Integracija i validacija simuliranog radnog okruženja za robotsku ruku u realnoj i virtualnoj konfiguraciji oko-ruka

Dominiković, Marin

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Marin Dominiković

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Prof. dr. sc. Filip Šuligoj, dipl. ing.

Student:

Marin Dominiković

Zagreb, 2024.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem se mentoru prof. dr. sc. Filipu Šuligoju na pomoći i na savjetima prilikom izrade završnog rada.

Također, zahvaljujem se svojim roditeljima, djevojci i prijateljima na podršci tijekom studija.

Marin Dominiković

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i 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	Prilog
Klasa: **602 – 04 / 24 – 06 / 1**	
Ur.broj: **15 – 24 –**	

ZAVRŠNI ZADATAK

Student: **Marin Dominiković** JMBAG: **0035222048**

Naslov rada na hrvatskom jeziku: **Integracija i validacija simuliranog radnog okruženja za robotsku ruku u realnoj i virtualnoj konfiguraciji oko-ruka**

Naslov rada na engleskom jeziku: **Integration and validation of a simulated working environment for a robotic arm in real and virtual eye-hand configuration**

Opis zadatka:

Robotske ruke proizvođača Universal Robot predstavljaju jedne od najraširenijih platformi u industrijskoj kolaborativnoj robotici. S obzirom na sveprisutnu potrebu za automatizacijom i optimizacijom proizvodnih procesa, ovaj rad cilja na razvoj i validaciju simuliranog radnog okruženja korištenjem RoboDK simulacijske platforme. Integracijom 2D kamere u konfiguraciji oko-ruka, rad će obuhvatiti kalibraciju sustava i detekciju objekata u stvarnom i simuliranom prostoru. Automatsko generiranje objekata u simulacijskom okruženju nakon njihove detekcije u stvarnom svijetu omogućuje precizno planiranje i provjeru zadatka manipulacije prije njihove fizičke izvedbe.

U sklopu završnog rada potrebno je:

- Postaviti i kalibrirati robotsku ruku i 2D kameru u konfiguraciji oko-ruka kako bi se omogućila precizna detekcija i lokalizacija objekata.
- Implementirati algoritam za detekciju i lokalizaciju poznatih objekata u fizičkom radnom prostoru robota koristeći metode strojnog vida.
- Razviti softverski modul koji će automatski generirati detektirane objekte u RoboDK simulacijskom okruženju.
- Koristeći RoboDK, izraditi planove za manipulaciju objektima u simuliranom okruženju, uključujući analizu putanja i provjeru kolizija.
- Nakon simulacije, izvršiti fizičku manipulaciju objektima koristeći robotsku ruku.
- Provesti kvantitativne i kvalitativne analize usporedbom performansi i točnosti između simuliranog i stvarnog robotskog sustava, s posebnim naglaskom na parametre poput točnosti, robusnosti i efikasnosti.

U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
30. 11. 2023.	**1. rok:** 22. i 23. 2. 2024. **2. rok (izvanredni):** 11. 7. 2024. **3. rok:** 19. i 20. 9. 2024.	**1. rok:** 26. 2. – 1. 3. 2024. **2. rok (izvanredni):** 15. 7. 2024. **3. rok:** 23. 9. – 27. 9. 2024.

Zadatak zadao:

Doc. dr. sc. Filip Šuligoj

Predsjednik Povjerenstva:

Prof. dr. sc. Damir Godec

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

SAŽETAK

U kontekstu napredovanja automatizacije i poboljšanja učinkovitosti proizvodnih procesa, ovaj rad istražuje pristup razvoju i validaciji simuliranog radnog okruženja koristeći RoboDK. Istraživanje se fokusira na integraciju 2D kamere u konfiguraciji oko-ruka kako bi se olakšala kalibracija vizualnog sustava i detekcija objekata unutar stvarnih i simuliranih domena. Srž rada leži u implementaciji metoda lokalizacije predmeta, koristeći 2D kameru za stvaranje dinamičkog digitalnog blizanca unutar RoboDK simulacijskog okruženja. Teorijske osnove, uključujući metodologije usklađivanja značajki i dvostruke procese kalibracije za robotsku ruku i kameru, temeljito su istražene i objašnjene. Koristeći OpenCV za praktičnu implementaciju, rad detaljno opisuje primjenu dvije različite tehnike kalibracije uz primjenu dva algoritma za usklađivanje značajki. Nakon toga, demonstrira se integracija ovih elemenata u RoboDK Python API-u, što rezultira automatskim generiranjem i vizualizacijom detektiranih objekata iz stvarnog svijeta unutar simulacije. Ovaj automatizirani proces ne samo da olakšava precizno planiranje i verifikaciju robotskih manipulacija, već i osigurava njihovu točnu izvedbu u stvarnom svijetu. Rad završava procjenom performansi sustava, prezentirajući empirijska mjerenja odstupanja od stvarnih vrijednosti, čime se naglašava učinkovitost i pouzdanost predloženog rješenja u optimizaciji proizvodnih procesa kroz napredne simulacijske tehnike.

Ključne riječi: robotska ruka, oko-ruka, OpenCV, RoboDK

SUMMARY

In the context of advancing automation and improving the efficiency of manufacturing processes, this paper explores an approach to developing and validating a simulated work environment using RoboDK. The research focuses on integrating a 2D camera in an eye-to-hand configuration to facilitate the calibration of the vision system and the detection of objects within both real and simulated domains. The essence of the work lies in the implementation of object localization methods, using a 2D camera to create a dynamic digital twin within the RoboDK simulation environment. Theoretical foundations, including feature matching methodologies and dual calibration processes for both the robotic arm and the camera, are thoroughly investigated and explained. Utilizing OpenCV for practical implementation, the paper details the application of two different calibration techniques along with the deployment of two feature matching algorithms. Subsequently, the integration of these elements into the RoboDK Python API is demonstrated, resulting in the automatic generation and visualization of detected real-world objects within the simulation. This automated process not only facilitates precise planning and verification of robotic manipulations but also ensures their accurate execution in the real world. The paper concludes with an assessment of the system's performance, presenting empirical measurements of deviations from actual values, thereby highlighting the efficiency and reliability of the proposed solution in optimizing manufacturing processes through advanced simulation techniques.

Key words: robotic arm, eye-to-hand, OpenCV, RoboDK

1. UVOD

Ključnu ulogu u transformaciji i napretku današnjeg svijeta ima automatizacija, povećavajući produktivnost i kvalitetu, istovremeno smanjujući troškove. Jedan od sinonima za automatizaciju je i robotska ruka, svestrani alat koji može izvršavati ponavljajuće zadatke s preciznošću i učinkovitošću. Od proizvodnih montažnih linija do kirurških zahvata, robotske ruke su revolucionirale različite sektore, smanjujući ljudske pogreške i povećavajući proizvodnju. Osim toga, integracija vizualnih sustava s robotskim rukama dodatno je proširila njihove sposobnosti. Vizijski sustavi omogućuju robotima da percipiraju i tumače svoje okruženje, omogućujući dinamično donošenje odluka i prilagodbu u složenim okruženjima. Spajanjem industrijskih kamera i algoritama, robotske ruke opremljene vizualnim sustavima mogu obavljati zadatke s većom točnošću, brzinom i sigurnošću, čime se optimiziraju proizvodni zadaci i potiče inovacija u različitim industrijama.

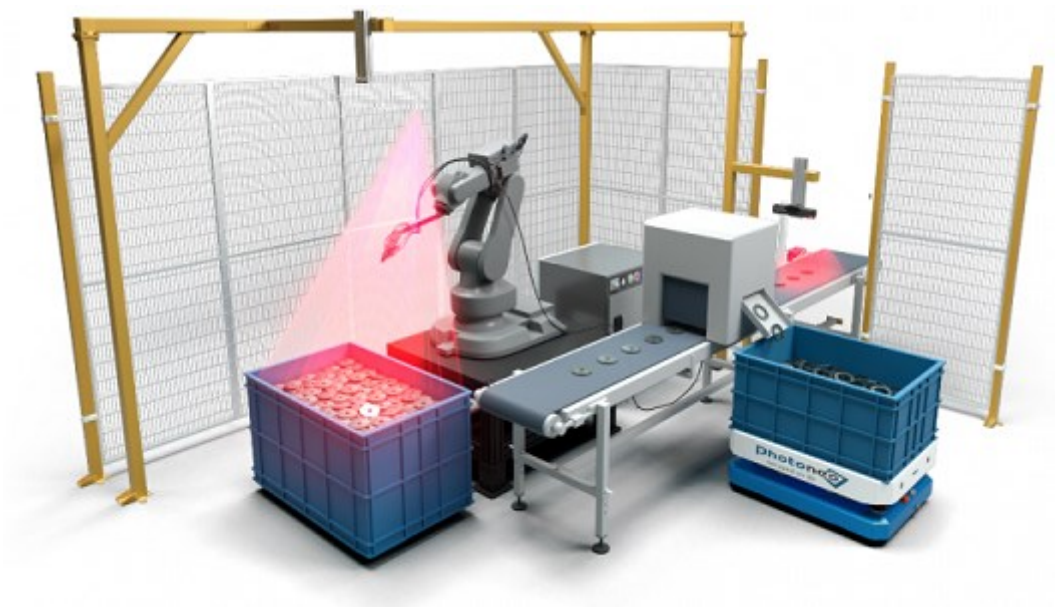
Paralelno s tim, dolazak digitalnih blizanaca značajno je utjecao na krajolik proizvodnje, pa tako i automatizacije. Digitalni blizanci virtualne su replike fizičkih sustava ili procesa, omogućavajući nadzor u stvarnom vremenu, analizu i optimizaciju. Stvaranjem digitalnog blizanca robotske ruke ili cijele proizvodne linije tvrtke mogu simulirati različite scenarije, identificirati potencijalne neefikasnosti i implementirati strategije prediktivnog održavanja. Ovaj proaktivni pristup ne samo da smanjuje vrijeme neprekidnosti, već i optimizira iskorištavanje resursa i poboljšava ukupnu učinkovitost sustava. Dodatno, digitalni blizanci olakšavaju daljinsko upravljanje i suradnju, omogućujući stručnjacima da nadgledaju i optimiziraju procese iz bilo kojeg dijela svijeta. Dok automatizacija nastavlja evoluirati, integracija robotskih ruku, vizualnih sustava i digitalnih blizanaca igrat će sve važniju ulogu u povećanju učinkovitosti, inovacije i konkurentnosti u globalnoj ekonomiji.

U ovome radu će se predstaviti pojednostavljeno rješenje razvoja digitalnog blizanca za konfiguraciju robotske ruke i kamere koje će omogućiti prepoznavanje predmeta i njihovih poza u prostoru te generaciju istih u simuliranom okruženju koristeći softverske pakete RoboDK i OpenCV.

1.1. STROJNI VID

Kako bi roboti i strojevi izvodili radnje bez ljudske pomoći ili unaprijed programiranih radnji, potrebni su im alati za percepciju svog okoliša. Jedan, ako ne i jedini, način za digitalnu percepciju svijeta oko sebe je korištenje strojnog vida. Strojni vid je sveobuhvatan naziv za

tehnologiju i metode koje se koriste za primanje, izvlačenje i obradu informacija na temelju slika za potrebe automatske inspekcije, upravljanja i kontrole procesa. Naziv također ne opisuje samo korišten softver pri obradi informacija, nego i hardver korišten tijekom primanja i obrade informacija kao i svih popratnih dijelova složenog sustava. Primjerice, jedan sustav strojnog vida se sastoji od kamere ili više njih, leća, osvjetljenja, softvera koji se služi računalnim vidom za obradu i analizu slike kao i algoritama za pronalaženje bitnih značajki te na kraju izlaza u obliku zaslona ili pokreta robotske ruke. Zbog primjene raznovrsnog tehničkog znanja, strojni vid se smatra disciplinom sistemskog inženjerstva. Razne su primjene strojnog vida, ali su najčešće u industrijskim okruženjima za planiranje puta robotske ruke ili autonomnih robota, planiranje hvata za robotske hvataljke, kontrolu kvalitete prilikom montaže, dok se u posljednje vrijeme razvija njena primjena u autonomnoj vožnji.



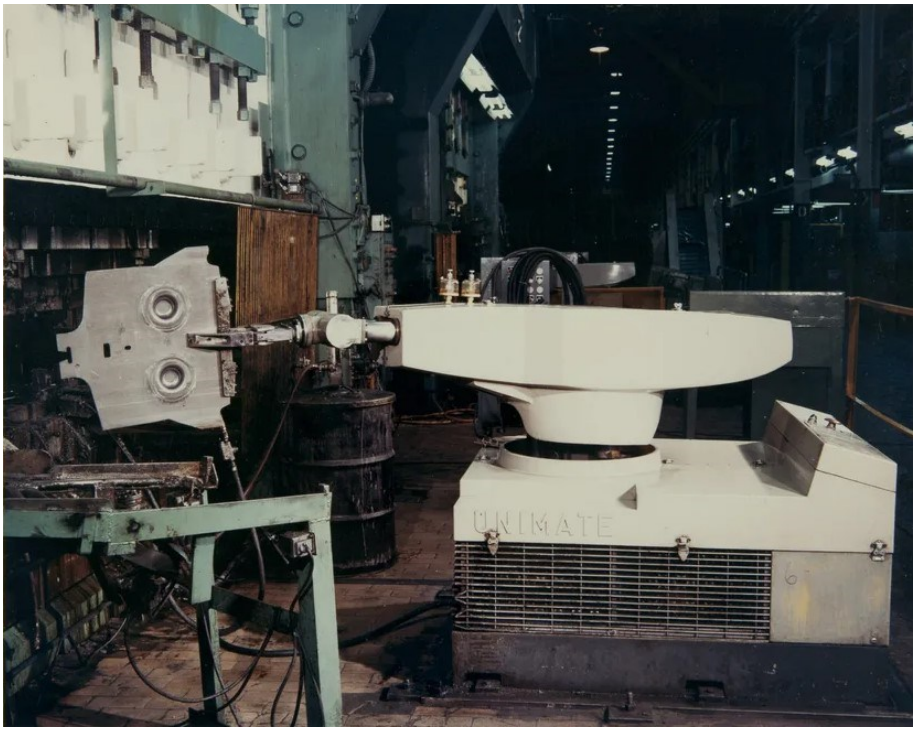
Slika 1.1 Prikaz robotskog sustava s vizijskim sustavom

Računalni vid se posebno ističe u ovoj disciplini zbog svoje kompleksnosti i spektra rješenja i pristupa kojima raspolaže. Glavni joj je zadatak krajnje razumijevanje i stvaranje odluka temeljenih na dobivenim podacima korištenjem teoretskih i algoritamskih pristupa. Glavna primjena u robotici je prepoznavanje i lokaliziranje predmeta, što je ključno za osposobljavanje robota za autonomno rukovanje predmetima.

1.2. ROBOTSKE RUKE

Povijest robota i robotskih ruku zanimljivo je putovanje koje se prepliće s evolucijom tehnologije i industrije. Koncept automata, ili samostalno djelujućih strojeva, seže unatrag do drevnih civilizacija, ali tek u 20. stoljeću postavljeni su temelji za modernu robotiku. Sam pojam "robot" skovao je 1920. godine češki dramatičar Karel Čapek u svojoj drami "R.U.R." ili "Rossumovi univerzalni roboti", prikazujući budućnost u kojoj umjetna bića nazvana "roboti" obavljaju rad za ljude.

Razvoj robotskih ruku, međutim, doživio je značajan skok u 1950-im i 1960-im godinama. Prvi programabilni robot, nazvan Unimate, predstavljen je na proizvodnoj liniji General Motorsa 1961. godine. Ova robotska ruka obavljala je zadatke koji su bili teški u opasnim uvjetima, revolucionirajući industriju proizvodnje. Označio je početak upotrebe robotskih ruku u industrijskim primjenama, postavljajući temelje za široku primjenu robotike u proizvodnji.



Slika 1.2 Prvi industrijski robot „Unimate“

Od tada, napredak u računalnoj tehnologiji, znanosti o materijalima i umjetnoj inteligenciji doveo je do razvoja sofisticiranijih i svestranijih robotskih sustava. Danas su robotske ruke ključne komponente moderne proizvodnje te kad se spominju u kontekstu strojarstva, obično se govori o SCARA ili delta robotima kao i najčešće rukama sa 6 stupnjeva slobode. Obavljajući

zadatke s preciznošću i učinkovitošću koju ljudi ne mogu doseći, roboti se koriste se u širokom rasponu industrija, od montaže automobila do proizvodnje elektronike, pa čak i u delikatnim područjima poput farmaceutike i prehrambene industrije.

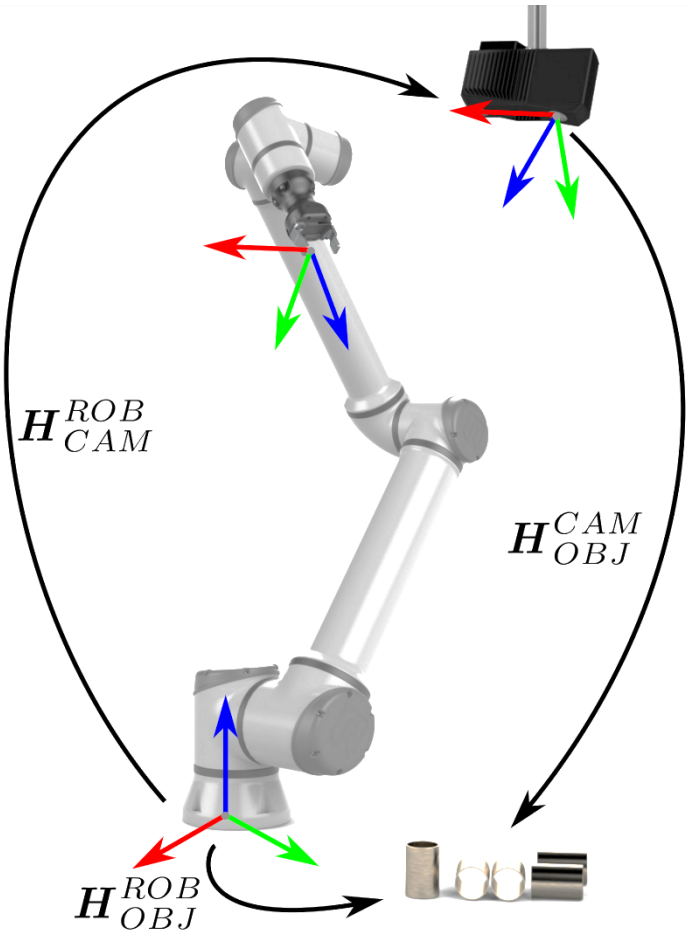
Neprestani napredak u robotici i dalje pomjera granice mogućeg, vodeći do sve više kolaborativnih robota (koboti) koji rade uz ljude, dodatno poboljšavajući produktivnost i inovacije. Povijest robota i robotskih ruku nije samo kronika tehnološkog napretka, već svjedočanstvo ljudske domišljatosti i neprekidne težnje za efikasnošću i napretkom.

1.3. Pristup problemu

Cilj ovog završnog rada je osmisliti sustav koji će prepoznavati i lokalizirati predmete u radnom prostoru robota te ih simultano generirati u simulacijskom softveru RoboDK i automatski planirati hvatanje predmeta. Postoji bezbroj pristupa kako detektirati i lokalizirati predmete koristeći metode dubokog učenja koje su u nekim slučajevima robusnije fleksibilnije, no zato dosta kompliciranije za rad i postavljanje za razliku od algoritamskih pristupa, te će se zbog jednostavnosti, ovaj rad se bazirati na klasičnim metodama računalnog vida. Klasični pristupi poput „template-matchinga“ i „feature-matchinga“ često i jesu baza raznim metodama dubokog učenja, no prvo je potrebno razumjeti osnove da bi se olakšao rad i razumijevanje naprednih metoda. Iako se slike mogu obrađivati na bezbroj načina, slijed operacija se uglavnom svodi na pre-processing da bi se slika očistila od šuma i nepotrebnih informacija, te da bi se istaknule nama ključne informacije i zatim izvlačenje bitnih podataka pomoću kontura, značajki ili boja. Ovakav pristup je relativno brz i efikasan, pogotovo uzimajući u obzir računalnu snagu današnjih računala. Baš zbog te efikasnosti su iskorišteni „feature-matching“ algoritmi čija je implementacija poprilično jednostavna, no i dalje dovoljno robusna da nemaju problema s velikim spektrom predmeta. Nakon pronalaska ključnih točaka u sceni, točke koje se podudaraju će biti iskorištene za homografiju koja će nam dati matricu transformacije, točnije podatke o pozi nekog predmeta na slici. Korištenjem te iste transformacije moguće je transformirati granični okvir predmeta iz kojeg možemo izračunati geometrijsko središte predmeta i tako dobiti središnju točku predmeta u pikselima.

Nakon što pronađemo središte traženog predmeta, ključno je transformirati koordinate kamere u koordinatni sustav robota kroz postupak eye-to-hand kalibracije. Ovaj složeni proces zahtijeva pronalaženje definirane kalibracijske ploče poznatih dimenzija unutar vidnog polja kamere, koja je montirana na hvataljku robota, te bilježenje poza zglobova robota. Eye-to-hand kalibracija omogućuje precizno određivanje transformacijske matrice koja opisuje odnos

između koordinatnog sustava kamere i koordinatnog sustava robota, olakšavajući tako transformaciju koordinata predmeta u koordinatni sustav robota. Ključni koraci u ovom procesu uključuju prikupljanje obilja slika kalibracijske ploče iz različitih položaja i orijentacija kamere, te zapisivanje poza zglobova robota u istim situacijama. Kvalitetan set podataka, prikupljen uz raznolike uvjete i varijacije, ključan je za točnost i pouzdanost kalibracije. Stoga, akumulacija što većeg broja slika i poza robota u različitim scenarijima značajna je kako bi kalibracijski postupak bio što precizniji i pouzdaniji, što rezultira optimalnim performansama robotskog sustava u detekciji i manipulaciji predmeta.



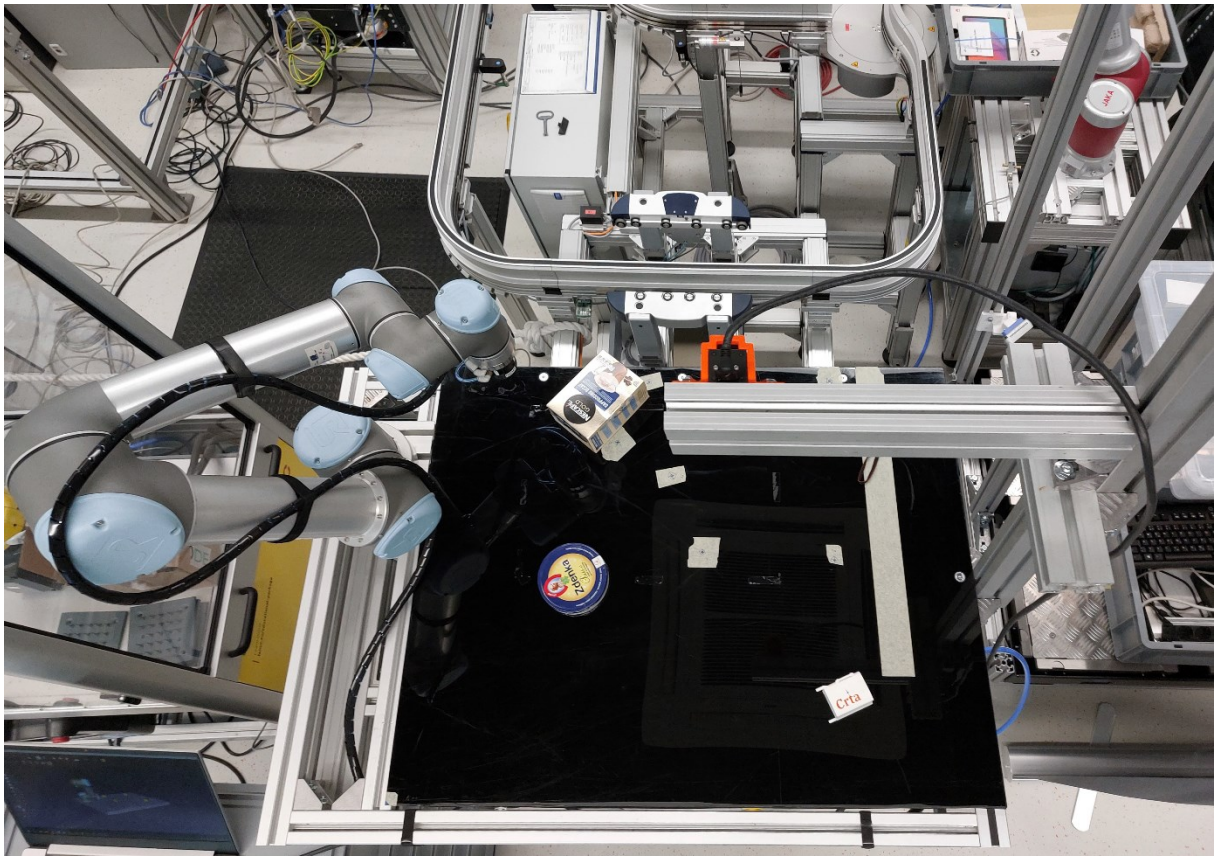
Slika 1.3 Transformacije koordinatnih sustava [1]

Slike će se prikupljati pomoću industrijske kamere fiksno montirane iznad radnog prostora robota čije je vidno polje dovoljno široko da pokrije cijeli stol. Glavni ograničavajući faktor je rezolucija kamera zbog čega je potrebno kameru montirati na visinu na kojoj neće ometati robota u kretanju no da se i dalje sačuvaju svi detalji na slici.

Nakon pronalaska središta i rotacije predmeta, traženi predmeti će se translatirati i zarotirati u RoboDK gdje će biti prikazana virtualna preslika njih u stvarnom svijetu. Simulacijsko okruženje nema samo ulogu za prikaz predmeta, već i za planiranje putanje za hvatanje. Korištenjem RoboDK Python API-ja (application programming interface), moguće je jednostavno povezati i koristiti podatke dobivene s pomoću OpenCV-a. Kako je zamišljeno da se za hvatanje koristi vakuumska hvataljka, podaci o točnoj veličini predmeta nisu potrebni, samo gdje se nalazi središte te na kojoj je visini. RoboDK također sadrži sve potrebne upravljačke programe za pomicanje robota, što omogućuje jednostavno povezivanje i upravljanje stvarnim robotima unutar simulacijskog okruženja.

2. POSTAV SUSTAVA

Sustav za robotsko rukovanje i simulaciju se sastoji od nekoliko glavnih elemenata koji su često zastupljeni u industrijskom okruženju s čime je osigurana fleksibilnost i podrška za pojednostavljeno postavljanje i korištenje. Sustav strojnog vida se sastoji od industrijske kamere ImagingSource DFK AFUX178-M12 za prikupljanje podataka i programa u Pythonu koji je razvijen koristeći OpenCV programsku knjižnicu za obradu slike. Hvatanje predmeta vrši se kolaborativnim robotom Universal Robots UR5 s vakuumskom hvataljkom. Korišten simulacijski softver RoboDK se služi za prikaz virtualnog okruženja i planiranje putanje robota. Dodatno, sva komunikacija između robota i računala odvija se putem TCP/IP protokola na lokalnoj mreži.



Slika 2.1 Postav robotskog sustava

2.1. 2D industrijska kamera

Neizostavnu stavku svakog vizijskog sustava, a tako i automatiziranog, čine industrijske kamere, koje se zbog svoje konstrukcije i funkcionalnosti razlikuju od svakodnevnih kamera. Zbog uvjeta u kojima se često nalaze i svrhe koje imaju, industrijske kamere moraju biti robusne i visoko funkcionalne što znači da trebaju biti otporne na proizvodne uvjete u kojima su često

povišene temperature i koncentracije prašine i raznih česta kao i veća učestalost vibracija i vlage. Također, moraju biti lako prilagodljive i podesive a istovremeno i dosljedne, pri čemu proizvođači nude softver kojim se lako da prilagođavati razne postavke kamere, poput fokusa, ekspozicije, kontrasta i sličnog. Uz to, moraju pružati slike visoke rezolucije čime se znatno olakšava pronalaženje bitnih detalja i te nam istovremeno daje mogućnost da digitalno izrežemo sliku kako bi smanjili potrebno računalnu snagu za obradu slike bez znatnog gubitka informacija.

ImagingSource DFK AFUX178-M12, koja se odlikuje maksimalnom rezolucijom od 3072x2048 piksela, USB 3.0 sučeljem i 1/1.8 inčnim Sony CMOS IMX178LQJ senzorom. Izabrana kamera sadrži upravljački program i popratni softver, *IC Imaging Control*, koji omogućuje jednostavno povezivanje i podešavanje parametara kamere. Dodatno se može i pronaći sva tehnička dokumentacija kao i nacrti što pojednostavljuje izradu nosača i potencijalne dodatne opreme. Iako kamera dolazi s relativno malim senzorom, u industrijskim primjenama to često nema prevelikog utjecaja jer korisnici kamere mogu utjecati na osvjetljenje. Kamere s većim senzorima pri istoj rezoluciji imaju veće piksele od manjih, što dopušta senzorima da prime više svjetla i time smanje šum. To se danas može lako primijetiti na mobilnim uređajima koji koriste male senzore; fotografije na dnevnom svjetlu ili pod jakim svjetlima izgledaju odlično, no u tamnijim okruženjima se pojavljuje šum i umanjenje detalja.

Slika 2.2 ImagingSource DFK AFUX178-M12

ImagingSource DFK AFUX178-M12

Dimenzije	36x36x30 mm
Senzor	Sony IMX178LQJ
Rezolucija	3072x2048 px
Broj slika u sekundi	22 fps pri 3072x2048px
Komunikacija	USB 3.0
Radna temperatura	od -5 ℃ do 45 ℃

Tablica 2.1 Specifikacije industrijske kamere

2.2. Kolaborativni robot

U naravi rješavanja ovakvog problema se nalazi potreba za što fleksibilnijim robotom te su kolaborativni roboti odličan izbor za takav problem. Ono po čemu se razliku je od tradicionalnih robotskih ruku je to što ne zahtijevaju nužno kaveze i barijere, već u sebi imaju ugrađene senzore sile i druge algoritamske značajke koje ih sprečavaju da nanesu štetu sebi, a i još važnije ljudima oko sebe. Jedna pozitivna posljedica korištenja senzora sile je to što dozvoljava laku interakciju s ljudima, točnije, robot se može jednostavno pomicati guranjem, te je i stoga olakšano programiranje i dovođenje robota u željene pozicije. Universal Robots UR5 se pronašao kao odličan izbor jer sadrži sve navedene predispozicije te nudi šest stupnjeva slobode, nosivost od 5 kilograma, doseg od 850mm te privjesak za učenje s korisničkim sučeljem PolyScope. Privjesak pruža praktičan i intuitivan način za programiranje robota, upravljanje digitalnim izlazima i postavljanjem sigurnosnih ravnina.

Slika 2.3 Universal Robots UR5

Universal Robots UR5

Nosivost	5 kg
Doseg	850 mm
Maksimalna brzina	±180°/sek
Ponovljivost	±0.1 mm
Komunikacija	TCP/IP 100Mbit, Modbus TCP, Profinet, EthernetIP
Radna temperatura	Od 0 do 50℃

Tablica 2.2 Specifikacije kolaborativnog robota

2.3. Vakuumska hvataljka

Kao radni alat robota, korištena je vakuumska hvataljka konstruirana u laboratoriju CRTA-e.

Slika 2.4 Vakuumska hvataljka

2.4. Programska knjižnica OpenCV

Odabir OpenCV-a kao Python biblioteke za okosnicu strojnog vida je bio ekonomičan izbor, prvenstveno zbog sveobuhvatnog spektra značajki i ugleda kao robusne, open-source biblioteke za obradu slika i računalni vid. OpenCV (Open Source Computer Vision Library) poznata je po svojoj opsežnoj zbirci algoritama i funkcija koje se bave raznim aspektima računalnog vida i obrade slika. To uključuje temeljne operacije kao što su hvatanje i manipulacija slikama, kao i naprednije funkcije kao što su otkrivanje objekata, ekstrakcija značajki i prepoznavanje uzoraka. Svestranost i širina OpenCV-a čine ga idealnim izborom za razvoj sofisticiranih aplikacija strojnog vida koje zahtijevaju obradu i analizu u stvarnom vremenu. OpenCV je prvotno razvio Intel ranih 2000-tih u nadi da će stvoriti optimiziranu više-platformsku knjižnicu otvorenog koda koja će ubrzati razvoj vizijskih aplikacija [2].

Štoviše, kompatibilnost OpenCV-a s Pythonom, jednim od najpopularnijih programskih jezika za znanstveno računarstvo i strojno učenje, doprinosi njegovoj privlačnosti. Pythonova jednostavnost i čitljivost, u kombinaciji s njegovim opsežnim ekosustavom biblioteka, čine ga izvrsnim izborom za razvoj projekata strojnog vida. Integracija OpenCV-a s Pythonom olakšava brzu izradu i iteraciju, omogućujući učinkovit razvoj i testiranje složenih vizualnih algoritama. Nadalje, aktivna zajednica i bogatstvo obrazovnih resursa koji okružuju OpenCV pružaju neprocjenjivu podršku za rješavanje problema i proširenje mogućnost. Ovaj ekosustav ne samo da ubrzava proces razvoja već i osigurava korištenje najnovijih dostignuća u tehnologiji računalnog vida.

Slika 2.5 OpenCV logo

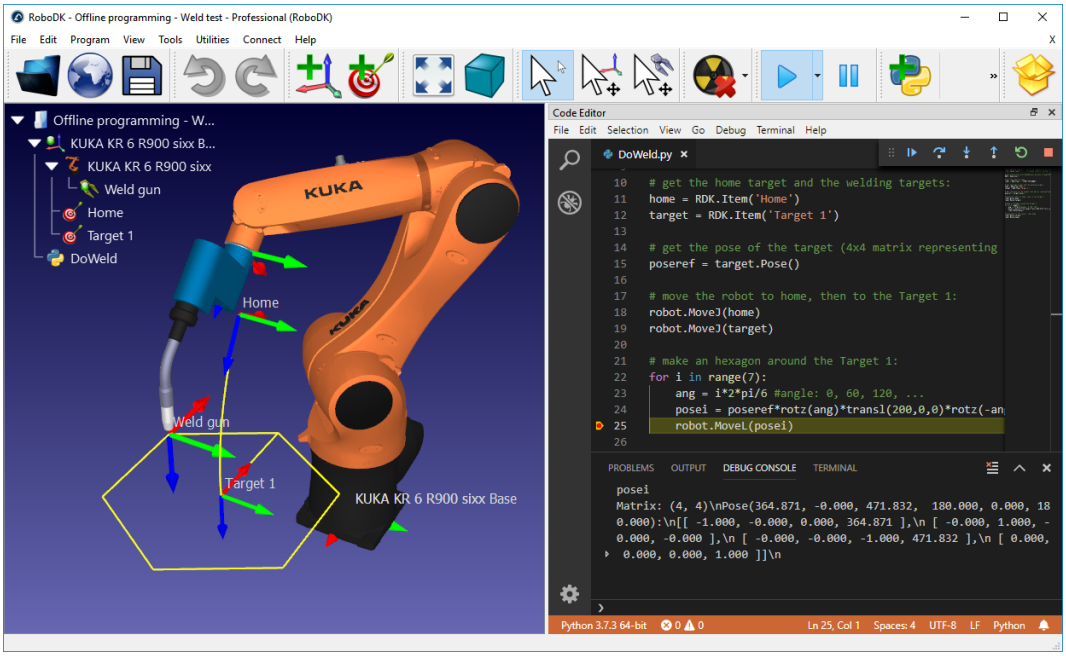
2.5. Simulacijski softver

Korištenje RoboDK-a za simulaciju i programiranje robota podudara se s potrebom za sveobuhvatnom, korisniku prilagođenom platformom za upravljanje i simulaciju robota. RoboDK je poznat po svojoj širokoj kompatibilnosti s velikim rasponom robota, uključujući kolaborativne robote, industrijske manipulatore i još specijaliziranije robotske sustave. Ova fleksibilnost čini RoboDK idealnim alatom za projekte koji mogu uključivati različite vrste robota, omogućujući besprijekornu integraciju i kontrolu unutar ujedinjenog okruženja. Njegova opsežna biblioteka robotskih modela i mogućnost jednostavnog uvoza novih osigurava da se projekt može prilagoditi različitim zahtjevima hardvera bez značajnog preoblikovanja.

Jedna od ključnih prednosti RoboDK-a je njegovo intuitivno sučelje i jednostavnost s kojom korisnici mogu simulirati i programirati robotske operacije. Platforma podržava vizualno programiranje, što može značajno smanjiti složenost programiranja robota, posebno za korisnike bez velikog iskustva u robotici. Takav pristup je posebno koristan za projekte koji zahtijevaju brze cikluse razvoja i izradu prototipova, jer omogućuje brza prilagođavanja i optimizaciju robotskih putanja i operacija. Mogućnosti simulacije RoboDK-a nude sposobnost testiranja i validacije robotskih zadataka u virtualnom okruženju prije njihove implementacije u stvarnom svijetu, minimizirajući rizik od pogrešaka i osiguravajući sigurnost i učinkovitost robotskih operacija. Ova sposobnost ključna je za projekte koji uključuju složene ili precizne zadatke, gdje bi testiranje u stvarnom svijetu moglo biti dugotrajno, skupo ili potencijalno opasno.

Druga prednost RoboDK-a je njegov Python API koji pruža sučelje za skriptiranje robotskih simulacija što dozvoljava jednostavnu automatizaciju repetitivnih radnji i olakšava implementaciju algoritamske logike. Dodatno, korištenjem skripti se mogu lako mijenjati parametri sustava poput robotskih putanja, brzina te se može upravljati digitalnim ulazima i izlazima. Sučelje nije ograničeno samo na robote pa se mogu lako stvarati i pomicati virtualni predmeti unutar simulacije što je izrazito zgodno za ovaj zadatak.

Kako je sučelje razvijeno u Pythonu, može se lako integrirati za drugim programskim knjižnicama kao prije već spomenutim OpenCV-om pa se tako smanjuje složenost programa i uklanja potreba za bilo kakvim bazama podataka ili nekim načinom komunikacije između različitih programa.



Slika 2.6 Prikaz grafičkog sučelja RoboDK

3. KALIBRACIJA VIZIJSKOG SUSTAVA

U svijetu robotike, precizna koordinacija između robotskih manipulatora i sustava za strojni vid od presudne je važnosti za aplikacije koje uključuju prepoznavanje, lociranje i interakciju s objektima u radnom prostoru. Proces poznat kao 'hand-to-eye' (s eng. ruka-oko) kalibracija smatra se ključnim korakom za postizanje ove sinkronizacije, osobito u scenarijima gdje robot koristi statičnu kameru za prepoznavanje objekata i manipuliranje. 'Hand-to-eye' kalibracija sustavni je proces određivanja prostorne povezanosti između koordinatnog sustava kamere ("oko") i koordinatnog sustava robotskog krajnjeg efektora ("ruka"). Prostorni odnos je ključan za transformaciju dvodimenzionalnih slikovnih podataka koje snima kamera u trodimenzionalni koordinatni sustav robota, omogućujući robotu da precizno locira i djeluje s objektima u radnom prostoru. Važnost 'hand-to-eye' kalibracije u takvim okolnostima je naročita zbog izravnog utjecaja na sposobnost sustava da obavlja zadatke s potrebnom točnošću i pouzdanošću.
U ovom slučaju, potrebno je transformirati koordinate kamere gdje su pronađeni traženi predmeti u koordinatni sustav robota, točnije bazu robota. Uz postojanje 'hand-to-eye' kalibracije, postoji 'hand-in-eye' kalibracija, za slučaj kad se kamera nalazi uz alat robota.

3.1. Homogene transformacije

Postupak kalibracije kamere svodi se na postupak rješavanja sustava jednadžbi s jednom nepoznatom matricom pri čemu su već poznati odnosi između baze i alata robota te odnos predmeta i kamere. Postoji više načina na koje se te jednadžbe mogu riješiti i uglavnom se dijele na dvije skupine: izračun rotacije, tek zatim translacije i simultano pronalaženje rotacije i transformacije. Za ovaj postupak je potrebno zapisati poze robota koje zapravo predstavljaju odnos alata i baze robota, istovremeno prikupljajući slike kalibracijske ploče preko kamere. Kalibracijska ploča je prihvaćena hvataljkom na robota i uperena u što više poza prema kameri, nastojeći iskoristiti što više pomaka zglobova robota. Postupak pronalaska homogene transformacije zahtijeva minimalno dvije poze, ali preporučuje se korištenje što većeg broja poza radi povećanja preciznosti i točnosti kalibracije [3].

Problem se svodi na sljedeće jednadžbe:

$$AX = XB$$

Što predstavlja za hand-to-eye konfiguraciju:

$${}^{b}T_{g}^{(i)}\ {}^{g}T_{c}\ {}^{c}T_{t}^{(i)} = {}^{b}T_{g}^{(j)}\ {}^{g}T_{c}\ {}^{c}T_{t}^{(j)}$$

$$\left({}^{b}T_{g}^{(j)}\right)^{-1}\ {}^{b}T_{g}^{(i)}\ {}^{g}T_{c} = {}^{g}T_{c}\ {}^{c}T_{t}^{(j)}\ \left({}^{c}T_{t}^{(i)}\right)^{-1}$$

$$A_i X = X B_i$$

${}^{b}T_{g}$ predstavlja transformaciju od baze do alata (hvataljke) koju se dobije inverznom kinematikom od zapisanih poza.
${}^{c}T_{t}$ predstavlja transformaciju od kamere do predmete koju smo dobili traženjem kalibracijske ploče na slici.
${}^{g}T_{c}$ predstavlja transformaciju od alata, tj. hvataljke do kamere koju računamo.

Za kalibraciju kamere u ovom zadatku korištena je *charucoboard,* vrsta kalibracijske ploče koje kombinira klasični šahovski uzorak s ArUco markerima. ArUco markeri su razvijeni na Sveučilištu u Cordobi s ciljem da se ubrza pronalazak pouzdanih markera na sceni s ciljem procjene poze kamere markera koji se koriste u aplikacijama računalnog vida kako bi se olakšalo detektiranje i procjena položaja objekata na slikama i video snimkama. ArUco marker je crno-bijeli kvadratni uzorak koji se lako može prepoznati i dekodirati s pomoću algoritama računalnog vida. Ti markeri se često koriste u aplikacijama proširene stvarnosti, robotici i za potrebe kalibracije kamere [4].
Dizajn ArUco markera sastoji se od crnog kvadratnog okvira s binarnom matricom u središtu koja kodira ID markera. Kontrast između crnih i bijelih područja unutar kvadrata omogućuje robusno detektiranje čak i u različitim svjetlosnim uvjetima ili pod različitim kutovima i udaljenostima od kamere. Veličina binarne matrice (npr. 4x4, 5x5, 6x6 itd.) određuje broj jedinstvenih markera koji se mogu generirati, pri čemu veće matrice omogućuju veći broj različitih ID-ova.
ArUco markeri funkcioniraju tako što se postavljaju u okolinu ili pričvršćuju na objekte koje treba detektirati i pratiti s pomoću sustava kamere. Kada kamera snimi sliku koja uključuje jedan ili više ArUco markera, algoritmi računalnog vida obrađuju sliku kako bi detektirali kvadratne oblike markera, a zatim dekodirali binarne uzorke unutar njih kako bi identificirali

jedinstveni ID svakog markera. Jednom kada se marker detektira, algoritam može također odrediti orijentaciju i položaj markera u odnosu na kameru analizirajući distorziju kvadratnog oblika na snimljenoj slici. Taj proces, poznat kao procjena položaja, omogućava sustavu da razumije 3D položaj i orijentaciju markera u prostoru.

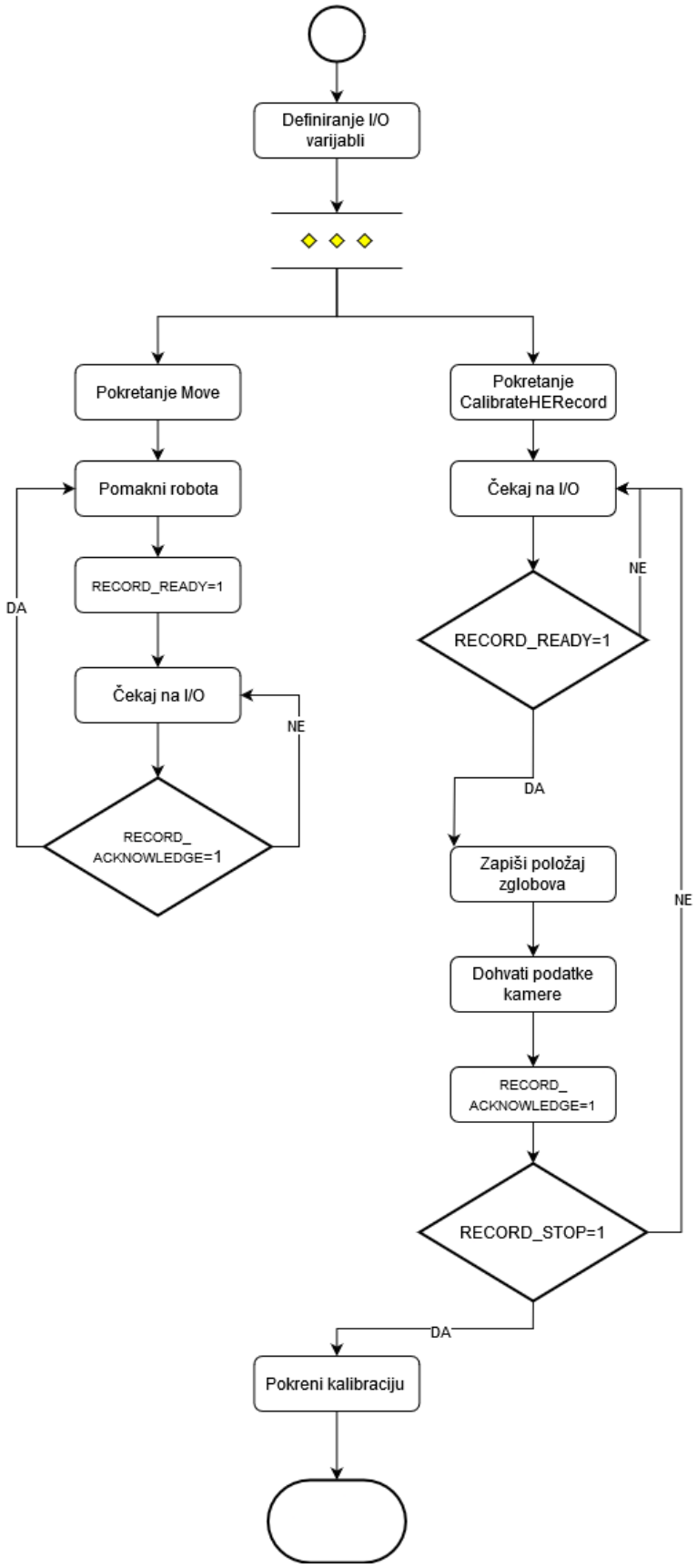
Charucoboard nastoji nadomjestiti nedostatke ArUco markera i šahovskih ploča. Glavna mana kod ArUco markera je niska preciznost kod pronalaska kutova markera iako su brzi, dok je glavna mana šahovskih ploča to što moraju biti u potpunosti vidljivi i ne smiju biti na ni jedan način prekriveni.

Ova kalibracija je provedena koristeći RoboDK i OpenCV za kontrolu robota i dohvaćanje podataka iz kamere. Za početak, potrebno je postaviti kalibracijsku ploču u hvataljku robota imajući na umu da se postavi što preciznije u samo središte hvataljke. Za te potrebe, isprintan je držač na koji se naljepljuje isprintani kalibracijski uzorak. Kalibracijski se uzorak može generirati proizvoljno koristeći OpenCV s naredbom

```
python gen_pattern.py -o charuco_board.svg --rows 7 --columns 5 -T
charuco_board --square_size 30 --marker_size 15 -f DICT_5X5_100.json.gz
```

pri čemu se definiraju veličine šahovskih kvadrata i markera u milimetrima, broj redaka i stupaca te veličina binarne matrice kod markera. Ti isti parametri se moraju i definirati u kodu za kalibraciju kamere zbog provedbe dobre usporedbe. Generirana SVG datoteka se može zatim isprintati na A4 papiru i koristit kao kalibracijska ploča.

Nakon toga, potrebno je definirati poze robota pri kojima će se vršiti kalibracija, što je u slučaju kolaborativnog robota, vrlo jednostavan zadatak. RoboDK nudi mogućnosti direktnog spajanja na robota preko lokalne mreže unošenjem IP adrese robota zbog čega je moguće ručno pomicati robota u željene poze i stvarati „targete“ u RoboDK s pomoću očitane poze robota.


Definiranje I/O varijabli
Pokretanje Move
Pomakni robota
RECORD_READY=1
Čekaj na I/O
RECORD_ ACKNOWLEDGE=1
DA
NE
Pokretanje CalibrateHERecord
Čekaj na I/O
RECORD_READY=1
NE
DA
Zapiši položaj zglobova
Dohvati podatke kamere
RECORD_ ACKNOWLEDGE=1
RECORD_STOP=1
NE
DA
Pokreni kalibraciju

3.2. Pojednostavljena kalibracija kamere

Osim korištenja kalibracijskih ploča i algoritama za pronalaženje uzoraka kao rješavanje sustava jednadžbi, postoji pojednostavljena kalibracija kamere koja je posebno prigodna za slučajeve kad je kamera statično montirana. U tom se slučaju definira poznato ishodište koordinatnog sustava koje je zajedničko kameri i svijetu i tri točke u bazi robota s koordinatama x, y i z. Pomoću te tri točke definiraju sva vektora koja su ravnini i treći koji skalarni umnožak ta dva vektora i predstavlja Z-os [5].

Tri točke u koordinatnom sustavu robota možemo zapisati kao:

$$P1 = (x1, y1, z1)$$

$$P2 = (x2, y2, z2)$$

$$P3 = (x3, y3, z3)$$

Tada koristeći te navedene točke računamo dva vektora:

$$V1 = P2 - P1$$

$$V2 = P3 - P1$$

Te se s njima definiraju jedinični vektori u, v i w:

$$u = \frac{V1}{||V1||}$$

$$v' = \frac{(V2)}{||V2||}$$

$$w = \frac{u \times v'}{||u \times v'||}$$

$$v = w \times u$$

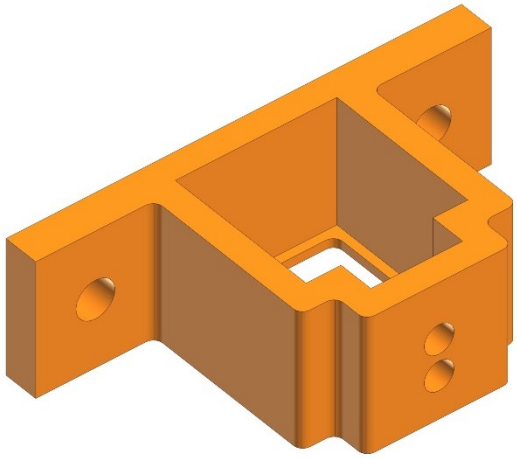
S pomoću dobivenih vektora možemo zapisati 4x4 matricu transformacije:

$${}^{W}_{R}T = \begin{bmatrix} u\_x & v\_x & w\_x & P1\_x \\ u\_y & v\_y & w\_y & P1\_y \\ u\_z & v\_z & w\_y & P1\_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

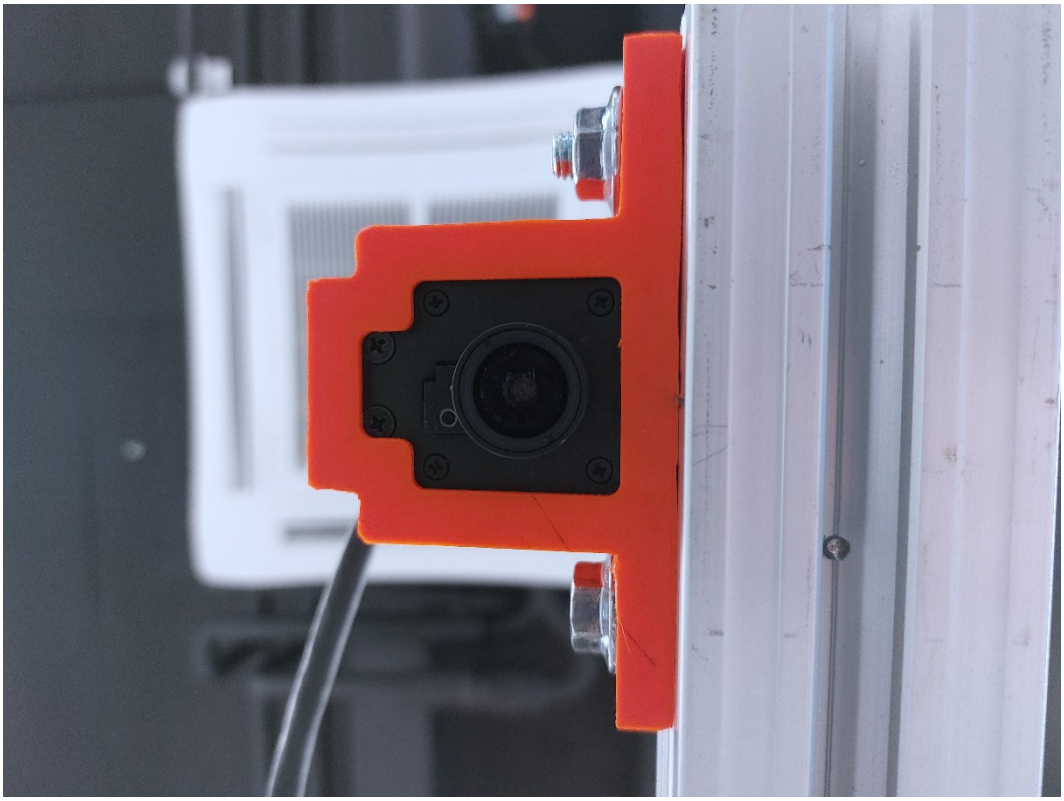
3.3. Montaža kamere

Kameru je potrebno montirati iznad radnog prostora robota tako da ne ometa robota u rukovanju s predmeta ali da pritom ima cijeli radni prostor u vidnom polju. Za montažu kamere su korišteni standardni aluminijski profili koji idu od ruba stola u visinu i prema središtu radnog stola. Kako kamera nema vlastito kućište za montažu, potrebno je izraditi nosač koji će omogućiti držanje kamere i jednostavnu montažu na aluminijske profile s T-utor maticama.

Nosač je konstruiran pomoću Solidworksa uz tehničk dokumentaciju proizvođača i 3D isprintan na Prusa MK3S printeru s PETG filamentom.



Slika 3.1 Model nosača kamere



Slika 3.2 Kamera postavljena na nosač

4. LOKALIZACIJA PREDMETA

Za detekciju predmeta i pronalaženje istih u ovom radu je korišten feature matching.

Značajke (eng. feature) predstavljaju sve karakteristične točke, rubove i kutove koji su mogu pronaći na slici zbog promjene intenziteta piksela. Glavni problem nastaje pri opisivanju tih karakterističnih točka jer se javlja potreba da opisi budu geometrijski invarijantni na translaciju, razmjer, orijentaciju kao i fotometrijski na svjetlinu i ekspoziciju [6]. Kroz vrijeme su razvijeni deseci feature detektora i deskriptora, ali u današnje vrijeme koristi se mal broj istih, tj. ostali su relevantni samo oni koji su dovoljno robusni a istovremeno i brzi za svakodnevnu primjenu. Najpopularnije metode su implementirane u OpenCV-u poput SURF-a („***S**peeded **U**p **R**obust **F**eatures*“), SIFT-a („***S**cale-**I**nvariant **F**eature **T**ransform*“), FAST-a („**F**eatures from **A**ccelerated **S**egment **B**est“), BRIEF-a („***B**inary **R**obust **I**ndependent **E**lementary **F**eatures*“) i ORB-a („***O**riented FAST and **R**otated **B**RIEF*“), no u radu su korišteni samo SIFT i ORB zbog robusnosti i brzine.

Početak razvoja feature matching algoritama seže od potrebe za „šivanjem“ (eng. stitching) više slika kako bi se stvorila panoramska fotografija, no s vremenom su se pronašle i razne druge primjene kao što su stereo vizija za procjenu dubine, prepoznavanje predmeta, praćenje kretanja, proširena stvarnost i SLAM.

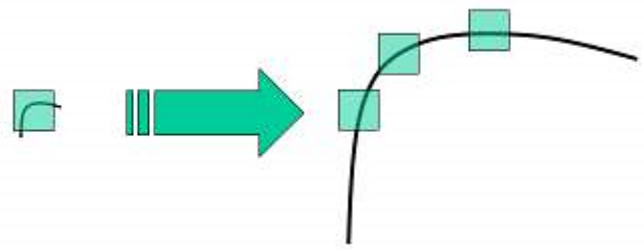
Proces feature matching se može podijeliti na četiri bitna koraka:

- Pronalaženje značajki (eng. „feature detection“)
- Opisivanje značajki (eng. „feature description“)
- Podudaranje značajki (eng. „feature matching“)
- Filtriranje dobrih podudaranja

4.1. SIFT algoritam

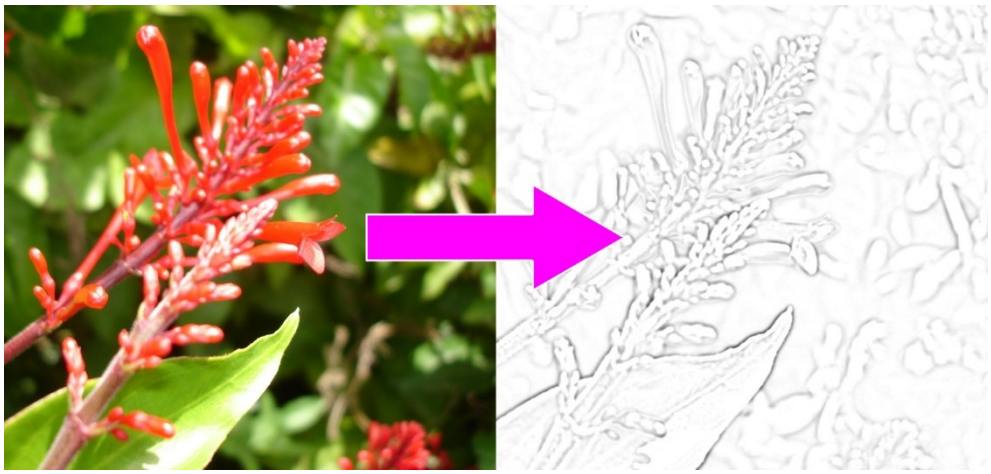
SIFT algoritam razvio je prof. David Lowe na Sveučilištu u Britanskoj Kolumbiji, koji je ujedno detektor i deskriptor značajki, kao zamjenu za Harris detektore, postojeći algoritam za pronalaženje značajki [7].Glavni nedostatak Harris detektora je bila invarijantnost na razmjer (eng. Scale). Primjerice, crta na slici XX na maloj skali predstavlja kut, koja prema Harrisu je promjena vrijednosti piksela u svakom smjeru, no pak kad se slika poveća, linija u istom prozoru ne predstavlja više kut, nego liniju. Harris-Laplace detektor postaje invarijantan na razmjer u odnosu na Harrisa koristeći Laplace Gaussovog filtera, točnije pronalazi rubove u

različitim skalama. Prostor skala predstavlja skup slika na koje je primijenjen Gaussov filter i čija je rezolucija smanjena u odnosnu na prethodnu.



Slika 4.1 Harris detektor

SIFT umjesto korištenja Laplacea Gaussovog filtera, koristi razliku Gaussa u istoj skali. Zamjenom te matematičke operacije, algoritam postoje puno računalno efikasniji i brži. U praksi, razlika Gaussovih filtera nam služi za pronalaženje rubova na slici.

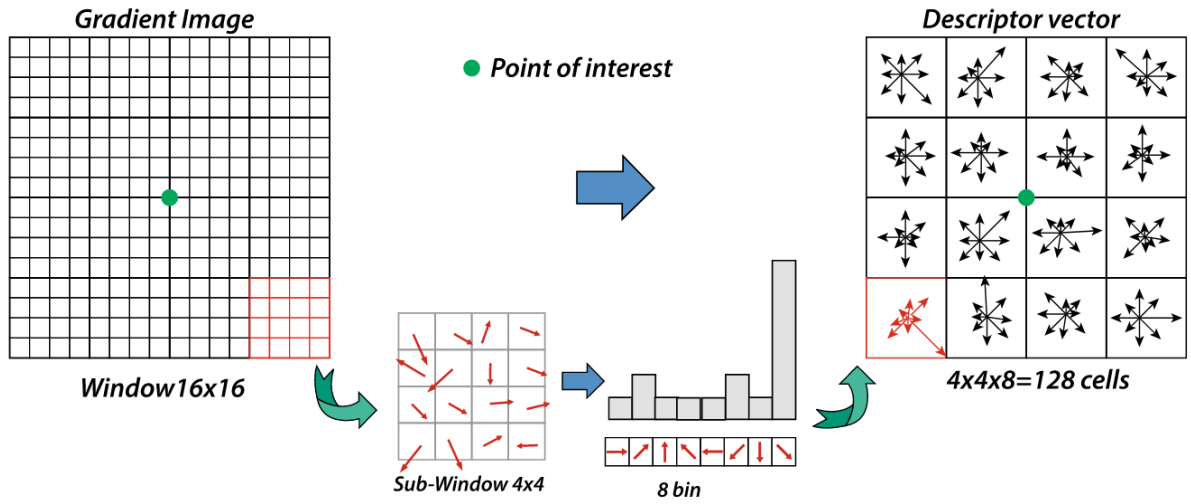


Slika 4.2 Rezultat razlike Gaussovih filtera

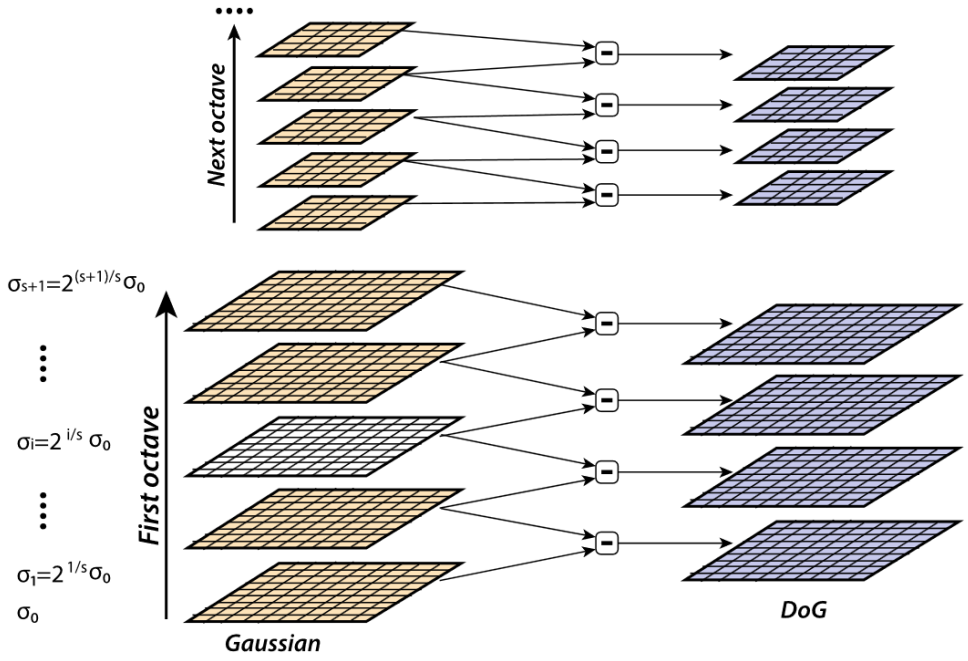
Provođenje Gaussovog filtriranja pri različitim oktavama (rezolucijama) je zgodan način za opisivanje detalja na slici kao i sveukupne slike. Skaliranjem slike na niže rezolucije stvaraju se ključne točke i mrlje (eng. Blob) koje prikazuju umanjenu verziju traženog predmeta s kojima se može opisati predmet kao da se nalazi na većoj udaljenosti na slici. Od preostalih kontura se traže lokalni maksimumi uspoređujući potencijalnu ključnu točku tj. piksel s okolnih 8 osam piksela i s 9 piksela iz prethodne i iduće skale. Nakon što se pronađu sve potencijalne ključne točke, filtriraju se koristeći 3D kvadratnu funkciju i provjeravajući je li izabrana točka lokalni ekstrem te ako je, zadržava se. Također se uklanjaju sve točke koje leže na rubovima koristeći 2x2 Hessian matricu, tako da ako su točke ispod nekog praga, uklanjaju se kako bi

detektirane točke bile robusnije na promjene. Potom se pomoću susjednih točaka računa dominantna orijentacija računajući magnitudu i orijentaciju gradijenta uzimajući u obzir težinski faktor sigma. Tako se stvara histogram orijentacija u 36 odjeljaka svakih 10°. Nakon popunjavanja svih odjeljaka, traži se najsnažniji gradijent i odbacuju svi koji su ispod 80% od najsnažnijeg.

Sljedeći korak je stvaranje deskriptora koji je invarijantan na razmjer i orijentaciju zbog čega su dosta važni prethodno izračunate orijentacije. Za svaku ključnu točku stvara se mreža koja je orijentirana prema dominantnom gradijentu, čineći ju invarijantnom na rotaciju. Oko točke se stvaraju 4x4 podmreže i pripadajući histogram s 8 različitih orijentacija. Svaki gradijent u regiji doprinosi intenzitetu orijentacija u histogramu čime se stvara kompaktna predodžba ključne točke. Plod svih ovih operacija je 128 dimenzijski vektor za svaku ključnu točku koji je prethodno normaliziran. Takav vektor nam dopušta da ga uspoređujemo s drugim vektorima u daljnjim koracima kako bi pronašli sličnosti između scena.



Slika 4.3 Vizualni prikaz SIFT deskriptora



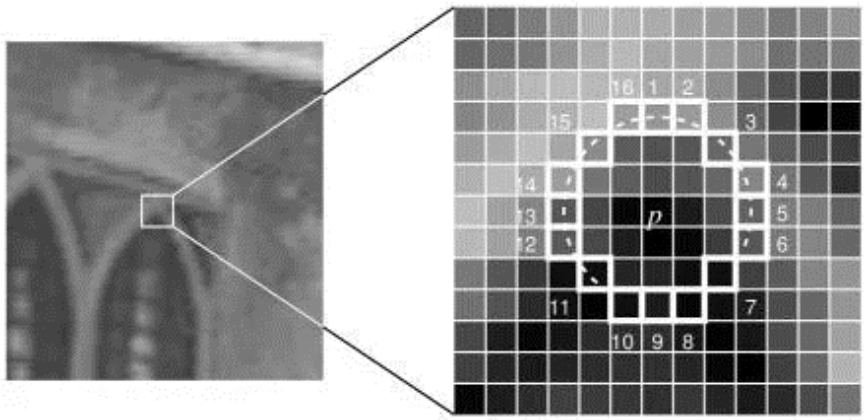
Slika 4.4 Vizualizacija oktava i skala kod SIFT detektora

4.2. ORB algoritam

ORB je detektor i deskriptor značajki kojeg su razvili vodeći razvojni programeri OpenCV-a 2011. godine u potrazi za alternativnih rješenjem nad SIFT-om i SURF-om koje bi bilo prikladno za „real-time“ izvedbe koje se ne oslanjaju na GPU. Iako su postojale izvedbe poznatih algoritama implementiranih da koriste GPU, mobilni uređaji, pogotovo u to vrijeme, nisu imali prikladne performanse za ikakvu hardversku akceleraciju.

ORB algoritam ponajviše predstavlja poboljšanja i kombinaciju postojećih rješenja, točnije FAST detektora i BRIEF deskriptora. FAST algoritam za prepoznavanje ključnih točaka je bio iznimno brz način za pronalaženje kutova koristeći vrlo jednostavan pristup. Za svaki piksel na slici stvara se kružno područje oko izabranog piksela, u originalnoj izvedbi 16 piksela, te se tražio N broj uzastopnih piksela istog ili sličnog intenziteta [8]. Kako bi se izvođenje dodatno ubrzalo, nisu uspoređivani svi pikseli, nego samo svaki drugi neparni (1,5,9,13). Ako najmanje 3 piksela od ta 4 nisu iznad niti ispod intenziteta srednjeg piksela, može se s velikom sigurnošću reći da to područje nije kut. Iako je taj pristup podosta brz te nudi dobar omjer performansi i točnosti, FAST detektor nije invarijantan na orijentaciju ni na razmjer.

ORB koristi FAST za pronalazak potencijalnih ključnih točaka koje potom filtrira s Harris detektorom kako bi se odbacili rubovi. Invarijatnost na razmjer se eliminira korištenjem



Slika 4.5 Vizualizacija FAST detektora

piramide skala slike, slično kao kod SIFT-a ali bez primjene Gaussovog filtera, na koju se primjenjuju FAST detektor i Harris detektor po svakoj skali. Isto tako, koristeći momente i centroid intenziteta, umjesto gradijenata kao SIFT, računa smjer ključne točke. Za kružno područje oko ključne točke računaju se momenti i centroid te je vektor koji spaja središte kružnice i centroida dominantna orijentacija [9].

BRIEF deskriptor je binarni deskriptor koji opisuje ključne točke provjeravajući jesu li susjedne točke većeg ili manjeg intenziteta od točke s kojem se uspoređuje, imajući za izlaz 1 ili 0. Rotirajući BRIEF uzima u obzir orijentaciju ključne točke iz detektora te svaki par točaka koji se uspoređuje rotira za tu vrijednost, čineći ga invarijantnim na rotaciju.

4.3. Implementacija u OpenCV-u

OpenCV dolazi s implementacijama već navedenih algoritama kao i s mnoštvom drugih deskriptora, funkcijama za manipulaciju slike i drugim zgodnim alatima za vizualizaciju. Prednost korištenja OpenCV-a je ne samo programska podrška i dokumentacija, već i sama zajednica korisnika čiji se primjeri, problemi i rješenja mogu jednostavno pronaći na internetu. OpenCV nudi sve svoje funkcije u C++ programskom jeziku i u Pythonu, iako se Python ne koristi za izvođenje algoritamski intenzivnih funkcija, već je samo „omot“ koji poziva

originalne funkcije napisane u C-u i C++-u čineći ga tako jednako brzim kao originalnu implementaciju no prednostima pojednostavljenog programiranja u Pythonu.

Definirana je funkcija *describe_kp(img)* koja ima samo za ulazni parametar ima sliku za koju je potrebno pronaći ključne točke i opisati ih.

```
def describe_kp(img):
    if FEATURES == FeatureDesc.ORB:
        detector = cv.ORB_create(10000)

    elif FEATURES == FeatureDesc.SIFT:
        detector = cv.SIFT_create()

    keypoints, descriptors = detector.detectAndCompute(img, None)

    return keypoints, descriptors
```

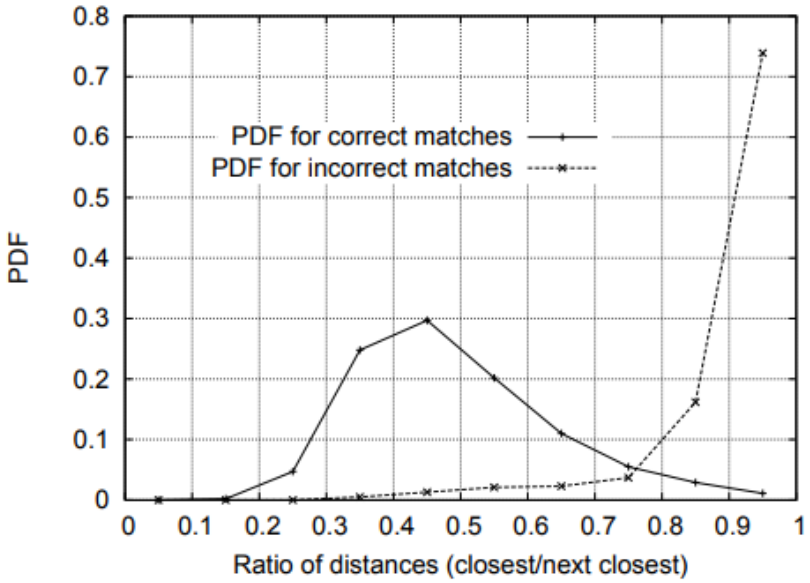
Ovisno o detektoru i deskriptoru koji se koristi, stvaraju se objekti koji povezuju odabrani detektori te se s njima dalje računaju ključne točke i opisuju iste. *ORB_create* i *SIFT_create* mogu primati također razne parametre koji imaju utjecaj na rad samih detektora poput broja ključnih točaka koje treba pronaći, praga za rubove i kontraste ili broj levela piramide kod skaliranja. Uz odabrani detektor, poziva se *detectAndCompute* koji za ulaznu varijablu ima sliku te su izlazne varijable lokacije ključnih točaka i opis istih. Ova funkcija *describe_kp* poziva se odvojeno za svaki predmet koji se traži i za sliku scene nakon čega se uspoređuju pronađene točke.

Definira se funkcija *find_object()* koja pokušava pronaći tražene predmete na sceni uspoređujući opisane točke. Jedina ulazna varijabla je slika predmeta koji se traži dok su izlazne varijable X, Y koordinate predmeta u koordinatnom sustavu robota i rotacija predmeta.

```
def find_object(item):

    img_object = cv.imread(item, cv.IMREAD_GRAYSCALE)

    if img_object is None or img_scene is None:
        print('Could not open or find the images!')
        exit(0)

    img_object = clahe.apply(img_object)

    if FEATURES == FeatureDesc.ORB:
        index_params = dict(algorithm=6,
                            table_number=6,
                            key_size=12,
                            multi_probe_level=2)
        search_params = {}
        matcher = cv.FlannBasedMatcher(index_params, search_params)


    elif FEATURES == FeatureDesc.SIFT:
        matcher =
cv.DescriptorMatcher_create(cv.DescriptorMatcher_FLANNBASED)

    keypoints_obj, descriptors_obj = describe_kp(img_object)

    knn_matches = matcher.knnMatch(descriptors_obj, descriptors_scene, 2)
```

Prvo se učitava slika predmeta kao crno-bijela slika na koju se potom primjenjuje CLAHE („*Contrast Limited Adaptive Histogram Equalization*“) čime smanjuje i normalizira kontrast na slici čime sve značajke postaju jednako izražene. Ovisno o korištenom detektoru, pozivaju se FLANN „matcheri“, skup algoritama za brzo pronalaženje susjednih točaka. Odabrana je KNN metoda (K-Nearest Neighbors) za k=2 što predstavlja da svaku pronađenu točku se pronalaze dvije najbliže susjedne točke.

Nakon pronalaska parova podudarajućih točaka, točke se filtriraju koristeći Loweov test omjera udaljenosti. Prema njegovom radu, točke za koje nismo sigurno jesu li odgovarajuća podudaranja, mogu se filtrirati ako najbliža točka nije ispod praga udaljenosti od druge točke. Prema mjerenjima na podacima od 40000 točaka, prag od oko 0.75 pokazuje najbolje rezultate.



Slika 4.6 Omjer udaljenosti i funkcije gustoće vjerojatnosti

```
ratio_thresh = 0.74
    good_matches = []
    for m,n in knn_matches:
        if m.distance < ratio_thresh * n.distance:
            good_matches.append(m)

    obj = np.float32([keypoints_obj[m.queryIdx].pt for m in
good_matches]).reshape(-1,1,2)
    scene = np.float32([keypoints_scene[m.trainIdx].pt for m in
good_matches]).reshape(-1,1,2)

    if len(good_matches) < 10:
        return None, None

    H, mask =  cv.findHomography(obj, scene, cv.RANSAC,4)

    theta = -atan2(H[1,0], H[0,0])

    h,w = img_object.shape
    pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)
    dst = cv.perspectiveTransform(pts,H)

    cntr, rotAngle = getOrientation(dst, img_scene)
```

Također, ako je broj dobrih podudaranja manji od 10, funkcija vraća prazne varijable, ukazujući na nepostojanje traženog predmeta.

Koristeći točke dobrih podudaranja, traži se homografija između traženog predmeta i scene kako bi mogli pronaći za koliko stupnjeva je zakrenut predmet. Homografija je transformacija koja mapira točke iz jedne plohe u drugu te uzima u obzir rotaciju, translaciju, skalu i perspektivnu distorziju. Kako bi se mogla izračunati matrica homografije, potrebno je imati minimalno četiri odgovarajuće točka nakon čega se može postaviti sustav linearnih jednadžbi gdje se traži matrica H [10].

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = H \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Pomoću izračunate matrice, može se izračunati rotacija predmeta oko Z-osi.

U nastavku se definiraju točke koje tvore pravokutni oblik čineći granični okvir traženog predmeta i koje se transformiraju pomoću izračunate matrice H kako bi saznali gdje se predmet nalazi na slici pomoću funkcije *perspectiveTransform.* Pozivajući funkciju *getCenter,* pronalazi se geometrijsko središte transformiranog graničnog okvira što predstavlja središte predmeta na sceni.

Uz pomoć prijašnje kalibracije i matrice transformacije koja se dobije, središta predmeta se transformiraju iz koordinatnog sustava slike u koordinatni sustav robota. Ovisno o tome želi li se koristiti kalibracija oko-ruka ili pojednostavljena kalibracija, pronađene točke se transformiraju sukladno. Potrebno je pripaziti od kojih točaka, točnije piksela, se traže pomaci. Kod oko-ruka kalibracije, transformacija zapravo predstavlja optičko središte kamere, tj. predmet koji se nalazi u sredini kadra bi imao jednake X i Y koordinate kao i kamera. Dakle, u idućim koracima je potrebno pronaći udaljenost središta predmeta od ishodišta koordinatnog sustava kamere i podijeliti ga s izmjerenim omjerom piksela i milimetara koja predstavlja koliko piksela čini jedan milimetar. Uvrštavajući te vrijednosti u matrice, možemo dobiti transformirane vrijednosti u koordinatnom sustavu baze robota. Dobivene vrijednosti su izlazne varijable pozicije i rotacije predmeta.

```
    if CALIB_TYPE == CalibType.SIMPLE:
        p0 = [1204 ,1846]
        p1 = cntr

        p = [p1[0] - p0[0], p1[1] - p0[1]]
        dx= p[0]
        dy = p[1]

        uv = np.array([[1,0,0,dx/px2mm],
                       [0,1,0,dy/px2mm],
                       [0,0,1,0],
                       [0,0,0,1]], dtype=np.float32)

        Rt = np.array([[0.717, 0.697, 0, -470],
                       [0.697, -0.717, 0, -724],
                       [0, 0, 1, 0],
                       [0, 0, 0, 1]], dtype=np.float32)

        rotAngle = -atan2(Rt[1,0], Rt[0,0])
        item_loc = np.matmul(Rt, uv)
        xyz = item_loc

    elif CALIB_TYPE == CalibType.HANDEYE:
        dx = cam_mtx5[0,2] - cntr[0]
        dy = cam_mtx5[1,2] - cntr[1]

        Rt = np.array([[0.7173754810141859, 0.6966179234465155,
0.009792240434646583, -581.95],
                       [0.6965090352371797, -0.7168042046648919, -
0.032663374101656324, -325.90951685],
                       [-0.01573477272280828, 0.030252287645668206, -
0.999418438903128, 672.4878897],
                       [0,0,0,1]], dtype=np.float32)

        Xc = ((dx*(Rt[2,3])/cam_mtx5[0,0]))
        Yc = ((dy*(Rt[2,3])/cam_mtx5[1,1]))

        loc = np.array([[1,0,0,-Xc],
                       [0,1,0,-Yc],
                       [0,0,1,0],
                       [0,0,0,1]], dtype=np.float32)

        rotAngle = -atan2(Rt[1,0], Rt[0,0])
        item_loc = np.matmul(Rt, loc)
        xyz = item_loc

    return xyz, theta + rotAngle
```

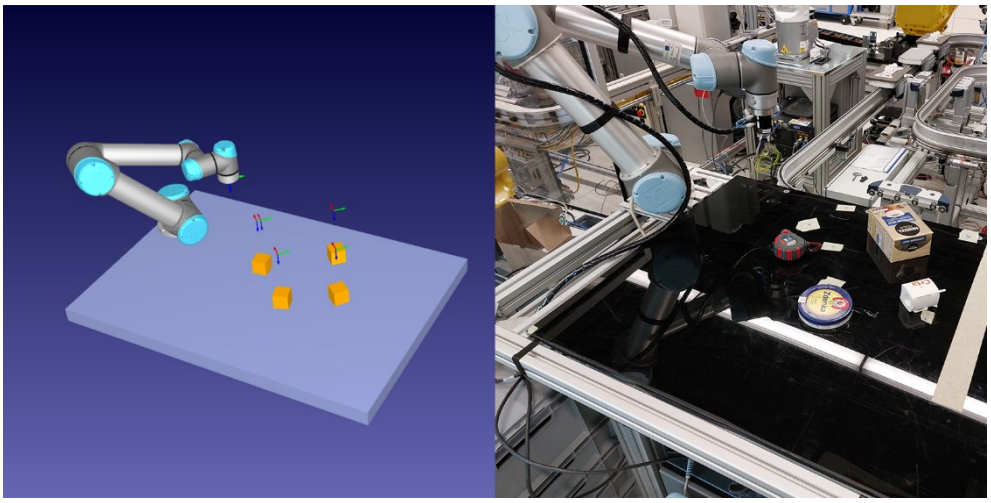
5. SIMULACIJA U ROBODK

Zbog svoje modularnosti i programske podrške u Pythonu, predmete i njihove koordinate možemo jednostavno i efektivno vizualizirati u virtualnom okruženju kako bi dobili presliku scene i automatski planirali hvatanje traženih predmeta.

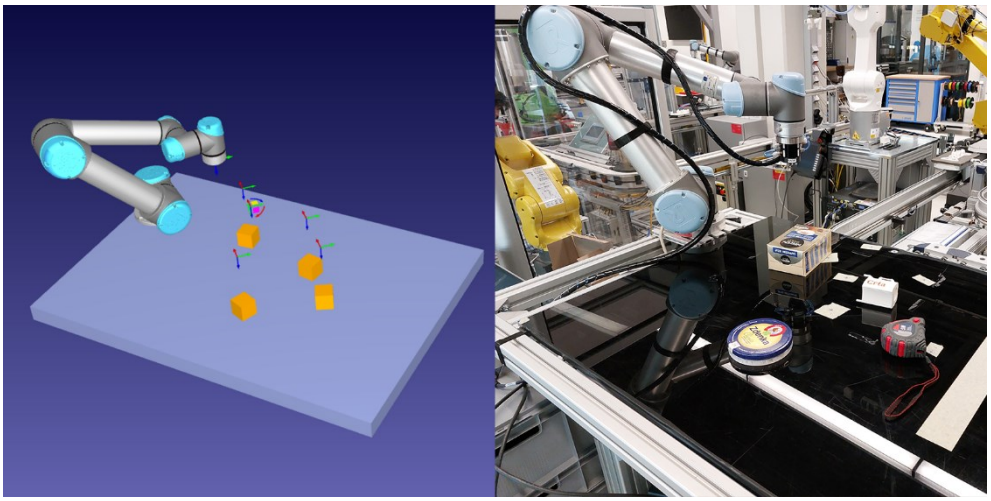
```
    home = RDK.Item("Home", 6)
    targets = RDK.ItemList(filter=robolink.ITEM_TYPE_TARGET)
    for target in targets:
        if target.equals(home):
            continue
        RDK.Delete(target)

    for item in items:
        RDK.Item(item["item"]).setVisible(False)

    for item in items:
        xyz, angle = find_object(img_scene, item["pic"], keypoints_scene,
descriptors_scene)
        if xyz is None:
            continue
        pose = TxyzRxyz_2_Pose([xyz[0,3],xyz[1,3],0,0,0,angle])
        target_pose = TxyzRxyz_2_Pose([xyz[0,3],xyz[1,3],200,
                                3.1415,0,0])
        RDK.Item(item["item"]).setPose(pose)
        RDK.Item(item["item"]).setVisible(True)

        target = RDK.AddTarget(item["item"])
        target.setPose(target_pose)
```

Prvotno se provjera postoje li targeti od prethodno pokrenutih programa te se brišu svi osim „Home“ pozicije te se svi predmeti koji se žele pomaknuti sakriju. Nakon toga, za svaki predmet se radi „feature-matching“ pomoću kojeg računamo koordinate i rotaciju kako je već prije objašnjeno. Postavljaju se poze predmeta i poze targeta, koje se razlikuju jedna od druge zbog toga što smjer Z-osi mora gledati u smjeru alata te povišen za sve predmete na istu visinu jer od te točke se spuštamo linearno za hvatanje. Nakon što se definiraju sve poze, računa se njihove udaljenost od baze robota koristeći euklidsku udaljenost te se sortiraju tako da je prvi predmet koji se hvata onaj najbliži robotu kako bi se izbjegle moguće kolizije. Opisani koraci su također vizualno prikazani u dijagramu toka u nastavku.

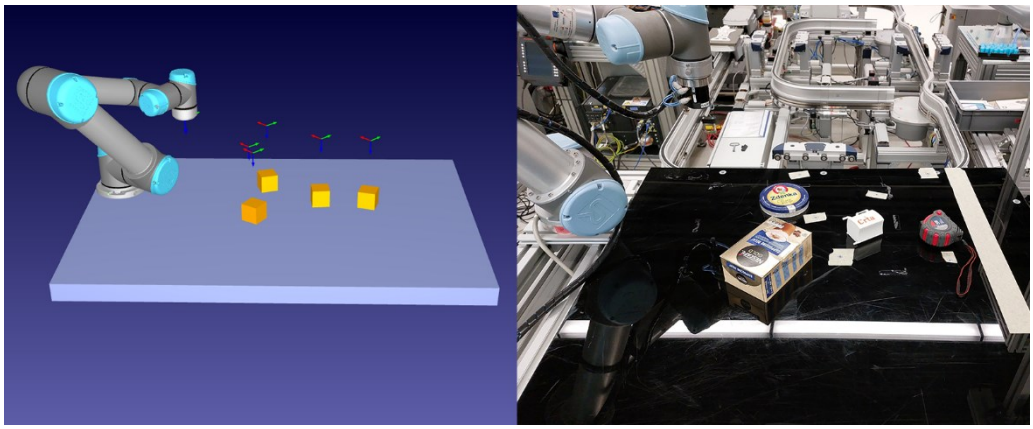
```
    target_dist = []

    for item in items:
        target = RDK.Item(item["item"], 6)
        pose = Pose_2_TxyzRxyz(target.Pose())
        distance = sqrt(pose[0]**2 + pose[1]**2)
        target_dist.append({"item" : item["item"], "distance" : distance,
"height" : item["height"]})

    sorted_targets = sorted(target_dist, key=lambda d: d['distance'])

    for item in sorted_targets:
        target = RDK.Item(item["item"], 6)
        robot.MoveJ(target)
        robot.MoveL(target.Pose()*transl(0,0,200-item["height"]))
        robot.MoveL(target)
        robot.MoveJ(basket)
```



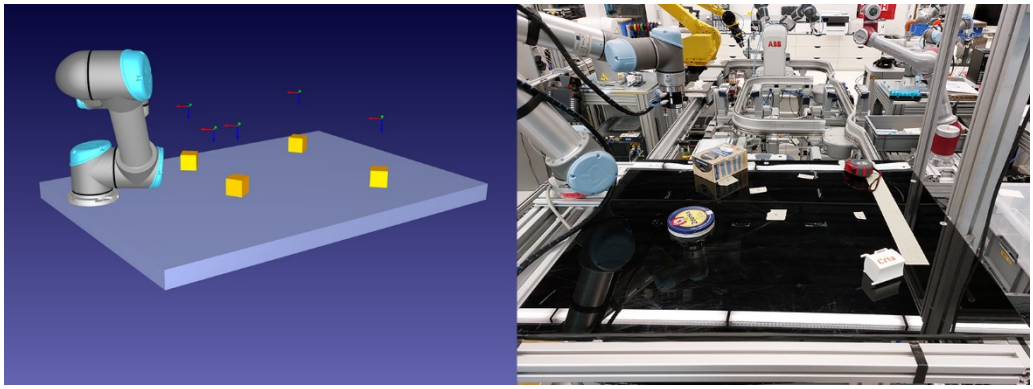
Slika 5.1 Prikaz stvarnog i simuliranog okruženja – situacija 1



Slika 5.2 Prikaz stvarnog i simuliranog okruženja – situacija 2



Slika 5.3 Prikaz stvarnog i simuliranog okruženja – situacija 3



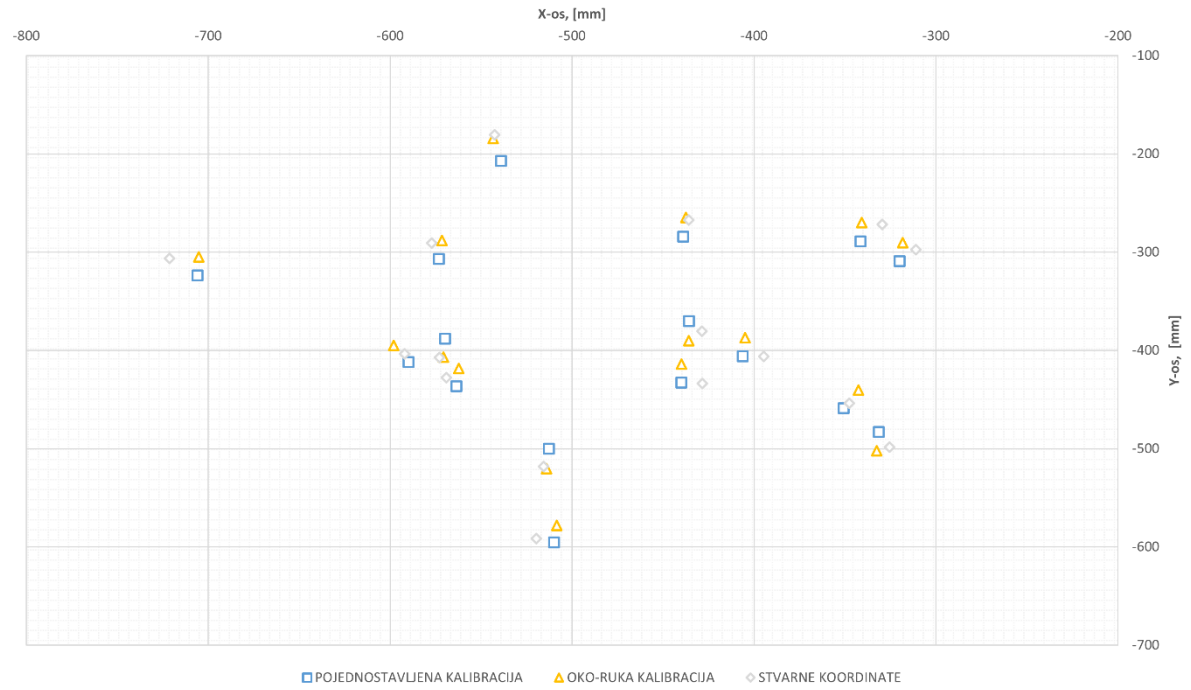
Slika 5.4 Prikaz stvarnog i simuliranog okruženja – situacija 4

Kao što je vidljivo, sustav uspješno pronalazi sve predmete i uspješno ih lokalizira i pozicionira na ispravnim pozicijama.

6. EVALUACIJA REZULTATA

Nakon što je cijeli sustav postavljen i upogonjen, napravljena su mjerenja stvarnih pozicija predmeta i uspoređena s onima koje se dobiju transformacijama koordinatnih sustava koristeći oba načina kalibracije. Predmeti su postavljeni u nasumične pozicije na radnom stolu te je zatim pokrenut program prepoznavanja predmeta dva puta, prvi put koristeći kalibraciju oko-ruka, a drugi put koristeći pojednostavljenu kalibraciju nakon čega su izmjerene pozicije predmeta dovodeći alat robota u središte predmeta.

U radu su korištena četiri predmeta različitih tekstura i oblika kako bi se testirala robusnost lokalizacije, od kojih su svi više-manje svakodnevni predmeti.



Slika 6.1 Prikaz stvarnih točaka u odnosu na kalibracije

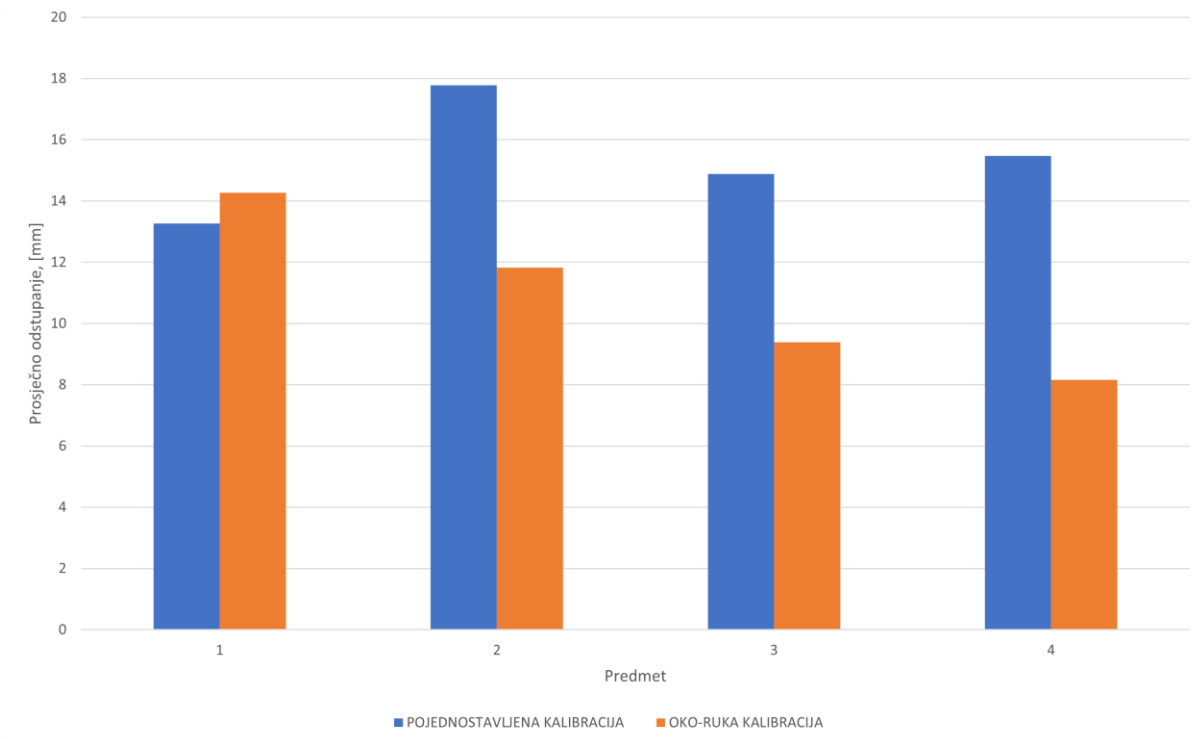
Gledajući prikaz svih mjerenja i odstupanja, može se vidjeti da koristeći obje kalibracije, rezultati su zadovoljavajući te s obzirom na veličinu predmeta, ne bi trebalo biti problema kod hvatanja predmeta. Također, primjećujemo da pri korištenju kalibracije oko-ruka, odstupanja bivaju sve veća kako se predmeti nalaze dalje od optičkog centra kamere.

Slika 6.2 Pozicije predmeta 1

Slika 6.3 Pozicije predmeta 2

Slika 6.4 Pozicije predmeta 3

Slika 6.5 Pozicije predmeta 4



Slika 6.6 Prosječna odstupanja svakog predmeta

Računajući prosječna odstupanja za svaki predmet u sva četiri mjerenja, možemo vidjeti da kalibracija oko-ruka ima znatno manju pogrešku, ukazujući na točniju transformaciju te potencijalno precizniji omjer piksela i milimetara. U svakom slučaju, zaključujemo da je preciznost sustava zadovoljavajuća za naše potrebe, no ovakav sustav ne bi mogao baratati manjim predmetima. Za takav slučaj, potrebno je odraditi kalibraciju pazeći više na postav kalibracijske ploče nastojeći da je što točnije centriran koristeći više poza robota te da su intrinzični parametri kamere preciznije izračunati. Također, važno je napometi kako odstupanja su kulminacija pogrešaka počevši od „feature-matchinga“, rektifikacije slike, kalibracije te same intrizične točnosti robota.

7. ZAKLJUČAK

U ovom radu je uspješno osmišljen, razvijen i implementiran cjelovit sustav koji značajno poboljšava mogućnosti upravljanja robotom i automatizacije unutar RoboDK simulacijskog okruženja. Integracijom metoda strojnog vida i RoboDK API-a, sustav stabilno locira i vizualizira predmete unutar radnog prostora omogućavajući jednostavno planiranje putanje unutar RoboDK. Automatsko generiranje predmeta i putanja predstavlja pojednostavljen način za razvijanje i prototipiranje integriranih vizijskih sustava, omogućavajući brzu i jednostavnu validaciju ispravnosti kalibracije kao i mogućih trajnih programa. Također je prikazana usporedba korištenja dviju vrsta kalibracija kao i korištenje više algoritama za prepoznavanje značajki dajući adekvatne rezultate nakon čega se može zaključiti da je sustav spreman za daljnji razvoj i potencijalnu primjenu. Uspoređujući odstupanja je vidljivo da su rezulati lokalizacije dobiveni pomoću oko-ruka kalibracije točniji za 5 milimetara od onih dobivenih pojednostavljenom kalibracijom, gdje je prosječno odstupanje za oko-ruka kalibraciju bilo 10,9 milimetara dok je za pojednostavljenu bilo 15,35 milimetara. Ovakav pristup nije samo praktičan, već i financijski efikasan, primjenjujući programske knjižnice otvorenog koda čime se dopušta razvoj bez ikakvih troškova licenciranja. Treba ipak imati na umu da razvoj, a pogotovo i održavanje ovakvog sustava, zahtjeva relativno specifična znanja, čime se potencijalno primjena neispoliranog sustava vjerojatno otežava.

Daljnji razvoj sustava koristeći duboko učenje ili pak 3D kamere otvara vrata za rješavanje šireg spektra rješenja koji bi potencijalno mogli biti robusniji i fleksibilniji.

LITERATURA

[1] Hand-Eye Calibration Problem ,https://support.zivid.com/en/latest/academy/applications/hand-eye/hand-eye-calibration-problem.html, 18.2.2024.

[2] OpenCV, https://en.wikipedia.org/wiki/OpenCV, 18.2.2024.

[3] Hand-Eye Calibration, https://campar.in.tum.de/Chair/HandEyeCalibration, 20.2.2024.

[4] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, M.J. Marín-Jiménez: Automatic generation and detection of highly reliable fiducial markers under occlusion, 2014.

[5] Filip Šuligoj: Vizijski sustavi u robotici, vježbe

[6] R. Szeliski, Computer Vision: Algorithms and Applications, 2020.

[7] David G. Lowe: Object recognition from local scale-invariant features, 1999.

[8] Edward Rosten, Tom Drummond: Machine Learning for High-Speed Corner Detection, 2006.

[9] Ethan Rublee, Gary Bradski: ORB: an efficient alternative to SIFT or SURF, 2011.

[10] Basic concepts of the homography explained with code, https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html, 18.2.204.

PRILOZI

I. Izvorni kod i stanica u RoboDK: https://github.com/blunttachi/simulated_opencv_robodk