Vizijski sustav za upravljanje industrijskim robotom pokretima ruke

Zidarić, Matija

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Matija Zidarić

Zagreb, 2024.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:

Izv. prof. dr. sc. Tomislav Stipančić, dipl. ing.

Student:

Matija Zidarić

Zagreb, 2024.

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

Srdačno zahvaljujem svojem mentoru izv. prof. dr. sc. Tomislavu Stipančiću na stručnim savjetima, strpljenju i motivaciji za završetak studija. Zahvale i njegovom asistentu Leonu Korenu, mag. ing. na pomoći oko postavljanja i debugiranja eksperimentalnog robotskog postava.

Hvala mojoj obitelji na podršci i strpljenju kada je bilo potrebno. Male stvari se jako cijene.

Želim se zahvaliti svim prijateljima koji su bili dio ove Zmajske odiseje kojoj došao je kraj i koji su sudjelovali u njoj. Ne želim nikoga posebno navoditi, jer oni koji znaju da sam im iskreno zahvalan od srca to već znaju i bez riječi.

M. Zidarić

„I have no special talent, I am only passionately curious.“

-Albert Einstein

“You can prove anything you want by coldly logical reason---if you pick the proper postulates.”

-Isaac Asimov - I, Robot

„Have no fear of perfection –you'll never reach it.“

-Salvador Dali

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

Središnje povjerenstvo za završne i diplomske ispite

Povjerenstvo za diplomske ispite studija strojarstva za smjerove:

Proizvodno inženjerstvo, inženjerstvo materijala, industrijsko inženjerstvo i menadžment, mehatronika i robotika, autonomni sustavi i računalna inteligencija

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 - 04 / 24 - 06 / 1**	
Ur.broj: **15 - 24 -**	

DIPLOMSKI ZADATAK

Student: **Matija Zidarić** JMBAG: 0035204411

Naslov rada na hrvatskom jeziku: **Vizijski sustav za upravljanje industrijskim robotom pokretima ruke**

Naslov rada na engleskom jeziku: **A vision system for controlling an industrial robot with hand movements**

Opis zadatka:

Modeli strojnog vida omogućuju računalima da prepoznaju značajke ruke te da ih prate u stvarnome vremenu. Intel RealSense D435 je stereo vizijska kamera s IR i RGB modulom, te je prikladna za primjene u robotici, virtualnoj stvarnosti (engl. Virtual Reality – VR), proširenoj stvarnosti (engl. Augmented Reality – AR) i slično. RoboDK je komercijalni softverski paket koji omogućuje online i offline programiranje različitih industrijskih robota.

U radu je potrebno:

- izraditi cjelovito softversko rješenje za prepoznavanje i praćenje ruke u stvarnome vremenu korištenjem Intel RealSense D435 kamere
- izraditi upravljačku programsku podršku za upravljanje industrijskim robotom
- povezati programsko rješenje za prepoznavanje i praćenje ruke s industrijskim robotom
- dobiveno softversko rješenje eksperimentalno evaluirati te dati kritički osvrt na rad aplikacije.

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

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
9. svibnja 2024.	11. srpnja 2024.	15. – 19. srpnja 2024.

Zadatak zadao: Predsjednik Povjerenstva:

Izv. prof. dr. sc. Tomislav Stipančić Prof. dr. sc. Ivica Garašić

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS TEHNIČKE DOKUMENTACIJE

BROJ CRTEŽA	Naziv iz sastavnice
AH-1	Adapter hvataljke
PH-1-20	Prst hvataljke V1-20
PH-2-20	Prst hvataljke V2-20
APH-1-1-20	Adaptivni prst hvataljke
APH-1-2-20	Adaptivni prst hvataljke

SAŽETAK

Moćne kamere i suvremeni modeli umjetne inteligencije omogućuju zanimljivu sintezu i različite aplikacije. Fokus rada jest izrada vizijskog sustava koji prepoznaje geste ruke te upravlja industrijskim robotom u realnome vremenu. U *Python* programskome jeziku koristi se *MediaPipe Hands* model za prepoznavanje i praćenje ruke koju snima Intel RealSense D435 kamera, a *RoboDK* je posrednik za komunikaciju s robotima. Zbog šuma sustava izvršeno je digitalno filtriranje koordinata. Robotske hvataljke izrađene su 3D tiskanjem za potrebe eksperimentalne evaluacije. Aplikacija je testirana na UR5 i Fanuc LR Mate 200iC industrijskim robotima.

Ključne riječi: Vizijski sustav, industrijska robotika, Python, MediaPipe Hands, RealSense D435, RoboDK, digitalni filter, 3D tiskanje, UR, Fanuc

SUMMARY

Powerful cameras and modern artificial intelligence models allow for interesting synthesis and different applications. The focus of this thesis is the development of a vision system that recognizes hand gestures and controls the industrial robot in real time. In the Python programming language, a MediaPipe hands model is used to recognize and track the hand being captured by the Intel RealSense D435 camera, withRoboDK as the intermediary for communication with robots. Due to system noise, digital filtering of coordinates was carried out. Robot grips have been made by 3D printing for the purpose of experimental evaluation. The application was tested on UR5 and Fanuc LR Mate 200ic industrial robots.

Keywords: Vision system, industrial robotics, Python, MediaPipe Hands, RealSense D435, digital filter, 3D printing, UR, Fanuc

1. UVOD

Danas je robotika svuda oko nas; nije više ograničena na klasične tvornice, već se može pronaći na cesti u obliku vozila, u kući u obliku usisavača ili pak kosilice, u bolnicama kao sustav za operacije i liječenje ljudi itd. Aktualan je trend je povećanja interakcije između ljudi i robota te povećanja njezine kvalitete i jednostavnosti izvedbe. Današnji modeli umjetne inteligencije i dostupne moćne kamere kao „oči“ sustava omogućavaju svestranost i razvoj raznih aplikacija, bilo za rekreativne ili profesionalne primjene.

Ovaj rad bavi se sintezom industrijskih robota sa suvremenim modelima umjetne inteligencije za prepoznavanje ruke u svrhu izrade svestranog vizijskog sustava za upravljanje industrijskim robotima. Već su napravljena slična rješenja, međutim ideja ove aplikacije jest fleksibilnost primjene na različitim robotima u realnome vremenu.

Svrha takvog sustava jest lakše i prirodnije upravljanje robotima pokretima i gestama ruke, te njihovo programiranje. Danas je također prisutan i trend korištenja kobota, međutim i dalje je veliki broj industrijskih robota, a ova aplikacija bi se mogla iskoristiti za elegantnije upravljanje tim robotima, pa i za izradu jednostavnih programa.

1.1. Struktura rada

U drugom poglavlju rada navedena je teorijska podloga s najvažnijim pojmovima te kratki opis korištenih alata, opreme i programa. Također je opisana i kratka razrada ideje. Treće poglavlje rada opisuje razradu ideje rada. Četvrto poglavlje bavi se usporedbom i odabirom filtera za obradu koordinata kamere. Peto poglavlje rada sastoji se od opisa računalnog koda. Opisan je dijagram toka programa, te pojedini programski blokovi i kreiranje korisničkog programa. Kratko je opisan postupak pripreme upravljačkih programa (engl. *driver*) na robotima za rad. Slijedi opis korisničkog sučelja aplikacije, te način korištenja aplikacije i njene značajke. Peto poglavlje objašnjava postupak izrade adaptivnih prihvatnica (engl. *gripper*) robota za eksperimentalne postavke. Sedmo poglavlje opisuje postupak stvaranja *RoboDK* radne stanice kao priprema prije početka korištenja aplikacije. Osmo poglavlje rada je eksperimentalna evaluacija, te opisuje testiranje aplikacije na dva različita robota i prikaz različitih značajki aplikacije. U zadnjem dijelu rada nalazi se kritički osvrt i zaključak. Na kraju rada jest prilog koji sadrži *Python* računalni kod aplikacije i izrade grafova, dijagram toka programa i tehničku dokumentaciju.

2. TEORIJSKA PODLOGA RADA, KORIŠTENA OPREMA I ALATI

2.1. Robotika

Interdisciplinarno znanstveno područje kojem je fokus na projektiranju, konstruiranju, upravljanju i primjeni robota naziva se robotika. Baza robotike je mehatronika, odnosno objedinjuje znanstveno-tehnička područja strojarstva, elektrotehnike i elektronike, automatizacije, računarstva i umjetne inteligencije.

Automatizirani strojevi koji imaju višestruku funkciju i automatizirani su, nazivaju se robotima. Sastoje se od mehaničkog dijela konstrukcije, pogonskih uređaja (elektromotora ili pogonjenih hidrauličkim ili pneumatskih aktuatorima), senzora, upravljačkog uređaja i računalnog programa. Roboti se mogu podijeliti prema različitim kriterijima poput pokretljivosti (mobilni i statični), strukturi osnovne konstrukcije („klasični“ mehatronički, biotronički i bioroboti), osnovnoj namjerni (industrijski, edukacijski, podvodni, zračni odnosno leteći, za rad u svemiru i specijalnim uvjetima poput radijacije, medicinski, osobni ili pak vojni) i veličini (makro, mikro i nano roboti). Inteligentni roboti su roboti koji posjeduju sposobnost učenja, rasuđivanja i donošenja zaključaka na temelju signala iz okoline, a imaju visok stupanj autonomije (mobilne, funkcionalne ili organizacijske).

Umjetnik i znanstvenik Leonardo da Vinci (1452. – 1519.) imao je jedan od prvih koncepata koji je odgovarao robotu – pokretni stroj u obliku lava. Termin robot uveo prvi je uveo češki književnik Karel Čapek (1890. – 1938.) u svojoj poznatoj drami R. U. R. – Rossumovi Univerzalni Roboti. Koristio je riječ *„robotnik“* koja se može prevesti kao radnik, rob i sl., a zamišljen je kao čovjekoliki stroj koji posjeduje sposobnost rasuđivanja. Termin robotika uveo je Isaac Asimov (1920. – 1992.) u svom znanstveno-fantastičnom djelu Izmotavanje (*Runaround*) koje je objavljeno 1942. Početkom 1960-ih u SAD-u George Devol (1912. – 2011.) i Joseph Engelberger (1925. – 2015.) započeli su komercijalnu proizvodnju prvog industrijskog robota pod nazivom Unimate. [1]

2.1.1. Industrijska Robotika

Industrijski roboti mogu se programirati za obavljanje opasnih, prljavih i/ili ponavljajućih zadataka (4D – *Dangerous, Dirty, Dumb, Dull*) s dosljednom preciznošću i točnošću, a industrijski se roboti sve više koriste u raznim industrijama i aplikacijama. Dolaze u širokom rasponu modela s različitim dohvatom, nosivošću i brojem osi kretanja (SSG – stupnjevi slobode gibanja).

I u proizvodnji i u manipulaciji robot koristi krajnji efektor ili krajnji alat za ruku (EOAT – *End Of Arm Tooling*) za držanje i manipuliranje komada na kojem se proces izvodi. Radnje robota upravljane su kombinacijom softvera za programiranje i upravljanjem. Njihova automatizirana funkcionalnost omogućuje im rad danonoćno i vikendom, rad s opasnim materijalima i u izazovnim okruženjima, te tako oslobađa osoblje za obavljanje drugih zadataka. Robotska tehnologija također povećava produktivnost i profitabilnost dok eliminira radno intenzivne aktivnosti koje bi mogle uzrokovati fizički napor ili potencijalnu ozljedu radnika.

2.1.1.1. Primjena industrijske robotike

Industrijski se roboti koriste u raznim primjenama, a neke od njih su niže navedene:

Rukovanje: manipuliranje različitim proizvodima poput vrata automobila pa sve do jajeta, industrijski roboti su brzi i moćni, kao i spretni i osjetljivi. Primjene uključuju izuzimanje i odlaganje od pokretne trake do pakiranja i opsluživanja stroja, gdje robot unosi sirovine ili poluproizvode u procesnu opremu za obradu kao što su strojevi za injekcijsko prešanje, CNC strojevi, preše itd.

Paletizacija: industrijski roboti utovaruju kartonske kutije ili druge zapakirane artikle na paletu prema definiranom uzorku. Robotski uređaji za paletizaciju oslanjaju se na fiksni položaj ili prilagođene vizijske sustave koji se povezuju s individualnim komponentama tereta, gradeći od jednostavnih do složenih uzoraka slojeva na vrhu palete koji maksimiziraju stabilnost tereta tijekom transporta. Postoje tri primarne vrste paletiranja: inline ili slojevito oblikovanje, depaletiziranje ili istovar i mješoviti slučaj.

Rezanje: zbog svoje opasne prirode, laserski, plazma i vodeni rezači često se koriste s robotima. Stotine različitih putanja rezanja mogu se isprogramirati za robota, što rezultira preciznim i točnim praćenjem putanje uz veću fleksibilnost od većine namjenskih strojeva za rezanje.

Završna obrada: roboti s više osi mogu brusiti, podrezivati, čistiti, polirati i čistiti gotovo svaki dio izrađen od bilo kojeg materijala za dosljednu kvalitetu završne obrade.

Brtvljenje i lijepljenje: za nanošenje brtvila ili ljepila, robot točno slijedi putanju s dobrom kontrolom brzine dok održava dosljedan sloj ljepljive podloge. Roboti se često koriste u automobilskoj industriji za brtvljenje prozora, kao i u procesima pakiranja za automatizirano brtvljenje kutija proizvoda.

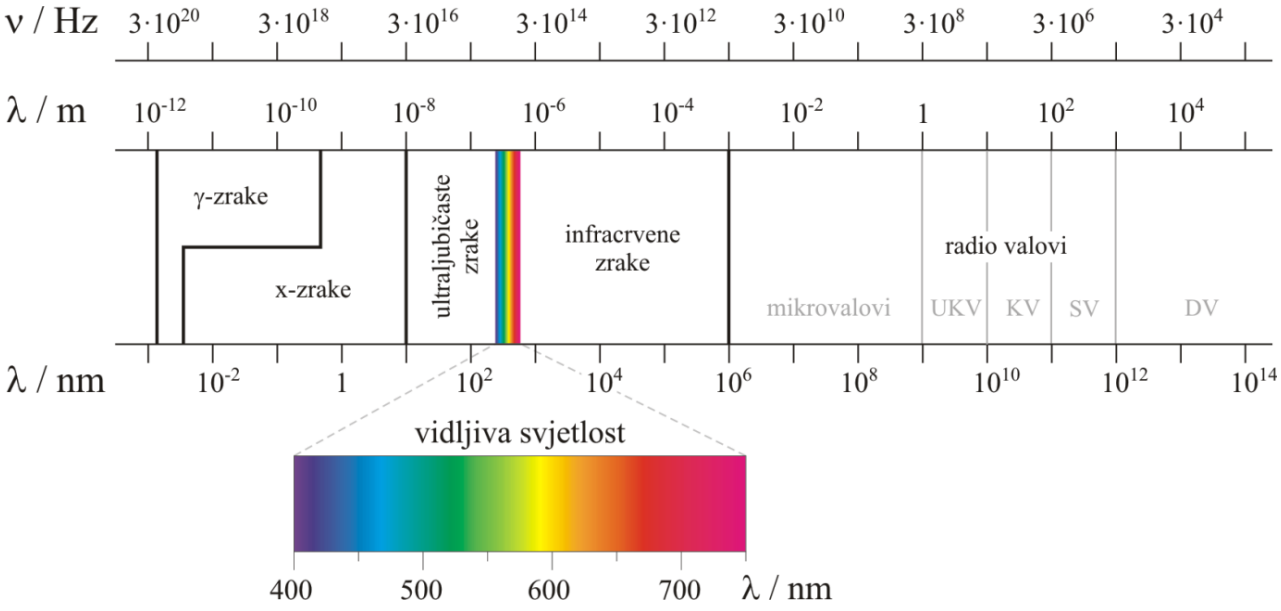
Raspršivanje: zbog hlapljive i opasne prirode boja i premaza na bazi otapala, roboti se koriste u primjenama sprejanja kako bi se kontakt s ljudima sveo na najmanju moguću mjeru. Roboti za

bojanje obično imaju tanke ruke jer nemaju veliku težinu, ali im je potreban maksimalan pristup i fluidnost kretanja kako bi oponašali ljudsku tehniku nanošenja boje i premaza.

Zavarivanje: koristeći se za šavno (MIG, TIG, elektrolučno i lasersko) i točkasto zavarivanje, roboti izvode precizne zavare. Također se mogu i precizno kontrolirati parametri kao što su snaga struje, dovod žice, protok plina i sl. [2]

2.2. Vid

Organ čovjeka i mnogih životinja koji pretvara informacije svjetlosti u živčane impulse naziva se oko. Ljudi posjeduju par očiju što omogućava stereo percepciju, odnosno procjenu udaljenosti objekata. [3] Prozirni, prednji dijelovi oka lome zrake svjetlosti i projiciraju umanjenu i obrnutu sliku na fotosenzitivnu mrežicu gdje specijalizirane živčane stanice obavljaju pretvorbu u električne živčane impulse. [4] Od svih ljudskih osjetila, oko je najvažnije jer preko 90% informacija okoline se percipira njime uključujući percepciju dubine, boje i oblika. Ljudsko oko može razlikovati 10^6 nijansi boja, a vidni kut iznosi 200°. Vidljivi spektar koji ljudsko oko može zamijetiti odgovara nizu boja koje nastaju rasapom Sunčeve svjetlosti. Sastoje se od ljubičaste valne duljine 380 nm, preko plave, zelene, žute, narančaste i crvene koja iznosi 780 nm. [5]



Slika 1. Spektar elektromagnetskog zračenja [6]

2.2.1. Računalni vid

Računalni vid područje je umjetne inteligencije koje omogućava računalima i sustavima da izvuku korisne informacije iz digitalnih slika, videa i drugih vizualnih ulaza, te da poduzmu određene radnje ili daju preporuku na temelju tih informacija. Učenje (trening) računalnog vida mora biti u razumnom vremenu koristeći kamere, podatke i algoritme u odnosu na čovjeka koji

ima prednost doživotnog konteksta za učenje. Relativno velika količina podataka je potrebna za kvalitetno učenje računalnog vida, a najčešće se koriste tehnologije dubokog učenja i konvolucijskih neuronskih mreža. Konvolucijske neuronske mreže pomažu algoritmu strojnoga učenja ili dubokog učenja na način da razlažu sliku na piksele kojima se daje marking (*label*). Ti labeli (oznake) koriste se u konvolucijama da mreža predvidi što „vidi“. Neuronska mreža pokreće konvolucije i provjerava točnost svojih predviđanja u nizu ponavljanja sve dok se predviđanja ne počnu ostvarivati. Slično kao što čovjek stvara sliku na daljinu, neuronska mreža prvo razaznaje oštre rubove i jednostavne oblike, a zatim popunjava informacije ponavljanjem konvolucija i predviđanja. Konvolucijske neuronske mreže se koriste za razumijevanje pojedinačnih slika, a povratne neuronske mreže (RNN) se koriste na sličan način za obradu video aplikacija za niz slika koje su povezane.

Razni su problemi s kojima se računalni vid susreće poput segmentacije, osvjetljenja, skaliranja, deformiranja, okluzija i preklapanja, pozadinske buke, kretanja, varijacija unutar iste klase, višeznačnosti, percepcije itd. Dio problematike rješava se prilikom prikupljanja samih podataka poput drugog načina osvjetljenja (direktno, pozadinsko, kupolno...) ili pak korištenjem kamera koje mogu vidjeti u drugome spektru svjetla (poput infracrvene kamere). Drugi način za rješavanje problema je programski, točnije, koriste se različiti filteri poput box filtera, Gaussovog filtera, median filtera i slično. Problemi segmentacije i varijacija unutar iste klase rješavaju se korištenjem umjetnih neuronskih mreža, ali ta rješenja ne moraju nužno biti jednostavna. [7]

2.3. Python

Python je svestran i široko korišten viši interpretacijski programski jezik opće namjene. Stvorio ga je Guido van Rossum 1990. Godine stvorio ga je Guido van Rossum, a inspiracija za ime je televizijska serija *Monty Python's Flying Circus*. Načini programiranja u Pythonu su proceduralno, strukturno i objektno orijentirano. Velik broj standardnih biblioteka, ali i broj paketa trećih strana, čini Python jako popularnim za razne aspekte programiranja. Neka područja primjene pythona su razvoj aplikacija i web aplikacija, analiza podataka, strojno učenje i umjetna inteligencija, znanstveno računarstvo, automatizacija različitih procesa i sl. Zbog svoje sintakse i relativno blage krivulje učenja, python je dobar program za početnike u programiranja, no svejedno je popularan izbor i za iskusnije programere zbog svoje fleksibilnosti i mogućnostima koje nudi. [8]

2.4. OpenCV

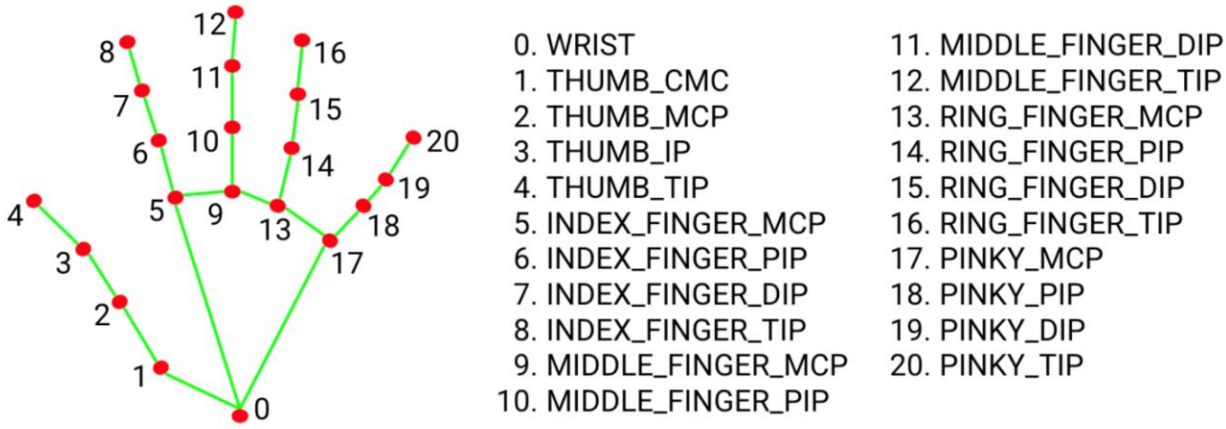
OpenCV (engl. *Open Source Computer Vision Library* – knjižnica otvorenog koda za računalni vid) visoko je optimizirana biblioteka otvorenog koda koja sadrži 2500 algoritama za računalni vid i strojno učenje. Podržava različite programske jezike, uključujući *C++*, *Python* i *Java*, te radi na više platformi kao što su Windows, Linux, macOS, iOS i Android. *OpenCV* popularan je alat za aplikacije poput obrade slika i videa, detekcije objekata i prepoznavanja lica. Održava ga neprofitna Open Source Vision Foundation i besplatan je za komercijalnu upotrebu pod licencom Apache 2. [9]

2.5. MediaPipe

MediaPipe googleov je razvojni okvir otvorenog koda koji nudi prilagodljiva rješenja za izradu aplikacija, a koja se temelje na umjetnoj inteligenciji i strojnome učenju. Neka od gotovih rješenja su prepoznavanje objekata, klasifikacija i segmentacija slika, prepoznavanje lica, poza i gesti. *MediaPipe* uključuje alate poput *MediaPipe Model Maker* za prilagodbu modela i *MediaPipe Studio* za vizualizaciju i procjenu rješenja. Podržava više platformi uključujući Android, web, Python i iOS, što ga čini popularnim među profesionalnim programerima, ali i istraživačima i inženjerima. [10] U ovome radu fokus je na prepoznavanju ruku i gesti ruku.

2.5.1. MediaPipe Hand landmarker (prev. Orijentiri ruke)

MediaPipe Hand Landmarker omogućava prepoznavanje orijentira ruke. Može se koristiti za lociranje ključnih točaka i renderiranje vizualnih efekata na njima. Radi sa slikovnih podacima s modelom strojnog učenja, a ulaz mogu biti statični podaci – slike ili kontinuirani tok – video. Moguće je prepoznati orijentaciju ruku u koordinatama slike, globalnim koordinatama, broj prepoznatih ruku, te raspoznavanje lijeve i desne ruke. Paket modela otkriva lokalizaciju 21 točke koordinata ruke, odnosno zgloba šake.



Slika 2. Prikaz orijentira ruku Hand Landmarker modela [11]

Model je učen na približno 30 000 slika iz stvarnoga svijeta, kao i na nekoliko renderiranih umjetnih modela ruku. S obzirom na to da je pokretanje modela detekcije dlanova dugotrajno u načinu rada videa ili trenutne snimke, *Hand Landmarker* koristi granični okvir definiran modelom orijentira ruke u jednome okviru za lokalizaciju područja ruku za sljedeće okvire. [11]

Tablica 1. Pregled evaluacije modela prepoznavanja ruku ovisno o različitim kriterijima [12]

		MNAE	**Standardna devijacija**
Regija	Europa	10.61	14.44
Tip boje kože	2	4.67	4.54
Spol	Muško	5.38	7.87

U tablici 1. prikazan je pregled podataka za mjerenje performasi modela *MediaPipe Hands*. Podaci su za cjeloviti model (*Full*), te uzimaju u obzir regiju uzimanja podataka – za ovaj slučaj Europa, tip boje kože 2 (prema Fitzpatrick klasifikaciji boje kože [13]), te muški spol. Podaci su podijeljeni na standardnu devijaciju, te na MNAE (engl. *Mean of Normalized Absolute Error by palm size* – srednja vrijednost normalizirane apsolutne pogreške prema veličini dlana) koji uzima u obzir unificiranje skaliranja uzoraka. To postiže mjerenjem veličine dlana od točke zapešća (0. Wrist) do prvog zgloba srednjeg prsta (9. MCP), kao što se može vidjeti na slici 9. Prema tablici 1. može se vidjeti da je za model greška od najmanje 4.54 mm, do čak 14.44 mm. [12]

2.5.2. Kazuhito00 model za prepoznavanje gesti

Kazuhito Takahasni izradio je na bazi *MediaPipea* program u *Pythonu* za prepoznavanje znakova i gesti ruku na bazi jednostavnog modela strojnog učenja. Program je prilagodljiv za učenje novih modela, odnosno za prepoznavanje novih gesti. U ovome radu neće se koristiti ta medota, već samo postojeći model. Zanimljivo ga je koristiti jer nije zahtjevan za procesor računala, a daje dobre rezultate za prepoznavanje gesti. U usporedbi s *MediaPipeom,* daje bolje i konzistentnije prepoznavanje. Prednost je što on točke prepoznaje iz slike u 2D koordinate, a *MediaPipe* koristi 3 koordinate. [14]

2.6. RoboDK

RoboDK simulator je industrijskih robota koji omogućuje integraciju i programiranje složenih robotskih rješenja. Intuitivno sučelje omogućuje offline programiranje robota bez direktnog znanja programiranja, ali isto tako ima i mogućnost povezivanja s *Pythonom, C#* i *C++* jezikom, matlabom i sl.

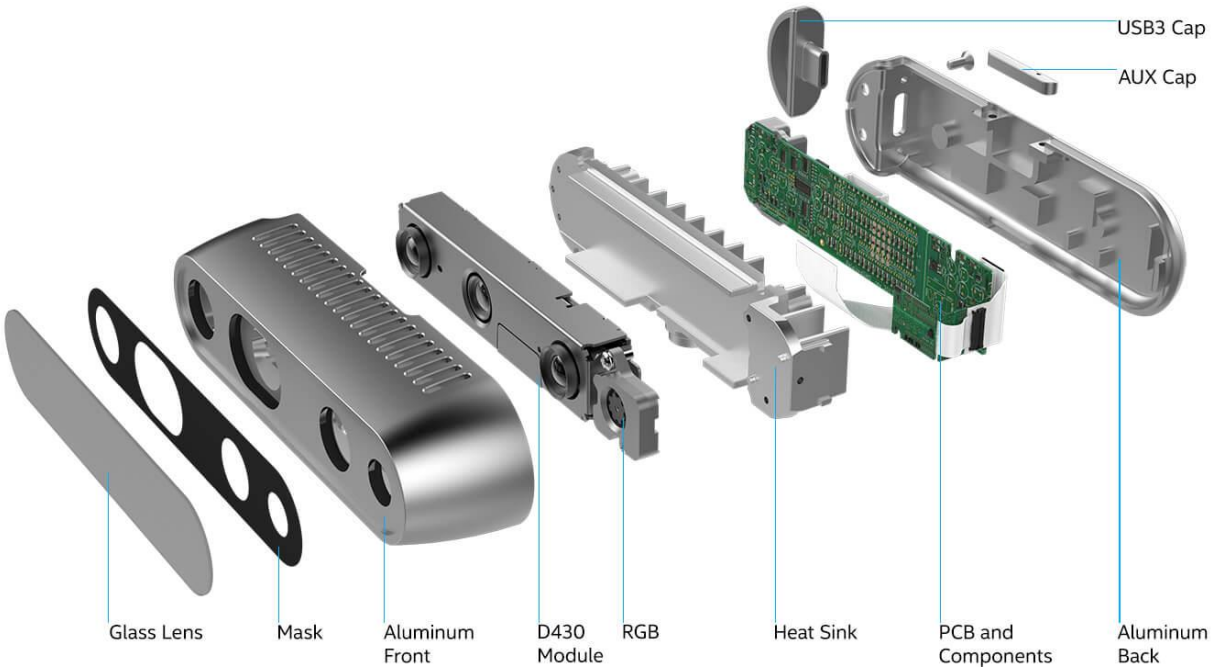
Tvrtku *RoboDK*, kao tvrtku kćer laboratorija *CoRo* na ETS sveučilištu u Montrealu, osnovao je Albert Nubiola 2015. U početku je *RoboDK* robotska knjižnica imala 200 robota s preko 20 proizvođača, a danas ima preko 1000 robota s preko 50 proizvođača. Trenutno je tvrtka bazirana u Kanadi i Europi, te ima tisuće aktivnih korisnika.

Primjene *RoboDK* su raznolike poput manipulacije, zavarivanja, višeosnog glodanja i bušenja, bojanja i nanošenja premaza, ispitivanja, kalibracije i sl. Aplikacija ima mogućnost simuliranja i konvertiranja NC programa u robotske programe (G kod ili APT-CLS instrukcije), te pritom automatski uzeti u obzir limite pojedinih osi, optimizaciju putanje robota da se izbjegne singularnost i kolizije. Neki od primjera integracije *RoboDK* su: sinkronizacija robota za ispitivanje trupa zrakoplova u NASA-i, izrada skulptura glodanjem robotima u Neoset Designsu, izrada robotizirane praonice zrakoplova u Wilder Systems i mnogi drugi.

Python API (engl. *Application Programming Interface* – sučelje za programiranje aplikacija) integriran je unutar *RoboDK* po standardu, te omogućuje jako dobru povezanost napisanih programa u pythonu prema *RoboDK*. Također su dostupni i mnogi primjeri programa na službenoj *RoboDK* stranici. [15]

2.7. *Intel RealSense D435* – dubinska kamera

Intelova *RealSense* stereo dubinska kamera *D435* ima širok spektar primjene u robotici, virtualnoj stvarnosti (engl. *Virtual Reality* – VR), proširenoj stvarnosti (engl. *Augumented Reality* – AR) i sl. Dometa do 10 m, ova kompaktna kamera se lako integrira u različite sustave.



Slika 3. Dijelovi Intel Realsense D435 kamere [16]

Tablica 2. Tehničke karakteristike *Intel Realsense D435* kamere [16]

Značajke	Uporaba: izvana/iznutra	Idealan raspon rada: 0.3 – 3 m
Dubina	Dubinska tehnologija: Stereoskopska Minimalna udaljenost (z-os) pri najvećoj rezoluciji: ~28 cm Preciznost dubine: < 2 % na 2 m Tehnologija senzora slike: Globalni zatvarač	Kut kamere: 87° x 58° Dubinska rezolucija: do 1280 x 720 Broj slika po sekundi: do 90 fps
RGB	RGB rezolucija 1920 x 1080 RGB broj slika po sekundi: 30 fps Tehnologija RGB senzora: Rotirajući zatvarač	Kut RGB senzora: 69° x 42° Rezolucija RGB senzora: 2 MP
Glavne komponente	Modul kamere: Intel RealSense Modul D430 + RGB kamera	Procesor za slike: Intel RealSense Vision Processor D4
Hardverske karakteristike	duljina x dubina x visina: 90 mm x 25 mm x 25 mm	Konektori: USB-C 3.1 Gen 1 Montaža: 1 x 1/4 -20 UNC navoj 2 x M3 navoj

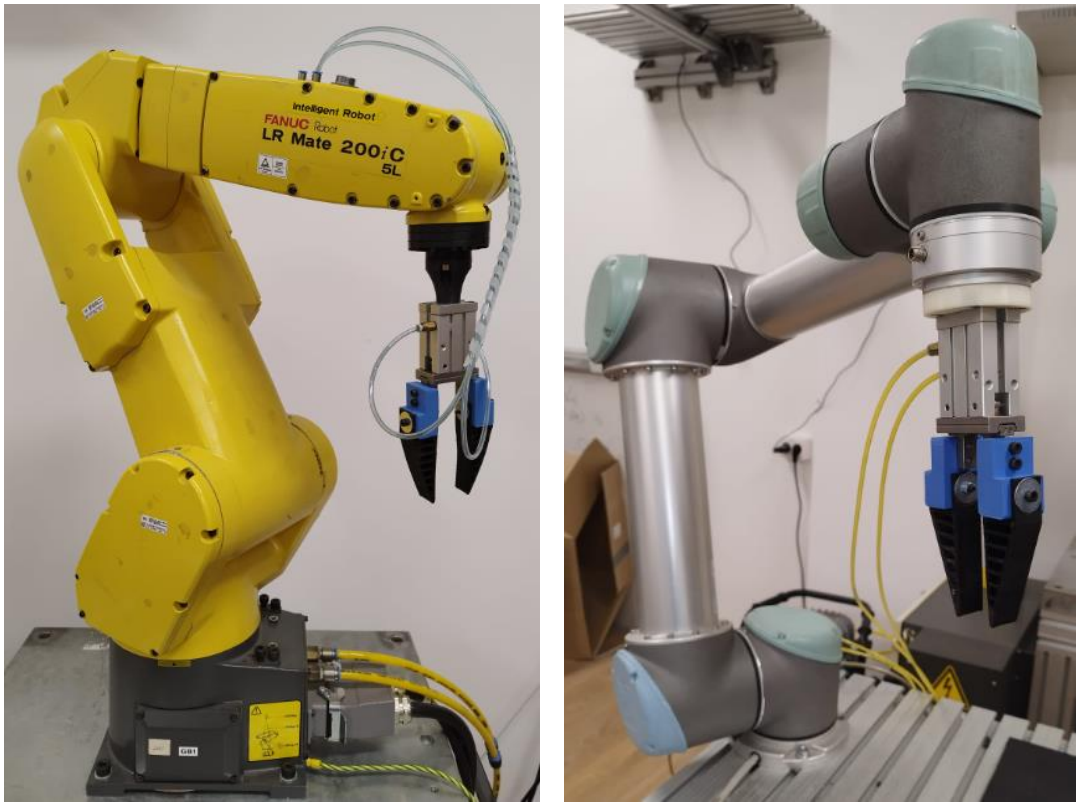
Kombinacija širokog kuta kamere i globalnog senzora zatvarača kamere omogućuje primjenu u robotskoj navigaciji, prepoznavanju objekata i u upotrebi smanjene vidljivosti. *Intel RealSense* SDK 2.0 posjeduje opciju samokalibriranja, unutar čipa, za D400 stereo kameru koja omogućuje jednostavnu kalibraciju za manje od 15 s. Samu kameru moguće je podesiti za rad u različitim okruženjima, ali isto tako su dostupne i gotove postavke. Mogući profili za odabir su profil visoke preciznosti (engl. *High Accuracy*), visoke gustoće (engl. *High Density*), srednje gustoće (engl. *Medium Density*), profil ruke (engl. *Hand*) i standardni profil (engl. *Default*). [16] Tehničke karakteristike prikazane su u tablici 2., a dijelovi kamere na slici 3.

2.8. Korišteni roboti

2.8.1. Industrijski robot LR Mate – 200iC 5L

FANUC LR Mate 200iC je kompaktna robotska ruka velike brzine, tankog profila i male težine. Visoka točnost pozicioniranja i veličina površine radnog prostora čine ga idealnim za razne primjene automatizacije. Integracija alata na LR Mate 200iC pojednostavljena je strateški smještenim pneumatskim i električnim priključcima, te s dva elektromagnetska ventila dvostrukog djelovanja. Lakši i kompaktniji od prethodnih modela LR Mate, 200iC ima isti tlocrt i uzorak vijaka za zapešće kao i prethodne generacije za jednostavnu nadogradnju. Zatvoreni

mehanički dizajn eliminira zapinjanje kabela i crijeva. Kompaktni R-30iA Mate kontroler izgrađen je na pouzdanosti standardnog FANUC R-30iA kontrolera. Koristan je u klimatiziranim uvjetima kao što su prehrambeni laboratorij, farmaceutska, medicinska i obrazovna okruženja. R-30iA Mate povezuje se sa standardnim FANUC iPendantom za jednostavno programiranje. [17]



Slika 4. Industrijski roboti Fanuc LR Mate 200 iC 5L (lijevo) i UR5 (desno) (slika autora)

2.8.1.1. Značajke robota

- Maksimalna nosivost Robota: 5 kg
- Najveći doseg: 892 mm
- Ponovljivost: 0.03 mm
- Kontroler: R-30iA Mate

2.8.2. Industrijski robot Universal Robots – UR5

UR5 svestran je i lagan kobot proizvođača *Universal Robots*. Kapacitet nosivosti od 5 kg, doseg od 850 mm i dizajn za širok spektar primjena čine ga idealnim za zadatke poput montaže, testiranja, manipulacije predmetima i slično. UR skupina robota je poznata po svojoj jednostavnoj integraciji, programiranju i fleksibilnosti što ga čini popularnim izbor za uvod u robotizaciju kod mnogih tvrtki. Zanimljiva je niska potrošnja električne energije u odnosu na druge industrijske robote iste klase. Robot dolazi s CB privjeskom za učenje koji ima

karakterističan ekran na dodir, te je intuitivan i jednostavan za korištenje. [18] U tablici 3. prikazana je usporedba raspona i brzine gibanja za svaku os korištenih robota.

2.8.2.1. Značajke robota

- Maksimalna nosivost Robota: 5 kg
- Najveći doseg: 850 mm
- Ponovljivost: 0.03 mm
- Kontroler: CB2

Tablica 3. Usporedba brzine i raspon gibanja industrijskog robota LR Mate – 200iC 5L i UR5[17][18]

	LR Mate 200iC 5L		**UR5**	
	Brzina gibanja osi	Raspon gibanja osi	Brzina gibanja osi	Raspon gibanja osi
Axis 1	270 °/s	± 340°	180 °/s	± 360°
Axis 2	270 °/s	± 230°	180 °/s	± 360°
Axis 3	270 °/s	± 373°	180 °/s	± 360°
Axis 4	450 °/s	± 380°	180 °/s	± 360°
Axis 5	450 °/s	± 240°	180 °/s	± 360°
Axis 6	720 °/s	± 720°	180 °/s	± 360°

3. RAZRADA IDEJE

Ideja rada jest napraviti aplikaciju koja koristi kameru za snimanje ruke korisnika, te u realnome vremenu prenosi gibanja ruke na gibanje industrijskog robota. Problemi su prepoznavanje ruke i gesti za razne operacije robota – poput otvorene i zatvorene hvataljke, te rad aplikacije u realnom vremenu zbog potencijalne latencije. Glavni dio prepoznavanja pokreta ruke i gesti izveo bi se korištenjem rješenja *MediaPipe*, koji se pokazao kao provjereno rješenje za razne aplikacije. Intelova *RealSense* kamera nudi mogućnosti preciznog prepoznavanja udaljenosti objekta u prostoru. Potrebno je povezati i uskladiti *MeadiaPipe* i *RealSense* značajke u realnome vremenu. Korištenje komercijalnog programa *RoboDK*, aplikaciji omogućuje dobru prilagodljivost za različite industrijske robote uz što manje izmjene koda.

Različite konfiguracije robotskih ćelija i radnih stanica traže i prilagodbu aplikacije, što također treba uzeti u obzir. Na početku koda zadale bi se dimenzije radne stanice u Kartezijevom koordinatnom sustavu. Važni su i sigurnost te ispitivanje rada aplikacije na stvarnom robotu, stoga se uzima u obzir izrada robotske prihvatnice adekvatne za eksperimentalni postav. Korištenjem 3D printera i metode brze izrade prototipova (engl. *rapid prototyping*) izradili bi se adapteri za postojeće robotske prihvatnice.

4. FILTRIRANJE OČITANIH KOORDINATA KAMERE

Prilikom rada primijećeno je znatno rasipanje, odnosno titranje koordinata zapešća koje kamera očitava. Iako je *Intel RealSense D435* kamera kalibrirana i namještena na način rada za veliku preciznost (*'High accuracy'*), razlika očitanja za stacionarni objekt je znala biti i preko 2,5 mm. Veći problem postavlja sami model *MediaPipea* koji titra i na koji je teže utjecati, a čija je moguća greška od 4.54 do 14.44 mm (tablica 1.). Kombinacija te dvije pojave rezultira očitanjem koordinata zamjetnog odstupanja koji se posebno ističe prilikom mirovanja ruke kada ne bi trebalo doći do promjene koordinata, odnosno ne želi se postići pomicanje robota. Za eksperimentalnu primjenu to je dovoljno velika preciznost, posebice jer korisnik može aktivnim upravljanjem to kompenzirati, međutim problem je kada se te koordinate šalju u *RoboDK* jer dolazi do pojave 'plesanja robota', odnosno titranje očitanja se prenosi na gibanje robota, te kvaliteta rada znatno pada. Prirodni tremor ruke je zanemaren jer se pretpostavlja da je iznos manji od 1 mm.

4.1. *Moving Average Filter* (prev. Filter Pokretnog Prosjeka)

Filter pokretnog prosjeka tehnika je u obradi i analizi različitih signala, a može izgladiti kratkoročne fluktuacije, te istaknuti cikluse ili dugoročne trendove. Ideja je da se računa prosjek nekog fiksnog broja uzastopnih točaka, odnosno podataka. Taj prosjek zatim zamijeni centralnu vrijednosnu točku, odnosno podatak i efektivno smanjuje varijancu – šum podataka. Primarni parametar filtra pokretnog prosjeka je veličina, odnosno broj točaka za računanje prosjeka, a označava se s N. Veći broj točaka za računanje prosjeka znači da će filter biti glađi, ali i tromiji na promjene u podacima. Manji broj točaka čini filter osjetljivijim na promjene u podacima, ali može unijeti veći šum nakon filtriranja. Koristi se u analizi tržišta i obradi podataka za znanstvene i inženjerske probleme. [19]

4.2. *Exponential Smoothing* (prev. Eksponencijalno izglađivanje)

Eksponencijalno izglađivanje tehnika je filtriranja koja primjenjuje eksponencijalno opadajuće težine na prošle podatke kako bi predvidjela vremenske nizove. Tehnika je posebno korisna kada su u pitanju podaci koji na prvi pogled nemaju jasan trend ili uzorak. Parametar filtera je faktor izglađivanja alfa (α), a kreće se između 0 i 1, te određuje koliko je veliki utjecaj prošlih očitanja. Viši iznos faktora izglađivanja daje veću težinu nedavno očitanim podacima, te čini model osjetljivim na promjene. Manji faktor pak filtrira podatke dajući veću težinu starijim podacima.

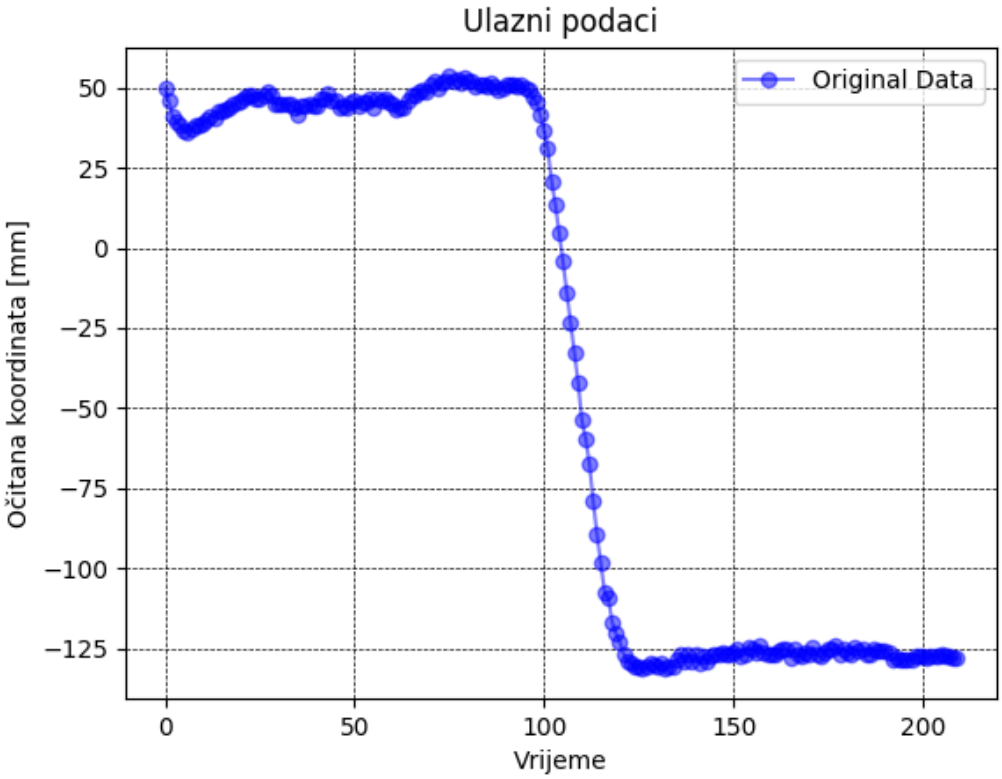
Uvođenjem dodatnih faktora beta (β) i gama(γ) postižu se varijante dvostrukog i trostrukog eksponencijalnog izglađivanja koji mogu dobro opisivati trendove i obrasce.[20]

4.3. Savitzky-Golay Filter (S-G filter)

Savitzky-Golay filter oblik je digitalnog filtera koji je dizajniran za izglađivanje skupa podataka tako da se očuvaju oblici i značajke signala poput širine i visine vrhova signala. Način na koji se to postiže je uklapanje uzastopnih nizova podskupova podataka s polinomom niskog stupnja (npr. 2. ili 3.) metodom linearnih najmanjih kvadrata. Glavni parametri Savitzky-Golay filtera su veličina skupa podataka i stupanj polinoma. Veličina skupa podataka utječe tako da veći broj podataka daje glađe rezultate, ali pak može isfiltrirati neke važne značajke signala, dok veličina polinoma određuje fleksibilnost filtra da prati lokalne trendove podataka. S-G filter posebno je zanimljiv u potručju spektroskopije i detaljne analize signala. [21]

4.4. Usporedba parametara filtera

Ovdje su uspoređeni različiti tipovi parametara ovisno o filtru, te su odabrani najbolji. Za ulazne podatke uzet je skup od 210 koordinata u prostoru.

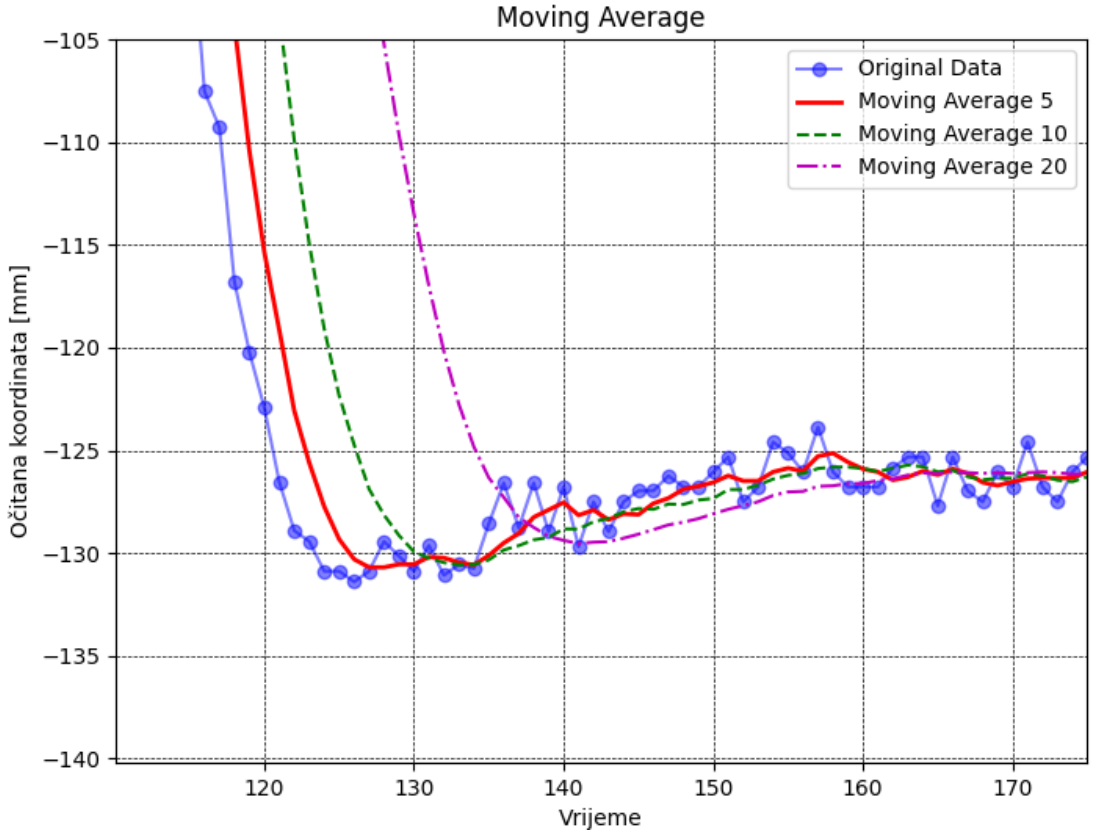


Slika 5. Prikaz ulaznih podataka za x koordinatu

Podaci su snimljeni na način da je ruka mirovala, napravljen je pomak ruke i na kraju ruka ponovno miruje. Na apscisi se nalazi redni broj točke iz uzorka, te je označena kao vrijeme bez mjerne jedinice jer nije eksplicitno mjereno vrijeme već su samo redom snimane koordinate.

Na ordinati su navedene vrijednosti koordinata u mm. Za analizu filtera uzete su vrijednosti od 110.-175. ulazne koordinate *x*, točke zapešća (engl. *wrist coordinates*) (slika 6.). To područje zanimljivo je jer uzima u obzir završetak pokreta ruke i njeno mirovanje. Grafovi su izrađeni korištenjem *Pythonove* knjižnice *Matplotlib*.

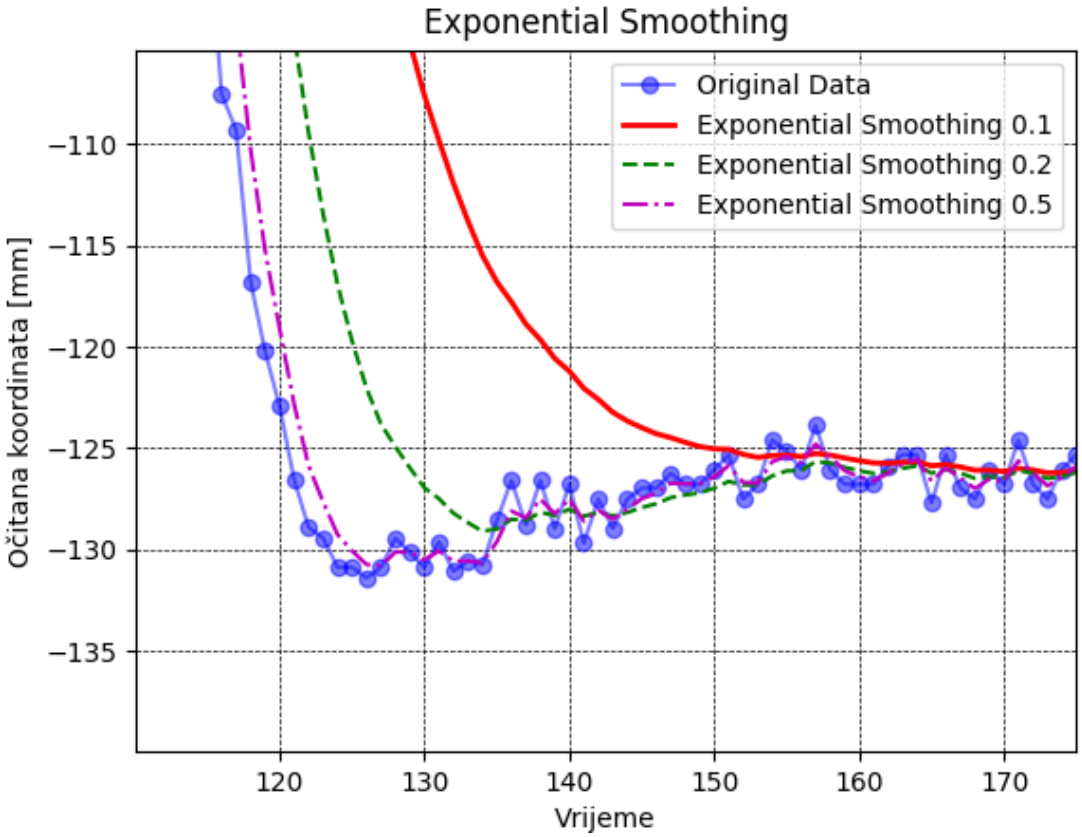
Za filter pokretnog prosjeka uzete su vrijednosti 5, 10 i 20 točaka (slika 7.). Vrijednost od 20 točaka ima veliko kašnjenje nakon završetka gibanja, dok vrijednost od 5 točaka počinje pokazivati slabe osobine filtra, tj. filtar previše prati originalne podatke i ne uklanja šumove. Kompromisno rješenje je logično težina od 10 točaka.



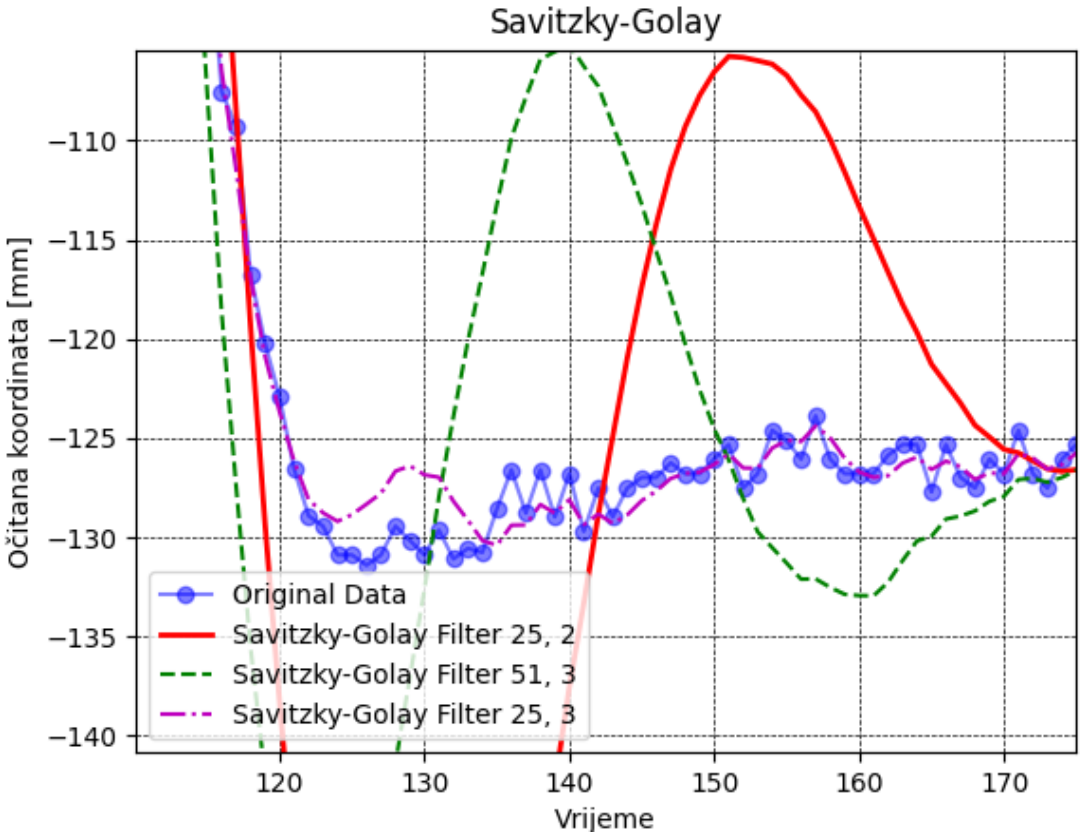
Slika 6. Usporedba različitih parametara filtera pomičnog prosjeka

Eksponencijalno izglađivanje uspoređeno je korištenjem tri faktora izglađivanja: 0.1, 0.2 i 0.5 (Slika 8.). Ponovno se pojavljuje kašnjenje i to kod vrijednosti od 0.1, dok 0.5 izrazito prati podatke, te pokazuje slabe karakteristike filtra. Kompromisni rješenje je vrijednost težinskog faktora od 0.2.

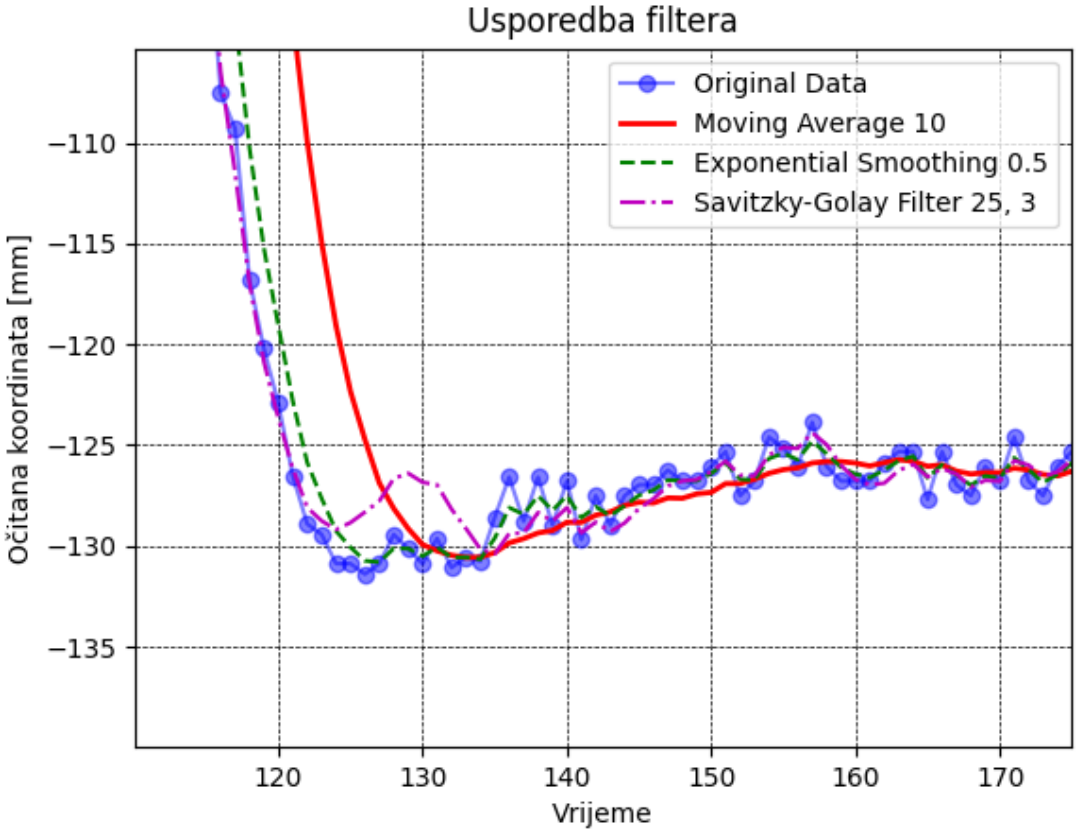
Savitzky-Golay filtar uspoređen je koristeći tri para parametara za broj točki i red polinoma: (25, 2), (51, 3), (25, 3) (Slika 9.). Za prva dva para vrijednosti primijećen je veliki prebačaj filtra, dok je za treći par primijećen manji prebačaj, ali slabe karakteristike filtera. Kao najbolja vrijednost uzima se treći par od 25 točaka i polinom trećeg stupnja.



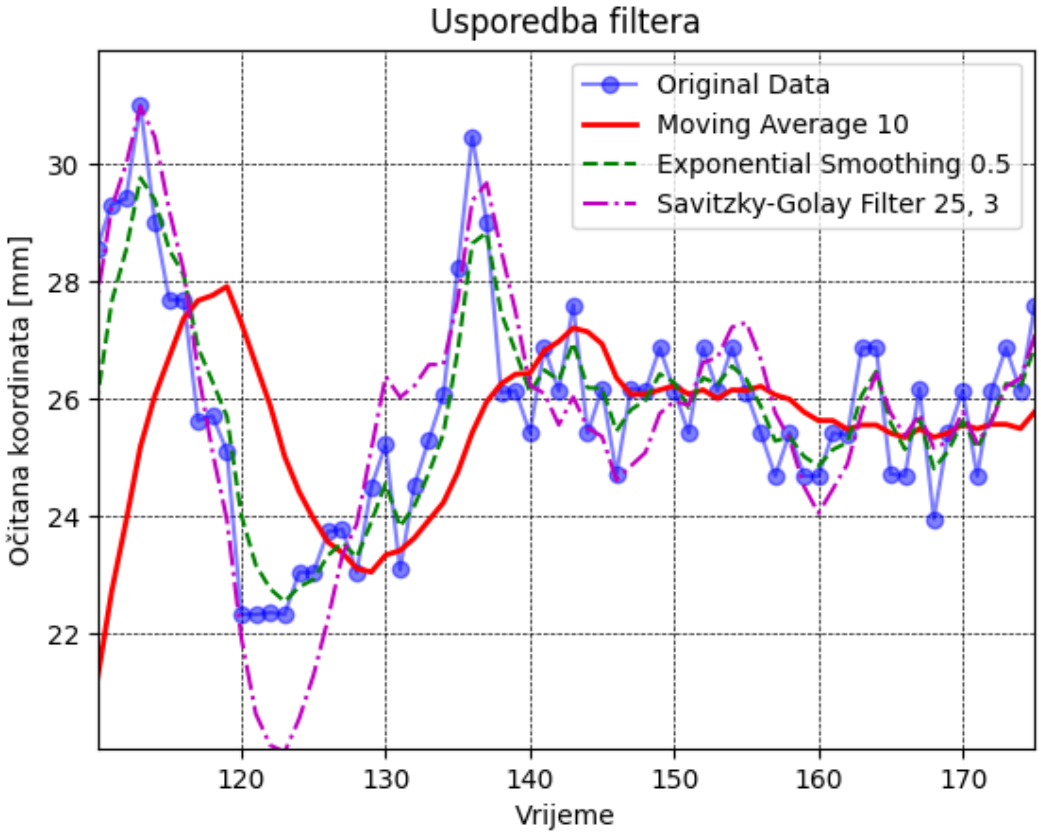
Slika 7. Usporedba različitih parametara filtra pomičnog prosjeka



Slika 8. Usporedba različitih parametara Savitzky-Golay filtra



Slika 9. Usporedba različitih parametara filtera na x koordinate



Slika 10. Usporedba različitih parametara filtera na y koordinate

Na slici 10. prikazana je usporedba triju različitih filtera s ranije odabranim parametrima, te se može vidjeti da su filteri Savitzky-Golay (25, 3) i eksponencijalno izglađivanje (0.5) bolje pratili pomak koordinate, ali prilikom mirovanja loše filtriraju šum. Pomični prosjek (10) ima kašnjenje prilikom pomaka, međutim bolje filtrira koordinate prilikom mirovanja, a to je ono što se želi postići.

Na slici 11. prikazana je usporedba istih filtera s istim parametrima kao i na slici 10., te je uzet isti raspon koordinata (od 110.-175.) za y koordinate. Radi se o većem šumu podataka nego kod x-koordinata, te se ponovno može primijetiti da pomični prosjek (10) ima najbolje karakteristike filtera koje tražimo za ovaj slučaj.

5. IZRADA PROGRAMSKOG RJEŠENJA

U ovome poglavlju opisat će se dijagram toka aplikacije koji se nalazi u prilogu, te računalni kod napisan u programskom jeziku *Python*. Nakon opisa koda slijedi opis korisničkog sučelja i način rada aplikacije.

5.1. Dijagram toka aplikacije

5.1.1. Početak programa

Na početku programa odvijaju se dvije glavne funkcije. Kreira se novi korisnički program iz predloška. U slučaju da postoji korisnički program, kreirat će se tako da mu se promijeni ime. Druga funkcija je početak rada programa i sastoji se od ograničenja radnog prostora robota, te se inicijalizira *RoboDK* i povezuje s radnom stanicom. U slučaju neuspješnog povezivanja, ono se ponovno inicijalizira. Nakon toga se pomiče robot u početni položaj koji je definiran u radnoj stanici. Započinje inicijalizacija kamere za unos videa, te inicijalizacija MediaPipea koji će pratiti ruku i prepoznavati geste.

5.1.2. Glavna petlja programa

Nakon početka programa ulazimo u glavnu petlju počinje dohvaćanje slike kamere, te manipulacija i obrada slika za *MediaPipe*. Slijedi prilagodba koordinatnih sustava između *MediaPipea* i kamere, te se dohvaćaju koordinate ruke. Provjerava se jesu li koordinate ruke unutar radnog prostora. Ako jesu, program se nastavlja, ako ne onda se ponovno dohvaćaju koordinate ruke. Slijedi filtriranje koordinata kako bi se smanjio šum sustava. Prepoznavanje geste ruke je idući korak, te se provjerava je li prepoznata nova gesta ili ne. Za slučaj da je prepoznata nova gesta, ovisno o gesti odvija se specifična radnja. Za prepoznat prst (engl. *pointer*) stvara se nova meta na trenutnim koordinatama, za prepoznat otvoreni dlan (engl. *open*) šalje se naredba za otvaranje hvataljke, a za šaku (engl. *closed*) šalje se naredba za zatvaranje hvataljke. Na kraju bilo koje od tih naredbi dolazi do zapisa te naredbe u korisnički program. Ako nova gesta nije prepoznata, provjerava se je li robot zauzet (engl. *busy*);ako je, onda se čeka da završi prethodnu radnju. Kada je robot slobodan, pomiče se u trenutnu koordinatu ruke i ta se koordinata ispisuje. Petlja se vrti sve dok korisnik ne odluči izaći iz programa.

5.1.3. Završetak programa

Na naredbu korisnika da izlazi iz programa, prekida se petlja i zatvaraju se svi potprogrami, odnosno procesi. Prekida se veza s kamerom, sprema se trenutno stanje *RoboDK* radne stanice kao i trenutno stanje korisničkog programa, te se zatvaraju. Time završava program.

5.2. Import modula

U samoj aplikaciji koristi se niz knjižnica (engl. *libraries*), s već napravljenim funkcijama, a navedene su niže:

- *robodk* – distribuirana ulazna točka za *Pythonov* API. Zajednički „roditelj" za sve potpakete i module
    - *robolink* – podmodul - sučelje između *RoboDK* i *Python*a. Bilo koji objekt unutar *RoboDK* stabla može se dohvatiti i predstavljen je kao Item (objekt)
    - *robomath* – podmodul - robotskih alata za *Python* koji omogućuje rad s transformacijama položaja i dobivanja Eulerovih kutova za različite proizvođače robota. Svi post procesori ovise o ovom modulu
- *cv2 – OpenCV* knjižnice za manipulaciju i obradu slika
- *mediapipe* – knjižnice za prepoznavanje i praćenje ruke i gesti ruke
- *numpy* – matematičke knjižnice za obradu matrica i nizova
- *pyrealsense2* – knjižnica za korištenje Intel RealSense kamere
- *time* – knjižnica za korištenje time funkcija za mjerenje stvarnog vremena, odnosno pauziranja
- *shutil* – modul koji omogućuje manipulaciju .txt podataka i skupine podataka poput kopiranja, dodavanja teksta, brisanja i slično

5.3. Korištene konstante

U aplikaciji se koristi veći broj konstanti, te su one navedene na početku kako bi se olakšale izmjene programa u svrhu podešavanja.

- MAX_NUM_HANDS – je opcija koja proizlazi iz *MediaPipe* knjižnice, a omogućuje detekciju jedne ili dvije ruke u isto vrijeme
- MIN_DETECTION_CONFIDENCE - (0.0 do 1.0) – minimalna vrijednost koja se uspoređuje s modelom da bi detekcija bila uspješna. Standardno 0.5
- MIN_TRACKING_CONFIDENCE - (0.0 do 1.0) – minimalna vrijednost koja se uspoređuje s modelom da bi praćenje dlana bilo uspješno. Standardno 0.5
- RESOLUTION_X i Y – rezolucija ekrana. Standardno 640 x 480 piksela, a kamera podržava rad s 1280 x 720 piksela

- ROBOT_CELL_WIDTH (HEIGHT, DEPTH) – (x, y, z) ograničenja detekcije ruke u mm. Koristi se za ograničavanje detekcije, odnosno pomicanje robotske ruke u kartezijevom sustavu.
- num_target – početni broj '*Targeta*', odnosno mete koju definira korisnik u *RoboDK*
- POINT_THRESHOLD – granična vrijednost između trenutne i iduće koordinate da bi došlo do promjene koordinate – odnosno pomaka.

5.4. Inicijalizacija programa

U programu se koriste različiti alati koje je potrebno na početku inicijalizirati, odnosno pravilno podesiti za dobar rad programa.

5.4.1. Inicijalizacija RoboDK

Uspostavlja se veza između samog *RoboDK* programa i *Python* skripte (naredba *robolink*), te između *RoboDK* i samog robota (naredba *connect*). Odabrani način rada je rad simulacije (RUNMODE_SIMULATE), a moguće je još odabrati i *online* ili *offline* način rada. Naredba RDK.setRunMode(1) određuje rad upravljačkog programa robota direktno sa simulacijom, a dodatne opcije se mogu provjeriti na *RoboDK Python* API stranici.

```
RDK = Robolink()                    # Uspostavljanje veze sa RoboDK
RDK.Connect()                       # Povezivanje robota i RoboDK
RDK.setRunMode(RUNMODE_SIMULATE)    # Uspostavljanje veze između robota
                                    # i simulatora
RDK.setRunMode(1)                   # Korištenje upravljačkog programa
                                    # robota zajedno sa simulatorom
```

5.4.2. Dodavanje RoboDK datoteke u kojoj se nalazi radni robot

Postavlja se poveznica na mjesto gdje je spremljena *RoboDK* radna stanica, te se definira robot s kojim se upravlja unutar same radne stanice tako da se definira kao objekt (*Item*) koji onda *Python* API dohvaća.

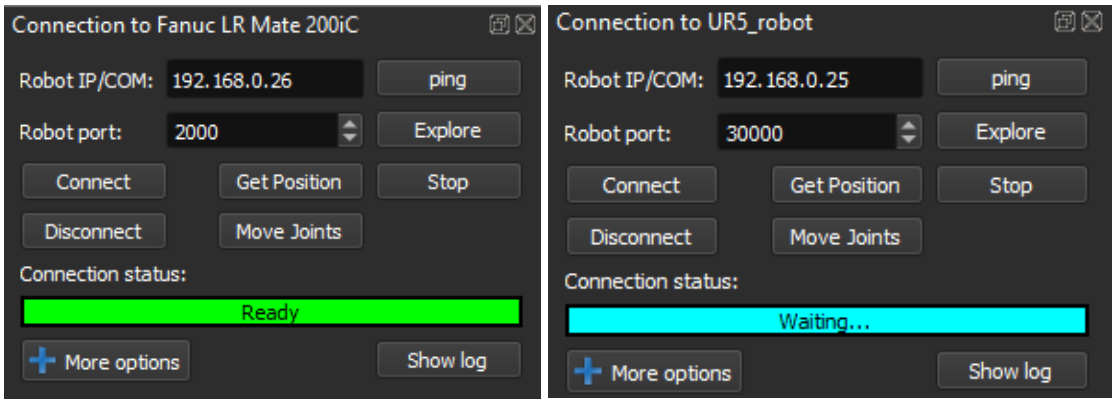
```
# poveznica na .rdk datoteku
RDK.AddFile("C:\\Users\\Zmaj\\Desktop\\Fanuc_LR_Mate_200ic.rdk")
robot = RDK.Item('Fanuc LR Mate 200iC')  # Dohvaćanje robota po imenu
robot.setSpeed(1)                        # Zadavanje brzine robota [mm/s]
robot.setJoints([0, 0, 0, 0, 0, 0])      # Postavljanje svih osi robota u 0
robot.Connect('192.168.0.26')            # Povezivanje RoboDK i robota
                                         # preko IP adrese
```

Podešavaju se i parametri poput brzine (moguće je podesiti brzinu linearnih i krivuljnih gibanja robota), početnih kuteva zglobova robota, te se povezuje s robotom preko IP adrese robota. Za upotrebu drugog robota, odnosno radne stanice potrebno je izmijeniti ime radne stanice, putanju gdje je spremljena radna stanica, ime robota i IP adresu robota.

5.4.3. Provjera povezanosti RoboDK i robota

Ispisivanje u terminal je li robot povezan s *RoboDK*, te pauza od 20 sekundi za ručno povezivanje unutar samog *RoboDK* po potrebi.

```
print(robot.ConnectedState())
time.sleep(20)
print(robot.ConnectedState())
```



Slika 11. Prikaz uspješno povezanog robota i *RoboDK* (lijevo) i čekanja na povezivanje (desno) (slika autora)

5.4.4. Pomicanje robota u početni referentni položaj

'Target_reference' je meta unutar *RoboDK* programa koja se definira na početku kao početni položaj robotske ruke za daljnje sigurno izvođenje programa, te se dohvaća poza robota nakon referenciranja, odnosno joint (krivuljnog) pomaka.

```
target = RDK.Item('Target_reference')  # dohvaćanje Target_reference mete
robot.MoveJ(target)                    # Joint pomak robota u metu
pose_ref = robot.Pose()                # dohvaćanje trenutnog položaja robota
```

5.4.5. Inicijalizacije RealSense poveznice s kamerom

Pokretanje poveznice s *RealSense* kamerom, te konfiguriranje. Dodatno konfiguriranje boje u bgr8 formatu s 30 hz osvježavanjem slike, te dodatno konfiguriranje dubine u format z16 s 30 hz osvježavanjem slike. Moguće je mijenjati mnoštvo parametara kamere za prilagođeni profil rada, međutim ovdje će se koristiti već gotovi profili. Definira se način rada kamere za visoku preciznost (engl. *High Accuracy*), jer se pokazao kao najbolje rješenje za ovu aplikaciju.

```
# Inicijalizacija i konfiguriranje RealSense kamere
pipeline = rs.pipeline()
config = rs.config()
config.enable_stream(rs.stream.color, RESOLUTION_X, RESOLUTION_Y,
rs.format.bgr8, 30)
config.enable_stream(rs.stream.depth, RESOLUTION_X, RESOLUTION_Y,
rs.format.z16, 30)
pipeline.start(config)


depth_sensor = pipeline_profile.get_device().first_depth_sensor()
```

```
# Definiranje postavki RealSense kamere na način
# prijenosa visoke preciznosti ("High Accuracy")
preset_range = depth_sensor.get_option_range(rs.option.visual_preset)
print(preset_range)
for i in range(int(preset_range.max)):
    visual_preset =
depth_sensor.get_option_value_description(rs.option.visual_preset, i)
    print(visual_preset)
    if visual_preset == 'High Accuracy':
        depth_sensor.set_option(rs.option.visual_preset, i)
```

5.4.6. Inicijalizacija Mediapipe hands

Definira se mp_drawing za ocrtavanje orijentira ruke i njihovo povezivanje. Definira se i modul za praćenje ruku – povezan je s konstantama na početku programa. Opcija static_image_mode je stavljena kao *False* jer se radi o snimci, a ne radu sa slikama.

```
# modul za anotacije i vizualizacije na slikama
mp_drawing = mp.solutions.drawing_utils
# Hands API - modul za prepoznavanje i praćenje ruku
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(
    static_image_mode=False,
    max_num_hands=MAX_NUM_HANDS,
    min_detection_confidence=MIN_DETECTION_CONFIDENCE,
    min_tracking_confidence=MIN_TRACKING_CONFIDENCE,
)
```

5.4.7. Inicijalizacija aligna, GestureClassifiera

Definiraju se opcije za korištenje RGB modula kamere, te povezivanje s tim modulom. Također se inicijalizira prepoznavanje geste i deklarira prazan string za spremanje prepoznatih gesti.

```
align_to = rs.stream.color    # povezivanje aligna (poravnavanje slike) sa
RealSense kamerom
align = rs.align(align_to)

GC = GestureClassifier()      # inicijalizacija gesture classifier-a
temp_gest = ''          # prazan string za spremanje prepoznatih gesti
```

5.4.8. Streaming Moving Average Filter

Kao što je navedeno u prethodnome poglavlju, najbolji filter za ovu primjenu je pomični prosjek čiji su parametri 10, odnosno uzima u obzir skup od 10 prethodnih točaka. Stvaranje klase filtera pomičnog prosjeka, te definiranje broja točaka koje se uzimaju u obzir.

```
smax = StreamingMovingAverage(10)
smay = StreamingMovingAverage(10)
smaz = StreamingMovingAverage(10)
```

5.4.9. Funkcija clamp

Prilokom rada aplikacije, koordinata zapešća ruke (*wrist* – 0) bi mogle izaći iz vidnog polja kamere, te bi došlo do greške u programu. Kako bi se to izbjeglo koristi se funkcija clamp_x i clamp_y koja vraća najmanju (1), odnosno najveću (rezolucija manje jedan piksel) vrijednost koordinate zapešća, dok se vidi ili je prepoznata ruka na slici.

```
def clamp_x(n):
    return max(min(n, RESOLUTION_X - 1), 1)


def clamp_y(n):
    return max(min(n, RESOLUTION_Y - 1), 1)
```

5.5. Glavna funkcija programa

5.5.1. Dobivanje slike od kamere i manipulacija slike

Na početku glavne funkcije se dohvaćaju globalne varijable temp_gest, num_target i last_point koje služe za spremanje prepoznate geste, broja trenutne mete koju kreira korisnik, odnosno vrijednosti posljednje koordinate za usporedbu..

Sljedeći korak je dohvaćanje slika iz videa, odnosno karakteristika po boji i dubini koje nam nudi RealSense kamera. Slijedi daljnja manipulacija sa slikama na način da se poravnaju slike, odnosno kalibriraju za prepoznavanje dubine. Slika se zatim sprema u niz (arrey) kako bi se njome moglo manipulirati u *MediaPipeu*. Zrcaljenje slike je također napravljeno po x osi, odnosno horizontalno kako bi korištenje aplikacije bilo lakše i intuitivnije za korisnika. Na kraju je pretvorba slike iz BGR – koji koristi *OpenCV* u RGB – koji koristi *MediaPipe*.

```
def capture_from_cam(program):      # početak funkcije i uvođenje
    global temp_gest                # globalnih varijabli
    global num_target
    global last_point

    # Dohvaćanje slike kamere i priprema RGB i depth značajki slike
    frames = pipeline.wait_for_frames()
    color_frame = frames.get_color_frame()
    depth_frame = frames.get_depth_frame()

    # Poravnavanje RGB i depth značajki slike
    aligned_frames = align.process(frames)
    aligned_depth_frame = aligned_frames.get_depth_frame()
    depth_intrin =
aligned_depth_frame.profile.as_video_stream_profile().intrinsics

    # Pretvorba slike u niz - za rad sa MediaPipeom
    image = np.asanyarray(color_frame.get_data())

    image = cv2.flip(image, 1)    # Zrcaljenje slike

    # Pretvorba slike iz BGR u RGB
    results = hands.process(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
```

5.5.2. Prepoznavanje geste ruke

Iz pripremljene slike sada je jednostavno provući je kroz model za prepoznavanje geste.

Zbog samog modela koji prepoznaje geste, u testiranju je došlo do problema između prepoznavanja 'OK' i 'Open' geste. S obzirom da se 'OK' gesta ne koristi, jednostavno se izjednačila sa gestom 'Open' i na taj način je riješen problem izmjeničnog prepoznavanja gesti.

Tu je i dodana iznimka da ako nije prepoznat depth_frame ili color_frame, da se program nastavi, odnosno da se ne prekine cijeli rad aplikacije.

```
# Prepoznavanje geste
gest = GC.run_recognition(results, image.shape[1], image.shape[0])
# Ako je prepoznata gesta 'OK'
# Izjednači je sa gestom 'Open'
gest = 'Open' if gest == 'OK' else gest

# Nastavi program dok se depth i RGB značajke ne prepoznaju
if not depth_frame or not color_frame:
    return None
```

5.5.3. Početak glavne petlje

Slijedi početak glavne petlje unutar funkcije gdje dolazi do crtanja orijentira ruke, te njihovo povezivanje. Tu se dohvaćaju koordinate zapešća koje prepoznaje *MediaPipe*. Petlja se pokreće u trenutku kada model prepozna ruku na slici.

```
# za prepoznatu ruku iscrtaj karakteristične točke
if results.multi_hand_landmarks:
    for hand_landmarks in results.multi_hand_landmarks:
        mp_drawing.draw_landmarks(
            image, hand_landmarks, mp_hands.HAND_CONNECTIONS)

        # Dohvaćanje koordinata zapešća (wrist)
        wrist = hand_landmarks.landmark[mp_hands.HandLandmark.WRIST]
```

5.5.4. Pretvorba koordinata ruke

Kako dolazi do greške i prekida programa u slučaju kada koordinate ruke, odnosno zapešća nisu unutar vidnog polja kamere, koristi se prije definirana funkcija clamp.

```
        # Ograničavanje koordinata na rezoluciju slike
        # Normalizacija koordinata
        wrist_x = clamp_x(int((wrist.x * image.shape[1] -
        image.shape[1]) * - 1))
        wrist_y = clamp_y(int(wrist.y * image.shape[0]))

        # Prilagodba koordinata kamere I MediaPipea
        depth_value_mm =
        aligned_depth_frame.get_distance(wrist_x, wrist_y)
        point = rs.rs2_deproject_pixel_to_point
        (depth_intrin, [wrist_x, wrist_y], depth_value_mm)
```

Tako pripremljene koordinate se dohvaćaju, te se koriste za prepoznavanje dubinske koordinate. Točke wrist_x i wrist_y sada su u pikselima, te se konvertiraju u koordinate kako bi odgovarale dubinskoj točci depth_value_mm. Potrebno je i napraviti rotaciju koordinatnog sustava ako bi pokreti ruke bili pravilno orijentirani prema stvarnim koordinatama. Također se za dubinski koordinatu oduzima vrijednost 2 (vrijednost u metrima), kako bi se dobio odmak od kamere i bolji radni prostor. Naravno, ta vrijednost se može potencijalno i prilagoditi. Koordinate su

trenutno u metrima. Nakon navedenih manipulacija, koordinate se spremaju u listu, te se ispisuju u mm zaokružene na 2 decimalna mjesta.

```
# Rotacija koordinatnog sustava (vrijednosti u m)
point[1] = -point[1]
point[2] = -(point[2] - 2)

# Spremanje koordinata u listu
point = [smax.process(point[0]),
         smay.process(point[1]),
         smaz.process(point[2])]

# Ispisivanje koordinata
print(f'X: {round(point[0], 2)},
        Y: {round(point[1], 2)},
        Z: {round(point[2], 2)}')
```

5.5.5. Provjera pomaka koordinata ruke

Zbog problema titranja koordinata ruke, a koje se prenosi u simulaciju te na kraju na gibanje robota, postavljena je granična vrijednost točke (POINT_THRESHOLD), odnosno koordinate ispod koje je očitana koordinata jednaka prošloj koordinati. Drugim riječima, ako je pomak manji od granične vrijednosti, do pomaka nije došlo, odnosno nije došlo do promjene koordinate. Ova opcija u kombinaciji s filterom pomičnog prosjeka daje najbolje rezultate za mirno gibanje robota, ali ipak dolazi do diskretizacije sustava. Provjera se jednostavno izvršava tako da ako varijabla last_point nije prazna, provjerava se apsolutna vrijednost između koordinata liste point i last_point, te uspoređuje s graničnom vrijednosti točke.

```
if len(last_point) != 0:
      if all(abs(a - b) < POINT_THRESHOLD for a, b in
      zip(last_point, point)):
            point = last_point

last_point = point
```

5.5.6. Ispisivanje geste i ograničenja radnog prostora

Ovdje je dio koda koja ispisuje prepoznatu gestu u gornjem lijevom kutu, te se ispisuje na dnu radnog prozora ograničenje radnog prostora koja su definirana u konstantama na početku programa. Za ispis se koristi naredba cv2.putText koja definira položaj, boju, tip fonta, oblik, skaliranje i debljinu fonta.

```
# Ispisivanje prepoznate geste
cv2.putText(image,
            text=f"{gest}",          # tekst ispisa
            org=(50, 50),            # koordinate ispisa
            fontFace=cv2.FONT_HERSHEY_SIMPLEX,  # font
            fontScale=2,             # skaliranje fonta
            color=(0, 255, 0),       # boja fonta
            thickness=3)             # debljina fonta
```

```
# Ispisivanje veličine radnog prostora
cv2.putText (image,
             text=f"W: +/- {ROBO_CELL_WIDTH / 2}
                    H: {ROBO_CELL_HEIGHT},
                    D: +/- {ROBO_CELL_DEPTH / 2}",
             org=(50, 700),
             fontFace=cv2.FONT_HERSHEY_SIMPLEX,
             fontScale=1,
             color=(0, 255, 0),
             thickness=3)
```

5.5.7. Provjera koordinata

Važan dio koda je provjera jesu li koordinate zapešća unutar ranije definiranog radnog prostora. Koordinate se provjeravaju posebno za x, y i z, odnosno za širinu, visinu i dubinu. Pomicanje robota se vrši samo unutar tih koordinata, te svakako predstavljaju i sigurnosni aspekt korištenja aplikacije. Za prepoznatu točku zapešća unutar radnog prostora, ispisuju se koordinate na sliku u plavoj boji. U slučaju da uvjeti radnog prostota nisu zadovoljeni, ispisuje se poruka da je korisnik van radnog prostora u terminal (*„Out of range“*), te se ispisuje koordinate zapešća u crvenoj boji na sliku.

```
# Provjera jesu li koordinate unutar radnog prostora
if ((-ROBO_CELL_WIDTH / 2) <= round(point[0] * 1000, 2) <=
(ROBO_CELL_WIDTH / 2)
and
-300 <= round(point[1] * 1000, 2) <= ROBO_CELL_HEIGHT
and
(-ROBO_CELL_DEPTH / 2) < round(point[2] * 1000, 2) <
(ROBO_CELL_DEPTH / 2)):

# Ispis da su koordinate unutar radnog prostora
    print("Inside Work Range")


    # Ispisivanje koordinata ruke na sliku
    cv2.putText(image,
                text=f"{round(point[0], 2)},"
                     f"{round(point[1], 2)},"
                     f"{round(point[2], 2)}",
                org=(-1 * (wrist_x - image.shape[1]),
                     wrist_y + 20),
                fontFace=cv2.FONT_HERSHEY_SIMPLEX,
                fontScale=1,
                color=(245, 0, 0),
                thickness=3)

...

else:
    print("Out of range")
    # Ispis da su koordinate van dosega robota
    # Ispis crvenih koordinata ruke na sliku
    cv2.putText(image, ...
```

5.5.8. Pomicanje robota u koordinate ruke

Za situaciju kada su koordinate ruke, odnosno zapešća (0 – wrist) unutar radnog prostora, šalje se naredba robotu approach koja pomiče robota u tu koordinatu. Pomak koji se šalje zaokružen je na dvije decimale, sadrži tri koordinate, te tri rotacije oko svake koordinate kako se ne bi pojavila greška u gibanju unutar *RoboDK*, prilikom promjene minimalnog i maksimalnog dozvoljenog kuta zakreta ruke. Kako z os izlazi iz kamere, a u simulaciji je z os vertikalna, tako su i zamijenjene točke y i z, odnosno point[1] i point[2].

```
# Pomicanje robota prema koordinatama
approach = target.Pose() * transl(round(point[0], 2),
                                  round(point[2], 2),
                                -(round(point[1], 2)))
                                * rotx(0.1)
                                * roty(0.1)
                                * rotz(0.1)
```

5.5.9. Provjera je li robot zauzet gibanjem

Naredbom robot.Busy() provjerava se je li robot trenutno zauzet, te ako nije izvršava se pomak. Zbog prije napravljenih ograničenja programa, napravljena je iznimka za grešku *„Target not reachable“* kako se ne bi prekidao program svaki puta kada bi meta (*Target*) bila van dosega robota, odnosno kada bi ruka korisnika bila van radnog prostora robota.

```
# Provjera jel robot u pomaku ili ne
# Ako robot nije u pomaku, izvrši pomak
    if robot.Busy() == 0:

      ...

        try:
            robot.MoveJ(approach, blocking=False)

        # Zanemari grešku ako robot ne može dosegnuti metu
        # i ispiši 'Target not reachable'
        # Ako robot nije u pomaku, izvrši pomak
        try:
            print(approach)
            robot.MoveJ(approach, blocking=False)
        except TargetReachError:
            print('Target not reachable')
```

5.5.10. Opcije otvaranja i zatvaranja hvataljke, stvaranja mete

Unutar petlje koja provjerava je li robot trenutno zauzet, uspoređuje se prepoznata gesta s otvorenom i zatvorenom šakom i tako se šalje signal robotu da otvori i zatvori hvataljku. Ovdje je potrebno prilagoditi kod ovisno kojim robotom upravljamo. Za UR5 koriste se naredbe za digitalni izlaz (engl. *Digital Output*), odnosno naredba robot.setDO(0, 0) za otvaranje i DO(0, 1) za zatvaranje hvataljke. Upravljanje hvataljkom Fanuc LR Mate 200iC izvršava se tako da je potrebno pokrenuti program unutar pripadajućeg privjeska za učenje

robot.RunInstruction('GR_OP', robolink.INSTRUCTION_CALL_PROGRAM) za otvaranje, odnosno promijeniti u 'GR_CL' za zatvaranje. Pritom se sprema naredba u korisnički program. Ispruženi kažiprst gesta je koja stvara metu na trenutnoj poziciji ruke. Pritom se sprema naredba u korisnički program za dodavanje mete i pomicanje u tu metu. Mete se nazivaju *'User_Target_1'*, te se povećavaju za po jedan kako su stvorene.

```
# Provjera prepoznate geste
# Open/Close - Otvorena/Zatvorena hvataljka
# Pointer - stvaranje mete u RoboDK
if gest != temp_gest:
    temp_gest = gest
    if gest == 'Open':
        # robot.setDO(0, 0)
        robot.RunInstruction(
        'GR_OP',
        robolink.INSTRUCTION_CALL_PROGRAM)
        program.write(
        "robot.RunInstruction('GR_OP',
        robolink.INSTRUCTION_CALL_PROGRAM)" + "\n\n")
    elif gest == 'Close':
        # robot.setDO(0, 1)
        robot.RunInstruction(
        'GR_CL', robolink.INSTRUCTION_CALL_PROGRAM)
        program.write(
        "robot.RunInstruction('GR_CL',
        robolink.INSTRUCTION_CALL_PROGRAM)" + "\n\n")
    elif gest == 'Pointer':
        ti = RDK.AddTarget(f'User_Target_{num_target}', 1)
        pose_tar = robot.Pose()
        ti.setPose(pose_tar)
        ti.setAsCartesianTarget()
        time.sleep(1)
        program.write(
        f"ti =RDK.Item(f'User_Target_{num_target}')" + "\n"
        "robot.MoveJ(ti)" + "\n\n")
        num_target += 1
```

5.6. Stvaranje korisničkog programa

Predložak (Template.txt) je dio koda s kojim započinje svaki program koji stvara korisnik, a sadrži sve inicijalne radnje koje su potrebne za pokretanje programa. Naredbe se većinom preklapaju s naredbama u početku aplikacije. Kraj predloška je linija koja govori kao slijedi dio programa koji je kreirao korisnik. Naravno, predložak mora biti prilagođen kao i originalni program za specifičnog robota, ograničenja radnog prostora i *RoboDK* radnu stanicu.

```
from robodk.robolink import *
from robodk import robolink

# Pokretanje RoboDK API:
RDK = Robolink()
RDK.Connect()
RDK.setRunMode(RUNMODE_SIMULATE)
RDK.setRunMode(1)
```

```
RDK.AddFile("User_program.rdk")
robot = RDK.Item('Fanuc LR Mate 200iC')
robot.setSpeed(10, 10)
robot.Connect('192.168.0.26')

print(robot.ConnectedState())
time.sleep(20)
print(robot.ConnectedState())

target = RDK.Item('Target_reference')
robot.MoveJ(target)
pose_ref = robot.Pose()

# **********User created program**********
```

U ovome djelu koda se odvija kopiranje predloška i provjera je li novi program postoji već pod tim imenom. U slučaju da postoji, dodaje mu se na kraj imena broj za jedan veći od prethodnog. Korištene naredbe za manipulaciju dokumentima su shutil naredbe, koje su primjerene za datu primjenu.

```
src = "Template.txt"
dst = "Robot_program.py"

# Provjera jel postoji već program sa tim imenom
# Ako postoji, sprema se pod prvim većim brojem
i = 1
while os.path.exists(dst):
    dst = f"Robot_program{i}.py"
    i += 1

shutil.copy(src, dst)    # Izrada kopije predloška ("Template")i
                         # Spremanje u "Robot_program.py"
file = open(dst, "a")    # otvaranje nove datoteke i omogućeno dodavanje na
                         # kraj (append)
```

5.7. Kraj programa

Kraj programa definira boolean varijablu base_coord_captured koja se koristi za *while* petlju sve dok korisnik pritiskom na tipku 'q' ne odluči prekinuti rad aplikacije.

```
base_coord_captured = False          # Početna izjava za rad petlje programa

while not base_coord_captured:       # Rad programa u petlji
    capture_from_cam(file)           # Vraćaj vrijednosti funkcije
    if cv2.waitKey(1) & 0xFF == ord('q'): # Zatvori aplikaciju ako korisnik
        break                            # pritisne tipku 'q'

file.close()                    # Zatvori datoteku

cv2.destroyAllWindows()         # Zatvori sve prozore

pipeline.stop()                 # Zaustavi vezu sa RealSense kamerom

RDK.Save("User_program.rdk")    # Spremi RoboDK program
RDK.CloseRoboDK()               # Zatvori RoboDK program
```

Unutar petlje se vraćaju vrijednosti funkcije capture_from_cam za spremanje u datoteku. Kada se to dogodi, zatvaraju se sve datoteke, zatim se zatvaraju svi radni prozori i prekida se veza s RealSense kamerom. Na kraju se sprema trenutna *RoboDK* radna stanica i zatvara.

5.8. Pokretanje programa na FANUC LR Mate 200iC industrijskome robotu

Prije rada na robotu potrebno je instalirati upravljačke programe (engl. *driver*). Na mrežnoj stranici https://robodk.com/doc/en/Robots-Fanuc-RoboDK-driver-Fanuc.html nalaze se detaljne upute za preuzimanje, instalaciju i podešavanje upravljačkih programa.

Postupak pokretanja programa, odnosno priprema FANUC LR mate 200iC robota, za rad s RoboDK programom je sljedeći:

1. Napraviti *„Reset“* na način da se provjeri jesu li udarna tipkala na privjesku za učenje i upravljačkoj ploči podignuta. Zatim pritisnuti *„FCTN“* (*Function*) i odabrati *„Abort All“*. Nakon toga pritisnuti tipku „*Reset“* na privjesku za učenje i provjeriti jesu li sva polja zelena ili žuta.
2. Pomaknuti sklopku na privjesku za učenje iz pozicije *„ON“* u *„OFF“* poziciju.
3. Na upravljačkoj ploči okrenuti ključ u poziciju *„Auto“*.
4. Provjerili jesu li se pojavile greške ili ne
5. Odabrati program pod rednim brojem 131 – RDK_S3
6. Na upravljačkoj ploči pritisnuti tipku *„CYCLE START“*

Slika 12. Privjesak za učenje ne javlja greške i robot je spreman za rad (slika autora)

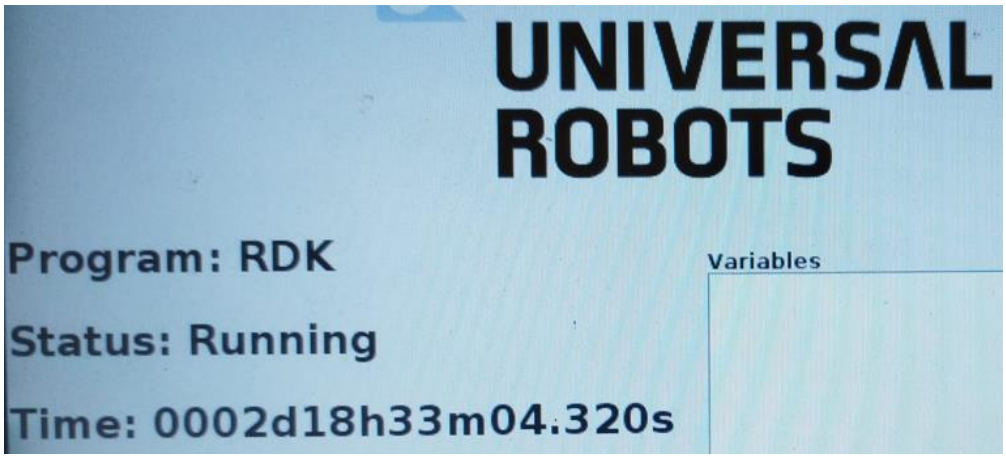
Slika 13. Privjesak za učenje javlja greški i potrebno je napraviti *„reset“* (slika autora)

Tipka *„CYCLE START“* bi trebala sada svijetliti zeleno i to je znak da je robot spreman za rad s *RoboDK*. Unutar *RoboDK* potrebno je provjeriti konekciju s robotom. U slučaju prestanka rada programa, potrebno je ponoviti postupak.

Slika 14. Upravljačka ploča robota spremnog za rad (slika autora)

5.9. Pokretanje programa na UR5 robotu

Za pokretanje programa na UR5 robotu, također je potrebno instalirati upravljačke programe (engl. *driver*). Na mrežnoj stranici https://robodk.com/doc/en/Robots-Universal-Robots.html nalaze se detaljne upute za preuzimanje, instalaciju i podešavanje upravljačkih programa. Povezuje se robot s lokalnom mrežom putem mrežnog kabela. Pokreće se upravljački program, odnosno skripta na privjesku za učenje, za komunikaciju između *RoboDK* i robota. Udarno tipkalo ne smije biti uključeno. Kao i za Fanuc, i ovdje se provjerava konekcija unutar *RoboDK*.



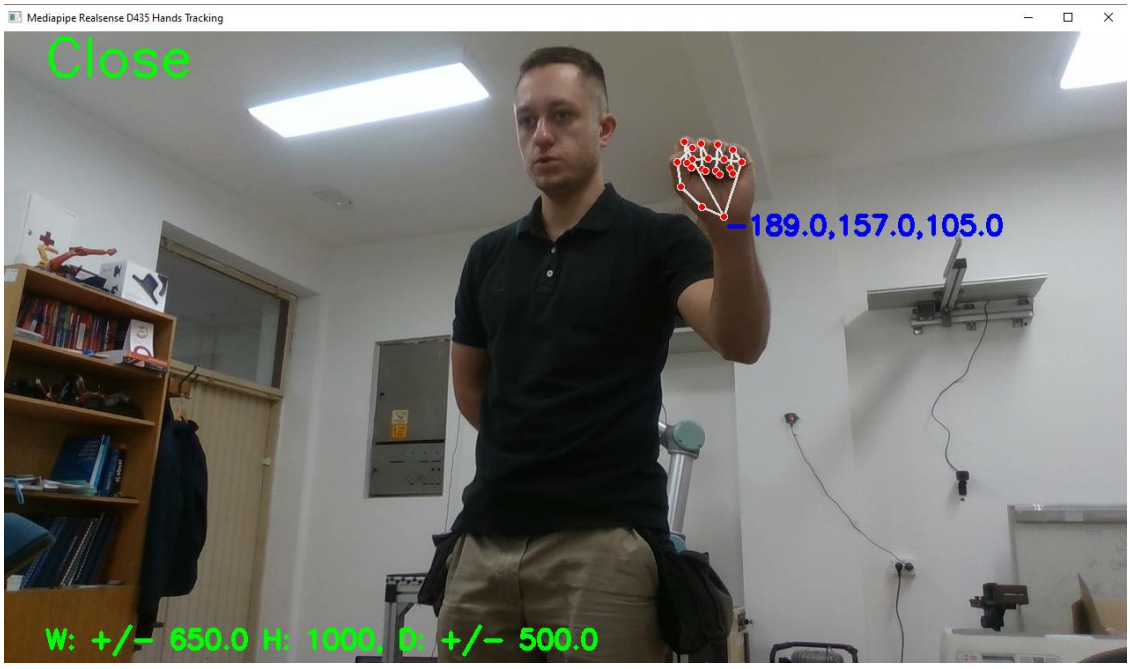
Slika 15. Upravljački program instaliran na privjesku za učenje robota UR5 (slika autora)

5.10. Korisničko sučelje i korištenje aplikacije

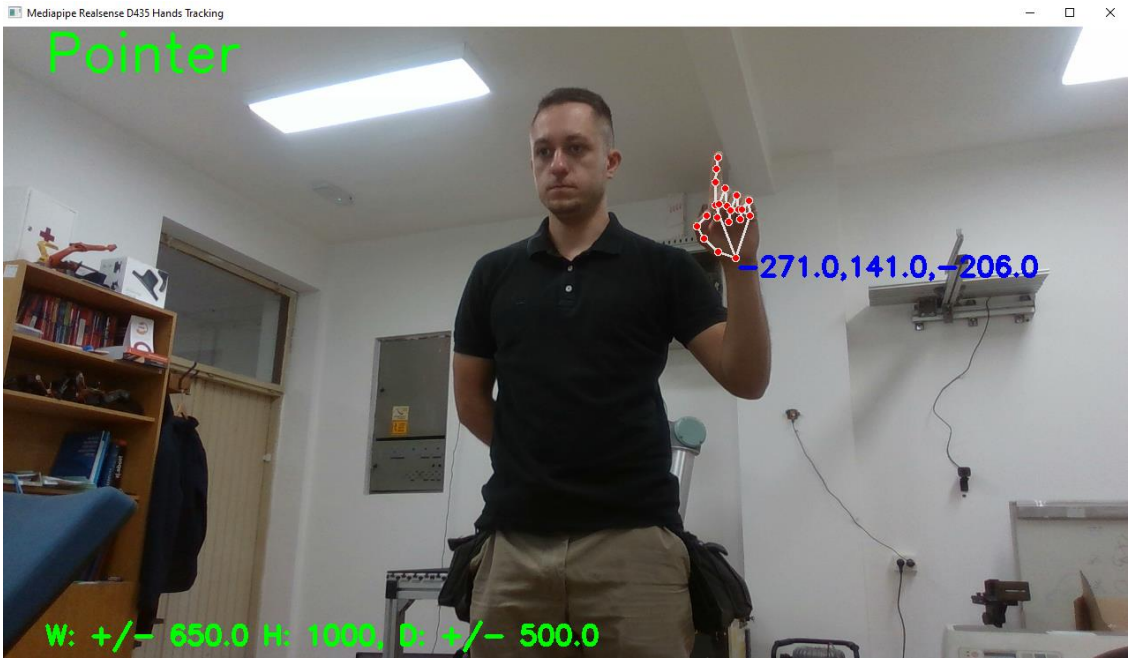
Prije početka rada potrebno je definirati *RoboDK* radnu stanicu s robotom i referentnom metom (*Target_reference*) u koju će se pomaknuti robot u inicijalnom dijelu. Ovisno o robotu, potrebno je provesti radnje za povezivanje robota i *RoboDK*. U točkama 4.8. i 4.9. opisan je postupak za FANUC LR Mate 200ic i UR5 robot. Korisnik mora imati priključenu kameru *Intel RealSense D435*, te osiguran nesmetan radni prostor oko sebe. Na početku programa potrebno je izvršiti navedene izmjene koje se tiču naziva *RoboDK* radne stanice, naziva robota, IP adrese robota, te podešavanje svih navedenih konstanti koje definiraju rad programa.

Aplikacije se koristi intuitivno pomicanjem ruke, te prati što se događa na ekranu kamere i/ili simulacije unutar *RoboDK*. Crvena boja koordinata upućuje da je zapešće korisnika van radnog prostora. U gornjem lijevom kutu ispisuje se prepoznata gesta – *Open, Close* i *Pointer*. Gesta *Open* otvara prihvatnicu robota i jednaka je otvorenom dlanu (slika 17.). Gesta *Close* zatvara prihvatnicu robota i jednaka je zatvorenoj šaci (slika 18.).

Slika 16. Prikaz korisničkog sučelja – prepoznata gesta *Open* (otvorena hvataljka) i koordinate u crvenoj boji koje znače da je koordinata van definiranog radnog prostora (slika autora)



Slika 17. Prikaz korisničkog sučelja – prepoznata gesta *Close* (zatvorena hvataljka) i koordinate u plavoj boji koje znače da je koordinata unutar definiranog radnog prostora (slika autora)



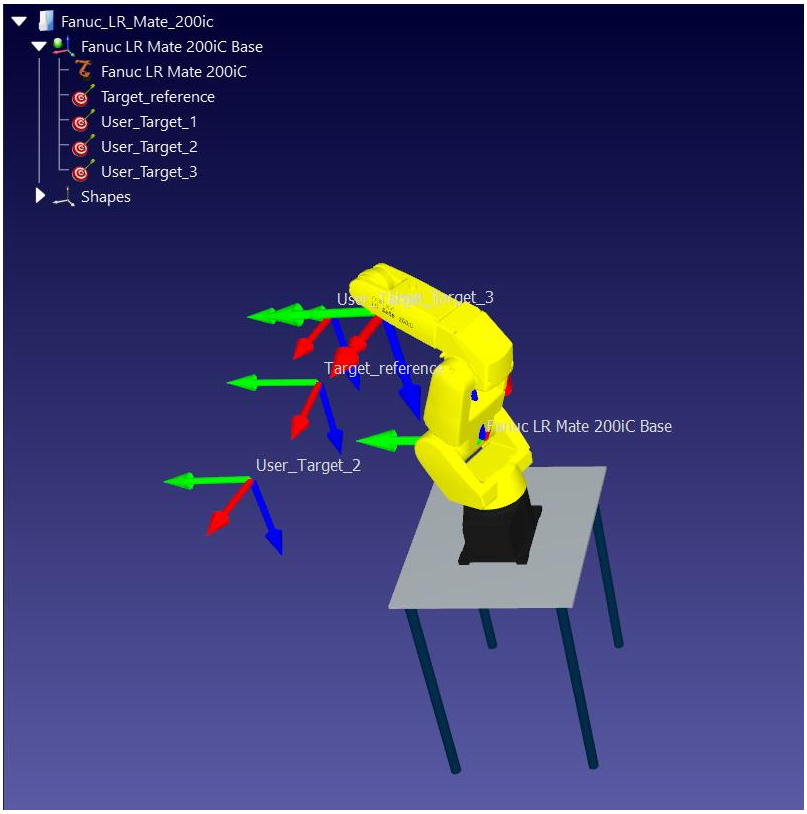
Slika 18. Prikaz korisničkog sučelja – prepoznata gesta *Pointer* (stvaranje mete) i koordinate u plavoj boji koje znače da je koordinata unutar definiranog radnog prostora (slika autora)

Na kraju gesta *Pointer* stvara metu (target) na trenutnim koordinatama (lika 19.) pod nazivom *„User_Target_1“*. Imena novih meta se povećavaju za 1 za svaku novu metu. Kada korisnik želi prekinuti rad aplikacije, mora pritisnuti tipku 'q'. Sve prepoznate geste se spremaju u korisnički program koji se kasnije može samostalno pokrenuti.

Ovdje prikazani kod automatski je generiran gestama korisnika, a sastoji se od tri mete, krivuljnih pomaka u te mete, te 3 niza otvaranja i zatvaranja robotske hvataljke.

```
# **********User created program**********
robot.RunInstruction('GR_OP', robolink.INSTRUCTION_CALL_PROGRAM)

robot.RunInstruction('GR_CL', robolink.INSTRUCTION_CALL_PROGRAM)

ti = RDK.Item(f'User_Target_1')
robot.MoveJ(ti)

robot.RunInstruction('GR_OP', robolink.INSTRUCTION_CALL_PROGRAM)

ti = RDK.Item(f'User_Target_2')
robot.MoveJ(ti)

robot.RunInstruction('GR_OP', robolink.INSTRUCTION_CALL_PROGRAM)

robot.RunInstruction('GR_CL', robolink.INSTRUCTION_CALL_PROGRAM)

robot.RunInstruction('GR_OP', robolink.INSTRUCTION_CALL_PROGRAM)

robot.RunInstruction('GR_CL', robolink.INSTRUCTION_CALL_PROGRAM)

ti = RDK.Item(f'User_Target_3')
robot.MoveJ(ti)
```

Program je samostalan, odnosno može se pokrenuti i izvesti bez intervencije korisnika, što otvara mogućnosti brzog online programiranja robota za razne zadatke. Izgled *RoboDK* radne stanice prikazan je na slici 20. Početak svakog korisničkog programa sadži instrukcije potrebne za početak rada, a objašnjene su u točki 5.6.



Slika 19. Primjer *RoboDK* radne stanice koja se kreira iz korisničkog programa (slika autora)

6. IZRADA NADOGRADNJE ROBOTSKE PRIHVATNICE

Za potrebe ispitivanja rada sustava ideja je tehnologijom 3D tiskanja izraditi jednostavne robotske prste koji odgovaraju postojećoj robotskoj hvataljki. 3D tiskanje danas je jako popularna tehnologija za hobiste, ali i za sve veći broj tvrtki i sveučilišta zbog niske cijene 3D tiskača, raznovrsnosti materijala, točnosti i prihvatljivog vremena izrade.

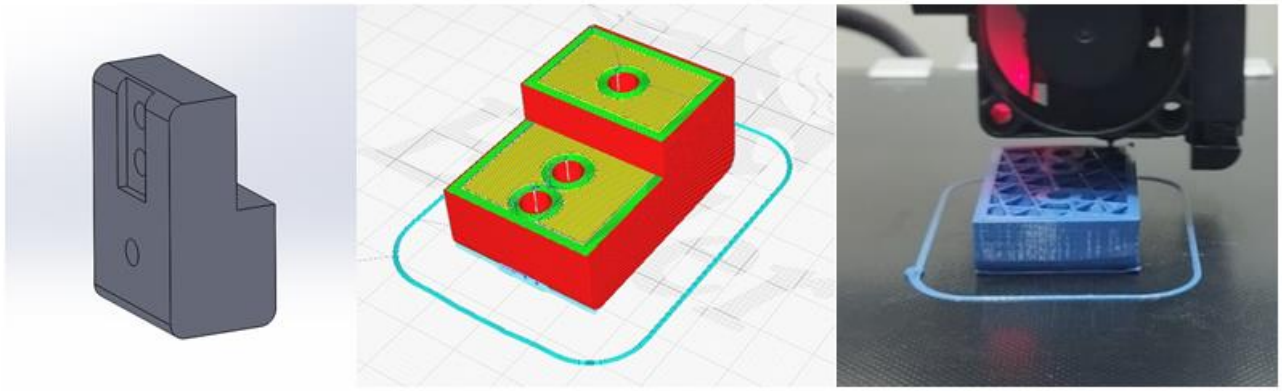
U CAD programu *SolidWorks* nacrtani su adapteri za postojeću prihvatnicu, te dvije verzije prihvatnih prstiju. Nakon toga su na modificiranom Ender 3 tiskaču izrađeni prototipovi za testiranje. Priprema za printanje napravljena je u otvorenome programu *Cura*.

6.1. Postojeća prihvatnica

Postojeća prihvatnice je MHZ2-20DN-A310 japanskoga proizvođača SMC. [22] MHZ2 serija je standardnih paralelnih hvataljki pogonjenih zrakom. Linearne vodilice omogućuju visoku krutost i preciznost. N u oznaci označava da se radi o uskoj hvataljki, čiji je hod od 17.2 mm, kada je otvorena, do 7.2 mm kada je zatvorena. Dio kratkog hoda prihvatnice pokušat će se kompenzirati izradom jednostavnih adaptivnih prstiju.

6.2. Izrada adaptera prihvatnice

Adapter je izrađen od PLA proizvođača *AzureFilm*. PLA (engl. *Polylactic acid* – Polilaktična kiselina) popularan je materijal za 3D tiskanje zbog niske temperature taljenja, visoke čvrstoće, niske termalne ekspanzije i dobrog prianjanja za podlogu. Spada u ekonomične polimere koji se proizvode iz obnovljivih izvora, te se smatra najkorištenijom bioplastikom u svijetu. [23] Dobra mehanička svojstva uz tehnologičnost, te cijena čine ovaj materijal dobrim izborom za izradu prototipova jednostavnih pozicija koje nisu mehanički znatno opterećena.



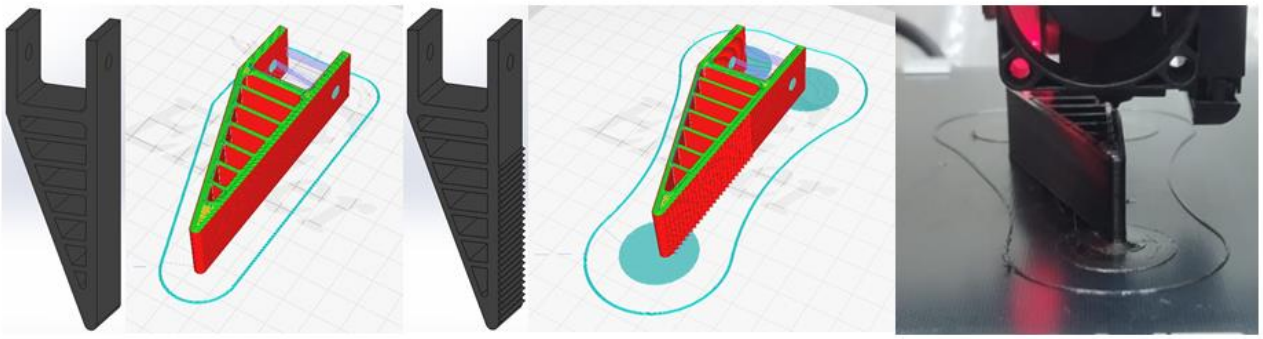
Slika 20. Model adaptera (lijevo), priprema za tiskanje (sredina) i tiskanje (desno) (slika autora)

Tablica 4. Parametri za izradu adaptera prihvatnice

Visina sloja	0.2 mm
Ispuna	20 %
Broj stijenki otiska	5

6.3. Izrada prstiju prihvatnice

Prsti hvataljke izrađeni su od materijala TPU proizvođača *AzureFilm* (*Flexible Filamet 85A*). TPU (engl. *Thermoplastic polyurethane* - Termoplastični Poliuretan) je blok kopolimer koji posjeduje svojstva plastomera i elastomera. To mu omogućuje dobra svojstva izdržljivosti, fleksibilnosti, istezljivosti i odlična vlačna čvrstoća za polimer. Kombinacija navedenih svojstava čini TPU pogodnim za široku primjenu u automobilskoj industriji, u izradi ožičenja i kablova, tekstilne obloge, brtve, potplate obuće i sl. [24] Izrada 3D tiskanjem kompleksnija je od tiskanja PLA materijala, ali dobrim parametrima i podešenim strojem mogu se dobiti kvalitetni rezultati.



Slika 21. Model i priprema za tiskanje prsta hvataljke (Lijevo verzija 1, sredina verzija 2) i tiskanje prsta hvataljke (desno) (slika autora)

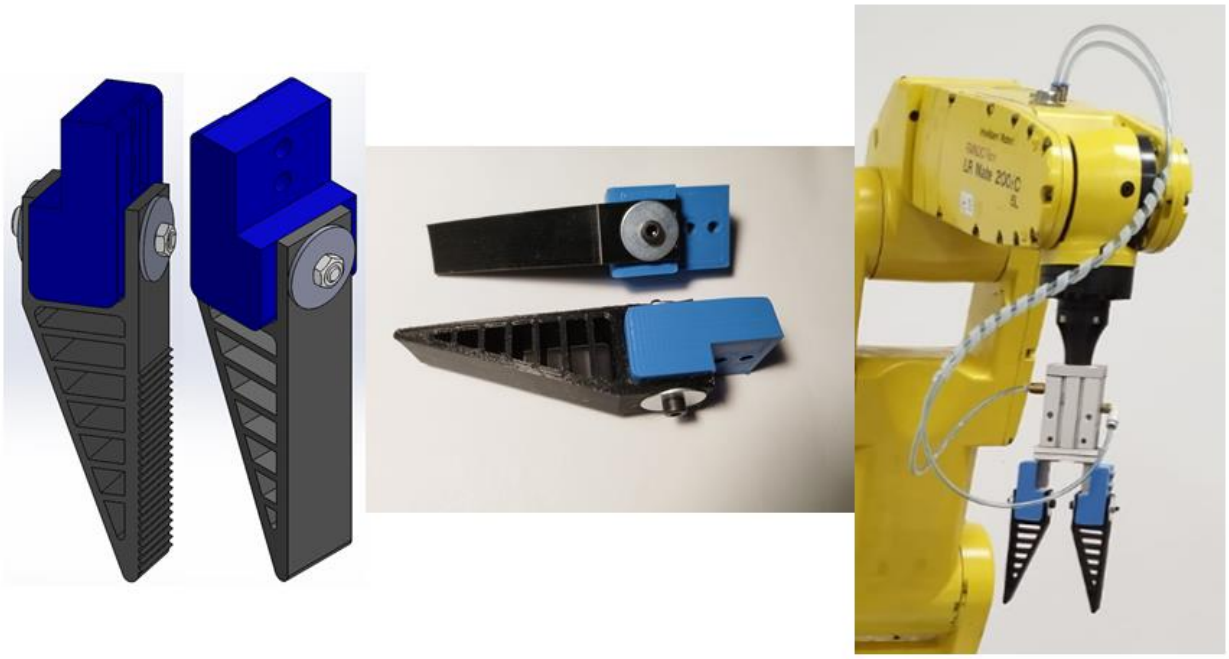
Tablica 5. Parametri za izradu prstiju hvataljke

Visina sloja	0.2 mm
Ispuna	40 %
Broj stijenki otiska	4

6.4. Izrada sklopa adaptivnih prstiju

Sklop se jednostavno montira korištenjem dviju širokih podloški M4x20, te vijka M4x35 i pripadajuće matice. Zamišljeno je da se stezanjem deformira prst, te se time ostvari potrebna kvalitetna mehanička veza cijelog sklopa. Orijentacija tiskanja je bila takva da se slojevi adaptera i prsta prihvatnice dobro nose s opterećenjima koja se pojavljuju u eksploataciji. Montaža sklopa na postojeću hvataljku ostvaruje se jednostavno preko dva para 2 x M4x20 vijaka. Isto tako je moguće mijenjati konfiguraciju prstiju tako da je ravni dio prsta s unutarnje

strane ili je pak kosina prsta s unutarnje strane. Izrađene su dvije verzije prstiju prihvatnice, deblje i tanje stijenke. Prihvatnica s tanjom stijenkom ima blaga nazubljenja na ravnoj strani da bi se smanjila mogućnost klizanja, odnosno ispadanja predmeta. Geometrija, odnosno debljina hvataljki nije podvrgnuta računima, već se smatra početnim iterativnim postupkom jer to niti nije fokus ovog rada.

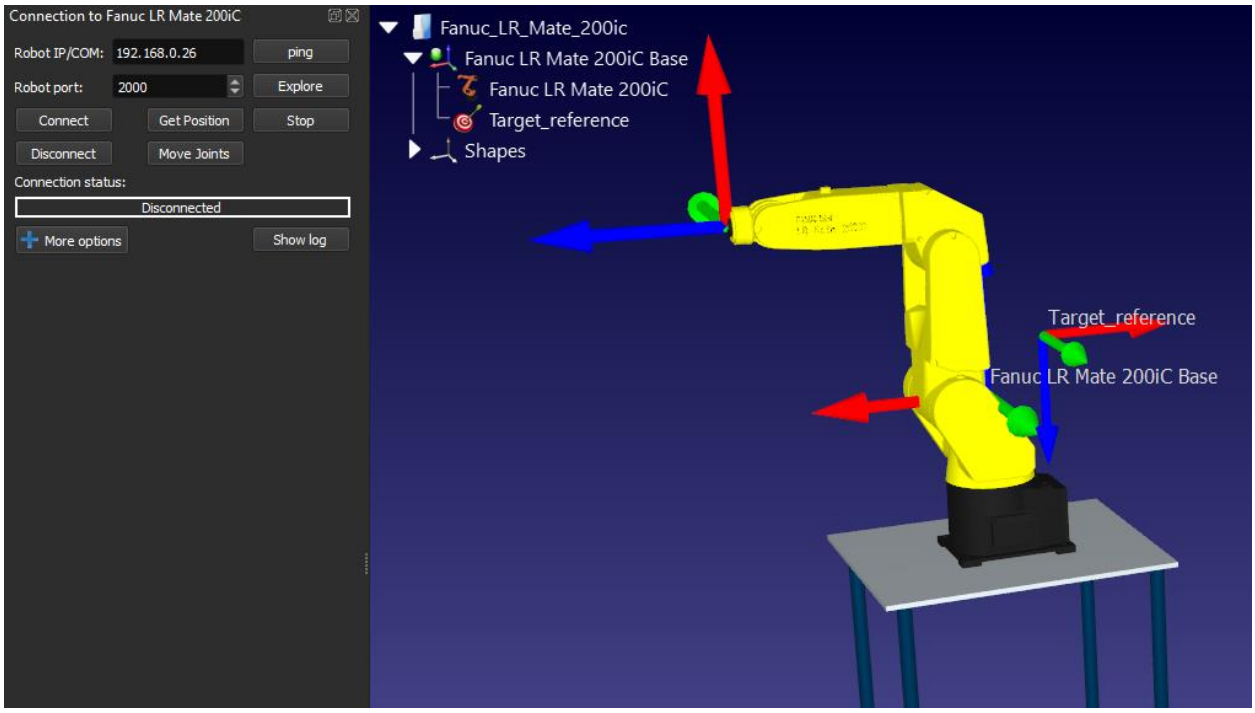


Slika 22. Model sklopa adaptivnih prstiju (lijevo), sastavljeni sklop (sredina) i montirani sklop adaptivnih prstiju na hvataljku robota (slika autora)

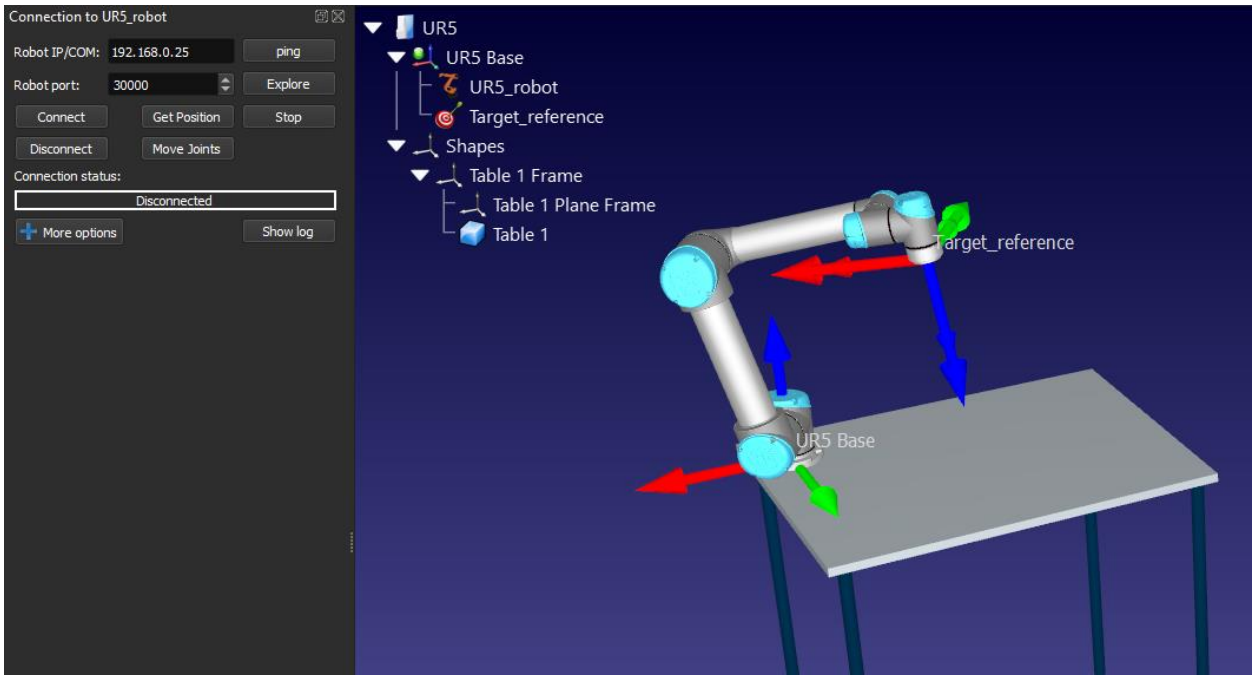
7. IZRADA RoboDK RADNE STANICE

Za potrebe izrade i evaluacije ovo rada, napravljene su dvije *RoboDK* radne stanice. U *RoboDK* programu prvo odabiremo robota iz bogate knjižnice robota koju program posjeduje. Nakon što smo unijeli robota, definiramo njegov naziv, te naziv radne stanice. Unutar programa koristi se *Add-in* (dodatak) *RoboDK Shape*. On omogućuje dodavanje raznih oblika poput kutije, pomične trake, ograde, sfere i slično, no za potrebe izrade ove radne stanice odabire se stol, po uzoru na stol unutar laboratorija u kojem se vrši eksperimentalno testiranje (slika 25.). Za objekt mogu se definirati dimenzije i boje. Nakon dodavanja stola, korištenjem koordinatnih sustava stola i robota, namješta se međusobna pozicija objekata. Potrebno je još definirati metu *„Target_reference“* u nekom sigurnom položaju u koji će se robot pomaknuti prilikom početka programa. Valja biti pažljiv na nazive programa, robota i referentne mete, kako bi se ti nazivi poklapali s onima unutar programa.

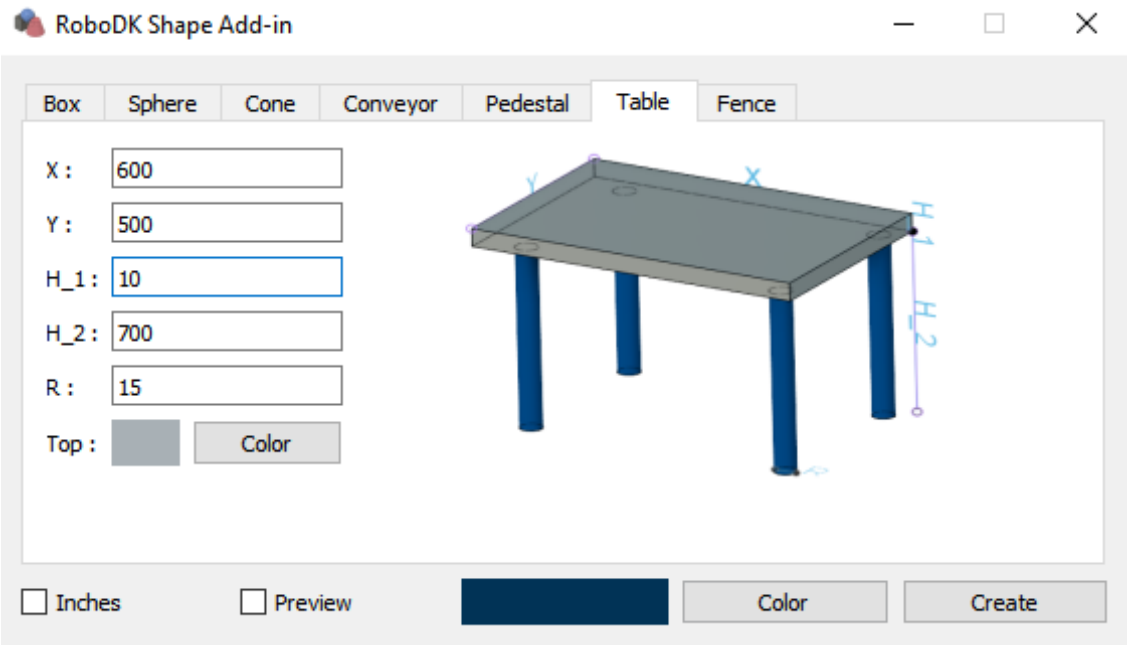
Unutar *RoboDK* moguće je dodati i hvataljke te definirati TCP (engl. *Tool Center Point* – točka središta alata). Zbog eksperimentalne prirode postava to u ovome slučaju nije provedeno, međutim u slučaju da bi se izvodila primjena u kasnijem stadiju razvoja aplikacije to bi bila jedna od mogućnosti koja bi se vjerojatno primijenila.



Slika 23. *RoboDK* radna stanica za industrijski robot Fanuc LR Mate 200ic (slika autora)



Slika 24. *RoboDK* radna stanica za industrijski robot UR5 (slika autora)



Slika 25. Definiranje objekta – stol, unutar *RoboDK* (slika autora)

8. EKSPERIMENTALNA EVALUACIJA DOBIVENOG RJEŠENJA

Rad aplikacije eksperimentalno se provjerio u laboratoriju. Postav je takav da se koristio samo jedan robot u trenutku testiranja, te da korisnik ima upravljački privjesak uz sebe kako bi mogao zaustaviti program ili je sa strane bila osoba koja je zaustavila rad programa u slučaju nužde. Prilikom testiranja UR5 kobota, korišten je stol na kojem je robot montiran. Za testiranje Fanuc LR mate 200iC robota korišten je stol UR5 robota koji je prije demontiran ili pomaknut u siguran položaj van dosega trenutno operativnog robota. Brzine testiranja LR Mate 200iC robota bile su kratkotrajno do 80 %, a za većinu testiranja do 60 %. UR5 je prilikom većine testiranja bio na 100 % brzine, jer je pokazao generalno bolju izvedbu rada aplikacije, odnosno upravljivost.

8.1. Testiranje filtera i POINT_THRESHOLD-a na rad sustava

Izvršeno je testiranje filtera na robotu, odnosno odabir najmanje vrijednosti između koordinata da bi se ostvario pomak. Testiralo se tako da je korisnik stao ispred kamere, te je pozicionirao ruku na prsa kako bi je što vise fiksirao, te stavio mogući prirodni tremor na najmanju vrijednost (slika 26.).

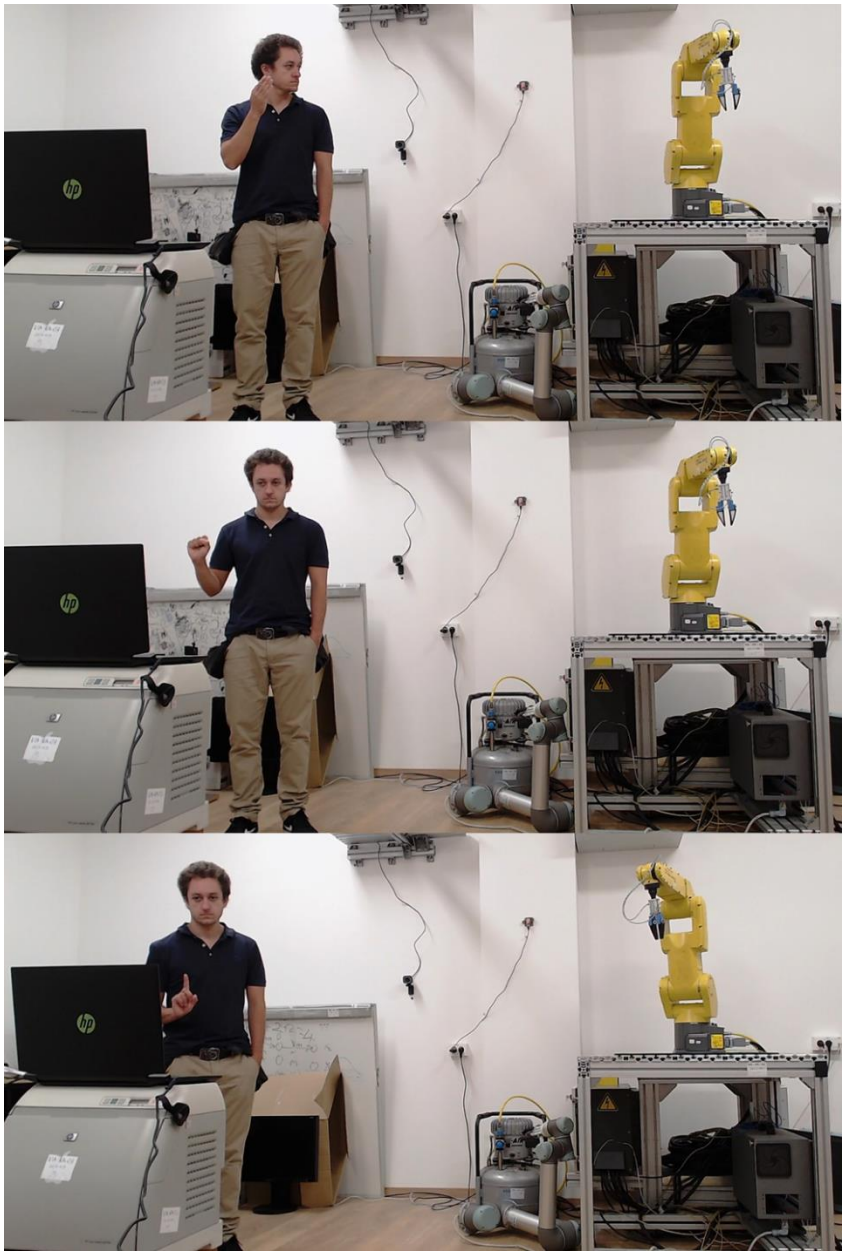
Slika 26. Korisnik miruje s fiksiranom rukom – testiranje najmanjih pokreta zbog šuma sustava (slika autora)

Prvo se isključio filter i nije postavljen POINT_THRESHOLD, te je zamijećeno 'plesanje robota'. Zatim se testiralo na isti način sa uključenim filterom pomični prosjek (engl. *Moving Average*) koji uzima 10 točaka. Očekivano se primijetilo smanjeno pomicanje robota iako ruka miruje, odnosno koordinate su se mijenjale u manjem spektru. Postavljen je

POINT_THRESHOLD na vrijednost 7, odnosno najmanji pokret ruke je trebao biti 7 mm kako bi se ostvarilo gibanje robota, aplikacija nije bilježila pomicanje ruke i robot miruje dok korisnik drži ruku mirnom. Naravno na kraju je rezultat diskretizacija sustava za 7 mm, odnosno vrijednost točke POINT_THRESHOLD.

8.2. Testiranje izrade korisničkog programa

Testirala se izrada korisničkog programa, odnosno korisnik je direktno upravljao robotom putem aplikacije, gestama ruke je stvorio nekoliko meta (*Targeta*), te je simulirao nekoliko otvaranja i zatvaranja hvataljke (slika 26.).

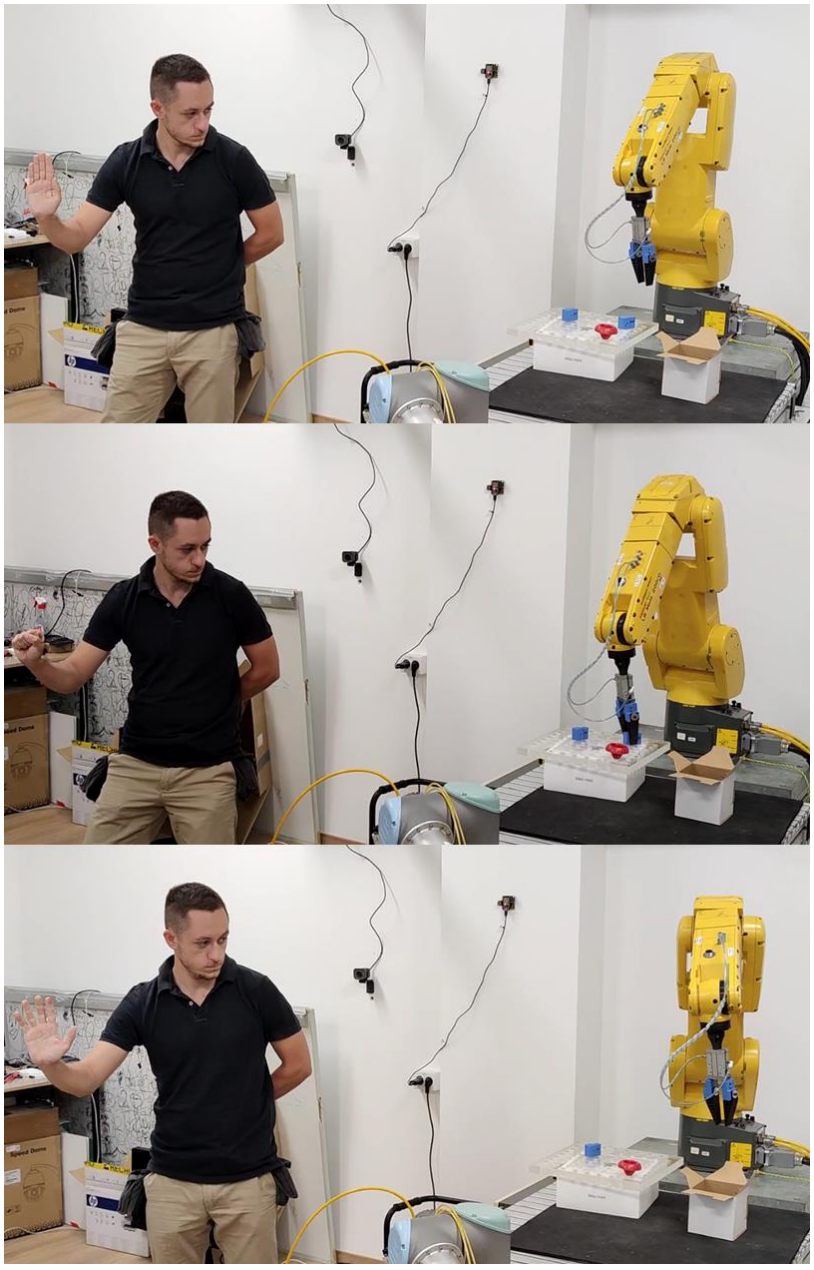


Slika 27. Kreiranje korisničkog programa – pomicanje robota (gore), zatvaranje hvataljke robota (sredina) i stvaranje mete (dole) (slika autora)

Korisnički program potom je pušten u rad samostalno, te je uspješno izveden. Računalni kod korisničkog programa i *RoboDK* radna stanica može se vidjeti u točki 5.10.

8.3. Testiranje izuzimanja predmeta – LR Mate 200iC

Iduće testiranje bilo je na istome robotu, a testiralo se izuzimanje predmeta, te ubacivanje u kutiju (slika 27.).



Slika 28. Testiranje industrijskog robota Fanuc LR Mate 200iC – izuzimanje predmeta i ubacivanje u kutiju (slika autora)

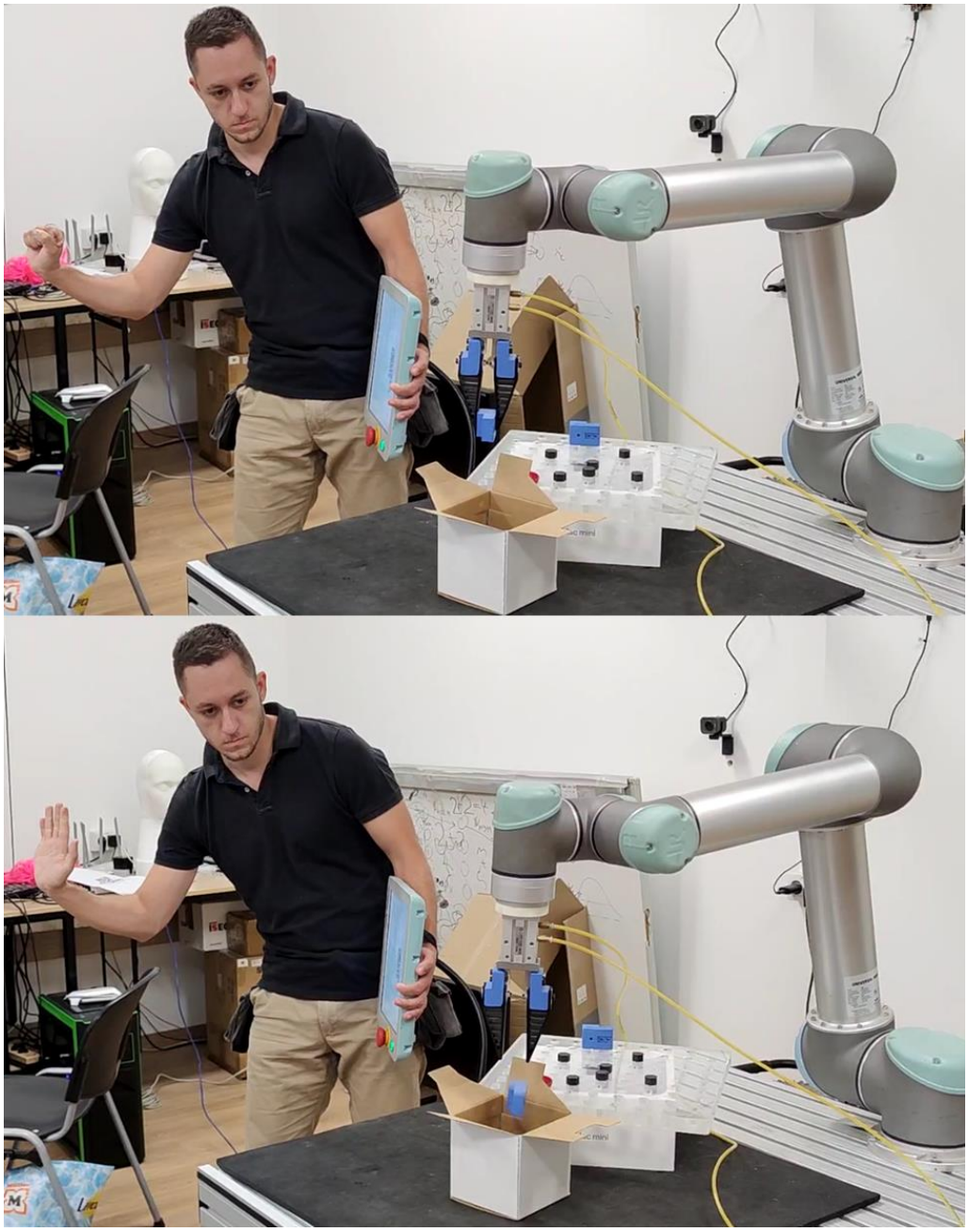
Gibanja između udaljenijih točaka bila su fluidnija, tj. prirodnija, međutim u trenutku kada se precizno pozicioniralo, kao recimo netom prije izuzimanja predmeta došlo je do izražene

diskretizacije pokreta. Pokazano je da sustav radi za relativno precizne operacije izuzimanja predmeta, te da nije složen za korištenje.

8.4. Testiranje izuzimanja predmeta – UR5

Uspješno je testirana aplikacija i na potpuno drugom robotu, odnosno kobotu proizvođača *Universal Robotics* – UR5. Rad sustava bio je lakši i fluidniji nego na prethodnom robotu. Izvršeno je izuzimanje predmeta, te odbacivanje predmeta u kutiju. (slika 29. i 30.).

Slika 29. Testiranje kobota UR5 – pozicioniranje i izuzimanje predmeta (slika autora)



Slika 30. Testiranje kobota UR5 – pomicanje i ubacivanje predmeta u kutiju (slika autora)

9. KRITIČKI OSVRT

Prilikom izrade vizijskog sustava za upravljanje industrijskim robotom pokretima ruke zapaženo je puno faktora koji utječu na kvalitetu rada aplikacije. Ta zapažanja odnose se na model prepoznavanja ruke *MediaPipea,* kamere *Intel RealSense D435,* komercijalni program *RoboDK,* hvataljke koje su izrađene, korisničko sučelje te samu eksperimentalnu prirodu aplikacije.

MediaPipe Hands je vrhunski alat čija je namjena za zabavu i istraživanje, dok rad ove aplikacije zapravo premašuje tu namjenu. Kako je model temeljen na podacima koji imaju određena odstupanja, tako i sami model pokazuje odstupanja, odnosno unosi znatan šum u rad aplikacije. Osvjetljenje, boja pozadine ili odjeće znatno mogu utjecati na dobro i sigurni prepoznavanje ruke i gesti ruke. Pojavljuje se i problem ako u kadar kamere uđe druga osoba, te se prepozna njezina ruka umjesto ruke korisnika, kada dolazi do neželjenog pomaka robota i potencijalno opasne situacije. Rješenje bi bilo izrada vlastitog modela za prepoznavanje ruke, te potencijalno i uvođenje nekih dodatnih sigurnosnih mjera poput recimo narukvice oko ruke korisnika ili slično.

Kamera *Intel RealSense D435* pokazala se iznimno kvalitetnom i točnom. Unutarnja kalibracija i postavke za visoku točnost ('*High Accuracy*') omogućuju precizna očitanja, međutim kada se kamera okrene prema stacionarnom objektu, recimo zidu, može se uočiti određeno titranje, šum kamere koji je dovoljno zamjetan za rad aplikacije. Također po specifikacijama kamere, mjerenja odstupaju manje od 2 % do 2 m, ali to je već vrijednost od mogućih 40 mm (na većoj udaljenosti i više) što može utjecati na preciznost rada.

Velika fleksibilnost programa za *online* i *offline* programiranje robota – *RoboDK* omogućila je povezivanje s robotima različitih proizvođača. Međutim ta veza nije ista za svaki robot, pojavljuju se razlike u radu između UR5 i LR Mate 200iC robota, ne može se softverski podesiti puna brzina ili to predstavlja sigurnosni rizik, pojavljuju se titranja u samome radu aplikacije koja se uglavnom ne prenose na gibanja robota, ali predstavljaju prepreku prije viših stupnjeva razvoja sustava za komercijalnu primjenu. Prednost, ali i problem *RoboDK* je samostalno izbjegavanje singulariteta u gibanjima robota, ali u eksperimentalnoj evaluaciji aplikacije moguća su stanja znatnih zakretanja pojedinih osi robota, te ulaska u područje singulariteta, te se taj problem također mora otkloniti prije viših faza razvoja aplikacije.

Tehnologija 3D tiskanja omogućila je jednostavnu i jeftinu izradu adaptera, odnosno prstiju robotske prihvatnice. U slučaju kolizije ne dolazi do znatne štete jer se definiraju gumeni prsti

robotske hvataljke. Kao poboljšanje za lakši rad korisnika, svakako bi trebalo uzeti u obzir hvataljke koje imaju tri ili više prstiju. Mogući je i razvoj hvataljki koje su više prilagođene korisniku, odnosno neki oblik adaptivnih hvataljki koje su razvijene evolucijskim algoritmima ili pak topološkom analizom.

Za uspješan daljnji razvoj aplikacije svakako treba uzeti u obzir i UI/UX dizajn. Jednostavno korisničko sučelje treba redizajnirati i nadograditi, te dodati još neke opcije poput pauziranja rada aplikacije, mijenjanja brzine, upotreba dodatnih gesti za te opcije, pa čak i druge ruke. Važno je naglasiti da se treba riješiti i problem sigurnog početka i kraja rada aplikacije, jer trenutno korisnik mora pritisnuti tipku 'q' na tipkovnici za kraj rada što nije uvijek sigurno, a za početak rada pozicija ruke u prostru također ne mora biti u sigurnoj poziciji za robota. Kako bi se dobila i šira slika kvalitete aplikacije, bilo bi dobro da se izvrši testiranje na više korisnika i u više različitih situacija, te čak i testiranje rada aplikacije na dalju, što ne bi trebao predstavljati znatan iskorak.

Iako je riješen problem titranja sustava i postignut je rad u realnom vremenu, to je žrtvovano diskretizacijom sustava na prag najmanje razlike između dviju točki da bi se ostvarilo gibanje. Taj prag je dovoljno velik da sustav postane grub, odnosno finesa pokreta čovjeka se ne prenosi na robota, iako znamo da robot ima mogućnost dobrog oponašanja preciznosti pokreta čovjeka – ali ne u realnom vremenu.

Zbog svega navedenoga aplikacija nije spremna za komercijalnu uporabu, iako je ostvareno dokazivanje koncepta.

10. ZAKLJUČAK

Izrada diplomskog rada tražila je sintezu više područja i različitih tehnologija. Iako je ideja relativno jednostavna – kamera snima ruku, generiraju se koordinate ruke i pošalju robotu koji se tamo pomakne, svaki eksperimentalni, praktični rad pokazuje probleme stvarnog svijeta, pa tako i ovaj. Povezivanje različitih tehnologija poput *RealSense* kamere, modela za prepoznavanje ruku *MediaPipea*, programa za programiranje robota *RoboDK* i stvarnih industrijskih robota nije lako niti jednostavno iako se koristio svestrani programski jezik *Python*. Radilo se na povezivanju sustava koji nisu zamišljeni da rade zajedno, a veći problem je rad sustava u stvarnom vremenu, s minimalnim kašnjenjem i minimalnom greškom uz to da je sustav maksimalno fleksibilan za rad sa što više različitih robota uz minimalne izmjene. Eksperimentalna evaluacija pokazala je mogućnost rada aplikacije na nekoliko načina i na dva različita robota, različitih proizvođača. Rad s aplikacijom je relativno jednostavan uz primitivno korisničko sučelje, međutim priprema same aplikacije za rad na različitim robotima nije nužno jednostavno te traži određenu razinu znanja rada s robotom na koji se implementira aplikacija.

Moguća su poboljšanja na gotovo svim aspektima aplikacije – sigurnost, posebno za početak i kraj rada aplikacije, dodatno smanjenje šuma aplikacije primjenom drugih filtera signala, odnosno drugačijim modelom za prepoznavanje koordinata ruke, drugačiji i prilagođeni izvršni član robota (poput adaptivnih hvataljki), primjena druge kamere ili drugačije tehnologije dohvaćanja koordinata ruke (recimo LIDAR – *Light Detection and Ranging*, tehnologija detekcije svjetlosnim signalima).

Zaključak je da je moguće napraviti vizijski sustav za upravljanje različitim industrijskim robotima pokretima ruke u realnome vremenu, ali uz ograničenja diskretizacije sustava, te da je otvorena mogućnost napretka u praktički svim podsustavima aplikacije.

LITERATURA

[1] robotika. Hrvatska tehnička enciklopedija, mrežno izdanje. Leksikografski zavod Miroslav Krleža, https://tehnika.lzmk.hr/robotika/ Pristupljeno: 19.09.2023.

[2] MHI: „Learning Center – Fundamentals – Industrial Robots“, https://www.mhi.org/fundamentals/robots Pristupljeno: 19.09.2023.

[3] Wikipedija: „Ljudsko oko“, https://hr.wikipedia.org/wiki/Ljudsko_oko, Pristupljeno 24.06.24.

[4] Wikipedija: „Mrežnica(oko), https://hr.wikipedia.org/wiki/Mre%C5%BEnica_(oko) Pristupljeno 24.06.24.

[5] spektar. Hrvatska enciklopedija, mrežno izdanje. Leksikografski zavod Miroslav Krleža, 2013. – 2024., https://www.enciklopedija.hr/clanak/spektar, Pristupljeno 24.06.2024

[6] Kemijski rječnik – elemtromagnetski spektar, https://glossary.periodni.com/rjecnik.php?hr=elektromagnetski+spektar, Pristupljeno 24.06.2024.

[7] Zidarić, M. (2022). Sustav za prepoznavanje rukom pisanog teksta (Završni rad). Zagreb: Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje, https://urn.nsk.hr/urn:nbn:hr:235:218726

[8] Wikipedia: „Python“..https://en.wikipedia.org/wiki/Python_(programming_language) Pristupljeno19.06.2024

[9] OpenCV, https://opencv.org/, Pristupljeno 19.06.2024.

[10] Google AI for Developers: Google AI Edge – MediaPipe, https://ai.google.dev/edge/mediapipe/solutions/guide, Pristupljeno 18.06.2024.

[11] Google AI for Developers: Google AI Edge – MediaPipe – Hand Landmarked, https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker, Pristupljeno 20.06.2024.

[12] MediaPipe Hands (Lite/Full) - Based on Model Cards for Model Reporting, In Proceedings of FAT* Conference (FAT*2019). ACM, New York, NY, USA, 9 pages

[13] Fitzpatrick, T. B. (1975). "Soleil et peau" [Sun and skin Sunce i koža]. Journal de Médecine Esthétique (na francuskom)

[14] Takahashi Shigeki; github: Kazuhito00 - hand-gesture-recognition-using-mediapipe Apache-2.0, https://github.com/Kazuhito00/hand-gesture-recognition-using-mediapipe license, Pristupljeno 20.06.2024.

[15] RoboDK, https://robodk.com/, Pristupljeno 18.06.2024.

[16] Intel RealSense: Depth Camera D435, https://www.intelrealsense.com/depth-camera-d435/ Pristupljeno 18.06.2024.

[17] eurobots: used robots – Fanuc LR Mate 200IC/5L, https://www.eurobots.net/used_fanuc-robots-lr-mate-200ic-5l-en.html, Pristupljeno 19.09.2023

[18] Universal Robots – Products – UR5, https://www.universal-robots.com/products/ur5-robot/ Pristupljeno 19.09.2023.

[19] Brown, R. G. (1963). Smoothing, Forecasting and Prediction of Discrete Time Series. Prentice-Hall

[20] Gardner, E. S. (1985). Exponential smoothing: The state of the art. Journal of Forecasting, 4(1), 1-28

[21] Savitzky, A., & Golay, M. J. E. (1964). Smoothing and Differentiation of Data by Simplified Least Squares Procedures. *Analytical Chemistry*, 36(8), 1627-1639

[22] SMCpneumatics: MHZ2-20DN-A310, https://www.smcpneumatics.com/MHZ2-20DN-A310.html Pristupljeno: 24.06.2024.

[23] Wikipedia: Polylactic acid, https://en.wikipedia.org/wiki/Polylactic_acid Pristupljeno: 24.06.2024.

[24] Omnexus – selection guide – thermoplastic polyurethanes, https://omnexus.specialchem.com/selection-guide/thermoplastic-polyurethanes-tpu Pristupljeno: 24.06.2024.

PRILOZI

Python kod – Vizijski sustav za upravljanje industrijskim robotom pokretima ruke

```
from robodk import robolink
from robodk.robolink import *
from robodk.robomath import *
import cv2
import mediapipe as mp
import numpy as np
import pyrealsense2 as rs
import time
from Gesture_classifier import GestureClassifier
import shutil

MAX_NUM_HANDS = 1                    # standardno 1, može se izmijeniti u 2
MIN_DETECTION_CONFIDENCE = 0.5       # standardno 0.5
MIN_TRACKING_CONFIDENCE = 0.5        # standardno 0.5
RESOLUTION_X = 1280                  # rezolucija x, standardno 640 (1280)
RESOLUTION_Y = 720                   # rezolucija y, standardno 480 (720)
ROBO_CELL_WIDTH = 1300               # širina robotske čelije, x koordinata
ROBO_CELL_HEIGHT = 1000              # visina robotske čelije, y koordinata
ROBO_CELL_DEPTH = 1000               # dubina robotske čelije, z koordinata
num_target = 1                       # početni broj mete
POINT_THRESHOLD = 5                  # minimalna razlika između
                                     # koordinata za očitanje pomaka

RDK = Robolink()                     # uspostavljanje veze sa RoboDK
RDK.Connect()                        # povezivanje robota i RoboDK
RDK.setRunMode(RUNMODE_SIMULATE)     # uspostavljanje veze između robota i
simulatora
RDK.setRunMode(1)       # korištenje upravljačkog programa robota zajedno
sa simulatorom

# Postavke za Fanuc LR Mate 200ic
# poveznica na .rdk datoteku
RDK.AddFile("C:\\Users\\Zmaj\\Desktop\\Fanuc_LR_Mate_200ic.rdk")
robot = RDK.Item('Fanuc LR Mate 200iC')  # dohvaćanje robota po imenu
robot.setSpeed(1)                        # zadavanje brzine robota [mm/s]
robot.setJoints([0, 0, 0, 0, 0, 0])      # postavljanje svih osi robota u 0
robot.Connect('192.168.0.26')        # povezivanje RoboDK i robota preko IP
adrese

# Postavke za UR5
# poveznica na .rdk datoteku
# RDK.AddFile("C:\\Users\\Zmaj\\Desktop\\UR5.rdk")
# robot = RDK.Item('UR5_robot')          # dohvaćanje robota po imenu
# robot.setSpeed(1)                      # zadavanje brzine robota [mm/s]
# robot.setJoints([0, 0, 0, 0, 0, 0])    # postavljanje svih osi robota u 0
# # robot.Connect('192.168.0.25')    # povezivanje RoboDK i robota preko IP
adrese

print(robot.ConnectedState())        # Ispisivanje povezanosti robota
time.sleep(20)                       # Pauza 20 s za ručno povezivanje
print(robot.ConnectedState())
```

```
target = RDK.Item('Target_reference')  # dohvaćanje Target_reference mete
robot.MoveJ(target)                # Joint pomak robota u metu
pose_ref = robot.Pose()            # dohvaćanje trenutnog položaja robota

# Inicijalizacija i konfiguriranje RealSense kamere
pipeline = rs.pipeline()
config = rs.config()
config.enable_stream(rs.stream.color, RESOLUTION_X, RESOLUTION_Y,
rs.format.bgr8, 30)
config.enable_stream(rs.stream.depth, RESOLUTION_X, RESOLUTION_Y,
rs.format.z16, 30)
pipeline.start(config)

depth_sensor = pipeline_profile.get_device().first_depth_sensor()

# Definiranje postavki RealSense kamere na način
# prijenosa visoke preciznosti ("High Accuracy")
preset_range = depth_sensor.get_option_range(rs.option.visual_preset)
print(preset_range)
for i in range(int(preset_range.max)):
    visual_preset =
depth_sensor.get_option_value_description(rs.option.visual_preset, i)
    print(visual_preset)
    if visual_preset == 'High Accuracy':
        depth_sensor.set_option(rs.option.visual_preset, i)

# Deklariranje liste last_point za usporedbu koordinata
last_point = []

# Initialize refresh_time
refresh_time = time.time()

# Inicijalizacija modula za anotacije i vizualizacije Ruku na slikama
mp_drawing = mp.solutions.drawing_utils
# konfiguriranje modula Hands
mp_hands = mp.solutions.hands
hands = mp_hands.Hands(
    static_image_mode=False,
    max_num_hands=MAX_NUM_HANDS,
    min_detection_confidence=MIN_DETECTION_CONFIDENCE,
    min_tracking_confidence=MIN_TRACKING_CONFIDENCE,
)

align_to = rs.stream.color  # povezivanje aligna (poravnavanje slike) sa
RealSense kamerom
align = rs.align(align_to)

GC = GestureClassifier()    # inicijalizacija gesture classifier-a
temp_gest = ''      # prazan string za spremanje prepoznatih gesti

# Definiranje klase filtera pomičnog prosjeka ("Moving Average")
class StreamingMovingAverage:
    def __init__(self, window_size):
        self.window_size = window_size
        self.values = []
        self.sum = 0
```

```
    def process(self, value):
        self.values.append(value)
        self.sum += value
        if len(self.values) > self.window_size:
            self.sum -= self.values.pop(0)
        return float(self.sum) / len(self.values)

# Definiranje veličine skupa točaka za filter
smax = StreamingMovingAverage(10)
smay = StreamingMovingAverage(10)
smaz = StreamingMovingAverage(10)


# Definiranje funkcije clamp_x i clamp_y
def clamp_x(n):
    return max(min(n, RESOLUTION_X - 1), 1)


def clamp_y(n):
    return max(min(n, RESOLUTION_Y - 1), 1)

def capture_from_cam(program):  # početak funkcije i uvođenje globalnih
varijabli
    global temp_gest
    global num_target
    global last_point

    # Dohvaćanje slike kamere i priprema RGB i depth značajki slike
    frames = pipeline.wait_for_frames()
    color_frame = frames.get_color_frame()
    depth_frame = frames.get_depth_frame()

    # Poravnavanje RGB i depth značajki slike
    aligned_frames = align.process(frames)
    aligned_depth_frame = aligned_frames.get_depth_frame()
    depth_intrin =
aligned_depth_frame.profile.as_video_stream_profile().intrinsics

    # Pretvorba slike u niz - za rad sa MediaPipeom
    image = np.asanyarray(color_frame.get_data())

    image = cv2.flip(image, 1)    # Zrcaljenje slike

    # Pretvorba slike iz BGR u RGB
    results = hands.process(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))

    # Prepoznavanje geste
    gest = GC.run_recognition(results, image.shape[1], image.shape[0])
    # Ako je prepoznata gesta 'OK'
    # Izjednači je sa gestom 'Open'
    gest = 'Open' if gest == 'OK' else gest

    # Nastavi program dok se depth i RGB značajke ne prepoznaju
    if not depth_frame or not color_frame:
        return None
```

```
    # za prepoznatu ruku iscrtaj karakteristične točke
     if results.multi_hand_landmarks:
         for hand_landmarks in results.multi_hand_landmarks:
             mp_drawing.draw_landmarks(
                 image, hand_landmarks, mp_hands.HAND_CONNECTIONS)

             # Dohvaćanje koordinata zapešća (wrist)
             wrist = hand_landmarks.landmark[mp_hands.HandLandmark.WRIST]

             # Ograničavanje koordinata na rezoluciju slike
             # Convert the normalized coordinates to pixel coordinates
             wrist_x = clamp_x(int((wrist.x * image.shape[1] -
                               image.shape[1]) * -1))
             wrist_y = clamp_y(int(wrist.y * image.shape[0]))

             # Prilagodba koordinata kamere I MediaPipea
             depth_value_mm = aligned_depth_frame.get_distance(wrist_x,
                                                               wrist_y)
             point = rs.rs2_deproject_pixel_to_point(
                     depth_intrin, [wrist_x, wrist_y], depth_value_mm)

             # Rotacija koordinatnog sustava (vrijednosti u m)
             point[1] = -point[1]
             point[2] = -(point[2] - 2) # - 2 označava pomak ishodišta
                                        # 2 m od kamere

             # Spremanje koordinata u listu
             point = [smax.process(point[0]), smay.process(point[1]),
smaz.process(point[2])]

             # Ispisivanje koordinata u terminal
             print(f'X: {round(point[0], 2)},
                     Y: {round(point[1], 2)},
                     Z: {round(point[2], 2)}')

             # Provjeri je li razlika između zadnje očitane koordinate i
trenutno očitane
             # koordinate veća od granične vrijednosti (threshold)
             # Ako da, onda je došlo do pomaka, a ako ne onda nije došlo do
pomaka
             # I trenutna koordinata je jednaka idućoj
             if len(last_point) != 0:
                 if all(abs(a - b) < POINT_THRESHOLD
                 for a, b in zip(last_point, point)):
                     point = last_point

             last_point = point

             # Ispisivanje prepoznate geste
             cv2.putText(image,
                         text=f"{gest}",          # tekst ispisa
                         org=(50, 50),            # koordinate ispisa
                         fontFace=cv2.FONT_HERSHEY_SIMPLEX,  # font
                         fontScale=2,             # skaliranje fonta
                         color=(0, 255, 0),       # boja fonta
                         thickness=3)             # debljina fonta
```

```
# Ispisivanje veličine radnog prostora
cv2.putText(image,
            text=f"W: +/- {ROBO_CELL_WIDTH / 2}
                   H: {ROBO_CELL_HEIGHT},
                   D: +/- {ROBO_CELL_DEPTH / 2}",
            org=(50, 700),
            fontFace=cv2.FONT_HERSHEY_SIMPLEX,
            fontScale=1,
            color=(0, 255, 0),
            thickness=3)

# Provjera jesu li koordinate unutar radnog prostora
if ((-ROBO_CELL_WIDTH / 2) <= round(point[0], 2) <=
   (ROBO_CELL_WIDTH / 2)
and
   -300 <= round(point[1], 2) <= ROBO_CELL_HEIGHT
and
    (-ROBO_CELL_DEPTH / 2) < round(point[2], 2) <
    (ROBO_CELL_DEPTH / 2)):
    print("Inside Work Range") # Ispis da su koordinate unutar
                               # radnog prostora

    # Ispisivanje koordinata ruke na sliku
    cv2.putText(image,
                text=f"{round(point[0], 2)},"
                     f"{round(point[1], 2)},"
                     f"{round(point[2], 2)}",
                org=(-1 * (wrist_x - image.shape[1]),
                     wrist_y + 20),
                fontFace=cv2.FONT_HERSHEY_SIMPLEX,
                fontScale=1,
                color=(245, 0, 0),
                thickness=3)

    # Pomicanje robota prema koordinatama
    # Zaokruživanje vrijednosti koordinata na 2 decimale
    # Slanje minimalne rotacije osi da ne dođe do greške
    # u RoboDK
    approach = target.Pose() * transl(round(point[0], 2),
                                      round(point[2], 2),
                                      -(round(point[1], 2)))
                                      * rotx(0.1)
                                      * roty(0.1)
                                      * rotz(0.1)

    # Provjera je li robot u pomaku ili ne
    # Ako robot nije u pomaku, izvrši pomak
    if robot.Busy() == 0:
        # Provjeri je li došlo do promjene geste ruke
        # Open/Close - Otvorena/Zatvorena hvataljka
        # Pointer - stvaranje mete u RoboDK
        if gest != temp_gest:
            temp_gest = gest
            if gest == 'Open':
                # robot.setDO(0, 0)
                robot.RunInstruction('GR_OP',
                robolink.INSTRUCTION_CALL_PROGRAM)
```

```
                        # Zapis instrukcije u korisnički program
                        program.write(
                            "robot.RunInstruction('GR_OP',
                             robolink.INSTRUCTION_CALL_PROGRAM)"
                             + "\n\n")
                    elif gest == 'Close':
                        #robot.setDO(0, 1)
                        robot.RunInstruction('GR_CL',
                        robolink.INSTRUCTION_CALL_PROGRAM)
                        # Zapis instrukcije u korisnički program
                        program.write(
                            "robot.RunInstruction('GR_CL',
                             robolink.INSTRUCTION_CALL_PROGRAM)"
                             + "\n\n")
                    elif gest == 'Pointer':
                        # Stvori novu metu (Target) na trenutnim
                        # koordinatama ruke
                        ti = RDK.AddTarget(
                             f'User_Target_{num_target}', 1)
                        pose_tar = robot.Pose()
                        ti.setPose(pose_tar)
                        ti.setAsCartesianTarget()
                        time.sleep(1)
                        # Zapis instrukcije u korisnički program
                        program.write(
                            f"ti=RDK.Item(f'User_Target_
                            {num_target}')" + "\n"
                            "robot.MoveJ(ti)" + "\n\n")
                        num_target += 1

                # Zanemari grešku ako robot ne može dosegnuti metu
                # Ako robot nije u pomaku, izvrši pomak
                try:
                    print(approach)
                    robot.MoveJ(approach, blocking=False)
                except TargetReachError:
                    print('Target not reachable')

            else:
                print("Out of range")
                # Ispis da su koordinate van dosega robota
                # Ispis crvenih koordinata ruke na sliku
                cv2.putText(image,
                            text=f"{round(point[0], 2)},"
                                 f"{round(point[1], 2)},"
                                 f"{round(point[2], 2)}",
                            org=(-1 * (wrist_x - image.shape[1]),
                                 wrist_y + 20),
                            fontFace=cv2.FONT_HERSHEY_SIMPLEX,
                            fontScale=1,
                            color=(0, 0, 255),
                            thickness=3)

    # Ispiši naziv programa
    cv2.imshow('Mediapipe Realsense D435 Hands Tracking', image)
```

```
base_coord_captured = False # Početna izjava za rad petlje

# Make a copy of the robot program template
src = "Template.txt"
dst = "Robot_program.py"

# Provjera jel postoji već program sa tim imenom
# Ako postoji, sprema se pod prvim većim brojem
i = 1
while os.path.exists(dst):
    dst = f"Robot_program{i}.py"
    i += 1

shutil.copy(src, dst)    # Izrada kopije predloška ("Template") i
                         # Spremanje u "Robot_program.py"
file = open(dst, "a")    # Otvaranje nove datoteke i omogućeno dodavanje na
                         # kraj (append)

while not base_coord_captured:      # Rad programa u petlji
    capture_from_cam(file)          # Vraćanje vrijednosti funkcije
    if cv2.waitKey(1) & 0xFF == ord('q'):   # Zatvori aplikaciju ako
korisnik
        break                               # pritisne tipku 'q'

file.close()                    # Zatvori datoteku

cv2.destroyAllWindows()         # Zatvori sve prozore

pipeline.stop()                 # Zaustavi vezu sa RealSense kamerom

RDK.Save("User_program.rdk")    # Spremi RoboDK program
RDK.CloseRoboDK()               # Zatvori RoboDK program
```

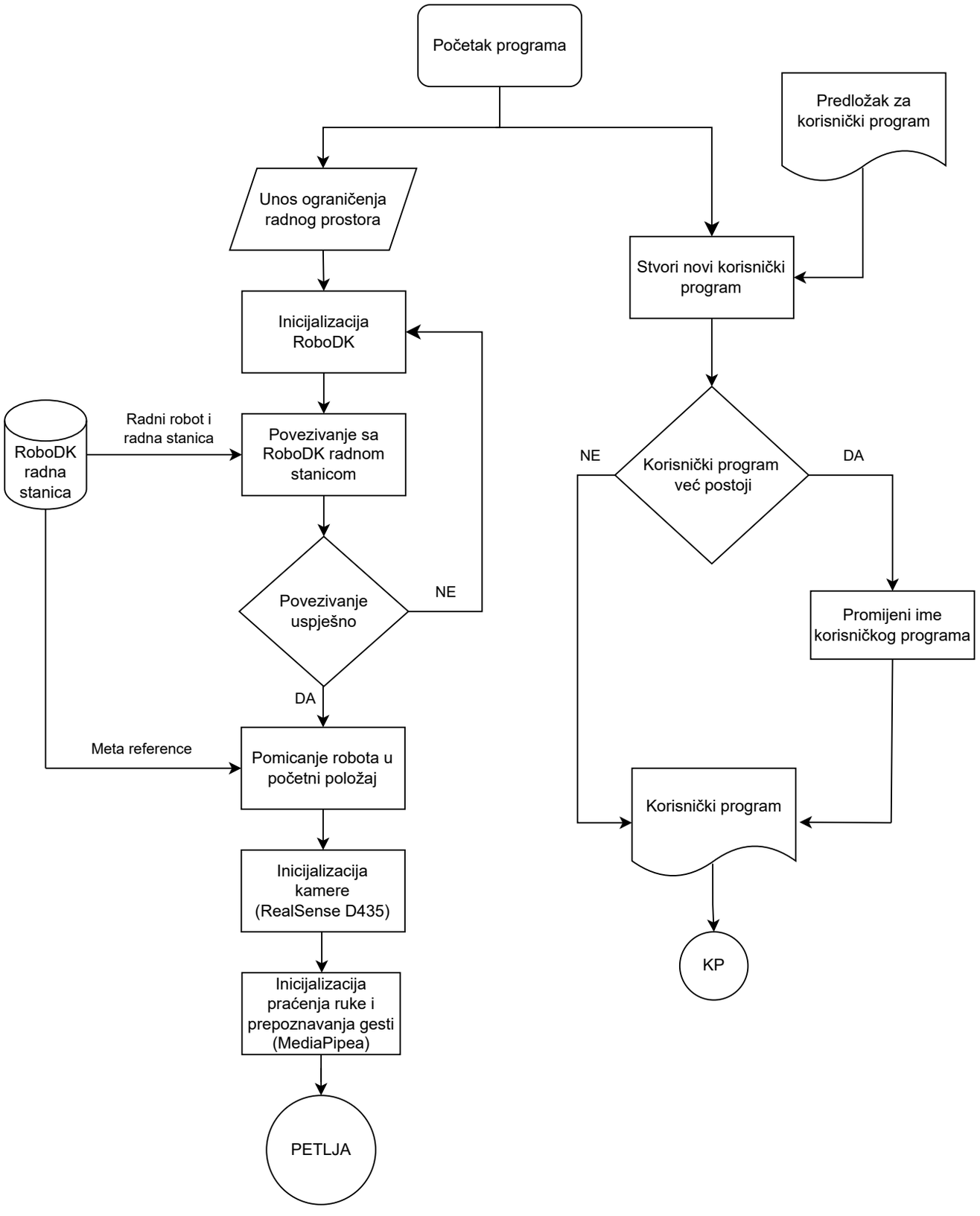
Python kod – izrada grafova filtera

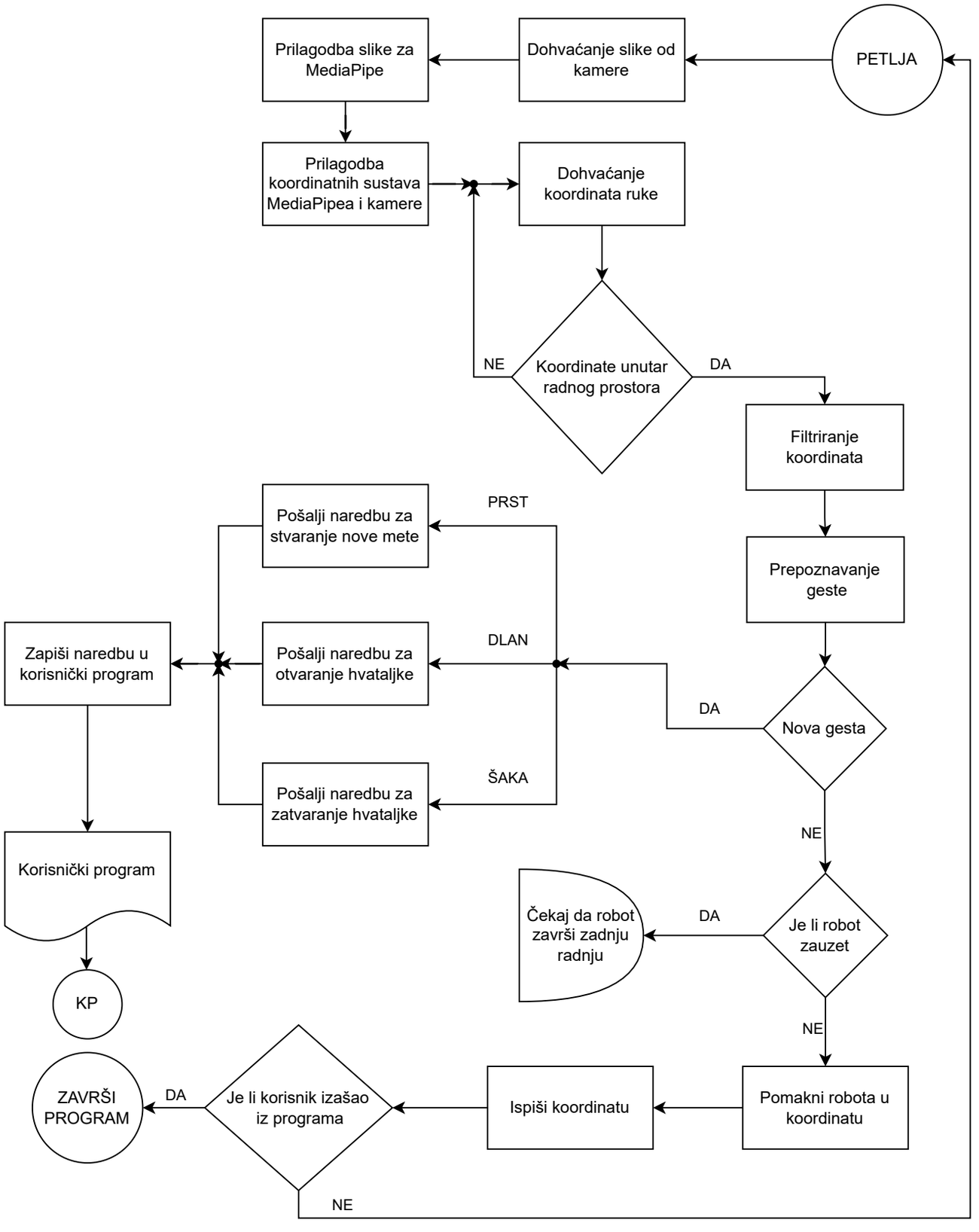
```
# Python skripta za plotanje grafova. Uključuje 3 filtera: Moving Average,
# Exponential Smoothing i Savitzky-Golay. Učitavanje podataka iz .txt dato-
# teke. Podešavanje parametara filtera prilikom inicijalizacije. Plotanje
# 4 usporedba različitih filtera|graf za svaki tip filtera s 3 različita
# parametra


# Import modula za izradu grafova
import matplotlib.pyplot as plt
# Import Savitzky-Golay filtera
from scipy.signal import savgol_filter

# definiranje klase filtera pomičnog prosjeka
class StreamingMovingAverage:
    def __init__(self, window_size):
        self.window_size = window_size
        self.values = []
        self.sum = 0

    def process(self, value):
        self.values.append(value)
        self.sum += value
        if len(self.values) > self.window_size:
            self.sum -= self.values.pop(0)
        return float(self.sum) / len(self.values)


# definiranje klase filtera eksponencijalnog izglađivanja
class StreamingExponentialSmoothing:
    def __init__(self, alpha):
        self.alpha = alpha
        self.prev_value = None

    def process(self, value):
        if self.prev_value is None:
            self.prev_value = value
        else:
            self.prev_value = (value - self.prev_value) *
            self.alpha +  self.prev_value
        return self.prev_value


# definiranje klase Savitzky-Golay filtera
class StreamingSavitzkyGolayFilter:
    def __init__(self, window_size, poly_order):
        self.window_size = window_size
        self.poly_order = poly_order
        self.values = []

    def process(self, value):
        self.values.append(value)
        if len(self.values) < self.window_size:
            return value
        if len(self.values) > self.window_size:
            self.values.pop(0)
            return savgol_filter(self.values,
                                 self.window_size,
                                 self.poly_order)[-1]
```

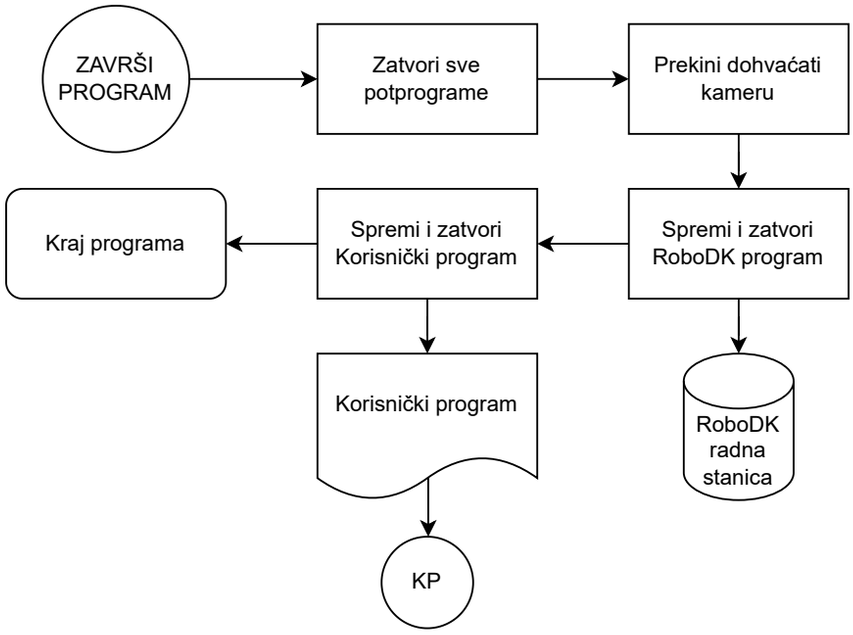
```
# Učitavanje podataka iz .txt datoteke
with open("Data_coordinates_x.txt", "r") as f:
    data = [float(line.strip()) for line in f.readlines()]

# Inicijalizacija filtera sa različitim parametrima
sma_1 = StreamingMovingAverage(5)
sma_2 = StreamingMovingAverage(10)
sma_3 = StreamingMovingAverage(20)

ses_1 = StreamingExponentialSmoothing(0.1)
ses_2 = StreamingExponentialSmoothing(0.2)
ses_3 = StreamingExponentialSmoothing(0.5)

ssg_1 = StreamingSavitzkyGolayFilter(51, 2)
ssg_2 = StreamingSavitzkyGolayFilter(51, 3)
ssg_3 = StreamingSavitzkyGolayFilter(25, 3)

# Process data
sma_1_values = []
sma_2_values = []
sma_3_values = []

ses_1_values = []
ses_2_values = []
ses_3_values = []

ssg_1_values = []
ssg_2_values = []
ssg_3_values = []

for value in data:
    sma_1_value = sma_1.process(value)
    ses_1_value = ses_1.process(value)
    ssg_1_value = ssg_1.process(value)
    sma_1_values.append(sma_1_value)
    ses_1_values.append(ses_1_value)
    ssg_1_values.append(ssg_1_value)

    sma_2_value = sma_2.process(value)
    ses_2_value = ses_2.process(value)
    ssg_2_value = ssg_2.process(value)
    sma_2_values.append(sma_2_value)
    ses_2_values.append(ses_2_value)
    ssg_2_values.append(ssg_2_value)

    sma_3_value = sma_3.process(value)
    ses_3_value = ses_3.process(value)
    ssg_3_value = ssg_3.process(value)
    sma_3_values.append(sma_3_value)
    ses_3_values.append(ses_3_value)
    ssg_3_values.append(ssg_3_value)

# Plotanje usporedbe filtera
plt.plot(data, 'bo-', label='Original Data', alpha=0.5)
# plavi krugovi i puna linija
plt.plot(sma_2_values, 'r-', label='Moving Average 10', linewidth=2)
# crvena puna linija
plt.plot(ses_3_values, 'g--', label='Exponential Smoothing 0.5')
# zelena crtkana linija
```

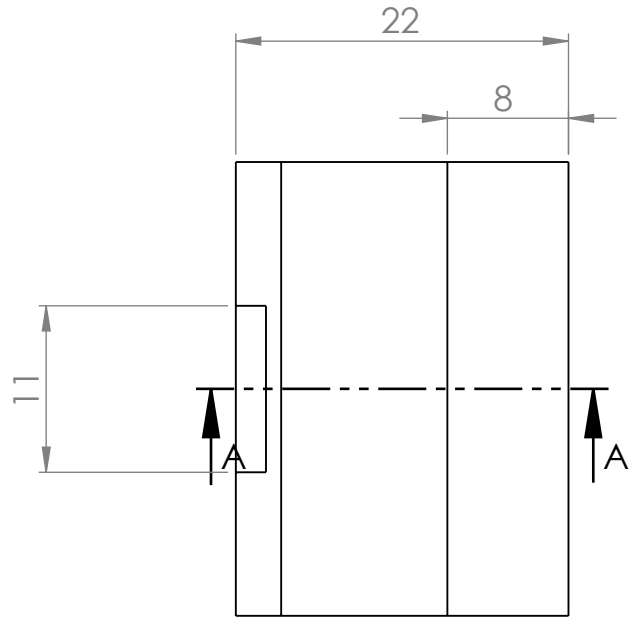
```
plt.plot(ssg_3_values, 'm-.', label='Savitzky-Golay Filter 25, 3')
# magenta crta točka linija
plt.grid(which='both', color='black', linestyle='--', linewidth=0.5)
plt.legend()
plt.title("Usporedba filtera")
plt.xlabel("Vrijeme")
plt.ylabel("Očitana koordinata [mm]")
plt.show()

# Plotanje pomičnog prosjeka
plt.plot(data, 'bo-', label='Original Data', alpha=0.5)
# plavi krugovi i puna linija
plt.plot(sma_1_values, 'r-', label='Moving Average 5', linewidth=2)
# crvena puna linija
plt.plot(sma_2_values, 'g--', label='Moving Average 10')
# zelena crtkana linija
plt.plot(sma_3_values, 'm-.', label='Moving Average 20')
# magenta crta točka linija
plt.grid(which='both', color='black', linestyle='--', linewidth=0.5)
plt.legend()
plt.title("Moving Average")
plt.xlabel("Vrijeme")
plt.ylabel("Očitana koordinata [mm]")
plt.show()

# Plotanje eksponencijalnog izglađivanja
plt.plot(data, 'bo-', label='Original Data', alpha=0.5)
# plavi krugovi i puna linija
plt.plot(ses_1_values, 'r-', label='Exponential Smoothing 0.1',
linewidth=2)  # crvena puna linija
plt.plot(ses_2_values, 'g--', label='Exponential Smoothing 0.2')
# zelena crtkana linija
plt.plot(ses_3_values, 'm-.', label='Exponential Smoothing 0.5')
# magenta crta točka linija
plt.grid(which='both', color='black', linestyle='--', linewidth=0.5)
plt.legend()
plt.title("Exponential Smoothing")
plt.xlabel("Vrijeme")
plt.ylabel("Očitana koordinata [mm]")
plt.show()

# Plotanje Savitzky-Golay filtera
plt.plot(data, 'bo-', label='Original Data', alpha=0.5)
# plavi krugovi i puna linija
plt.plot(ssg_1_values, 'r-', label='Savitzky-Golay Filter 25, 2',
linewidth=2)  # crvena puna linija
plt.plot(ssg_2_values, 'g--', label='Savitzky-Golay Filter 51, 3')
# zelena crtkana linija
plt.plot(ssg_3_values, 'm-.', label='Savitzky-Golay Filter 25, 3')
# magenta crta točka linija
plt.grid(which='both', color='black', linestyle='--', linewidth=0.5)
plt.legend()
plt.title("Savitzky-Golay")
plt.xlabel("Vrijeme")
plt.ylabel("Očitana koordinata [mm]")
plt.show()
```

DIJAGRAM TOKA APLIKACIJE VIZIJSKOG SUSTAVA ZA UPRAVLJANJE INDUSTRIJSKIM ROBOTOM POKRETIMA RUKE

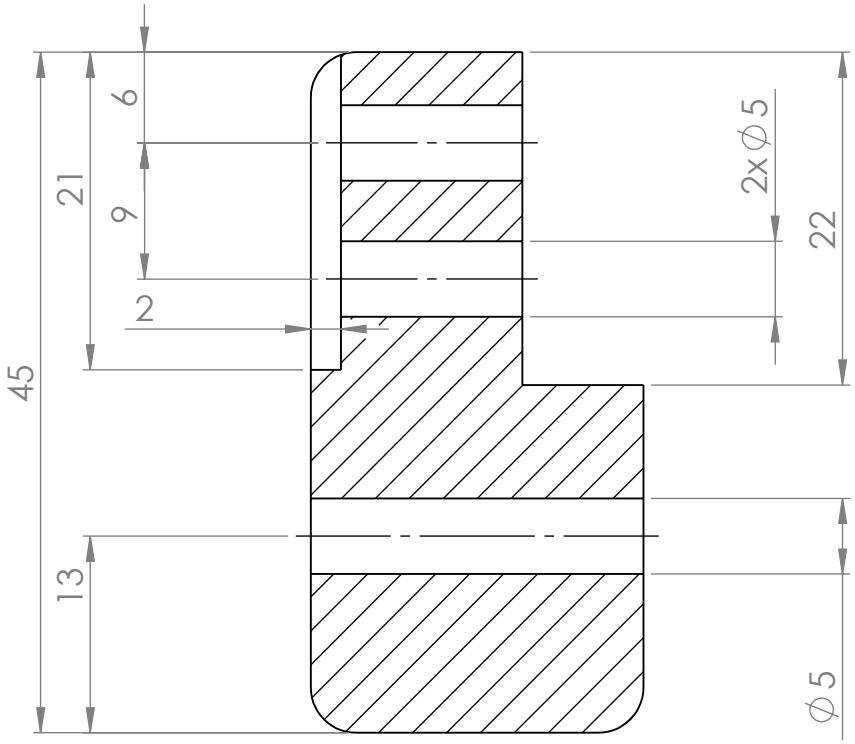



PETLJA
Dohvaćanje slike od kamere
Prilagodba slike za MediaPipe
Prilagodba koordinatnih sustava MediaPipea i kamere
Dohvaćanje koordinata ruke
Koordinate unutar radnog prostora
NE
DA
Filtriranje koordinata
Prepoznavanje geste
Nova gesta
DA
NE
PRST
Pošalji naredbu za stvaranje nove mete
DLAN
Pošalji naredbu za otvaranje hvataljke
ŠAKA
Pošalji naredbu za zatvaranje hvataljke
Zapiši naredbu u korisnički program
Korisnički program
KP
Je li robot zauzet
DA
Čekaj da robot završi zadnju radnju
NE
Pomakni robota u koordinatu
Ispiši koordinatu
Je li korisnik izašao iz programa
DA
ZAVRŠI PROGRAM
NE


ZAVRŠI PROGRAM
Zatvori sve potprograme
Prekini dohvaćati kameru
Kraj programa
Spremi i zatvori Korisnički program
Spremi i zatvori RoboDK program
Korisnički program
RoboDK radna stanica
KP



A-A
M



	Datum	Ime i prezime	Potpis
Projektirao	20.06.24.	Matija Zidarić	
Razradio			
Crtao	20.06.24.	Matija Zidarić	
Pregledao			

FSB Zagreb

Objekt:	Objekt broj:
	R. N. broj:

Napomena:
Nekotirani radijusi iznose R3

Kopija

Materijal: PLA

Masa:

Mjerilo originala
1:1

Naziv:
Adapter hvataljke

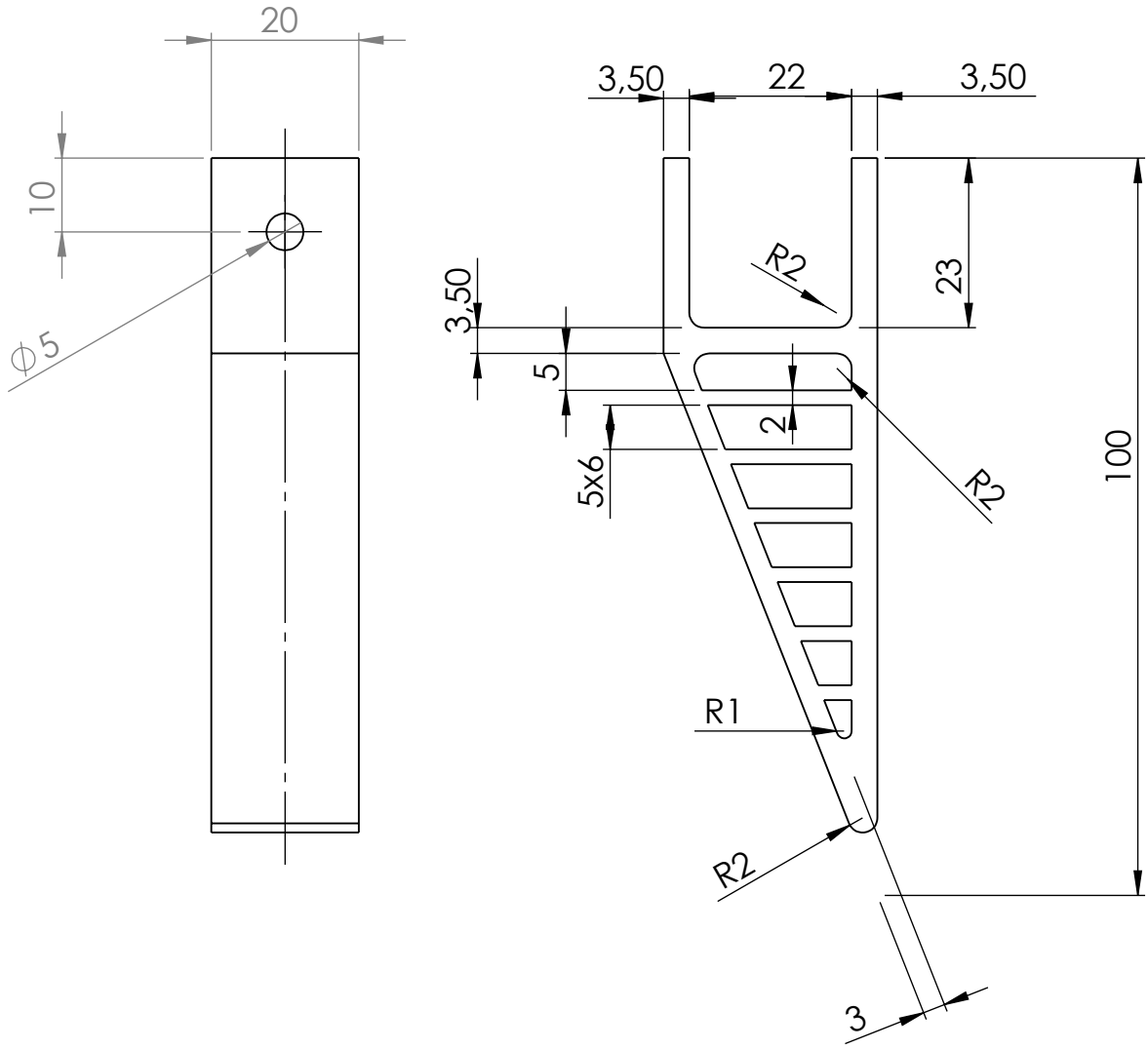
Pozicija: 1

Format: A4

Listova: 5

Crtež broj: AH-1

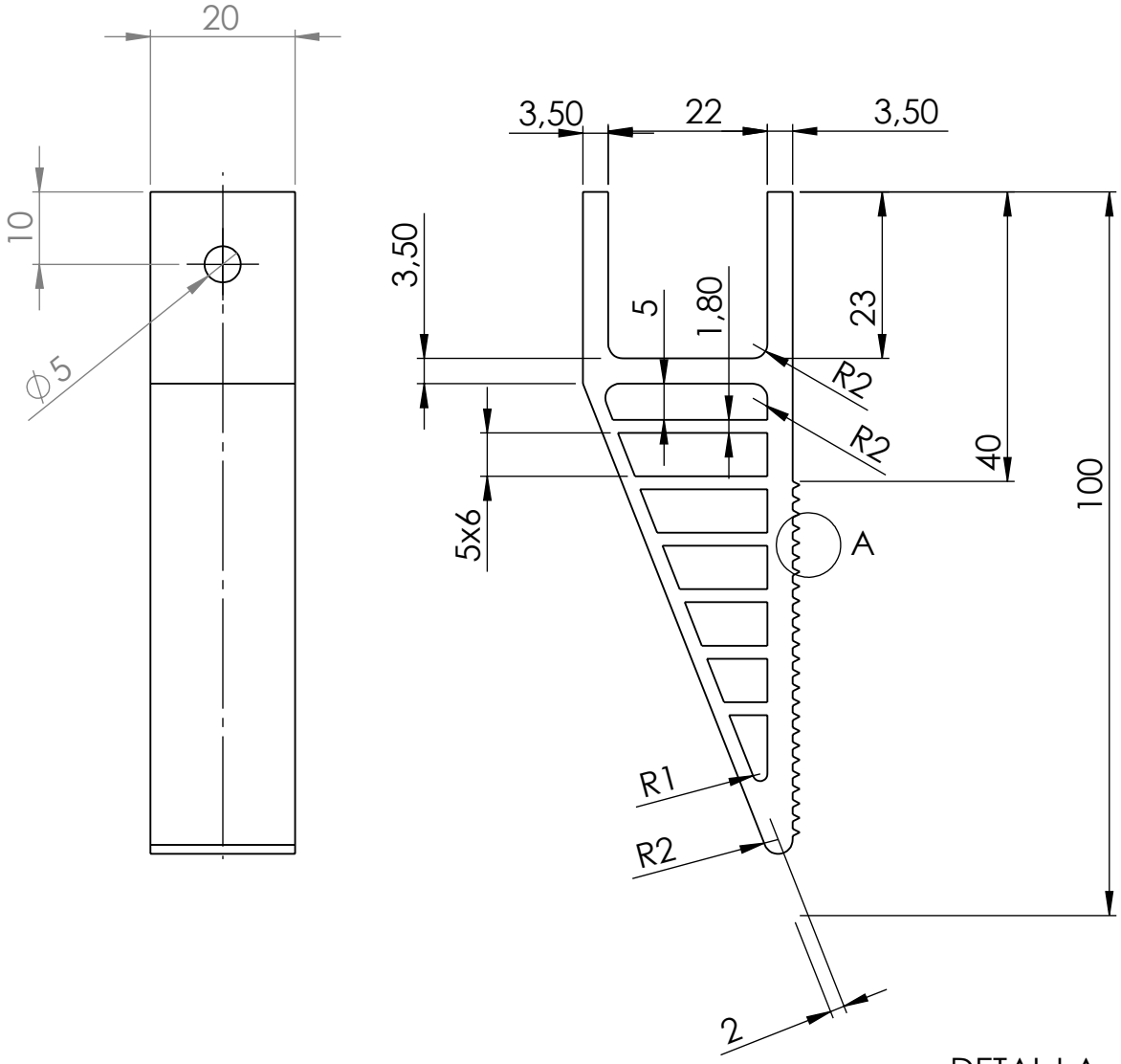
List: 1



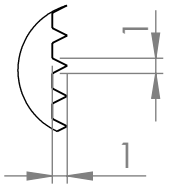
	Datum	Ime i prezime	Potpis
Projektirao	20.06.24.	Matija Zidarić	
Razradio			
Crtao	20.06.24.	Matija Zidarić	
Pregledao			

FSB Zagreb

Objekt:	Objekt broj:
	R. N. broj:
Napomena:	Kopija
Materijal: TPU	Masa:
Mjerilo originala 1:1	Naziv: Prst hvataljke V1 - 20
Pozicija: 2.1	Format: A4
Listova: 5	List: 2
Crtež broj: PH-1-20	



DETALJ A
M 2 : 1

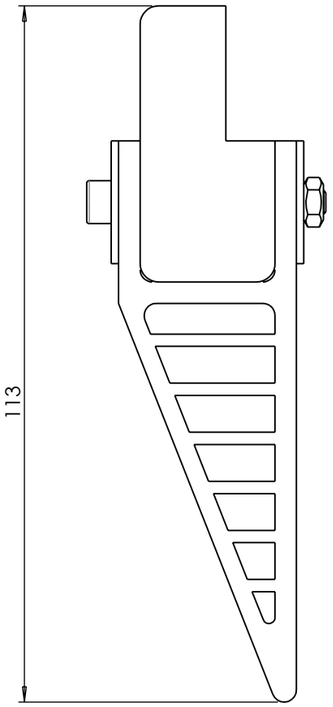


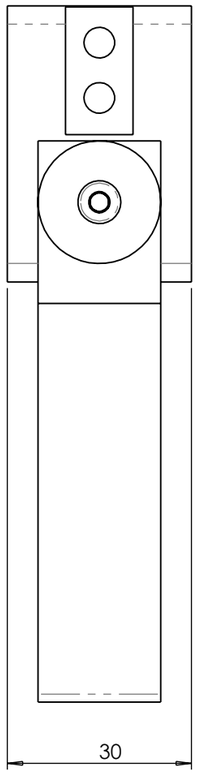
	Datum	Ime i prezime	Potpis
Projektirao	20.06.24.	Matija Zidarić	
Razradio			
Crtao	20.06.24.	Matija Zidarić	
Pregledao			

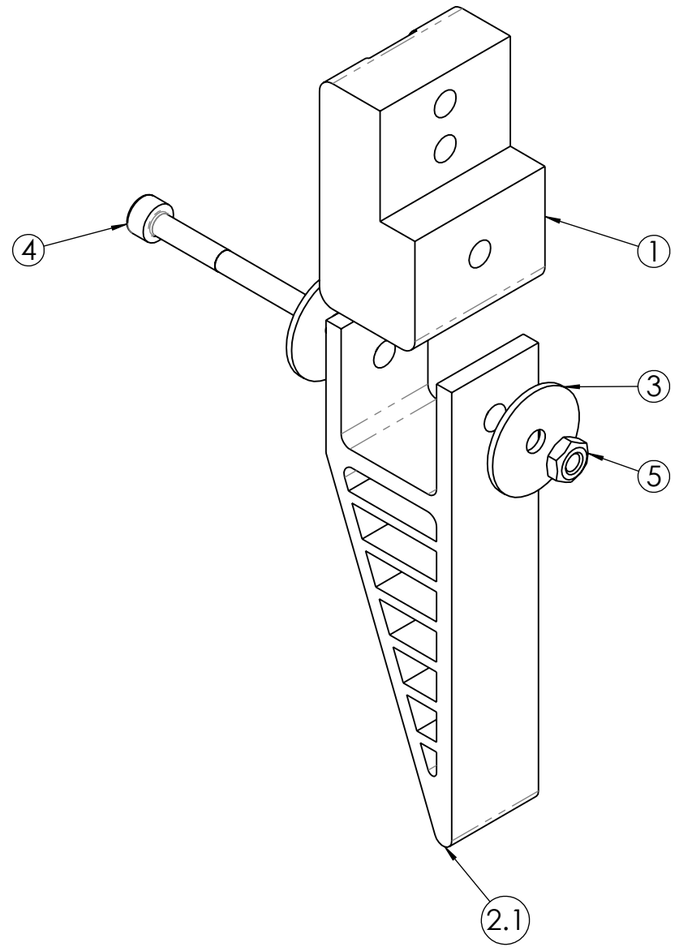
FSB Zagreb

Objekt:	Objekt broj:
	R. N. broj:
Napomena:	Kopija
Materijal: TPU	Masa:
Mjerilo originala 1:1	Naziv: Prst hvataljke V2 - 20
Pozicija: 2.2	Format: A4
Listova: 5	
Crtež broj: PH-2-20	List: 3

Design by CADLab







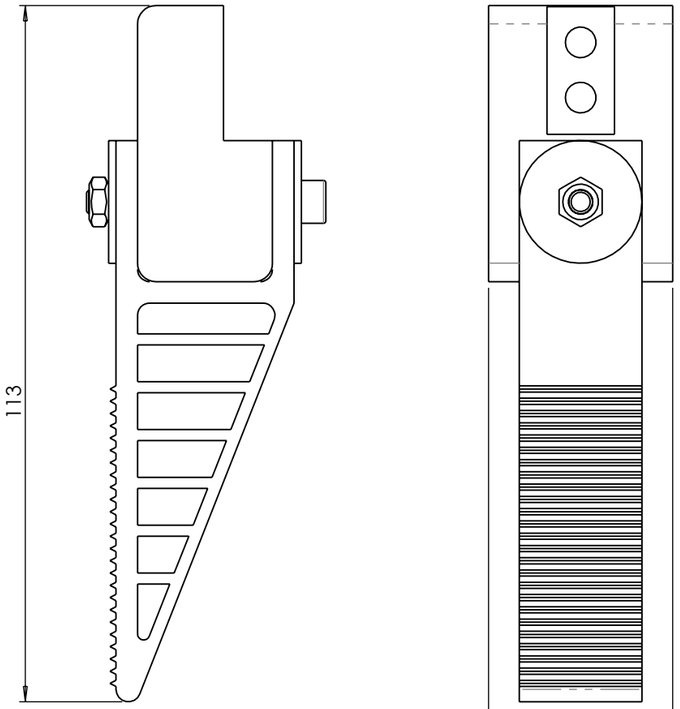
Poz.	Naziv dijela	Kom.	Crtež broj Norma	Materijal	Sirove dimenzije Proizvođač	Masa
1	Adapter Hvataljke	1		PLA		
2.1	Prst hvataljke V1-20	1		TPU		
3	Podloška 4x20	2	DIN 88104			
4	Vijak M4x25	1	DIN 912			
5	Matica M4	1	DIN 934			

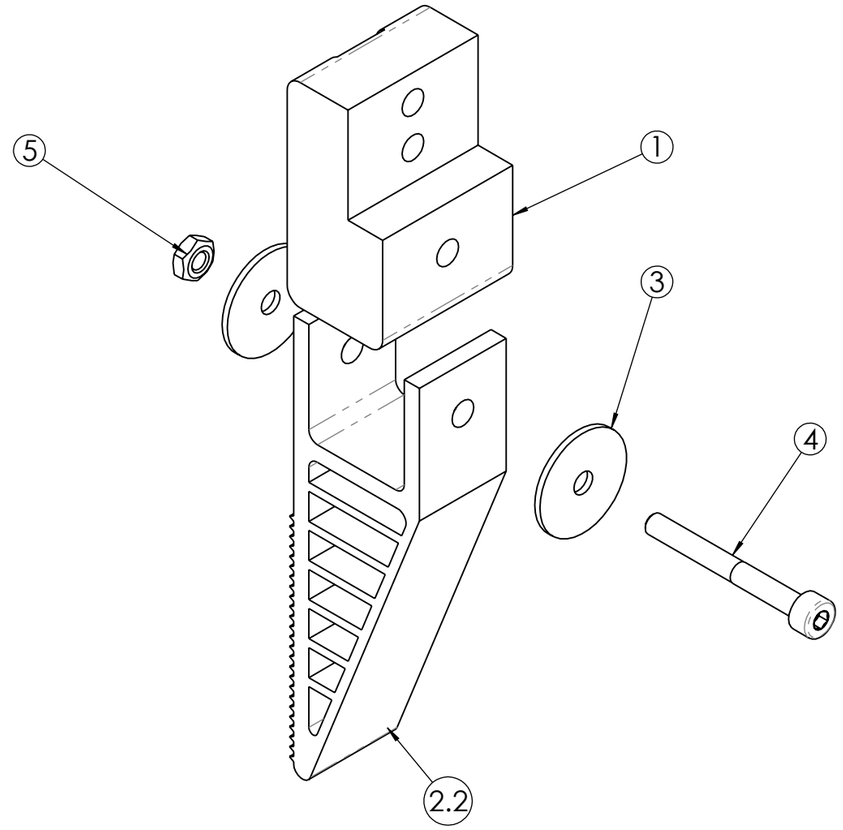
Broj naziva - code		Datum	Ime i prezime	Potpis	FSB Zagreb
	Projektirao	20.06.24.	Matija Zidarić		
	Razradio				
	Crtao	20.06.24.	Matija Zidarić		
	Pregledao				
	Mentor				

ISO - tolerancije	Objekt:	Objekt broj:
		R. N. broj:
	Napomena:	Kopija
	Materijal: / Masa:	
Mjerilo originala 1:1	Naziv: Adaptivni prst hvataljke 1	Pozicija:
	Crtež broj: APH-1-1-20	Format: A3 / Listova: 5 / List: 4

Design by CADLab

0 10 20 30 40 50 60 70 80 90 100





Poz.	Naziv dijela	Kom.	Crtež broj Norma	Materijal	Sirove dimenzije Proizvođač	Masa
1	Adapter Hvataljke	1		PLA		
2.2	Prst hvataljke V2-20	1		TPU		
3	Podloška 4x20	2	DIN 88104			
4	Vijak M4x25	1	DIN 912			
5	Matica M4	1	DIN 934			

Broj naziva - code		Datum	Ime i prezime	Potpis
	Projektirao	20.06.24.	Matija Zidarić	
	Razradio			
	Crtao	20.06.24.	Matija Zidarić	
	Pregledao			
	Mentor			

FSB Zagreb

ISO - tolerancije	Objekt:	Objekt broj:
		R. N. broj:
	Napomena:	Kopija
	Materijal: / Masa:	
Mjerilo originala 1:1	Naziv: Adaptivni prst hvataljke 2	Pozicija:
	Crtež broj: APH-2-1-20	Format: A3 / Listova: 5 / List: 5

Design by CADLab

0 10 20 30 40 50 60 70 80 90 100