slike snimljene kamerom koja je otklonjena iz referentnog položaja, dohvaćaju se koordinate odabranih i pronađenih elemenata slike, te se formira sustav prema izrazu (2-5).

$$\mathbf{PH} = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x'_1 & y_1x'_1 & x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & -1 & x_iy'_1 & y_1y'_1 & y'_1 \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2x'_2 & y_2x'_2 & x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & -1 & x_2y'_2 & y_2y'_2 & y'_2 \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3x'_3 & y_3x'_3 & x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & -1 & x_3y'_3 & y_iy'_3 & y'_3 \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4x'_4 & y_4x'_4 & x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & -1 & x_4y'_4 & y_4y'_4 & y'_4 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = 0 \tag{2-5}$$

Kako bi se sustav (2-5) mogao riješiti potrebno je formirati matricu **P** prema izrazu (2-3) i (2-4). U izrazu (2-5) je vidljivo kako je matrica **P** dimenzije 8x9, a ne 9x9 kao što je ranije rečeno, te se kao zadnju redak matrice **P** uzima vektor dan u izrazu (2-6).

$$\mathbf{p_i} = [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1] \tag{2-6}$$

.

Nakon uvrštavanja vektora $\mathbf{p_i}$ danog izrazom izrazom (2-6) u matricu **P**, dobiva se sustav jednadžbi dan izrazom (2-7) čijim rješavanjem se dobiva matrica homografije **H**.

$$\mathbf{PH} = \begin{bmatrix} -x_1 & -y_1 & -1 & 0 & 0 & 0 & x_1x'_1 & y_1x'_1 & x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & -1 & x_1y'_1 & y_1y'_1 & y'_1 \\ -x_2 & -y_2 & -1 & 0 & 0 & 0 & x_2x'_2 & y_2x'_2 & x'_1 \\ 0 & 0 & 0 & x_2 & y_2 & -1 & x_2y'_2 & y_2y'_2 & y'_2 \\ -x_3 & -y_3 & -1 & 0 & 0 & 0 & x_3x'_3 & y_3x'_3 & x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & -1 & x_3y'_3 & y_3y'_3 & y'_3 \\ -x_4 & -y_4 & -1 & 0 & 0 & 0 & x_4x'_4 & y_4x'_4 & x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & -1 & x_4y'_4 & y_4y'_4 & y'_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \\ h_3 \\ h_4 \\ h_5 \\ h_6 \\ h_7 \\ h_8 \\ h_9 \end{bmatrix} = 0 \tag{2-7}$$

Vektor $\mathbf{p_i}$ dan izrazom (2-6) je moguće uvrstiti zato što zadnje rješenje matrice homografije, odnosno vrijednost rješenja $h_9$, uvijek ima vrijednost 1, te se na taj način dobiva nehomogeni sustav kojeg je moguće riješiti. Rješavanjem dobivenog sustava jednadžbi (2-7) dobiva se matrica **H** dimenzije 3x3 pomoću koje je moguće proračunati projicirane koordinate bilo

koje točke *p(x, y)* prema izrazu (2-1) koji opisuje transformaciju elemenata slike s jednog pogleda kamere na scenu na drugi.

### 2.1.7. Rekonstrukcija slike

Rekonstrukcija slike predstavlja postupak nadomještanja vrijednosti elemenata slike koji nedostaju u slici nakon njene obrade. Postoje razne vrste rekonstrukcije slike koje se koriste u praksi, a za rekonstrukciju slike u ovom radu korištena je interpolacija metodom najbližeg susjeda [12] koja na jednostavan način omogućava aproksimaciju vrijednosti praznih elemenata slike. Primjenom matrice homografije na neku sliku na kojoj se vrši kompenzacija otklona mogu se pojaviti elementi slike koji nemaju vrijednosti u rezultantnoj slici, te je te vrijednosti potrebno nadomjestiti. Interpolacija metodom najbližeg susjeda za svaki element slike koji nedostaje pronalazi pozicijski najbliži element slike u slici prije kompenzacije otklona, bez uzimanja u obzir ostalih susjednih elemenata slike. Uzmimo kao primjer sliku koja je snimljena kamerom otklonjenom za 1 centimetar od referentnog položaja. Na slici će se prvo izvršiti kompenzacija otklona položaja kamere od referentnog položaja pomoću matrice homografije. Kompenzacija otklona se vrši prema izrazu (2-1) odnosno koordinate elemenata slike se množe s matricom homografije, te se elementi slike upisuju unutar međuspremnika na novoproračunate koordinate. U slici unutar međuspremnika se pojavljuju elementi slike koji nemaju vrijednosti, te se te vrijednosti rekonstruiraju metodom najbližeg susjeda. Metoda funkcionira tako da se prvo pronađe inverz matrice homografije. Nakon toga se u slici na kojoj je izvršena kompenzacija otklona pronalaze elementi slike kojima nedostaju vrijednosti. Kada se pronađe element slike kojem nedostaje vrijednost tada se pomoću inverza matrice homografije proračunavaju koordinate tog elementa unutar slike prije kompenzacije otklona i uzima se vrijednost najbližeg elementa slike, odnosno vrši se aproksimacije vrijednosti pomoću slike prije kompenzacije otklona i inverzne matrice homografije.

## 2.2. Precizno praćenje objekata na temelju matrice homografije

U radu [13] je predstavljen okvir (engl. *framework*) za praćenje objekata koji može učinkovito pratiti 2-D afine pokrete objekata na slici, te je istražena metoda grubo-finog (engl. c*oarse-to-fine*) praćenja. Predstavljeno rješenje se sastoji od nekoliko koraka. Na početku se odabire područje slike unutar kojeg se nalazi objekt. Odabiranje područja slike unutar kojeg se nalazi objekt se vrši na temelju prvog okvira. Nakon toga se izvlače SIFT (engl. *scale-invariant feature transformation* - SIFT) značajne točke koje se koriste kao model. Regija od interesa je locirana pomoću PF (engl. *particle filter*) algoritma i sljedećeg okvira, te se uz pomoć najvećih vjerojatnosti regije interesa

vrši detektiranje značajnih SIFT točaka. Set značajni točki između modela i najveće regije interesa dobiven je pomoću SIFT transformacije, te se nakon toga vrši procjena matrice homografije pomoću RANSAC (engl. *RANdom SAmple Consensus*) algoritma. Kao krajnji rezultat cjelokupnog postupka dobivena je lokacija i pozicija objekta unutar slike. Testiranje je provedeno na stolnom računalu unutar *MATLAB* simulacijskog okruženja, te je testiranje pokazalo da opisana metoda može precizno pratiti složene pomake objekta odnosno Rubikove kocke, koja je korištena kao objekt koji će se pratiti. Kreirano rješenje je pokazalo izrazito veliku točnost i preciznost, no objekti koji se prate ne smiju biti fluidi, već čvrsti objekti, te je to najveća mana predstavljenog rješenja.

## 2.3. Linearno rješavanje matrice beskonačne homografije iz epipola

Beskonačna matrica homografije igra važnu ulogu u afinoj rekonstrukciji. Afina rekonstrukcija predstavlja rekonstrukciju koja ne ovisi euklidskim transformacijama. Dolazi do novog ograničenja između homografije beskonačne ravnine i epipola. Sukladno novom ograničenju, matricu beskonačne homografije moguće je linearno riješiti i dobiti afinu rekonstrukciju pomoću triangulacijskog principa. Programsko rješenje u radu [14] je podijeljeno u šest koraka. Prvi korak je detekcija ključnih točaka sa slike. Nakon detekcije ključnih točaka, vrši se usklađivanje ključnih točaka pomoću RANSAC metode. Poslije pridruživanja ključnih točaka pomoću RANSAC metode vrši se proračun fundamentalne matrice pomoću algoritma s više točaka (engl. *multi-point algorithm*) i metode najmanjih kvadrata. U četvrtom koraku se vrši proračun matrice homografije, te se nakon toga kao korak pet radi izračun apsolutnog konusa i dobivaju se intristični parametri kamere pomoću *Cholesky*-jeve metode. Zadnji korak u algoritmu je dodavanje *Gaussian*-ovog šuma. Testiranje je provedeno pomoću kamere rezolucije 600x480 gdje su snimljene dvije slike. Kao rezultat dobiveni su intrinsični parametri kamere s različitim razinama *Gaussian*-ovog šuma, te slika afine rekonstrukcije. Beskonačna matrica homografije je uspješno dobivena linearnim pristupom, a afina rekonstrukcija je uspješno dobivena pomoću triangulacijskog principa. Postignuti rezultati afine rekonsturkcije nad korištenim skupom podataka ukazuju kako je kreirano programsko rješenje praktično i realistično.

## 2.4. Brza metoda mozaika iz zraka bazirana na ORB metodi i matrici homografije

Mozaik slike (panoramske slike) imaju jednu od važnijih primjena u okviru računalnog vida. Najčešće se primjenjuju kako bi se više slika spojilo u jednu sliku velike rezolucije i velikog kuta gledanja. Rad [15] pokušava riješiti problem male brzine i male točnosti izvlačenja značajki iz

slika, te je predložena nova metoda temeljena na ORB (engl. *Oriented FAST and Rotated* - BRIEF) metodi i matrici homografije. Predloženo rješenje kao prvi korak izdvaja značajke s ulaznih slika. Nakon toga se koristi ORB metoda za spajanje slika koja spajanje slika radi pomoću prethodno izdvojenih značajki s ulaznih slika. Zatim uz pomoć mjera sličnosti, proces uparivanja pronalazi odgovarajuće značajke proračunavanjem *Hamming*-ove udaljenosti, te se vrši procjena transformacije ulaznih slika na temelju podudarnih parova i na samom kraju vrši se postupak miješanja (engl. *blending*) pomoću kojeg se otklanjaju nepravilnosti koje su nastale prilikom spajanja slika. Rješenje je implementirano pomoću korištenja biblioteke *OpenCV* verzije 3.4.1 i operacijskog sustava *Windows*. Predložena metoda je testirana na skupu slika koje su prikupljene iz zraka pomoću dronova kamerom rezolucije 640x480 elemenata slike. Kako bi se rezultati korištenja ORB metode mogli adekvatno evaluirati, korištene su i još neke metode poput SIFT i SURF (engl. *Speeded Up Robust Features*). Prilikom testiranja predloženog rješenja, korištena su tri različita algoritma, ORB, SIFT i SURF. Budući da je u radu naglasak na brzini, najbolje rezultate je pokazala ORB metoda koja je u 0.301 sekundi detektirala 500 značajnih od kojih je 68 točki dobro upareno. SURF metoda se je pokazala kao druga najbrža metoda koja je u prosjeku detektirala 908 točki značaja od kojih je njih 363 uspješno upareno. Najlošijom, odnosno najsporijom metodom se pokazala SIFT metoda, koja je u prosjeku detektirala 2285 značajnih točki od koji je dobro upareno njih 1920, te joj je bilo potrebno 1.949 sekundi. Eksperimentalni rezultati pokazali su da ORB metoda može učinkovito ubrzati detekciju značajki, te da istodobno podudaranje više slika može značajno smanjiti kumulativnu pogrešku.

## 2.5. Otkrivanje planarne homografije u paru slika

U radu [16] se predlaže algoritam koji vrši detekciju planarne homografije pomoću ne kalibriranih slika. Detekcija se vrši pomoću RANSAC sheme koja se temelji na linearnom proračunu elemenata matrice homografije pomoću četiri točke. Predloženi algoritam sastoji se od nekoliko koraka, prvi korak je detekcija uglova na paru slika. Nakon toga, vrši se normalizacija korelacija varijance i prikupljanje parova s dovoljno visokim rezultatom korelacije. Zatim se odabiru četiri točke iz ukupnog seta točaka koji je na raspolaganju, te se vrši proračun homografije. Parovi koji se podudaraju s homografijom se odabiru, te se prethodna dva koraka ponavljaju sve dok se ne skupi dovoljan broj parova koji se podudaraju s homografijom. Na samom kraju se vrši proračun nove matrice homografije pomoću svih parova koji se podudaraju s prethodno izračunatom matricom homografije. Predloženi algoritam testiran je na setu slika. Rezultati testiranja su pokazali kako algoritam može efikasno detektirati vidljive planarne strukture.

Testiranje algoritma je također pokazalo da bi algoritam mogao biti korišten na setovima koji su zahvaćeni raznim smetnjama kao što je na primjer šum.

## 2.6. Kombiniranje konvolucijske neuronske mreže i fotometrijskog usavršavanja za preciznu procjenu homografije

Procjena homografije najčešće se odnosi na proračun matrice dimenzije 3x3 koja preslikava točke između dvije slike planarne (ravninske) scene ili dvije slike snimljene na istoj lokaciji. U radu [17] je predložen način proračuna matrice homografije pomoću hibridnog okvira *HomoNetComb* koji uključuje metodu dubokog učenja. Konkretno, lagana (engl. *lightweight*) konvolucijska neuronska mreža osmišljena je za izračunavanje početne procjene homografije, gdje se mreža uči od kraja do kraja, koristeći veliku količinu parova slika generiranih iz javno dostupnog skupa podataka. Rješenje koje je predloženo u radu kao ulazne parametre koristi par sivih slika i maksimalno odstupanje između četiri kuta (ruba) između slika, a kao rezultat na izlazu se dobije matrica homografije. Tijek izvođenja programskog rješenja je sljedeći:

1. Ulazne slike se skaliraju na 32x32 elemenata slike.
2. Računa se relativni vektor pomaka.
3. Računa se vektor apsolutnog pomaka slike.
4. Računa se inicijalna matrica homografije.
5. Postavljaju je parametri završetka za fotometrijsko pročišćivanje.
6. Kreira se prazna siva slika dimenzije 32x32 elementa slike gdje su vrijednosti svih elemenata slike 1.
7. Inicijalizira se iteracijski indeks i njegova vrijednost se postavlja na 0.
8. Primjenjuje se iskrivljavanje (engl. *Warp*) slike s matricom homografije.
9. Računa se fotometrijski gubitak.
10. Vrši se ažuriranje matrice homografije pomoću metode gradijentnog spusta.
11. Računa se smanjenje gubitaka tijekom zadnje iteracije.
12. Vrijednost iteracijskog indeksa se povećava za 1.
13. Vrši se ponavljanje svih koraka od koraka 8. sve dok se ne zadovolji kriterij zaustavljanja.

Kako bi se provelo testiranje predloženog algoritma, korišten je set slika koji se sastoji od 200 000 parova slika i stolno računalo s *Intel(R) Core(TM) i7-7700HQ* procesorom, *NVIDIA GeForce GTX 1060* i 16 GB radne memorije.

Rezultati dobiveni tijekom testiranja pokazuju kako je pomoću konvolucijske neuronske mreže je moguće predvidjeti matricu homografije čak i kad na sceni nedostaju neke od tekstura, no točnost rješenja bi se mogla dodatno poboljšati eksplicitnim geometrijskim modeliranjem i optimizacijom.

# 3. PRIJEDLOG ALGORITMA ZA DETEKCIJU OTKLONA POLOŽAJA KAMERE I KOREKCIJU POLOŽAJA KAMERE

U ovom poglavlju dan je kratak opis korištenih tehnologija, te je predstavljena implementacija algoritma na osobnom računalu, kao i samo programsko rješenje za ADAS razvojnu ploču. Kao i što je ranije spomenuto, moderna vozila opremljena su brojnim senzorima koji u kombinaciji s naprednim sustavom za pomoć u vožnji pomoću odgovarajuće softverske podrške obrađuju podatke koje prikupljaju od senzora, te djeluju na upravljački sustav vozila ili upozoravaju vozača pri upravljanju vozilom. Jedan od senzora koji se koriste su i video kamere koje su pričvršćene na vozila i snimaju voznu traku i okolinu vozne trake, a napredni sustav za pomoć u vožnji zatim obrađuje dobiveni video signal, okvir po okvir. Iako su video kamere dobro pričvršćene na vozilo, moguće je da s vremenom dođe do određenog otklona položaja, odnosno pomaka pojedine kamere. Često se radi o jako mali otklonima koje je teško kompenzirati ponovnim namještanjem položaja kamere, te je iz tog razloga predloženo programsko rješenje navedenog problema.

Programsko rješenje je bazirano na matrici homografije. Ideja rješenja je da se nakon montiranja kamere vozilo pozicionira na točno određenu lokaciju te se snimi referentna slika određenog uzorka na kojem je lagano pronaći četiri značajne točke. Zatim se, u slučaju otklona kamere, vozilo pozicionira na istu lokaciju te se ponovo snimi isti uzorak. Na obje slike istog uzorka moguće je detektirati značajne točke, te pomoću njihovih pozicija izračunati matricu homografije koja se može koristiti za ispravljanje slike koja se dobiva pomoću kamere kao da nije došlo do otklona kamere. Budući da se prilikom kompenzacije otklona kamere u ispravljenoj slici pojavljuju prazni elementi slike, potrebno je nadomjestiti elemente slike, odnosno napraviti rekonstrukciju praznih elemenata slike. Za rekonstrukciju se koristi interpolacijska metoda najbližeg susjeda, koja pomoću koordinata praznih elemenata u ispravljenoj slici i inverza matrice homografije pronalazi elemente slike koji nedostaju u slici prije kompenzacije otklona. U nastavku je dan kratki opis korištenih tehnologija, te implementacija programskog rješenja kreiranog na osobnom računalu i prijedlog programskog rješenja razvijenog za realnu ADAS razvojnu ploču.

## 3.1. Pregled korištenih tehnologija za izradu programskog rješenja

Za razvoj programskog rješenja implementiranog na osobnom računalu, korišteno je računalo opremljeno *Intel(R) Core(TM) i7-4790* procesorom, 16 GB radne memorije i integriranim *Intel(R) HD Graphics 4600* grafičkim procesorom. Programsko rješenje je na osobnom računalu

implementirano pomoću programskog jezika *Python* i nekoliko različitih biblioteka. O korištenim tehnologijama je rečeno nešto više u nastavku.

### 3.1.1. *Python* programski jezik

*Python* programski jezik nastao je 1980-ih godina kao nasljednik programskom jeziku *ABC*. Verzija *Python 2.0* predstavljena je javnosti 2000. godine, te su uvedene funkcionalnosti poput rada s listama (engl. *List comprehensions*) i upravljanja memorijom s brojanjem referenci (engl. *garbage collection system with reference counting*). 2008. godine predstavljena je nova verzija *Pythona* pod nazivom *Python 3.0*. Za izradu programskog rješenja na osobnom računalu korištena je *Python 3.7.8* verzija, te biblioteke *NumPy* [4] i *OpenCV* kako bi se implementacija programskog rješenja ubrzala.

### 3.1.2. *NumPy* biblioteka

*NumPy* biblioteka je temeljni paket za znanstveno računanje unutar *Python* programskog jezika. *NumPy* biblioteka svojim korisnicima pruža brojne brze operacije nad nizovima, uključujući matematičke i logičke operacije. Tijekom izrade programskog rješenja na osobnom računalu, *NumPy* biblioteka je korištena kako bi se olakšao rad s matricama i vektorima.

### 3.1.3. *OpenCV* biblioteka

*OpenCV* (engl. *Open Source Computer Vision Library, OpenCV*) je biblioteka otvorenog koda (engl. *open source*) za računalni vid s naglaskom na primjenu u stvarnom vremenu (engl. *real time*). Biblioteka je izrađena kako bi pružila zajedničku infrastrukturu za programe računalnog vida, te sadrži više od 2500 optimiziranih algoritama, što uključuje sveobuhvatan skup klasičnih i najsuvremenijih algoritama za računalni vid i strojno učenje. Za potrebe izrade ovog rada primarno je korištena za testiranje, te kako bi se olakšao postupak učitavanja slika.

## 3.2. Implementacija programskog rješenja za detekciju otklona položaja kamere i korekciju položaja kamere na osobnom računalu

Implementacija programskog rješenja na osobnom računalu je podijeljena u osam koraka. Koraci će prvo biti navedeni, a zatim će svaki korak biti detaljnije objašnjen. Koraci su redom:

1. Učitavanje referentne i testne slike.
2. Konverzija zapisa boja slike iz RGB u HSV prostor.
3. Filtriranje referentne i testne slike po boji.
4. Detekcija značajnih točaka na referentnoj i testnoj slici.

5. Proračun matrice homografije.
6. Primjena proračunate matrice homografije na testnu sliku.
7. Rekonstrukcija testne slike pomoću interpolacije metodom najbližeg susjeda.
8. Prikaz testne slike nakon obrade.

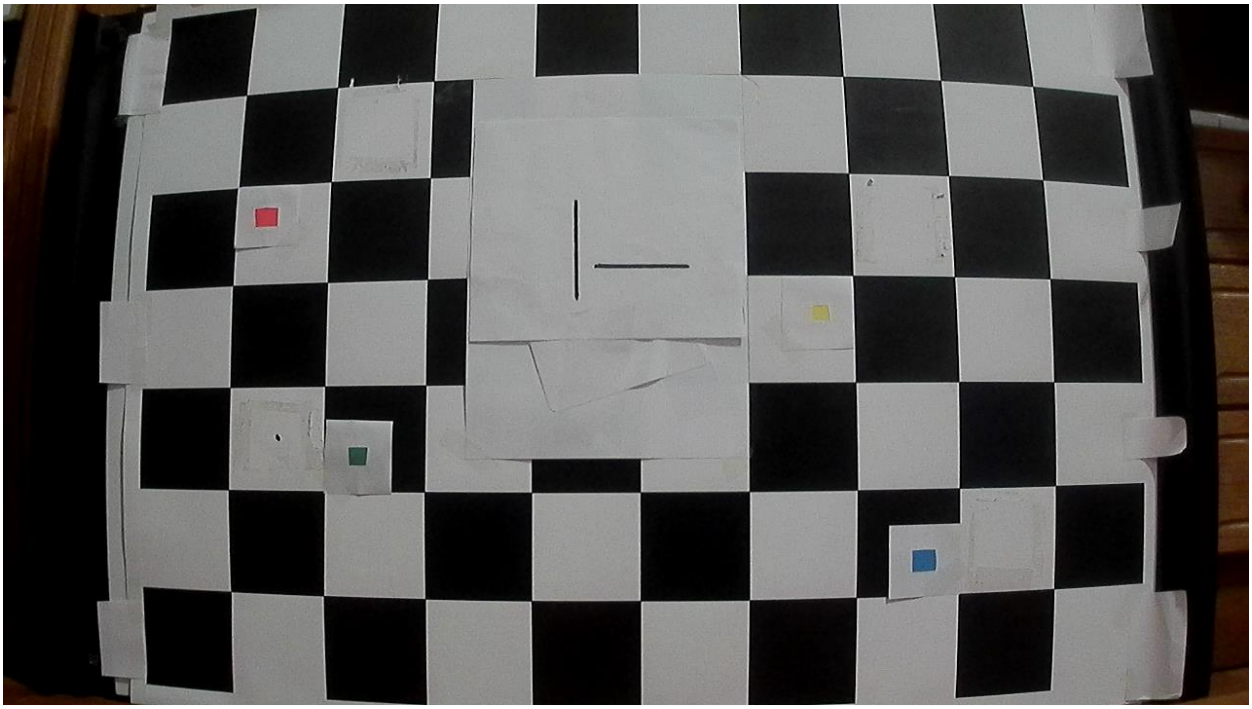
### 3.2.1. Učitavanje referentne i testne slike

Kao što je već ranije rečeno, za izradu programskog rješenja na osobnom računalu korišten je *Python* programski jezik, te *NumPy* i *OpenCV* programske biblioteke. Kao prvi korak kreirana je programska skripta, te su unutar skripte uključene programske biblioteke *NumPy* i *OpenCV*. Referentna slika 3.2. je slika snimljena s kamerom koja je pričvršćena na vozilo u nekakvom idealnom položaju. Bolje rečeno to je slika snimljena bez ikakvog otklona kamere, dok je testna slika snimljena s nekakvim otklonom kamere od referentnog položaja, kao što je na slici 3.3. kamera otklonjena od referentnog položaja za 3 centimetra unazad i 3 centimetra u desno. Referentna i testna slika su snimljene automotiv kamerom i imaju rezoluciju od 1280x720 elemenata slike. Na slikama 3.2. i 3.3. je moguće vidjeti da referentna i testna slika imaju četiri pravokutne oznake koje su različitih boja, te upravo te oznake predstavljaju točke značaja koje će se detektirati. Kako bi se referentna i testna slika mogle učitati, kreirane su dvije varijable naziva *rgbImgRef* i *rgbImgSrc,* te su učitane pomoću funkcije *imread* koja se nalazi unutar *OpenCV* biblioteke. Kako bi funkcija mogla učitati sliku, kao parametre joj je potrebno predati putanju do slike. Na slici 3.1. dan je primjer koda koji učitava referentnu i testnu sliku.

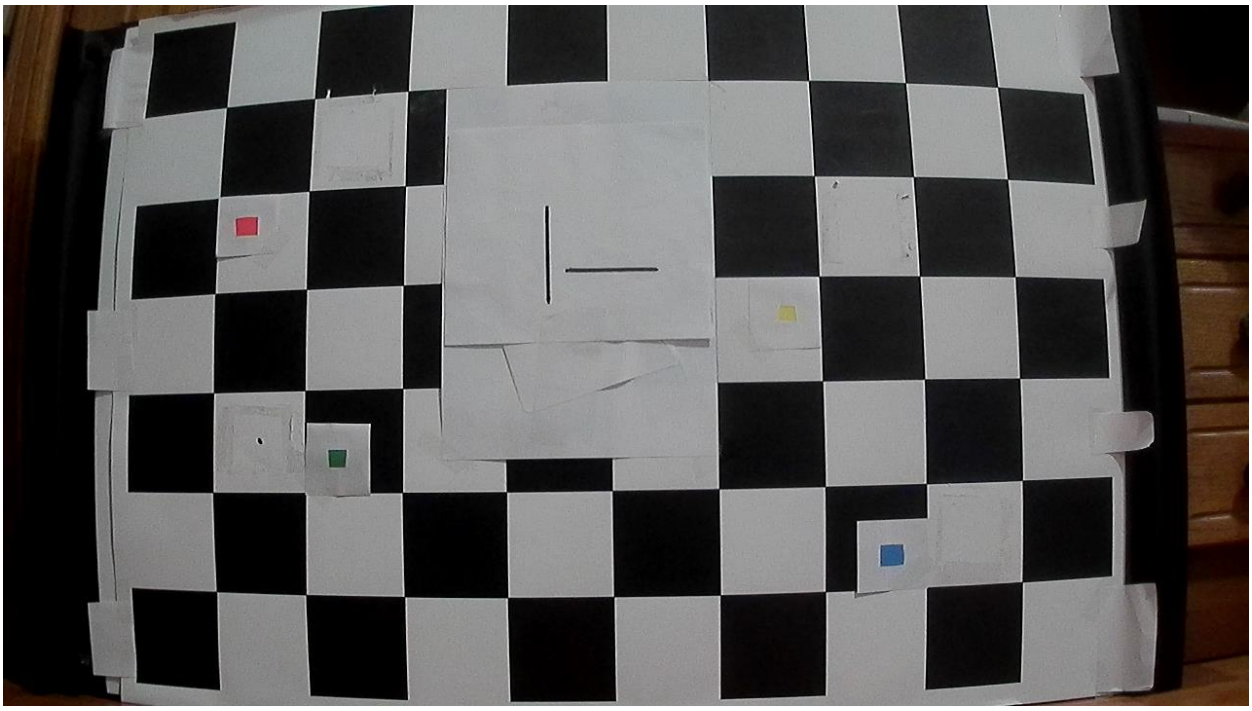
***Linija*** ***Kod***

```
1:      # uključene korištene programske biblioteke
2:      import cv2
3:      import numpy as np
4:
5:      # putanje do korištene referentne i testne slike
6:      rgbImgRefPath = „refImg/refImg.jpg“
7:      rgbImgSrcPath = „srcImg/srcImg.jpg“
8:
9:      # učitavanje korištene testne i referentne slike
10:     rgbImgRef = cv2.imread(rgbImgRefPath)
11:     rgbImgSrc = cv2.imread(rgbImgSrcPath)
```

Sl. 3.1. Učitavanje referentne i testne slike.



Sl. 3.2. Referentna slika.



Sl. 3.3. Testna slika.

Kod vidljiv na slici 3.1. u drugoj i trećoj liniji uključuje *OpenCV* i *NumPy* biblioteke. U šestoj i sedmoj liniji kreirana su dvije varijable tipa *string* koje predstavljaju putanje na kojima se nalaze referentna i testna slika, a na desetoj i jedanaestoj liniji su slike učitane u varijable pomoću

*imread* funkcije iz *OpenCV*-a. Na slici 3.2. se nalazi korištena referentna slika, a na slici 3.3. korištena testna slika koje su učitane i prikazane na zaslonu osobnog računala kodom koji se nalazi na slici 3.1.

### 3.2.2. Konverzija zapisa boja slike iz RGB u HSV prostor

Konverzija zapisa boja slike napravljena je kako bi se četiri značajke (četiri specifična objekta) koje se nalaze na uzorku koji se snima uspješno detektirale. Svaka značajka predstavljena je svojom bojom, te su korištene crvena, žuta, zelena i plava boja. Pomoću programskog rješenja kreiranog na osobnom računalu kreirano je rješenje za ADAS razvojnu ploču. Ranije je spomenuto kako je implementacija programskog rješenja za korištenu ADAS razvojnu ploču izvršena u *C* programskom jeziku pomoću *VisionSDK* programskog okruženja. Kako za *C* programski jezik ne postoji biblioteka *OpenCV*, sav kod korišten za obradu slike je samostalno pisan. Na slikama 3.2. i 3.3. se mogu vidjeti spomenute značajke. Budući da su značajne točke predstavljene različitim bojama, najlakši način za detekciju značajnih toči određenih boja je upotrebom HSV prostora boja. Nakon što se referentna i testna slika učitaju pomoću *OpenCV* funkcije *imread* elementi slike su zapisani unutar RGB modela boja, no *R* i *G*, odnosno crveni i zeleni kanal RGB slike su zamijenjeni, te se vrši promjena iz BGR u HSV prostor boja. Na slici 3.4. se nalazi *Python* kod koji predane elemente referentne i testne slike zapisane u BGR modelu boja pretvara u elemente slike zapisane pomoću HSV modela boja. Na početku programskog koda su uključene sve potrebne biblioteke. Nakon toga se nalazi definicija funkcije koja vrši konverziju elemenata referentne i testne slike zapisanih u BGR (RGB) modelu boja u elemente slike zapisane pomoću HSV modelu boja. Kod funkcionira na sljedeći način. Prilikom pozivanja funkcije *calcHSV*, predaju se gornja i donja granica HSV elemenata slike za crvenu, žutu, zelenu i plavu komponentu elemenata slike, te se vrši skaliranje vrijednosti komponenti boja kako bi se dobile vrijednosti u intervalu od 0 do 1. Nakon skaliranja potrebno je odrediti minimalnu i maksimalnu vrijednost između predane tri komponente, te izračunati razliku između maksimalne i minimalne vrijednosti komponenti. Sljedeći korak je proračun *H, S* i *V* komponenti. *V* komponentu je najjednostavnije za odrediti, te je njen iznos jednak maksimalnoj vrijednosti između prethodno skaliranih komponenti. Nakon toga proračunava se vrijednost *S* komponente. Ako je maksimalna vrijednost skaliranih komponenti 0, tada je vrijednost *S* komponente 0, inače se vrijednost *S* komponente proračunava tako da se razlika između maksimalne i minimalne vrijednosti skaliranih komponenti podjeli s iznosom maksimalne skalirane komponente.

***Linija*** ***Kod***

```
1:      # imports
2:      import cv2
3:      import numpy as np
4:
5:      def calcHSV(r, g, b):
6:          newR = r / 255
7:          newG = g / 255
8:          newB = b / 255
9:
10:         mx = max(newR, newG, newB)
11:         mn = min(newR, newG, newB)
12:         v = mx
13:         delta = mx - mn
14:
15:         if mx == 0:
16:             s = 0
17:         else:
18:             s = delta / mx
19:
20:         if mx == mn:
21:             h = 0
22:         else:
23:             if mx == newR:
24:                 h = (newG - newB) / delta
25:                 if newG <= newB:
26:                     h += 6
27:             elif mx == newG:
28:                 h = ((newB - newR) / delta) + 2
29:             elif mx == newB:
30:                 h = ((newR - newG) / delta) + 4
31:
32:             h /= 6
33:
34:             if h * 180 >= 180:
25:                 h = 0
26:
37:         return h * 180, s * 255, v * 255
```

Sl. 3.4. Primjer koda funkcij koji vrši pretvorbu elemenata slike iz RGB u HSV model boja.

Na samom kraju izračunava se vrijednost za *H* komponentu. Ako su maksimalna i minimalna vrijednost skaliranih komponenti jednake, tada je iznos *H* komponente jednaka nuli, a ako nije onda ju je potrebno proračunati na osnovu skaliranih RGB komponenti, te postoje tri

slučaja. Prvi slučaj je kada je vrijednost crvene komponente jednaka maksimalnoj vrijednosti skaliranih komponenti, u tom slučaju *H* komponenta se proračunava prema izrazu (3-1), gdje *greenVal* predstavlja iznos skalirane zelene komponente, *blueVal* predstavlja vrijednost skalirane plave komponente, a *maxVal* i *minVal* maksimalnu i minimalnu vrijednost između skalirane crvene, zelene i plave komponente.

$$H = \frac{(greenVal - \ blueValue)}{maxVal - minVal} \tag{3-1}$$

Ako je vrijednost skalirane zelene komponente manja ili jednaka vrijednosti plave skalirane komponente, tada se proračunatoj vrijednosti *H* komponente dodaje 6. Drugi slučaj je kada je vrijednost zelene komponente jednaka maksimalnoj vrijednosti skaliranih komponenti, u tom slučaju se vrijednost *H* komponente proračunava prema izrazu (3-2), gdje *blueVal* predstavlja vrijednost skalirane plave komponente, a *redVal* predstavlja iznos skalirane crvene komponente.

$$H = \frac{(blueVal - \ redValue)}{maxVal - minVal} + 2 \tag{3-2}$$

Treći slučaj je kada je vrijednost plave komponente jednaka maksimalnoj vrijednosti skaliranih komponenti, u tom slučaju se vrijednosti *H* komponente proračunava prema izrazu (3-3).

$$H = \frac{(redVal - \ greenValue)}{maxVal - minVal} + 4 \tag{3-3}$$

Na samom kraju, vrši se skaliranje vrijednosti. *H* komponenta se skalira na vrijednosti iz intervala od 0 do 180, a *S* i *V* komponente na vrijednosti u intervalu od 0 do 255.

### 3.2.3. Filtriranje referentne i testne slike po boji

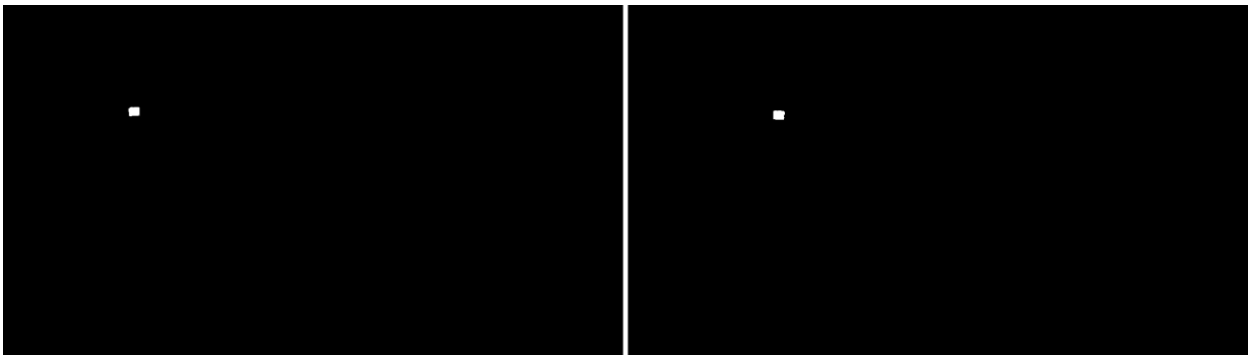
Nakon što su referentna i otklonjena slika prebačene iz BGR (RGB) u HSV prostor boja potrebno je izvršiti filtriranje slike po boji. Filtriranje slike se provodi pomoću odgovarajućih maski za crvenu, žutu, zelenu i plavu boju jer su to objekti od interesa na uzorku koji se snima (vidi slike 3.2. i 3.3.). Boje koje se filtriraju su definirane pomoću maski koje se sastoje od gornje i donje granice za svaku komponentu HSV elementa slike. Na slici 3.5. se nalazi programski kod napisan u *Python* programskom jeziku koji pomoću gornje i donje granice maske za žutu boju vrši filtriranje slike.

***Linija*** ***Kod***

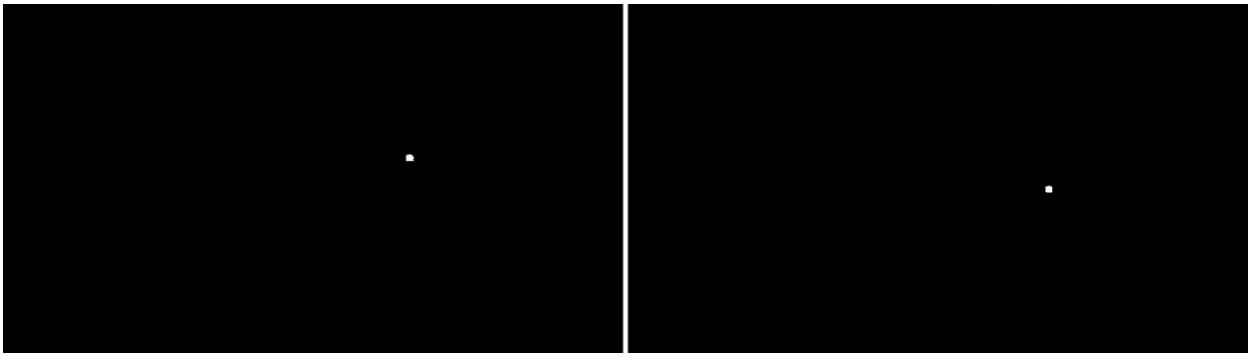
```
1:      # gornja i donja granica za žutu masku
2:      lowerYellow = np.array([20, 70, 150])
3:      upperYellow = np.array([30, 255, 255])
4:
5:      # filtriranje referentne slike za žutu boju
        yellowFiltered = cv2.inRange(refHsvImg, lowerYellow, upperYellow)
```

Sl. 3.5. Primjer maske za filtriranje žute boje unutar *Python* programskog jezika.

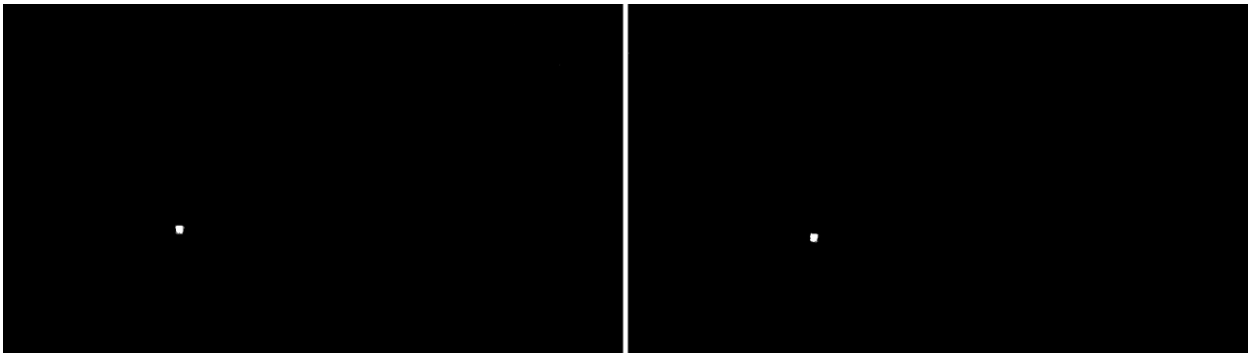
Ovdje je bitno napomenuti da je maske boja potrebno ručno podesiti jer vrijednosti pojedinih komponenti ovise o osvjetljenju na slici. Referentna i testna slika su snimljene pod identičnim uvjetima što se tiče osvjetljena pa je jednaki filter korišten i za referentnu i testnu sliku. Na slikama 3.6, 3.7, 3.8. i 3.9 se nalaze referentna i testna slika 3.2. i 3.3. koje su filtrirane po boji. Lijeva slika je referentna, a desna testna slika, te su referentna i testna slika filtrirane pomoću maski koje se nalaze u tablici 3.1.



Sl. 3.6. Referentna (lijevo) i testna (desno) slika nakon primjene maske za crvenu boju.



Sl. 3.7. Referentna (lijevo) i testna (desno) slika nakon primjene maske za žutu boju.



Sl. 3.8. Referentna (lijevo) i testna (desno) slika nakon primjene maske za zelenu boju.

Sl. 3.9. Referentna (lijevo) i testna (desno) slika nakon primjene maske za plavu boju.

Tablica 3.1. Donje i gornje granice korištene pri filtriranju crvene, žute, zelene i plave boje.

| *Element* | *H* | *S* | *V* |
|---|---|---|---|
| *Crvena – donja granica* | 140 | 100 | 100 |
| *Crvena – gornja granica* | 179 | 255 | 255 |
| *Žuta – donja granica* | 20 | 80 | 80 |
| *Žuta - gornja granica* | 30 | 255 | 255 |
| *Zelena – donja granica* | 36 | 60 | 60 |
| *Zelena – gornja granica* | 86 | 255 | 255 |
| *Plava – donja granica* | 94 | 80 | 80 |
| *Plava – gornja granica* | 126 | 255 | 255 |

### 3.2.4. Detekcija značajnih točaka s referentne i testne slike

Nakon izvršene filtracije boja na slici, sljedeći korak je detekcija značajnih točaka s referentne i otklonjene slike. Konverzija boja i filtriranje slike po boji je odrađeno upravo kako bi se značajne točke mogle detektirati na slici. Recimo da je izvršeno filtriranje slike i da je filtrirana crvena boja. Nakon filtriranja slike po boji kao rezultat filtriranja detektirana je crvena oznaka (vidi sliku 3.6.), te su dohvaćene koordinate svih crvenih elemenata slike koji se nalaze unutar definiranih granica HSV modela boja za crvenu boju. Za proračun matrice homografije dovoljna

je koordinata samo jednog crvenog elementa slike, te se ostale koordinate mogu odbaciti. Ovdje je bitno napomenuti da i na referentnoj i na testnoj slici se dohvaćaju koordinate istog elementa slike koji odgovara istom dijelu crvenog objekta. Primjer koda pomoću kojeg se vrši filtriranje boja i dohvaćanje željenih koordinata dan je na slici 3.10. Na slici 3.10. u drugoj liniji koda je pozvana *OpenCV* funkcija, koja kao prvi parametar prima sliku u HSV prostoru boja i pomoću donje i gornje granice crvene boje (vidi tablicu 3.1.) vrši filtriranje slike, te kao rezultat vraća crno-bijelu sliku na kojoj se nalaze samo crveni elementi slike koji su prikazani kao bijeli elementi slike, a svi elementi slike koji nisu crvene boje kao crni elementi slike. Nakon toga je u petoj liniji pozvana samostalno napisana funkcija pomoću koje se dohvaćaju koordinate crvenih elemenata slike, odnosno, kao povratnu vrijednost funkcija vraća koordinate pronađenog 32. crvenog elementa slike. 32. element slike je odabran budući da nije vršena implementacija algoritma pomoću kojeg bi se vršio odabir središnjeg element slike filtrirane boje, te je pomoću ovog izvršena aproksimacije središnjeg elementa slike filtrirane boje. Kada se funkcija pozove vrijednost zastavice *isFound* postavi se na *vrijednos* istina (engl. *true*). Zatim se prolazi kroz svaki element slike, te se provjerava koji dio slike se trenutno obrađuje te koja je vrijednost trenutnog elementa slike. Na primjeru referentne i testne slike koje se nalaze na slikama 3.2. i 3.3. je moguće vidjeti da su crvena, žuta, zelena i plava oznaka smještene svaka na svojem dijelu slike, odnosno ako bi se slika podijelila na četiri dijela, na slikama 3.2. i 3.3. je vidljivo kako se svaka oznaka nalazi unutar svoje četvrtine slike. Ako je vrijednost trenutnog elementa slike jednaka 255 i ako se element slike nalazi na gornjoj lijevoj četvrtini slike, tada se radi o crvenom elementu slike, te se dohvaćaju njegove koordinate. Isti princip je i za elemente slike žute, zelene i plave boje, no ti elementi se nalaze na drugim dijelovima slike. Žuti elementi slike se nalaze u gornjoj desnoj četvrtini, zeleni u donjoj lijevoj i plavi u donjoj desnoj. Da bi se dohvatile koordinate za sve četiri boje funkciju je potrebno pozvati četiri puta gdje je prilikom poziva funkcije svaki put potrebno predati odgovarajuće donje i gornje granice boje.

***Linija Kod***

```
1:      # filtriranje crvenih elemenata slike
2:      redPixels = cv2.inRange(hsvImgRef, lowerRed, upperRed)
3:
4:      # dohvaćanje koordinata crvenih elemenata slike
5:      redSrcPtsX, redSrcPtsY = getPoints(redPixels)
6:
7:      # funkcija koja dohvaća koordinate elemenata slike određene boje
8:      def getPoints(img):
9:          isFound = False
10:
11:         for i in range(img.shape[0]):
12:             for j in range(img.shape[1]):
13:                 pixel = img[i][j]
14:
15:                 if i <= img.shape[0] / 2:
16:                     if j<= (img.shape[1]) and pixel == 255:
17:                         ptsX = i
18:                         ptsY = j
19:                         isFound = True
20:                     elif j > (img.shape[1] / 2) and pixel == 255:
21:                         ptsX = i
22:                         ptsY = j
23:                         isFound = True
24:
25:                 elif i > img.shape[0]:
26:                     if j <= (img.shape[1] / 2) and pixel == 255:
27:                         ptsX = i
28:                         ptsY = j
29:                         isFound = True
30:                     elif j > (img.shape[1] / 2) and pixel == 255:
31:                         ptsX = i
32:                         ptsY = j
33:                         isFound = True
34:                 if isFound:
25:                     break
26:
37:         return ptsX, ptsY
```

Sl. 3.10. Funkcija za filtriranje slike po boji i dohvaćanje koordinata značajnih točki.

### 3.2.5. Proračun matrice homografije

Sljedeći korak, nakon što se dobiju koordinate značajnih točki s referentne i testne slike je proračun matrice homografije. Kao i što je ranije rečeno, matrica homografije je matrica dimenzije

3x3 koju je moguće proračunati tako da se riješi sustav od devet jednadžbi s devet nepoznanica dan izrazom (2-7). Unutar biblioteke *OpenCV* se nalazi funkcija pomoću koje je moguće proračunati matricu homografije. Funkcija kao parametre prima dva seta točaka. Jedan set točaka predstavlja četiri točke koje su dobivene iz referentne slike, a drugi, četiri točke koje su dobivene iz testne slike. Funkcija iz dobivenih setova točaka proračunava matricu homografije, te je vraća kao povratnu vrijednost. Na slici 3.11. se nalazi programski kod koji pomoću biblioteke *OpenCV* vrši proračun matrice homografije.

***Linija*** ***Kod***

```
1:      # proračun matrice homografije
2:      homographyMatrix = cv2.findHomography(srcPts, refPts)
```

Sl. 3.11. Primjer poziva funkcija za proračun matrice homografije.

Na slikama 3.6, 3.7, 3.8. i 3.9. se nalaze referentna (lijevo) i testna (desno) slika na kojoj su detektirane značajne točke. Pomoću dohvaćenih koordinata formirana matrica **P** koja se nalazi u izrazu (3-4), te je izvršen proračun matrice homografije rješavanjem sustava (2-7). Dobivena matrica homografije dana je u izrazu (3-5).

$$\mathbf{P} = \begin{bmatrix} -246 & -220 & -1 & 0 & 0 & 0 & 246*276 & 220*276 & 276 \\ 0 & 0 & 0 & 246 & 220 & -1 & 246*211 & 220*211 & 211 \\ -808 & -314 & -1 & 0 & 0 & 0 & 808*836 & 314*836 & 836 \\ 0 & 0 & 0 & 808 & 314 & -1 & 808*313 & 314*313 & 313 \\ -337 & -460 & -1 & 0 & 0 & 0 & 337*360 & 460*360 & 360 \\ 0 & 0 & 0 & 337 & 460 & -1 & 337*459 & 460*459 & 459 \\ 922 & -556 & -1 & 0 & 0 & 0 & 922*953 & 556*953 & 953 \\ 0 & 0 & 0 & 922 & 556 & -1 & 922*562 & 556*562 & 562 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3-4}$$

$$\boldsymbol{H} = \begin{bmatrix} 1.01484261 & -6.80217294e-02 & 3.81092510e+01 \\ 2.32114441e-02 & 9.75351795e-01 & -1.17370348e+01 \\ 3.82128821e-05 & -9.54998366e-05 & 1 \end{bmatrix} \tag{3-5}$$

### 3.2.6. Primjena proračunate matrice homografije na testnu sliku

Matrica homografije se u radu primjenjuje za korekciju otklona položaja kamere. Postupak korekcije otklona pomoću matrice homografije je prilično jednostavan, te se sastoji od svega nekoliko koraka. Prvi korak je prolazak kroz sve elemente testne slike i dohvaćanje koordinata elemenata slike. Nakon što su koordinate elementa testne slike dohvaćene, potrebno je formirati

vektor dimenzije 3x1 i koordinate smjestiti unutar vektora gdje će se na prvom elementu vektora nalaziti *x* koordinata trenutnog elementa slike, na drugom *y* koordinata, te na trećem vrijednost 1. Sljedeći korak je proračun novih koordinata $x'$ i $y'$ pomoću matrice homografije. Izraz (3-6) prikazuje matematički zapis proračuna novih koordinata za elemente testne slike pomoću matrice homografije **H** i novo kreiranog vektora unutar kojeg se nalaze *x* i *y* koordinate trenutnog elementa testne slike.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{3-6}$$

Nove koordinate elemenata testne slike se dobiju tako da se matrica homografije **H** pomnoži s novokreiranim vektorom unutar kojeg se nalaze *x* i *y* koordinate elemenata slike, gdje se kao rezultat dobije vektor dimenzije 1x3 unutar kojeg se na prva dva mjesta nalaze nove $x'$ i $y'$ koordinate. Cijeli postupak se ponavlja za svaki element slike. Biblioteka *OpenCV* sadrži funkciju pomoću koje je moguće proračunati nove koordinate elemenata slike. Navedena projekcija je napravljena u *Pythonu* kako bi se olakšalo pisanje koda u programskom jeziku *C* za ADAS razvojnu ploču.. U nastavku slijedi slika 3.12. na kojoj se nalazi programski kod kreiran u *Python* programskom jeziku pomoću kojeg se proračunavaju nove koordinate, te se vrši kompenzacija otklona položaja kamere.

Kreirana funkcija vidljiva na slici 3.12. naziva *warpImg* kao parametre prima testnu sliku i transponiranu [18] matricu homografije, odnosno matricu homografije kojoj su zamijenjeni redci i stupci. Na samom početku se kreira nova matrica unutar koje se pohranjuje slika nakon kompenzacije otklona i *lookup tablica* (engl. *lookup table*) koja će se popuniti prilikom kompenzacije otklona, a koristit se prilikom rekonstrukcije slike. Nakon što su kreirane sve potrebne matrice kreće izvođenje algoritma.

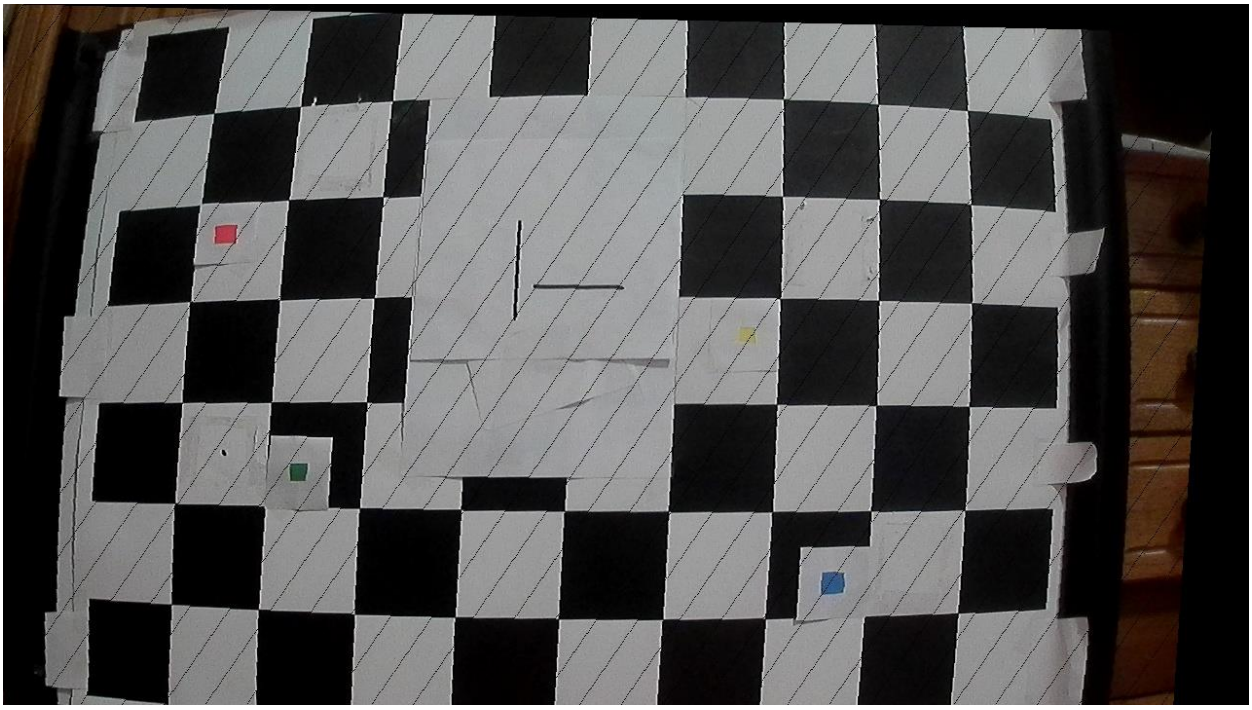
***Linija Kod***

```
1:      warpedImg = warpImg(srcImg, np.transpose(homographyMatrix))
2:
3:      def warpImg(img, homographyMatrix):
4:          height, width, colorLayer = img.shape
5:          warpedImg = np.zeros((height, width, colorLayer), dtype='uint8')
6:          lookupTable = np.zeros((height, width), dtype='int8')
7:
8:          for i in range(warpedImg.shape[0]):
9:              for j in range(warpedImg.shape[1]):
10:                 oldCrd = [i, j, 1]
11:
12:                 newCrd = np.dot(homographyMatrix, oldCrd)
13:                 newX = newCrd[0]
14:                 newY = newCrd[1]
15:
16:                 if newX < 0:
17:                     newX = int(0)
18:                 elif newX > 1280:
19:                     newX = int(1280)
20:                 else:
21:                     newX = int(round(newCrd[0]))
22:
23:                 if newY < 0:
24:                     newY = int(0)
25:                 elif newY > 720:
26:                     newY = int(720)
27:                 else:
28:                     newY = int(round(newCrd[1]))
29:
                    if newX < warpedImg.shape[0] and newX <
30:     warpedImg.shape[1]:
31:                     warpedImg[newX][newY] = img[i][j]
32:                     lookupTable[newX][newY] = -1
33:
            return fixMissPixels(warpedImg, lookupTable, img,
34:     homographyMatrix)
```

Sl. 3.12. Funkcija koja vrši kompenzaciju otklona kamere.

Algoritam radi kao što je objašnjeno u prethodnom odlomku, prolazi se kroz svaki element testne slike, te se dohvaćaju koordinate svakog elementa slike. Zatim se proračunavaju nove koordinate pomoću matrice homografije i vrše se provjere vrijednosti dobivenih koordinata. Vrijednosti koordinata za *X* os slike se trebaju nalaziti u intervalu od 0 do 1280, a za *Y* os u

intervalu od 0 do 720, a ako nisu postavljaju se na maksimalnu odnosno minimalnu vrijednost iz navedenih intervala, te se na samom kraju vrši upisivanje obrađivanog elementa testne slike u novu matricu i upisivanje oznake -1 u *lookup tablicu*. Na samom kraju se prilikom vraćanja slike s ispravljenim otklonom poziva funkcija *fixMissPixels* koja kao parametre prima sliku s ispravljenim otklonom, *lookup tablicu*, testnu sliku i matricu homografije. Na slici 3.13. se nalazi testna slika nakon kompenzacije otklona položaja kamere. U nastavku je dano nešto više detalja o funkciji pomoću koje se vrši rekonstrukcija praznih elemenata slike pomoću interpolacijske metode najbližeg susjeda.



Sl. 3.13. Testna slika nakon kompenzacije otklona položaja kamere pomoću matrice homografije.

### 3.2.7. Rekonstrukcija testne slike pomoću interpolacije metodom najbližih susjeda

Ako se slika 3.13. promotri malo bolje, na slici je moguće vidjeti kako neki elementi slike koji bi trebali biti popunjeni imaju nedefinirane vrijednosti, jednostavnije rečeno na slici su se pojavile „rupe“. Kako bi se doskočilo nastalom problemu, nakon što se proračunaju nove koordinate elemenata testne slike, vrši se rekonstrukcija slike. Rekonstrukcija praznih elemenata testne slike izvodi pomoću interpolacije metodom najbližih susjeda.

Algoritam prolazi kroz sve elemente testne slike nakon kompenzacije otklona, te na istoj pronalazi $x'$ i $y'$ koordinate elemenata slike koji nedostaju. Pomoću pronađenih koordinata i inverzne matrice homografije, algoritam u testnoj slici prije kompenzacije otklona pronalazi najbliži element $x, y$ te na taj način vrši rekonstrukciju elemenata testne slike nakon kompenzacije

otklona.. U tablici na slici 3.14. je moguće vidjeti kako izgleda kod koji vrši rekonstrukciju slike. Funkcija *fixMissPixels* pomoću *lookup tablice* pronalazi elemente testne slike koji nedostaju. Funkcija prolazi kroz *lookup tablicu* i traži praznine u nizu vrijednosti, odnosno vrijednost 0. Kada se unutar *lookup tablice* pronađe vrijednost 0, poziva se funkcija *nearestNeighbors* koja kao parametre prima koordinate praznog elemenata testne slike, testnu sliku prije kompenzacije otklona i inverznu matricu homografije. Unutar funkcije se pomoću inverzne matrice homografije proračunavaju koordinate elemenata slike koji nedostaju, te se prazni elementi slike dohvaćaju iz testne slike prije kompenzacije otklona. Dohvaćeni elementi se zatim kopiraju i pohranjuju unutar matrice gdje se nalazi slika na kojoj je izvršena kompenzacija otklona. Postupak se ponavlja za svaki element slike koji nedostaje.

***Linija Kod***

```
1:      def fixMissPixels(warpedImg, lookupTable, img, homographyMatrix):
2:          height, width, colorLayer = warpedImg.shape
            interpolatedImg = np.zeros((height, width, colorLayer),
3:      dtype='uint8')
4:
5:          invHomographyMatrix = np.linalg.inv(homographyMatrix)
6:          for i in range(height):
7:              for j in range(width):
                    if lookupTable[i][j] != -1 and lookupTable[i][j - 1] ==
8:      -1 and lookuptable[i - 1][j] == -1:
                        interpolatedImg[i][j] == nearestNeighbors(i, j, img,
9:      invHomographyMatrix)
10:                     warpedImg[i][j] = interpolatedImg[i][j]
11:
12:         return warpedImg
13:
14:
15:     def nearestNeighbors(i, j, img, invHomographyMatrix):
16:         height, width, _ = img.shape
            foundRow, foundCol, _ = np.dot(invHomographyMatrix, np.array([i,
17:     j, 1])
18:
            if np.floor(roundRow) == foundRow and np.floor(foundCol) ==
19:     foundCol:
20:             foundRow, foundCol = int(foundRow), int(foundCol)
21:             return img[foundRow][foundCol]
            if np.abs(np.floor(foundRow) - foundRow) <
22:     np.abs(np.ceil(foundRow) - foundRow):
23:             foundRow = int(np.floor(foundRow))
24:         else:
25:             foundRow = int(np.ceil(foundRow))
26:
            if np.abs(np.floor(foundCol) - foundCol) <
27:     np.abs(np.ceil(foundCol) - foundCol):
28:             foundCol = int(np.floor(foundCol))
29:         else:
30:             foundCol = int(np.ceil(foundCol))
31:
32:         if foundRow > width:
33:             foundRow = width
34:         If foundCol > height:
35:             foundCol = height
36:
37:         return img[x][y]
```

Sl. 3.14. Funkcije koje vrše rekonstrukciju praznih vrijednosti elemenata slike.

### 3.2.8. Prikaz testne slike nakon obrade

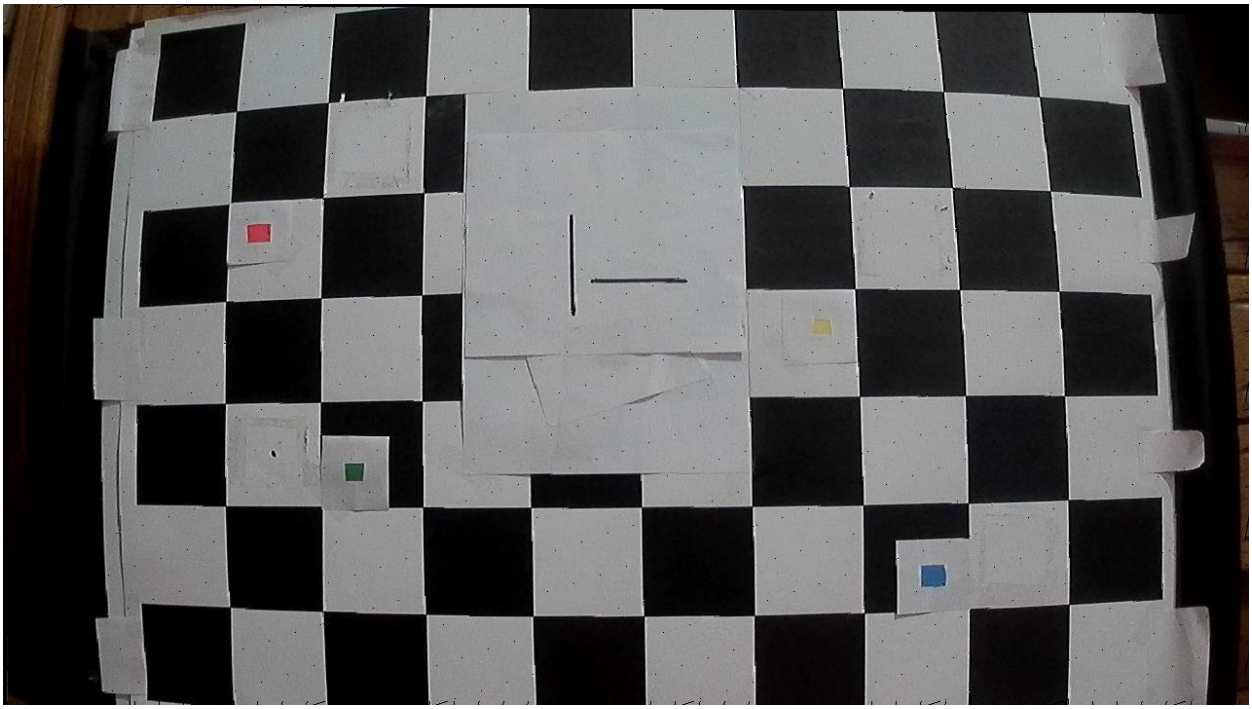
Rekonstrukcija slike nakon kompenzacije otklona je predzadnji korak u implementiranom programskom rješenju kreiranom na osobnom računalu pomoću programskog jezika *Python*, te je kao zadnji korak potrebno prikazati testnu sliku s kompenziranim otklonom i izvršenom rekonstrukcijom praznih elemenata slike kako bi se mogli uvjeriti da je implementirano rješenje uspješno kompenziralo otklon i rekonstruiralo sliku. Prikaz testne slike s kompenziranim otklonom i rekonstruiranim elementima slike koji nedostaju je napravljen pozivom funkcije unutar biblioteke *OpenCV*. Korišteni programski kod za prikaz testne slike nakon kompenzacije otklona i rekonstrukcije slike se nalazi na slici 3.15.

***Linija*** ***Kod***

```
1:      # prikaz testne slike nakon kompenzacije otklona i rekonstrukcije
2:      cv2.imshow(„warpedImg“, warpedImg)
3:      cv2.waitKey(0)
```

Sl. 3.15. Primjer koda korištenog za prikaz testne slike nakon kompenzacije otklona i rekonstrukcije.

Na slici 3.15. je iz biblioteke *OpenCV* pozvana funkcija je *imshow* koja kao parametre prima naziv prozora (engl. *window*) i matricu unutar koje je pohranjena slika. Nakon poziva funkcije *imshow* potrebno je pozvati i funkciju *waitKey* kako se prozor sa slikom ne bi odmah zatvorio. Kao parametar funkciji *waitKey* je predana vrijednost 0 pomoću koje pritiskom na bilo koju tipku s tipkovnice kreirani prozor sa slikom zatvara. Na slici 3.16. se nalazi testna slika nakon kompenzacije otklona kamere i rekonstrukcije praznih elemenata slike.



Sl. 3.16. Testna slika nakon kompenzacije otklona položaja i rekonstrukcije praznih elemenata slike.

## 3.3. Pregled tehnologija korišteni za razvoj programskog rješenja za realnu ADAS razvojnu ploču

Kao što je već ranije spomenuto rješenje za ADAS razvojnu ploču je implementirano pomoću *C* programskog jezika unutar *VisionSDK* programskog okruženja. U nastavku ovog poglavlja dano je nešto više detalja o programskom jeziku *C* i razvojnom okruženju *VisionSDK* kao i o samoj ADAS razvojnoj ploči na kojoj je programsko rješenje razvijeno i testirano, te je nakon toga predstavljeno implementirano programsko rješenje.

### 3.3.1. Programski jezik *C*

*C* je veoma moćan programski jezik generalne primjene, može se koristiti za razvoj softverskih rješenja kao što su operacijski sustavi, baze podataka, prevoditelji (engl. *compilers*) i sl. Razvio ga je *Dennis Ritchie* u periodu između 1972. i 1973. za razvoj uslužnih (engl. *utilities*) programa koji rade na *Unixu*. *C* je nasljednik programskog jezika *B* [19], te je korišten za ponovnu implementacija jezgre operacijskog sustava *Unix*. Tijekom 1980-ih, *C* je postupno stekao popularnost kod velikog broja programera, te je postao jedan od najčešće korištenih programskih jezika. Od 1989. godine je standardiziran od strane američkog nacionalnog instituta za standarde (engl. *American National Standards Institute* – ANSI) [20], te ne dugo nakon toga i od strane

međunarodne organizacije za standardizaciju (engl. *International Organization for Standardization* – ISO) [21].

### 3.3.2. ADAS ALPHA razvojna ploča

Za izradu rada korištena je ADAS ALPHA razvojna ploča koja se sastoji od tri sustava na čipu (engl. *System on a Chip* - SoC). Ploču je razvila tvrtka *Texas Instruments*. Prvi SoC kratice SCV (engl. *Surround Camera View* - SCV) koristi se za kružni pregled oko vozila (engl. *Surround View*). FFN (engl. *Front view camera near angle stereoscopic view, Front view camera wide angle, Night vision camera*) SoC koristi se za snimanje stanja ispred vozila i FUS (engl. *Fusion*) SoC koji se koristi za obradu informacija SCV i FFN SoC-a. Sva tri SoC-a imaju iste značajke, te su one u nastavku navedene.

1. 2x *ARM Cortex A15* jezgre s frekvencijom do 1150 MHz po jezgri.
2. 2x *ARM Cortex-M4* jezgre s frekvencijom 200 MHz po jezgri.
3. 2x sljedeće generacije (engl. *next-generation*) *C66x* fiksne pomične decimalne točke (engl. *fixed-/floating-point*) *DSP* jezgre s frekvencijom do 750 MHz.
4. HW video kodek akcelerator za kodiranje/dekodiranje *Full HD* videa (*IVA HD 1080p* video).
5. HW akcelerator za obradu slike i grafike (engl. *Graphics Engine*).
6. 4x *Embedded Vision Engine (EVE).*
7. *Video Processing Engine (VPE).*
8. 1.5 GB DDR3 memorije.
9. 32 MB *flash* memorije.
10. *EEPROM* spojen preko *I2C* konekcije.
11. 1x *HDMI* izlaz.
12. 1x *JTAG.*
13. 10x *UART.*
14. 2x *DCAN.*
15. 1x utor za *microSD* karticu.
16. 4x *SPI.*
17. 5x *I2C.*
18. 1x *QSPI.*

Svaki od navedena tri SoC-a je moguće pokrenuti u tri različita načina rada *Debug* (*JTAG*), *SD* i *QSPI Flash*. Pri korištenju *Debug* načina rada, na korištenu ADAS razvojnu ploču je moguće

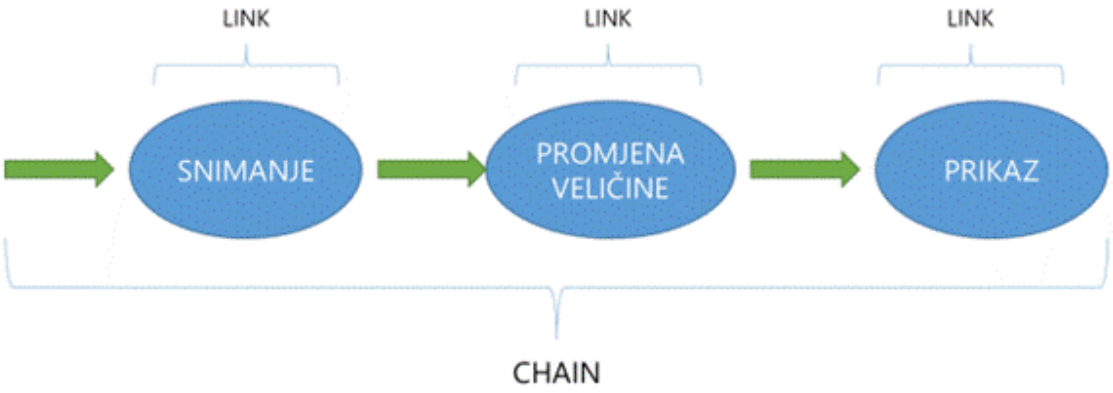
spojiti uređaj za otklanjanje programskih grešaka, te koristiti program za ispravljanje grešaka (engl. *Debugger*). *SD* način rada je najčešće korišten način rada u praksi, te se pomoću *SD* načina rada i *microSD* kartice pokreću programska rješenja na ploči. Odabiranje načina rada se vrši pomoću podešavanja prekidača koji se nalaze na samoj ploči. *ADAS ALPHA* razvojna ploča posjeduje veliki broj konektora. Na ploči se nalaze dva *Ethernet* konektora i deset konektora za kamere od koji je šest za SCV SoC i četiri za FFN SoC, te tri utora za *microSD* karticu pri čemu svaki odgovara pojedinom SoC-u. Navedeni utori za *microSD* karticu služe kao primarni način pokretanja programskih rješenja kreiranih za *ADAS ALPHA* razvojnu ploču.

### 3.3.3. *VisionSDK* programsko okruženje

*VisionSDK* je programsko razvojno okruženje (engl. *Software Development Kit* - SDK) koje je razvila tvrtka *Texas Instruments* za potrebe programiranja razvijenih ADAS SoC-ova. *VisionSDK* programsko okruženje pruža brojne mogućnosti kao što su preuzimanje, snimanje, predobrada i prikaz videa, te omogućava razvoj algoritama koji služe za obradu videa i slike. Omogućava i podršku da se obrada različitih tokova podataka izvršava na različitim procesorskim jedinicama (engl. *Central Processing Unit* – CPU) i pomoću različitih sklopovskih akceleratora, s ciljem učinkovitog korištenja različitih SoC-ova. Sam *VisionSDK* zasnovan je na okviru zvanom veze i lanci (engl. *Links and Chains*), a korisničko programsko sučelje (engl. *Application Programming Interface* - API) ovog okvira zove se *Link API*. Okvir ne omogućava automatsko balansiranje opterećenjem različitih procesora, nego je to zadatak korisnika.

### 3.3.4. Links and Chains okvir

*Links and Chains* je okvir koji se koristi pri definiranju dijagrama toka slučaja upotrebe (engl. *Usecase*), gdje se jedan lanac (engl. *Chain*) sastoji od više međusobno povezanih linkova (karika). Slika 3.17. prikazuje grafički prikaz opisane strukture. Svaki Link definira određeni algoritam, hardversku vezu na ploči, obradu signala i sl.




Sl. 3.17. Jedan lanac povezan s više veza.

Svaki link se izvršava kao zasebna nit (engl. *Thread*) i ima svoj poštanski sandučić (engl. *Mailbox*) preko kojeg ostali linkovi vrše komunikaciju. Svaki link ima jednu ili više ulaznih i izlaznih konekcija. Svaka od tih konekcija može u sebi sadržavati jedan ili više logičkih kanala Na slici 3.18. prikazan je link.


Sl. 3.18. Primjer linka.

Programsko rješenje implementirano na ADAS razvojnu ploču se sastoji od nekoliko linkova od kojih su najbitniji *Capture*, *Split*, *Select*, *VPE*, *Null*, *Alg_WarpImg* i *Display*. *Capture* link je korišten za snimanje videa, *Split* link za podjelu snimljenog videa na dva signala, a *VPE* link za promjenu formata boja. Budući da je programsko rješenje za ADAS razvojnu ploču implementirano na SCV SoCu koji zahtjeva da se koriste dvije kamere, signal koji snima jedna kamera se šalje na *Null* link jer se signal koji snima druga kamera ne koristi prilikom obrade, te se za odabiranje signala koji će se slati na *Null* link koristi *Select* link. Implementirani programski kod, kreiran pomoću programskog rješenja implementiranog na osobnom računalu, koji vrši kompenzaciju otklona položaja kamere je predstavljen *Alg_WarpImg* linkom, a pomoću *Display* linka je prikazano na zaslonu monitora.

## 3.4. Implementacija programskog rješenja detekcije otklona položaja kamere i korekcije položaja kamere za primjenu u ADAS algoritmima

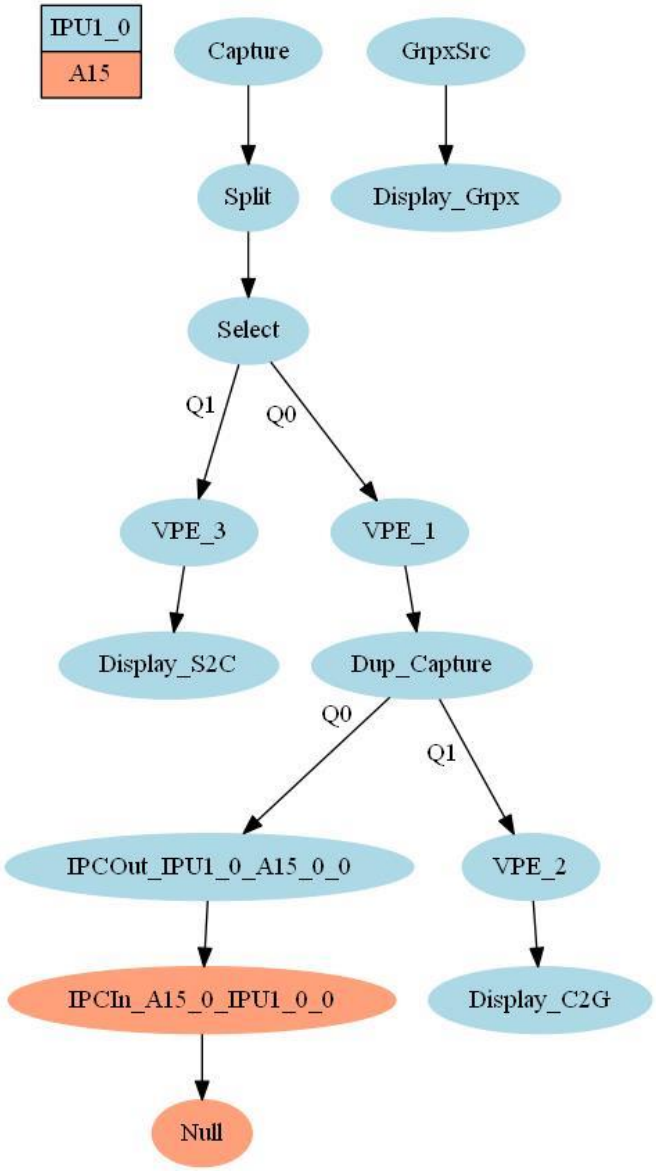
Programsko rješenje implementirano na ADAS platformu zasnovano je na programskom rješenju implementiranom na osobnom računalu. U nastavku je detaljno opisano programsko rješenje implementirano na ADAS razvojnoj ploči. Programsko rješenje je podijeljeno u nekoliko koraka. U poglavlju 3.4.1. kreiran je dijagram toka slučaja upotrebe za snimanje referentne i testne slike, zatim u poglavlju 3.4.2. je dodan vlastiti algoritam, nakon toga u poglavlju 3.4.3. kreiran je dijagram toka slučaja upotrebe, te se u poglavlju 3.4.4 nalazi implementacija predloženog rješenja za detekciju otklona položaja kamere i korekciju položaja kamere.

### 3.4.1. Dijagram toka slučaja upotrebe - snimanje referentne i testne slike

Referentna i testna slika koje su korištene prilikom implementacije programskog rješenja na osobnom računalu, snimljene su korištenjem dijagrama toka slučaja upotrebe. Dijagram toka slučaja upotrebe kreiran za potrebe snimanja referentne i testne slike se sastoji od *Capture*, *Split*, *Select*, *VPE*, *Display* i *Null* linkova, te se nalazi na slici 3.19. Pomoću *Capture* linka snima se video zapis i predaje *Split* linku koji vrši razdvajanje dobivenog ulaznog signala. Nakon toga pomoću *Select* linka se vrši odabir signala kamere koji će se obrađivati. Pomoću *VPE* linka podešava se format boje obrađivanog signala na *YUV 420P*, te se zatim signal korištenjem linka *Dup* duplicira kako bi se jedna kopija signala poslala na zaslon spojenog monitora korištenjem *Display* linka, a druga kopija signala na osobno računalo korištenjem *Null* linka, kako bi se poslani signal mogao pohraniti u obliku video zapisa na osobnom računalu. Video signal poslan na računalo korištenjem *Null* linka se šalje preko mreže u YUV formatu, te se na računalu pohranjuje kao binarna datoteka (engl. *bin file*). Kako bi se snimljeni video signal mogao otvoriti koristi se program naziva *YUV Player*. *YUV Player* omogućava pokretanje i prikaz video datoteka snimljenih pomoću ADAS razvojne ploče u YUV formatu slike. Osim pokretanja i prikaza video zapisa *YUV Player* omogućava i spremanje odabranih okvira video signala, te su tako dobivene referentna i testna slika koje se koriste za proračun matrice homografije.

Implementirano rješenje za ADAS razvojnu ploču neće vršiti proračun matrice homografije, već samo kompenzaciju otklona položaja kamere i rekonstrukciju praznih elemenata slike, a matrica homografije će se proračunavati pomoću programskog rješenja implementiranog na osobnom računalu. Proračun matrice homografije se ne vrši na ADAS razvojnoj ploči kako bi se smanjilo vrijeme potrebno za izvođenje programskog rješenja implementiranog na ADAS

razvojnoj ploči. Postupak kreiranja dijagrama toka objašnjen je u točki 3.4.3. prilikom kreiranja dijagrama toka slučaja upotrebe za implementirano programsko rješenje.




Sl. 3.19. Dijagram toka slučaja upotrebe za snimanje referentne i testne slike.

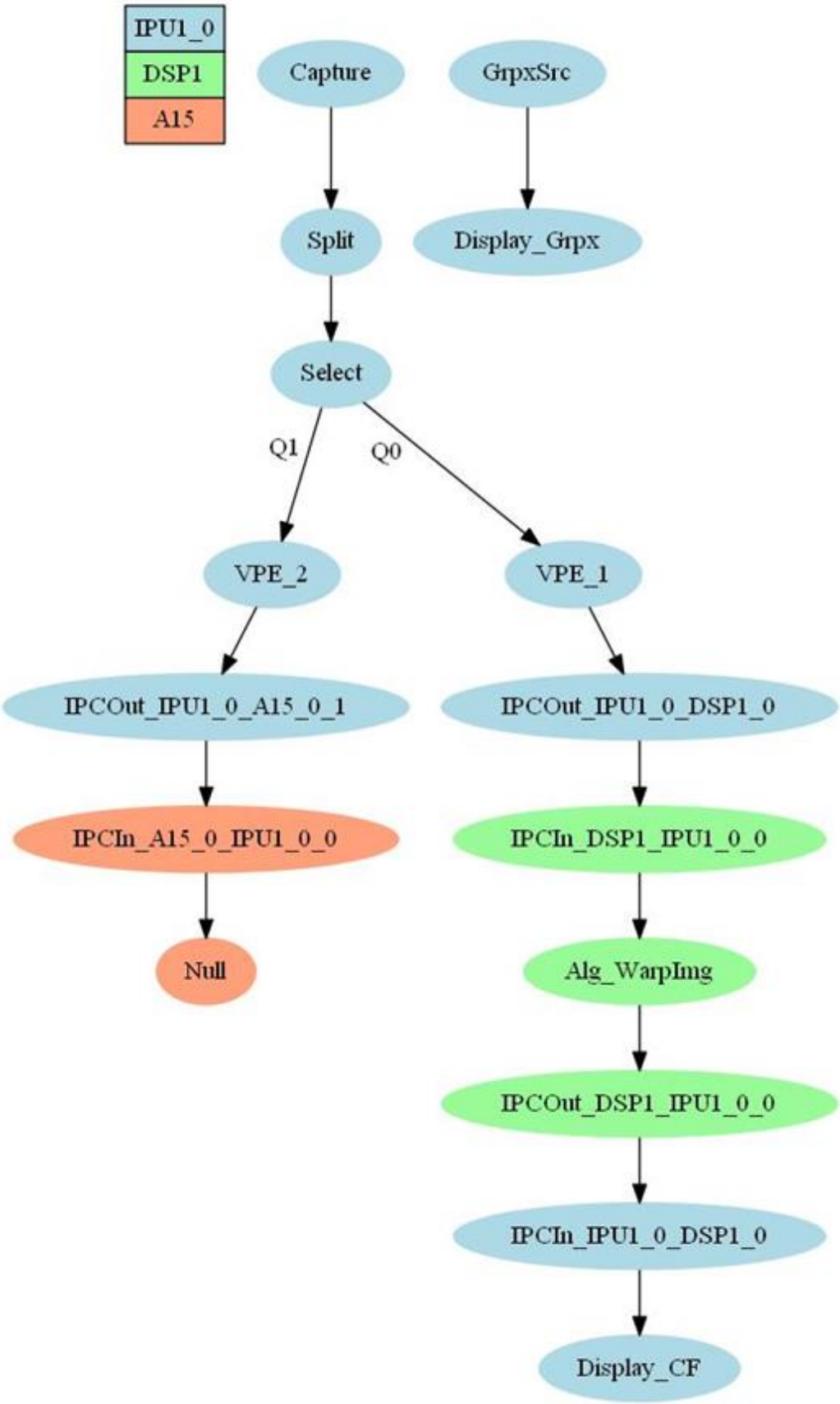
### 3.4.2. Dodavanje vlastitog algoritma

Za uspješnu implementaciju algoritma za detekciju otklona položaja kamere i kompenzaciju položaja kamere potrebno je kreirati novi algoritam i dodati ga u izgradnju. Prvi korak je unutar mape u kojoj se nalaze svi algoritamski linkovi kreirati novu mapu naziva *warpImg* i unutar kreirane mape kreirati nekoliko potrebnih datoteka. Prva datoteka koja se kreira je *warpImg.c* datoteka unutar koje se nalazi implementacija rješenja za korekciju otklona položaja kamere. Zatim je kreirana zaglavna datoteka *iWarpImg.h* unutar koje se nalaze prototipi funkcija

životnog ciklusa (engl. *life cycle*) algoritma. Sljedeće datoteke koje je potrebno dodati su *warpImgLink_algPlugin.c* i popratna zaglavna datoteka *warpImgLink_priv.h*. Unutar *warpImgLink_algPlugin.c* datoteke se nalaze funkcije životnog ciklusa linka, te se unutar zaglavne datoteke *warpImgLink_priv.h* nalaze prototipi funkcije životnog ciklusa linka. Na kraju je kreirana *SRC_FILES.MK* datoteka unutar koje se nalaze nazivi datoteka koje će se dodati prilikom izgradnje programskog rješenja. Nakon što su sve datoteke uspješno kreirane započeta je implementacije programskog rješenja.

### 3.4.3. Dijagram toka slučaja upotrebe – korekcija otklona položaja kamere

Prvi korak prilikom implementacije predloženog rješenja je kreiranje dijagrama toka slučaja upotrebe sa svim potrebnim linkovima i algoritmima. Rješenje je implementirano na SCV SoC-u i sastoji se od nekoliko linkova i jednog algoritma. Na slici 3.20. nalazi se kreirani dijagram toka slučaja upotrebe. Za kreiranje dijagrama toka slučaja upotrebe se koristi tekstualni dokument unutar kojeg se nalazi opis dijagrama toka slučaja upotrebe. Tekstualni dokument je potrebno kreirati unutar *VisionSDK* direktorija koji se nalazi na putanji *VisionSDK/vision_sdk/examples/tda2xx/src/usecases/*. Na navedenoj putanji se kreira nova mapa, te se unutar kreirane mape kreira nova tekstualna datoteka unutar koje se nalazi opis dijagrama toka slučaja upotrebe. Nakon toga je uz pomoć *Windows* konzole pokrenut generator *VisionSDK* slučaja upotrebe. Generator pomoću kreiranog tekstualnog dokumenta kreira potrebne veze i linkove. Prvi link je *Capture* link koji snima video s kamere i šalje ga prema *Split* linku, koji dobiveni video s kamere šalje prema *VPE_1* i *VPE_2* linkovima. Na SVC SoC-u se moraju koristiti dvije kamere istovremeno kako bi se SoC mogao koristiti, no obrađuje se video samo s jedne kamere. Kako bi se doskočilo nastalom problemu, video koji snima druga kamera je poslan na *Null* link kako se ne bi obrađivao. *VPE_1* link vrši promjenu formata slike u *YUV 420P* [22], te tako priprema dobiveni video s kamere za obradu *Alg_WarpImg* algoritmom koji pomoću matrice homografije vrši korekciju otklona položaja kamere, te se na samom kraju obrađeni video šalje na zaslon za prikaz pomoću *Display_CF* linka. Na slici se mogu vidjeti *IPU1_0, DSP1* i *A15* korišteni procesori.




Sl. 3.20. Dijagram toka slučaja upotrebe implementiranog programskog rješenja.

### 3.4.4. Implementacija algoritma za korekciju otklona kamere

Predloženo programsko rješenje za ADAS razvojnu ploču implementirano je pomoću *C* programskog jezika. Cijeli algoritam predloženog rješenja se nalazi unutar *warpImg.c* datoteke.

Kreirano programsko rješenje se sastoji od nekoliko koraka. Na samom početku pokreće se funkcija *Alg_Warp_ImgCreate* unutar koje se vrši dinamička alokacija memorije svih potrebnih matrica koje se koriste. Navedena funkcija će se pokrenuti i izvršiti samo jednom prilikom pozivanja kreiranog algoritma, te se unutar navedene funkcije vrši alokacija memorije za matrice homografije i njen inverz. Također se unutar *Alg_Warp_ImgCreate* funkcije vrši inicijalizacija matrica koje će se koristiti za obradu okvira slike. Unutar funkcije *Alg_WarpImgCreate* izvršen je prolazak kroz sve elemente slike pomoću dvije *for* petlje. Budući da se koristi YUV format za obradu okvira pomoću dva *if* grananja se provjerava koja komponenta YUV elementa slike se obrađuje, te se dohvaćaju *x* i *y* koordinate svakog elementa i proračunavaju nove koordinate pomoću funkcije *warpImg*. Funkcija *warpImg* kao parametre prima tri polja. Unutar prvog polja se nalazi matrica homografije, unutar drugog koordinate trenutnog elementa slike, a treće polje je međuspremnik unutar kojeg se pohranjuju novo-proračunate koordinate trenutnog elementa slike. Funkcija u međuspremnik pohranjuje elemente slike na novo-proračunate koordinate. Prilikom pohrane elemenata slike na njihove nove pozicije vrši se upisivanje infromacija o obrađenim elementima slike unutar *lookup tablice*. Informacije o obrađenim elementima slike se upisuju na iste pozicije unutar *lookup tablice* na koje su unutar međuspremnika pohranjeni elementi testne slike nakon kompenzacije otklona slike. Nakon toga poziva se funkcija *fixMissingPixels* koja kao parametre prima međuspremnik unutar kojeg se nalazi slika s kompenziranim otklonom, međuspremnik s trenutnim okvirom koji se obrađuje i *lookup tablicu*. Funkcija *fixMissingPixels* zatim prolazi kroz sve elemente *lookup tablice* i provjerava je li vrijednost elementa različita od -1. Ako je, na tom mjestu je potrebno izvršiti rekonstrukciju elementa slike u međuspremniku unutar kojeg se nalazi okvir s kompenziranim otklonom kamere. Kao i kod programskog rješenja implementiranog na osobnom računalu, rekonstrukcija se vrši pomoću metode najbližeg susjeda, te je kreirana funkcija *nearestNeighbors* koja kao parametre prima *x* i *y* koordinatu trenutnog elementa slike, testnu sliku, inverznu matricu homografije i međuspremnik unutar kojeg će se pohraniti rekonstruirani element slike. Na samom kraju se poziva funkcija *memcpy* pomoću koje se vrši zamjena trenutnog okvira s okvirom na kojem je kompenziran otklon kamere i izvršena rekonstrukcija elemenata slike koji nedostaju. Kada je algoritam gotov s radom, poziva se funkcija *Alg_WarpImgDelete* koja oslobađa svu alociranu memoriju. U tablici prikazanoj na slici 3.21. se nalazi *warpImg.c* datoteka.

### 3.4.1. Pokretanje predloženog programskog rješenja na realnoj ADAS razvojnoj ploči

Kako bi se kreirano programsko rješenje na realnoj ADAS razvojnoj ploči moglo pokrenuti potrebno je izgraditi rješenje pomoću *VisionSDK* programskog okruženja. Nakon što je programsko rješenje uspješno izgrađeno potrebno je kopirati generirane datoteke na memorijsku karticu uz pomoć koje će se programsko rješenje pokrenuti na ploči. Za komunikaciju između realne ADAS razvojne ploče i računala korištena je univerzalni asinkroni prijemnik-odašiljač (engl. *Universal Asynchronous Receiver-Transmitter* - UART) i *TeraTerm* programski alat. Na ploču je kao izlaz spojen vanjski zaslon.

Linija Kod

```
1:  #include "iWarpImg.h"
2:  #include <src/utils_comm/include/utils_mem.h>
3:  #include "math.h"
4:
5:  UInt8 *outputPtr = NULL;
6:  UInt8 *streamPtr = NULL;
7:  Int8 *lookupTable = NULL;
8:  long double *warpPtr = NULL;
9:  long double *invWarpPtr = NULL;
10:
11: Void setWarpMatrix(long double *warpPtr)
12:
13: Void setInvWarpMatrix(long double *invWarpPtr)
14:
15: Void warpImg(long double *warpPtr, int *oldCrd, int *newCrd)
16:
    Void matrixMultiDouble(long double *warpPtr, int *oldCrd, double
17: *newCrd)
18:
    Void nearestNeighbors(int colIdx, int rowIdx, UInt8 *streamPtr, long
19: double *invMatH, UInt8 *pixel)
20:
    Void fixMissingPixels(UInt8 *outputPtr, UInt8 *streamPtr, Int8
21: *lookupTable)
22:
23: Void setToZero(Int8 *lookupTable)
24:
    Alg_WarpImg_Obj * Alg_WarpImgCreate(Alg_WarpImgCreateParams
25: *pCreateParams)
26:
    Int32 Alg_WarpImgProcess(Alg_WarpImg_Obj *algHandle, UInt8 *inPtr[],
27: UInt32 width, UInt32 height, UInt32 inPitch[], UInt32 dataFormat)
28:
    Int32 Alg_WarpImgControl(Alg_WarpImg_Obj *pAlgHandle,
29: Alg_WarpImgControlParams *pControlParams)
30:
31: Int32 Alg_WarpImgStop(Alg_WarpImg_Obj *algHandle)
32:
33: Int32 Alg_WarpImgDelete(Alg_WarpImg_Obj *algHandle)
```

Sl. 3.21. Izgled *warpImg.c* programske datoteke.

# 4. TESTIRANJE PREDLOŽENOG PROGRAMSKOG RJEŠENJA ZA DETEKCIJU OTKLONA POLOŽAJA KAMERE I KOREKCIJU POLOŽAJA KAMERE

U ovom poglavlju nalazi se pregled provedenog testiranja predloženog programskog rješenja za detekciju otklona položaja kamere i korekciju položaja kamere. Na početku je opisan set slika koji je korišten prilikom testiranja. Zatim je opisan način na koji je testiranje provedeno, te su predstavljeni dobiveni rezultati. Na samom kraju se nalazi osvrt na dobivene rezultate.

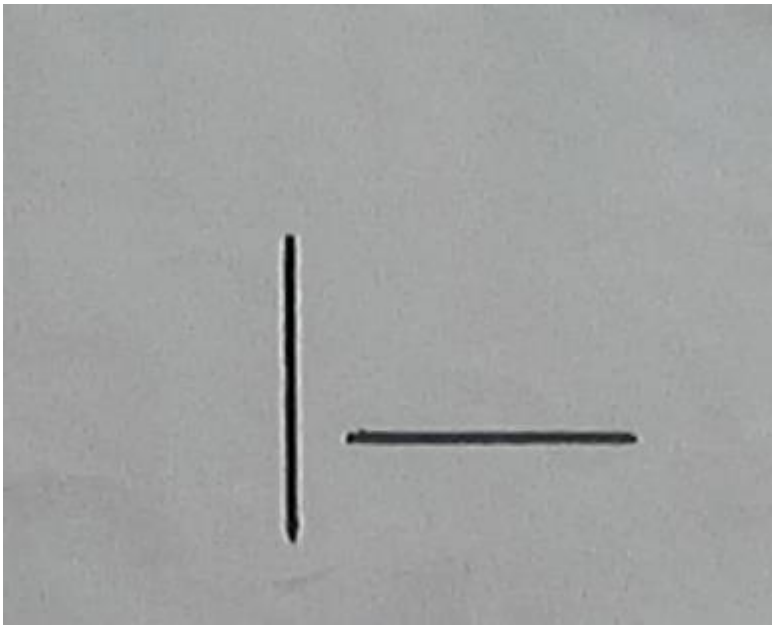
## 4.1. Baza signala nad kojom je provedeno testiranje

Baza korištenih signala je samostalno snimljena pomoću *ADAS ALPHA* razvojne ploče i kamere spojene na ploču. Korišteni signali su okviri rezolucije 1280x720 elemenata slike koji su izvađeni iz video zapisa koji je snimljen u *YUV 420P* formatu boja. Ukupno je korišteno 55 okvira. Okviri su snimljeni tako što je ploča s uzorkom postavljena ispred makete vozila na kojoj se nalazila *ADAS ALPHA* razvojna ploča i korištena automotiv kamera. Otklon položaja kamere je vršen pomakom makete vozila.

Prvi okvir koji je snimljen je referentni okvir (vidi sliku 3.2.), te je nakon njega snimljeno 24 okvira kamerom koja je pomaknuta naprijed, nazad, lijevo, desno, naprijed lijevo, naprijed desno, nazad lijevo i nazad desno za 1, 2 i 3 centimetra. Nakon toga su snimljena još 24 okvira kamerom koja je zarotirana za 5 stupnjeva u desno pri istim translacijama kamere kao i prethodna 24 okvira. Na samom kraju snimljeno je 6 okvira na kojima je kamera zarotirana u lijevu i desnu stranu za 5, 10 i 15 stupnjeva. Okviri su iz video zapisa izvađeni pomoću *YUV Player*-a, te su pohranjeni kao RGB okviri u *bitmap* (engl. *bitmap*) formatu. Video zapisi su snimljeni u zatvorenom prostoru pri uključenim rasvjetnim elementima, te je za provođenje testiranja korištena je ADAS razvojna ploča, *Python* programski jezik i osobno računalo koje se sastoji od *Intel(R) Core(TM) i7-4790* procesora, 16 GB radne memorije i integriranog *Intel(R) HD Graphics 4600* grafičkog procesora.

## 4.2. Provedeno testiranje

Testiranje je provedeno na osobnom računalu pomoću programskog jezika *Python*. Kreirana je programska skripta koja koristi funkcije koje se nalaze unutar *OpenCV* biblioteke, te je vršena usporedba okvira između okvira dobivenih obradom programskog rješenja kreiranog na osobnom računalu u programskom jeziku *Python* (vidi poglavlje 3.2.) i programskog rješenja implementiranog na ADAS razvojnoj ploči (vidi poglavlje 3.4.).



Sl. 4.1. Horizontalna i vertikalna linija.

Na korištenim okvirima su detektirane dvije linije crne boje vidljive na slici 4.1. koje se nalaze na sredini referentne i testne slike od kojih je jedna horizontalna, a druga vertikalna. S linija su dohvaćene po dvije točke (koordinate) te su proračunate jednadžbe pravaca prema izrazu (4-2). Pomoću proračunatih jednadžbi pravaca izvršen je proračun apsolutnih vrijednosti kutova prema izrazu (4-3), određen je kut između horizontalnog pravca na referentnoj slici i horizontalnog pravca na testnoj slici, te vertikalnog pravca na referentnoj slici i vertikalnog pravca na testnoj slici. Vrijednosti kutova dobivene za vertikalne i horizontalne linije su dane u tablicama koje slijede u nastavku poglavlja, te su dobivene vrijednosti kutova različite od 0. U idealnom slučaju navedene vrijednosti odstupanja bi trebale imati vrijednost 0, što bi značilo da je uspješno kompenziran otklon kamere s maksimalnom točnošću. U izrazu (4-1), *k* predstavlja koeficijent smjera pravca pomoću kojeg se može proračunati kut između dva pravca prema izrazu (4-3). Da bi se dobili koeficijenti smjera pravaca, korišten je izraz (4-2) gdje $x_1$ i $y_1$ predstavljaju koordinate prve točke potrebne za proračun jednadžbe pravca, a $x_2$ i $y_2$ koordinate druge točke.

$$y = kx + l \tag{4-1}$$

$$y - y_1 = \frac{y_2 - y_1}{x_2 - x_1}\,(x - x_1) \tag{4-2}$$

$$tg\alpha = \left|\frac{k_2 - k_1}{1 + k_1 k_2}\right| \tag{4-3}$$

## 4.3. Rezultati testiranja

Rezultati testiranja programskog rješenja implementiranog na *ADAS ALPHA* razvojnoj ploči, koje podrazumijeva detekciju otklona položaja kamere i korekciju položaja kamere uspoređeni su s rezultatima programskog rješenja implementiranog na osobnom računalu. Dobiveni rezultati se nalaze unutar tablica koje su dane u nastavku poglavlja, te su vrijednosti unutar tablica izražene u stupnjevima.

### 4.3.1. **Otklon kamere pomakom u jednom smjeru**

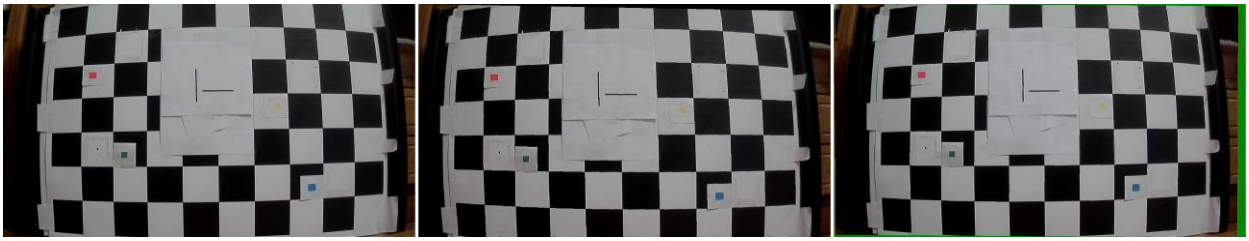
Unutar tablice 4.1. se nalaze rezultati za jednu translacija kamere za okvire koji su obrađeni pomoću programskog rješenja implementiranog na osobnom računalu i rješenja implementiranog na *ADAS ALPHA* razvojnoj ploči. Rezultati dobiveni prema izrazu (4-3) predstavljaju odstupanja kompenzacije otklona kamere između referentne slike i testnih slika obrađenih programskim rješenjem implementiranim na osobnom računalu predstavljenog u poglavlju 3.2. i programskog rješenja implementiranog na ADAS razvojnoj ploči predstavljenog u poglavlju 3.4. Odstupanja su kao što je već ranije rečeno izražena u stupnjevima. Rezultati su dobiveni prema izrazu (4-3).

Tablica 4.1. Odstupanja nastala prilikom kompenzacije otklona kamere u jednom smjeru.

| *Element* | *Python vertikalna* | *ADAS vertikalna* | *Python horizontalna* | *ADAS horizontalna* |
|---|---|---|---|---|
| 1 cm *unazad* | 0.6413 | 0.6282 | 1.2738 | 1.3048 |
| 2 cm *unazad* | 1.2002 | 0.7263 | 0.6437 | 0.7366 |
| 3 cm *unazad* | 1.7189e-07 | 1.3479 | 0 | 0 |
| 1 cm *naprijed* | 0.0816 | 0.0816 | 0.0247 | 0.6740 |
| 2 cm *naprijed* | 1.1935 | 0.6821 | 0.6903 | 0 |
| 3 cm *naprijed* | 0.0283 | 0.1411 | 0 | 1.4144 |
| 1 cm *lijevo* | 0 | 1.8476 | 0.6740 | 0.6662 |
| 2 cm *lijevo* | 2.0454 | 0.7848 | 0 | 0.7538 |
| 3 cm *lijevo* | 0.6095 | 0.6095 | 0.7742 | 1.6847 |
| 1 cm *desno* | 0.6511 | 0.1128 | 0 | 0.9094 |
| 2 cm *desno* | 0.2621 | 0.0715 | 0.7162 | 0.8681 |
| 3 cm *desno* | 1.3804 | 0.6821 | 0 | 0.7742 |
| *Srednje odstupanje* | 0.6744 | 0.6430 | 0.3997 | 0.8155 |

U tablici 4.1. se nalaze rezultati za otklone kamere prema naprijed, nazad, lijevo i desno za 1, 2 i 3 centimetra izraženi u stupnjevima. Na rezultatima je vidljivo da se radi o jako malim

odstupanjima što je očekivano budući da je izvršena samo translacija kamere. Ako promotrimo vrijednosti srednjeg odstupanja za vertikalnu i horizontalnu liniju implementiranih programskih rješenja pri otklonu kamere pomakom u jednom smjeru, vidimo da je srednja vrijednost odstupanja približno jednaka. Na slici 4.2. prikazan je okvir koji je snimljen kamerom otklonjenom u lijevo za 2 centimetra (a), te isti okvir nakon obrade programskim rješenjem implementiranim na osobnom računalu (b) i programskim rješenjem implementiranim na ADAS razvojnoj ploči (c). Na prikazanim okvirima je izvršena kompenzacija otklona položaja kamere i rekonstrukcija praznih elemenata slike.



a) b) c)

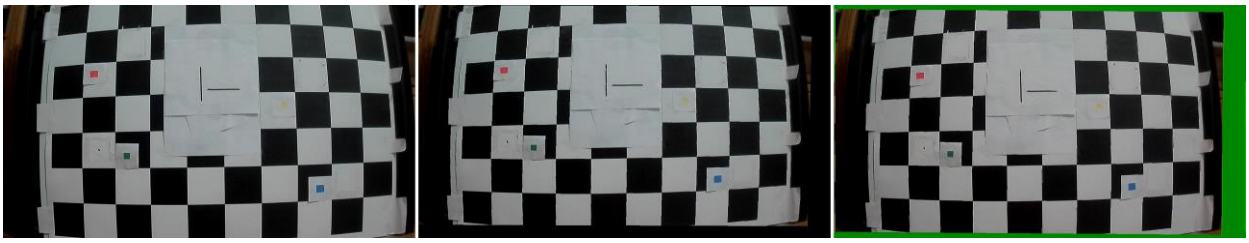
Sl. 4.2. Primjer izvršene kompenzacije otklona položaja kamere i rekonstrukcije slike za otklon kamere u lijevo za 2 centimetra (a) testna slika prije kompenzacije otklona (b) testna slika nakon obrade programskim rješenjem implementiranim na osobnom računalu (c) testna slika nakon obrade programskim rješenjem implementiranim za ADAS razvojnu ploču.

### 4.3.2. Otklon kamere pomakom u dva smjera

U tablici 4.2. se nalaze rezultati dobiveni nakon otklona kamere pri dvije translacije. Kamera je pomicana za 1, 2 i 3 centimetra iz referentnog položaja prema naprijed lijevo, naprijed desno, nazad lijevo i nazad desno. Dobiveni rezultati predstavljeni u tablici 4.2. pokazuju kako su odstupanja za programsko rješenje implementirano na osobnom računalu manjeg iznosa nego u prethodnom slučaju, dok su vrijednosti odstupanja za programsko rješenje implementirano na ADAS razvojnoj ploči nešto većeg iznosa. Rezultati odstupanja dobiveni za rješenje implementirano na ADAS razvojoj ploči su očekivanih vrijednosti budući da je kamera iz referentnog položaja otklonjena u dva smjera, odnosno izvršen je složeniji otklon kamere. Na slici 4.3. prikazan je okvir koji je snimljen kamerom otklonjenom 3 centimetra naprijed i 3 centimetra lijevo (a), te isti okvir nakon obrade programskim rješenjem implementiranim na osobnom računalu (b) i programskim rješenjem implementiranim na ADAS razvojnoj ploči (c). Na prikazanim okvirima je izvršena kompenzacija otklona položaja kamere i rekonstrukcija praznih elemenata slike.

Tablica 4.2. Odstupanja nastala prilikom kompenzacije otklona kamere u dva smjeru.

| *Element* | *Python vertikalna* | *ADAS vertikalna* | *Python horizontalna* | *ADAS horizontalna* |
|---|---|---|---|---|
| 1 cm *unazad* 1 cm *lijevo* | 0.0886 | 0.7113 | 0 | 0.7441 |
| 2 cm *unazad* 2 cm *lijevo* | 0.6903 | 1.1459e-07 | 0.7848 | 0.7848 |
| 3 cm *unazad* 3 cm *lijevo* | 0.7441 | 1.4321 | 0.7538 | 1.7098 |
| 1 cm *unazad* 1 cm *desno* | 0.0593 | 0.1118 | 0.6821 | 0 |
| 2 cm *unazad* 2 cm *desno* | 4.5837e-07 | 0.7539 | 0 | 1.3479 |
| 3 cm *unazad* 3 cm *desno* | 0.6821 | 0.0818 | 0.7957 | 0.7345 |
| 1 cm *naprijed* 1 cm *lijevo* | 0.0621 | 0.6987 | 0.9392 | 1.3479 |
| 2 cm *naprijed* 2 cm *lijevo* | 2.8648e-07 | 1.0886e-06 | 0.6585 | 1.3479 |
| 3 cm *naprijed* 3 cm *lijevo* | 6.3025e-07 | 1.3169 | 0 | 0.7162 |
| 1 cm *naprijed* 1 cm *desno* | 0.6366 | 0.6740 | 0.6740 | 0 |
| 2 cm *naprijed* 2 cm *desno* | 0.6024 | 0.0366 | 0 | 1.2590 |
| 3 cm *naprijed* 3 cm *desno* | 0.6095 | 1.3022 | 0.6437 | 1.2013 |
| *Srednje odstupanje* | 0.3479 | 0.5933 | 0.4943 | 0.9328 |



a) b) c)

Sl. 4.3. Primjer izvršene kompenzacije otklona položaja kamere i rekonstrukcije slike za otklon kamere za 3 centimetra naprijed i 3 centimetra u lijevo (a) testna slika prije kompenzacije otklona (b) testna slika nakon obrade programskim rješenjem implementiranim na osobnom računalu (c) testna slika nakon obrade programskim rješenjem implementiranim za ADAS razvojnu ploču.

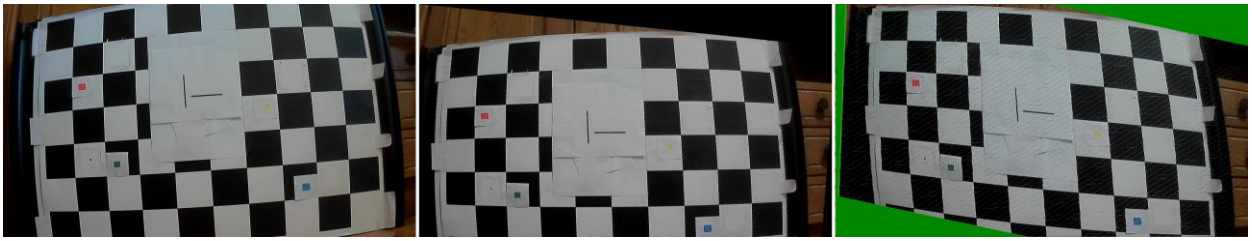
### 4.3.3. Otklon kamere rotacijom

U tablicama 4.3. se nalaze rezultati dobiveni nakon otklona kamere rotacijom u lijevu i desnu stranu, za iznose od 5, 10 i 15 stupnjeva od referentnog položaja. Dobiveni rezultati predstavljeni u tablici 4.3. pri otklonima od 5, 10 i 15 stupnjeva pokazuju da s porastom kuta otklona kamere raste i odstupanje nastalo prilikom kompenzacije otklona položaja kamere. Jednostavnije rečeno, što je veći kut za koji je kamera otklonjena iz referentnog položaja, veće je i odstupanje nastalo prilikom komenzacije otklona. Odstupanje prilikom kompenzacije raste budući da implementirana programska rješenja umjesto dohvaćanja koordinata središnjeg elementa oznake na referentnoj i testnoj slici dohvaćaju koordinate 32. elementa detektirane oznake (vidi poglavlje 3.2.4.).

Tablica 4.3. Odstupanja nastala prilikom kompenzacije otklona kamere rotacijom.

| *Element* | *Python vertikalna* | *ADAS vertikalna* | *Python horizontalna* | *ADAS horizontalna* |
|---|---|---|---|---|
| 5 ° *u lijevo* | 2.2918e-07 | 0.6903 | 1.4712 | 3.5000 |
| 10 ° *u lijevo* | 0.6987 | 2.6324 | 0.7639 | 7.3996 |
| 15 ° *u lijevo* | 5.7296e-08 | 3.9005 | 0 | 8.8068 |
| 5 ° *u desno* | 4.0107e-07 | 3.0128 | 0.8069 | 3.1840 |
| 10 ° *u desno* | 2.6026 | 4.7636 | 0 | 5.1944 |
| 15 ° *u desno* | 3.1775 | 5.0178 | 0.6662 | 7.5494 |
| *Srednje odstupanje* | 1.0800 | 3.3362 | 0.6180 | 5.9390 |

Programsko rješenje impelentirano na ADAS razvojoj poči ima veće vrijednosti srednjeg odstupanja od programskog rješenja implementiranog na osobnom računalu. Na slici 4.4. prikazan je okvir koji je snimljen kamerom otklonjenom za 10 stupnjeva u lijevo (a), te isti okvir nakon obrade programskim rješenjem implementiranim na osobnom računalu (b) i programskim rješenjem implementiranim na ADAS razvojnoj ploči (c). Na prikazanim okvirima je izvršena kompenzacija otklona položaja kamere i rekonstrukcija praznih elemenata slike.

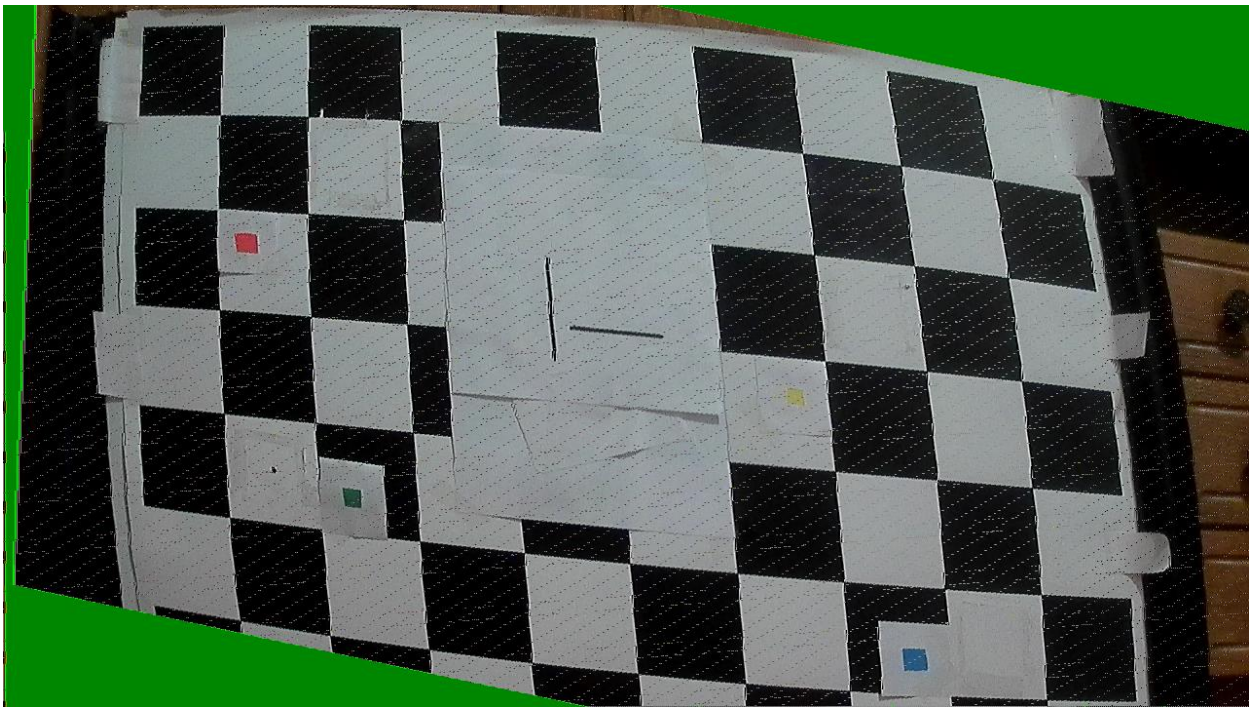


a) b) c)

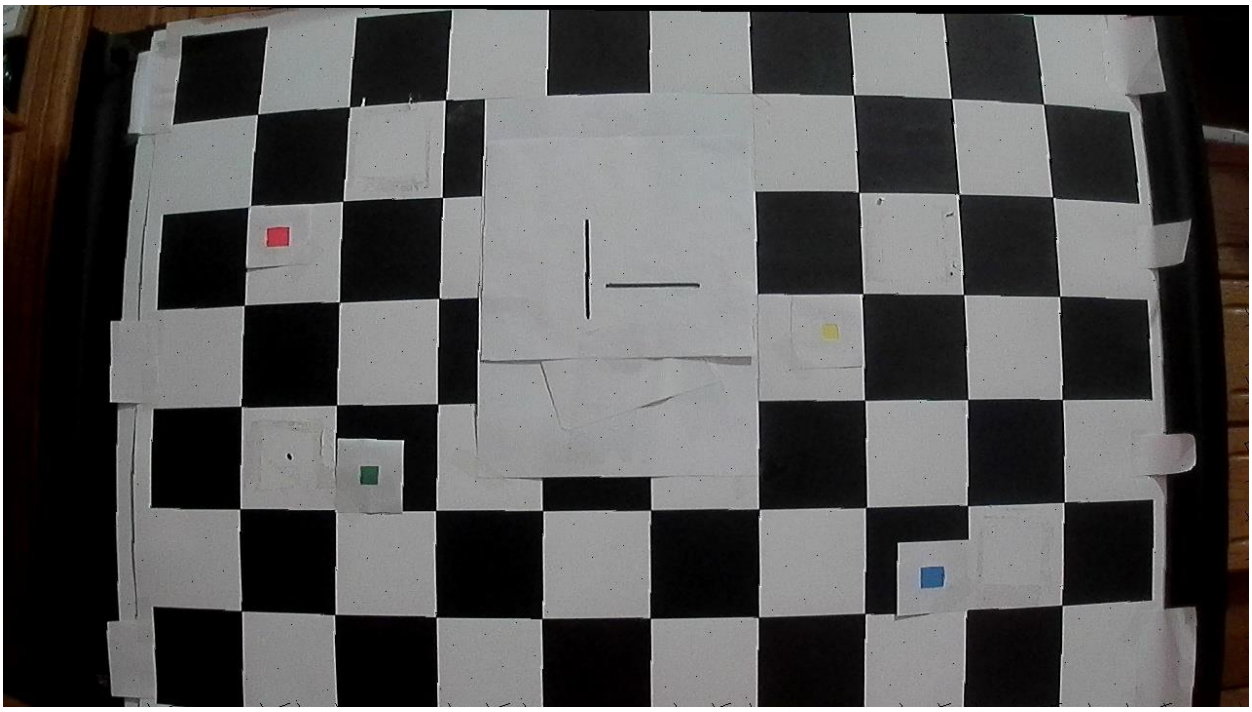
Sl. 4.4. Primjer izvršene kompenzacije otklona položaja kamere i rekonstrukcije slike za otklon kamere 10 stupnjeva u desno (a) testna slika prije kompenzacije otklona (b) testna slika nakon obrade programskim rješenjem implementiranim na osobnom računalu (c) testna slika nakon obrade programskim rješenjem implementiranim za ADAS razvojnu ploču.

Ako na slici 4.4. malo bolje promotrimo sliku (c), moguće je primijetiti kako programsko rješenje implementirano na ADAS razvojnoj ploči nije u potpunosti izvršilo rekonstrukciju praznih elemenata slike. Na slici 4.5. se nalazi uvećana slika 4.4. (c). Programsko rješenje implementirano na osobnom računalu je na testnoj slici 4.4. (a), koja je od referentnog položaja otklonjena rotacijom u desno za 10 stupnjeva, uspješno izvršilo rekonstrukciju praznih elemenata slike, no na slici 4.6. je moguće vidjeti sliku koja je od referentnog položaja otklonjena za 3 centimetra unazad gdje prazni elementi slike nisu u potpunosti rekonstruirani. Iz navedene slike je vidljivo da programskog rješenje implementirano na osobnom računalu također sadrži istu manu kao i programsko rješenje implementirano na ADAS razvojnoj ploči. Mogući razlog navedenog

problema je loša implementacija interpolacijske metode najbližeg susjeda predstavljene u poglavlju 3.2.7.



Sl. 4.5. Testna slika nakon obrade programskim rješenjem implementiranim za ADAS razvojnu ploču.



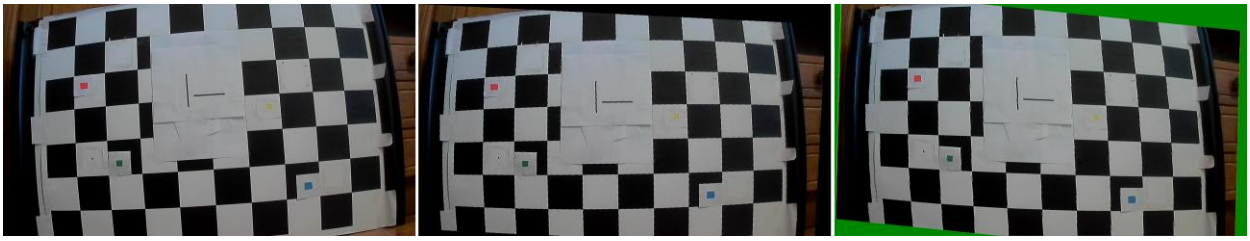
Sl. 4.6. Testna slika otklonjena za 3 centimetra unazad nakon obrade programskim rješenjem implementiranim na osobnom računalu.

### 4.3.4. Otklon kamere rotacijom i pomakom kamere u jednom smjeru

U tablici 4.4. se nalaze rezultati za otklone prilikom jedne rotacije i jedne translacije kamere. Kamera je zarotirana 5 stupnjeva u desno, te je zatim otklonjena prema naprijed, nazad, lijevo i desno za 1, 2 i 3 centimetra. Iz rezultata je moguće vidjeti da je razlika sada znatno veća nego kod prva dva slučaja gdje su izvršene samo translacije kamere u odnosu na referentnu sliku. Razlog navedenog porasta vrijednosti otklona je radi složenijeg otklona položaja kamere (kamera je rotirana te je vršen pomak kamere u jednom smjeru), te vršene aproksimacije središnjeg elementa slike. Ranije je rečeno da se za *x* i *y* koordinate pomoću kojih se vrši proračun matrice homografije uzimaju koordinate 32. elementa značajnog objekta (crvene, žute zelene i plave oznake), te da se tako vrši aproksimacija središnjeg elementa unutar značajnog objekta. Na slici 4.7. prikazan je okvir koji je snimljen kamerom otklonjenom za 5 stupnjeva u desno i 2 centimetra lijevo (a), te isti okvir nakon obrade programskim rješenjem implementiranim na osobnom računalu (b) i programskim rješenjem implementiranim na ADAS razvojnoj ploči (c). Na prikazanim okvirima je izvršena kompenzacija otklona položaja kamere i rekonstrukcija praznih elemenata slike.

Tablica 4.4. Odstupanja nastala prilikom kompenzacije otklona kamere rotacijom i pomakom u jednom smjeru.

| *Element* | *Python vertikalna* | *ADAS vertikalna* | *Python horizontalna* | *ADAS horizontalna* |
|---|---|---|---|---|
| 5 ° *u lijevo* 1 cm *unazad* | 1.3229 | 2.5154 | 0.6511 | 2.0214 |
| 5 ° *u lijevo* 2 cm *unazad* | 2.5857 | 3.3715 | 1.3528 | 1.3539 |
| 5 ° *u lijevo* 3 cm *unazad* | 1.4669 | 1.9417 | 0.6662 | 2.6324 |
| 5 ° *u lijevo* 1 cm *naprijed* | 0.0067 | 1.3751 | 0.6740 | 3.3637 |
| 5 ° *u lijevo* 2 cm *naprijed* | 0.0137 | 1.2038 | 0.6437 | 3.2892 |
| 5 ° *u lijevo* 3 cm *maprijed* | 0.5960 | 1.7808 | 0.6821 | 2.1476 |
| 5 ° *u lijevo* 1 cm *lijevo* | 1.3169 | 3.2520 | 0.6585 | 2.8271 |
| 5 ° *u lijevo* 2 cm *lijevo* | 0.0128 | 1.2835 | 0 | 2.0454 |
| 5 ° *u lijevo* 3 cm *lijevo* | 0.6161 | 0.6713 | 0.6585 | 2.0953 |
| 5 ° *u lijevo* 1 cm *desno* | 1.1992 | 1.4857 | 0.6987 | 2.8336 |
| *5 ° u lijevo* 2 cm *desno* | 0.0479 | 1.3239 | 0 | 1.3639 |
| 5 ° *u lijevo* 3 cm *desno* | 1.8677 | 2.6630 | 1.3931 | 2.5174 |
| *Srednje odstupanje* | 0.9210 | 1.9056 | 0.6732 | 2.3742 |



a) b) c)

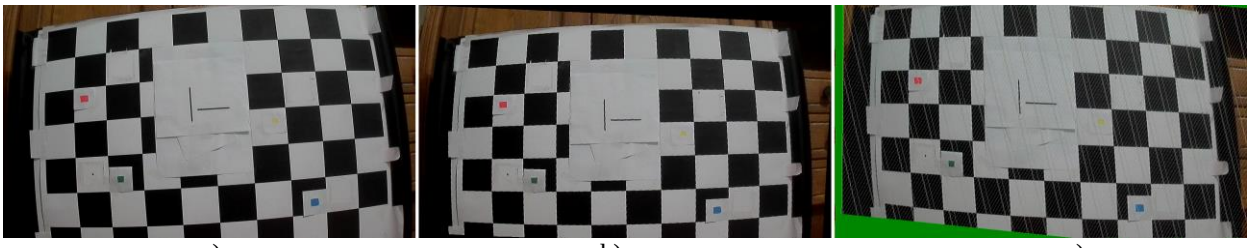
Sl. 4.7. Primjer izvršene kompenzacije otklona položaja kamere i rekonstrukcije slike za otklon kamere 5 stupnjeva u desno i 2 centimetra u lijevo (a) testna slika prije kompenzacije otklona (b) testna slika nakon obrade programskim rješenjem implementiranim na osobnom računalu (c) testna slika nakon obrade programskim rješenjem implementiranim za ADAS razvojnu ploču.

### 4.3.5. Otklon kamere rotacijom i pomakom u dva smjera

U tablici 4.5. se nalaze rezultati dobiveni pri pomaku kamere za 1, 2 i 3 centimetra prema naprijed lijevo, naprijed desno, nazad lijevo i nazad desno i rotaciji od referentnog položaja za 5 stupnjeva. Odstupanja koja se nalaze u tablici 4.5. predstavljaju vrijednosti najsloženijih testiranih odstupanja budući da je kamera rotirana u jednom smjeru, te otklonjena od referentnog položaja u dva smjera. Dobivene vrijednosti srednjeg odstupanja su očekivane budući da se na testnim slikama vrši kompenzacija rotacije i otklona kamere u dva smjera. Slika 4.8. prikazuje okvir snimljen kamerom koja je rotirana u desno za 5 stupnjeva, te pomaknuta za 3 centimetra unazad i 3 centimetra lijevo u odnosnu na referentni položaj kamere (a), te isti okvir nakon obrade programskim rješenjem implementiranim na osobnom računalu (b) i programskim rješenjem implementiranim na ADAS razvojnoj ploči (c). Na prikazanim okvirima je izvršena kompenzacija otklona položaja kamere i rekonstrukcija praznih elemenata slike.

Tablica 4.5. Odstupanja nastala prilikom kompenzacije otklona kamere rotacijom i pomakom u jednom smjeru.

| *Element* | *Python vertikalna* | *ADAS vertikalna* | *Python horizontalna* | *ADAS horizontalna* |
|---|---|---|---|---|
| 5 ° *u lijevo* 1 cm *unazad* 1 cm *lijevo* | 0.6428 | 0.8247 | 0.6987 | 2.7927 |
| 5 ° *u lijevo* 2 cm *unazad* 2 cm *lijevo* | 0.6438 | 2.4896 | 0.6740 | 2.2605 |
| 5 ° *u lijevo* 3 cm *unazad* 3 cm *lijevo* | 0.6585 | 2.0454 | 0.7538 | 2.2906 |
| 5 ° *u lijevo* 1 cm *unazad* 1 cm *desno* | 1.3468 | 2.4853 | 2.5734 | 2.5448 |
| 5 ° *u lijevo* 2 cm *unazad* 2 cm *desno* | 1.5927 | 2.6946 | 1.7493 | 2.2315 |
| 5 ° *u lijevo* 3 cm *unazad* 3 cm *desno* | 1.9730 | 2.0334 | 1.6740 | 2.0214 |
| 5 ° *u lijevo* 1 cm *naprijed* 1 cm *lijevo* | 1.7135 | 2.4501 | 2.3904 | 2.5911 |
| 5 ° *u lijevo* 2 cm *naprijed* 2 cm *lijevo* | 1.3451 | 2.8473 | 0.6366 | 1.3979 |
| 5 ° *u lijevo* 3 cm *naprijed* 3 cm *lijevo* | 1.9979 | 3.2705 | 1.4155 | 0.7250 |
| 5 ° *u lijevo* 1 cm *naprijed* 1 cm *desno* | 2.3388 | 2.0958 | 0.7252 | 1.5353 |
| 5 ° *u lijevo* 2 cm *naprijed* 2 cm *desno* | 1.1604 | 1.7037 | 0 | 2.0454 |
| 5 ° *u lijevo* 3 cm *naprijed* 3 cm *desno* | 1.8087 | 2.0454 | 0.7162 | 3.9182 |
| *Srednje odstupanje* | 1.4352 | 2.2488 | 1.1673 | 2.1962 |



a) b) c)

Sl. 4.8. Primjer izvršene kompenzacije otklona položaja kamere i rekonstrukcije slike za otklon kamere 5 stupnjeva u desno, 3 centimetra unazad i 3 centimetra lijevo (a) testna slika prije kompenzacije otklona (b) testna slika nakon obrade programskim rješenjem implementiranim na osobnom računalu (c) testna slika nakon obrade programskim rješenjem implementiranim za ADAS razvojnu ploču.

### 4.3.6. Vrijeme izvođenja kreiranog programskog rješenja

Vremena izvođenja programskog rješenja implementiranog na osobnom računalu i ADAS razvojnoj ploči su izmjerena i predstavljena u tablici 4.6. Vrijeme izvođenja programskog rješenja implementiranog u programskom jeziku *Python* na osobnom računalu je izmjereno korištenjem programske biblioteke *timeit* [23] i funkcije *timeit*, a vrijeme izvođenja programskog rješenja implementiranog na ADAS razvojnu ploču pomoću biblioteke *time.h* [24] i funkcije *clock()*. Dobivena vremena izvođenja implementiranih programskih rješenja su izražena u sekundama. Kao što je vidljivo u tablici 4.6. vrijeme potrebno za obradu jednog okvira pomoću programskog

rješenja implementiranog na osobnom računalu je 19.8966 sekundi, a programskog rješenja implementiranog na ADAS razvojnoj ploči 13.9276 sekundi što nije dovoljno brzo za primjenu u sustavima stvarnog vremena (engl. *real time systems*). U okviru izrade programskog rješenja implementiranog na ADAS razvojnoj ploči, jedan od zadataka je bio postići funkcionalnost rada implementiranog programskog rješenja u stvarnom vremenu, no navedena funkcionalnost nije postignuta budući da nije izvršena optimizacija predloženog algoritma i njegove implementacije.

Tablica 4.6. Vremena izvođenja implementiranih programskih rješenja.

| *Element* | *Vrijeme izvođenja [s]* |
|---|---|
| *Programsko rješenje implementirano na osobnom računaluu* | 19.8966 |
| *Programsko rješenje implementirano na ADAS razvojnoj ploči* | 13.9276 |

## 4.4. Osvrt na dobivene rezultate

Dobiveni rezultati su pokazali kako programsko rješenje implementirano na *ADAS ALPHA* razvojnoj ploči vrši kompenzaciju otklona kamere pri različitim otklonima koji uključuju rotaciju i translaciju kamere. Na slici 4.5. je moguće vidjeti da prilikom snimanja okvira programsko rješenje implementirano na ADAS razvojnoj ploči nije u potpunosti izvršilo rekonstrukciju praznih elemenata slike, te na slici 4.6. da isti problem postoji i s programskim rješenjem implementiranim na osobnom računalu. Programska rješenja implementirana na osobnom računalu i ADAS razvojnoj ploči ponekad ne izvrše rekonstrukciju praznih elemenata slike u potpunosti, te tijekom izrade rada nije otkriveno koji je uzrok navedenog problema. Od ukupno 55 testnih okvira testiranih na ADAS programskom rješenju, problem se javio na njih 36, što je ukupno 65.45 posto, dok se na programskom rješenju implementiranom na osobnom računalu navedeni problem javio na 18 od ukupno 55 testnih okvira što čini 32.73 posto. Vremena izvođenja programskih rješenja su izmjerena, te oba programska rješenja imaju jako velika vremena izvođenja što ih čini nepogodnima za primjenu u sustavima stvarnog vremena.

# 5. ZAKLJUČAK

U radu je predstavljano kreirano programsko rješenje za problem detekcije otklona položaja kamere i korekciju položaja kamere, te je izvršena implementacija programskog rješenja na ADAS razvojnoj ploči. Ideja predloženog programskog rješenja je da se pomoću matrice homografije izvrši korekcija otklona položaja kamere, te pomoću interpolacije metodom najbližeg susjeda kompenzacija nastalih praznina unutar slike.

Prilikom izrade programskog rješenja prvo je izvršena implementacija na osobnom računalu pomoću *Python* programskog jezika, a zatim na ADAS razvojnoj ploči. Implementacija na osobnom računalu je izvršena kako bi se olakšala implementacija programskog rješenja za ADAS razvojnu ploču. Nakon izvršene implementacije programskih rješenja na osobnom računalu i ADAS razvojnoj ploči provedeno je testiranje. Kreirana programska rješenja su testirana na setu od 55 slika snimljenih samostalno pomoću automotiv kamere rezolucije 1280x720 elemenata slike pri različitim otklonima položaja kamere, te su predstavljeni dobiveni rezultati. Programsko rješenje implementirano na osobnom računalu pokazalo je bolje rezultate od programskog rješenja implementiranog na ADAS razvojnoj ploči. Rezultati dobiveni testiranjem predloženog programskog rješenja za ADAS razvojnu ploču pokazuju kako je na predloženom rješenju potrebno još puno rada. Kompenzacija otklona na programskom rješenju implementiranom na ADAS razvojnoj ploči nije idealna, postoje odstupanja u odnosu na dobivene rezultate pomoću programskog rješenja implementiranog na osobnom računalu. Navedena odstupanja prilikom kompenzacije otklona kamere bi imale manje vrijednosti ako bi se kamera prethodno kalibrirala, te ako bi se izvršila implementacija algoritma koji bi prilikom dohvaćanja koordinata za proračun matrice homografije uzimao koordinate središnjeg elementa značajnog objekta (oznaka definiranih crvenom, žutom, zelenom i plavom bojom).

Jedan od problema koji je primijećen je i rekonstrukcija praznina s okvira, odnosno praznih elemenata slike. Problem pri rekonstrukciji nije u potpunosti razjašnjen, te se još uvijek ne zna zašto se ponekad pojavljuje. Predloženo programsko rješenje implementirano na ADAS razvojnoj ploči koristi interpolaciju metodom najbližeg susjeda, no postoje brojne druge metode interpolacije kao što su linearna i bilinearna metoda interpolacije. Jedno od mogućih poboljšanja bi bila implementacija metode interpolacije koja ima veću efikasnost rekonstrukcije elemenata slike koji nedostaju.

Prilikom testiranja programskog rješenja opaženo je kako se predloženi algoritam ne izvršava u stvarnom vremenu (engl. *realtime*) kao što je to zamišljeno, te bi bilo potrebno napraviti i

optimizaciju programskog koda. Prilikom pokretanja programskog rješenja na zaslonu spojenom na ADAS razvojnu ploču se prikazuje video s niskim brojem okvira u sekundi (engl. *frame rate*). Dobiveni rezultati pokazuju kako implementirano programsko za primjenu u sustavima stvarnog vremena kao što je ADAS nije prihvatljivo budući da je video dobiven s kamere potrebno obrađivati u stvarnom vremenu.

# LITERATURA


[1] „OpenCV: Basic concepts of the homography explained with code“. https://docs.opencv.org/master/d9/dab/tutorial_homography.html (pristupljeno pros. 09, 2020).

[2] „What is Python? Executive Summary“, *Python.org*. https://www.python.org/doc/essays/blurb/ (pristupljeno pros. 09, 2020).

[3] „About“. https://opencv.org/about/ (pristupljeno stu. 09, 2020).

[4] „What is NumPy? — NumPy v1.19 Manual“. https://numpy.org/doc/stable/user/whatisnumpy.html (pristupljeno stu. 09, 2020).

[5] „Color models and color spaces - Programming Design Systems“. https://programmingdesignsystems.com/color/color-models-and-color-spaces/index.html (pristupljeno stu. 06, 2020).

[6] „Color Space: Definition & Conversion“, *Study.com*. https://study.com/academy/lesson/color-space-definition-conversion.html (pristupljeno pros. 09, 2020).

[7] B. AG, „What is the RGB color space?“, *Basler AG*. /en/sales-support/knowledge-base/frequently-asked-questions/what-is-the-rgb-color-space/15179/ (pristupljeno pros. 09, 2020).

[8] „RGB Color Model | How It Work | Uses & Example | Advantages“, *EDUCBA*, lip. 08, 2019. https://www.educba.com/rgb-color-model/ (pristupljeno pros. 09, 2020).

[9] „HSV (Hue, Saturation and Value)“. https://www.tech-faq.com/hsv.html (pristupljeno pros. 09, 2020).

[10] M. Podpora, G. Korba´s, i A. Kawala-Janik, „YUV vs RGB – Choosing a Color Space for Human-Machine Interaction“, *Ann. Comput. Sci. Inf. Syst.*, sv. Vol. 3, ruj. 2014, doi: 10.15439/2014F206.

[11] „geometry - How to compute homography matrix H from corresponding points (2d-2d planar Homography)“, *Mathematics Stack Exchange*. https://math.stackexchange.com/questions/494238/how-to-compute-homography-matrix-h-from-corresponding-points-2d-2d-planar-homog (pristupljeno stu. 03, 2020).

[12] A. Angel, „Nearest Neighbor Interpolation“. https://www.imageeprocessing.com/2017/11/nearest-neighbor-interpolation.html (pristupljeno stu. 27, 2020).

[13] M. Zhang, Y. Hou, i Z. Hu, „Accurate Object Tracking Based on Homography Matrix“, u *2012 International Conference on Computer Science and Service System*, kol. 2012, str. 2310–2312, doi: 10.1109/CSSS.2012.573.

[14] Y. Zhao, S. Wang, J. Wang, i H. Ding, „Linear Solving the Infinite Homography Matrix from Epipole“, u *2010 Second International Conference on Computer Modeling and Simulation*, sij. 2010, sv. 1, str. 92–96, doi: 10.1109/ICCMS.2010.278.

[15] G. YANG, X. CHANG, i Z. JIANG, „A Fast Aerial Images Mosaic Method Based on ORB Feature and Homography Matrix“, u *2019 International Conference on Computer, Information and Telecommunication Systems (CITS)*, kol. 2019, str. 1–5, doi: 10.1109/CITS.2019.8862133.

[16] E. Vincent i R. Laganiere, „Detecting planar homographies in an image pair“, u *ISPA 2001. Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis. In conjunction with 23rd International Conference on Information Technology Interfaces (IEEE Cat.*, lip. 2001, str. 182–187, doi: 10.1109/ISPA.2001.938625.

[17] L. Kang, Y. Wei, Y. Xie, J. Jiang, i Y. Guo, „Combining Convolutional Neural Network and Photometric Refinement for Accurate Homography Estimation“, *IEEE Access*, sv. 7, str. 109460–109473, 2019, doi: 10.1109/ACCESS.2019.2933635.

[18] „Transpose of a Matrix - Definition, Properties and Examples“, *BYJUS*. https://byjus.com/maths/transpose-of-a-matrix/ (pristupljeno pros. 09, 2020).
[19] „The Programming Language B“. https://www.bell-labs.com/usr/dmr/www/bintro.html (pristupljeno stu. 13, 2020).
[20] „American National Standards Institute - ANSI Home“, *American National Standards Institute - ANSI*. https://www.ansi.org:443/ (pristupljeno stu. 13, 2020).
[21] „ISO - International Organization for Standardization“, *ISO*. https://www.iso.org/home.html (pristupljeno stu. 13, 2020).
[22] „Detailed explanation of the YUV420 data format - Programmer Sought“. https://www.programmersought.com/article/1718160124/ (pristupljeno stu. 30, 2020).
[23] „timeit — Measure execution time of small code snippets — Python 3.9.1rc1 documentation“. https://docs.python.org/3/library/timeit.html (pristupljeno pros. 01, 2020).
[24] „C Library - <time.h> - Tutorialspoint“. https://www.tutorialspoint.com/c_standard_library/time_h.htm (pristupljeno pros. 01, 2020).

# SAŽETAK

U radu je razvijeno i implementirano programsko rješenja za *ADAS ALPHA* razvojnu ploču na temu detekcije otklona položaja kamere i korekcije položaja kamere za primjenu u ADAS algoritmima. Prvo je izvršena implementacija programskog rješenja na osobnom računalu pomoću *Python* programskog jezika, te biblioteke *OpenCV* i *NumPy*. Na osnovu programskog rješenja implementiranog na osobnom računalu izvršena je implementacija na ADAS razvojnoj ploči pomoću programskog jezika *C* i *VisionSDK* programskog okruženja. Kreirano programsko rješenje implementirano na ADAS razvojnoj ploči je potom testirano, te je vršena usporedba rezultata dobivenih pomoću programskog rješenja implementiranog na osobnom računalu i programskog rješenja implementiranog na ADAS razvojnoj ploči. Kreirana rješenja su testirana na setu slika snimljenih pomoću automotiv kamere spojene na ADAS razvojnu ploču, rezolucije 1280x720 elemenata slike. Set slika korišten pri testiranju je samostalno snimljen, te se sastoji od 55 slika snimljenih kamerom otklonjenom od referentnog položaja prema naprijed, nazad, lijevo, desno, naprijed lijevo, naprijed desno, nazad lijevo i nazad desno za 1, 2 i 3 centimetra. Kamera je i rotirana u lijevu i desnu stranu za 5, 10 i 15 stupnjeva. Dobiveni rezultati prilikom testiranja su predstavljeni, te su navedeni detektirani problemi. Za detektirane probleme su predložena neka od mogućih rješenja.

Ključne riječi: ADAS, detekcija otklona položaja kamere, *OpenCV,* računalni vid, *VisionSDK*

## ABSTRACT

The paper develops and implements software solutions for the ADAS ALPHA development board on the topic of camera position deflection detection and camera position correction for application in ADAS algorithms. First, the software solution was implemented on a personal computer using the Python programming language, and the OpenCV and NumPy libraries. Based on the software solution implemented on the personal computer, the implementation was performed on the ADAS development board using the programming language C and the VisionSDK programming environment. The software solution implemented on the ADAS development board was then tested, and the results obtained using the software solution implemented on a personal computer and the software solution implemented on the ADAS development board were compared. The created solutions were tested on a set of images taken using an automotive camera connected to an ADAS development board, with a resolution of 1280x720 image elements. The set of images used in the test was taken independently, and consists of 55 images taken by the camera removed from the reference position forward, backward, left, right, forward left, forward right, back left, and back right by 1, 2, and 3 centimeters. The camera is also rotated to the left and right by 5, 10, and 15 degrees. The results obtained during testing are presented, and the detected problems are listed. Some of the possible solutions have been suggested for the detected problems.



Keywords: ADAS, camera position deviation detection, computer vision, OpenCV, VisionSDK

# ŽIVOTOPIS

Željko Sabo rođen je 13. srpnja 1995. godine u Bjelovaru. 2002. godine upisuje Osnovnu školu Lipik u Lipiku, te je 2010. godine završava i iste godine upisuje Tehničku školu Daruvar u Daruvaru. Srednju školu završava 2014. godine i polaže ispit iz državne mature, nakon kojeg upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku smjer „Računarstvo“. Preddiplomski studij završava 2018. godine, te na istom fakultetu upisuje diplomski studij, smjer „Programsko inženjerstvo“.

_____

Potpis autora