Kalibracija vizijskog sustava korištenjem virtualne projekcije

Leljak, Zvonimir

S v e u č i l i š t e u Z a g r e b u

Fakultet strojarstva i brodogradnje

Smjer: Mehatronika i robotika

Diplomski rad

Zvonimir Leljak

0035174616

Mentor: prof.dr.sc. Bojan Jerbić

Komentor: dipl. Ing. Filip Šuligoj

Zagreb, rujan 2016.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

Diplomski rad

Zvonimir Leljak

0035174646

Mentor: prof. dr. sc. Bojan Jerbić, dipl. ing.

Komentor: dipl. Ing. Filip Šuligoj

Zagreb, 2016.

ZAHVALA

Diplomski rad izradio sam samostalno, služeći se znanjem stečenim tjekom studija, te literaturom i izvorima navedenim na kraju rada.

Zahvaljujem se mentoru prof. dr. sc. Bojanu Jerbiću na stručnim savjetima, strpljenju i potpori tijekom izrade ovog rada.

Također zahvaljujem se komentoru Filipu Šuligoju, dipl. ing., na stručnom vođenju kroz proces izrade diplomskog rada.

Dodatno se zahvaljujem mentoru, komentoru i O.Š. Biograd na Moru na mogućnosti korištenja potrebne opreme za izradu ovog rada.

Konačno, zahvaljujem se svojoj obitelji na potpori i strpljenju u periodu izrade diplomskog rada i općenito na potpori tijekom studiranja.

Velika hvala svima!

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
proizvodno inženjerstvo, računalno inženjerstvo, industrijsko inženjerstvo i menadžment, inženjerstvo materijala i mehatronika i robotika

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum 17 -03- 2016	Prilog
Klasa: 602-04/16-6/3	
Ur.broj: 15-1703-16-186	

DIPLOMSKI ZADATAK

Student: Zvonimir Leljak — Mat. br.: 0035174616

Naslov rada na hrvatskom jeziku: **Kalibracija vizijskog sustava korištenjem virtualne projekcije**

Naslov rada na engleskom jeziku: **Calibration of vision system using virtual projection**

Opis zadatka:

U okviru diplomskog rada potrebno je riješiti problem kalibracije vizijskog sustava s nepoznatim unutarnjim i vanjskim parametrima (komponenata sustava i odnosa s okruženjem), a koji koristi virtualnu projekciju umjesto fizičkog uzorka za kalibraciju.

Potrebno je osigurati automatsku kalibraciju projektora i kamere koji se proizvoljno postavljaju u odnosu na ravnu projekcijsku plohu. Kalibraciju implementirati koristeći OpenCV knjižnicu za obradu referentne i projecirane slike i pronalaženje kalibracijskih uzoraka. Na temelju prepoznavanja uzorka potrebno je izračunati afine transformacije između projektora, kamere i projekcijske plohe. Korištenjem dobivenih afinih transformacija potrebno je oblikovati ispravljenu sliku te utvrditi točnost kalibracije usporedbom ispravljene slike i kalibracijskog uzorka.

Zadatak zadan:	Rok predaje rada:	Predviđeni datum obrane:
14. siječnja 2016.	17. ožujka 2016.	23., 24. i 25. ožujka 2016.

Zadatak zadao: Predsjednik Povjerenstva:

Prof. dr. sc. Bojan Jerbić — Prof. dr. sc. Franjo Cajner

SADRŽAJ

POPIS SLIKA

1. UVOD

1.1. Vizijski sustavi

Vizijski sustavi ili strojni vid danas zahtijevaju ozbiljno razmatranje bilo kojeg proizvođača koji želi unaprijediti kvalitetu svoje automatizirane proizvodnje. Vizijske sustave možemo shvatiti kao "oči računala' 'koje mogu identificirati, proučavati i komunicirati kritične informacije kako bi se eliminirale skupe pogreške, te povećala produktivnost i zadovoljstvo kupaca uslijed konzistentne isporuke kvalitetnog proizvoda. Primarno, ovakvi sustavi se koriste u postupku online inspekcije, naime vizijski sustav može obavljati složene i opetovane monotone zadatke uz veliku brzinu, točnost i konzistenciju. Greške ili devijacije u proizvodnom procesu na taj se način brzo ispituju i premošćuju, što dozvoljavaju brze izmjene u kontroli sustava, umanjujući količinu otpada i skupe zastoje u proizvodnji. Vizijski sustavi nisu isključivo korišteni samo za rješavanje zadataka inspekcijskog tipa već se mogu primijeniti i vodeći robote koji imaju razne zadatke, primjerice skupljanje dijelova, pozicioniranje komponenata, uklanjanje ili dopremanje tekućina ili zavarivanje konstrukcija.

Sustavi strojnog vida raznih su oblika i veličina s ciljem bolje prilagodbe zadatku i primjeni, iako svi imaju iste glavne sastavne elemente. Svaki vizijski sustav ima jedan ili više senzora koji formiraju slike potrebne za izradu analize i uključuju programsko sučelje inspekcije, te procesor koji izvršava definirani program tijeka inspekcije. Dodatno, svi sustavi imaju mogućnost komuniciranja rezultata odgovarajućoj opremi kontrole i praćenja koje vrši operator. vizijskih sustava, Bitno je poznavati značajne razlike između vizijskih sustava obzirom na navedene mogućnosti primjene s ciljem odabira učinkovitog sustava rješavanja određenog procesnog industrijskog problema. Također, važno je odabrati odgovarajuću vrstu osvjetljenja i optike za specifičan problem, u protivnom, nedostatak svijetla ili jak odsjaj mogu ozbiljno ugroziti kvalitetu slike, a što rezultira programom s pogreškama ili nemogućnost obavljanja zadanog programa.

Postoje razne vrste izvedbe vizijskih sustava na tržištu, Najjednostavnija podjela vizijskih sustava je na one s jednim vizijskim senzorom (kamerom) te one s dva ili više senzora. Odluka o upotrebi vrste sustava primarno ovisi o potrebnom broju senzora.

Pri odabiru jednako su bitni i ostali faktori kao specifikacije, cijena i okolina u kojoj sustav mora raditi. Na primjer, pametne kamere izvedbom konstrukcije mogu podnositi rad u štetnim okolinama bolje od sustava s više kamera (*multi-camera*), a *multi-camera* sustavi su jeftiniji i bolje obavljaju složene zadatke.

Razlikovanje vizijskih sustava na drugi način su razlike u jedinici za obrađivanje podataka u uređaju. Za mnoge primjene, kao što je auto industrija, poželjno je imati više nezavisnih inspekcija kroz proizvodnu liniju. Pametne kamere, u tom slučaju dobar su odabir jer mogu same obrađivati podatke, mogu biti programirane za specifične zadatke i prilagodljive su prema potrebama bez ometanja rada drugih uređaja u liniji. Na taj je način obrada podataka podijeljena na više kamera u proizvodnoj liniji. Slično tome, neke proizvodne linije mogu biti bolje riješene centraliziranom obradom podataka. Neke linije imaju od 16 do 32 senzora za finalnu inspekciju složenih sklopova na kraju proizvodnje linije. U tom slučaju sustav s više kamera bi bio bolji odabir jer je jeftiniji i jednostavniji za rukovanje.

Prilikom odabira vizijskog sustava posebnu pozornost treba posvetiti programskom sučelju. Mogućnosti programa moraju odgovarati primjeni, programiranju i potrebama rada senzora. Ako uvjet mogućnosti programa nije ispunjen troškovi i vrijeme za kontroliranje sustava kakav želimo se povećavaju. Za jednostavnu i rutinsku kontrolu određenog sustava treba birati programsko sučelje koje je jednostavno za korištenje, odnosno ne zahtjeva programiranje, uključuje osnovne mogućnosti poput prepoznavanja uzoraka, uspoređivanje uzoraka ili skeniranje bar-koda, te sadrži sučelje za komunikaciju sa sličnim uređajima koristeći standardne protokole. S druge strane, ako su zahtjevi zadatka kompleksni, tada njegovo rješavanje zahtijeva programiranje koje se obavlja u naprednijim programskim paketima koji pružaju dodatnu prilagodljivost i kontrolu.

U svakom slučaju mora se izabrati takav program s kojim se mogu kontrolirati svi vizijski sustavi koji postoje u proizvodnoj liniji u slučaju brzog obavljanja potrebnih prilagodbi uslijed mijenjanja zahtjeva zadatka.

Ljudsko oko može dobro vidjeti u različitim uvjetima osvjetljenja dok strojni vid nije toliko napredan. Zato primijenjeni način osvjetljenja objekta inspekcije mora rezultirati na snimci jasno vidljivim svim željenim dijelovima objekta bez odsjaja ili zamračenja.

Osvjetljenje mora biti konstantno i regulirano. Promjene u refleksiji svjetla trebaju zabilježiti vizijski sustavi, uzrokovane razlikama i promjenama svojstava objekta koji je u inspekciji, a nisu izazvane promjenama izvora svjetlosti. Također je bitno postaviti takvo osvjetljenje koje će bolje izraziti elemente promatranog objekta koji su bitni za inspekciju. Posljedice načina osvjetljenja mogu se vidjeti na primjeru slika teksta. Na desnoj slici osvjetljenje je postavljeno tako da se jasno mogu vidjeti i očitati slova, dok se zbog lošeg osvjetljenja, kako je to prikazano na lijevoj slici, ne mogu očitati slova.

Slika 1. Primjer posljedica postavki osvjetljenja

Pravilnim postavljanjem osvjetljenja inspekcija je brža i točnija. Loše osvjetljenje je, s druge strane, najčešći uzrok grešaka strojnog vida. Generalno, ambijentalno osvjetljenje nije dobro rješenje i neće dati dobre rezultate. Osvjetljenje prostorija i postrojenja može oslabiti, pokvariti se ili može biti prepriječen put osvjetljenja drugim objektima što uzrokuje u vizijskom sustavu pogrešnu interpretaciju.

Svi vizijski sustavi imaju jedan ili više vizijskih senzora ili kamera u slučaju *multi-camera* sustava. Svaki senzor mora imati objektiv za skupljanje svjetla reflektiranog s objekta upravljanja i okoline kako bi se mogla formirati slika na senzoru. Točno izabrani objektiv omogućava dobar pregled okoline (*field of view* - FOV) i objekta promatranja, te prikladnu radnu udaljenost (od okoline i objekta na kojem se vrši inspekcija) na kojoj se uređaj postavlja. Stoga su pregled koji želimo i udaljenost senzora glavni uvjeti za odabir objektiva.

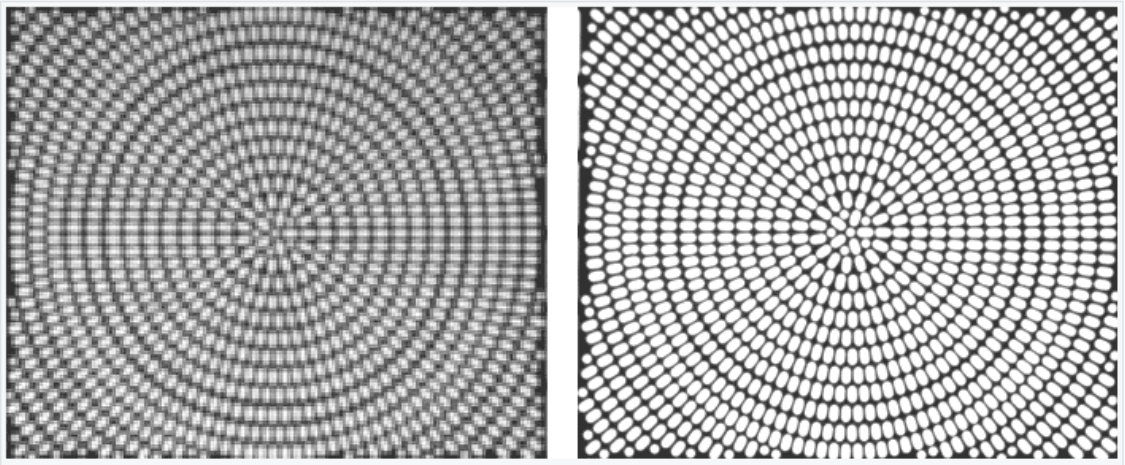
Pregled ili vidno polje može se shvatiti kao područje koje želimo obuhvatiti inspekcijom. Tipični primjer je predmet inspekcije širok pola centimetra i dugačak jedan centimetar za kojega je poželjno imati polje pregleda malo veće od jednog centimetra u koje možemo pozicionirati predmet. Kod odabira preglednosti bitno je razmotriti specifični odnos dužine i visine vizije pojedine kamere. Najčešći odnos senzora koji se koriste za vizijske

sustave je 4:3, takav bi uređaj bio dobar odabir za predmet dimenzija 1 x 0.5 cm koji sam odabrao za primjer, dok bi za predmet okvirnih dimenzija 1 x 0.75 cm bila potrebna kamera s drugačijim odnosom dužine i širine slike.

Radna udaljenost je određena odmakom prednjeg dijela senzora do predmeta inspekcije. U točnijem tumačenju moraju se još uzeti u obzir veličina objektiva i oblik senzora ili kamere.

Iz zahtijevanog područja preglednosti i radne udaljenosti može se odabrati željena fokalna udaljenost objektiva. Fokalna udaljenost je uobičajena specifikacija objektiva i u teoriji označava udaljenost iza objektiva na kojoj se susreću zrake svjetlosti koje prilaze objektivu iz beskonačnosti, odnosno međusobno su paralelne, te prolaze kroz objektiv. Česti iznosi fokalnih udaljenosti objektiva korištenih za strojni vid su 9 mm, 12 mm, 16 mm, 25 mm, 35 mm i 55 mm. Izračunata potrebna fokalna udaljenost čiji rezultat se vjerojatno neće podudarati s ponuđenim objektivima uvjetuje biranje objektiva sa iznosom fokalne udaljenosti najbliže dobivenom rezultatu, a što zahtjeva dodatno podešavanje radne udaljenosti za postizanje željenog područja preglednosti.

Senzori slike vizijskih sustava pretvaraju svjetlost koja prolazi kroz optiku u električne signale. Ti signali pretvaraju se u digitalni oblik i imaju niz vrijednosti koje sačinjavaju piksele i obrađivani su tijekom inspekcije vizijskim sustavom. Rezolucija ili preciznost inspekcije ovisi o radnoj udaljenosti, području preglednosti i broju fizičkih piksela u senzoru slike. Standardni senzori slike imaju 640 x 480 fizičkih piksela u senzoru, a svaki od njih je kvadrat dimenzija 7.4 mikrometra. Iz takve strukture može se izračunati rezolucija izražena u jedinicama duljine što je korisno za inspekciju. Rezolucija u obliku broja i rasporedbe fizičkih piksela je navedena u specifikacijama uređaja.



Slika 2. Primjer kako rezolucija utječe na kvalitetu slike

Senzori slike koje koriste vizijski sustavi su specijalizirani uređaji i prema tome skuplji od primjerice web kamera. Primarno je poželjno imati kvadratne fizičke piksele zbog preciznosti i lakšeg izračuna za mjerenja. Sekundarno, rad senzora slike može biti uzrokovan signalom drugih senzora, poput kontaktnih senzora koji se mogu koristiti za javljanje da je predmet inspekcije na željenom mjestu. Treća bitna prednost za industrijsku primjenu je sofisticirana kontrola izloženosti i brzi elektronski kapci koji mogu uhvatiti sliku predmeta koji se gibaju kroz proizvodnu liniju. Dodatna prednost industrijskih 'pametnih' kamera jest sučelje koje omogućava komunikaciju s drugim senzorima slike i kamerama.

Vrlo je bitno razmotriti kako će predmeti inspekcije biti prezentirani vizijskom sustavu za inspekciju. Ako predmeti nisu prezentirani na konzistentan način, odnosno ako nisu svi, bez izuzetka, prezentirani u odgovarajućim i predviđenim uvjetima, nećemo dobiti željene rezultate. Potrebno je osigurati orijentaciju površine predmeta inspekcije prema senzoru slike koji je predviđen za inspekciju željenog dijela. Nadalje moramo odlučiti hoće li dijelovi biti inspicirani u mirovanju ili tijekom gibanja. Ako se predmet giba bit će potrebno 'zamrznuti' predmet što se postiže ili trenutnim paljenjem i gašenjem svjetla ili korištenjem brzih elektroničkih kapaka koji mogu biti dodatak uređaju. Također u slučaju da se inspekcija vrši tijekom gibanja mora postojati mehanizam uvjetovanja koji će dati informaciju senzoru slike kada uzme uzorak. Taj mehanizam je obično generiran fotografskim senzorom koji detektira kada prednji rub predmeta ulazi u kadar odnosno područje inspekcije. Ako se inspekcija vrši

na predmetu koji je pozicioniran ispred senzora slike primjerice robotskom rukom i tijekom inspekcije miruje, senzor će biti uzrokovan od strane PLC uređaja ili same robotske ruke.

Predmet razmatranja može biti i brzina. Ako se inspiciraju dijelovi pri velikim brzinama potrebno je optimizirati pozicioniranje dijelova kako bi se smanjilo vrijeme obrađivanja podataka. Tako se prilikom konstruiranja vizijskog sustava mora obratiti pažnja da sve komponente imaju vlastite zahtjeve širine frekvencijskog pojasa za uređaj koji obrađuje podatke. Dakle kada se odabire vizijski sustav za inspekciju pri velikim brzinama potrebno je razabrati one komponente koje su nužne od onih koje bi bilo dobro imati.

1.2. Uvod u problem

U ovom radu razrađuje se rješenje problema kalibracije sustava što je presudno za trodimenzijske sustave računalnog vida, a posebno za sustave mjerenja trodimenzijskih oblika. Cijela zadana problematika kalibracije sustava uključuje kalibraciju kamere, što je opsežno proučavano unutar područja tehnoloških znanosti, te nakon kalibracije kamere umjeravanje projektora. Teoretski, projekcija podataka na plohu može se shvatiti kao snimka druge kamere. U praksi, postoje dvije glavne razlike koje čine umjeravanje projektora složenijim od kalibracije kamere. Prva razlika. što projektori ne mogu interpretirati površinu koju osvjetljavaju sa mjerljivom vezom između 2D projicirane točke i 3D osvijetljene točke, u tom slučaju je potrebno koristiti kameru. Druga razlika je onemogućeno računanje koordinate trodimenzijske točke zato što je kalibracijski uzorak projiciran i nije vezan sa koordinatnim sustavom stvarnosti. To objašnjava vrlo mali broj praktičnih metoda za izračunavanje unutarnjih parametara projektora. U ovom radu predstavljena je metoda kalibracije za sustave projektora i kamere kojoj je cilj točnost i jednostavnost za korištenje.

Metoda koju se razrađuje u ovom radu nije površinski bazirana, odnosno cilj metode je isti kao i kod površinski bazirane metode ali kroz postupak ne smijemo znati fizičke značajke iz stvarnog svijeta koje daju informacije o površini projekcije. Naravno, kroz rad će se radi preglednosti cilja i boljeg razumijevanja problema demonstrirati plošno bazirana metoda.

Za rješavanje problema koriste se algoritmi koji se sastoje od više dijelova uvjetno o metodi. Zbog toga što problem rješavamo programski, za rješenja, većine matematičkih problema koriste se gotove matematičke funkcije knjižnice OpenCV u programskom sučelju Python. Glavni proizvedeni algoritam sastoji se od više manjih algoritama koje ćemo moći pozivati ovisno o okruženju u kojem se nalazimo i koji problem želimo riješiti. Glavni i osnovni algoritam je onaj za prepoznavanje rubova i sortiranje točaka pomoću kojega započinjemo svaku analizu slike. Kada su poznate koordinate rubova možemo koristiti druge algoritme koji ispravljaju sliku u okviru plošno bazirane metode, metode bez fizičkih značajki ili nam vraćaju vanjske parametre kamere i projektora.

U poglavlju 2 prikazano je načelo predložene metode. U odjeljcima 3 i 4 opisani su glavni matematički koraci za rješenje problema. U poglavlju 5 simulirano je računanje parametara kamere uz pomoć nasumično generirane orijentacije i translacije slike. Kroz poglavlje 6 razvija se plošno bazirana metoda za ispravljanje slike te računanje vanjskih parametara kamere i projektora. U odjeljku 4 poglavlja 6 počinje razvoj metode bez fizičkih značajki iz okoline. Kroz poglavlje 7 rad uspoređujemo s drugim radovima koji se bave istim problemom te navodimo i ističemo bitne razlike. U poglavlju 8 dani su zaključci sa raspravom o budućem radu i završno u poglavlju 9 dan je programski kod.

2. RAZVOJ ALGORITMA KALIBRACIJE SUSTAVA PROJEKTOR-KAMERA

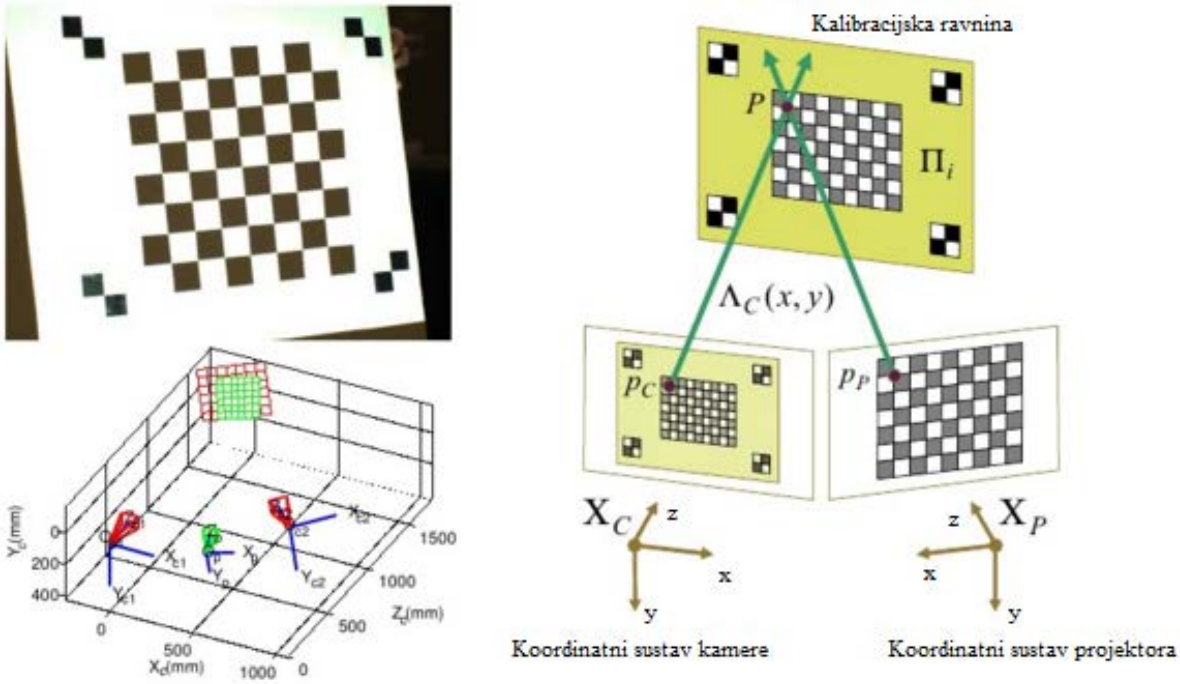
Cilj predstavljene metode je otkrivanje unutrašnjih i vanjskih parametara za kameru i projektor u sustavu. Projekcija i slikanje ili snimanje prikazani su standardnim modelom aperture kamere sa unutarnjim parametrima koji uključuju fokalnu udaljenost, fokusnu točku, faktor izmicanja piksela i ukupni broj piksela, te vanjskim parametrima, odnosno rotacijom i translacijom koordinatnih sustava kamere i projektora u odnosu na koordinatni sustav stvarnog svijeta.

Ključna točka razumijevanja ponuđene metode kalibracije je shvatiti projektor kao suprotnost kamere (odašilje obilježja dvodimenzijske slike u trodimenzijske zrake) što čini proces kalibracije projektora istim kao onog kalibracije kamere. Sa takvim pristupom može se primijeniti bilo koja standardna metoda za kalibraciju kamere za zadatak kalibracije projektora. Glavni zahtjev ove metode je nalaženje trodimenzijskih (položaja) točaka projektiranog uzorka kako bi ih nakon toga mogli koristiti zajedno s podacima dvodimenzijskih (položaja) točaka slike koju projiciramo, a da bi na kraju mogli saznati unutarnje i vanjske parametre projektora.

Predložena metoda kalibracije cijelog sustava podijeljena je u nekoliko koraka:

- Kalibracija kamere korištenjem šahovnice
- Pronalaženje kalibracijske ravnine u koordinatnom sustavu kamere
- Projektiranje uzorka šahovnice na plohu i pronalaženje rubova
- Korištenje algoritama za interpretaciju točaka i nalaženje njihovih trodimenzijskih koordinata
- Kalibracija projektora uz pomoć veza između dvodimenzijskih točaka projiciranog uzorka i nađenih trodimenzijskih točaka

Kroz sljedeće odlomke se detaljnije objašnjavaju postupci ovih glavnih koraka. Prije pristupanju detaljima potrebno je pobliže objasniti okruženje i aparate koje metoda zahtijeva. Slika prikazuje primjer postavljenog sustava za kalibraciju kamere i projektora.



Slika 3. Primjer kalibracijskog sustava kamere i projektora

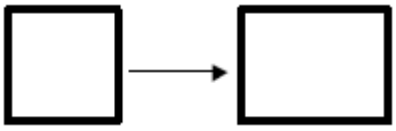
Prije svega potrebni su projektor i kamera, te kalibracijska ploha potrebna za njihovu kalibraciju, a ona je (kalibracijska ploha) ravna površina koja ima na sebi isprintan ili pričvršćen kalibracijski uzorak i prazan bijeli prostor na koji se može projicirati isti taj kalibracijski uzorak. Nakon svega potrebno je odabrati tip i dimenzije kalibracijskog uzorka koji će u ovom slučaju biti crno bijela šahovnica. Bitno je napomenuti da fokus projektora mora biti prilagođen za ispravnu udaljenost kako bi kamera mogla uzeti što kvalitetniji uzorak za analizu.

3. KALIBRACIJA KAMERE

U procesu kalibraciju korištena je Zhangova metoda koja koristi planarni uzorak šahovnice. Planarni uzorak šahovnice koji se za mnoga istraživanja pokazao laganim za korištenje omogućuje jednostavniju konstrukciju dobrog i točnog algoritma za otkrivanje unutarnjih i vanjskih parametara kamere. Algoritam je implementiran u programskom jeziku *Python* u kojem su korišteni OpenCV, NumPy i SciPy moduli. Python programski jezik je jednostavan za korištenje dok je program sporiji od primjerice C++ programskog jezika, ali će za demonstraciju metode biti dovoljno dobar. Najčešće korišteni programski jezici za kalibraciju kamere su C++ s OpenCV knjižnicom i Matlab.

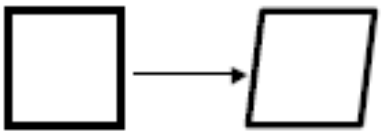
Primarno, kada se radi kalibracija kamere, traže se unutarnje veličine kamere koje utječu na proces snimanja. Rezultat utjecaja unutarnjih parametara može se opisati s nekoliko tipičnih efekata a to su:

- Pozicija centra slike (tipično nije na koordinatama [širina/2 , visina/2])
- Fokalna udaljenost
- Različiti faktori skaliranja za redove i stupce piksela



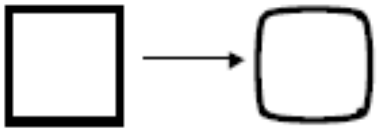
Slika 4. Posljedica faktora skaliranja

- Faktor iskrivljenja



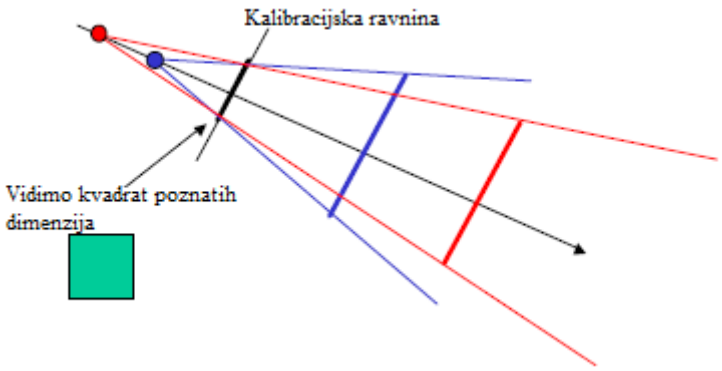
Slika 5. Posljedica iskrivljenja

- Radijalna distorzija



Slika 6. Radijalna distorzija

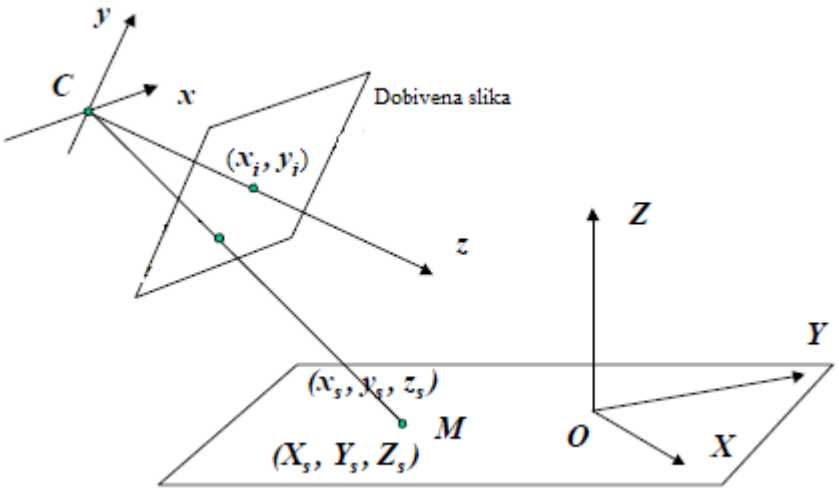
Dobra kalibracija je bitna za zadatke u kojima se mora ostvariti interakcija sa stvarnim svijetom ili konstruirati model nekog objekta. Za takve slučajeve bitna je mogućnost procjene pozicije.



Slika 7. Procjena pozicije uz pomoć kamere

Za Zhangovu metodu kalibracije potrebno je kalibracijski uzorak u obliku šahovnice u različitim orijentacijama postaviti ispred kamere. Nakon snimanja i čišćenja slike primjenjuje se algoritam koji izvlači pronađene rubove projiciranog uzorka kao točke, te ih sortira kako bi se mogle povezati s izvornim točkama slike uzorka, nakon čega dolazimo do parametara transformacije koordinatnih sustava iz sljedećeg izraza.

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$



Slika 8. Prikaz veze između koordinata na slici i u prostoru

U ovom radu korišten je algoritam koji radi tako da traži linearno rješenje najmanjih kvadrata, odnosno linearnu regresiju pravca za kombinacije dužine n+1 na šahovnici s n redova i stupaca, te nakon toga provjerava da li je zbroj udaljenosti od istih točaka do pravca unutar određene veličine i odabire one točke i pravce koji to jesu. Takvim postupkom algoritam mora doći do 2(n+1)+2 pravaca, a zatim bilježi dvije točke koje su najudaljenije iz dobivenih skupova, te ponovo za skupove koji sadrže te točke odabire najudaljenije. Tako smo dobili 4 točke koje označavaju krajnje rubove šahovnice. Taj postupak se ponavlja dok se ne identificiraju sve točke, odnosno rubovi na šahovnici. Nakon što algoritmom prepoznamo i odredimo točke i povežemo ih s uzorkom koji nije distorziran mogu se matematički saznati rješenja unutarnjih i vanjskih parametara kamere u zatvorenoj formi, dok se parametri radijalne distorzije četvrtog reda mogu saznati linearnim rješenjem najmanjih kvadrata. Na kraju je primijenjena nelinearna minimizacija projicirane greške riješena Lavenberg-Marquardtovom metodom, s čime su finalno rafinirani parametri za kameru.

4. RAZVOJ ALGORITMA ZA ODREĐIVANJE POLOŽAJA TOČAKA DEFINIRANIH S TRI DIMENZIJE

Kao što je navedeno, glavni cilj zadatka je nalaženje trodimenzijskih točaka projiciranog kalibracijskog uzorka kako bi se kombinirajući s geometrijom dvodimenzijske slike uzorka moglo kalibrirati projektor uz pomoć kamere kao senzora za uspostavljanje povratne veze. Za zadani cilj potrebno je odraditi sljedeće korake:

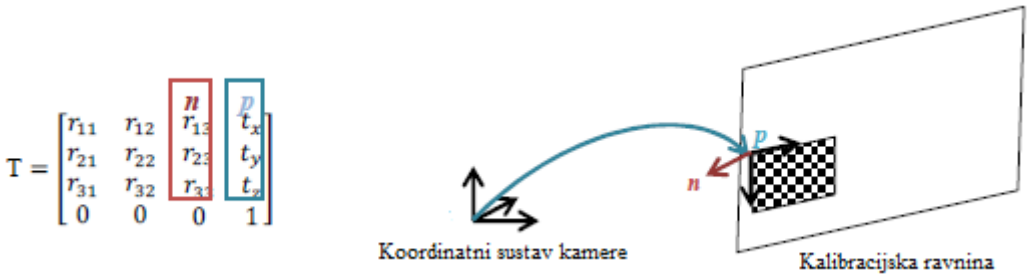
- Računanje jednadžbi koje opisuju kalibracijsku ravninu u odnosu na koordinatni sustav kamere
- Pronalaženje rubova projiciranog kalibracijskog uzorka u obliku šahovnice
- Računanje jednadžbi pravaca koji označavaju zrake svjetlosti koje udaraju u točke
- Računanje križanja pravaca i ploha za određivanje trodimenzijske koordinate za svaku točku

4.1. Računanje jednadžbe kalibracijske ravnine u koordinatnom sustavu kamere

Kad su poznati parametri kamere, jednostavno je računati jednadžbe koordinatnog sustava kalibracijske ravnine u koordinatnom sustavu kamere.

Dobar način za definiranje ravnine u trodimenzijskom prostoru jest pronalaženjem specifične točke i vektora normale na plohu. Do ovog stadija izvršavanja metode kalibracije projektora imamo vanjske parametre kamere u odnosu na gornji lijevi kut kalibracijske ravnine pa se može matematički definirati njena jednadžba.

Translacijski vektor vanjskih parametara (zadnji stupac) daje koordinate ishodišta (označen slovom p) koordinatnog sustava ravnine u koordinatnom sustavu kamere, dok je treći stupac matrice rotacije vektor normale (oznaka n) na ravninu koja na sebi ima pričvršćen kalibracijski uzorak.



Slika 9. Transformacija koordinatnih sustava

Tada ako vektor *p* označava koordinate poznate točke na ravnini, a vektor *n* opisuje vektor normale na ravninu, željena ravnina je skup točaka *r* za koje vrijedi:

$$n \cdot (r - p) = 0$$

Pa se može zapisati:

$$n = a\hat{x} + b\hat{y} + c\hat{z}$$

$$r = x\hat{x} + y\hat{y} + z\hat{z}$$

Gdje su $\hat{x}$, $\hat{y}$ i $\hat{z}$ kartezijski jedinični vektori i definiraju točku *d* kao:

$$d = -n \cdot p$$

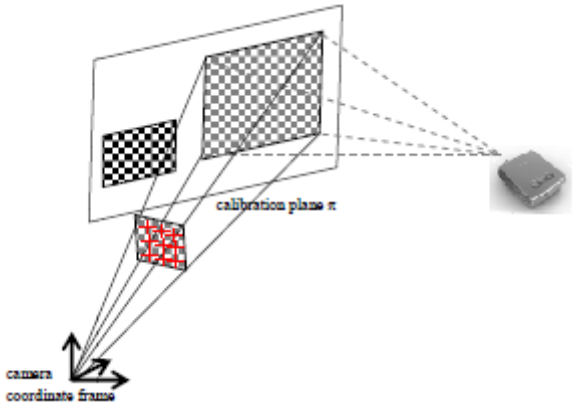
I na kraju izraz koji definira ravninu je:

$$ax + by + cz + d = 0$$

U kojem su a, b, c i d poznati realni brojevi, a od kojih a, b, i c nisu nula.

4.2. Detekcija rubova projicirane šahovnice

Sljedeći korak sastoji se u izvlačenju, odnosno detekciji rubova iz projicirane šahovnice (slika mora biti projicirana u kalibracijskoj ravnini) uslikane kamerom. Taj zadatak se radi u programskom jeziku *Python* koristeći funkciju *CornerHarris* iz OpenCV knjižnice.



Slika 10. Detekcija rubova na slici kalibracijskog uzorka

4.3. Računanje jednadžbi pravaca zraka svjetlosti

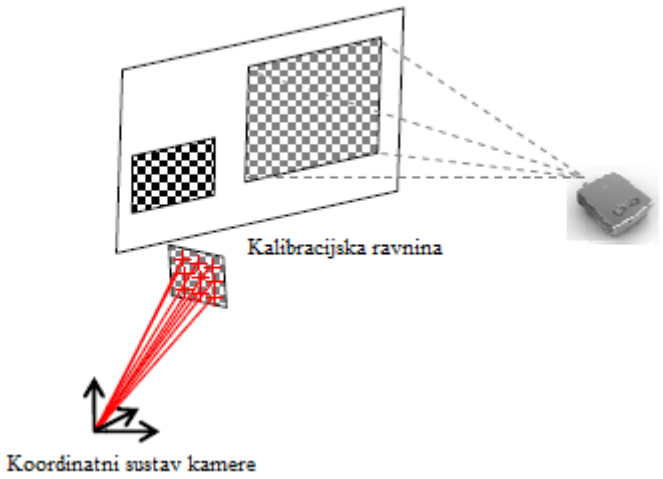
Sljedeći je korak saznati izraze trodimenzijskih zraka ili pravaca koje se odbijaju od rubova šahovnice, čije točke smo doznali u prethodnom koraku i prolaze kroz fokalnu točku kamere.

Za računanje jednadžbi trodimenzijskih pravaca koji prolaze kroz rub označen (C_x, C_y) na ravnini slike koristimo sljedeću transformaciju koja sadrži parametre kamere:

$$\begin{bmatrix} sR_x \\ sR_y \\ sR_z \\ s \end{bmatrix} = [T_u \quad T_v]^{-1} \begin{bmatrix} C_x \\ C_y \\ 1 \end{bmatrix},$$

gdje je vektor zraka (R_x ,R_y ,R_z) skaliran faktorom s.

Na slici 12 se može vidjeti grafička reprezentacija zraka koje računamo. Iako su prikazane s određenom dužinom, u modelu su izračunate s varijablom *s* koja označava udaljenost od ishodišta, te je točno tu varijablu potrebno odrediti kako bi mogli saznati trodimenzijske točke rubova, odnosno potrebno je naći križanja zraka s kalibracijskom ravninom.



Slika 11 Prikaz zraka svjetlosti koje prolaze kroz rubove

4.4. Križanje pravaca zraka svjetlosti s kalibracijskom ravninom

Položaj križanja pravaca zraka svijetlosi računa se u svrhu određivanja trodimenzijskih pozicija projiciranih rubova. U suštini, vrijednosti koje tražimo su one veličine faktora *s* u vektoru pravca, koji opisuje zrake svjetlosti koje prolaze kroz rubove šahovnice kojima definiramo točke na pravcu, a dio su kalibracijske ravnine. Potreban je poznati izraz kalibracijske ravnine:

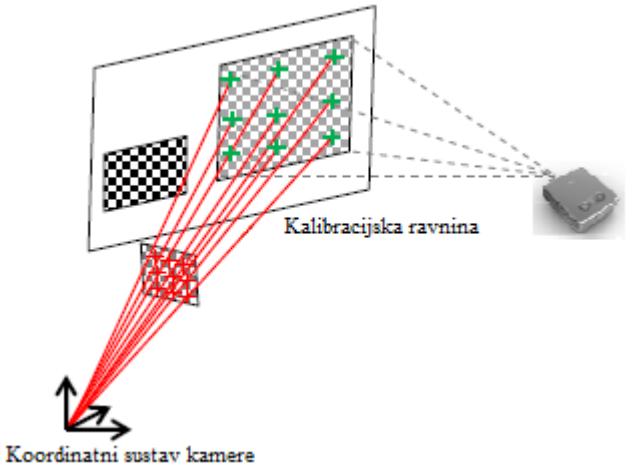
$$ax + by + cz + d = 0$$

Mijenjamo vrijednosti varijabli s onima iz izraza pravca:

$$a(sR_x) + b(sR_y) + c(R_z) + d = 0$$

Kako su *a* ,*b*, *c* poznati parametri kao i R_x, R_y, R_z, može se izračunati varijabla *s*.

Kada imamo veličinu *s*, poznate su nam trodimenzijske pozicije projiciranih rubova šahovnice. Taj postupak se ponavlja za sve specifične točke slike, i to na više slika kako bismo dobili što veći broj točaka za kalibraciju zbog njene točnosti.



Slika 12 Križanja zraka svjetlosti s kalibracijskom ravninom

4.5. Kalibracija projektora

Kao što je spomenuto, projektor može biti opisan istim fizičkim modelom kao i kamera, za razliku od kamere koja bilježi dolazne zrake svjetlosti od strane slike, projektor ih projicira u stvarni svijet. Ako imamo poznate trodimenzijske koordinate projiciranih točaka, kalibracija projektora ima isti postupak kao i kalibracija kamere, koji je objašnjen u prethodnom poglavlju.

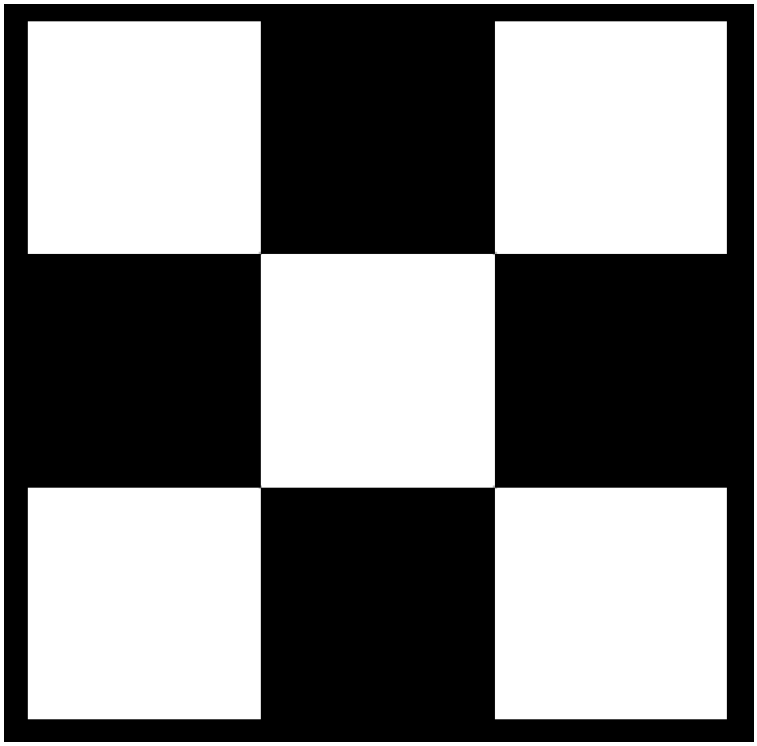
Projicirane trodimenzijske točke procesiraju se kako je navedeno u prethodnom poglavlju. Trodimenzijske koordinate točaka dobiju se iz točaka u dvije dimenzije iz projicirane slike do koje se jednostavno dolazi slikanjem projiciranog kalibracijskog uzorka, te pronalaženjem rubova na šahovnici uzorka. Prije izvršavanja postupka mora se osigurati da je kamerom uslikana slika u istoj rezoluciji u kojoj projektor projicira istu sliku.

Kada su poznate veze između dvodimenzijskih i trodimenzijskih točaka možemo početi kalibraciju projektora koristeći poznatu metodu za kalibraciju kamere. Kao što je predstavljeno u trećem poglavlju, kalibracijska funkcija slijedi Zhangovu metodu i minimizira grešku projekcija slika uz pomoć funkcije najmanjih kvadrata, te tako dolazi do točnijih unutarnjih parametara projektora uključujući radijalnu distorziju i vanjskih parametara, odnosno trodimenzijskih koordinata projektora.

Nakon što je proces kalibracije završen došli smo do unutarnjih i vanjskih parametara projektora. Za procjenu rezultata biti će prikazana greška ponovljene projekcije slike i grafikon, kojim će se pokazati koordinatni sustavi kamere, projektora i kalibracijske ravnine.

5. SIMULACIJA RAČUNANJA PARAMETARA KAMERE

U simulaciji je korišten uzorak šahovnice s karakterističnim dimenzijama izraženim u stvarnim jedinicama, odnosno metrima kako bi se na kraju moglo dobiti stvarnu matricu transformacije za pretpostavljeni slučaj simulacije. Kroz simulaciju se pretpostavlja da kamera ima neodređenu poziciju u odnosu na kalibracijsku plohu, te određenu radijalnu distorziju, pomak centra distorzije i pomak fokusa slike zbog malih proizvodnih grešaka na leći. Slika 14. prikazuje kalibracijski uzorak.

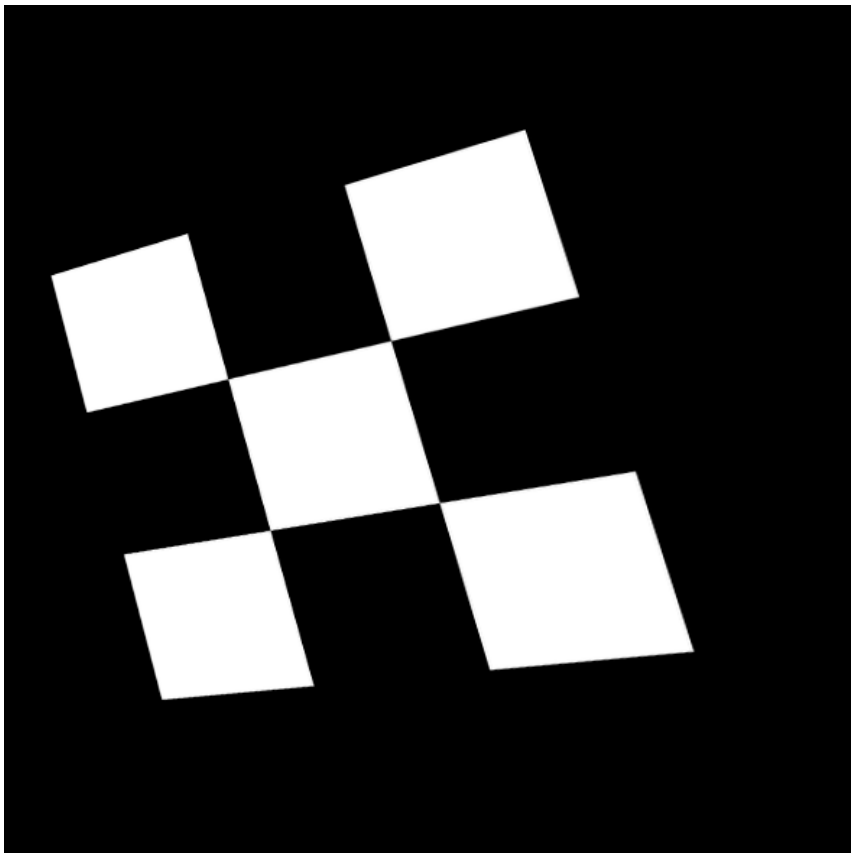


Slika 13. Korišteni kalibracijski uzorak

Pretpostavlja se da je kalibracijski uzorak pričvršćen za kalibracijsku površinu i snima ga se kamerom. Tada je uzorak percipiran kao zakrenut i translatiran u odnosu na koordinatne osi kamere, što je rezultat pomicanja kamere u odnosu na kalibracijsku plohu. Ako se uzme u obzir da se dubina slike mjeri na osi z, simulacija dubine biti će izvršena OpenCV naredbom *cv2.resize* dok se translacije po osima x i y mogu simulirati naredbom *cv2.wrapAlfine*.

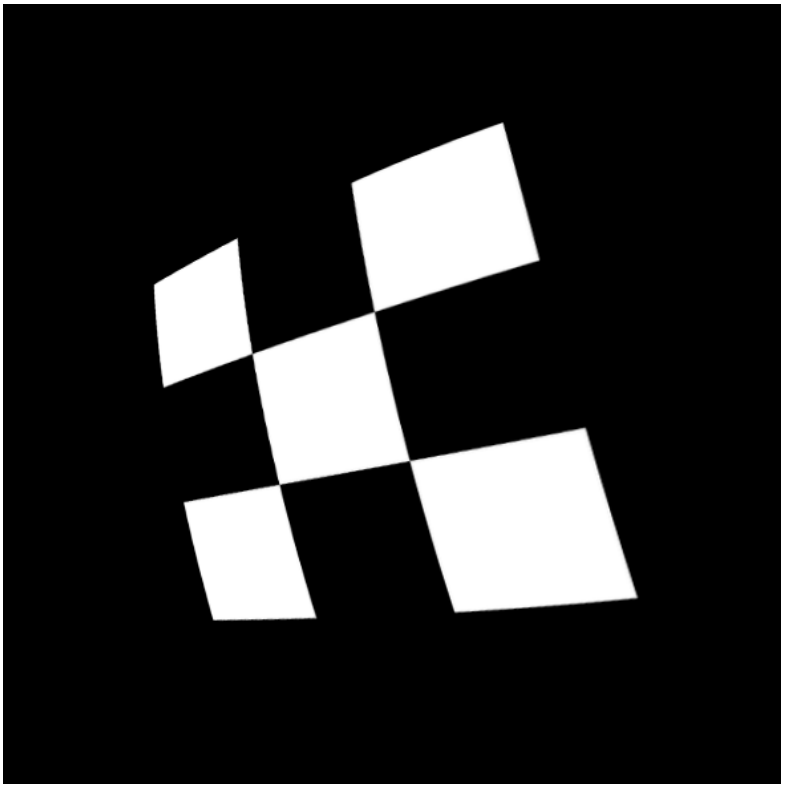
Rotacije kalibracijskog uzorka oko koordinatne osi kamere prikazane su OpenCV naredbama *cv2.getPerspectiveTransform*, za računanje matrica rotacije, te *cv2.warpPerspective*, za prikaz nove slike za dobivenu matricu rotacije. Svi parametri potrebni za stvaranje nove simulirane slike biti će nasumično generirani brojevi unutar određenih granica. Valja napomenuti da je zbog simetričnog kalibracijskog uzorka nemoguće prepoznavati rotacije oko osi z veće od 45º i manje od -45º.

Nakon nasumično generirane translacije i rotacije dobili smo novu sliku za koju prvo moramo proizvesti efekt ribljeg oka, odnosno simulirati radijalnu distorziju te nakon toga primijeniti algoritam koji će prepoznati i grupirati točke. Simulirana pozicija prikazana je na slici 15.



Slika 14. Kalibracijski uzorak nakon pomicanja kamere

Na novoj rotiranom i translatiranom kalibracijskom uzorku na kojem zapravo simuliramo pomicanje kamere u odnosu na kalibracijsku plohu moramo proizvesti efekt ribljeg oka što je postignuto OpenCV funkcijom *cv2.undistort* koja je namijenjena da ispravlja distoziranu sliku uz pomoć izračunatih parametara. Koeficijenti distorzije nisu odabirani nasumično već su za svaki slučaj korišteni isti, te je korištena ista koordinata centra distorzije koja nije u centru ekrana i ista fokalna točka, zbog činjenice da nijedna kamera nema leću bez greške. Slika 16 sada prikazuje simulaciju slike kamere koja je u nedefiniranoj poziciji u odnosu na kalibracijsku plohu i ima određenu radijalnu distorziju prilikom slikanja.

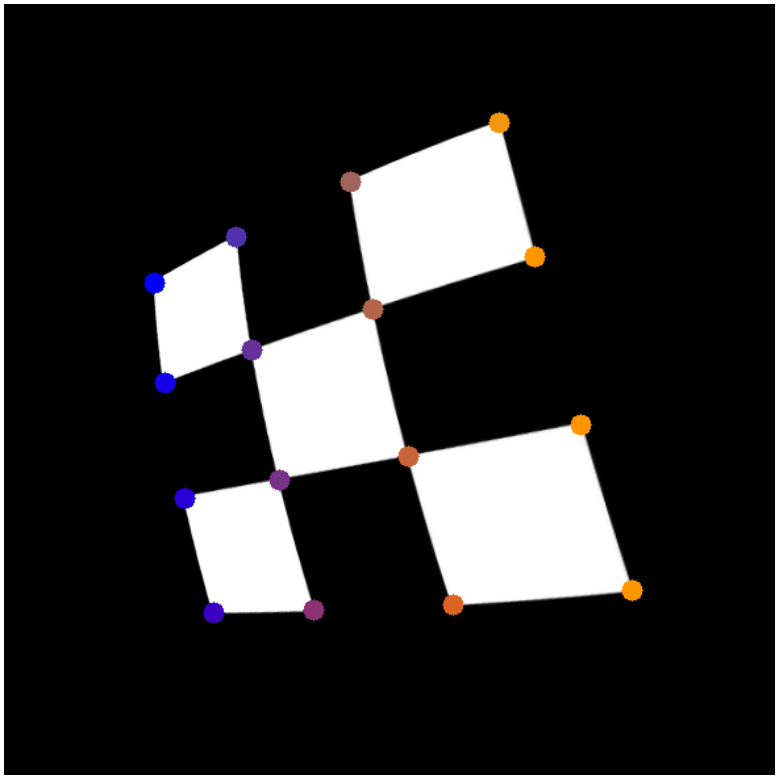


Slika 15. Simulacija radijalne distorzije kalibracijskog uzorka

Na simuliranoj slici kamere sada je primijenjen algoritam za traženje i grupiranje točaka koji prvo koristi funkciju *cornerHarris*. Funkcija za svaki piksel računa matricu kovarijantnih gradijenata dimenzija 2x2 za određenu, u ovom slučaju 2x2, okolinu piksela, te se nakon toga rubovi mogu naći kao lokalni maksimumi dobivene vrijednosti. Zbog toga što su rezultati pikseli nakupljeni oko rubova, grupiramo ih u liste, te usrednjavamo iznose njihovih x i y koordinata kako bismo dobili točnije koordinate rubova.

Kada imamo definirane točke rubova, rješenjem najmanjih kvadrata za kombinacije grupacija od 4 točke za ovakav kalibracijski uzorak filtriraju se oni pravci koji imaju najbolju regresiju, izuzimaju dijagonalni pravci, te je rezultat mreža pravaca od kojih su 4 međusobno približno paralelna i približno okomita s ostala 4. Na kraju je odabrana grupu (a) pravaca, te na njima prepoznajemo redoslijed točaka po iznosu njihove x ili y koordinate.

Kao rezultat algoritma imamo grupirane točke tako da ih program može povezati s onima iz kalibracijskog uzorka. Sada su dobivene točke uzorka i ako se znaju karakteristične dimenzije uzorka može se izračunati matrica transformacije u koordinatni sustav kalibracijske plohe iz koordinatnog sustava kamere.



Slika 16. Pronađene točke rubova u određenom redoslijedu

Sljedeće što treba napraviti jest odrediti rotaciju i translaciju kamere iz dobivenih točaka. Za određivanje tih vanjskih parametara koristimo takozvani model aperture kamere u kojemu je kadar opisan kao projekcija trodimenzijskih točaka na sliku koristeći sljedeću transformaciju.

$$s\begin{bmatrix}u\\v\\1\end{bmatrix}=\begin{bmatrix}f_x & 0 & c_x\\0 & f_y & c_y\\1 & 1 & 0\end{bmatrix}\begin{bmatrix}r_{11} & r_{12} & r_{13} & t_1\\r_{21} & r_{22} & r_{23} & t_2\\r_{31} & r_{32} & r_{33} & t_3\end{bmatrix}\begin{bmatrix}X\\Y\\Z\\1\end{bmatrix}$$

Gdje su (X,Y,Z) trodimenzijske koordinate točaka u stvarnom svijetu, (u,v) koordinate projiciranih točaka izražene u pikselima i matrice A i R unutarnji i vanjski parametri kamere.

Korištena je funkcija *cv2.calibrateCamera* iz OpenCV knjižnice. Funkcija vraća unutarnje i vanjske parametre kamere dok su ulazne varijable dvije skupine točaka. Algoritam je baziran na Zhangovoj metodi i kao zadnji korak koristi Lavenberg-Marquardtovu optimizaciju za minimizaciju grešaka ponovljenih projekcija koji je koristan za ispravljanje radijalne distorzije, odnosno računanje unutarnjih parametara kamere.

Prva skupina točaka koju zahtjeva funkcija *calibrateCamera* su trodimenzijske točke kalibracijskog uzorka koji kako je pretpostavljeno leži u x-y ravnini kalibracijske plohe pa su mu zadnji elementi odnosno iznosi na z osi jednaki nuli. Koordinate karakterističnih točaka kalibracijskog uzorka izražene su u metrima.

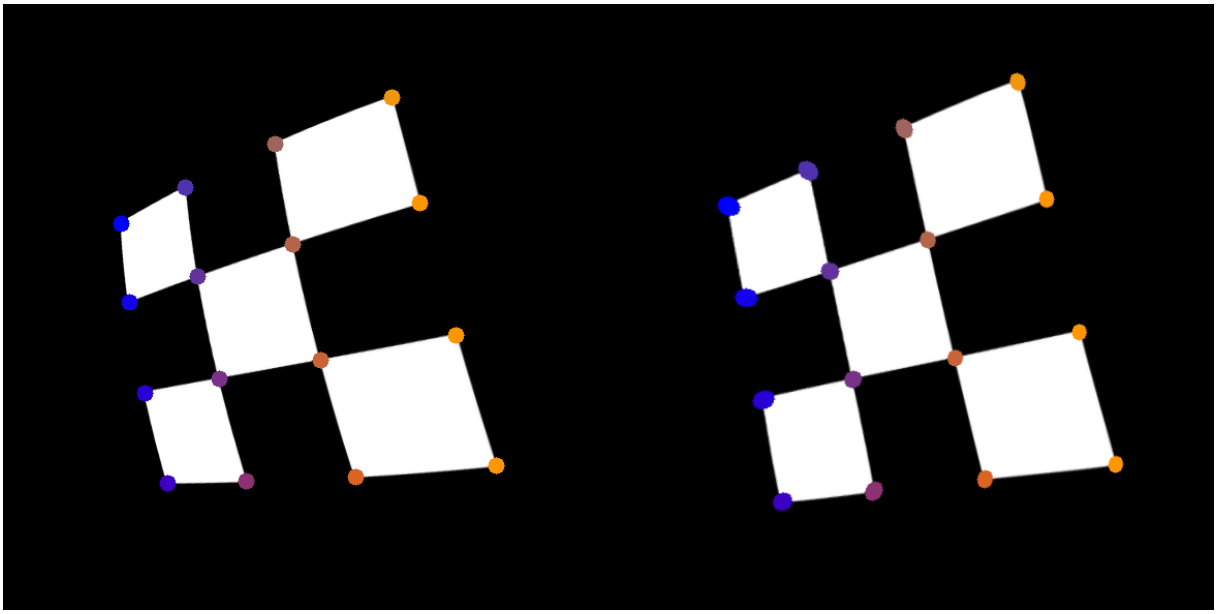
$$T_o=\begin{bmatrix}[0\ \ 0\ \ 0] & [0.15\ \ 0\ \ 0] & [0.3\ \ 0\ \ 0] & [0.45\ \ 0\ \ 0]\\ [0\ \ 0.15\ \ 0] & [0.15\ \ 0.15\ \ 0] & [0.3\ \ 0.15\ \ 0] & [0.45\ \ 0.15\ \ 0]\\ [0\ \ 0.3\ \ 0] & [0.15\ \ 0.3\ \ 0] & [0.3\ \ 0.3\ \ 0] & [0.45\ \ 0.3\ \ 0]\\ [0\ \ 0.45\ \ 0] & [0.15\ \ 0.45\ \ 0] & [0.3\ \ 0.45\ \ 0] & [0.45\ \ 0.45\ \ 0]\end{bmatrix}$$

Druga skupina točaka su dvodimenzijske projicirane točke izražene u pikselima. Za korištenje funkcije potrebno je navesti ih u istom redoslijedu kao što se navode točke kalibracijskog uzorka.

5.1. Simulacija računanja unutarnjih parametara kamere

Funkcija za računanje parametara kamere *calibrateCamera* na dva od pet izlaza daje matricu aperture kamere i koeficijente radijalne distorzije. Matrica aperture kamere daje informaciju o koordinatama fokalne točke i koordinate centra slike dok koeficijenti radijalne distorzije aproksimiraju radijalnu distorziju snimka. Kada znamo ta dva unutarnja parametra optimira se nova matrica apreture funkcijom *getOptimalNewCameraMatrix* koja prima oba dobivena parametra i vraća novu matricu s kojom se može rekonstruirati slika sa smanjenom radijalnom distorzijom pomoću funkcije *undistort*.

Slika 18. prikazuje rezultat kompenzacije unutarnjih parametara kamere.



Slika 17. Usporedba rezultata sa distordiranim stanjem

Sljedeći izrazi su izračunata kompenzirana matrica aperture i koeficijenti radijalne distorzije:

$$A = \begin{bmatrix} 290 & 0 & 355 \\ 0 & 331.3 & 430.9 \\ 0 & 0 & 1 \end{bmatrix}$$

$$k = [-2.82 \quad -0.0531 \quad -0.352 \quad -0.163 \quad 7115]$$

5.2. Simulacija računanja vanjskih parametara kamere

Nakon ponovljenog korištenja funkcije *calibrateCamera* može se dobiti vektor rotacije i vektor translacije na dva od pet izlaza funkcije.Na kraju za dani primjer na slikama imamo izračunate vektore rotacije i translacije kamere u odnosu na kalibracijsku površinu.

$$Rotacija = \begin{bmatrix} -25.23 \\ -2.73 \\ 26.35 \end{bmatrix}$$

$$Translacija = \begin{bmatrix} -0.32 \\ -0.052 \\ 1.43 \end{bmatrix}$$

Za koje matrica transformacije izgleda:

$$\mathrm{T} = \begin{bmatrix} 0.8951 & -0.1489 & -0.4203 & -0.32 \\ 0.4433 & 0.3979 & 0.8032 & -0.052 \\ 0.0476 & -0.9053 & 0.4221 & 1.43 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

6. EKSPERIMENTALNA PROVJERA ALGORITMA KALIBRACIJE

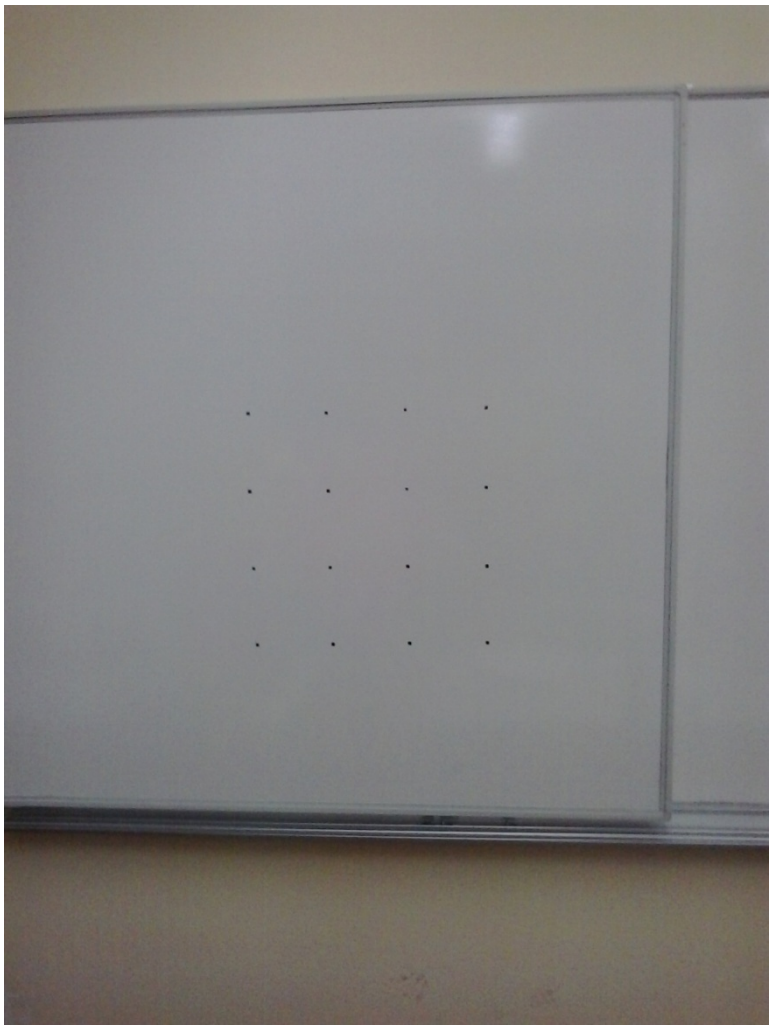
6.1. Oprema i proces ispitivanja

Eksperimentalni postav sastoji se od kamere, projektora i prezentacijske ploče. Korištena kamera je Phillips web-kamera modela SPC650NC, prezentacijski projektor i prezentacijska ploča. Prezentacijska ploča je jako reflektivna površina pa se osvjetljenje mora primijeniti tako da ne daje odsjaj na predmetu promatranja dok će svaki odsjaj koji se pojavi pokraj predmeta promatranja biti očišćen postavljanjem pragova na 256-bitnoj slici pomoću funkcije *cv2.threshold*. Tokom ispitivanja koristi se preklopna ploča tako da se na njezinoj prednjoj strani nalazi nacrtan kalibracijski uzorak, dok je površina kada je ploča raskloplјena prazna kako bi se na nju mogla projicirati slika. Na početku ispitivanja kamera uzima prvi uzorak kalibracijskog uzorka koji je na prednjoj strani preklopne ploče tijekom čega je svijetlost projektora blokirana. Nakon slikanja prvog uzorka ploča se rasklapa i pušta se svjetlost projektora na prazni dio ploče iza preklopnog dijela, te kamera ponovo slika projiciranu sliku.

Slika 18. Phillips SPC650NC

6.2. Proces ispravljanja slike

Kao što je napisano u prethodnom poglavlju prvo obavljamo slikanje kalibracijskog uzorka koji je pričvršćen na preklopni dio prezentacijske ploče. Kalibracijski uzorak se sastoji od 16 točaka koje predstavljaju rubove šahovnice koju projicira projektor i s pravilnim rasporedom definiraju kalibracijsku ravninu. Na sljedećoj slici je prikazan kalibracijski uzorak na kalibracijskoj ravnini.



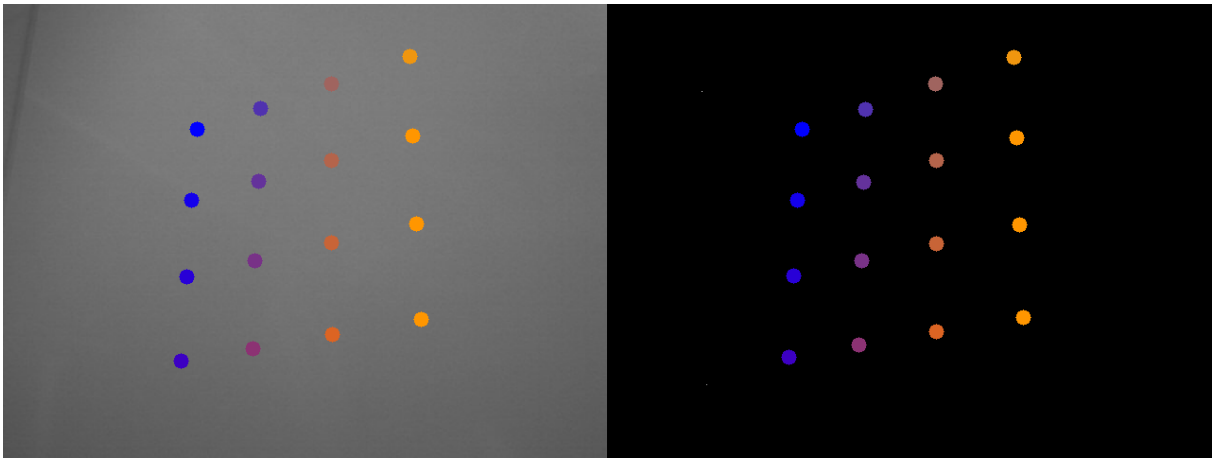
Slika 19. Kalibracijski uzorak iz perspektive promatrača

U sljedećem postupku kamera koja se nalazi na stolu ispred prezentacijske ploče slika kalibracijski uzorak. Rezultat slikanja biti će slika koja zbog grešaka kamere ima radijalnu i perspektivnu distorziju. Iz pozicije točaka na ovoj slici može se saznati pozicija i orijentacija kamere, te ispraviti radijalnu distorziju uzrokovanu slikanjem ako znamo stvarne koordinate

točaka na prezentacijskoj ploči. U program za ispravljanje slike unesene su stvarne koordinate točaka sa ishodištem u najvišoj lijevoj točki, a dimenzije su izmjerene ravnalom.

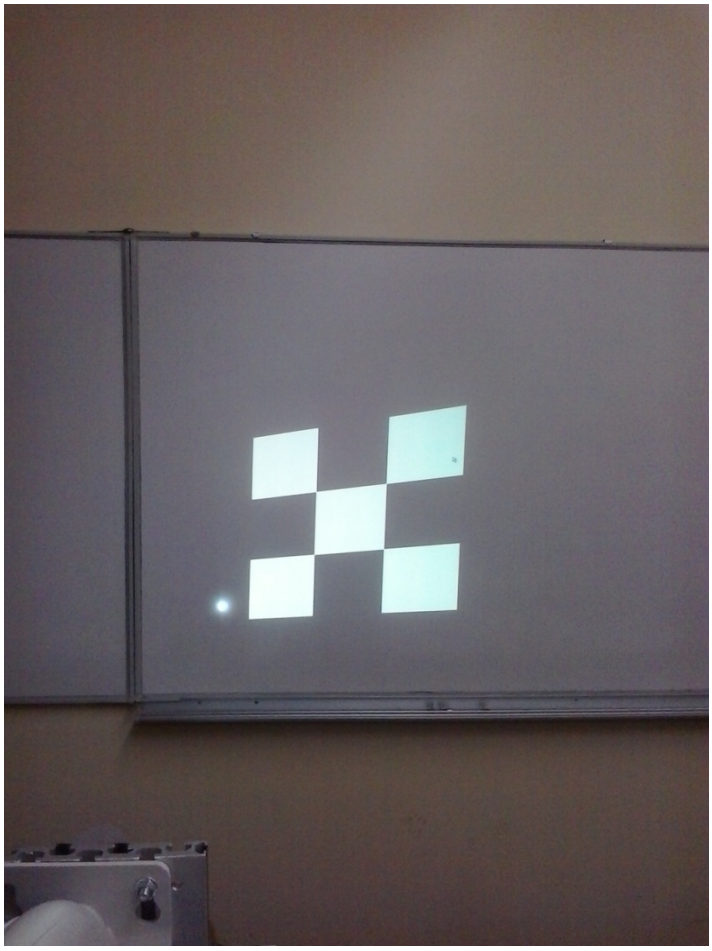
Slika 20 Kamerom slikan kalibracijski uzorak

Sljedeći korak je pronalaženje točaka i njihovo sortiranje u željeni redoslijed pomoću algoritma objašnjenog u prethodnim poglavljima. Rezultat pronalaženja i sortiranja točaka vidljiv je na lijevoj strani slike 22. Pomoću pronađenih točaka i poznatih koordinata u kalibracijskoj ravnini mogu se dobiti koeficijenti radijalne distorzije, te pozicija i orijentacija kamere. Navedene parametre kamere doznajemo pomoću OpenCV funkcije *cv2.calibrateCamera*. Izračunate koeficijente radijalne distorzije primjenjujemo na sliku 21, no prije toga mora se na istu sliku primijeniti prag pomoću funkcije cv2.threshold koja će sve piksele unutar određenog praga bojati u crnu boju, a ostale u bijelu. Funkcija pragovanja koristi zato što će slika na kojoj je ispravljena radijalna distorzija imati praznog mjesta na rubovima koja će biti bojana crno pa se može desiti da algoritam za traženje točaka na takvom mjestu prepozna rub i takva pojava uzrokuje greške u daljnjem radu programa. Radijalna distorzija ispravlja se funkcijom *cv2.undistort* a rezultat funkcije može se vidjeti na desnoj slici slike 22. Iz točaka pronađenih na ispravljenoj slici računa se matrica rotacije koja će se kasnije primjenjivati.



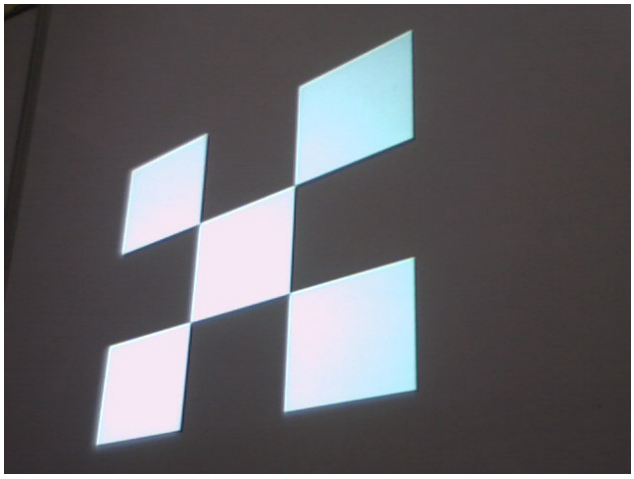
Slika 21 Ispravak radijalne distorzije

Nakon slikanja i obrade podataka kalibracijskog uzorka okrećemo prezentacijsku podlogu i puštamo svjetlo projektora. Slika 23 prikazuje prezentacijsku površinu, odnosno kalibracijsku ravninu iz perspektive promatrača.



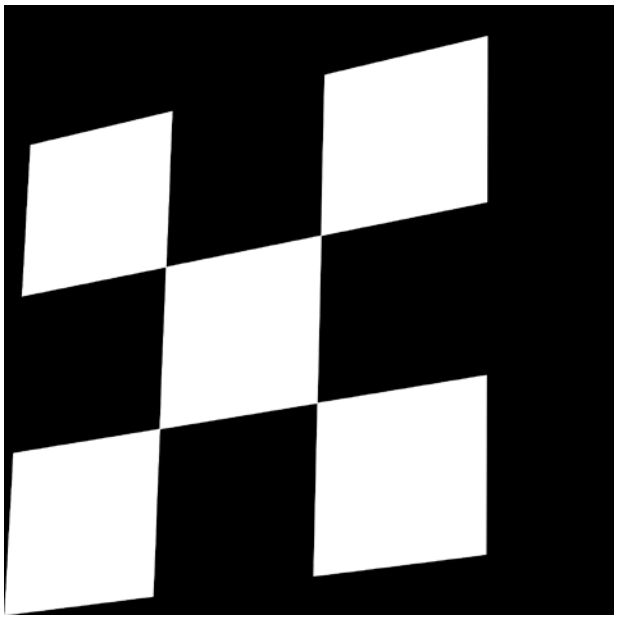
Slika 22 Projekcija slikana iz perspektive promatrača

Slika 23 prikazuje kamerom slikani kalibracijski uzorak. Ova slika ima iste one nedostatke uzrokovane greškom kamere kao i prva slika kalibracijskog uzorka pa je potrebno ispraviti je koristeći koeficijente dobivene iz prethodnog postupka



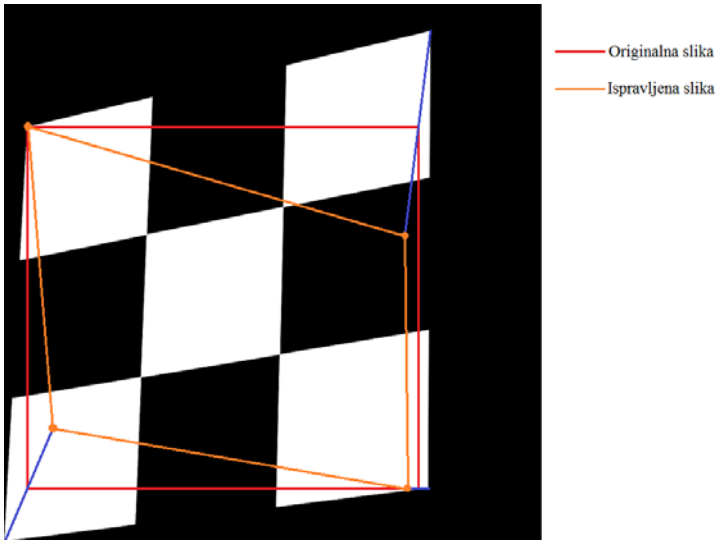
Slika 23 Uzorak slikan kamerom

Nakon ispravljanja slike koristimo prethodno izračunatu matricu rotacije kako bi dobili projicirani uzorak u perspektivi kalibracijske površine. Na takvoj slici koja nema radijalnu i perspektivnu distorziju sada pokrećemo algoritam koji traži rubove kao točke. Kada su nam poznate koordinate točaka rubova možemo na izvornu sliku primijeniti matricu transformacije dobivenu pomoću dobivenih točaka kako bismo dobili sliku u originalnoj rezoluciji pomoću koje se proizvodi rektificirana slika.



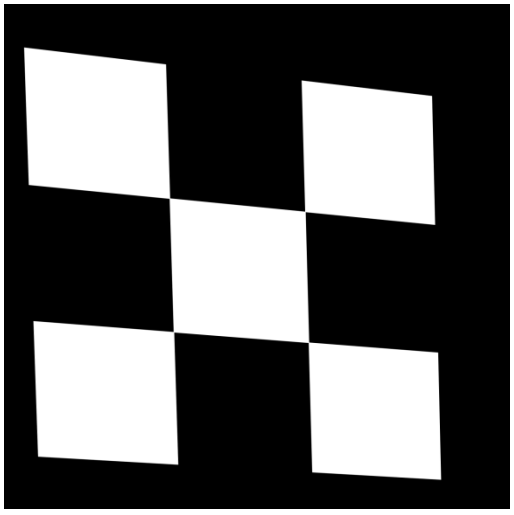
Slika 24 Projekcija uzorka u perspektivi kalibracijske podloge

Kada imamo sliku u perspektivi prezentacijske ravnine može se geometrijski proizvesti ispravljena slika na sljedeći način. Unutar površine projiciranog uzorka traži se najveća rezolucija sa jednakim omjerima visine i širine takva koja ne prelazi okvire točaka, te se nakon toga traže koordinate točaka za konstruiranje rotacijske matrice za transformaciju početne slike u ispravljenu. Koordinate potrebnih točaka doznajemo pomoću točkaste simetrije gdje su rubovi pronađene originalne slike unutar projiciranog uzorka točke simetrije.



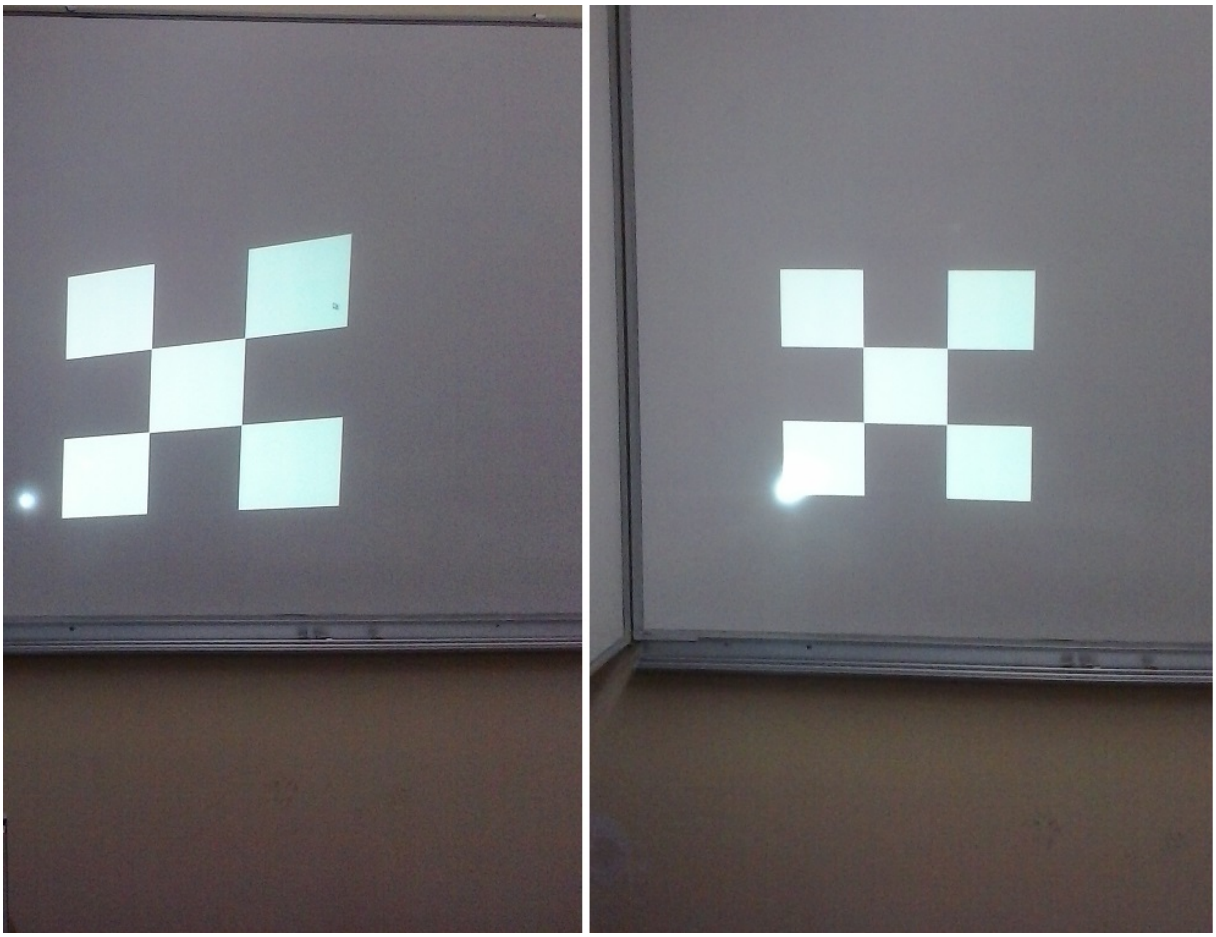
Slika 25 Postupak ispravljanja slike

Krajnji rezultat je slika koja će nakon projekcije na kalibracijskoj površini dati ispravljenu sliku i prikazana je na slici 25.



Slika 26. Perspekcijski ispravljena slika

Rezultat projekcije slike 26 prikazan je na desnoj strani slike 27 slikanoj iz perspektive promatrača ispred kalibracijske površine, odnosno prezentacijske ploče.



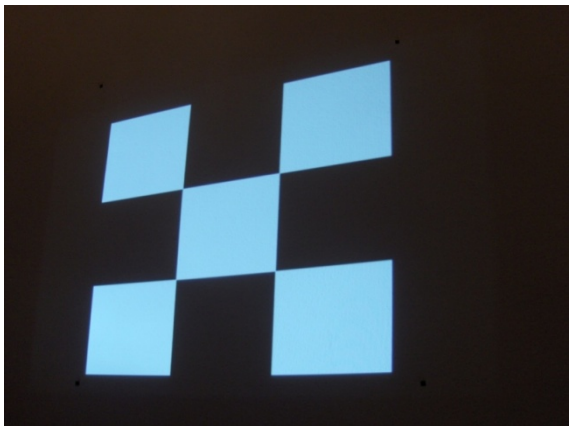
Slika 27 Rezultat programa za ispravljanje slike

6.3. Računanje vanjskih parametara kamere i projektor

Za računanje vanjskih parametara kamere i projektora, odnosno iznose za definiranje njihovih koordinatnih sustava koristimo dva procesa. Kroz prvi proces saznajemo matricu transformacije za prebacivanje koordinatnog sustava iz kamere u koordinatni sustav podloge. Za tu operaciju koristimo samo četiri točke zanemarujući radijalnu distorziju.

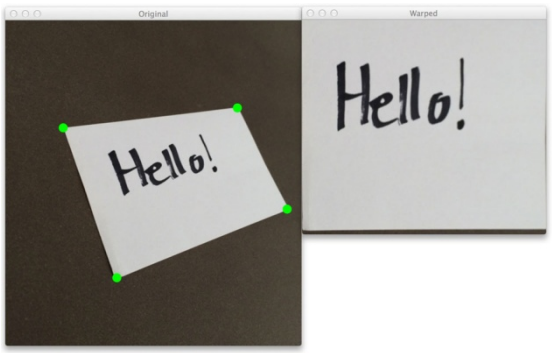
Slika 28 Točke koje definiraju ravninu projekcije

U drugom djelu programa projiciramo uzorak na podlogu te se nakon snimanja koristi prethodno izračunata matrica transformacije kako bi ispravili perspektivu i dobili sliku iz perspektive podloge. kako bi dalje mogli iskoristiti algoritam za obradu slike i prepoznavanje točaka za računanje koordinatnog sustava s obzirom na zid.



Slika 29 Projicirana slika

Rotacijsku matricu računamo pomoću funkcije *getperspectivetransform* koja na ulazu zahtjeva 4 točke izražene u koordinatama piksela.



Slika 30 Primjer ispravljanja slike

Funkcija računa 3x3 matricu perspektive takvu da vrijedi:

$$\begin{bmatrix} t_i x_i' \\ t_i y_i' \\ t_i \end{bmatrix} = M \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Gdje su (x_i', y_i') koordinate rubova željene slike, a (x_i, y_i) koordinate polazne slike.

Za prikaz transformirane slike potrebni su koeficjenti perspektivne transformacije:

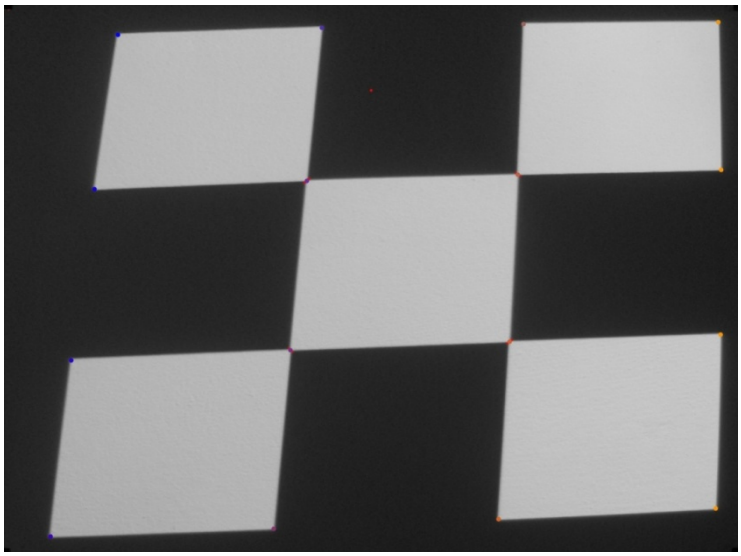
$$x_i' = \frac{c_{00}x_i + c_{01}y_i + c_{02}}{c_{20}x_i + c_{21}y_i + c_{22}}$$

$$y_i' = \frac{c_{10}x_i + c_{11}y_i + c_{12}}{c_{20}x_i + c_{21}y_i + c_{22}}$$

Koeficjenti su izračunati rješenjem linearnog sustava jednadžbi:

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & x_1x_1' & y_1x_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & x_2x_2' & y_2x_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & x_3x_3' & y_3x_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & x_4x_4' & y_4x_4' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & x_1y_1' & y_1y_1' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & x_2y_2' & y_2y_2' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & x_3y_3' & y_3y_3' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & x_4y_4' & y_4y_4' \end{bmatrix} \begin{bmatrix} c_{00} \\ c_{01} \\ c_{02} \\ c_{10} \\ c_{11} \\ c_{12} \\ c_{20} \\ c_{21} \end{bmatrix} = \begin{bmatrix} x_1' \\ x_2' \\ x_3' \\ x_4' \\ y_1' \\ y_2' \\ y_3' \\ y_4' \end{bmatrix}$$

Na sljedećoj slici mogu se vidjeti odabrane točke rubova šahovnice bojane u nijanse od plave do narančaste dok su izostavljene točke obojene crvenom bojom. Može se primijetiti da algoritam zna odabrati rubove šahovnice iz skupa točaka koje je pronašla funkcija *corrnerHarris*. Analizirana je slika nakon perspektivne transformacije pomoću matrice do koje smo došli u prethodnim radnjama. Za tu sliku se može zaključiti da se nalazi u perspektivi podloge za projekciju, te da sadrži perspektivnu iskrivljenost uzrokovanu samo orijentacijom projektora.



Slika 31 Slika u perspektivi projekcijske podloge

Nakon analize prve slike saznajemo izraze za koordinatne osi kamere u odnosu na koordinatni sustav podloge. Rotacijski odnosi su izraženi u radijanima:

$$R_k = \begin{bmatrix} -0.2599 \\ 0.2364 \\ 0.0447 \end{bmatrix} rad$$

$$T_k = \begin{bmatrix} -38.85 \\ -28.20 \\ 110.24 \end{bmatrix} cm$$

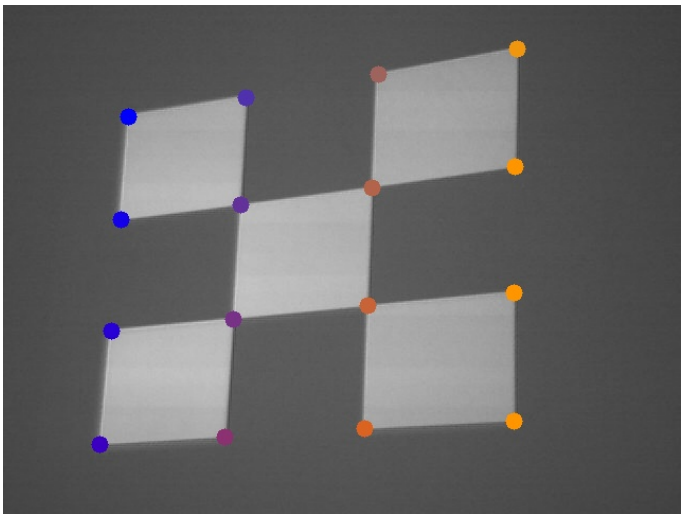
I na kraju računamo koordinatne osi projektora u odnosu na koordinatni sustav zida:

$$R_k = \begin{bmatrix} -0.6006 \\ 0.1198 \\ 0.0016 \end{bmatrix} rad$$

$$T_k = \begin{bmatrix} -123.7 \\ -76.37 \\ 122.3 \end{bmatrix} cm$$

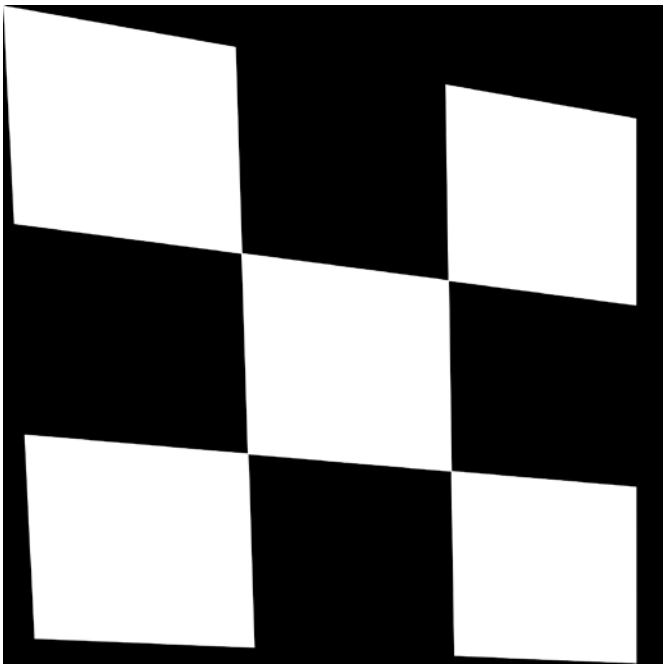
6.4. Računanje matrice rotacije za transformaciju iz koordinatnog sustava kamere u koordinatni sustav projektora bez korištenja fizičkih značajki

Pomoću poznatog kalibracijskog uzorka moguće je izračunati matricu rotacije koja će opisati odnos rotacije između koordinatnog sustava kamere i projektora koristeći samo jednu projekciju bez fizičkih značajki iz okoline. Fizički ne možemo zaključiti ništa o ravnini projekcije. Proces računanja matrice rotacije se sastoji samo iz snimanja projicirane slike te njezine obrade. Da bi saznali matricu rotacije prvo moramo poznavati relevantne točke za ispravak perspektive do kojih ćemo doći pomoću metode opisane u poglavlju 6.2 na slici 26. Kao što je poznato, algoritam prepoznavanja točaka na postojećoj slici traži najmanji kvadrat unutar iskrivljenog uzorka, te osnom simetrijom pronalazi koordinate točaka koje se koriste za računanje matrice rotacije pomoću funkcije *getperspectivetransform* što je objašnjeno u poglavlju 8.3.



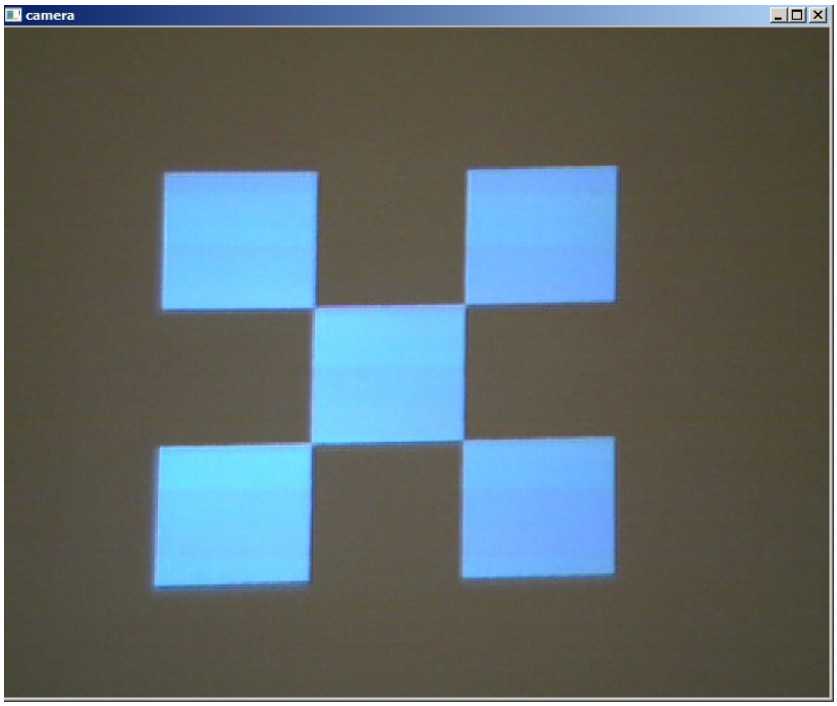
Slika 32 Iskrivljena slika u perspektivi kamere sa prepoznatim rubovima

Na slici može se vidjeti snimljeno stanje distorzije slike nastalo projiciranjem slike i pronađene točke tubova.



Slika 33 Ispravljena slika za projiciranje

Slika prikazuje rezultat ispravljanja slike u koordinatnom sustavu kamere. Ovu sliku ćemo sada projicirati na projekcijsku podlogu, snimiti kamerom koja je ostala na istoj poziciji i orijentaciji kao i pri prvom snimanju, te komentirati rezultat.



Slika 34 Snimljena projekcija slike 33

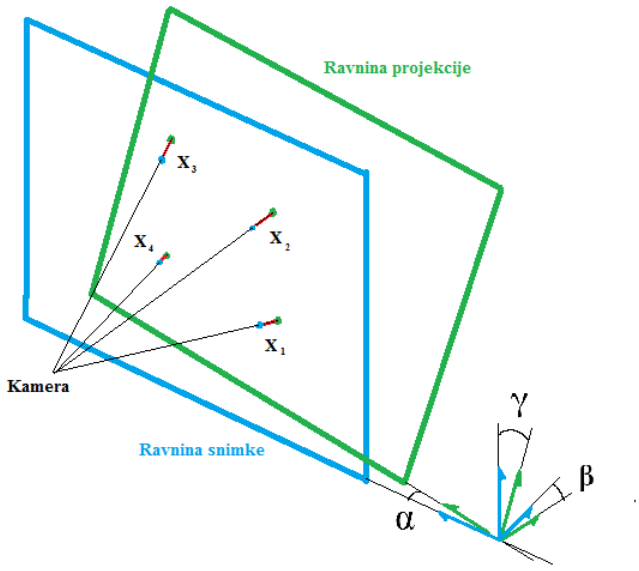
Slika prikazuje snimku projiciranog ispravka, odnosno rezultata ispravljanja. Može se zaključiti da je postupak ispravljanja dobar. Sa rezultata se mogu primijetiti mala odstupanja što se može pripisati pogrešci zbog rezolucije.

Do matrice rotacije dolazimo pomoću funkcije *calibratecamera* kojoj ćemo na ulazu zadati točke ispravnih rubova, pomoću funkcije ćemo računati rotaciju koordinatnog sustava projektora u koordinatni sustav kamere. Ovakvim pristupom rotacija oko osi y će biti negativna.

$$M = \begin{bmatrix} 17.6 \\ -34.4 \\ 4.18 \end{bmatrix}$$

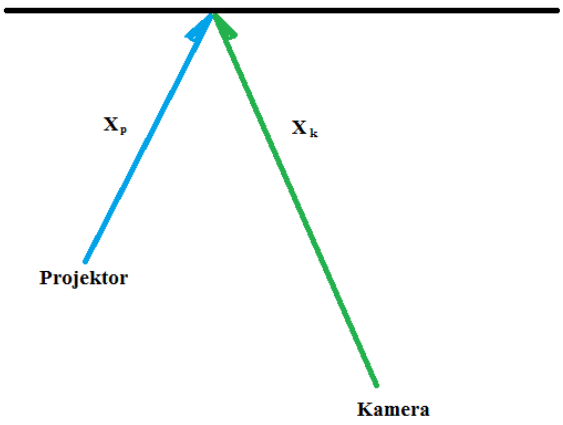
6.5. Ispravak slike pomoću projekcije

Kod ispravljanja slike koristeći samo projekciju podrazumijeva se ispravljanje slike bez korištenja fizičkih značajki iz stvarnog svijeta. Korištenjem dvije slike koje se razlikuju po svojoj veličini, gdje je jedna umanjena za 10%, te njihovom projekcijom pokušava se analizom udaljenosti između karakterističnih točaka saznati matrica transformacije za računanje koordinatnog sustava podloge projekcije iz koordinatnog sustava kamere. Ako izračunamo udaljenosti između karakterističnih točaka na slikama koje nisu projicirane ili ako su projicirane i snimljene iz kuta projekcije, one će biti istog iznosa. Zbog perspektive udaljenosti između istih karakterističnih točaka će se razlikovati i upravo na analizi razlike možemo procijeniti orijentaciju zida naspram kamere.



Slika 35 Procjena rotacijskog odnosa ravnina

$$X_1 = C\ k_1$$
$$X_2 = C\ k_2$$
$$X_3 = C\ k_3$$
$$X_4 = C\ k_4$$

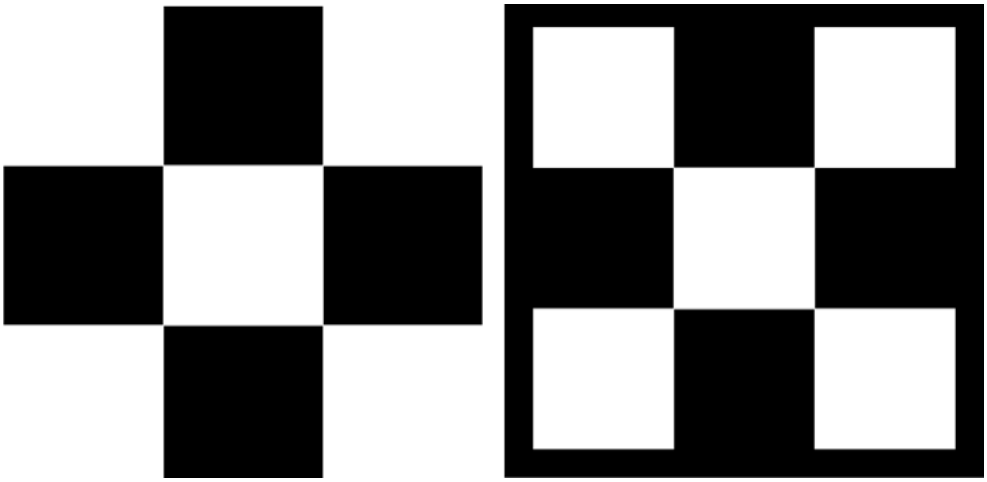


Slika 36 Procjena udaljenosti

$$k_k x_k + k_p x_p = C$$

Spomenuti koeficijent C procjenjujemo iterativnim postupkom dok je koeficijent k_k koeficijent kamere koji daje informaciju koliko piksela označava određenu udaljenost u fizičkom svijetu, naravno ne funkcionira bez dijela jednadžbe koja opisuje udaljenost projektora od projekcijske površine što se određuje koeficijentom k_p. Oba koeficijenta procjenjujemo iterativim postupkom odnosno iskustvenom metodom pokušaja i promašaja.

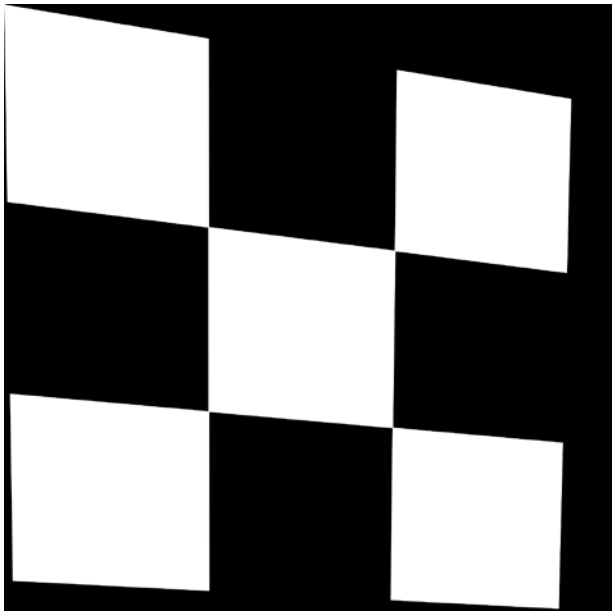
Pomoću navedenih jednadžbi možemo procijeniti matricu rotacije za opisivanje ravnine projekcije. Kroz postupak ćemo saznati parametre k tako da mjerimo razlike između istih točaka projiciranih slika. Konstanta C je konstanta koja opisuje udaljenost fokalnih točaka kamere i projektora u odnosu na projekcijsku površinu. Kako ne znam niti jedne od navedenih udaljenosti, konstantu *C* moramo saznati iterativnim postupkom.



Slika 37 Odnosi dvije različite projekcije

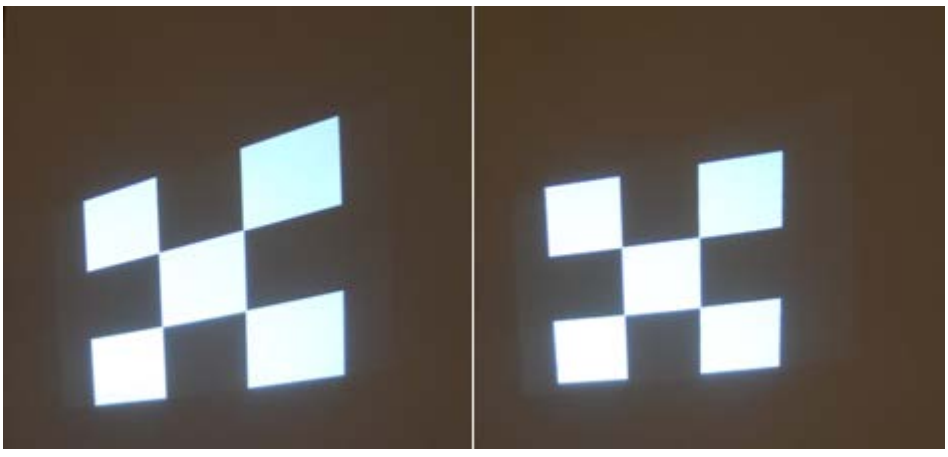
Nakon korištenja poznatog postupka traženja točaka i njihovog razvrstavanja, te analize dolazim do koordinata karakterističnih točaka za svaku projiciranu sliku. Kada znamo koordinate možemo izračunati udaljenosti između točaka na istim pozicijama sa obije slike nakon čega odnosom tih udaljenosti i originalnih udaljenosti sa početnih slika koje projiciramo kroz prve dvije projekcije možemo dobiti omjere koji daju informaciju o orijentaciji kamere prema koordinatnom sustavu zida. Bitno je napomenuti da snimljene slike projekcija moramo obraditi tako da budu iste rezolucije i omjera visine i širine kao i originalna projicirana slika kako bi sve dimenzije koje uspoređujemo bile odgovarajuće za usporedbu.

Poznate omjere sada možemo iskoristiti za geometrijsko ispravljanje slike kod kojega točke perspektive udaljujemo od rubova najmanje moguće pravokutne slike unutar promatranog uzorka u smjeru određenom točkom ruba najmanje moguće slike i ruba uzorka za iznos umanjen izračunatim omjerom. Ispravljena slika ovakvom metodom prikazana je na slici 33.



Slika 38 Ispravljena slika metodom opisanom u poglavlju 6.3

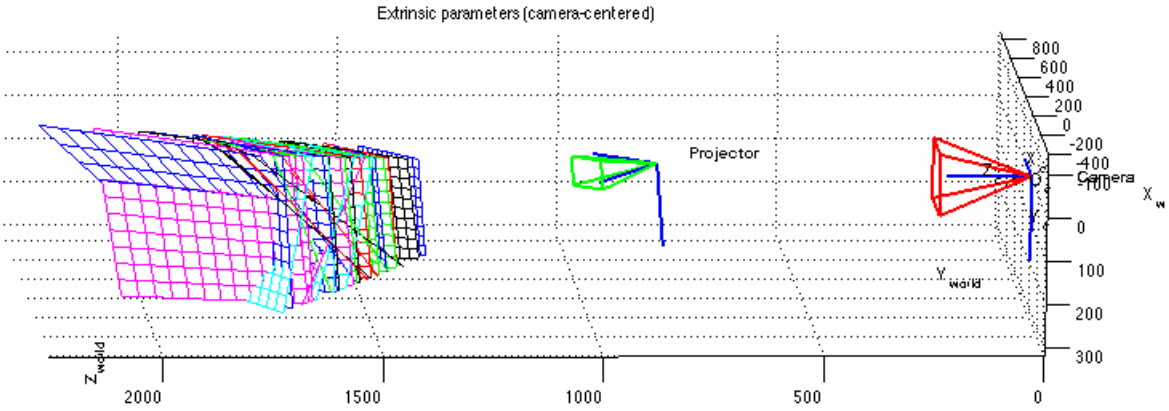
Rezultati ispravka vidljivi su na slici 34. Može se primijetiti da slika nije potpuno ispravljena, ali rezultati iz više pozicija pokazuju da će slika težiti ispravku što dokazuje da je postupak ispravljanja dobar. Zbog činjenice da se odnosi ispravljanja perspektive stvaraju iz udaljenosti kamere i pojedine karakteristične točke njihovi iznosi ne mogu biti negativni što bi značilo da će ispravak uvijek zadržati određenu grešku koja se smanjuje s brojem iteracija, ali se smanjuje i slika.



Slika 39 Pregled rezultata

7. Usporedba sa sličnim radovima

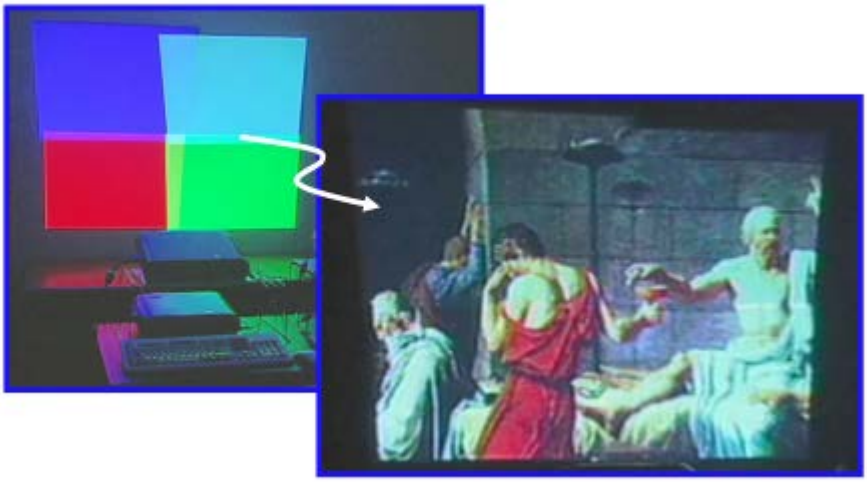
Gabriel Falacao, Natalia Hutos i Joan Massich u svom znanstvenom radu nazvanom "Ravninski orijentirana kalibracija sustva kamere i projektora" u potpunosti su razvili metodu za računanje svih odnosa triju elemenata koji sudjeluju u kalibraciji, a to su kamera, projektor i kalibracijska podloga. Razvijeno je programsko sučelje za kalibraciju kamere i projektora pomoću poznatih fizičkih značajki kalibracijske ravnine. Korisničko sučelje također je konstruirano kako bi bilo lako za upotrebu, te dodatno daje lako razumljive rezultate s grafičkim prikazom.



Slika 40 3D prikaz kalibracije

Michael Brown, Aditi Majumder i Ruigang Yang u svome znanstvenom radu koristili su više kamera na čijim snimkama algoritam pronalazi veze između piksela koji su reflektirani iz iste površine u prostoru kako bi se mogla proračunati jednadžba koja opisuje ravninu projekcije. U njihovom istraživanju korišteno je do pet kamera čije informacije je trebalo iskoristiti na način da se preklapajuće slike projektora sklope u jednu smislenu cjelinu. Ovakav zadatak iziskuje puno kompliciranije algoritme koji rade na principu kod kojega se prepoznaju linije preklapanja projekcija te se saznaje veza istih na drugim snimkama iz čega se matematički može zaključiti pozicija projektora, i matematički opis ravnine projekcije. Kod

ovakve metode za rješavanje zadanog problema morali su se definirati pozicije i orijentacije kamera u odnosu na glavnu kameru. Ovakav sustav se razlikuje od ovog rada po tome što je u teoriji rješiv i definiran što kod kalibracije sustava projektor-kamere bez fizičkih značajki iz okoline to nije slučaj, ali je mnogo opsežniji i koristi opsežnije i robusnije algoritme.



Slika 41 Rezultat spajanja slika iz više projektora na naepoznatoj ravnini

8. ZAKLJUČAK

Kroz ovaj rad su objašnjene i demonstrirane neke metode za kalibraciju projektora i kamere. Kroz prethodna poglavlja objašnjeni su teoretski koraci metode kalibracije vanjskih i unutarnjih parametara kamere i vanjskih parametara projektora, simulirani su neki osnovni problemi kod manipulacije perspektive u programskom sučelju OpenCV i na kraju je provedeno ispitivanje ispravljanja slike. Unutarnji parametri kamere uzeti su u obzir samo kod simulacije za demonstraciju dok se kroz eksperimentalni dio zanemario utjecaj grešaka kao što su radijalna distorzija te pomak fokusa kamere zbog toga što su kroz sva snimanja tražene točke bile relativno blizu središnjih piksela kamere unutar kojih se stvarni fokus mora nalaziti. Za zadani cilj ispravljanja slike uz korištenje fizičkih značajki iz prostora, odnosno plošno bazirana metoda pokazala je točnost unutar tolerancije u iznosu od 0.5 stupnjeva što je izmjereno ravnalom te nakon bilježenja dobivenih idealnih točaka izmjeren je kut zaokreta projektora. Ovakva metoda također se pokazala laganom za korištenje. Jedna od mana metode je ta što je bazirana, odnosno ovisna o kalibracijskoj površini i uzorku, te zbog toga svaka promijenjena pozicija kamere zahtjeva novu kalibraciju kamere kako bi se mogla izvršiti kalibracija procesora.

Kod eksperimentalne metode bazirane na kameri izmjereno je odstupanje u odnosu na plošno baziranu metodu u iznosu od 0.5 stupnjeva za računanje matrica rotacije što se može interpretirati kao tolerancija u iznosu jednog stupnja. Sveukupni ispravak slike pomoću metode bazirane na kameri zahtjeva bolja mjerenja kako bi se mogli unijeti točniji koeficijenti za procjenu udaljenosti kamere i projektora od projekcijske površine. Istraživanja kroz ovaj rad dovode do zaključka kako usprkos nedovoljnoj točnosti koeficijenata slika teži ispravnoj projekciji.

9. PROGRAMSKI KOD

9.1. Simulacijski kod

```
import cv2
import numpy as np
import random
import math
from itertools import combinations
import warnings
warnings.filterwarnings('error')

np.set_printoptions(threshold=np.nan)

res = cv2.imread('C:\\Users\\Korisnik\\Desktop\\fax\\ProjektMR\\sample4.png',0)
xs=random.randrange(7,11,1)*0.1
print xs
res = cv2.resize(res,None,fx=xs, fy=xs, interpolation = cv2.INTER_CUBIC)

rows,cols = res.shape
alfa=random.randrange(0,40,1)
rangle = np.deg2rad(alfa)
nw = (abs(np.sin(rangle)*rows) + abs(np.cos(rangle)*cols))
nh = (abs(np.cos(rangle)*cols) + abs(np.sin(rangle)*cols))
M = cv2.getRotationMatrix2D((nw*0.5,nh*0.5),alfa,1)
rot_move = np.dot(M, np.array([(nw-cols)*0.5, (nh-rows)*0.5,0]))
M[0,2] += rot_move[0]
M[1,2] += rot_move[1]
res = cv2.warpAffine(res, M, (int(math.ceil(nw)), int(math.ceil(nh))), flags=cv2.INTER_LANCZOS4)
bres=res

m=5
Ax=int(math.ceil(nw)/5)
Ay=int(math.ceil(nh)/5)
Bx=int(math.ceil(nw)/5)
By=int(math.ceil(nh)/5)
Cx=int(math.ceil(nw)/5)
Cy=int(math.ceil(nh)/5)
Dx=int(math.ceil(nw)/5)
Dy=int(math.ceil(nh)/5)
z=150
k=100
A=[random.randrange(m,Ax,1),random.randrange(m,Ay,1)]
B=[cols-random.randrange(m,Bx,1),random.randrange(m,By,1)]
C=[cols-random.randrange(m,Cx,1),rows-random.randrange(m,Cy,1)]
D=[random.randrange(m,Dx,1),rows-random.randrange(m,Dy,1)]


pts1 = np.float32([[0,0],[cols,0],[cols,rows],[0,rows]])

pts2 = np.float32([A,B,C,D])

M = cv2.getPerspectiveTransform(pts1,pts2)


res = cv2.warpPerspective(res,M,(int(math.ceil(nw)), int(math.ceil(nh))))
cpt=random.randrange(0,10000,1)
cv2.imwrite('rez\\image%04i.png'%cpt,res)
cv2.imshow('dst',res)
```

```
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()

class UnionFind:

    def __init__(self):
        self.weights = {}
        self.parents = {}

    def __getitem__(self, obj):

        if obj not in self.parents:
            self.parents[obj] = obj
            self.weights[obj] = 1
            return obj

        path = [obj]
        root = self.parents[obj]
        while root != path[-1]:
            path.append(root)
            root = self.parents[root]

        for ancestor in path:
            self.parents[ancestor] = root
        return root

    def union(self, obj1, obj2):
         roots = [self[obj1], self[obj2]]
        heavier = max([(self.weights[r],r) for r in roots])[1]
        for r in roots:
            if r != heavier:
                self.weights[heavier] += self.weights[r]
                self.parents[r] = heavier

def groupTPL(TPL, distance=15):
    U = UnionFind()

    for (i, x) in enumerate(TPL):
        for j in range(i + 1, len(TPL)):
            y = TPL[j]
            if max(abs(x[0] - y[0]), abs(x[1] - y[1])) <= distance:
                U.union(x, y)

    disjSets = {}
    for x in TPL:
        s = disjSets.get(U[x], set())
        s.add(x)
        disjSets[U[x]] = s

    return [list(x) for x in disjSets.values()]

def naditocke(slika):
    gray = np.float32(slika)
    dst=cv2.cornerHarris(gray,2,3,0.04)
    ind=np.where(dst>0.01*dst.max())
    x=[]
    listatocaka=[]
    p,d=list(ind[0]),list(ind[1])

    for i in range(len(p)):
        x.append((p[i],d[i]))
    grupirani=groupTPL(x)
    for i in grupirani:
        x=0
```

```
        y=0
        for j in i:
            y+=j[0]
            x+=j[1]
        listatocaka.append((int(round(float(x)/float(len(i)))),(int(round(float(y)/float(len(i)))))))

    dst = cv2.dilate(dst,None)
    img=cv2.cvtColor(res,cv2.COLOR_GRAY2RGB)

    img[dst>0.01*dst.max()]=[0,0,255]

    bimg=img

    return listatocaka,bimg
listatocaka,bimg=naditocke(res)
def najblize(Azad):
    T=[]
    L=[]

    for i in combinations(Azad,4):
        x = [i[0][0],i[1][0],i[2][0],i[3][0]]
        y = [i[0][1],i[1][1],i[2][1],i[3][1]]
        try:
            fit = np.polyfit(x,y,1)
        except Warning:

            funL=np.poly1d([0,int(round((x[0]+x[1]+x[2]+x[3])/4.))])
            T.append(list(i))
            L.append(list(funL)[0])
            continue
        funL = np.poly1d(fit)
        uda=0
        for h in range(4):
            uda+=abs(list(funL)[0]*float(x[h])-1.*float(y[h])+list(funL)[1])\
           /(math.sqrt((list(funL)[0])**2+1.**2))
        if uda < 20:
            T.append(list(i))
            L.append(tuple(funL))
    N=[]
    for i in L:
        n=L.index(i)
        Dv=[]
        for k in combinations(T[n],2):
            Dv.append(k)
        H=[]
        for j in Dv:
            F=math.sqrt((j[0][0]-j[1][0])**2+(j[0][1]-j[1][1])**2)
            H.append(F)

        N.append(Dv[H.index(max(H))])
    Dij=[]
    T0=[]
    for n in N:
            F=math.sqrt((n[0][0]-n[1][0])**2+(n[0][1]-n[1][1])**2)
            Dij.append(F)
    T0.append(N[Dij.index(max(Dij))])
    p=[]
    T0=list(T0[0])
    for i in T:
        b=0
        for k in i:

            if k==T0[0] or k== T0[1]:
                b+=1
```

```
    if b==1:
        p.append(i)

T1=[]
for i in T0:
    for n in p:
        Max=[]
        for o in n:
            if o == T0[0] or o == T0[1]:
                Max.append(0)
                continue
            F=math.sqrt((i[0]-o[0])**2+(i[1]-o[1])**2)
            Max.append(F)
        for x in range(4):
            if n[x]==i:
                T1.append(n[Max.index(max(Max))])
T1=list(set(T1))
To=[]
To.append(T0)
To.append(T1)
h=0
for s in To:
    for i in range(10):
        f=0
        for j in range(4):

            if T[i-h][j]==s[0] or T[i-h][j]==s[1]:
                f+=1
        if f==2:
            del T[i-h]
            del L[i-h]
            h+=1
O=[[],[]]
for i in combinations(L,2):
    if abs(np.rad2deg(np.arctan(i[0][0])-np.arctan(i[1][0])))<110\
       and abs(np.rad2deg(np.arctan(i[0][0])-np.arctan(i[1][0])))>60:
        if abs(np.rad2deg(np.arctan(i[0][0])))<50 and abs(np.rad2deg(np.arctan(i[1][0])))<50:
            if np.arctan(i[0][0])<0:
                O[0].append(i[0])
                O[1].append(i[1])
            else:
                O[1].append(i[0])
                O[0].append(i[1])
        else:
            if abs(np.arctan(i[0][0]))<abs(np.arctan(i[1][0])):
                O[0].append(i[0])
                O[1].append(i[1])
            else:
                O[1].append(i[0])
                O[0].append(i[1])
O0=O[0]
O1=O[1]
O0=list(set(O0))
O1=list(set(O1))
Set0=[]
Set1=[]
for i in O0:
    ind=L.index(i)
    Set0.append(T[ind])
for i in O1:
    ind=L.index(i)
    Set1.append(T[ind])

Dulj=[]
```

```
    for i in Set1:
        zbr=0
        for j in i:
            zbr+=j[0]
        Dulj.append(zbr/len(i))
    Set1=[x for (y,x) in sorted(zip(Dulj,Set1))]
    for i in Set1:
        i.sort(key=lambda x: x[1])

    Dulj=[]
    for i in Set0:
        zbr=0
        for j in i:
            zbr+=j[1]
        Dulj.append(zbr/len(i))
    Set0=[x for (y,x) in sorted(zip(Dulj,Set0))]
    for i in Set0:
        i.sort(key=lambda x: x[0])

    return Set0,Set1

S0,S1=najblize(listatocaka)

boja=[255,0,0]
for i in S1:
    for j in i:
        cv2.circle(bimg,(j[0],j[1]), 8,boja , -1)
        boja[0]-=20
        boja[2]+=20
    boja[1]+=50
objectpoints=[(0,0,0),(0,0.15,0),(0,0.3,0),(0,0.45,0),\
              (0.15,0,0),(0.15,0.15,0),(0.15,0.3,0),(0.15,0.45,0),\
              (0.3,0,0),(0.3,0.15,0),(0.3,0.3,0),(0.3,0.45,0),\
              (0.45,0,0),(0.45,0.15,0),(0.45,0.3,0),(0.45,0.45,0)]

objp=np.array(objectpoints,dtype=np.float32)
Toc= np.array(S1[0]+S1[1]+S1[2]+S1[3],dtype=np.float32)
camera_matrix = cv2.initCameraMatrix2D([objp],[Toc],\
                (int(math.ceil(nw)), int(math.ceil(nh))))
print camera_matrix
R1,R2,R3,R4,R5=cv2.calibrateCamera([objp]\
                    ,[Toc]\
                    ,(int(math.ceil(nw)), int(math.ceil(nh)))\
                    , camera_matrix, None, flags=cv2.CALIB_USE_INTRINSIC_GUESS)
h,  w = bimg.shape[:2]
newcameramtx, roi=cv2.getOptimalNewCameraMatrix(R2,R3,(w,h),1,(w,h))
bimg=cv2.undistort(bimg, R2, R3, None)
x,y,w,h = roi
bres = bres[y:y+h, x:x+w]
print R2
print R3
print np.rad2deg(R4)
print R5
cv2.imwrite('rez\\imag%04i.png'%cpt,bimg)
cv2.imshow('dst',bimg)
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

9.2. Eksperimentalni kod

```
import cv2
import numpy as np
import random
import time
import math
from itertools import combinations
import warnings
cpt=random.randrange(0,10000,1)
warnings.filterwarnings('error')

np.set_printoptions(threshold=np.nan)

camera_port=0
ramp_frames=60
camera=cv2.VideoCapture(camera_port)
def get_image():
   retval,im=camera.read()
   return im

for i in xrange(ramp_frames):
   temp =get_image()
print 'taking image...'
camera_capture = get_image()
file = "rez\\%ipocetnaslika1.png"%cpt
cv2.imwrite(file,camera_capture)
del(camera)
res = cv2.imread(file,0)
rows,cols = res.shape
bres=res
cv2.imshow('dst',res)
if cv2.waitKey(0) & 0xff == 27:
   cv2.destroyAllWindows()

class UnionFind:
   """Union-find data structure. Items must be hashable."""
   def __init__(self):
      """Create a new empty union-find structure."""
      self.weights = {}
```

```
        self.parents = {}

    def __getitem__(self, obj):
        """X[item] will return the token object of the set which contains `item`"""

        # check for previously unknown object
        if obj not in self.parents:
            self.parents[obj] = obj
            self.weights[obj] = 1
            return obj

        # find path of objects leading to the root
        path = [obj]
        root = self.parents[obj]
        while root != path[-1]:
            path.append(root)
            root = self.parents[root]

        # compress the path and return
        for ancestor in path:
            self.parents[ancestor] = root
        return root

    def union(self, obj1, obj2):
        """Merges sets containing obj1 and obj2."""
        roots = [self[obj1], self[obj2]]
        heavier = max([(self.weights[r],r) for r in roots])[1]
        for r in roots:
            if r != heavier:
                self.weights[heavier] += self.weights[r]
                self.parents[r] = heavier

def groupTPL(TPL, distance=25):
    U = UnionFind()

    for (i, x) in enumerate(TPL):
        for j in range(i + 1, len(TPL)):
            y = TPL[j]
            if max(abs(x[0] - y[0]), abs(x[1] - y[1])) <= distance:
                U.union(x, y)
```

```
    disjSets = {}
    for x in TPL:
        s = disjSets.get(U[x], set())
        s.add(x)
        disjSets[U[x]] = s

    return [list(x) for x in disjSets.values()]
def naditocke(slika1,slika2):
    gray = np.float32(slika1)
    dst=cv2.cornerHarris(gray,4,5,0.04)
    ind=np.where(dst>0.01*dst.max())
    x=[]
    listatocaka=[]
    p,d=list(ind[0]),list(ind[1])

    for i in range(len(p)):
        x.append((p[i],d[i]))
    grupirani=groupTPL(x)
    for i in grupirani:
        x=0
        y=0
        for j in i:
            y+=j[0]
            x+=j[1]
        listatocaka.append((int(round(float(x)/float(len(i)))),(int(round(float(y)/float(len(i)))))))
    dst = cv2.dilate(dst,None)
    img=cv2.cvtColor(slika2,cv2.COLOR_GRAY2RGB)
    img[dst>0.01*dst.max()]=[0,0,255]

    bimg=img
    cv2.imshow('rubovi',bimg)
    if cv2.waitKey(0) & 0xff == 27:
        cv2.destroyAllWindows()
    return listatocaka,bimg

def najblize(Azad):
    T=[]
    L=[]
```

```
for i in combinations(Azad,4):
    x = [i[0][0],i[1][0],i[2][0],i[3][0]]
    y = [i[0][1],i[1][1],i[2][1],i[3][1]]
    try:
        fit = np.polyfit(x,y,1)
    except Warning:

        funL=np.poly1d([0,int(round((x[0]+x[1]+x[2]+x[3])/4.))])
        T.append(list(i))
        L.append(list(funL)[0])
        continue
    funL = np.poly1d(fit)
    uda=0
    for h in range(4):
        uda+=abs(list(funL)[0]*float(x[h])-1.*float(y[h])+list(funL)[1])\
       /(math.sqrt((list(funL)[0])**2+1.**2))
    if uda < 30:
        T.append(list(i))
        L.append(tuple(funL))
N=[]
for i in L:
    n=L.index(i)
    Dv=[]
    for k in combinations(T[n],2):
        Dv.append(k)
    H=[]
    for j in Dv:
        F=math.sqrt((j[0][0]-j[1][0])**2+(j[0][1]-j[1][1])**2)
        H.append(F)


    N.append(Dv[H.index(max(H))])
Dij=[]
T0=[]
for n in N:
        F=math.sqrt((n[0][0]-n[1][0])**2+(n[0][1]-n[1][1])**2)
        Dij.append(F)
T0.append(N[Dij.index(max(Dij))])
p=[]
T0=list(T0[0])
```

```
for i in T:
    b=0
    for k in i:

        if k==T0[0] or k== T0[1]:
            b+=1
    if b==1:
        p.append(i)

T1=[]
for i in T0:
    for n in p:
        Max=[]
        for o in n:
            if o == T0[0] or o == T0[1]:
                Max.append(0)
                continue
            F=math.sqrt((i[0]-o[0])**2+(i[1]-o[1])**2)
            Max.append(F)
        for x in range(4):
            if n[x]==i:
                T1.append(n[Max.index(max(Max))])
T1=list(set(T1))
To=[]
To.append(T0)
To.append(T1)
h=0
for s in To:
    for i in range(10):
        f=0
        for j in range(4):

            if T[i-h][j]==s[0] or T[i-h][j]==s[1]:
                f+=1
        if f==2:
            del T[i-h]
            del L[i-h]
            h+=1
O=[[],[]]
for i in combinations(L,2):
```

```
        if abs(np.rad2deg(np.arctan(i[0][0])-np.arctan(i[1][0])))<110\
          and abs(np.rad2deg(np.arctan(i[0][0])-np.arctan(i[1][0])))>60:
           if abs(np.rad2deg(np.arctan(i[0][0])))<50 and abs(np.rad2deg(np.arctan(i[1][0])))<50:
              if np.arctan(i[0][0])<0:
                 O[0].append(i[0])
                 O[1].append(i[1])
              else:
                 O[1].append(i[0])
                 O[0].append(i[1])
           else:
              if abs(np.arctan(i[0][0]))<abs(np.arctan(i[1][0])):
                 O[0].append(i[0])
                 O[1].append(i[1])
              else:
                 O[1].append(i[0])
                 O[0].append(i[1])
##    print O
    O0=O[0]
    O1=O[1]
    O0=list(set(O0))
    O1=list(set(O1))
    Set0=[]
    Set1=[]
    for i in O0:
       ind=L.index(i)
       Set0.append(T[ind])
    for i in O1:
       ind=L.index(i)
       Set1.append(T[ind])

    Dulj=[]
    for i in Set1:
       zbr=0
       for j in i:
          zbr+=j[0]
       Dulj.append(zbr/len(i))
    Set1=[x for (y,x) in sorted(zip(Dulj,Set1))]
    for i in Set1:
       i.sort(key=lambda x: x[1])
```

```
    Dulj=[]
    for i in Set0:
        zbr=0
        for j in i:
            zbr+=j[1]
        Dulj.append(zbr/len(i))
    Set0=[x for (y,x) in sorted(zip(Dulj,Set0))]
    for i in Set0:
        i.sort(key=lambda x: x[0])

    boja=[255,0,0]
    for i in Set1:
        for j in i:
            cv2.circle(bimg,(j[0],j[1]), 8,boja , -1)
            boja[0]-=20
            boja[2]+=20
        boja[1]+=50

    cv2.imshow('dst',bimg)
    if cv2.waitKey(0) & 0xff == 27:
        cv2.destroyAllWindows()
    return Set0,Set1,bimg



def matrica():
    razpox=[]
    razpoy=[]
    l=[S1[0][0],S1[0][-1],S1[-1][0],S1[-1][-1]]
    for i in combinations(l,2):
        razpox.append((abs(i[0][0]-i[1][0]),i))
        razpoy.append((abs(i[0][1]-i[1][1]),i))

    dimx=max(s[0] for s in razpox)
    dimy=max(s[0] for s in razpoy)

    img = cv2.imread('tocni uzorak.png',0)
    rows,cols = img.shape
```

```
    for i in razpox:
        if i[0]==dimx:
            mrx=i
            break

    for i in razpoy:
        if i[0]==dimy:
            mry=i
            break
    print mrx,mry
    print l
    minx=min(s[0] for s in l)
    maxx=max(s[0] for s in l)
    miny=min(s[1] for s in l)
    maxy=max(s[1] for s in l)
    if mrx[0]>mry[0]:
        dim=mrx[0]
        k=float(cols)/float(dim)
        tocka1=[-int(k*(l[0][0]-minx)),-int(k*(l[0][1]-miny))]
        tocka2=[rows+int(k*(maxx-l[2][0])),-int(k*(l[2][1]-miny))]
        tocka3=[-int(k*(l[1][0]-minx)),rows+int(k*(-l[1][1]+miny+dim))]
        tocka4=[cols+int(k*(-l[3][0]+maxx)),rows+int(k*(-l[3][1]+miny+dim))]

        pts1 = np.float32([tocka1,tocka2,tocka3,tocka4])
        pts2 = np.float32([[0,0],[cols,0],[0,rows],[cols,rows]])

        M = cv2.getPerspectiveTransform(pts1,pts2)

        dst = cv2.warpPerspective(img,M,(cols,rows))

    if mrx[0]<mry[0]:
        dim=mry[0]
        k=float(cols)/float(dim)
        tocka1=[-int(k*(l[0][0]-minx)),-int(k*(l[0][1]-miny))]
        tocka2=[rows+int(k*(-l[2][0]+minx+dim)),-int(k*(l[2][1]-miny))]
        tocka3=[-int(k*(l[1][0]-minx)),rows+int(k*(-l[1][1]+maxy))]
        tocka4=[cols+int(k*(-l[3][0]+minx+dim)),rows+int(k*(-l[3][1]+maxy))]

        pts1 = np.float32([tocka1,tocka2,tocka3,tocka4])
        pts2 = np.float32([[0,0],[cols,0],[0,rows],[cols,rows]])
```

```
        M = cv2.getPerspectiveTransform(pts1,pts2)

        dst = cv2.warpPerspective(img,M,(cols,rows))



    cv2.imshow('dst',dst)
    if cv2.waitKey(0) & 0xff == 27:
        cv2.destroyAllWindows()
    return dim,M,pts1,dst

listatocaka,bimg=naditocke(res,res)
cv2.imwrite('rez\\%ipronadene1.png'%cpt,bimg)
S0,S1,bimg=najblize(listatocaka)
cv2.imwrite('rez\\%iodabrane1.png'%cpt,bimg)
dim1,M1,ts1,dst=matrica()
cv2.imwrite('rez\\%iTOCNI1.png'%cpt,dst)

u=0.137
objectpoints=[(0,0,0),(0,u,0),(0,2*u,0),(0,3*u,0),\
            (u,0,0),(u,u,0),(u,2*u,0),(u,3*u,0),\
            (2*u,0,0),(2*u,u,0),(2*u,2*u,0),(2*u,3*u,0),\
            (3*u,0,0),(3*u,u,0),(3*u,2*u,0),(3*u,3*u,0)]

objp=np.array(objectpoints,dtype=np.float32)
Toc= np.array(S1[0]+S1[1]+S1[2]+S1[3],dtype=np.float32)
camera_matrix = cv2.initCameraMatrix2D([objp],[Toc],\
              (cols, rows))

R1,R2,R3,R4,R5=cv2.calibrateCamera([objp]\
                    ,[Toc]\
                    ,(cols, rows)\
                    , camera_matrix, None, flags=cv2.CALIB_USE_INTRINSIC_GUESS)

h,  w = bimg.shape[:2]
newcameramtx, roi=cv2.getOptimalNewCameraMatrix(R2,R3,(w,h),1,(w,h))
print R2,R3
ret,res = cv2.threshold(res,80,255,cv2.THRESH_BINARY_INV)
cv2.imshow('dst',res)
```

```
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()

res=cv2.undistort(res, R2, R3, None)

listatocaka,bimg=naditocke(res,res)
cv2.imwrite('rez\\%ipronadene1.png'%cpt,bimg)
S0,S1,bimg=najblize(listatocaka)
cv2.imwrite('rez\\%iodabrane1.png'%cpt,bimg)
dim1,M1,ts1,dst=matrica()
cv2.imwrite('rez\\%iTOCNI1.png'%cpt,dst)

camera=cv2.VideoCapture(camera_port)
def get_image():
    retval,im=camera.read()
    return im

for i in xrange(70):
    temp =get_image()
print 'taking image...'
camera_capture = get_image()
file = "rez\\%ipocetnaslika2.png"%cpt
cv2.imwrite(file,camera_capture)
del(camera)
res = cv2.imread(file,0)


rows,cols = res.shape
bres=res

cv2.imshow('dst',res)
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()

listatocaka,bimg=naditocke(res,res)
cv2.imwrite('rez\\%ipronadene2.png'%cpt,bimg)
S0,S1,bimg=najblize(listatocaka)
cv2.imwrite('rez\\%iodabrane2.png'%cpt,bimg)
dim2,M2,ts2,dst=matrica()
cv2.imwrite('rez\\%iTOCNI2.png'%cpt,dst)
```

```
img = cv2.imread('tocni uzorak.png',0)
rows,cols = img.shape



pts1=np.around(ts1, decimals=0)
pts2=np.around(ts2, decimals=0)
M = cv2.getPerspectiveTransform(pts2,pts1)

dst = cv2.warpPerspective(img,M,(cols+150,rows+150))
pts1 = np.float32([[0,0],[cols,0],[0,rows],[cols,rows])
pts2 = np.float32([[0,0],[cols,0],[0,rows],[cols,rows]])

cv2.imwrite('rez\\%iTOCNI3.png'%cpt,dst)
cv2.imshow('dst',dst)
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
tocka1=[int(abs(-ts2[0][0]+ts1[0][0])),int(abs(-ts2[0][1]+ts1[0][1]))]
tocka2=[cols-int(abs(-ts2[1][0]+ts1[1][0])),int(abs(-ts2[1][1]+ts1[1][1]))]
tocka3=[int(abs(ts2[2][0]-ts1[2][0])),rows-int(abs(-ts2[2][1]+ts1[2][1]))]
tocka4=[cols-int(abs(-ts2[3][0]+ts1[3][0])),rows-int(abs(-ts2[3][1]+ts1[3][1]))]
l=[tocka1,tocka2,tocka3,tocka4]
img = cv2.imread('tocni uzorak.png',0)
rows,cols = img.shape
print l
lx=[]
ly=[]
for i in l:
    lx.append(i[0])
for i in l:
    ly.append(i[1])
dminx=sorted(lx)[1]
dmaxx=sorted(lx)[2]
dminy=sorted(ly)[1]
dmaxy=sorted(ly)[2]
mrx=abs(dminx-dmaxx)
mry=abs(dminy-dmaxy)
if mrx>mry:
    dim=mry
    print 'mrx'
    tocka1=[-int(abs(-l[0][0]+dminx)),-int(abs(-l[0][1]+dminy))]
```

```
    tocka2=[cols+int(abs(dmaxx+l[1][0])),-int(abs(-l[1][1]+dminy))]
    tocka3=[-int(abs(-l[2][0]+dminx)),rows+int(abs(+l[2][1]-dminy-dim))]
    tocka4=[cols+int(abs(+l[3][0]-dmaxx)),rows+int(abs(+l[3][1]-dminy-dim))]

    pts1 = np.float32([tocka1,tocka2,tocka3,tocka4])
    pts2 = np.float32([[0,0],[cols,0],[0,rows],[cols,rows]])

    M = cv2.getPerspectiveTransform(pts1,pts2)

    dst = cv2.warpPerspective(img,M,(cols,rows))

if mrx<mry:
    dim=mrx
    print'mry'
    tocka1=[-int(abs(-l[0][0]+dminx)),-int(abs(-l[0][1]+dminy))]
    tocka2=[cols+int(abs(l[1][0]-dminx-dim)),-int(abs(-l[1][1]+dminy)+50)]
    tocka3=[-int(abs(-l[2][0]+dminx)),rows+int(abs(l[2][1]-dmaxy))]
    tocka4=[cols+int(abs(l[3][0]-dminx-dim)),rows+int(abs(l[3][1]-dmaxy))]

    pts1 = np.float32([tocka1,tocka2,tocka3,tocka4])
    pts2 = np.float32([[0,0],[cols,0],[0,rows],[cols,rows]])

    M = cv2.getPerspectiveTransform(pts1,pts2)

    dst = cv2.warpPerspective(img,M,(cols,rows))

cv2.imwrite('rez\\%iTOCNI4.png'%cpt,dst)
cv2.imshow('dst',dst)
if cv2.waitKey(0) & 0xff == 27:
    cv2.destroyAllWindows()
```

9.3. Funkcija za računanje matrice rotacije za transformaciju iz koordinatnog sustava kamere u koordinatni sustav projektora bez korištenja fizičkih značajki

```
def ispravi():
    camera=cv2.VideoCapture(camera_port)
    for i in xrange(70):
        temp =get_image(camera)
    print 'taking image...'
    camera_capture = get_image(camera)
    file = "rez\\velika.jpg"
    cv2.imwrite(file,camera_capture)
    del(camera)
    res = cv2.imread(file,0)
    rows,cols = res.shape
    bres=res
    cv2.imshow('dst',res)
    if cv2.waitKey(0) & 0xff == 27:
        cv2.destroyAllWindows()
    R1,R2,R3,R4,R5 = kalibriraj()
    print R1
    print R2
    print R3
    print R4
    print R5
    listatocaka,bimg=naditocke(res,res)
    cv2.imwrite('rez\\%ipronadene1.png'%cpt,bimg)
    S0,S1,bimg=najblize(listatocaka,bimg)
    l1=[S1[0][0],S1[0][-1],S1[-1][0],S1[-1][-1]]
    cv2.imwrite('rez\\%iodabrane1.png'%cpt,bimg)
    dim1,M1,ts1,dst1=matrica(S1)
    file = "INST.png"
    INST = cv2.imread(file,0)
    cv2.imshow('dst',INST)
    if cv2.waitKey(0) & 0xff == 27:
        res = cv2.imread(file,0)
    rows,cols = res.shape
    bres=res
```

```
img = cv2.imread('tocni uzorak.png',0)
rows,cols = img.shape
print rows,cols
f=float(dim1)/float(cols)
lx=[]
ly=[]
for i in l1:
    lx.append(i[0])
for i in l1:
    ly.append(i[1])
dminx=sorted(lx)[1]
dmaxx=sorted(lx)[2]
dminy=sorted(ly)[1]
dmaxy=sorted(ly)[2]
mrx=abs(dminx-dmaxx)
mry=abs(dminy-dmaxy)

if mrx>mry:
    dim=mry
    ff=float(dim)/float(cols)
    koe=ff
    koe1=1./ff
    print koe1
    print dim,'mrx'
    tocka1=[-int(ff*abs(-l1[0][0]+dminx)),-int(float(abs(-l1[0][1]+dminy)))]
    tocka2=[cols+int(ff*abs(dmaxx-l1[1][0])),-int(abs(-l1[1][1]+dminy))]
    tocka3=[-int(ff*abs(-l1[2][0]+dminx)),rows+int(ff*abs(l1[2][1]-dminy-dim))]
    tocka4=[cols+int(abs(+l1[3][0]-dim-dminx)),rows+int(ff*abs(+l1[3][1]-dminy-dim))]
    print tocka4
    pts1 = np.float32([tocka1,tocka2,tocka3,tocka4])
    pts2 = np.float32([[0,0],[cols,0],[0,rows],[cols,rows]])
    M = cv2.getPerspectiveTransform(pts1,pts2)
    print M
    dst = cv2.warpPerspective(img,M,(cols,rows))
if mrx<mry:
    dim=mrx
    print dim,'mry'
    tocka1=[-int(koe*abs(-l1[0][0]+dminx)),-int(koe*abs(-l1[0][1]+dminy))]
```

```
        tocka2=[cols+int(koe*abs(l1[1][0]-dminx-dim)),-int(koe*abs(-l1[1][1]+dminy))]
        tocka3=[-int(koe*abs(-l1[2][0]+dminx)),rows+int(koe*abs(l1[2][1]-dmaxy))]
        tocka4=[cols+int(koe*abs(l1[3][0]-dminx-dim)),rows+int(koe*abs(l1[3][1]-dmaxy))]

        pts1 = np.float32([tocka1,tocka2,tocka3,tocka4])
        pts2 = np.float32([[0,0],[cols,0],[0,rows],[cols,rows]])

        M = cv2.getPerspectiveTransform(pts1,pts2)
        print M
        dst = cv2.warpPerspective(img,M,(cols,rows))
    cv2.imwrite('rez\\%iTOCNI4.png'%cpt,dst)
    cv2.imshow('dst',dst)
    if cv2.waitKey(0) & 0xff == 27:
        cv2.destroyAllWindows()
import cv2.cv as cv
def prozor():
    import time

    cv.NamedWindow("camera", 0)

    capture = cv.CaptureFromCAM(0)

    while True:
        img = cv.QueryFrame(capture)
        cv.ShowImage("camera", img)
        if cv.WaitKey(10) == 27:
            break
    cv.DestroyAllWindows()
```

LITERATURA

[1] Z. Zhang, A flexible new technique for camera calibration, IEEE Transactions on Pattern Analysis and Machine Intelligence Web Decker, K. H.: Elementi strojeva, Tehnička knjiga Zagreb, 1975.

[2] OpenCV Dokumentacija http://docs.opencv.org/

[3] Gabriel Falcao, Natalia Hurtos, Joan Massich, Plane-based calibration of a projector-camera system

[4] Michael Brown, Aditi Majumder, Ruigang Yang, Camera-based calibration Techniques for seamless multi-projector diyplays