Izrada korisničkog sučelja i algoritama za upravljanje gibanja Panda robotske ruke

Domitrek, Tin

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Tin Domitrek

Zagreb, 2022. godina.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

ZAVRŠNI RAD

Mentori:

Doc. dr. sc. Marko Švaco, dipl. ing.
Dr. sc. Filip Šuligoj, dipl. ing.

Student:

Tin Domitrek

Zagreb, 2022. godina.

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

Zahvaljujem se mentoru doc.dr.sc. Marku Švaci i komentoru dr.sc. Filipu Šuligoju na usmjeravanju, korisnim savjetima te podršci tijekom izrade završnog rada. Isto tako se zahvaljujem obitelji na podršci tijekom studija te djevojci i prijateljima koji su mi uljepšali studentske dane.

Tin Domitrek

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

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa: **602 – 04 / 22 – 6 / 1**	
Ur.broj: **15 - 1703 - 22 -**	

ZAVRŠNI ZADATAK

Student: **Tin Domitrek** JMBAG: **0035215426**

Naslov rada na hrvatskom jeziku: **Izrada korisničkog sučelja i algoritama za upravljanje gibanja Panda robotske ruke**

Naslov rada na engleskom jeziku: **User interface development and motion control programming for the Panda robotic arm**

Opis zadatka:

FCI (eng. „Franka Control Interface“) nudi brzu i izravnu dvosmjernu vezu niske razine s Franka Emika Panda robotskom rukom i šakom. Zahvaljujući mogućnosti upravljanja u stvarnom vremenu (s brzinom komunikacije od 1 kHz), idealno je sučelje za istraživanje upravljanja na niskoj razini. Zadatak je programirati C++ klase koje omogućavaju upravljanje robotom na niskoj razini (eng. „low-level control“) i slanje naredbi u realnom vremenu za vođenje robota na tri načina:

- Vođenje točka-točka sa slobodnim prijelazom: doseći zadane vrijednosti unutarnjih koordinata bez obzira na putanju,
- Slijedno vođenje točka-točka: doseći zadane vrijednosti unutarnjih koordinata bez obzira na putanju gdje sve osi započinju i završavaju gibanje u isto vrijeme
- Vođenje po vektoru položaja i interpolacije (linearno gibanje): linearno gibanje postiže se s interpolacijom, gdje kontroler određuje odgovarajuće brzine vrtnji svake osi robota.

Zadatak je potrebno napraviti koristeći Linux operativni sustav s „real-time“ kernelom i C++ programskom podrškom. Razvijeno korisničko sučelje treba omogućiti upisivanje različitih parametara za izvršavanje gibanja robota. Razvijenu programsku podršku potrebno je verificirati na Franka Emika Panda robotskoj ruci dostupnoj u Laboratoriju za autonomne sustave.

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

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
30. 11. 2021.	**1. rok:** 24. 2. 2022. **2. rok (izvanredni):** 6. 7. 2022. **3. rok:** 22. 9. 2022.	**1. rok:** 28. 2. – 4. 3. 2022. **2. rok (izvanredni):** 8. 7. 2022. **3. rok:** 26. 9. – 30. 9. 2022.
Zadatak zadao:	Komentor:	Predsjednik Povjerenstva:
Doc. dr. sc. Marko Švaco	Dr. sc. Filip Šuligoj	Prof. dr. sc. Branko Bauer

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

SAŽETAK

U ovom radu se opisuje postupak razvoja korisničkog sučelja za upravljanje na niskoj razini sa Franka Emika Panda robotskom rukom koja ima 7 stupnjeva slobode. Najprije se rješava problem spajanja gdje se moraju zadovoljiti zahtjevi za upravljanjem na niskoj razini, koji se rješava instalacijom real-time kernela na Linux Ubuntu sustavu i postavljanjem statičkih IP adresa kojim su povezani robot i radna stanica. Koristi se FCI (Franka Control Interface) izdan od strane Franka Emike koji služi kao posrednik između robota i sučelja, te libfranka C++ knjižnica koja sadrži kolekciju funkcija i kontrolera za upravljanje robotskom rukom i hvataljkom. Korištenjem Qt Creator alata koji se temelji na C++ programskom jeziku i libfranka knjižnice razvijeno je korisničko sučelje sa mogućnošću stvaranja sekvencijalnih zadataka koji dopuštaju otvaranje/zatvaranje hvataljke, point-to-point gibanje, linearno gibanje, gibanje zglobova, te kontrolu impedancije. Problem pozicije rješava se izračunom inverzne kinematike robotske ruke unutar algoritma, a orijentacija uz pomoć funkcija libfranke i slerpa.

Ključne riječi: Franka Emika Panda, libfranka, FCI, inverzna kinematika, point-to-point, linearno

SUMMARY

This paper describes the process of developing a low-level user control interface system for Franka Emika Panda robotic arm, which has 7 degrees of freedom. Firstly, the connection problem where low-level control requirements must be met is addressed, which is solved by installing a real-time kernel on a Linux Ubuntu system and setting static IP addresses to which the robot and workstation are connected. The FCI (Franka Control Interface) released by Franka Emika is used, which serves as an intermediary between the robot and the interface, and a libfranka C++ library which contains a collection of functions and motion controllers for controlling the robotic arm and hand. Using the Qt Creator tool based on the C++ programming language and the libfranka library, a user interface was developed with the possibility of creating sequential tasks that allow opening/closing of the gripper, point-to-point motion, linear motion, joint motion, and impedance control. The position problem is solved by calculating inverse kinematics of the robot hand inside an algorithm, and the orientation with the help of libfranka functions and slerp.

Key words: Franka Emika Panda, libfranka, FCI, inverse kinematics, point-to-point, linear

1. UVOD

Franka Emika Panda je robotska ruka razvijena u Njemačkoj koja se koristi u industriji, ali ponajviše u istraživanju i razvoju zbog svoje preciznosti i lakoće upravljanja. Dolazi sa Desk korisničkim sučeljem koje omogućava izradu raznoraznih kretnji po potrebi korisnika. Međutim, zanimljivije od Desk-a je FCI (Franka Control Interface), koje omogućava direktno upravljanje robotom na niskoj razini. Upravljanje na niskoj razini je vrsta upravljanja gdje očekujemo brze, precizne i jednostavne kretnje i zadatke. Moguće je samo uz dobru i brzu vezu sa upravljačkom stanicom (PC) koju dobivamo izradom sučelja u real-time kernel sustavu i korištenjem statičkih IP adresa. Uz FCI dolazi i libfranka, radi se o C++ knjižnici koja sadrži kolekciju funkcija, kontrolera i općih naredbi koji omogućuju upravljanje robotom. Ono što je zanimljivo u vezi libfranke je mogućnost izrade vlastitih jedinstvenih upravljačkih kontrolera i petlji. S obzirom na to što je zadatak ili što zahtijeva tvrtka, uz pomoć libfranke omogućeno je razraditi i riješiti bilo koji inženjerski problem automatizacije ili upravljanja. U sklopu ovog rada izrađeno je grafičko korisničko sučelje za upravljanje na niskoj razini sa Panda robotskom rukom uz pomoć FCI-a i libfranke. Qt je izabran kao alat za izradu sučelja zbog svoje jednostavnosti i zbog programiranja u C++ programskom jeziku. Algoritmi za upravljanje su izrađeni uz pomoć libfranka naredbi i uz preuzetu podršku za izračun inverzne kinematike.

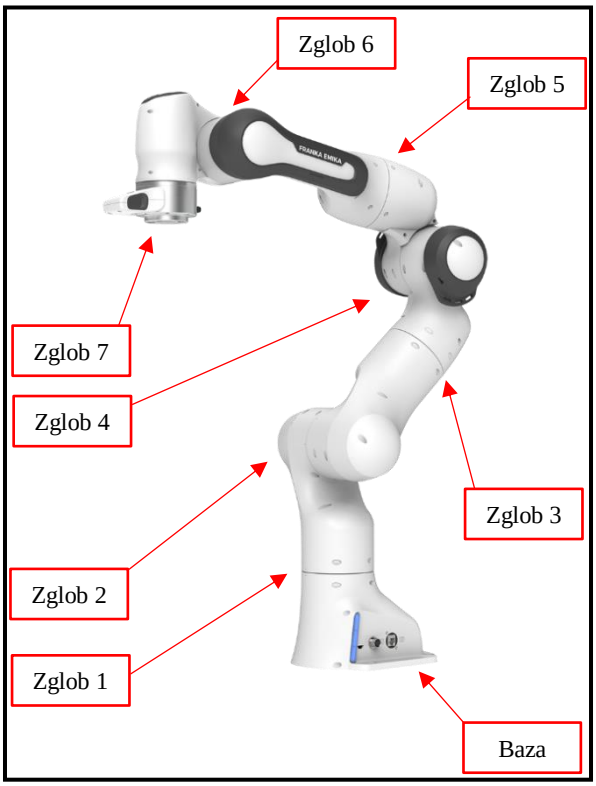
2. FRANKA EMIKA PANDA ROBOTSKI SUSTAV ZA UPRAVLJANJE

Za shvaćanje procesa spajanja robota na mrežu, te kasnije i mogućnosti upravljanja robotom najprije je potrebno upoznati se sa hardverom kojim se upravlja. U ovom se radu koristi Franka Emika Panda robotska ruka, razvijena u Njemačkoj. S njom dolazi kontrolna kutija, robotska ruka, hvataljka, te gumb za stopiranje kao najbitnije komponente. Robot se nalazi u laboratoriju CRTA-e na postolju gdje je spojen Ethernet kablom na kontrolnu kutiju, te statičkom IP adresom na mrežu. Kontrolna kutija je zatim također Ethernet kablom spojena na radnu stanicu (PC).

2.1. Robotska ruka

Robotska ruka ima 7 stupnjeva slobode sa senzorima zakretnog momenta na svakom zglobu, što omogućuje podesivu krutost i naprednu kontrolu okretnog momenta. Nosivost robota je 3 kg, ima doseg od 855 mm i pokrivenost radnog prostora od 94,5%. Koristi se većinom u istraživanju i edukaciji, te ima mnoge mogućnosti poput prepoznavanja puta alata, precizan pick and place, definiciju trajektorija, mogućnost preciznog sklapanja komponenti, testiranje GUI aplikacija, detekciju kolizija itd. Senzori ugrađeni u svaki zglob daju informacije o količini momenta u svakoj osi u bilo kojem trenutku, te na temelju toga robot pokreće sigurnosne funkcije koje mu omogućavaju izbjeći potencijalne sudare s okolinom ili nanošenje ozljeda čovjeku. Robot prepoznaje kada su zadana potencijalno opasna ili nemoguća gibanja te se pravovremeno zaustavlja.

Slika 1. Logo tvrtke Franka Emika [4]

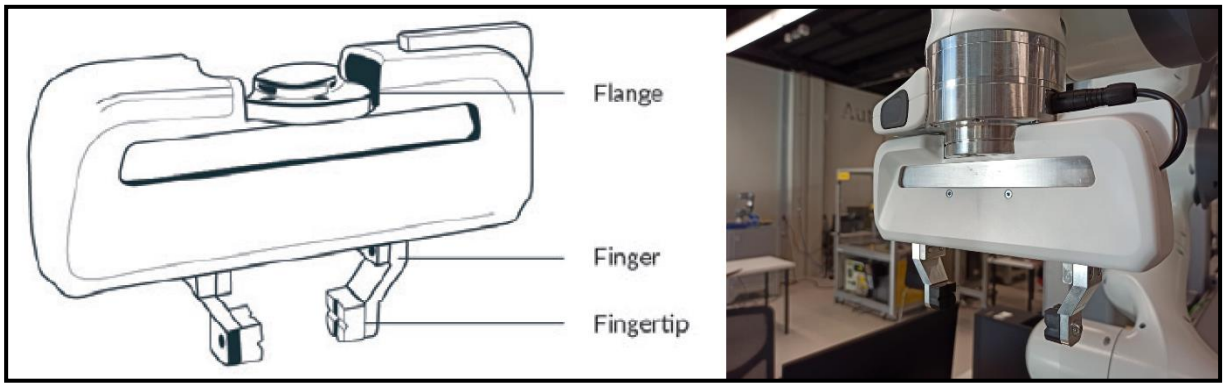


Slika 2. Robotska ruka [4]

Slika 3. Robot i radna stanica u labosu

2.2. Hvataljka

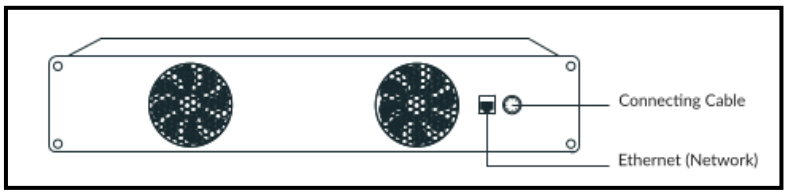
Hvataljka Franka Emika Panda robota ima kontinuiranu silu hvatanja od 70 N (maximalnu silu od 140 N) te može podići do 3 kilograma tereta. Maksimalna širina hvataljke je 80 mm što joj omogućava podići predmete srednjih veličina. Maksimalna brzina svakog prsta hvataljke je 50 mm/s. U sučelju će biti omogućeno upravljanje hvataljkom (otvaranje i zatvaranje) kao i dodavanje tih funkcija u program gibanja.



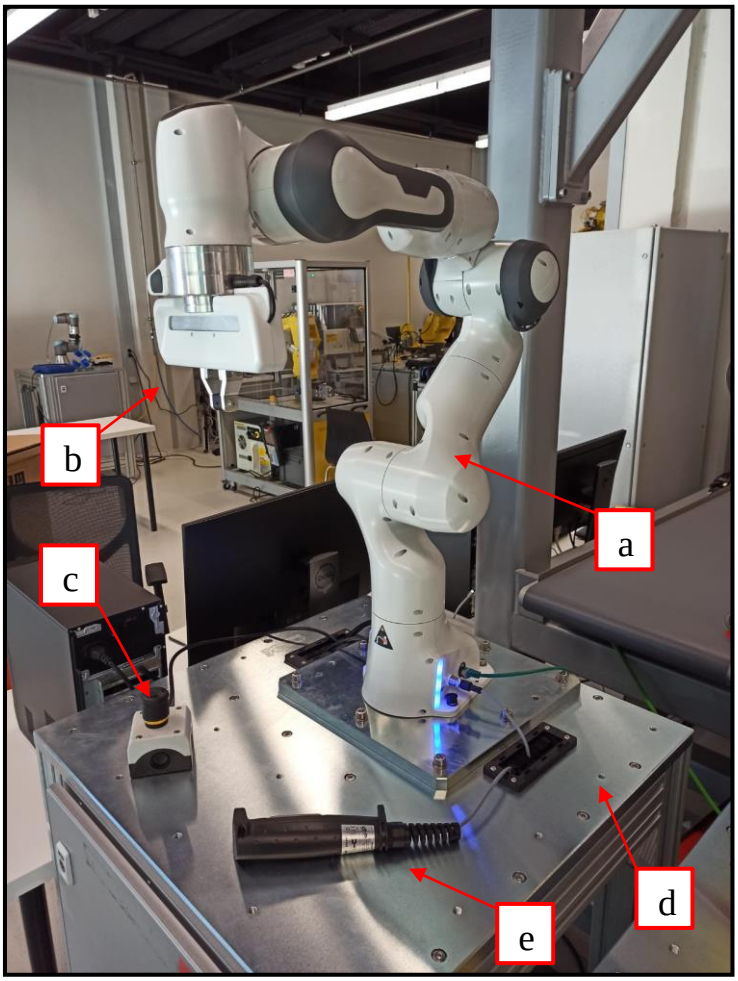
Slika 4. Shema hvataljke i hvataljka spojena na robota u labosu

2.3. Sustav

Kao što je i prije spomenuto, robot je pričvršćen na postolje i s kontrolnom kutijom je povezan preko Ethernet kabla, kutija je također drugim kablom spojena na radnu stanicu (PC). Povezivanje preko kontrolne kutije nam omogućuje upravljanje pomoću FCI-a. Radna stanica i robot moraju biti na istoj mreži, a to se postiže korištenjem statičkih IP adresa koje su postavljene u labosu. Hvataljka je pričvršćena na krajnju jedinicu robota (eng. end effector). Sve ostale komponente spojene su kablovima na bazu robotske ruke.



Slika 5. Shema kontrolne kutije (eng. Control) [4]



Slika 6. Robotska stanica: a) Robotska ruka, b) Hvataljka, c) Gumb za stopiranje, d) Postolje, e) Vanjski uređaj za upravljanje

2.4. Sigurnost

Sigurnost je veoma bitna jer je oprema sofisticirana i skupa. Oštećenje opreme mora se spriječiti pod svaku cijenu.

2.4.1. Osnove

Treba pripaziti na sljedeće stvari:

1. Provjeriti je li ruka postavljena na stabilno postolje i da se ne može prevrnuti, čak ni pri izvođenju brzih pokreta ili naglih zaustavljanja.

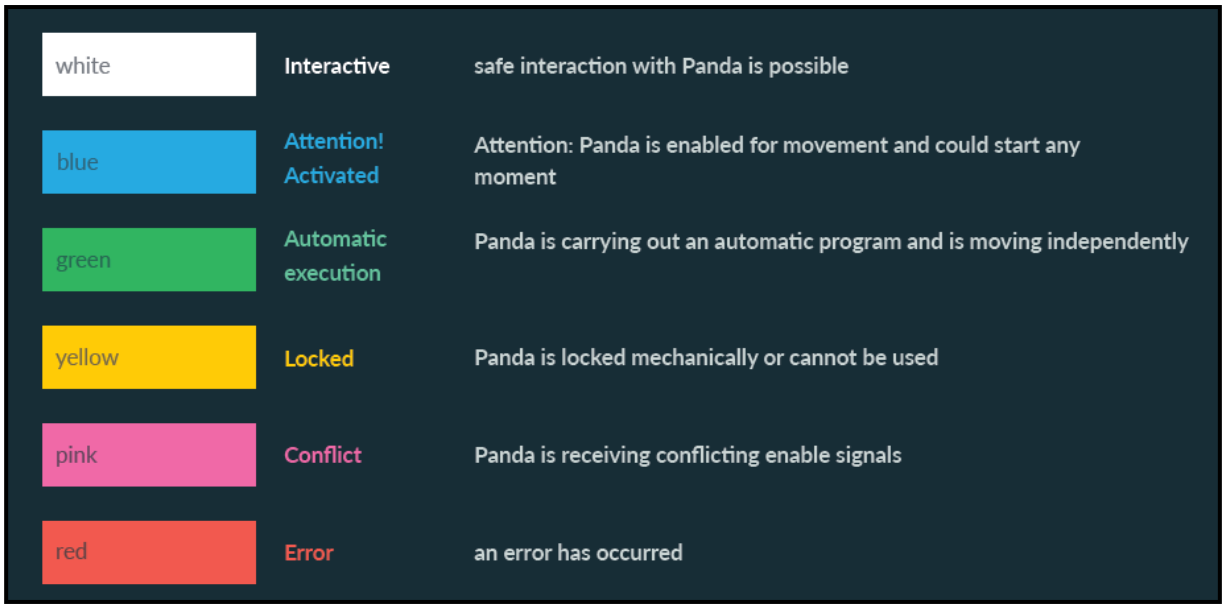
2. FCI daje potpunu i isključivu kontrolu nad rukom i hvataljkom. To znači da se Desk ne može koristiti u isto vrijeme kao i FCI.

3. Žuto svjetlo znači da su zglobovi zaključani, te se moraju otključati u Desk-u.

4. Bijelo svjetlo znači da je robot stopiran stop gumbom i da je u guiding mode-u (to znači da se može micati i voditi), te je potrebno opet pritisnuti stop gumb kako bi dobili plavo svjetlo na bazi tj. kako bi mogli nastaviti s naredbama gibanja i upravljanjem FCI-om.

5. U slučaju da mislimo da će doći do kolizije moramo brzo pritisnuti stop gumb kako bi zaustavili gibanje.

6. Crveno svjetlo na bazi znači ozbiljna šteta na hardveru i potrebno je provjeriti postoji li trajna šteta na robotu i javiti se nadležnoj osobi.



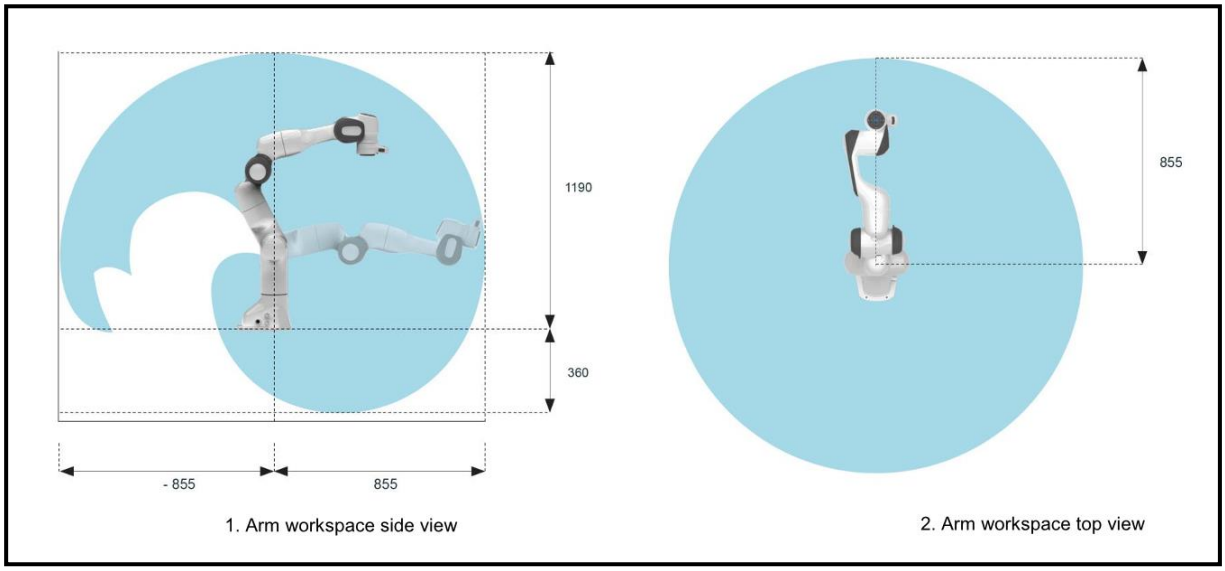
Slika 7. Slika statusnih indikatora iz Franka Emika priručnika [4]

2.4.2. Limitacije i radni prostor robota

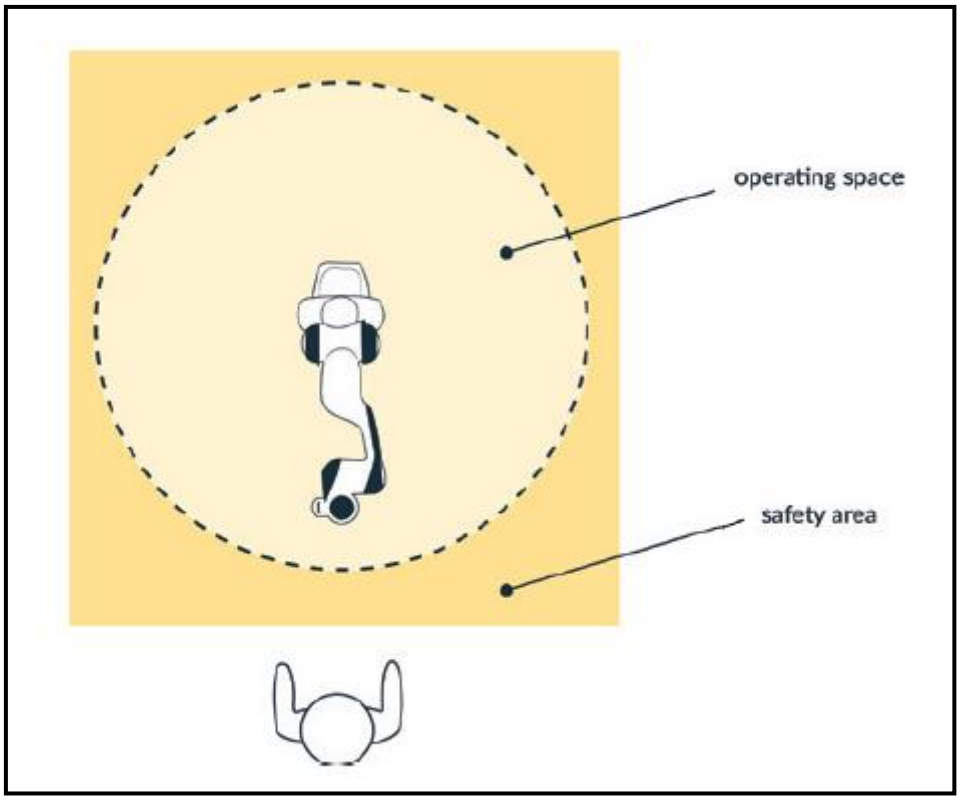
Preporučeno je poštivati limitacije i osigurati slobodan prostor oko robota u krugu 1 metar. Sučelje dopušta kreiranje gibanja u zadanom vremenu ili zadanom brzinom. Prije kreiranja i pokretanja programa u sučelju razmisliti da li je robot sposoban napraviti zadane kretnje visokim brzinama ili u kratkim vremenskim intervalima. Limitacije su u tablici (Tablica 1).

	Min.	***Max.***
X	-700 mm	700 mm
Y	-700 mm	700 mm
Z	200 mm	800 mm
Zglob 1	-166 °	166 °
Zglob 2	-101 °	101 °
Zglob 3	-166 °	166 °
Zglob 4	-176 °	-4 °
Zglob 5	-166 °	166 °
Zglob 6	-1 °	215 °
Zglob 7	-166 °	166 °
Brzina	> 0 rad/s	2 rad/s
Vrijeme	1 s	6 s

Tablica 1. Limitacije robota



Slika 8. Radni prostor robota [4]



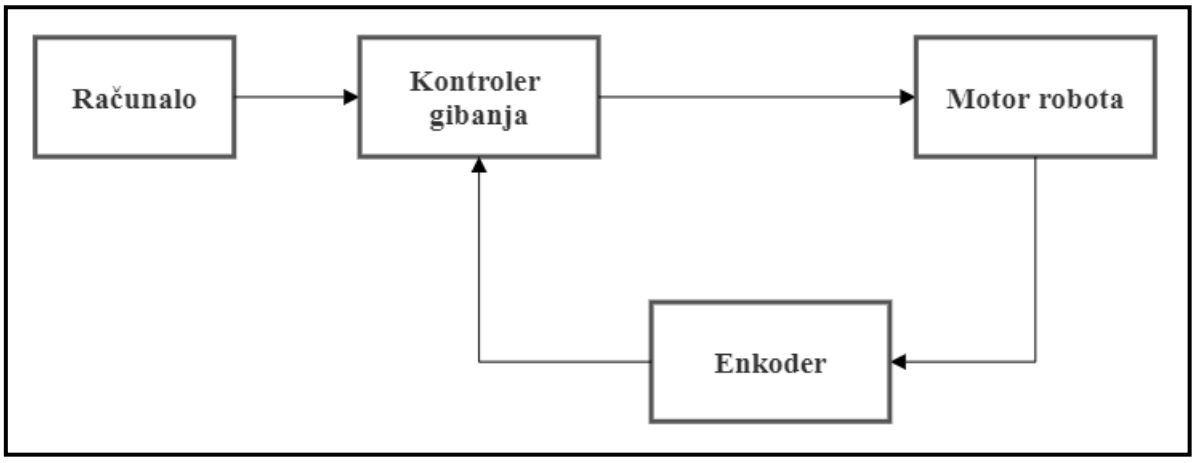
Slika 9. Sigurno područje rada [4]

3. UPRAVLJANJE NA NISKOJ RAZINI I SOFTVERSKA PODRŠKA

U sklopu završnog rada koristi se više različith softvera iz različitih dijelova industrije. Softver je usko povezan i omogućuje izradu jedinstvenog sučelja za upravljanje Franka Emika robotskom rukom i hvataljkom na niskoj razini.

3.1. Općenito o upravljanju

U robotici se upravljanje dijeli na tri razine: niska, srednja i visoka razina upravljanja. Niska razina upravljanja (eng. Low-level control) obuhvaća najjednostavnije i opće oblike gibanja koja se izvršavaju u kratkom vremenu npr. gibanje od jedne točke do druge ili gibanje zglobova u određene koordinate. Srednja razina uzima u obzir konkretne probleme poput računanja i navigiranja do željenog položaja ili podizanja predmeta. Visoka razina se temelji na rješavanju diskretnih programskih problema u duljem vremenskom razdoblju, uz pažljivo planiranje puta i zadataka. Sučelje razvijeno u sklopu završnog rada koristi upravljanje na niskoj razini, putem kojeg omogućava različite vrste gibanja u koordinate unutar radnog prostora robota u kratkom vremenu.



Slika 10. Blok dijagram upravljanja na niskoj razini

3.1.1. Zahtjevi za low-level control upravljanjem

Prije početka izrade sučelja i upravljanja robotom pomoću libfranka knjižnice i FCI-a potrebno je ispuniti zahtjeve za low-level kontrolom i slanjem naredbi vezom od 1 kHz. Franka Panda robot šalje podatke brzinom 1 kHz, te je važno konfigurirati radnu stanicu na način da se minimalizira kašnjenje (eng. latency). [4] Moramo garantirati da sljedeće vremenske vrijednosti budu manje od 1 ms:

1. RTT (round trip time tj. vrijeme odziva) između radne stanice (PC) i FCI.
2. Execution time tj. vrijeme izvršavanja našeg motion generatora ili kontrolne petlje.
3. Vrijeme potrebno da robot procesira naše podatke i korak internog kontrolera.

3.1.2. Detaljno postavljanje radne stanice

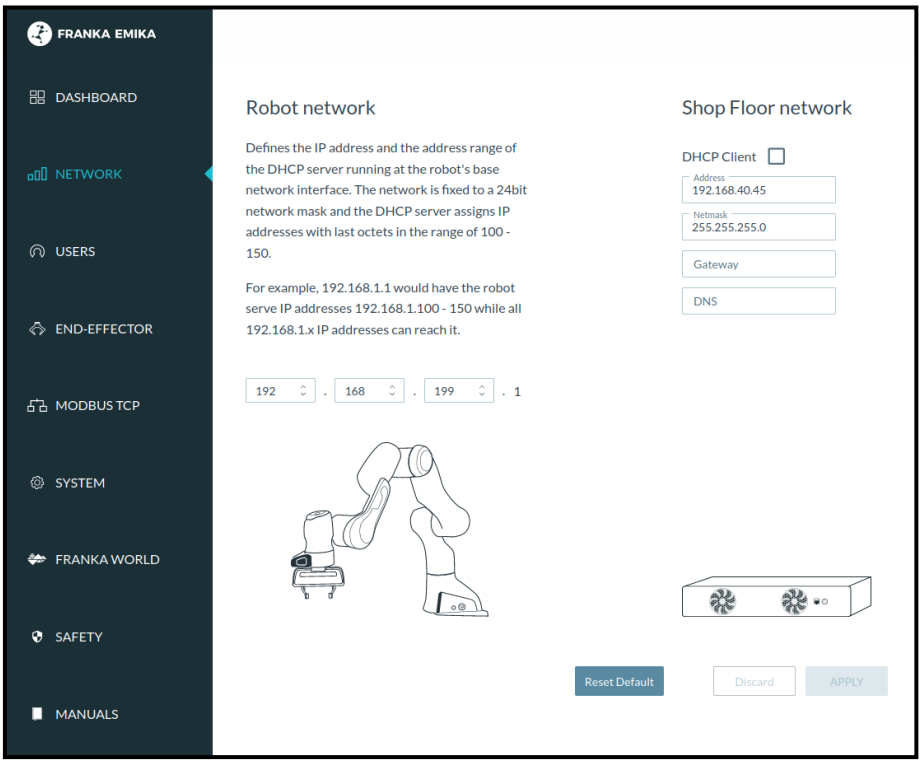
Zahtjevi za upravljanjem se moraju zadovoljiti, a to se postiže tako da kontrolni program radi s prioritetom u realnom vremenu pomoću real-time kernela verzije 5.9.1.-rt20 kojeg se mora instalirati, te ga postaviti kao prioritet pokretanja na Linux Ubuntu 20.04 sustavu.

Sljedeće se mora osigurati brza i stabilna veza između robota i radne stanice. Robot i radna stanica moraju biti na istoj mreži, a najlakši način da se to postigne je da se koriste statičke IP adrese (Slika 12 i 13). Radna stanica kojom se upravlja robotom putem FCI-a mora uvijek biti spojena Ethernet kablom u LAN port kontrolne kutije, a ne LAN port robotske ruke (spajanjem na robotsku ruku možemo koristiti Desk za upravljanje).

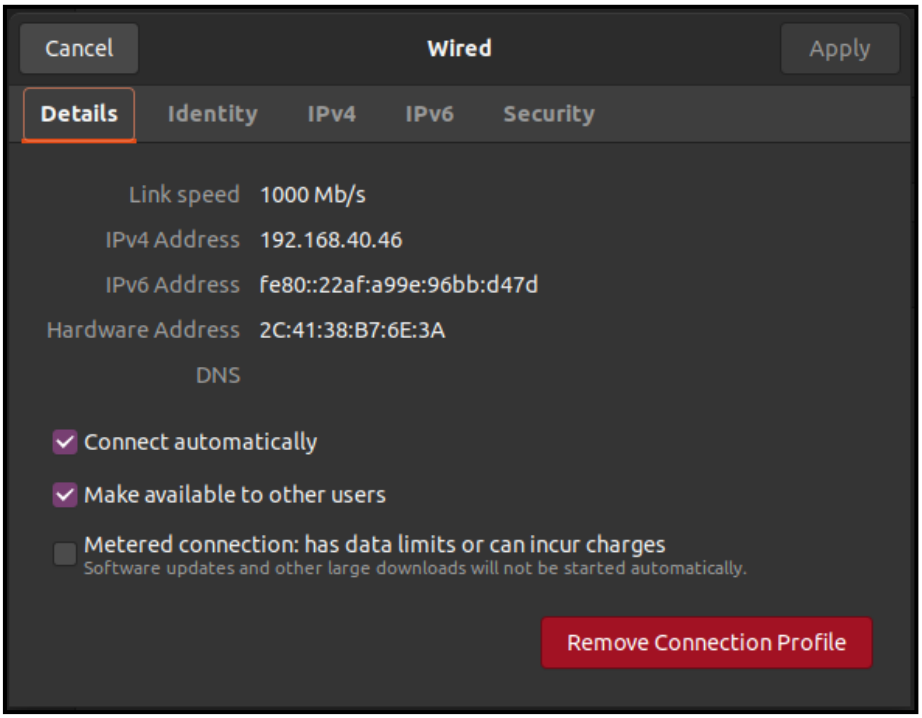
Robot se pokreće na kontrolnoj kutiji te se može korisiti nakon pojave žutog svjetla na bazi. Desk-u se pristupa putem sljedećeg linka: ***http://<fci-ip>/desk*** (gdje <fci-ip> predstavlja statičku IP adresu robota), zatim je potrebno otključati zglobove na gumbu *unlock joints* unutar Desk-a i pričekati plavo svjetlo na bazi, te u padajućem izborniku postavka odabrati *Activate FCI*. Kada su učinjeni gore navedeni koraci omogućeno je kontrolirati robota pomoću FCI-a. Za početak je dobro pokrenuti par primjera (eng. examples) iz libfranka knjižnice da se zna da sve radi kako treba i tek onda krenuti sa pokretanjem kompleksnijih gibanja i upravljanja sučeljem. Najlakši način za to učiniti je doći u mapu gdje se nalaze primjeri libfranke i pokrenuti *echo_robot_state* pomoću sljedeće naredbe u terminalu: *./examples/echo_robot_state <fci-ip>*

Vezu sa robotom se provjerava pomoću sljedeće naredbe: *ping <fci-ip>*

Za sve nejasnoće posjetiti Franka Control Interface (FCI) Documentation.



Slika 11. Postavljanje IP adrese robota u Desk-u

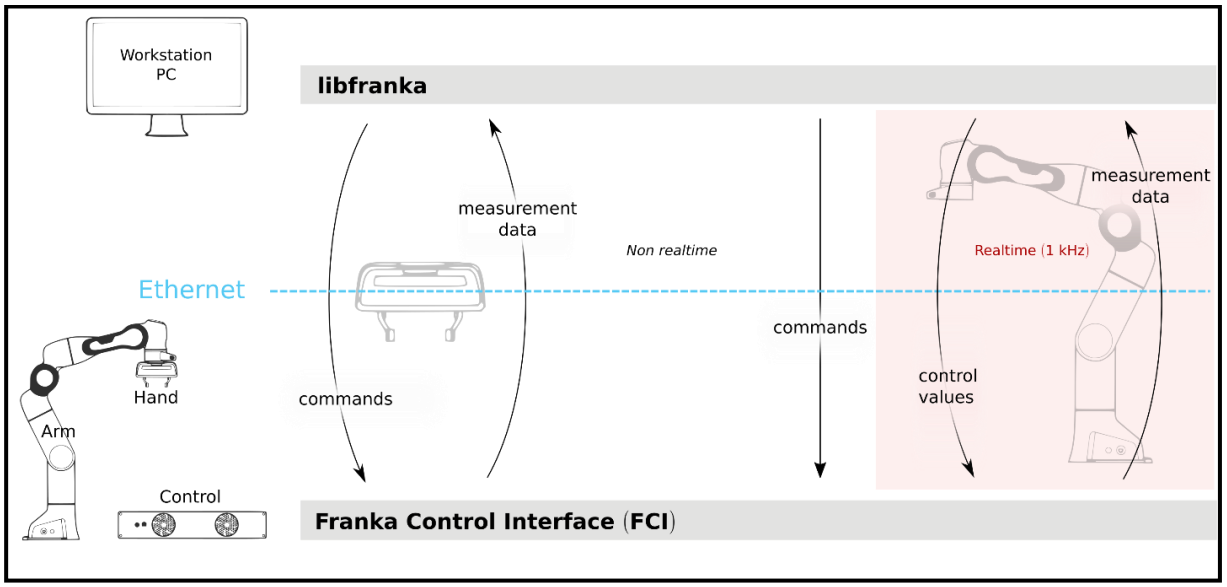


Slika 12. Postavljanje IP adrese radne stanice

3.1.3. Franka Control Interface

Franka Control Interface (FCI) služi kao posrednik između robota i razvijenog korisničkog sučelja za upravljanje. Omogućuje brzu i izravnu dvosmjernu vezu niske razine s robotskom rukom i hvataljkom. [4] Također pruža trenutni status robota i omogućuje njegovu izravnu kontrolu s pomoću računala. FCI daje očitanja pozicija zglobova, njihove brzine i vrijednosti okretnog momenta u zglobovima. Pruža i vanjske vrijednosti sila i momenata i niz informacija vezanih za kontakt s okolišem i kolizije. Sve te informacije su bitne kako bi robot preko programskog koda radio na pravilan i siguran način.

Paket FCI je potrebno instalirati/sinkronizirati sa Franka World stranice na kontrolnu kutiju. Stoga je potrebno imati FCI licencu na korisničkom profilu tvrtke/sveučilišta i imati registriranog kontrolera na tom računu.



Slika 13. Arhitektura Franka Control Interfacea [4]

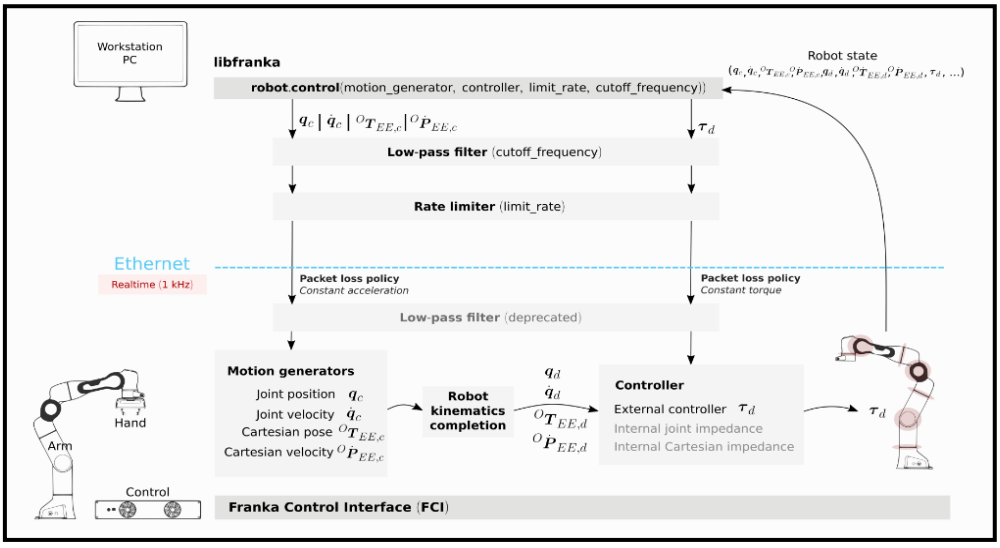
3.2. Softverska podrška

Programiranje se odvijalo u C++ programskom jeziku. Koristila se libfranka C++ knjižnica izdana od strane Franka Emika-e, a Qt Creator alat je korišten za kreiranje grafičkih korisničkih sučelja.

3.2.1. Libfranka i C++

Libfranka je C++ knjižnica koja implementira klijentsku stranu FCI-a za stvaranje jedinstvenih kontrolnih petlji i gibanja. Upravlja mrežnom komunikacijom s kontrolom i pruža sljedeće mogućnosti:

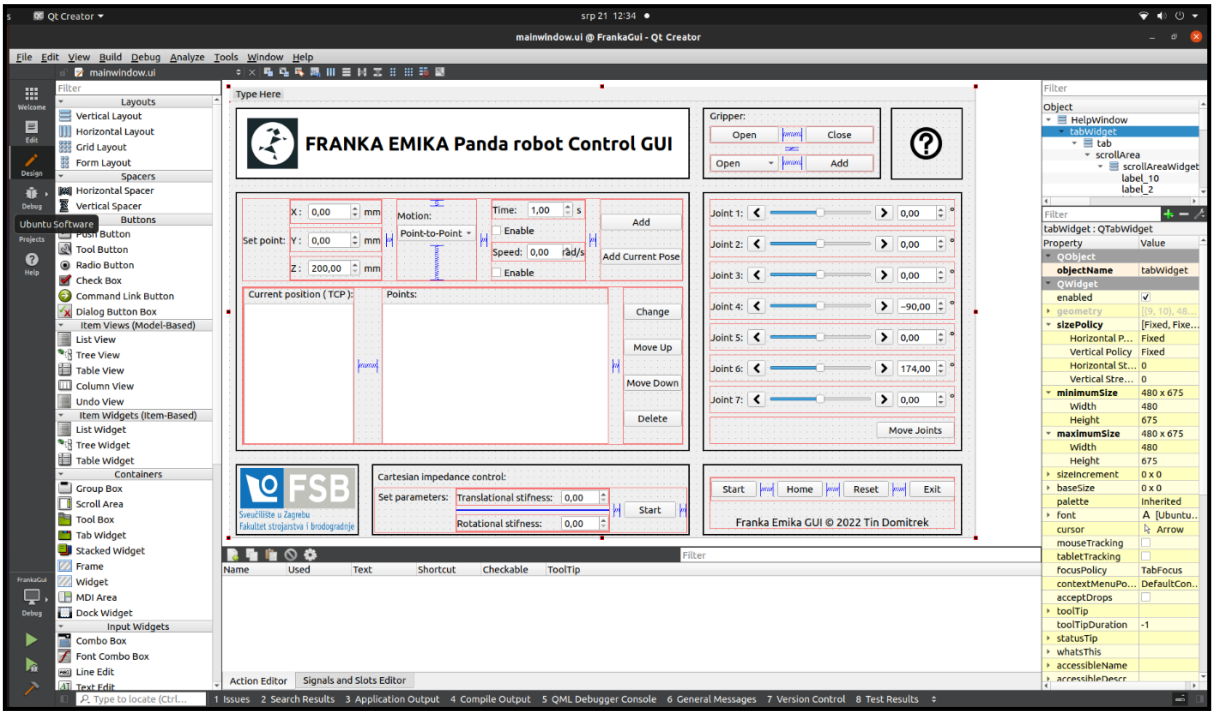
1. Izvršavanje naredbi koje nisu u stvarnom vremenu za upravljanje rukom i konfiguraciju parametara hvataljke. To uključuje naredbe poput *move*, *grasp*, *stop* za upravljanje hvataljkom, te npr. *setCollisionBehavior*, naredba koja postavlja pragove za detekciju kontakta i sudara za ruku.
2. Izvršavanje naredbi u stvarnom vremenu za pokretanje vlastitih kontrolnih petlji. Generatori gibanja koji definiraju gibanje robotske ruke u kartezijske koordinate i kontroleri, koji definiraju direktno slanje momenata gibanja u zglobove. Primjer: *generate_joint_velocity_motion* .
3. Očitavanje stanja robota za dobivanje senzorskih podataka vezom od 1 kHz, to uključuje pozicije zglobova, poziciju u kartezijskim koordinatama, orijentaciju end effectora.
4. Izračunavanje željenih kinematičkih i dinamičkih parametara.



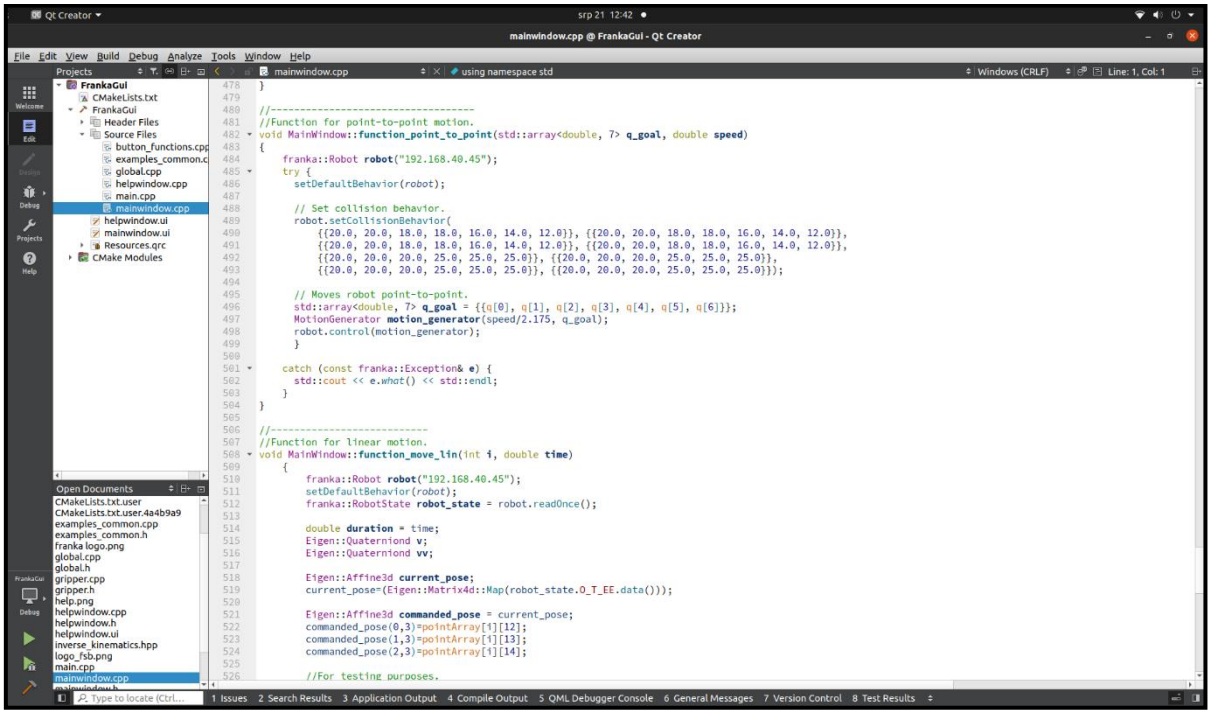
Slika 14. Primjer kontrolne petlje u stvarnom vremenu [4]

3.2.2. Qt

Qt je softver za jednostavnu izradu grafičkih korisničkih sučelja. Softver je višeplatformski što znači da se sučelja izrađena u Qt-u mogu pokretati na različitim platformama. Pisan je u C++ jeziku i ima uključen GCC (GNU Compiler Collection) za kompajliranje aplikacija. [5]



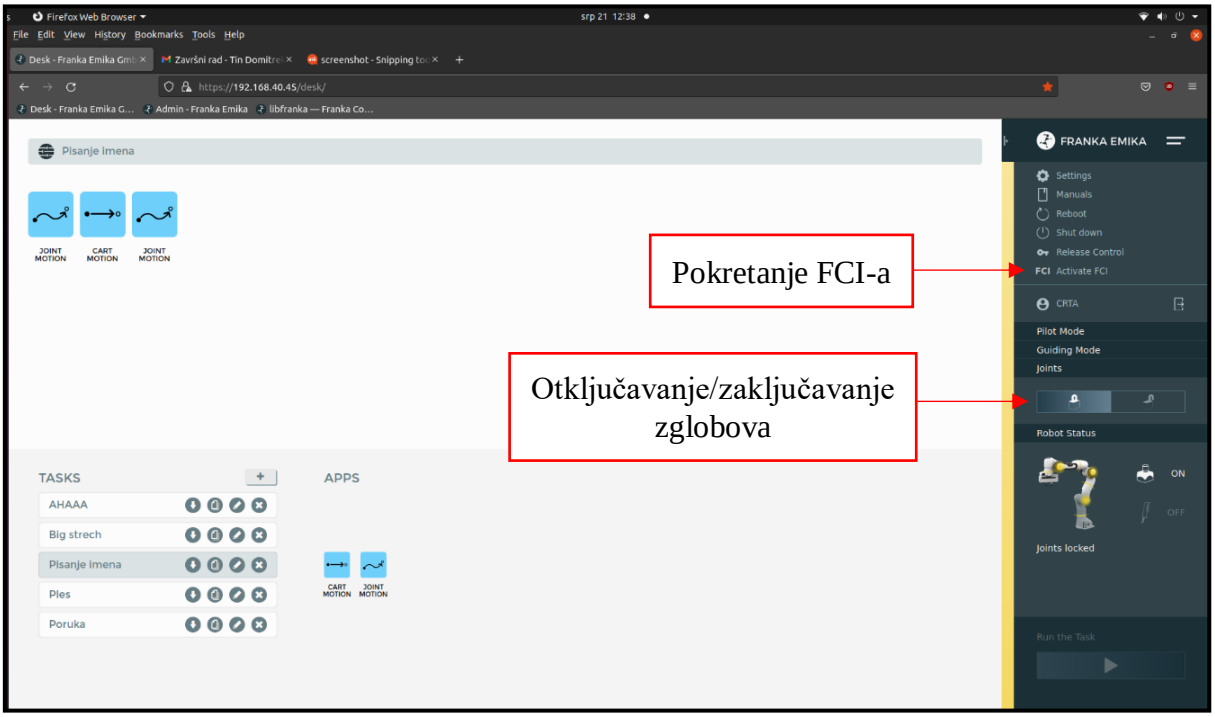
Slika 15. Izgled GUI dizajner prozora



Slika 16. Izgled prozora za programiranje

3.2.3. Desk

Desk je intuitivno web-based korisničko sučelje za programiranje Panda robota. Pomoću njega je moguće kreirati zadatke i sekvence zadataka koji dopuštaju razna gibanja i mnoge druge mogućnosti poput hvatanja, slaganja, pritiskanja itd. [4] Desk-u se također može pristupiti pomoću sljedećeg URL-a: ***https://robot.franka.de***. U smislu ovog rada, Desk nam isključivo služi samo za pokretanje FCI-a i otključavanje/zaključavanje zglobova.



Slika 17. Desk korisničko sučelje na webu

4. SUČELJE

U ovom poglavlju su detaljno opisane funkcionalnosti sučelja. Odlučeno je izraditi sučelje jednostavnog izgleda kako bi fokus bio na funkcionalnosti i gibanjima.

4.1. Glavni prozor

U sučelju se nalaze sljedeći izbornici: point meni, gripper meni, joint meni i impedance meni. U donjem desnom uglu glavnog prozora su funkcije Start koja pokreće gibanje, Home koja vraća robota u početnu poziciju (pozicija je uzeta iz libfranka knjižnice), Reset koja resetira cijelo sučelje, time i briše točke u *Points* prozoru, te Exit kojom se izlazi iz sučelja. Najvažnije varijable su *point_counter* koja broji koliko imamo zadanih točki gibanja, te *row* kao pomoćna varijabla kod gumba Delete, Move Up/Down u koju spremamo poziciju reda u kojoj se nalazi odabrana točka iz *Points* prozora koja se zatim briše ili pomiče.

4.1.1. Point meni

Point meni sadrži prostor za upisivanje x, y i z koordinata. U padajućem izborniku se odabire vrsta gibanja (point-to-point ili linearno), te se *CheckBox-om* odabire da li će se gibanje izvoditi u zadanom vremenu ili zadanom brzinom. Korisnik upisuje vrijednosti koordinata željene pozicije gibanja i vrijednosti vremena ili brzine (ovisno o tome što je odabrano). Nakon klika gumba *Add*, program uzima vrijednosti iz *TextBox-ova* (prozor za upisivanje brojčanih vrijednosti) te ih sprema u globalne nizove nazvane *pointArray* (za koordinate x, y i z), *motionArray* (sprema vrstu gibanja) i *timeSpeedValueArray* (sprema vrijeme ili brzinu). Odabrane vrijednosti koordinata, gibanja i vremena/brzine se također upisuju u *Points* prozor kao točka gibanja. *Points* prozor sprema sve točke u koje će se robot sekvencijalno gibati. Točke se mogu mijenjati, pomaknuti za jednu poziciju gore i dolje ili brisati.

Klikom na gumb *Add Current Pose*, program uzima trenutnu poziciju i orijentaciju robota pomoću sljedećeg koda iz libfranka knjižnice:

```
1  franka::Robot robot(„192.168.40.45“);
2  setDefaultBehavior(robot);
3  franka::RobotState robot_state = robot.readOnce();
```

Prva linija koda deklarira robota kojeg ćemo koristiti (IP adresu), a druge dvije očitavaju stanje robota (njegovu poziciju i orijentaciju) i spremaju u niz *robot_state* sa 16 varijabli (transformacijska matrica). Zatim se dobivena orijentacija i pozicija, te uvjeti, spremaju na isti način kao i kod gumba *Add* u prije spomenute nizove, te *Points* prozor.

Current position (TCP) prozor prikazuje nam trenutne koordinate i transformacijsku matricu end effector-a tj. tool center pointa. Transformacijska matrica se dobiva istim kodom kao i kod gumba *Add Current Pose*. Prozor se nakon svakog gibanja resetira i upisuju se nove vrijednosti trenutne pozicije. Uz pomoć ovog gumba i *guiding mode-a* (pokreće se klikom na Stop gumb, bijelo svjetlo na bazi) moguće je voditi robota rukom, te dodati željene pozicije u program bez da znamo točne koordinate u prostoru.

4.1.2. Gripper meni

Gripper meni sadrži jednostavne funkcije za otvaranje i zatvaranje hvataljke. Gibanje se izvodi pomoću sljedećeg koda iz libfranka knjižnice:

```
1   franka::Gripper gripper(„192.168.40.45“);
2   double grasping_width = 0.08;
3   double speed = 0.5;
4   gripper.move(grasping_width, speed);
```

Prva linija koda deklarira robota (hvataljku), druga i treća linija određuju širinu zahvata (u ovom slučaju otvaranje, maksimalna širina 80 milimetara) tj. brzinu izvođenja gibanja (od 0 do 1, dakle u ovom slučaju 25 mm/s jer je maksimalna brzina 50 mm/s), a zadnja linija sadrži kontrolnu funkciju gibanja (motion generator).

Funkcije se također mogu dodati u program i ispisati u *Points* prozoru. Koristimo novi globalni niz nazvan *conditionArray* koji nam koristiti kao uvjet u *if()* funkciji glavnog gibanja (Start funkcija) na način da deklariramo gibanje koje želimo (1 za otvaranje hvataljke, 2 za zatvaranje hvataljke, 3 za joint motion, 4 za point-to-point /linear motion). Te vrijednosti se spremaju u *conditionArray* kod bilo kojeg dodavanja točke u *Points* prozor.

4.1.3. Joint meni

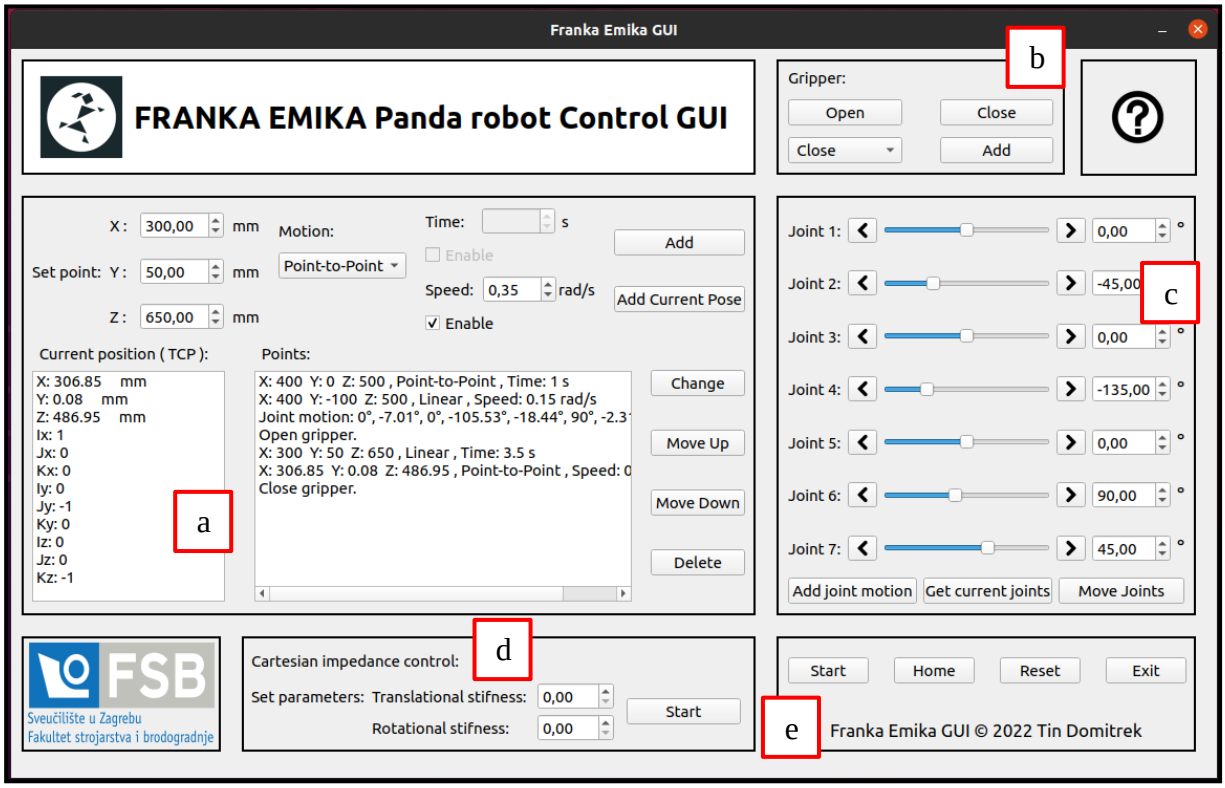
Joint meni dopušta pomicanje svih 7 zglobova u bilo koju vrijednost u stupnjevima unutar dopuštenih limitacija. Zadavanje vrijednosti moguće je pomoću slidera, gumba lijevo i desno ili upisivanjem vrijednosti u *TextBox*. Gumb *Move joints* pokreće gibanje u zadane vrijednosti kuteva zglobova pomoću motion generatora, postupak je opisan u 5.2. poglavlju. Gumb *Get current joints* upisuje trenutne vrijednosti kuteva zglobova u *TextBox-ove*. Stanje robota se čita na isti način kao i kod gumba *Add current pose*, međutim koristi se sljedeći kod za dobivanje vrijednosti kuteva:

```
1   joint_state = state.q;
```

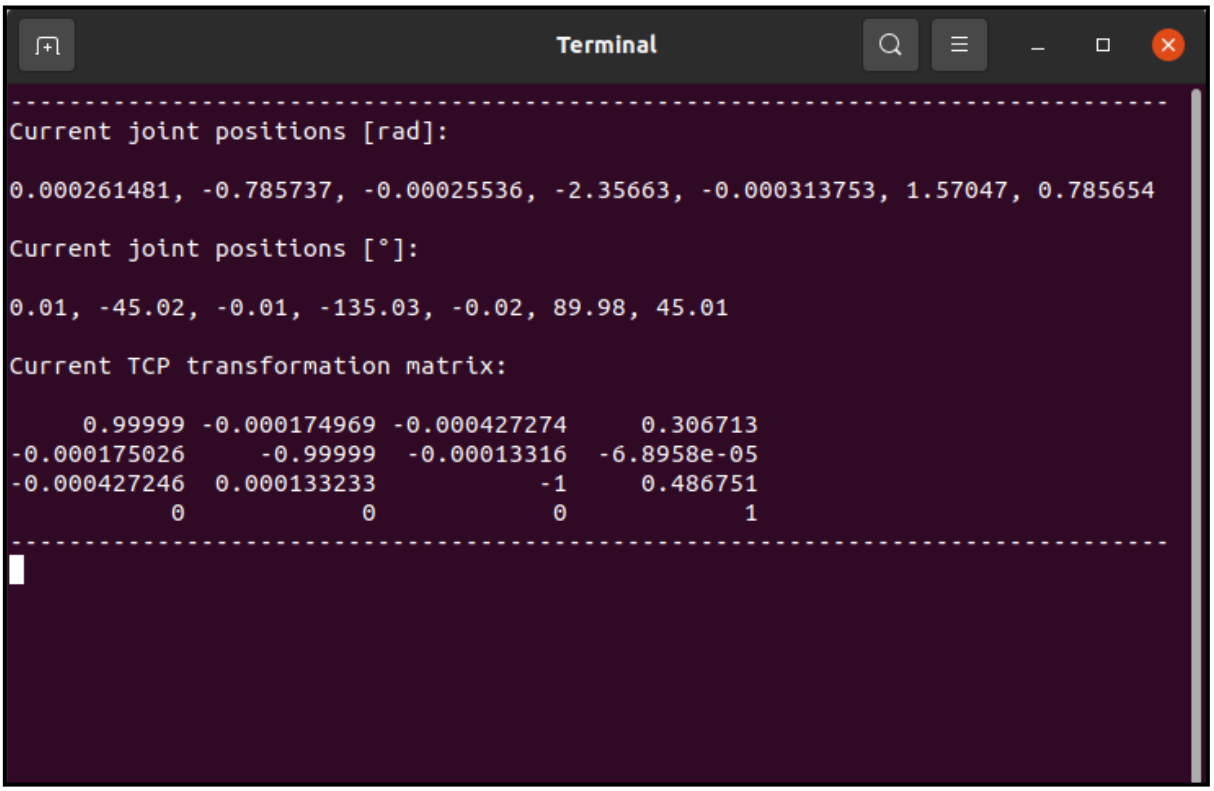
Kutevi se spremaju u niz od 7 vrijednosti nazvan *joint_state*. Kutevi i gibanje se također mogu dodati u glavni program (Start funkcija) sa *Add Joint Motion*.

4.1.4. Impedance meni

Impedance meni dopušta pokretanje kontrole impedancije. Moguće je zadati vrijednosti translacijske i rotacijske krutosti. Kontrolirajući impedanciju, kontroliramo kako se robot ponaša tijekom interakcije s okolinom definirajući njegovu krutost i prigušenje. Moguće je konfigurirati robota tako da se ponaša kao vrlo labava opruga ili nasuprot tome, ako je krutost visoka, robot bi se gibao samo pod velikim silama iz okoline. Kontrolu impedancije je potrebno zaustaviti stop gumbom kako bi se moglo nastaviti sa korištenjem sučelja.



Slika 18. Korisničko sučelje za upravljanje Franka Panda robotom: a) Points meni, b) Gripper meni, c) Joint meni, d) Impedance meni, e) Glavni meni



Slika 19. Ispisavanje pozicija zglobova i transformacijske matrice u terminalu

4.2. Help prozor

Klikom na upitnik u gornjem desnom uglu sučelja otvara se Help prozor sa bitnim informacijama vezanim za sučelje i Pandu.

4.2.1. General

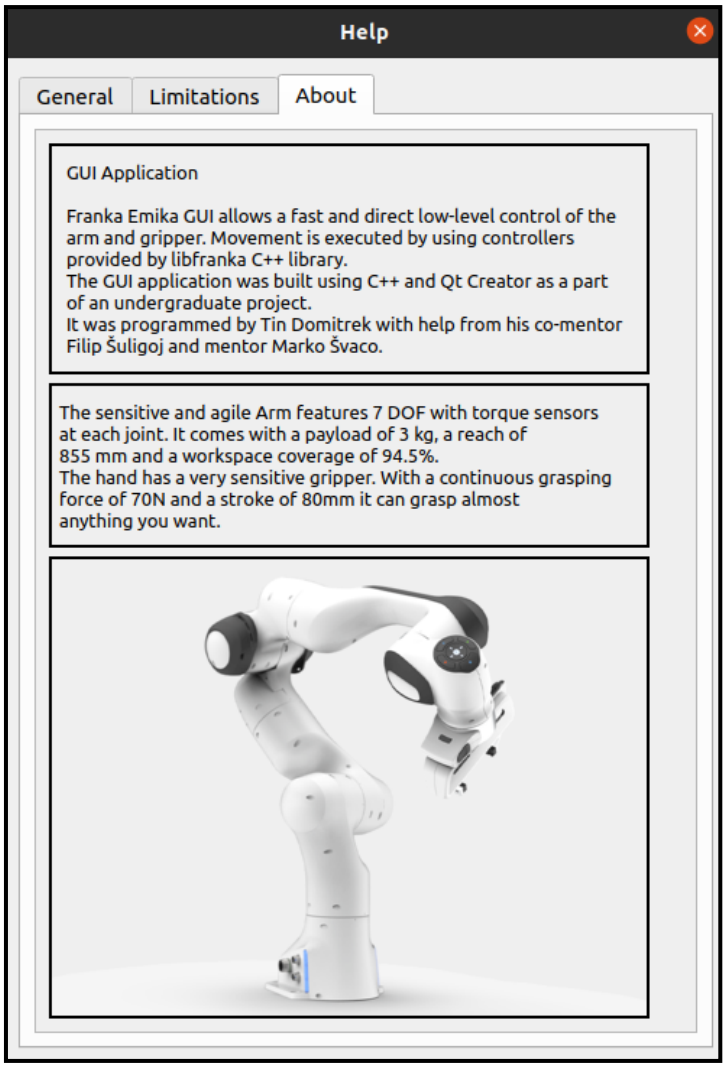
Sadrži opće informacije o sučelju i funkcijama koje sadrži.

4.2.2. Limitations

Sadrži limitacije u koordinatnom sustavu, limitacije kretnji zglobova u stupnjevima i limitacije kontrole impedancije.

4.2.3. About

Sadrži informacije o robotu i sučelju.



Slika 20. Help prozor

5. GIBANJE ROBOTA I ALGORITMI ZA UPRAVLJANJE

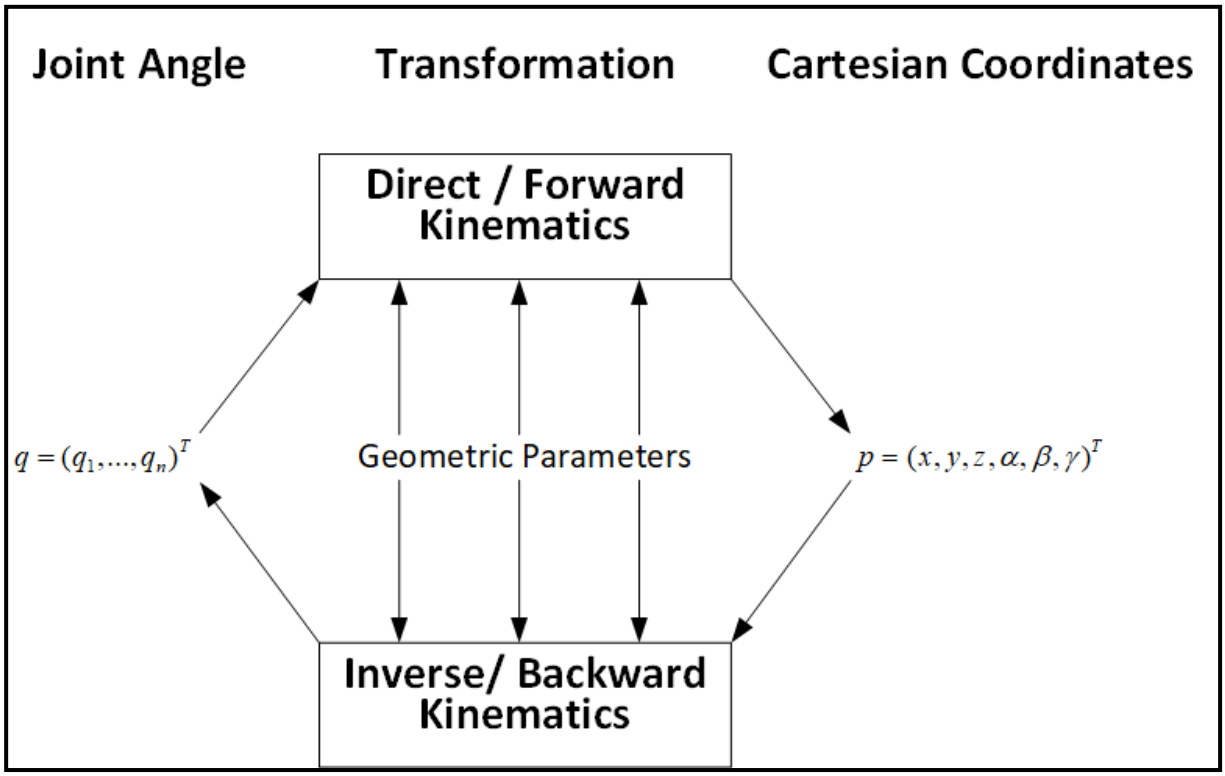
Cjelokupni proces gibanja robota obuhvaća mnoge aspekte od početka do kraja gibanja.

To su:

1. Kinematika robota koja se odnosi na geometrijsko proučavanje kretanja kinematičkih lanaca s više stupnjeva slobode koji tvore kinematičku strukturu. Opisuje pozu, brzinu, ubrzanje i sve izvedenice višeg reda pozicije tijela koja tvori mehanizam. [1] Za opis tijela u prostoru potrebno je 6 koordinata, tri translacije (pomaka) duž koordinatnih osi x, y,z, čime se postiže pozicioniranje točke nekog tijela u prostoru i tri rotacije (zakreta) oko koordinatnih osi, čime se omogućuje orijentacija tijela prema toj točki, tj. pozicioniranje druge točke tijela koja je čvrsto povezana s prvom. [2] Obuhvaća *forward* i *inverse* kinematiku s kojima se dobivaju pozicije koordinata i rotacije ako znamo pozicije zglobova, i obrnuto. U ovom radu se koristio algoritam za izračun inverzne kinematike panda robota u kojem se zglob 7 uzeo kao redundantan zglob (zglob ne mijenja poziciju) kako bi dobili jedinstveno rješenje inverzne kinematike.
2. Dinamika robota koja se odnosi na jednadžbe gibanja koje daju odnose između pokretačkih i kontaktnih sila koje djeluju na mehanizam robota, te ubrzanje i trajektorije gibanja koje rezultiraju iz tih vrijednosti. [1] Važan je niz metoda i algoritama u dinamici koji omogućuju računanje sljedećeg: inverzne dinamike (iz poznatih sila odrediti gibanja), direktne (forward) dinamike (iz poznatih gibanja odrediti sile), matrica inercije zglobnog prostora i matrica inercije operativnog prostora. [1] Panda robot koristi dinamiku kako bi radio precizne kretnje s greškama u redu 10^{-2} mm brzinama do 2.35 rad/s.
3. Planiranje gibanja koje obuhvaća planiranje kretnji bez sudara za robota od početka do kraja gibanja u zadanom statičkom okolišu. U sučelju razvijenom u sklopu završnog rada imamo praktično planiranje gibanja tako da uzmemo u obzir okoliš u kojem se nalazi Panda, te izvođenje testiranja kako bi se izbjegla šteta. Zadane limitacije opisane su u poglavlju *2.4.2.*
4. Kontrola gibanja koja obuhvaća stvaranje upravljačkih programa i petlji za gibanje robota. Razvijeni algoritmi dopuštat će stvaranje point-to-point i linearnih gibanje, te slijedno vođenje u kojem sve osi započinju i završavaju gibanje u isto vrijeme.

5.1. Inverzna kinematika

S obzirom na željeni krajnji položaj end effectora robota, inverzna kinematika (IK) može odrediti odgovarajuću konfiguraciju položaja zglobova za koju se end effector pomiče u ciljani krajnji položaj. Za razliku od forward kinematike (FK), roboti s više stupnjeva slobode općenito imaju više rješenja inverzne kinematike, što nam kroz testiranje daje više različitih rješenja i profila gibanja.



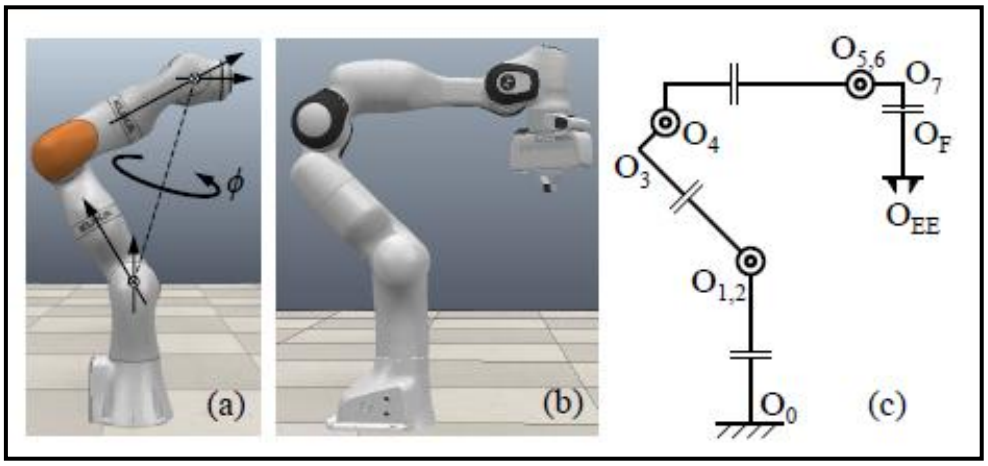
Slika 21. Inverse nasuprot forward kinematike [5]

5.1.1. Općenito o IK

Inverzna kinematika je težak problem, jer rješenje može, a i ne mora postojati. U slučaju kada postoji tada obično postoji više rješenja. Robot sa 6 stupnjeva slobode može imati 16 različitih konfiguracija kuteva koje dovode *end effector* u istu poziciju, za robota sa 7 stupnjeva slobode (Franka Emika Panda) to brojka je često beskonačna. [3]

Većina robota sa 7 stupnjeva slobode ima takozvano Eulerovo rame i Eulerov zglob. Međutim kod Pande to nije slučaj jer ona ima pomak na tim mjestima (Slika 21). Taj pomak čini konvencionalnu parametrizaciju neizvedivom pa se stoga koristi redundantan zglob kao što je i prije navedeno. Najprirodnije rješenje za to je zglob 7 koji algoritam tretira kao poznat i ne mijenja mu poziciju. [3]

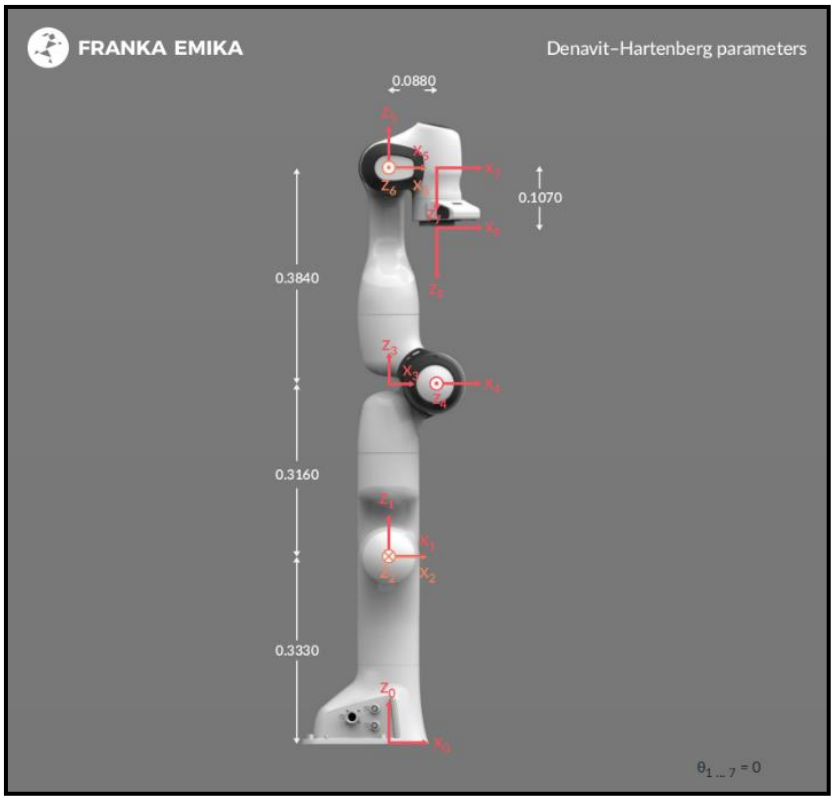
Algoritam koji se koristit u sučelju za računanje inverzne kinematike preuzet je sa sljedećeg linka na GitHubu: https://github.com/ffall007/franka_analytical_ik, gdje se nalazi i licenca. Algoritam se nalazi u prilogu.



Slika 22. a) Uobičajeni robot sa 7 SS, b) Panda, c) Offset Pande na zglobu 4 i 6 [3]

5.1.2. Rješavanje inverzne kinematike

Algoritam je izveden u potpunosti na temelju geometrije robota, tj. sve jednadžbe se daju protumačiti pomoću Denavit-Hartenberg parametara pande.



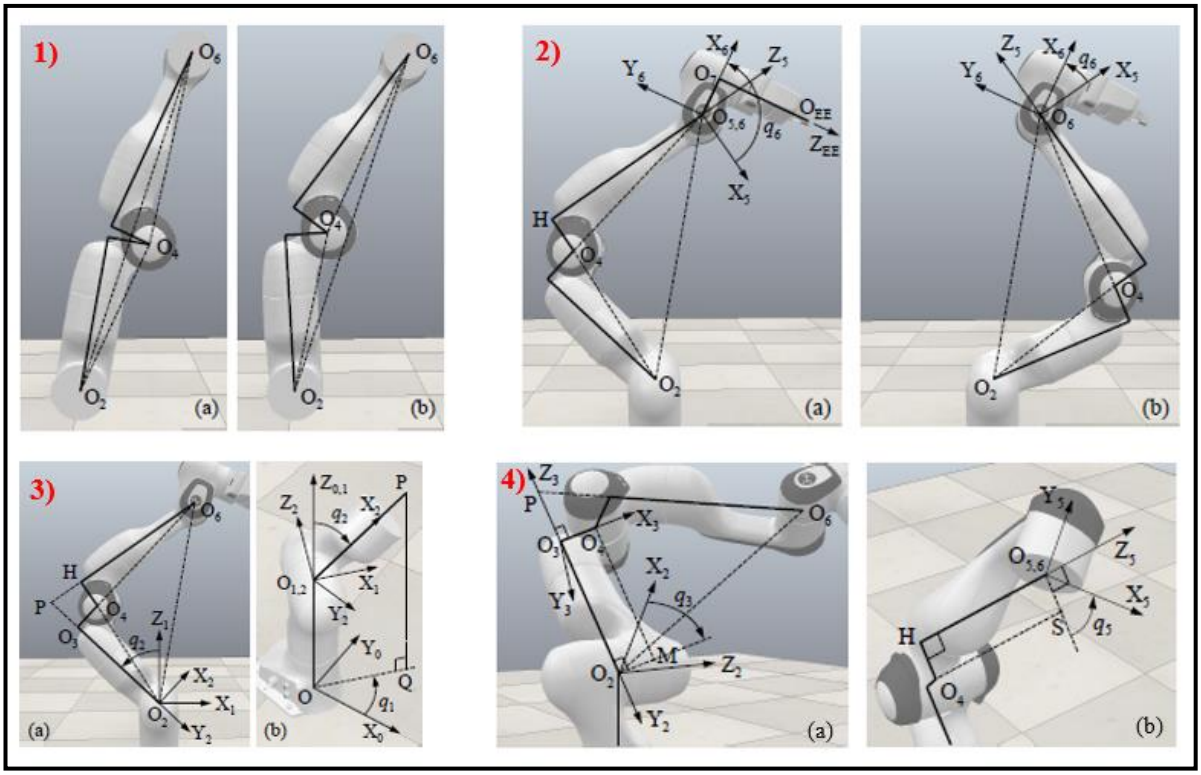
Slika 23. Denavit-Hartenberg parametri Pande [4]

Postupak rješavanja se provodi na sljedeći način:

1. Uzeti q_7 (zglob 7) kao redundantni zglob.
2. Istražiti trokut $\triangle O_2O_4O_6$ za riješiti dva moguća rješenja q_4.
3. Za svako rješenje q_4 riješiti dva moguća rješenja q_6.
4. Za svaku kombinaciju q_4-q_6 riješiti dvije mogućnosti (q_1, q_2).
5. Za svaku kombinaciju q_4-q_6-(q_1, q_2) riješiti jedinstveni q_3 i q_5.

Iz postupka rješavanja vidi se da je moguće dobiti 2^3=8 različitih rješenja za svaku jedinstvenu poziciju *end effectora*. [3] Više o postupku rješavanja u znanstvenom radu [3].

Unutar sučelja vrijednosti kartezijskih koordinata upisane od strane korisnika koje su spremljene u globalnom *pointsArray-u* se uzimaju u funkciju, pa se zatim unutar funkcije pomoću algoritma koji rješava inverznu kinematiku računaju odgovarajuće zglobne pozicije. Zglobne pozicije se zatim uzimaju u kontrolnu petlju uzetu iz libfranka knjižnice koja generira gibanje.



Slika 24. 1) Dva ekvivalentna rješenja q4, 2) Dva slučaja za računanje q6, 3) Računanje kuteva: a) q2, b) q1, 4) Računanje kuteva q3 i q5 [3]

5.2. Algoritam za gibanje u zglobne pozicije

Gibanje u zglobne pozicije je metoda interpolacije putanje koja određuje kretanje robota pomicanjem svakog zgloba izravno u zadani položaj tako da sve osi započnu i završe gibanje u isto vrijeme.

Zglobno gibanje je napravljeno na način da se u Joint meniju pomoću slidera, gumba lijevo/desno ili upisivanjem vrijednosti zadaju kutevi gibanja za svaki zglob. Nakon pritiska gumba *Move Joints* pokreće se kod. *Get current joints* dohvaća trenutne pozicije zglobova, a sa *Add current joints* se zglobno gibanje dodaje u *Points* prozor, tj. glavni program.

Prvo se deklarira robot i IP adresa pomoću sljedećeg koda (ovaj kod se uvijek koristi prije svih gibanja za deklaraciju robota):

```
1    franka::Robot robot(„192.168.40.45“);
```

Zatim se postavlja kontrola ponašanja pri sudaru sljedećim kodom (postavlja dopuštene vrijednosti vanjskih sila na robota i ponašanje pri koliziji, uvijek se koristi prije svakog gibanja):

```
1    Robot.setCollisionBehavior(DOPUŠTENE VRIJEDNOSTI);
```

Vrijednosti koje se upisuju u zagradu su odabrane prema preporuci iz libfranka knjižnice.

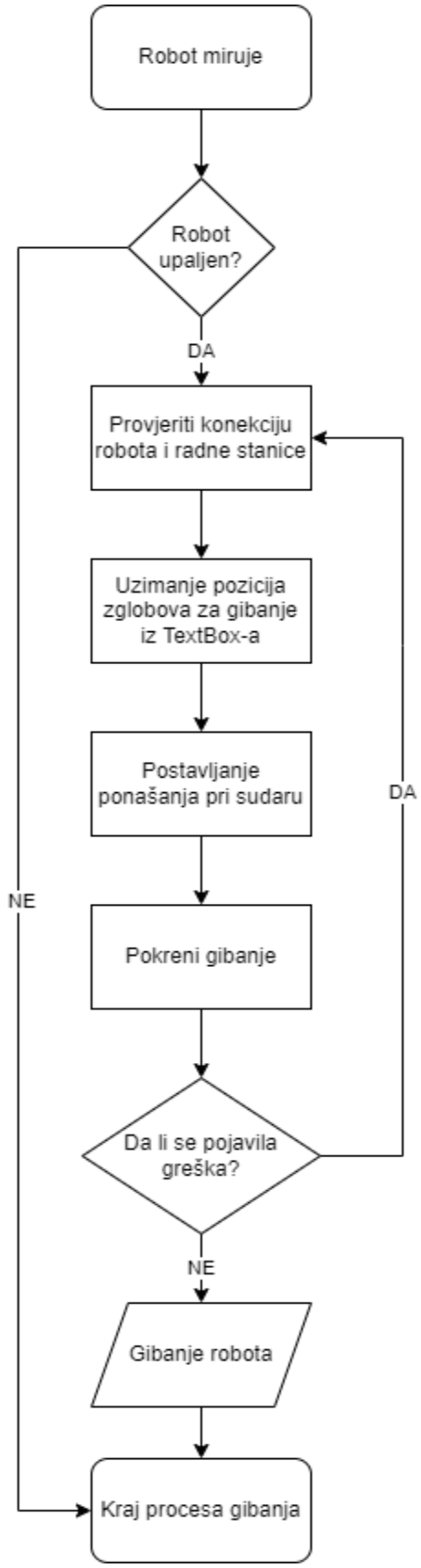
Sljedeći kod uzima vrijednosti u stupnjevima iz *TextBox-a* za sve zglobove koji se zatim pretvaraju u radijane, te ih sprema u niz *q* od 7 vrijednosti:

```
1    for(int i = 0; i < 7; i++){
2            q[i] = {(ui->spinBox_i->value())*M_PI/180};
3    }
```

Sljedeći kod za zglobno gibanje iz libfranka knjižnice pokreće robota:

```
1    std::array<double, 7> q_goal = {{q[0], q[1], q[2], q[3], q[4], q[5], q[6]}};
2    MotionGenerator motion_generator(0.2, q_goal);
3    robot.control(motion_generator);
```

Prva linija koda sprema vrijednosti kuteva iz niza *q* u niz *q_goal* koji se koristi u kontrolnoj funkciji. Druga linija koda deklarira kontrolnu funkciju i varijable (0.2 je brzina, *q_goal* je niz od 7 vrijednosti kuteva zglobova), a treća pokreće kontrolnu funkciju (motion generator).



Slika 25. Dijagram toka zglobnog gibanja

5.3. Algoritam za point-to-point gibanje

U point-to-point gibanju krajnja pozicija je određena, ali putanja kojom se dolazi do krajnje pozicije nije bitna, tj. robot sam izabire i izračunava putanju. Brzina i vrijeme trajanja gibanja se mogu definirati za pomake, omogućujući upravljaču da u sučelju konstruira složena slijedna gibanja. Algoritam je napravljen kao kombinacija generatora zglobnog gibanja i algoritma za računanje inverzne kinematike. Problem pozicije se rješava na način da se vrijednosti pomaka kartezijskih koordinata upisane od strane korisnika ubacuju u algoritam za inverznu kinematiku, koji ih pretvara u odgovarajuće krajnje zglobne pozicije. Zglobne pozicije dobivene iz algoritma se umjesto u *pointsArray* spremaju u novi niz nazvan *jointArray*. Niz *jointArray* se uzima u generator zglobnog gibanja koji generira point-to-point gibanje kroz glavni program (Start funkcija).

Prvo se deklarira robot i IP adresa, zatim se čita trenutna pozicija i orijentacija robota i dodaju se upisane vrijednosti kartezijskih koordinata:

```
1  franka::Robot robot("192.168.40.45");
2  setDefaultBehavior(robot);
3  franka::RobotState state = robot.readOnce();
4  std::array<double, 16> initial_pose = state.O_T_EE;
5  initial_pose[12] = pointArray[i][12];
6  initial_pose[13] = pointArray[i][13];
7  initial_pose[14] = pointArray[i][14];
```

Prva, druga i treća linija koda deklariraju i čitaju poziciju robota. Četvrta linija deklarira novi niz nazvan *initial_pose* koji sprema trenutnu poziciju i orijentaciju end effectora (O_T_EE). Zadnje tri linije koda mijenjaju kartezijske koordinate x, y, z (spremljene na mjesta 12, 13 i 14 u transformacijskoj matrici/nizu) iz trenutne pozicije u vrijednosti željenih koordinata upisanih od strane korisnika.

Sljedeći kod je vezan za inverznu kinematiku:

```
1  double q7 = joint_state[6];
2  for(int i = 0; i < 7; i++){
3          q[i] = franka_IK_EE_CC(initial_pose, q7, joint_state)[i];
4  }
```

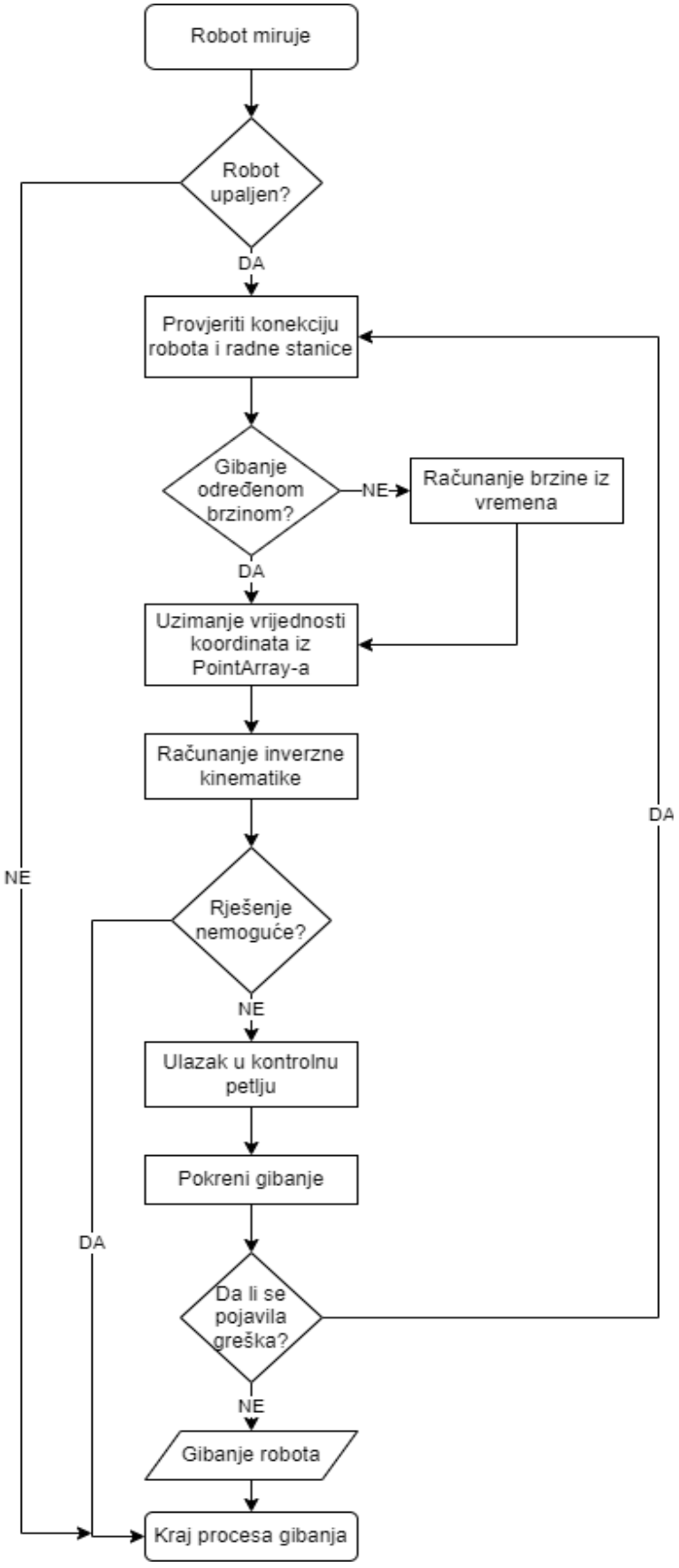
Prva linija deklarira zglob 7 kao redundantni. *For()* petlja koristi algoritam za inverznu kinematiku (*franka_IK_EE_CC()*) kako bi iz zadanih vrijednosti kartezijskih koordinata dobili vrijednosti zglobova u koje će se robot gibati. Kao ulazne parametre uzima početnu poziciju prije gibanja u kojoj smo kartezijske koordinate zamijenili sa onima upisanim od strane korisnika (*initial_pose*), vrijednost u radijanima redundantnog zgloba q_7 i *joint_state[i]* tj. trenutnu poziciju zgloba za kojeg računamo inverznu kinematiku.

Funkcija za point-to-point gibanje uzima vrijednost brzine i niz *q* od 7 vrijednosti izračunatih konačnih pozicija zglobova. U slučaju da je zadano gibanje u određenom vremenu koristimo sljedeći kod za dobivanje brzine iz zadanog vremena:

```
1   std::array<double, 7> diff;
2   for(int i = 0; i < 7; i++){
3           diff[i] = q[i] - joint_state[i];
4   }
5   double max = 0; double v;
6   for (int g = 0; g < 7; g++){
7           if (abs(diff[g]) > max){
8                   max = abs(diff[g]);
9           }
10  }
11  v = (max/timeSpeedValueArray[i][0])*1.25;
```

Prva linija deklarira novi niz za spremanje razlike između početne i krajnje pozicije kuteva nazvan *diff*. Slijedi *for()* petlja koja računa razliku pojedinog zgloba oduzimanjem *q[i]* vrijednosti od *joint_state[i]* vrijednosti. Sljedeća linija deklarira varijable za spremanje maksimalne vrijednosti razlike i brzine (*max* i *v*), a zatim slijede *for()* i *if()* petlje koje dolaze do maksimalne vrijednsoti razlike u radijanima koja se sprema u varijablu *max*. Zadnjom linijom se maksimalna vrijednost razlike dijeli sa spremljenim vremenom iz *timeSpeedValueArray-a* kako bi se dobila vrijednost brzine u rad/s. Vrijednost se množi sa 1.25 radi kalibracije točnosti.

Na kraju ulazimo u generator zglobnog gibanja sa izračunatim zglobovima pomoću inverzne kinematike i izračunate brzine, te se gibanje izvršava.



Slika 26. Dijagram toka point-to-point gibanja

5.4. Algoritam za linearno gibanje

Linearno gibanje je vrsta gibanja u kojoj se end effector robota od početne do krajnje točke giba po ravnoj liniji i po potrebi mijenja svoju orijentaciju. Brzina i vrijeme trajanja gibanja se mogu definirati za pomake kao i kod point-to-point gibanja.

Algoritam je napravljen pomoću lambda izraza koji omogućuje pisanje ugrađene funkcije unutar glavne funkcije koja se može koristiti za kratke isječke koda koji se neće ponovno koristiti i koje se ne isplati imenovati. [6] U kodu se također koristi Eigen C++ knjižnica za linearnu algebru: vektori, matrice, kvaternioni.

Prvo se deklarira robot i IP adresa, čita se stanje robota i deklariraju se dva kvaterniona *v* i *vv* uz pomoć Eigena. Vrijeme se uzima kao ulazna varijabla *time* iz *timeSpeedValueArray-a* i spremamo ga u *duration* varijablu.

```
1  franka::Robot robot("192.168.40.45");
2  setDefaultBehavior(robot);
3  franka::RobotState robot_state = robot.readOnce();
4  double duration = time;
5  Eigen::Quaterniond v; Eigen::Quaterniond vv;
```

Kvaternioni se uglavnom definiraju kao kvocijent dvaju vektora u trodimenzionalnom prostoru. Imaju primjenu u primijenjenoj matematici, posebice za izračune koji uključuju trodimenzionalne rotacije. [5] U algoritmu se koriste upravo za to, prikazivanje 3D orijentacija i rotacija end effectora.

Sljedeće se trenutna pozicija end effectora koju smo dobili u gornjem kodu sprema u 4x4 matricu pomoću Eigena, i dodaju se koordinate unesene od strane korisnika.

```
1  Eigen::Affine3d current_pose;
2  current_pose=(Eigen::Matrix4d::Map(robot_state.O_T_EE.data()));
3  Eigen::Affine3d commanded_pose = current_pose;
4  commanded_pose(0,3)=pointArray[i][12];
5  commanded_pose(1,3)=pointArray[i][13];
6  commanded_pose(2,3)=pointArray[i][14];
```

Prva linija koda deklarira *current_pose* kao transformacijsku matricu pomoću Eigena. U drugoj liniji se mapira *robot_state*, dakle pozicija i rotacija end effectora, u prije deklariran *current_pose*. Zatim deklariramo *commanded_pose* isto kao i *current_pose*, prepisujemo mu

vrijednosti iz *current_pose* i nadodajemo vrijednosti x, y i z koordinata iz *pointArray-a* upisane od strane korisnika jer je to pozicija u koju se želimo gibati.

Zatim se ulazi u kontrolnu petlju u kojoj se nalazi ugrađena lambda funkcija. Prvo se deklariraju potrebne varijable i matrice.

```
1  double time = 0;
2  Eigen::Affine3d initial_transform;
3  Eigen::Affine3d commanded_transform;
```

Time je varijabla u lambda funkciji koja služi kao uvjet jer se lambda funkcija odvija kao petlja, dakle funkcija će se izvršavati sve dok *time* ne dosegne odabrano ukupno vrijeme gibanja koje odabire korisnik. *Time* se mijenja za jednu milisekundu svakim prolaskom kroz funkciju. U *initial_transform* i *commanded_transform* ćemo spremiti trenutnu i zadanu transformacijsku matricu gibanja.

U kontrolnoj petlji *robot.control* se deklariraju sve varijbale koje se koriste i zatim se ulazi u lambda funkciju *CartesianPose*. U kontrolnu petlju, a zatim i u lambda funkciju ulazi se pomoću sljedećeg koda:

```
1  robot.control(
2  [&time,&initial_pose,&initial_transform,&commanded_transform,&commanded_pose,
   &duration, &v, &vv]
3  (const franka::RobotState& robot_state,
4  franka::Duration period) -> franka::CartesianPose {KOD LAMBDA FUNKCIJE}
5  )
```

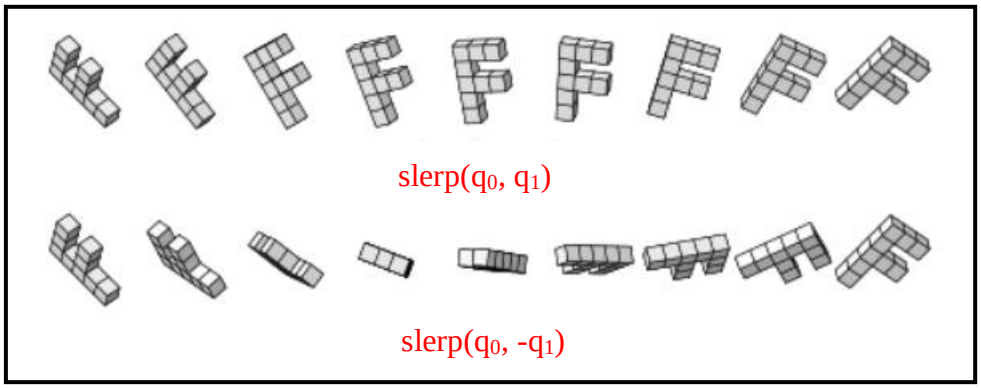
Sljedeći kod je vezan za lambda funkciju. Prvo se za vrijeme time = 0 izvršava sljedeća *if()* petlja. Ova petlja se izvršava samo na početku lambda funkcije:

```
1   if (time == 0.0) {
2           initial_pose = robot_state.O_T_EE_c;
3           initial_transform=(Eigen::Matrix4d::Map(robot_state.O_T_EE_c.data()));
4           commanded_transform.matrix()=commanded_pose.matrix();
5           v = (initial_transform.linear());
6           vv = (commanded_transform.linear());
7           return franka::CartesianPose(robot_state.O_T_EE_c);
8   }
```

Kao što je i prije navedeno, *initial_transform* i *commanded_transform* spremaju transformacijske matrice početne i krajnje pozicije gibanja. U kveternione se spremaju te iste vrijednosti, ali kao vektori, kako bi se kasnije mogla izračunati sferna linearna interpolacija (*slerp*) između ta dva kvaterniona. *Slerp* opisuje interpolaciju (s konstantnom kutnom brzinom) duž najkraćeg kružnog luka jediničnog radijusa na jediničnoj hipersferi između dva kvaterniona q_0 i q_1. [5] Definira se kao jedna od sljedećih četiri jednadžbe:

$$\begin{aligned} Slerp(q_0, q_1, t) &= q_0(q_0{}^{-1}q_1)^t \\ &= q_1(q_1{}^{-1}q_0)^{1-t} \\ &= (q_0q_1{}^{-1})^{1-t}q_1 \\ &= (q_1q_0{}^{-1})^t q_0 \end{aligned}$$

Parametar *t* (vrijeme) ide od 0 (gdje se izraz pojednostavljuje u q_0) do 1(gdje se pojednostavljuje u q_1).



Slika 27. Dva načina djelovanja slerpa s obzirom na orijentaciju kvaterniona [7]

Izlaskom iz *if()* dolazi se do dijela koda koji se izvodi svaku milisekundu do konačnog vremena. Najprije se definira nagib za linearnu interpolaciju pomoću sljedećeg koda:

```
1  double slope = (1 - std::cos(M_PI/ duration * time));
2  slope=slope*0.5;
```

Vrijednost se sprema u varijablu *slope*. Radi se o kosinusoidnoj funkciji oblika:

$$slope = \left(1 - \cos\frac{\pi}{duration * time}\right)$$

Slope se zatim množi sa 0.5 jer se želi doći samo do maksimalne vrijednosti kosinusoidne funkcije, a ne se vraćati u nulu.

Slijedi dio koda u kojem se deklarira nova pozicija u koju se robot giba kao *new_pose* i mijenjaju joj se vrijednosti u ovisnosti o *slope*, a samim time ovisi i o vremenu.

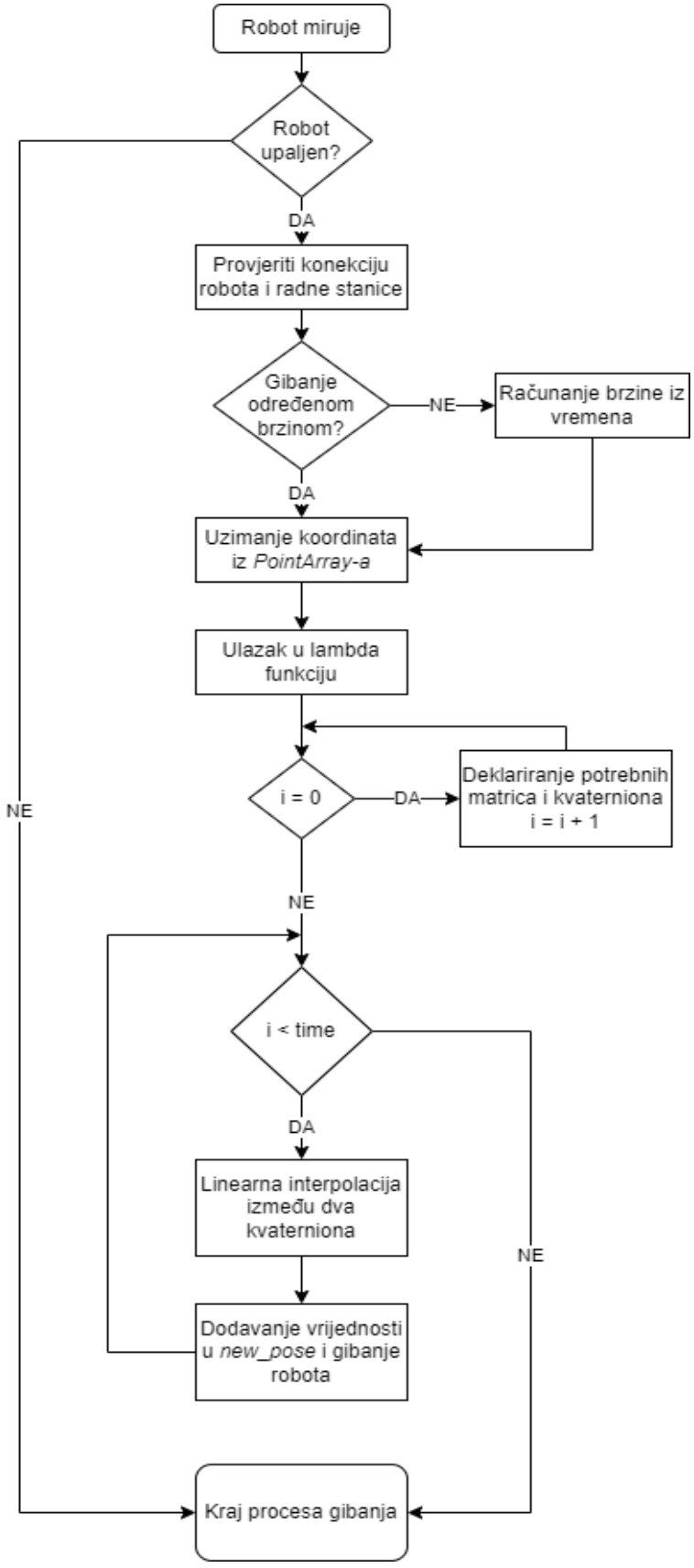
```
1  std::array<double, 16>  new_pose = initial_pose;
2  new_pose[12] = initial_pose[12]+slope*(commanded_transform.translation().x()-
   initial_pose[12]);
3  new_pose[13] = initial_pose[13]+slope*(commanded_transform.translation().y()-
   initial_pose[13]);
4  new_pose[14] = initial_pose[14]+slope*(commanded_transform.translation().z()-
   initial_pose[14]);
```

Uz svaki prolazak kroz funkciju vrijednosti se ponovno dodaju na *new_pose* za onoliko koliko se robot pomaknuo. Ta se vrijednost nadodaje svaku milisekundu vremena. Zatim se radi interpolacija između dva kvaterniona (*slerp*) koja daje vrijednosti rotacija koje se također spremaju u *new_pose*.

```
1  Eigen::Quaterniond result=v.slerp(slope,vv);
2  Eigen::Matrix3d R = result.normalized().toRotationMatrix();
3  new_pose[0]=R(0,0); new_pose[1]=R(1,0); new_pose[2]=R(2,0);
4  new_pose[4]=R(0,1); new_pose[5]=R(1,1); new_pose[6]=R(2,1);
5  new_pose[8]=R(0,2); new_pose[9]=R(1,2); new_pose[10]=R(2,2);
```

Prva linija deklarira *result* kao kvaternion rezultata interpolacije. Kvaternion *result* se zatim pretvara u transformacijsku matricu i njegove vrijednosti se dodaju u *new_pose*.

S ovom linijom završava lambda funkcija. *New_pose* se vraća u lambda funkciju koja se ponovno izvršava sa novom vrijednosti *new_pose* i novim vremenom sve dok ne dođemo do zadane vrijednosti iz *timeSpeedValuArray-a*.



Slika 28. Dijagram toka linearnog gibanja

5.5. Start funkcija

Glavni zadatak razvijenog korisničkog sučelja je pokretanje sekvencijalnih gibanja/zadataka. *Points* prozor prikazuje sve točke koje su zadane od strane korisnika i u koje će se robot gibati. Gibanja su dostupna u sljedeća četiri oblika: otvaranje/zatvaranje hvataljke, zglobno gibanje, point-to-point gibanje i linearno gibanje. Gibanja se u *Points* prozor dodaju pomoću četiri različita Add gumba koji su prije opisani. Uz sve ovo razrađena je glavna funkcija gibanja (Start funkcija) koja pokreće svako gibanje po redu onako kako to stoji u *Points* prozoru.

Na početku slijedi *for()* petlja koja se izvršava za i = 0 do 15 jer je određen maksimalan broj zadanih točci jednak 16.

```
1   for (int i = 0; i < 16; i++){
2   KOD
3   }
```

Unutar *for()* petlje nalazi se prva *if()* petlja kojom se određuje koja vrsta gibanja se izvršava pomoću *conditionArray-a* u koji se spremaju uvjeti (1 – otvaranje hvataljke, 2 – zatvaranje hvataljke, 3 – zglobno gibanje, 4 – point-to-point odnosno linearno gibanje). S obzirom na to koja vrijednost je spremljena u *conditionArray-u*, izvršava se odgovarajuće gibanje.

```
1   if (conditionArray[i][0] == 1){OTVARANJE HVATALJKE
2   } else if (conditionArray[i][0] == 2){ZATVARNAJE HVATALJKE
3   } else if (conditionArray[i][0] == 3){ZGLOBNO GIBANJE
4   } else {POINT-TO-POINT/LINEARNO GIBANJE
5   }
```

Zatim za point-to-point/linearno gibanje slijedi *if()* petlja koja određuje da li se izvršava point-to-point ili linearno gibanje i na kraju unutar te *if()* petlje je nova *if()* petlja koja određuje da li je zadano gibanje u određenom vremenu ili određenom brzinom. Koristi se novi niz *TORS_Array* koji sprema uvjete za vrijeme/brzinu (1 – vrijeme, 2 – brzina) i *motionArray* koji sprema uvjete za gibanje (1 – point-to-point, 2 – linearno). Te vrijednosti se zadaju i spremaju unutar Point menija. *Function_move_ptp* i *function_move_lin* su funkcije za point-to-point i linearno gibanje. Kod se nalazi na sljedećoj stranici.

Kod:

```
1   if (TORS_Array[i][0] == 1){
2           if (motionArray[i][0] == 1){
3                   function_move_ptp(time)
4           } else {
5                   function_move_linear(time)
6           }
7   } else {
8           if (motionArray[i][0] == 1){
9                   function_move_ptp(speed)
10          } else {
11                  function_move_linear(speed)
12          }
13  }
```

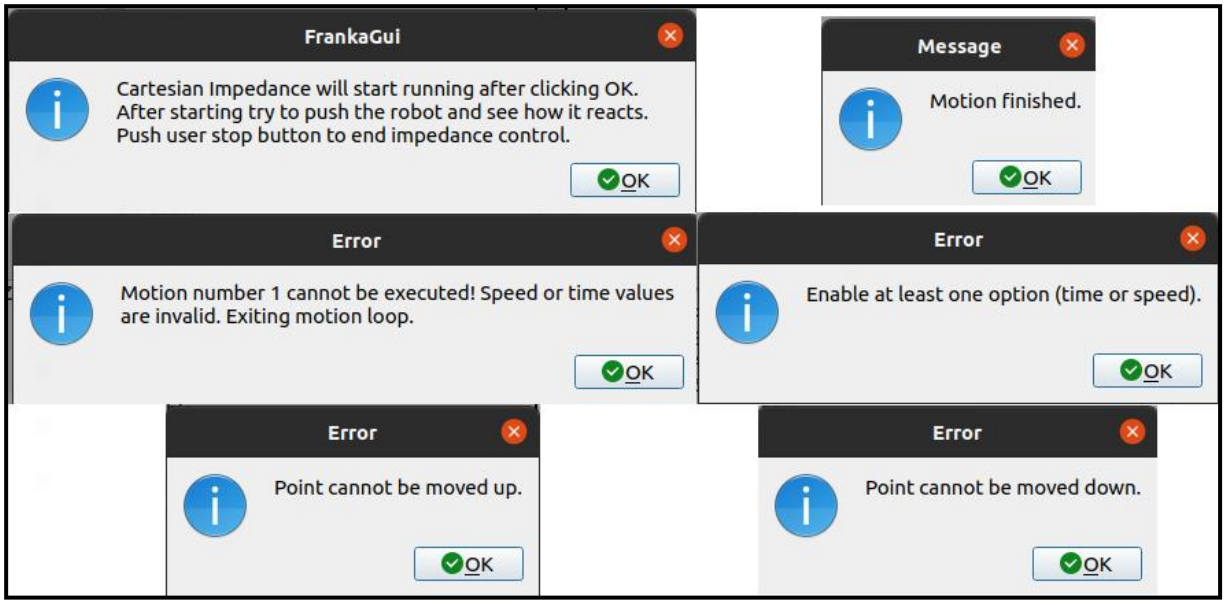
S ovom *if()* petljom završava Start funkcija. Pokreće se klikom na Start gumb, te kada su izvršena sva gibanja pojavljuje se statusna poruka *End of motion* i robot je opet u stanju mirovanja.

6. TESTIRANJE

Nakon izrade sučelja i algoritama za gibanje napravljeno je testiranje svih komponenti. Testiranje se provodi na gore opisanom Franka Emika Panda robotu koji se nalazi u laboratoriju CRTA-e.

6.1. Statusne poruke

Unutar sučelja se pojavljuju sljedeće statusne poruke vezane za niz upozorenja. Tiču se uglavnom sigurnosti robota (impedancija i sigurnosna funkcija koja ne dopušta izvršavanje nekih opasnih gibanja), funkcionalnosti unutar sučelja (micanje točaka gore/dolje i biranje vremena/brzine za gibanje), te statusna poruka za kraj gibanja.



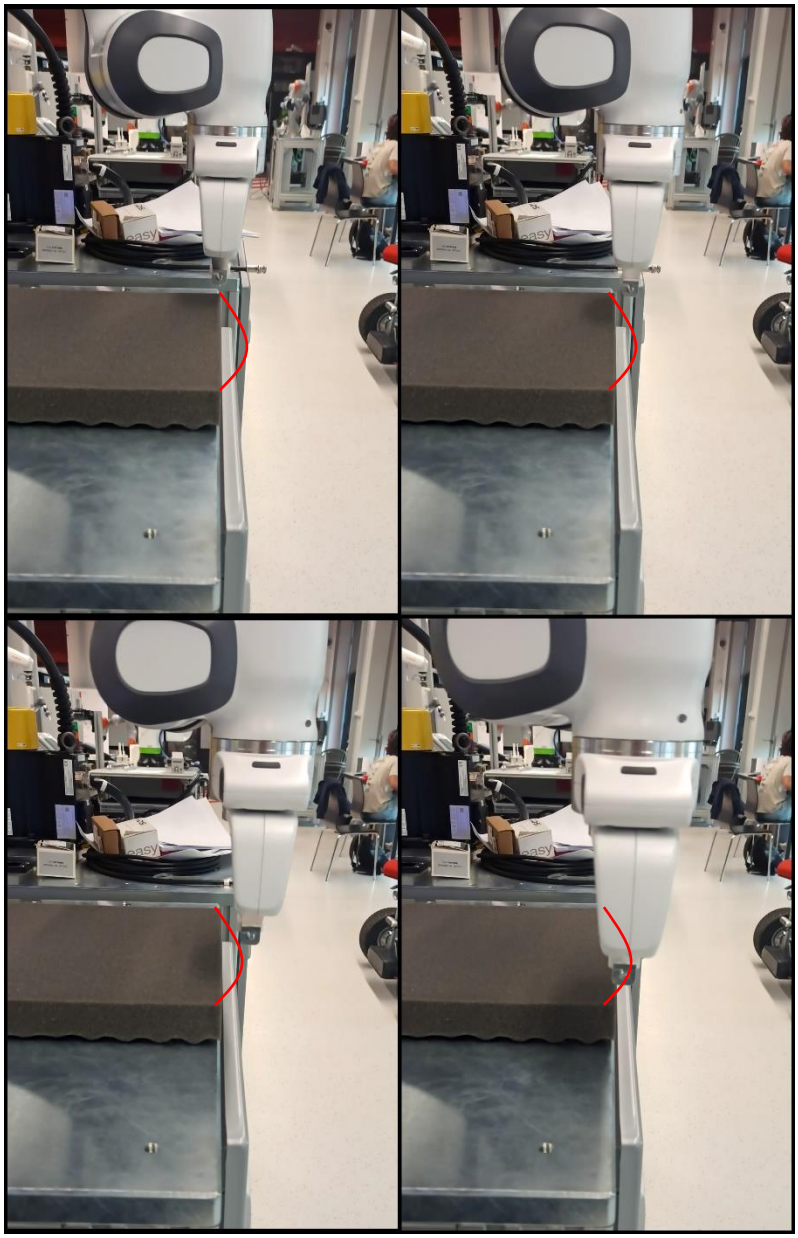
Slika 29. Statusna upozorenja unutar sučelja

6.2. Testiranje gibanja

Testiramo point-to-point i linearno gibanje u zadanom vremenu i zadanom putanjom po ravnom bridu.

6.2.1. Point-to-point gibanje

Point-to-point gibanje po ravnom rubu. Koordinate su zadane u sučelju, te se program pokreće. Kod gibanja se vidi malo odstupanje od ravnog brida.

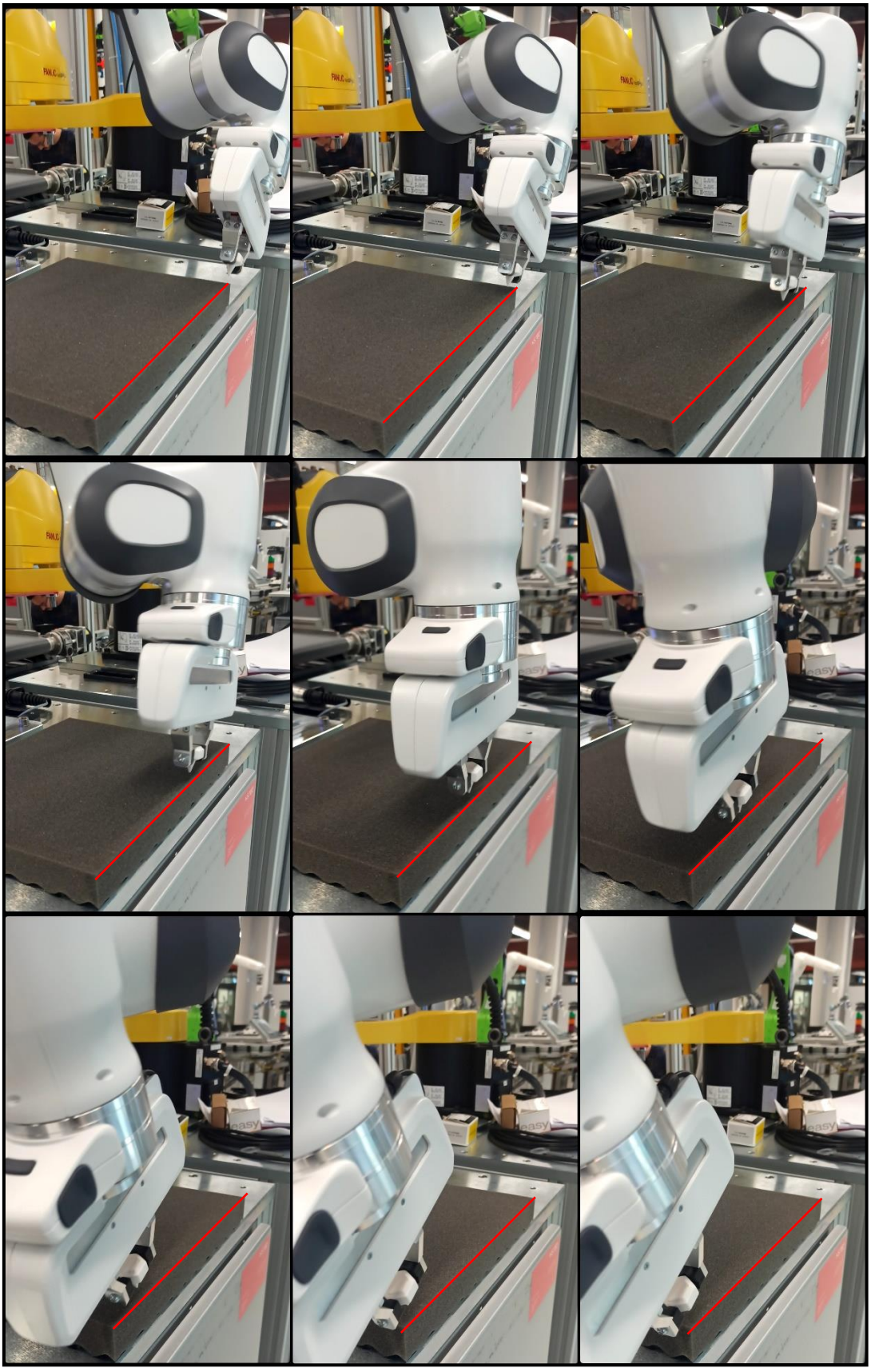


Slika 30. PTP gibanje po ravnom bridu

6.2.2. Linearno gibanje

Linearno gibanje po ravnom bridu na slici 31. Vidi se ravno kretanje robota po bridu. Na slici 32 dodatno linearno gibanje s promjenom orijentacije end-effectora.

Slika 31. Linearno gibanje po ravnom bridu



Slika 32. Linearno gibanje po ravnom bridu uz promjenu orijentacije

7. ZAKLJUČAK

Franka Panda je visokotehnološka robotska ruka koja uz FCI i libfranku postaje idealno rješenje za istraživanje upravljanja na niskoj razini. Uz pomoć vanjskog softvera poput Qt alata za izradu korisničkih sučelja koji je pisan u C++ programskom jeziku, razvijeno je sučelje za upravljanje na niskoj razini sa Franka Pandom. Franka Control Interface (FCI) i libfranka daju korisnicima na korištenje niz kompleksnih funkcija i primjera za gibanje iz svoje knjižnice, koji su već isprogramirani i spremni za pokretanje. Međutim smatram da je najveća čar libfranke, ali i strojarstva općenito pristupiti nekom problemu i suoćiti se s njim, te napraviti rješenje s kojim se možeš ponositi. Libfranka dopušta upravo to, stvaranjem vlastitih jedinstvenih kontrolnih petlji i kontrolera gibanja, tjerajući Pandu da bilo koju našu ideju pretvori u stvarnost, ili bilo koji problem u rješenje. Sučelje razvijeno u sklopu ovog završnog rada je upravo primjer toga, te mi je bila čast imati na raspolaganju takav stroj i upravljati njime.

LITERATURA

[1] Siciliano, B., Khatib O.: Springer Handbook of Robotics, 2016.

[2] Crneković, M.: "Industrijski i mobilni roboti, predavanja." *FSB Zagreb* (2016).

[3] He, Y., Liu, S.: Analytical Inverse Kinematics for Franka Emika Panda – a Geometrical Solver for 7-DOF Manipulators with Unconventional Design, 2014.

[4] Franka Emika GmbH: Franka Panda User Guide, 2018.

[5] https://en.wikipedia.org/wiki/

[6] https://www.geeksforgeeks.org/lambda-expression-in-c/

[7] https://splines.readthedocs.io/en/latest/rotation/slerp.html

PRILOZI

I. C++ kod