Primjena 3D vizijskog sustava za praćenje objekata robotom u realnom vremenu

Kirić, Nikola

Master's thesis / Diplomski rad

2015

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Mechanical Engineering and Naval Architecture / Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:797860

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-12-18**

Repository / Repozitorij:

Repository of Faculty of Mechanical Engineering and Naval Architecture University of Zagreb

Sveučilište u Zagrebu
Fakultet strojarstva i brodogradnje

DIPLOMSKI RAD

Nikola Kirić

Zagreb, 2015.

Sveučilište u Zagrebu
Fakultet strojarstva i brodogradnje

DIPLOMSKI RAD

Mentor:
Prof. dr. sc. Bojan Jerbić

Student:
Nikola Kirić

Zagreb, 2015.

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

Zahvaljujem se mentoru prof. dr. sc. Bojanu Jerbiću na odabiru teme. Također se zahvaljujem asistentu mag. ing. mech. Filipu Šuligoju na pruženoj stručnoj pomoći i korisnim savjetima tijekom izrade rada.

Posebno se zahvaljujem roditeljima koji su mi bili velika podrška tijekom studiranja i omogućili mi da ovaj studij uspješno privedem kraju.

_____

Nikola Kirić

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

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum	Prilog
Klasa:	
Ur.broj:	

DIPLOMSKI ZADATAK

Student: **NIKOLA KIRIĆ** Mat. br.: 0035181355

Naslov rada na hrvatskom jeziku: **PRIMJENA 3D VIZIJSKOG SUSTAVA ZA PRAĆENJE OBJEKATA ROBOTOM U REALNOM VREMENU**

Naslov rada na engleskom jeziku: **APPLICATION OF 3D VISION SYSTEM FOR OBJECT TRACKING BY ROBOT IN REAL TIME**

Opis zadatka:

U okviru diplomskog rada potrebno je izraditi upravljački program koji će omogućiti relativno vođenje alata robotske ruke u realnom vremenu, koristeći informacije 3D vizijskog sustava. Pri tome voditi računa o ostvarivanju što bržeg prijenosa i obradu podataka radi postizanja kontinuiranog kretanja robotskog alata u odnosu na objekt koji se prati. Također, potrebno je realizirati kontinuiranu komparativnu korekciju planirane putanje vrha alata robota uspoređivanjem robotskih koordinata s podacima vizijskog sustava.

Razvijeni upravljački program treba uključiti sljedeće funkcionalnosti:

- razmjenu podataka s robotom koristeći brzu komunikaciju na relaciji vizijski sustav-računalo-robot ("korisnik-poslužilac" programsku podrška)
- automatsku kalibraciju orijentacije alata (promjena rotacije koordinatnog sustava na vrh alata robota s obzirom na orijentaciju markera za praćenj)
- kontinuirano i brzo preuzimanje informacija položaja i orijentacije 2 višeplanarna markera iz vizijskog sustava,
- transformaciju podatka i izračun matrice transformacija koristeći C++ knjižnicu za linearnu algebru,
- upućivanje alata na kraju robotske ruke u određenu poziciju i orijentaciju unutar koordinatnog sustava jednog od markera uz mogućnost korekcije putanje za vrijeme gibanja robota.

Zadatak zadan:	Rok predaje rada:	Predviđeni datum obrane:
24. rujna 2015.	26. studenog 2015.	2., 3. i 4. prosinca 2015.

Zadatak zadao:

Prof. dr. sc. Bojan Jerbić

Predsjednik Povjerenstva:

Prof. dr. sc. Franjo Cajner

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS TEHNIČKE DOKUMENTACIJE

Broj crteža	**Naziv**	**Tip crteža**
M1-001-2015	Alat M1	Radionički

POPIS OZNAKA

Oznaka	Jedinica	Opis oznake
$\boldsymbol{\theta}$	-	Axis-angle zapis rotacije
θ	rad	Kut poniranja
ϕ	rad	Kut valjanja
ψ	rad	Kut skretanja
E	mm	Euklidska udaljenost
Q	-	Kvaternion
$\mathbf{R}$	-	Matrica rotacije (DCM matrica)
$\mathbf{R_{A-A}}$	-	Matrica transformacije iz Axis–angle u matricu rotacije
$\mathbf{R_E}$	-	Matrica transformacije iz Eulerovih kutova u matricu rotacije
$\mathbf{R_Q}$	-	Matrica transformacije iz kvaterniona u matricu rotacije
$\mathbf{T}$	-	Matrica homogene transformacije

SAŽETAK

Zadatak ovog diplomskog rada bio je primijeniti stereovizijski sustav za praćenje nekog objekta robotom u realnom vremenu. Pri tome je korišten robot UR5, stereovizijska kamera Polaris Vicra i računalo. U okviru ovog diplomskog rada izrađen je upravljački program koji kontinuirano prima podatke sa robota i stereovizijske kamere, izračunava poziciju i orijentaciju točke u koju se robot mora pomaknuti te je šalje robotu. Program je napisan u programskom jeziku C++ i detaljno je objašnjen u poglavlju „Upravljački program za praćenje", a kompletan programski kôd priložen je u prilogu A. U prethodnim poglavljima objašnjeni su svi pojmovi koji su korišteni za izradu upravljačkog programa, sustav Polaris i robot UR5. Upravljački program moguće je primijeniti na bilo kojem industrijskom robotu uz prilagodbu naredbi koje se šalju robotu. Jedno od mogućih područja primjene ovog sustava je transkranijalna magnetska stimulacija (TMS) gdje ovaj sustav bitno olakšava posao liječniku, a pritom povećava i udobnost pacijenta.

Ključne riječi: Robotika, Praćenje, TMS, Stereovizija, Polaris, UR5

1. UVOD

Riječ robot prvi put spominje Karel Čapek u svome dijelu R.U.R. (Rossumovi univerzalni roboti) 1920. godine, a potječe od češke riječi „*robota*" što u prijevodu znači težak rad. Robot je stroj koji bez radnika izvodi složene operacije na inteligentan način. Znanost koja se bavi robotima naziva se robotika. Taj termin izmislio je američki pisac Isaac Asimov u znanstveno-fantastičnoj priči „*Liar!*" 1941. godine. Godinu kasnije Isaac Asimov u kratkoj priči pod naslovom „*Runaround*" uvodi prva tri zakona robotike, a u kasnijim knjigama uveo je i nulti zakon robotike. Ti zakoni glase [1]:

0. Robot ne smije naštetiti čovječanstvu ili svojom pasivnošću dopustiti da se čovječanstvu našteti.
1. Robot ne smije naštetiti čovjeku ili svojom pasivnošću dopustiti da se čovjeku našteti, osim kad je to u suprotnosti s nultim zakonom.
2. Robot mora slušati ljudske naredbe, osim kad su one u suprotnosti s nultim ili prvim zakonom.
3. Robot treba štititi svoj integritet, osim kad je to u suprotnosti s nultim, prvim ili drugim zakonom.

Na kraju 2014. godine u svijetu je bilo oko 1,5 milijuna operativnih industrijskih robota, a prema trendu rasta na kraju 2018. godine trebalo bi ih biti oko 2,3 milijuna. [2]
Robotika danas zauzima značajno mjesto i u medicini. Tri su osnovna područja u kojima se primjenjuju robotski sustavi: kirurgija, dijagnostika i rehabilitacija. U kirurgiji se roboti primjenjuju u fazama obuke operatera, podrške operateru tijekom operacija, pa do zahvata na daljinu ili samostalnog rada. Roboti koji se primjenjuju u kirurgiji najčešće su industrijski roboti prilagođeni konkretnom zadatku. Prilagodbe se odnose na alate i prije svega na softversku podršku koja čini glavnu razliku između medicinskog i industrijskog robota.

1.1. Roboti u medicini

Napretku medicine kao znanstvene i stručne djelatnosti uvelike doprinosi razvoj tehnologije i njezina sve češća primjena u istoj. Primjena robota u području medicine započela je 1985. godine kada je robot Puma 560 korišten za postavljanje igle za biopsiju mozga koristeći vođenje kompjuterskom tomografijom. Međutim, iz sigurnosnih razloga njegova je primjena obustavljena. Nakon Pume predstavljen je robotski sustav PROBOT koji se koristio za transuretralnu resekciju prostate. Zatim su 1992. godine IBM i suradnici razvili robotski sustav ROBODOC koji se koristio u ortopedskoj kirurgiji. Danas je najpoznatiji i najkorišteniji robotski kirurški sustav da Vinci koji je opisan u sljedećem odjeljku.
Primjena robota u kirurgiji ima i značajne prednosti kako za pacijenta tako i za kirurga, a neke od tih prednosti su [3]:

- Prednosti za pacijenta:
  - manji bolovi
  - manji rezovi
  - manji ožiljci
  - skraćeno vrijeme oporavka
  - minimalna mogućnost infekcije i nastanka komplikacija
- Prednosti za kirurga:
  - veća preciznost
  - mirniji instrumenti
  - manje invazivne procedure

Uz navedene prednosti također može doći i do određenih financijskih ušteda. U tablici 1.1 navedeni su troškovi i uštede za dva robotska sustava: Da Vinci i Zeus. Iz tablice je vidljivo da se uštede postižu skraćenim boravkom pacijenta u bolnici. Značajni novčani iznos, čak četvrtina cijene robotskog sustava, izdvaja se za obuku osoblja bez koje rad sa takvim sustavom ne bi bio moguć.

Primjena robota u medicini sve više postaje alternativa klasičnom operativnom pristupu pacijentu. Navedene prednosti bitno olakšavaju rad medicinskom osoblju, pružaju veću sigurnost i povećavaju povjerenje pacijenta.

Tablica 1.1: Procjena početnih ulaganja i troškova štednje po zahvatu [4]

	Da Vinci	Zeus
Tržišna cijena	$1,000,000	$975,000
Održavanje/godišnje	$100,000	$100,000
Izobrazba osoblja	$250,000	$250,000
Ukupno	**$1,350,000**	**$1,325,000**
Smanjeni broj dana provedenih u bolnici	4.5 dana	3.5 dana
Uštede po postupku	**$9,000**	**$7,000**

1.1.1. Kirurški sustav da Vinci

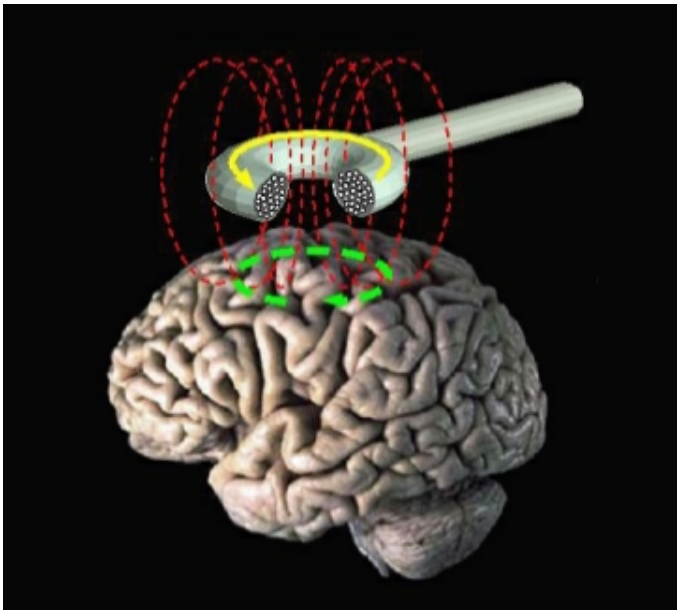
Kirurški sustav da Vinci [Slika 1.1] je robotski kirurški sustav koji je razvila tvrtka Intuitive Surgical s ciljem olakšanja složenih kirurških zahvata koristeći minimalno invazivni pristup. Sustavom upravlja kirurg putem kirurške konzole i na taj način se operativni

Slika 1.1: Kirurški sustav da Vinci

zahvat izvodi na daljinu. Najviše se koristi za prostatektomiju, a sve više i za popravak mitralnog zaliska i ginekološke kirurške zahvate. Da Vinci robotski sustav je u primjeni diljem svijeta, a procjenjuje se da je 2012. godine s njime provedeno preko 200 000 operacija. Trenutno je više od 3000 da Vinci sustava u uporabi od čega ih je oko 500 u Europi. Pacijenti imaju veliko povjerenje u taj sustav pa tako npr. u Njemačkoj sve više pacijenata ne želi ni pristupiti prostatektomiji ukoliko se zahvat ne obavlja navedenim robotskim sustavom.

1.2. Transkranijalna magnetska stimulacija

Transkranijalna magnetska stimulacija (TMS) (eng. *Transcranial magnetic stimulation*) je neinvazivna tehnika za direktnu stimulaciju neuralnog tkiva (cerebralni korteks, spinalni korijeni, kranijalni i periferni živci) i rutinska metoda u kliničkoj neurofiziologiji za ispitivanje funkcionalnog integriteta kortikospinalnog i kortikobulbarnog puta kod raznih neuroloških bolesti. [5] Tijekom TMS postupka generator magnetskog polja smješten je u neposrednoj blizini glave osobe koja prima tretman. Shematski dijagram TMS-a prikazan je na slici 1.2. Kako bi se dobili bolji rezultati najčešće se koristi navigacijska transkranijalna magnetska stimulacija (nTMS) (eng. *Navigated transcranial magnetic stimulation*). Precizna navigacija je moguća uz pomoć slika magnetske rezonancije i optičkog stereovizijskog sustava. Jedno od najboljih rješenja za nTMS nudi tvrtka Nexstim.



Slika 1.2: Transkranijalna magnetska stimulacija (shematski dijagram)

1.2.1. Nexstim

Nexstim je tvrtka iz Finske koja se bavi izradom sustava za TMS. Njihov glavni i najpoznatiji proizvod je sustav NBS (eng. *Navigated Brain Stimulation*), prikazan na slici 1.3. Taj sustav je spojen sa 3D navigacijskim sustavom i daje točnu i detaljnu mapu elokventnih funkcija kore mozga označavajući je na standardnom prikazu magnetske rezonancije (MRI). Uporaba navigacijskog sustava i stereotaktičkog MRI prikaza mozga kombinirana s TMS-om omogućava precizno neinvazivno pobuđivanje kore mozga. Sofisticirana obrada podataka u realnom vremenu omogućava točnost prikaza induciranog električnog polja unutar moždanog tkiva. U sustav je integriran i elektromiografski prikaz odgovora mišića.

Slika 1.3: Nexstim NBS

1.2.2. Axilum Robotics

Axilum Robotics je francuska tvrtka koja je razvila prvi robot posebno dizajniran za transkranijalnu magnetsku stimulaciju. Ideju za razvoj takvog robota dali su Pr Michel

de Mathelin i Dr Jack Foucher 2004. godine. Kod TMS je velik izazov osigurati preciznost stimulacije, a nekim pacijentima je potrebno i više od 30 terapija koje bi sve trebale biti identične. Iz toga je proizašla potreba za automatizacijom TMS sustava. Automatizirani postupak nudi mnoge prednosti: preciznost, ponovljivost, korekciju pomaka pacijenta, jednostavnost korištenja, sigurnost i udobnost operatera i pacijenta, pristup velikim područjima stimulacije i stalni kontakt između glave pacijenta i izvora magnetskog polja koji je osiguran senzorom sile.

Slika 1.4: TMS Robot

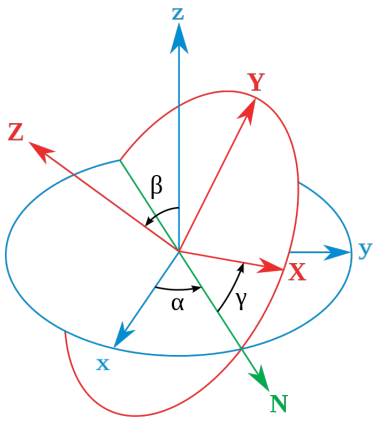
2. TEORIJSKA PODLOGA

U ovom poglavlju dana je teorijska podloga svih pojmova koji su bili korišteni prilikom izrade ovog rada. Ista služi za bolje razumijevanje problematike i načina rada razvijenog upravljačkog programa.

2.1. Eulerovi kutovi

Orijentacija jednog koordinatnog sustava u odnosu na drugi može biti određena sa tri kuta α, β i γ ili ϕ, θ i ψ koje nazivamo Eulerovi kutovi. Značenje pojedinih kutova ovisi o redoslijedu rotacija te je prije primjene potrebno proučiti o kojoj konvenciji se radi. Redoslijeda rotacija, tj. konvencija ima 12 i podijeljene su u dvije skupine:

1. Eulerovi kutovi (z-x-z, x-y-x, y-z-y, z-y-z, x-z-x, y-x-y)
2. Tait–Bryanovi kutovi (x-y-z, y-z-x, z-x-y, x-z-y, z-y-x, y-x-z)



Slika 2.1: Eulerovi kutovi po konvenciji z-x-z

Slika 2.1 prikazuje redoslijed rotacija po konvenciji z-x'-z'' u unutarnjim rotacijama ili z-x-z u vanjskim rotacijama. Pri tome kut α predstavlja rotaciju oko z-osi, kut β oko N-osi (ili x'-osi), a kut γ oko Z-osi (ili z''-osi). Danas se najčešće koristi konvencija z-y'-x'' ili z-y-x gdje se kutovi ϕ, θ i ψ nazivaju valjanje, poniranje i skretanje (eng. *yaw, pitch, roll*).

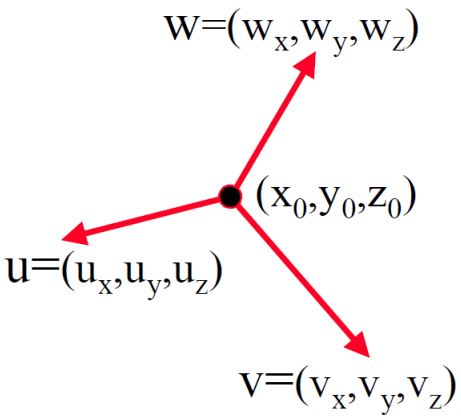
2.2. Matrica rotacije

Matrica rotacije $\mathbf{R}$ je 3×3 matrica koja opisuje orijentaciju nekog koordinatnog sustava u prostoru. Često se naziva i DCM matrica (Direction Cosine Matrix). Njezin opći oblik izgleda ovako:

$$\mathbf{R} = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \tag{2.1}$$

Ukoliko je koordinatni sustav zadan sa tri trodimenzionalna vektora $\vec{u}$, $\vec{v}$ i $\vec{w}$ kao na slici 2.2 tada orijentacija tog koordinatnog sustava glasi:

$$\mathbf{R} = \begin{bmatrix} \vec{u} & \vec{v} & \vec{w} \end{bmatrix} = \begin{bmatrix} u_x & v_x & w_x \\ u_y & v_y & w_y \\ u_z & v_z & w_z \end{bmatrix} \tag{2.2}$$



Slika 2.2: Koordinatni sustav opisan sa tri vektora

2.2.1. Transformacija iz Eulerovih kutova u matricu rotacije

Matrica transformacije $\mathbf{R_E}$ je izvedena prema x-y-z konvenciji množenjem triju matrica $\mathbf{R_X}$, $\mathbf{R_Y}$ i $\mathbf{R_Z}$ od kojih svaka od njih sadrži iznos rotacije oko određene osi.

$$\mathbf{R_X}(\phi) = \text{Roll}(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \tag{2.3}$$

$$\mathbf{R_Y}(\theta) = \text{Pitch}(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.4}$$

$$\mathbf{R_Z}(\psi) = \text{Yaw}(\psi) = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$$\mathbf{R_E} = \begin{bmatrix} \cos\theta\cos\psi & \cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi & \sin\phi\sin\psi - \cos\phi\sin\theta\cos\psi \\ -\cos\theta\sin\psi & \cos\phi\cos\psi - \sin\phi\sin\theta\sin\psi & \sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi \\ \sin\theta & -\sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix} \tag{2.6}$$

2.3. Kvaternioni

U matematici su kvaternioni (eng. *Quaternions*) algebarsko proširenje kompleksnih brojeva. Skup kvaterniona se označava sa $\mathbb{H}$ u čast irskom matematičaru Williamu Rowanu Hamiltonu koji ih je prvi formulirao. Kvaternion ima četiri dimenzije – jednu realnu i tri imaginarne. Svaka imaginarna dimenzija ima svoju vrijednost za $\sqrt{-1}$ što može biti i, j ili k. Veličina q predstavlja kvaternion:

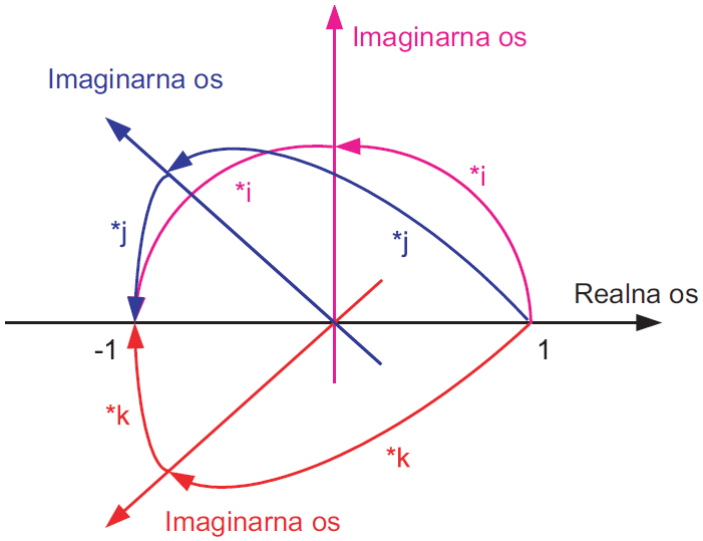
$$q = a + ib + jc + kd \tag{2.7}$$

Pravila koja opisuju i povezuju imaginarne dimenzije su:

$$\begin{aligned} i^2 = j^2 = k^2 = -1 \\ i \cdot j = -j \cdot i = k \\ j \cdot k = -k \cdot j = i \\ k \cdot i = -i \cdot k = j \end{aligned} \tag{2.8}$$

Tri različite vrijednosti za $\sqrt{-1}$ postoje jer su kvaternioni definirani u četverodimenzionalnom prostoru što znači da postoje tri različita puta iz točke T_1 u točku T_2. Na slici 2.3 grafički su predočena ta tri puta.

Prava svrha kvaterniona je različita od svrhe kompleksnih brojeva. Pomoću kompleksnog broja prikazuje se vektor u dvodimenzionalnom prostoru dok se pomoću kvaterniona prikazuje rotacija u trodimenzionalnom prostoru. Prilično je teško dati neko fizikalno



Slika 2.3: Mogući putovi od točke T_1 do točke T_2 [6]

objašnjenje kvaterniona, to je jednostavno neka veličina koja predstavlja rotaciju. Najbolji način za njegovo predstavljanje bio bi zapis:

$$Q = \cos\left(\frac{\alpha}{2}\right) + i\left[x \cdot \sin\left(\frac{\alpha}{2}\right)\right] + j\left[y \cdot \sin\left(\frac{\alpha}{2}\right)\right] + k\left[z \cdot \sin\left(\frac{\alpha}{2}\right)\right], \tag{2.9}$$

gdje je:

α - kut rotacije

x, y, z - komponente jediničnog vektora koji predstavlja os rotacije

Treba napomenuti da kompleksno i kod kvaterniona predstavlja rotaciju od 180° oko osi x, kompleksno j rotaciju za 180° oko osi y, a kompleksno k rotaciju za 180° oko osi z. Iz tog slijedi da $i^2 = -1$ predstavlja rotaciju od 360° oko osi x. Iz toga proizlazi da su kvaternion $Q_1 = a + ib + jc + kd$ i kvaternion $Q_2 = -a - ib - jc - kd$ isti, tj. da ova dva kvaterniona predstavljaju istu rotaciju. Osim do sada prikazana dva formata zapisivanja kvaterniona, oni se mogu zapisati i u vektorskom formatu na sljedeći način:

$$Q = \begin{bmatrix} q_w \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\alpha}{2}\right) \\ x \cdot \sin\left(\frac{\alpha}{2}\right) \\ y \cdot \sin\left(\frac{\alpha}{2}\right) \\ z \cdot \sin\left(\frac{\alpha}{2}\right) \end{bmatrix} \tag{2.10}$$

Različite literature koriste različite načine notacije kvaterniona. U tablici 2.1 dane su neke od najčešćih. U ovom radu korištena je *Lambova* notacija budući da se ona koristi i u C++ knjižnici za linearnu algebru – Eigen.

Tablica 2.1: Različite notacije kvaterniona

$Q = (q_0, q_1, q_2, q_3)$	*McCarthyev* zapis
$Q = (w, x, y, z)$	*Lambov* zapis pomoću realnih brojeva
$Q = (s, \vec{v})$	vektorska notacija (s je realni broj, a $\vec{v}$ vektor)
$Q = \{q_0, [q_1, q_2, q_3]\}$	često korišten zapis pomoću realnih brojeva

Za razliku od Eulerovih kutova, jedan kvaternion može predstavljati i osnovnu i složenu rotaciju. To proizlazi iz njegove četverodimenzionalnosti. Prednost kvaterniona je ta što se izbjegava veliki proračun i skraćuje se vrijeme računanja. Algebra s kvaternionima je puno jednostavnija nego algebra s Eulerovim kutovima, a to se prvenstveno odnosi na množenje matrica. [6]

2.3.1. Normiranje kvaterniona

Neka je $Q = q_w + i \cdot q_x + j \cdot q_y + k \cdot q_z$. Normiranje kvaterniona Q vrši se tako da se prvo izračuna norma n, pa se onda svi članovi kvaterniona podijele s tom normom:

$$n = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2} \tag{2.11}$$

$$Q_n = \frac{q_w}{n} + i \cdot \frac{q_x}{n} + j \cdot \frac{q_y}{n} + k \cdot \frac{q_z}{n} \tag{2.12}$$

Za normirani kvaternion vrijedi sljedeće svojstvo:

$$q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1 \tag{2.13}$$

2.3.2. Transformacija iz kvaterniona u matricu rotacije

Često je zgodno kvaternion prebaciti u matricu rotacije (DCM matricu). Ako je kvaternion zadan u obliku (2.10) tada se prebacivanje vrši prema sljedećim formulama:

$$\mathbf{R_Q} = \begin{bmatrix} q_w^2 + q_x^2 - q_y^2 - q_z^2 & 2 \cdot (q_x \cdot q_y - q_z \cdot q_w) & 2 \cdot (q_x \cdot q_z + q_y \cdot q_w) \\ 2 \cdot (q_x \cdot q_y + q_z \cdot q_w) & q_w^2 - q_x^2 + q_y^2 - q_z^2 & 2 \cdot (q_y \cdot q_z - q_x \cdot q_w) \\ 2 \cdot (q_x \cdot q_z - q_y \cdot q_w) & 2 \cdot (q_y \cdot q_z + q_x \cdot q_w) & q_w^2 - q_x^2 - q_y^2 + q_z^2 \end{bmatrix} \tag{2.14}$$

2.3.3. Transformacija iz matrice rotacije u kvaternion

Ova transformacija je nešto složenija nego transformacija kvaterniona u matriu rotacije. S obzirom da se radi o normiranom kvaternionu onda za njega vrijedi sljedeća relacija:

$$q_w^2 + q_x^2 + q_y^2 + q_z^2 = 1 \tag{2.15}$$

iz koje slijedi:

$$q_w = \sqrt{\frac{1 + R_{11} + R_{22} + R_{33}}{4}} = \sqrt{\frac{1 + \mathrm{tr}(\mathbf{R})}{4}} \tag{2.16}$$

gdje je suma dijagonalnih elemenata matrice $(R_{11} + R_{22} + R_{33})$ $\mathrm{tr}(\mathbf{R})$ – trag matrice $\mathbf{R}$. Da bi se mogao izračunati q_w, izraz ispod korijena mora biti veći od nule. U slučaju da je taj uvjet ispunjen, onda se članovi kvaterniona računaju prema sljedećim izrazima:

$$q_w = \sqrt{\frac{1 + \mathrm{tr}(\mathbf{R})}{4}} \tag{2.17}$$

$$S = \frac{1}{2 \cdot \sqrt{1 + \mathrm{tr}(\mathbf{R})}} \tag{2.18}$$

$$q_x = S \cdot (R_{32} - R_{23}) \tag{2.19}$$

$$q_y = S \cdot (R_{13} - R_{31}) \tag{2.20}$$

$$q_z = S \cdot (R_{21} - R_{12}) \tag{2.21}$$

Ako je izraz ispod korijena manji ili jednak nuli, tada je potrebno pronaći najveći član na dijagonali matrice $\mathbf{R}$. Ovisno o tome postoje tri moguća načina za računanje članova kvaterniona i oni su prikazani u tablici 2.2.

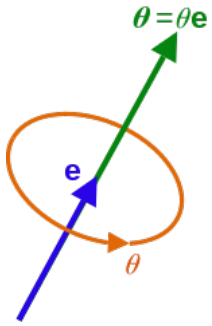
Tablica 2.2: Izrazi za računanje članova kvaterniona

Najveći član	R_{11}	R_{22}	R_{33}
S	$2\sqrt{1 + R_{11} - R_{22} - R_{33}}$	$2\sqrt{1 + R_{22} - R_{11} - R_{33}}$	$2\sqrt{1 + R_{33} - R_{11} - R_{22}}$
q_w	$\frac{R_{32} - R_{23}}{S}$	$\frac{R_{13} - R_{31}}{S}$	$\frac{R_{21} - R_{12}}{S}$
q_x	$\frac{S}{4}$	$\frac{R_{12} + R_{21}}{S}$	$\frac{R_{13} + R_{31}}{S}$
q_y	$\frac{R_{12} + R_{21}}{S}$	$\frac{S}{4}$	$\frac{R_{23} + R_{32}}{S}$
q_z	$\frac{R_{13} + R_{31}}{S}$	$\frac{R_{23} + R_{32}}{S}$	$\frac{S}{4}$

2.4. Axis–angle zapis rotacije

Axis–angle prikaz rotacije čini trodimenzionalni vektor $\boldsymbol{\theta}$ (2.22). Taj vektor se sastoji od jediničnog vektora $\mathbf{e}$ koji ukazuje na smjer osi vrtnje i kuta θ koji određuje iznos rotacije oko te osi. Grafički prikaz vektora $\boldsymbol{\theta}$ prikazuje slika 2.4.

$$\boldsymbol{\theta} = \theta\mathbf{e} = \begin{bmatrix} R_x & R_y & R_z \end{bmatrix} \tag{2.22}$$



Slika 2.4: Grafički prikaz vektora $\boldsymbol{\theta}$

2.4.1. Transformacija iz Axis–angle zapisa u matricu rotacije

$\boldsymbol{\theta}$ je trodimenzionalni vektor koji se ustvari sastoji od četiri veličine – parametara e_x, e_y i e_z koji određuju vektor $\mathbf{e}$ i kuta θ koji ih množi. Prije transofrmacije je potrebno izračunati iznos sva četiri parametra:

$$\theta = \sqrt{R_x^2 + R_y^2 + R_z^2} \tag{2.23}$$

$$e_x = \frac{R_x}{\theta} \tag{2.24}$$

$$e_y = \frac{R_y}{\theta} \tag{2.25}$$

$$e_z = \frac{R_z}{\theta} \tag{2.26}$$

Kada se zna iznos svih paramatara, transformacija u matricu rotacije vrši se prema sljedećim formulama:

$$\mathbf{R_{A-A}} = \begin{bmatrix} Ce_x^2 + \cos\theta & Ce_xe_y - e_z\sin\theta & Ce_xe_z + e_y\sin\theta \\ Ce_ye_x + e_z\sin\theta & Ce_y^2 + \cos\theta & Ce_ye_z - e_x\sin\theta \\ Ce_ze_x - e_y\sin\theta & Ce_ze_y + e_x\sin\theta & Ce_z^2 + \cos\theta \end{bmatrix} \tag{2.27}$$

gdje je $C = 1 - \cos\theta$

2.4.2. Transformacija iz matrice rotacije u Axis–angle zapis

Transformacija iz matrice rotacije **R** (2.1) u Axis–angle zapis vrši se prema sljedećim formulama:

$$\theta = \arccos\left(\frac{1}{2}\left[R_{11} + R_{22} + R_{33} - 1\right]\right) \tag{2.28}$$

$$e_x = \frac{R_{32} - R_{23}}{2\sin\theta} \tag{2.29}$$

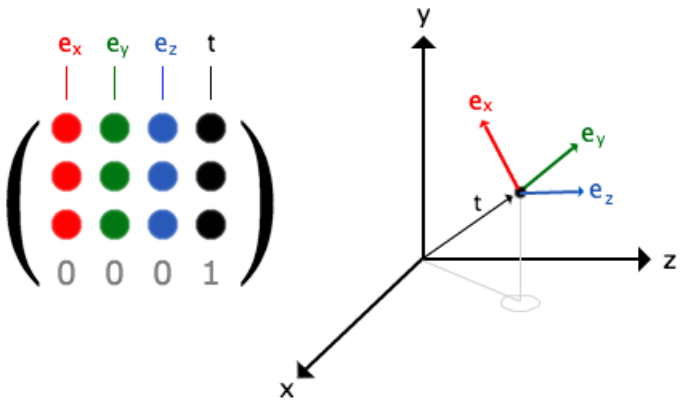
$$e_y = \frac{R_{13} - R_{31}}{2\sin\theta} \tag{2.30}$$

$$e_z = \frac{R_{21} - R_{12}}{2\sin\theta} \tag{2.31}$$

2.5. Matrica homogene transformacije

Pozicija i orijentacija jednog koordinatnog sustava u odnosu na drugi definirana je homogenom transformacijom, tj. matricom homogene transformacije. Matrica homogene transformacije je 4×4 matrica koja se sastoji od matrice rotacije (DCM matrice) i vektora translacije. Na slici 2.5 je slikovito objašnjen oblik matrice homogene transformacije, a njezin opći oblik glasi:

$$\mathbf{T} = \begin{bmatrix} a & b & c & t_x \\ d & e & f & t_y \\ g & h & i & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{2.32}$$



Slika 2.5: Slikovito objašnjenje matrice homogene transformacije

2.5.1. Transformacija iz jednog u drugi koordinatni sustav

Ukoliko u prostoru postoje dva koordinatna sustava O_1 i O_2, a njihov položaj i orijentacija je definirana u nekom trećem, baznom, koordinatnom sustavu O tada vrijede jednakosti:

$$\overset{O}{O1}\mathbf{T} \cdot \overset{O_1}{O_2}\mathbf{T} = \overset{O}{O_2}\mathbf{T} \tag{2.33}$$

$$\overset{O_1}{O_2}\mathbf{T} = \overset{O}{O1}\mathbf{T}^{-1} \cdot \overset{O}{O_2}\mathbf{T} \tag{2.34}$$

2.6. Kinematička analiza robota

Robot se može modelirati kao lanac krutih tijela (članaka) međusobno povezanih zglobovima gdje se na početku lanca nalazi nepomična baza robota, a na kraju lanca prihvatnica robota. Da bi se robot kretao u trodimenzionalnom prostoru potrebno je upravljati pozicijom i orijentacijom prihvatnice. Zbog toga je neophodno odrediti vezu između parametara zglobova robota te pozicije i orijentacije prihvatnice. Postoje dva osnovna pristupa u opisivanju kinematike robota, a to su:

1. Denavit–Hartenbergov analitički pristup
2. Numerički pristup na bazi Rodriguesove formule

Oba pristupa su identična po svojoj kompleksnosti za slučaj kada se glavne osi inercije poklapaju sa osima zglobova i zajedničkom normalom, što je česta pojava kod industrijskih robota. Denavit–Hartenbergov pristup je pogodniji kada se formiraju kinematičke jednadžbe robota, a pristup pomoću Rodriguesove formule je prikladiniji za formiranje dinamičkih jednadžbi robota. Denavit–Hartnbergov pristup se zasniva na homogenim transformacijama, koje definiraju položaj i orijentaciju jednog koordinatnog sustava u odnosu na drugi. U vezi sa pozicioniranjem robota pojavljuje se problem na koji način se zadaje željena pozicija robota. Razlikuju se dva načina zadavanja koordinata robota:

1. preko unutrašnjih koordinata robota (odnosno koordinata zglobova) i
2. preko vanjskih koordinata robota.

2.6.1. Pojam vanjskih i unutrašnjih koordinata

Pod unutrašnjim koordinatama jednog robota podrazumijevaju se skalarne veličine koje opisuju relativni položaj jednog segmenta u odnosu na drugi. Kod rotacijskog zgloba

unutrašnja koordinata je kut zakreta u zglobu, dok je kod translacijskog kinematskog para unutrašnja koordinata predstavljena linearnim pomakom uzduž osi zgloba. Unutrašnje koordinate se najčešće obilježavaju sa q_i $(i = 1, \ldots, n)$ i čine vektor unutrašnjih koordinata:

$$\mathbf{q} = [q_1, q_2, \ldots, q_n]^T \tag{2.35}$$

Broj unutrašnjih koordinata jednak je broju stupnjeva slobode gibanja robota.
Vanjskim koordinatama se opisuje položaj prihvatnice robota u odnosu na nepokretni koordinatni sustav koji je vezan za bazu robota. Pozicija prihvatnice se najčešće opisuje Descartesovim koordinatama, a orijentacija pomoću Eulerovih kutova ili pomoću Axis–angle zapisa. Vektor vanjskih koordinata čine koordinate neke točke na prihvatnici i kutevi zakreta koordinatnog sustava vezanog za prihvatnicu u odnosu na nepokretni koordinatni sustav u bazi robota. [7]

2.7. Direktna kinematika

Pri promjeni unutrašnjih koordinata robota mijenjaju se i vanjske. Problem određivanja vektora vanjskih koordinata preko zadanog vektora unutrašnjih koordinata naziva se direktni kinematički problem. Da bi se riješio direktni kinematički problem potrebno je prvo ustanoviti simboličku shemu robota u odnosu prema odabranom nepokretnom koordinatnom sustavu. Simbolička shema treba sadržavati sve bitne dimenzije robota. Položaj robota gdje sve unutrašnje koordinate imaju vrijednosti jednake nuli $(q_i = 0)$ naziva se nultim položajem. Slijedeći korak u rješavanju direktnog kinematičkog problema je pridruživanje koordinatnih sustava pojedinim segmentima robota kojih mora biti najmanje onoliko koliko ima stupnjeva slobode gibanja. To pridruživanje koordinatnih sistema najčešće se obavlja pomoću Denavit–Hartenbergove metode. [7]

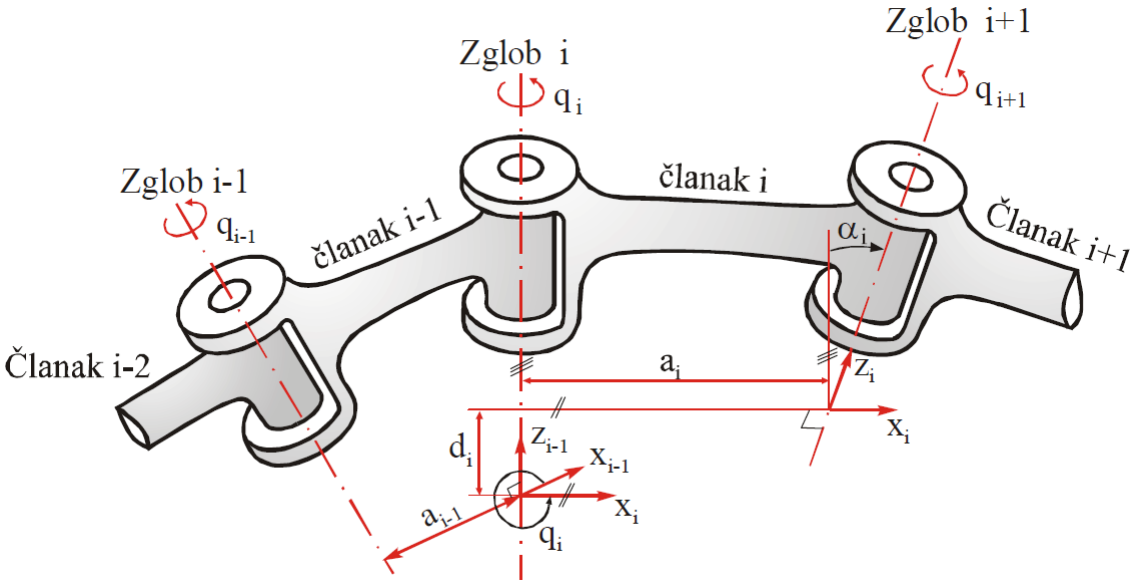
2.7.1. Denavit–Hartenbergova notacija

Da bi se mogle prikazivati određene fizičke i geometrijske veličine vezane za kretanje elemenata robota, potrebno je definirati određene referentne koordinatne sustave u kojima će se sve navedene veličine prikazivati i računati. Pomoću Denavit–Hartenbergove notacije određuje se veza između članaka (segmenata) robota. Prema toj konvenciji numeracija

članaka se vrši tako da je baza robota nulti članak, prvi pokretni nosi oznaku 1 i tako sve do slobodnog kraja. U slučaju da se radi o rotacijskim zglobovima tada se svaki članak može opisati sa dvije veličine:

1. dužina a_i – najkraći razmak između osi i–tog i $(i+1)$–vog zgloba
2. kut zakreta α_i – kut između osi u ravnini normalnoj na a_i, a mjeri se u smjeru suprotnom kretanju kazaljke na satu oko osi a_{i+1}

Svaka os zgloba ima dvije normale a_{i+1} i a_i, a razmak između njih se označava sa d_i. Označi li se lokalni koordinatni sustav i–tog članka sa $O_i x_i y_i z_i$, njegovo ishodište će biti postavljeno u točki presjeka zajedničke normale između osi i–tog i $(i+1)$–vog zgloba i same osi i–tog zgloba. U slučaju da se osi zglobova sijeku, ishodište koordinatnog sustava tada se postavlja u točku presjeka tih osi. Za slučaj da su osi paralelne, ishodište se bira tako da razmak uzduž osi zgloba (d) bude jednak nuli za sljedeći članak čije je ishodište definirano. Svi Denavit-Hartenbergovi parametri za rotacijski zglob su prikazani na slici 2.6.



Slika 2.6: Denavit–Hartenbergovi parametri za rotacijski zglob [7]

2.7.2. Rješenje direktnog kinematičkog problema

Nakon što su preko Denavit–Hartenbergovih parametara definirani koordinatni sustavi svih zglobova njihov položaj i orijentacija u prostoru može se prikazati u obliku lokalnih

homogenenih matrica transformacije $\mathbf{T_i}$. Konačan izraz za određivanje pozicije i orijentacije vrha alata robota iz njegove baze glasi:

$$ {}^{0}_{n}\mathbf{T} = \mathbf{T_1} \cdot \mathbf{T_2} \cdots \mathbf{T_n} \tag{2.36} $$

2.8. Kalibracija pozicije vrha alata

Kalibracija vrha alata ili pivotiranje alata je tehnika koja se u robotici koristi kako bi se precizno odredile koordinate vrha alata. Potrebno je dovesti vrh alata robota u istu točku u prostoru sa minimalno tri različite konfiguracije robota. Za svaku točku spremaju se njezine koordinate u baznom koordinatnom sustavu robota u trodimenzionalni vektor $\mathbf{t_i}$ i njezina orijentacija u DCM matricu $\mathbf{R_i}$. Jednadžba koja treba biti zadovoljena glasi [8]:

$$ \begin{bmatrix} \mathbf{R_2} - \mathbf{R_1} \\ \mathbf{R_3} - \mathbf{R_2} \\ \vdots \\ \mathbf{R_n} - \mathbf{R_{n-1}} \end{bmatrix} \mathbf{X_t} = \begin{bmatrix} \mathbf{t_1} - \mathbf{t_2} \\ \mathbf{t_2} - \mathbf{t_3} \\ \vdots \\ \mathbf{t_{n-1}} - \mathbf{t_n} \end{bmatrix} \implies \mathbf{A}\mathbf{X_t} = \mathbf{B} \tag{2.37} $$

iz koje slijedi da je:

$$ \mathbf{X_t} = (\mathbf{A}^T \times \mathbf{A})^{-1}\mathbf{A}^T \times \mathbf{B} \tag{2.38} $$

$\mathbf{X_t}$ je trodimenzionalni vektor koji se sastoji od X, Y i Z koordinata vrha alata. Maksimalna greška kalibracije δ može se izračunati prema sljedećoj formuli [8]:

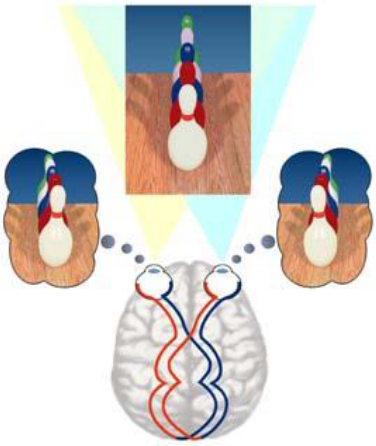
$$ \delta = \left\| \begin{bmatrix} \mathbf{R_2} - \mathbf{R_1} \\ \mathbf{R_3} - \mathbf{R_2} \\ \vdots \\ \mathbf{R_n} - \mathbf{R_{n-1}} \end{bmatrix} \mathbf{X_t} - \begin{bmatrix} \mathbf{t_1} - \mathbf{t_2} \\ \mathbf{t_2} - \mathbf{t_3} \\ \vdots \\ \mathbf{t_{n-1}} - \mathbf{t_n} \end{bmatrix} \right\| = \sqrt{\sum_{i=0}^{n-1} \left| (\mathbf{R_{i+1}} - \mathbf{R_i})\mathbf{X_t} - \mathbf{t_i} + \mathbf{t_{i+1}} \right|^2} \tag{2.39} $$

2.9. Stereovizija

Stereovizija je proces u vizualnoj percepciji prostora koji služi za dobivanje informacije o dubini iz dviju različitih projekcija prostora. Kao i mnoge druge grane znanosti i ova grana, koja se bavi robotskim stereovizijskim sustavima, ima svoje temelje u biološkim sustavima. Snalaženje čovjeka u prostoru bez stereovizije bilo bi znatno otežano. [9]

2.9.1. Biološki stereovizijski sustavi

Za razliku od nekih životinja, ljudi imaju oči smještene u tzv. frontalno–paralelnoj konfiguraciji što omogućuje promatranje istog prostora pod lagano različitim kutom kao što je prikazano na slici 2.7. Dva različita pogleda imaju mnoge stvari zajedničke, ali svako oko prikupi druge informacije. Svako oko registrira prostor koji se potom šalje u mozak na analizu. Kada dvije slike istovremeno dođu u centar mozga zadužen za vid, one se ujedinjuju u jednu sliku na takav način da se prvo nađu sličnosti na slikama, a potom se obrade malene razlike što na kraju rezultira 3D percepcijom prostora.

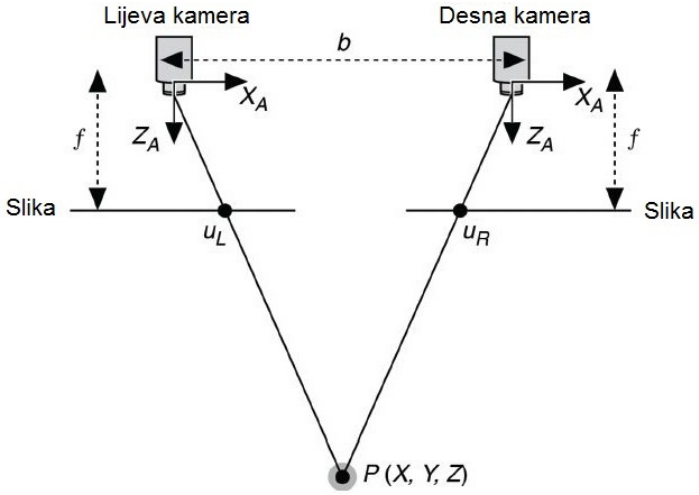


Slika 2.7: Biološki stereovizijski sustav [9]

2.9.2. Računalni stereovizijski sustavi

Računalna stereovizija je dio područja računalne vizije. Dvije kamere, koje su razmaknute za neku udaljenost, obuhvate slike istog prizora. Računalna aplikacija uspoređuje slike te pronalazi zajedničke elemente koji su na slikama na različitim mjestima zbog različitih kutova pod kojima kamere promatraju prizor. Računaju se iznosi razlika u koordinatama pojedinih predmeta na lijevoj i desnoj slici te se na taj način dobiva mapa razlika na temelju čega se dobiva udaljenost predmeta od ishodišta stereovizijske kamere.

Nužan uvjet za određivanje koordinata točke P je njezina vidljivost na lijevoj i na desnoj slici. Slika 2.8 prikazuje pojednostavljeni model stereovizijskog sustava gdje su: f – žarište kamere, X_A – X–os kamere, Z_A – optička os kamere, P – točka u prostoru sa koordinatama (X, Y, Z), u_L – projekcija točke P na slici lijeve kamere, u_R – projekcija točke P na slici



Slika 2.8: Pojednostavljeni model stereovizijskog sustava

desne kamere. Prvo se računa preklapanje lijeve i desne slike d:

$$d = u_L - u_R \tag{2.40}$$

Zatim se iz sličnosti trokuta dobije jednadžba za izračun dubine Z:

$$\frac{b-d}{Z-f} = \frac{b}{Z} \implies Z = \frac{fb}{d} \tag{2.41}$$

2.10. Euklidska udaljenost u 3D prostoru

Euklidska udaljenost je najkraći razmak između dvije točke u prostoru. Ako je točka T_1 zadana koordinatama (x_1, y_1, z_1), a točka T_2 koordinatama (x_2, y_2, z_2) tada se njihova euklidska udaljenost računa prema sljedećoj formuli:

$$E = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2} \tag{2.42}$$

2.11. TCP protokol

TCP (Transmission Control Protocol) je jedan od osnovnih protokola unutar IP grupe protokola. Korištenjem TCP protokola aplikacija na nekom od hostova umreženog u računalnu mrežu kreira virtualnu konekciju prema drugom hostu, te putem te ostvarene konekcije zatim prenosi podatke. Stoga ovaj protokol spada u grupu tzv. spojnih protokola, za razliku od bespojnih protokola kakav je primjerice UDP (User Datagram Protocol). TCP garantira pouzdanu isporuku podataka u kontroliranom redoslijedu od pošiljatelja

prema primatelju. Osim toga, TCP pruža i mogućnost višestrukih istovremenih konekcija prema jednoj aplikaciji na jednom hostu od strane više klijenata. TCP osigurava sigurnu, nedupliciranju poruku udaljenom korisniku. [10]
TCP upotrebljava određen raspon portova kojima razdjeljuje primjenske programe na strani pošiljatelja i primatelja. Svaka strana TCP konekcije ima dodijeljenu 16-bitnu oznaku za obje strane aplikacije (slanje, primanje). Portovi su u osnovi podijeljeni u 3 kategorije:

1. poznati portovi (0 – 1023)
2. registrirani portovi (1024 – 49150)
3. dinamički/privatni portovi (49151 – 65535)

2.11.1. Uspostavljanje veze

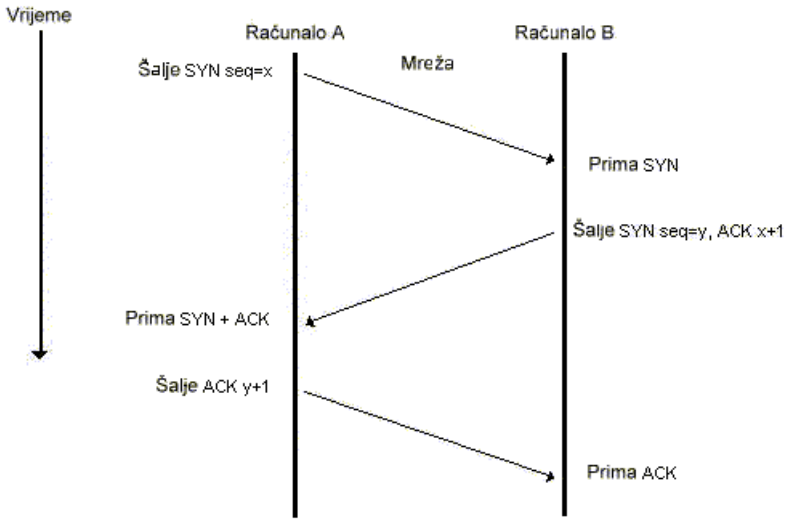
Proces koji se izvodi na jednom računalu prilikom uspostavljanja veze s procesom na nekom drugom računalu naziva se uspostavljanje veze (eng. *Connection Establishment*). Računalo koje traži uspostavu veze naziva se klijent (eng. *client*), a drugo računalo se naziva poslužitelj (eng. *server*). Klijentski proces informira klijentski TCP da želi uspostaviti vezu s poslužiteljem. Klijentsko računalo tada šalje poslužitelju prvi specijalni segment. Poslužitelj odgovara drugim specijalnim TCP segmentom i konačno klijent odgovara trećim specijalnim segmentom. Ova procedura se naziva "three-way handshake". U nastavku slijedi objašnjenje spomenutih segmenata:

1. Klijent prvi šalje specijalni TCP segment poslužitelju. Taj specijalni segment ne sadrži podatke aplikacijske razine. U zaglavlju segmenta jedan je od bitova zastavica – tzv. SYN bit, postavljen na 1. Iz tog razloga, taj specijalni segment zove se SYN segment. Nadalje, klijent odabire inicijalni redni broj (client_isn) i stavlja ga u polje za redni broj inicijalnog TCP SYN segmenta.
2. Pod pretpostavkom da IP datagram koji sadrži TCP SYN segment stigne do poslužitelja, on izdvaja TCP SYN segment iz datagrama, alocira TCP spremnik i varijable i šalje segment kojim odobrava uspostavu veze klijentu. Taj segment odobravanja veze također ne sadrži podatke aplikacijske razine, ali sadrži tri važne informacije u zaglavlju segmenta. Prvo, SYN bit je postavljen na 1. Drugo, Acknowledgment

polje zaglavlja TCP segmenta se namješta na isn+1. Na kraju, poslužitelj odabire svoj inicijalni redni broj (server_isn) i stavlja vrijednost u polje zaglavlja TCP segmenta.

3. Kada klijent primi segment odobravanja veze, također alocira spremnik i varijable u vezi. Klijent tada šalje poslužitelju još jedan segment koji potvrđuje da je dobio segment odobravanja veze. To radi tako da stavi vrijednost server_isn+1 u acknowledgement polje zaglavlja. SYN bit postavlja se u 0 budući da je veza uspostavljena.

Kada se sva tri koraka obave, klijent i poslužitelj jedan drugome mogu slati segmente koji sadrže podatke. U svakom budućem segmentu SYN će biti postavljen na 0. Slika 2.9 prikazuje „three-way handshake" proceduru. [11]



Slika 2.9: „Three-way handshake" [11]

3. NDI – NORTHERN DIGITAL INC.

Northern Digital Incorporated je kanadska tvrtka koja se bavi izradom i razvojem mjernih sustava. Osnovao ju je Jerry Krist 1981. godine na Sveučilištu Waterloo. Vodeća je tvrtka u proizvodnji naprednih 3D mjernih sustava koji se koriste u medicini, industriji i akademskim zajednicama. Trenutno je u primjeni više od 45.000 njihovih mjernih sustava od čega je preko 20.000 u bolnicama i medicinskim centrima diljem svijeta. Njihovi optički sustavi za praćenje se u medicini se, između ostalog, primjenjuju u dentalnoj implatologiji, neurokirurgiji, ortopediji, operaciji kralježnice te transkranijalnoj magnetskoj stimulaciji koja je područje primjene u ovom radu.

3.1. Polaris Optical Tracking System

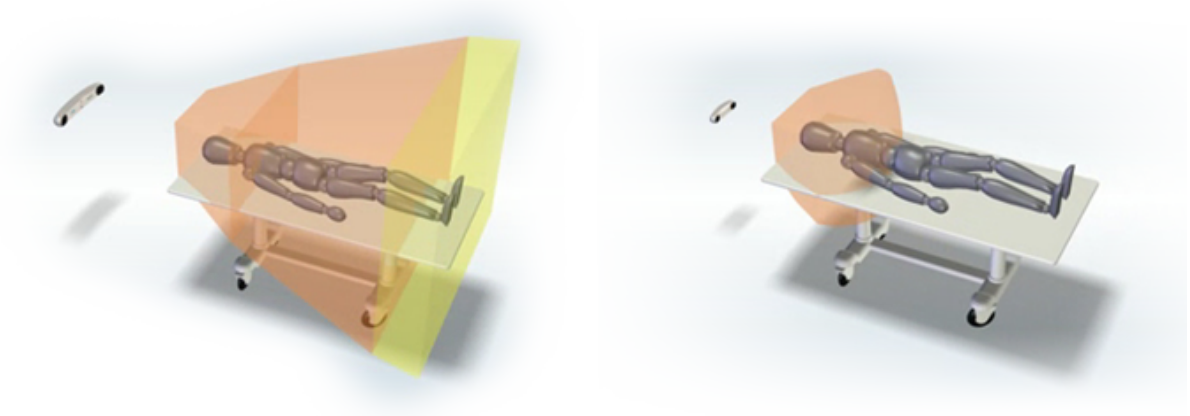
Optički sustav za praćenje Polaris jedna je od proizvodnih linija tvrtke NDI. Ona obuhvaća dvije stereovizijske kamere Polaris Vicra i Polaris Spectra [Slika 3.1], softversku podršku te razne alate, markere i pribor za njih. Pomoću Polaris sustava za praćenje moguće je pratiti 3D položaj i orijentaciju aktivnih ili pasivnih markera uz izuzetno veliku preciznost i pouzdanost. Optički sustav, tj. iluminator, emitira infracrvenu svjetlost koja se reflektira od retroreflektivnih pasivnih markera. Aktivni markeri infracrvenu svjetlost emitiraju sami. Reflektiranu ili emitaranu infracrvenu svjetlost primaju senzori u svakoj kameri te se preko stereovizije objašnjene u poglavlju 2.9 određuje pozicija markera u prostoru. Orijentaciju je moguće odrediti za alat sa minimalno tri markera.

Slika 3.1: Polaris Spectra i Polaris Vicra

Sustav Polaris se najviše primjenjuje u medicini u sljedećim područjima:

- Računalno podržana terapija
- Dentalna implantologija
- Kirurgija uha, grla i nosa
- Integracija s medicinskim robotima
- Neurokirurgija
- Ortopedija
- PET postupci
- Slikovno vođena radioterapija
- Kirurgija kralježnice
- Transkranijalna magnetska stimulacija

Glavna razlika između Polaris Vicre i Polaris Spectre je veličina mjernog volumena. Slika 3.2 prikazuje veličinu mjernog volumena u odnosu na čovjeka. Unutar tog volumena NDI garantira maksimalnu točnost i pouzdanost u određivanju pozicije i orijentacije markera. U tablici 3.1 navedene su glavne tehničke karakteristike sustava Polaris.

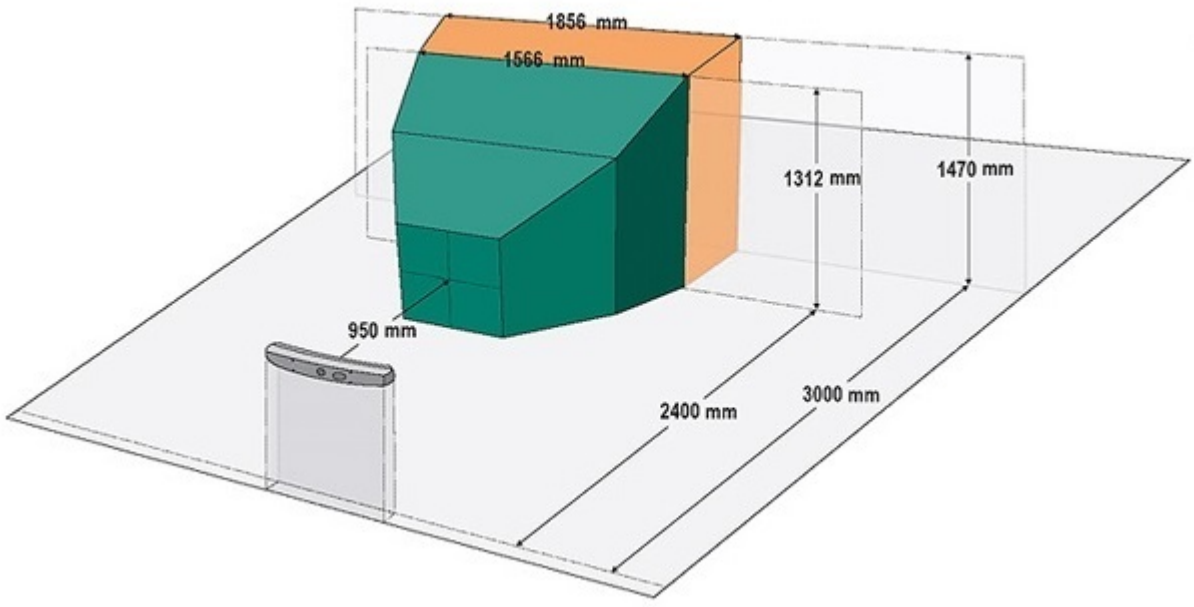


(a) Polaris Spectra **(b) Polaris Vicra**

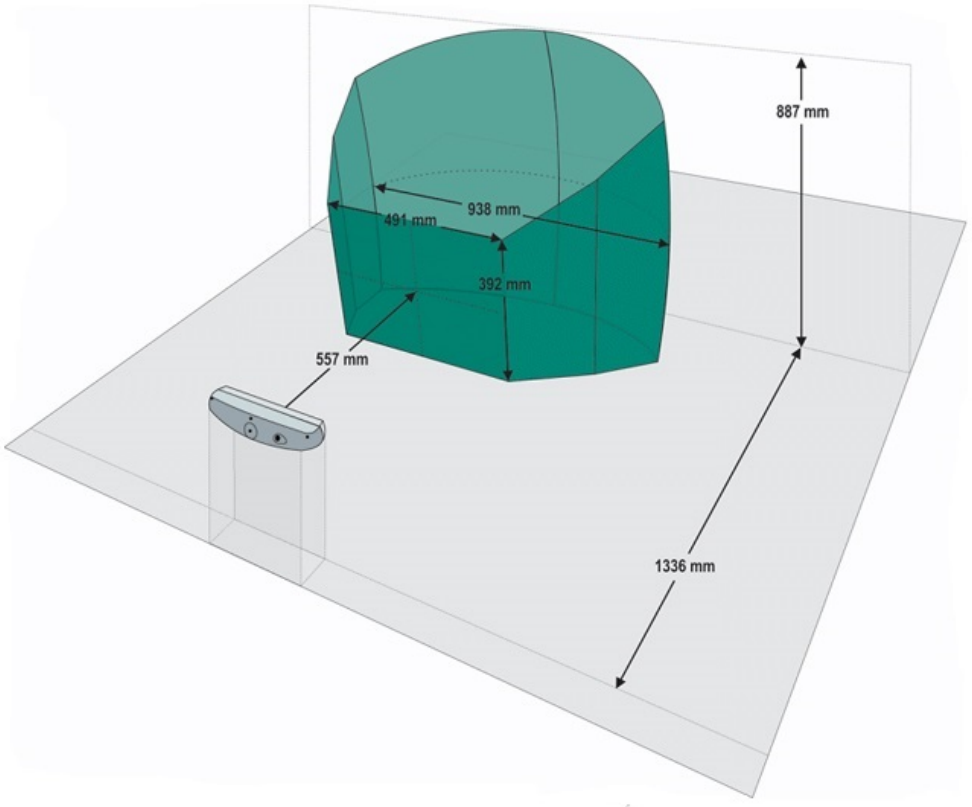
Slika 3.2: Područje mjernog volumena u odnosu na čovjeka

Tablica 3.1: Tehničke specifikacije sustava Polaris

	Polaris Spectra	**Polaris Vicra**
Točnost	0.25mm RMS	0.25mm RMS
Područje mjernog volumena	Slika 3.3	Slika 3.4
Maksimalna brzina ažuriranja	60Hz	20Hz
Radna temperatura	10°C do 40°C	10°C do 30°C
Maksimalni broj alata	15 pasivnih i 6 aktivnih	6 pasivna i 1 aktivni
Maksimalno markera na alatu	6 na jednoplanarnom, 20 na višeplanarnom	
Maksimalno markera ukupno	32 pasivina i 32 aktivna	32 pasivna
Dimenzije	613mm × 104mm × 86mm	273mm × 69mm × 69mm
Masa	1.9kg	0.8kg



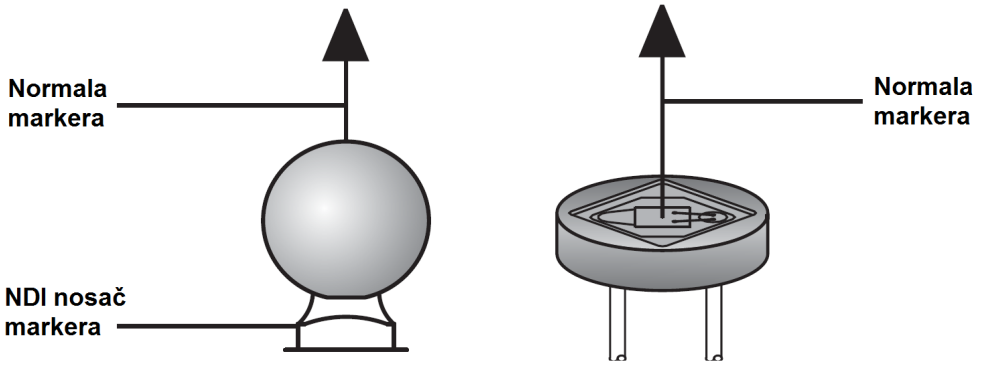
Slika 3.3: Polaris Spectra – mjerni volumen



Slika 3.4: Polaris Vicra – mjerni volumen

3.1.1. Markeri

U sustavu Polaris moguće je koristiti dvije vrste markera: aktivne i pasivne. Aktivni markeri sami imitiraju infracrvenu svjetlost i aktiviraju se električnim signalom. Pasivni markeri su retroreflektivne kuglice od kojih se svjetlost reflektira u svim smjerovima. Istraživanja su pokazala da nema razlike točnosti između aktivnih i pasivnih markera. [12]



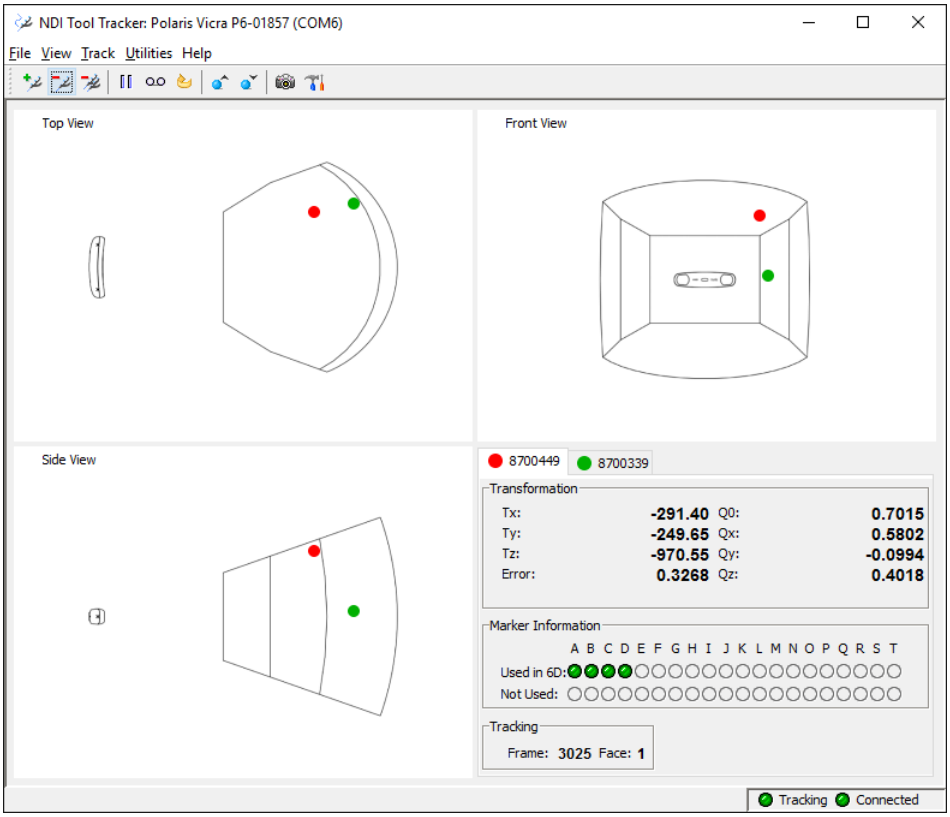
(a) Pasivni sferni marker **(b) Aktivni marker**

Slika 3.5: NDI markeri sa normalama

3.2. NDI Toolbox

NDI Toolbox je paket softverskih alata koji dolazi sa sustavom Polaris i služi za dijagnostiku, održavanje, ispitivanje i razvoj podrške. Programi unutar tog paketa su:

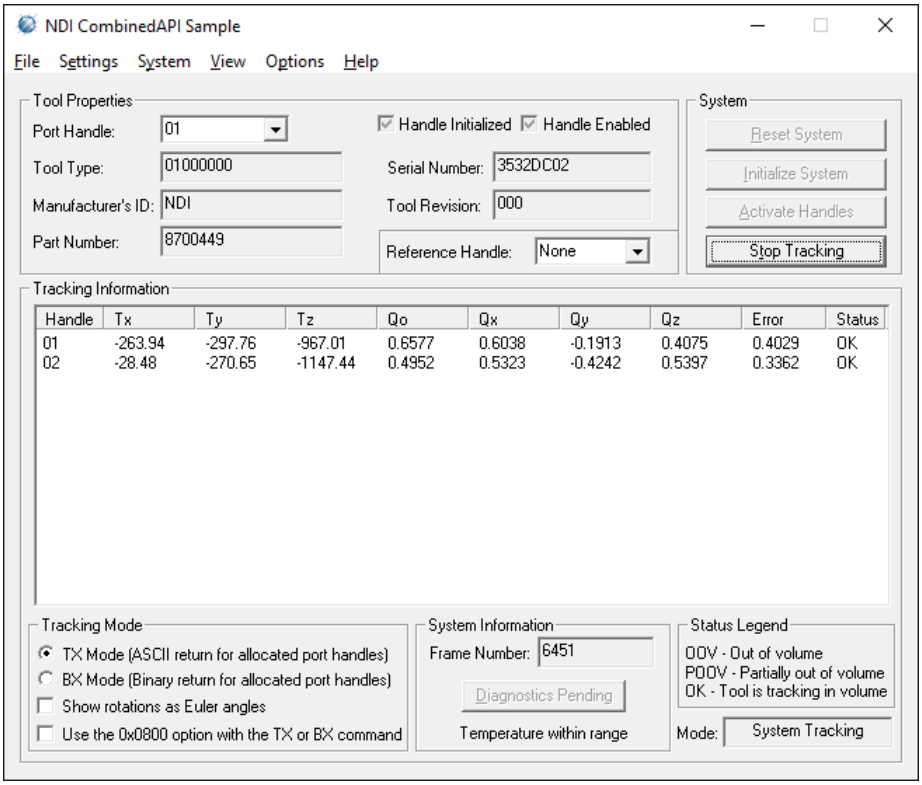
- NDI Configure – služi za pregled konfiguracije sustava, promjenu postavki i brzo identificiranje problema,
- NDI Image Capture – koristi se za prikaz slike od svake kamere što je korisno za otkrivanje potencijalnih smetnji,
- NDI Tool Tracker [Slika 3.6] – grafički i numerički prikazuje pozicije i orijentacije alata unutar mjernog volumena. Unutar tog programa moguće je i pivotirati alat, tj. precizno odrediti njegov vrh koristeći tehniku objašnjenu u poglavlju 2.8.



Slika 3.6: NDI Tool Tracker

3.3. NDI Polaris API

Polaris API (eng. *Application Program Interface*) je program otvorenog kôda (eng. *open source*) koji je moguće integrirati u vlastiti program. Njegovo sučelje je prikazano na slici 3.7. Napisan je u programskom jeziku C++. On prima položaj svakog markera koji kamera vidi, zatim prepoznaje prethodno učitane alate za praćenje i određuje njihov položaj i orijentaciju u prostoru. Taj program je korišten u ovom rada. Unutar tog programa dodan je programski kôd za TCP klijent. Kada se pritisne Start Tracking program se spaja na server i šalje mu položaj i orijentaciju svakog alata u obliku tekstualne poruke.



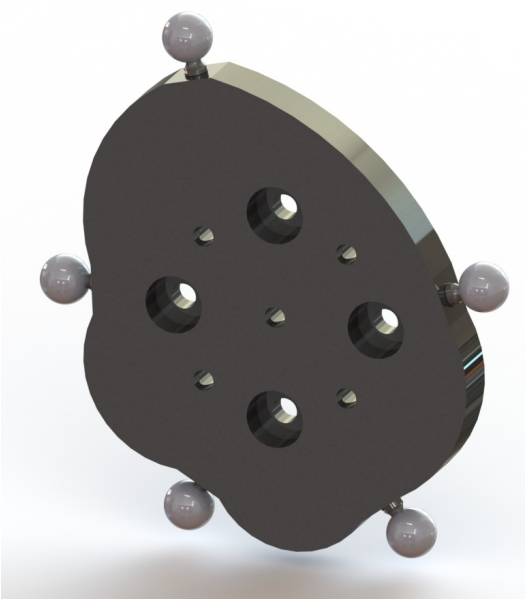
Slika 3.7: NDI Polaris API

3.4. NDI 6D Architect

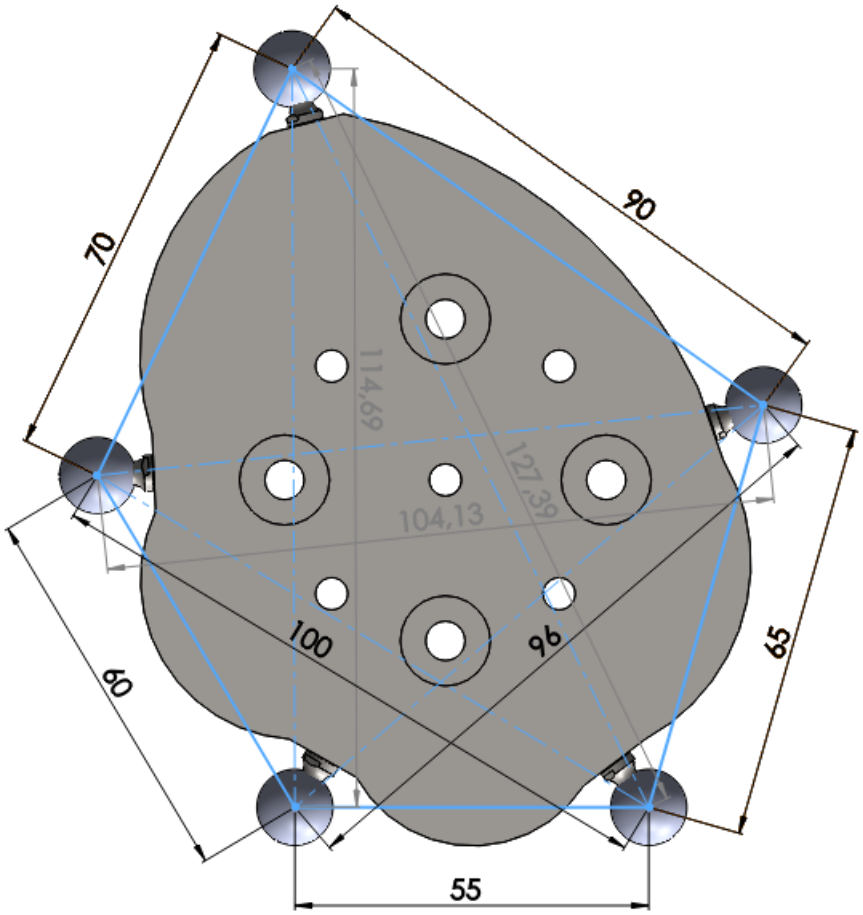
NDI 6D Architect je softverski alat koji se koristi za definiranje alata i stvaranje datoteke koja definira alat. Ekstenzija te datoteke je .rom. Kod praćenja pozicije i orijentacije alata prvo je potrebno učitati tu datoteku kako bi sustav mogao prepoznati alat u prostoru.

3.4.1. Kreiranje novog alata

U ovom diplomskom radu bilo je potrebno odrediti poziciju i orijentaciju vrha robota koji se može unutar mjernog volumena nalaziti u raznim nepogodnim konfiguracijama u kojima markeri nisu vidljivi. Zato je konstruiran novi alat koji omogućuje vidljivost barem tri markera iz raznih kutova. Taj alat sastoji se od pet pasivnih markera i prikazan je na slici 3.8, a njegova geometrija na slici 3.9. Alat je konstruiran u programskom paketu SolidWorks. Kod konstrukcije je trebalo paziti da svaka udaljenost između dva markera bude različita i da ta razlika bude minimalno 3.5mm. Udaljenosti i njihove razlike prikazene su u tablici 3.2. Postupak njegovog definiranja unutar programa 6D Architect je vrlo jednostavan. Prvi korak je odabir vrste alata i metode za određivanje pozicija svih markera. Moguće je ručno unijeti X, Y i Z koordinate svakog markera ili dati sustavu da sam snimi i odredi njihov položaj. Ovdje je korišten drugi način. Nakon što sustav snimi i odredi položaj svakog markera, treba definirati koordinatni sustav. Dovoljno je definirati bazu koordinatnog sustava u jednom od markera i jednu ravninu koja je određena sa tri markera. U sljedećem koraku moguće je ručno ili automatski odrediti normale svake kuglice. Zadnji korak je test alata i ukoliko on prođe dobiva se .rom datoteka koja definira alat. Tu datoteku potrebno je učitati u jednom od programa za praćenje (NDI Polaris API ili NDI Tool Tracker) kako bi sustav mogao prepoznati alat i ispisivati poziciju i orijentaciju prethodno definiranog koordinatnog sustava od alata.



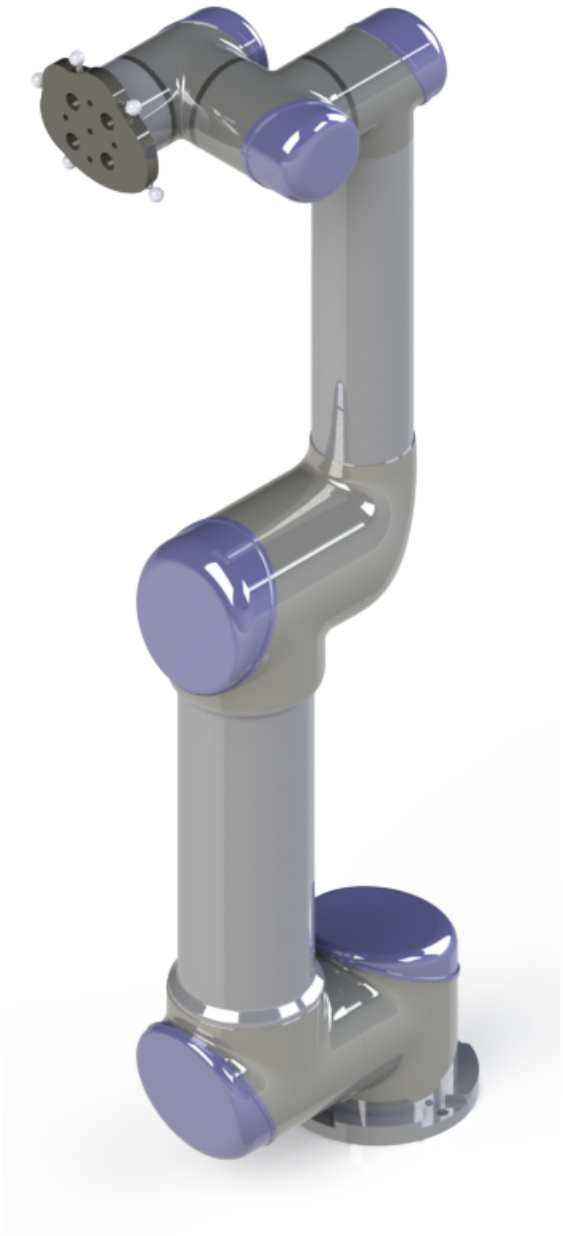
Slika 3.8: Alat M1



Slika 3.9: Geometrija alata M1

Tablica 3.2: Udaljenosti između markera i njihove razlike

Segment i	Udaljenost [mm]	Razlika [mm]
1	55	–
2	60	5
3	65	5
4	70	5
5	90	20
6	96	6
7	100	4
8	104,13	4,13
9	114,69	10,56
10	127,39	12,7

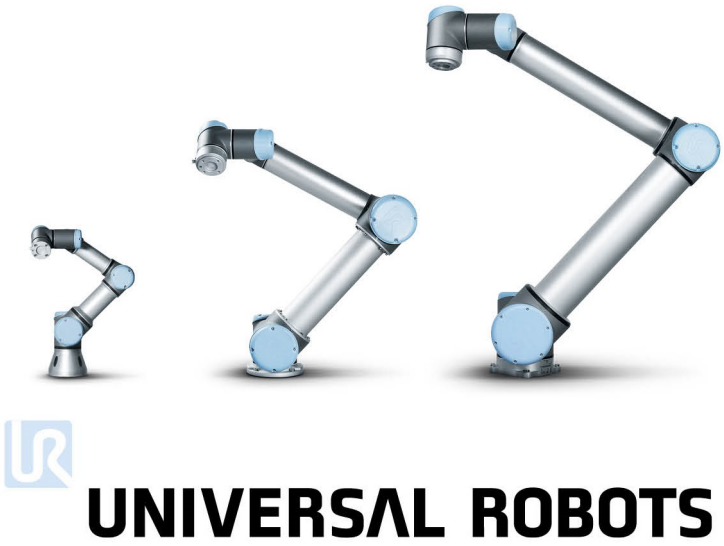


Slika 3.10: Alat M1 na robotu UR5

4. UNIVERSAL ROBOTS

Universal Robots proizvođač je fleksibilnih industrijskih robota sa sjedištem u Odenseu u Danskoj. Tvrtka su osnovali Esben Østergaard, Kasper Støy i Kristian Kassow 2005. godine. Tijekom zajedničkog istraživanja na Sveučilištu Syddansk došli do zaključka da tržištem robota dominiraju teški, skupi i nezgrapni roboti. Razvili su ideju da bi robotske tehnologije trebale biti dostupne malim i srednjim poduzećima. 2009. godine počeli su se prodavati prvi UR5 roboti na danskom i njemačkom tržištu. Od 2010. godine tvrtka stalno proširuje svoje aktivnosti. 2012. godine plasirali su na tržište drugi robot – UR10, a 2015. godine i treći robot – UR3. Universal Robots danas ima 140 zaposlenih i globalnu mrežu sa 200 distributera u 50 zemalja diljem svijeta. Njihove robote koriste i neka poznatija poduzeća kao što su Johnson & Johnson, VW, BMW, Oticon, itd.

Njihova tri glavna proizvoda su roboti UR3, UR5 i UR10 [Slika 4.1]. Brojke u imenu označavaju nosivost svakog robota u kilogramima. Svoj troje navedenih, vrlo su lagani 6–osni roboti čije su težine redom 11kg, 18kg i 28kg. Svi zglobovi mogu se rotirati $\pm 360°$ brzinom do $180° \, s^{-1}$, a zadnji zglob robota UR3 ima beskonačnu rotaciju. Točnost ponavljanja iznosi im ± 0.1mm. UR3, UR5 i UR10 su "Kolaborativni roboti" što znači da mogu raditi odmah uz osoblje bez sigurnosnih dodataka.



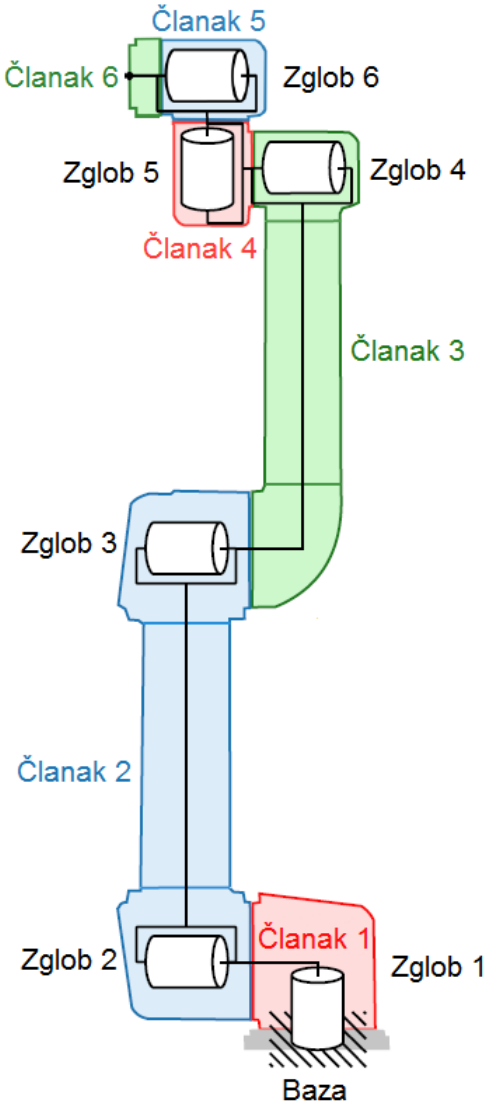
Slika 4.1: UR3, UR5 i UR10

4.1. UR5

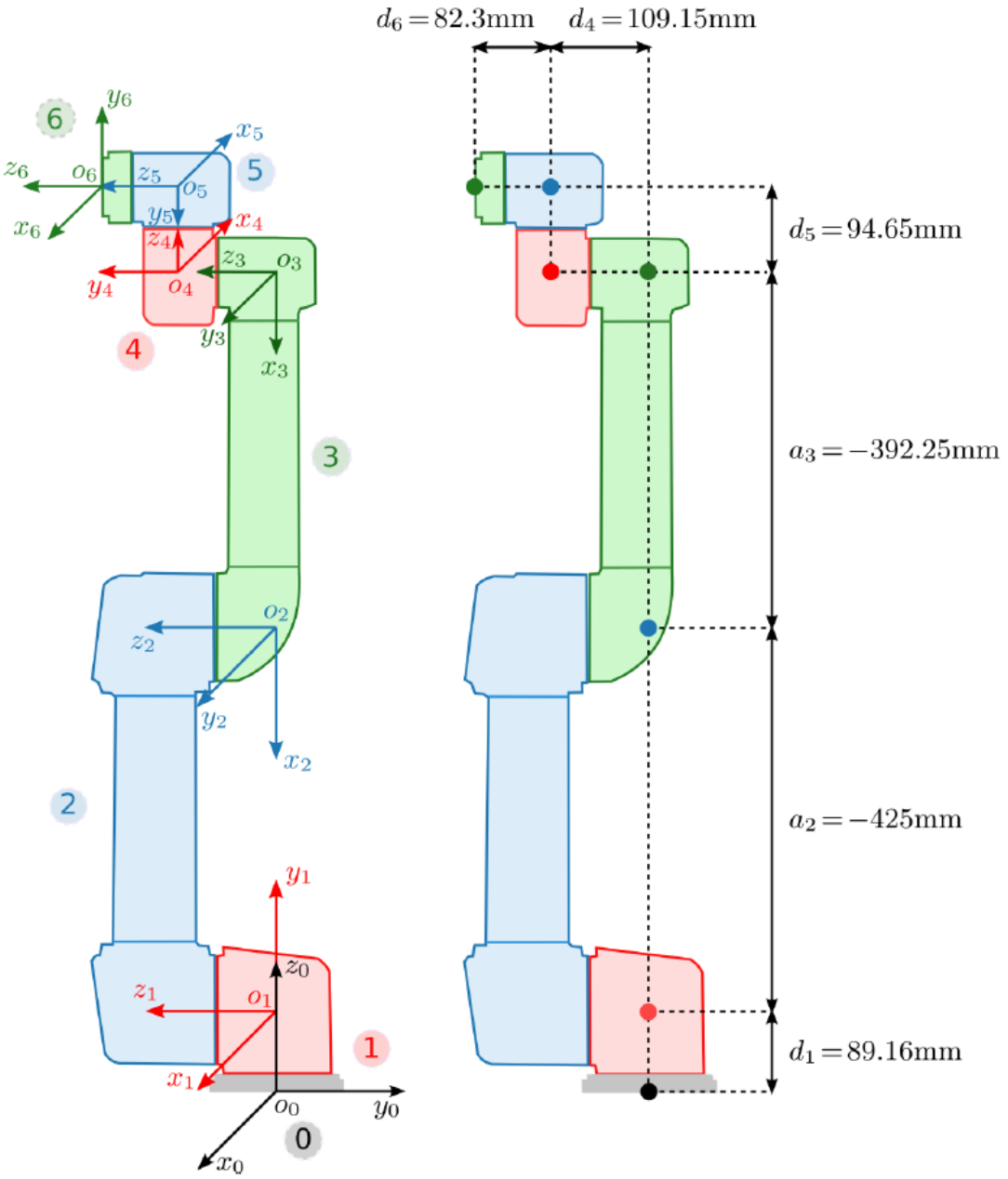
Robot UR5 korišten je u ovom diplomskom radu. Njegove karakteristike navedene su u tablici 4.1. Slika 4.2 prikazuje sve zglobove i segmente navedenog robota, a slika 4.3 DH parametre (lokalni koordinatni sustavi i udaljenosti između zglobova).

Tablica 4.1: UR5 Tehničke specifikacije

Težina:	18,4kg
Nosivost:	5kg
Doseg:	850mm
Radni opseg zglobova:	$\pm 360°$
Brzina svih zglobova:	$180°/s$
Brzina alata:	1m/s
Ponovljivost:	$\pm 0,1$mm
Broj stupnjeva slobode gibanja:	6
Veličina upravljačke jedinice:	475mm × 423mm × 268mm
Komunikacija:	TCP/IP 100 Mbit & Modbus TCP
Programiranje:	Grafičko korisničko sučelje PolyScope
Potrošnja energije:	$\approx$ 200W
Materijali:	Aluminij i Polipropilen (PP)
Temperaturno radno područje:	0°C do 50°C
Napajanje:	100 – 240 VAC, 50 – 60Hz



Slika 4.2: Zglobovi i segmenti robota UR5 [13]



Slika 4.3: DH parametri robota UR5 [13]

4.1.1. DH Parametri

Parametri robota UR5 izvedeni su prema Denavit–Hartenbergovoj konvenciji objašnjenoj u poglavlju 2.7.1. Oni su navedeni u tablici 4.2.

Tablica 4.2: DH parametri i njihove vrijednosti za robot UR5

Članak i	d_i [mm]	a_i [mm]	α_i [rad]
1	89.16	0	$\frac{\pi}{2}$
2	0	-425	0
3	0	-392.25	0
4	109.15	0	$\frac{\pi}{2}$
5	94.65	0	$-\frac{\pi}{2}$
6	82.3	0	0

U nastavku su navedene homogene matrice transformacije svih zglobova i matrica transformacije iz baze u vrh robota.

$$\mathbf{T}_1 = \begin{bmatrix} \mathbf{R}_1^0 & \mathbf{t}_1^0 \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_1 & 0 & \sin\theta_1 & 0 \\ \sin\theta_1 & 0 & -\cos\theta_1 & 0 \\ 0 & 1 & 0 & 0.08916 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

$$\mathbf{T}_2 = \begin{bmatrix} \mathbf{R}_2^1 & \mathbf{t}_2^1 \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & -0.425\cos\theta_2 \\ \sin\theta_2 & \cos\theta_2 & 0 & -0.425\sin\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

$$\mathbf{T}_3 = \begin{bmatrix} \mathbf{R}_3^2 & \mathbf{t}_3^2 \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & -0.392\cos\theta_3 \\ \sin\theta_3 & \cos\theta_3 & 0 & -0.392\sin\theta_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.3}$$

$$\mathbf{T}_4 = \begin{bmatrix} \mathbf{R}_4^3 & \mathbf{t}_4^3 \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_4 & 0 & \sin\theta_4 & 0 \\ \sin\theta_4 & 0 & -\cos\theta_4 & 0 \\ 0 & 1 & 0 & 0.1092 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

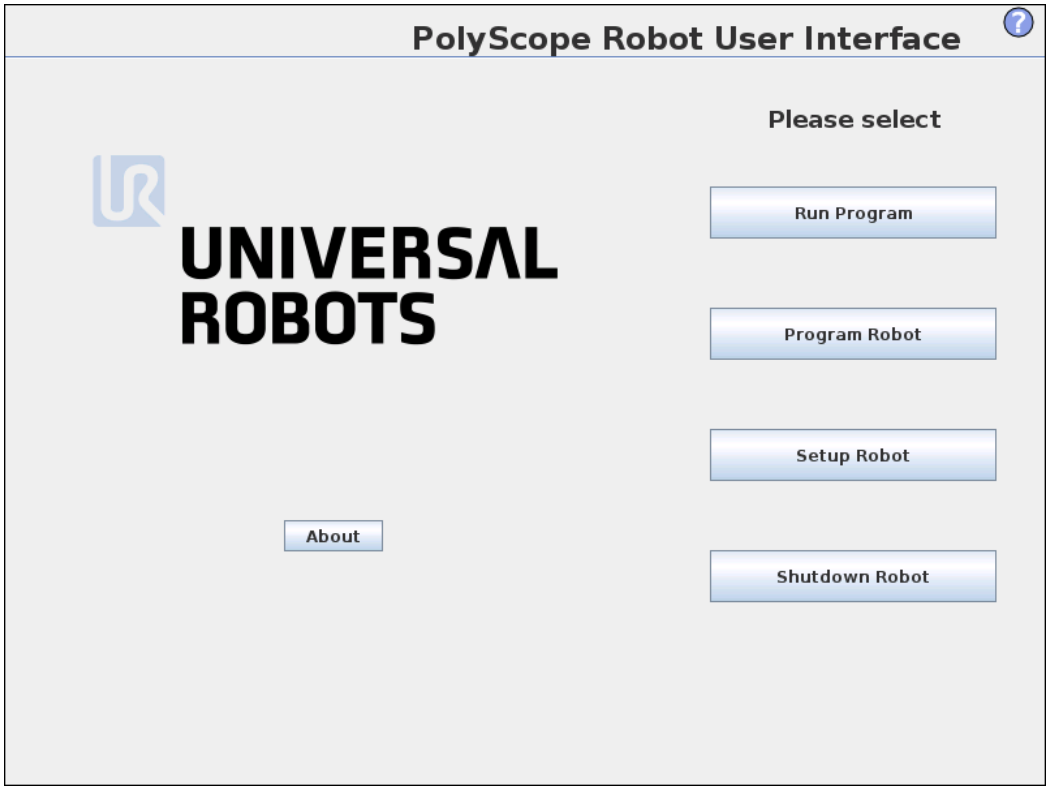
$$\mathbf{T}_5 = \begin{bmatrix} \mathbf{R}_5^4 & \mathbf{t}_5^4 \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_5 & 0 & -\sin\theta_5 & 0 \\ \sin\theta_5 & 0 & \cos\theta_5 & 0 \\ 0 & -1 & 0 & 0.0947 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.5}$$

$$\mathbf{T}_6 = \begin{bmatrix} \mathbf{R}_6^5 & \mathbf{t}_6^5 \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} \cos\theta_6 & -\sin\theta_6 & 0 & 0 \\ \sin\theta_6 & \cos\theta_6 & 0 & 0 \\ 0 & 0 & 1 & 0.0823 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.6}$$

$${}_6^0\mathbf{T} = \mathbf{T_1} \cdot \mathbf{T_2} \cdot \mathbf{T_3} \cdot \mathbf{T_4} \cdot \mathbf{T_5} \cdot \mathbf{T_6} = \begin{bmatrix} \mathbf{R}_6^0 & \mathbf{t}_6^0 \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} t_{11} & t_{12} & t_{13} & t_{14} \\ t_{21} & t_{22} & t_{23} & t_{24} \\ t_{31} & t_{32} & t_{33} & t_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.7}$$

4.2. Programiranje robota

UR robote je moguće programirati putem privjeska za učenje, skripte ili putem C–API sučelja. Privjesak za učenje se sastoji od 12–inčnog ekrana osjetljivog na dodir, sigurnosne tipke i jednog USB porta u koji je moguće uštekati tipkovnicu, miš ili USB stick. Korisničko sučelje u kojem se robot programira naziva se PolyScope i prikazano je na slici 4.4. Sučelje je vrlo jednostavno i intuitivno. Za razliku od ostalih industrijskih robota gdje je potrebno proći obuku za rad s robotom, kod UR robota to nije slučaj. Korisnik ovdje može u malom vremenu savladati osnove i izraditi neke jednostavne programe. Programiranje robota preko skripte je zahtjevnije i potrebno je znanje iz programiranja i poznavanje robotovog programskog jezika URScript. U ovom radu robot je programiran preko skripte tako da se naredbe ili cijeli program na njega šalje putem TCP protokola.



Slika 4.4: PolyScope korisničko sučelje

4.3. Komunikacija

Komunikaciju s robotom moguće je uspostaviti preko TCP protokola koji je objašnjen u poglavlju 2.11.1. Kako bi komunikacija bila moguća, robot i računalo moraju biti u istoj podmreži (eng. *subnet*). Na robotu su pokrenuta četiri poslužitelja (servera) na sljedećim portovima:

1. 29999 – Server nadzorne ploče (eng. *Dashboard server*)
2. 30001 – Primarni server
3. 30002 – Sekundarni server
4. 30003 – Real time server

Svaki od njih detaljno je objašnjen u sljedećim odjeljcima.

4.3.1. Dashboard server – port 29999

Spajanjem na server nadzorne ploče moguće je upravljati raznim opcijama na robotu ali ga na taj način nije moguće programirati. U nastavku su navedene i opisane neke naredbe

koje je moguće slati preko tog porta:

- load<program.urp> – učitavanje programa
- play – pokretanje programa
- stop – zaustavljenje programa
- pause – pauziranje programa
- shutdown – isključivanje robota

Korištenje tih naredbi može biti vrlo korisno u industriji gdje je više robota spojeno u jednu mrežu. Ukoliko su na njima već napisani programi tada se može s robotima upravljati sa jednog mjesta.

4.3.2. Primarni i sekundarni server – port 30001 i 30002

Primarni i sekundarni serveri služe za programiranje i dobivanje informacija o stanju robota. Nakon spajanja na jedan od tih servera primaju se podaci brzinom od 10Hz. Primarni server šalje stanje robota i dodatne poruke, a sekundarni samo stanje robota. Robotom je moguće upravljati preko tih servera tako da mu se šalju naredbe u obliku skripte. To može biti jedna naredba ili cijeli program. Pojedinačne naredbe koje se pošalju robotu na te portove on izršava trenutno bez obzira je li u toku rada ili miruje. Ukoliko se robotu šalje cijeli program, on ga izvršava nakon što je primi zadnji red programa, tj. naredbu end. Bitno je napomenuti da svaka naredba mora završavati sa „\n" što ustvari predstavlja novi red, tj. enter.

4.3.3. Real time server – port 30003

Real time server svim spojenim klijentima šalje informacije o stanju robota brzinom od 125Hz. Na njega je također moguće slati naredbe kao i na primarni i sekundarni server, međutim on se uglavnom koristi za brzo primanje stanja robota. Veličina i redosljed podataka se međusobno razlikuju ovisno o verziji softvera na robotu. U tablici 4.3 je prikazan sadržaj poruke koju šalje robot sa instaliranim softverom UR3.2. Ukupna veličina poruke je 1060 bajta. Skoro sve numeričke vrijednosti robot šalje u double preciznosti prema IEEE 754 standardu.

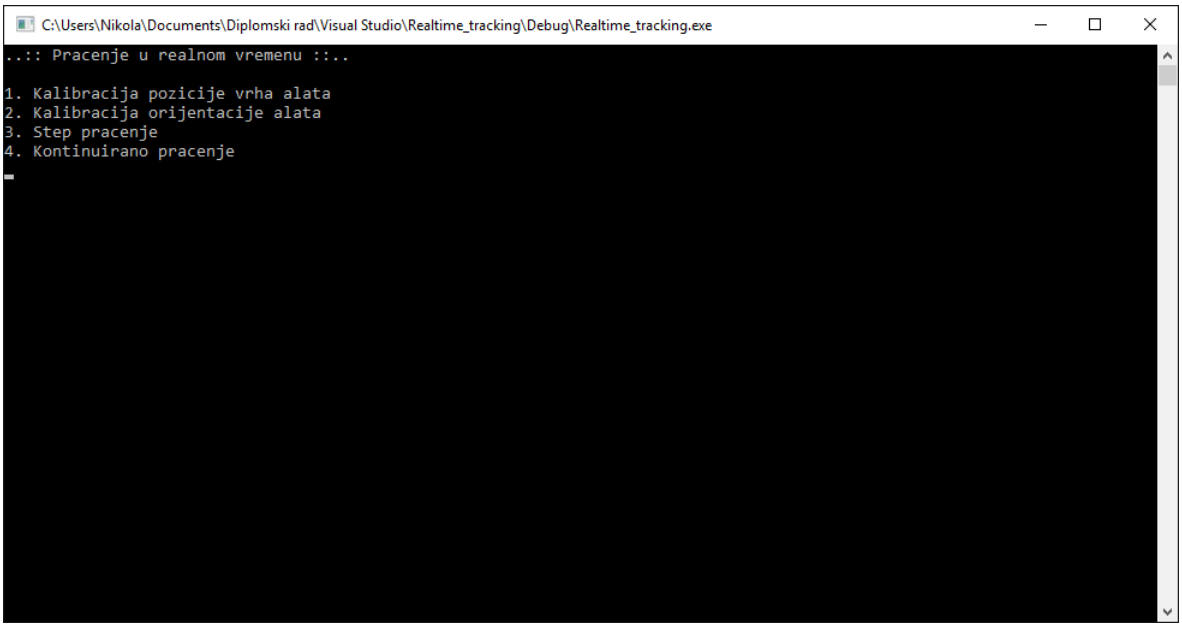
Tablica 4.3: Sadržaj poruke koju šalje Real time server

Naziv	Veličina [B]	Opis
Veličina poruke	4	Ukupna duljina poruke u bajtovima
Vrijeme	8	Proteklo vrijeme od početka rada kontrolera
$\mathbf{q_t}$	48	Ciljani položaj svakog zgloba
$\mathbf{\dot{q}_t}$	48	Ciljana brzina svakog zgloba
$\mathbf{\ddot{q}_t}$	48	Ciljana akceleracija svakog zgloba
$\mathbf{I_t}$	48	Ciljana struja u svakom zglobu
$\mathbf{M_t}$	48	Ciljani moment u svakom zglobu
$\mathbf{q_a}$	48	Trenutni položaj svakog zgloba
$\mathbf{\dot{q}_a}$	48	Trenutna brzina svakog zgloba
$\mathbf{I_a}$	48	Trenutna struja u svakom zglobu
$\mathbf{I_c}$	48	Napon kontrolera svakog zgloba
$\mathbf{TCP_a}$	48	Trenutne koordinate alata u kartezijevom k.s.
$\mathbf{T\dot{C}P_a}$	48	Trenutne brzine alata u kartezijevim koordinatama
$\mathbf{F}$	48	Sile u alatu
$\mathbf{TCP_t}$	48	Ciljane koordinate alata u kartezijevom k.s.
$\mathbf{T\dot{C}P_t}$	48	Ciljane brzine alata u kartezijevim koordinatama
Digitalni ulazi	8	Trenutno stanje digitalnih ulaza u bitovima
$\boldsymbol{\vartheta}$	48	Temperature svih zglobova u °C
Tajmer kontrolera	8	Vrijeme potrebno za izvršenje thread-a
Ispitna vrijednost	8	Vrijednost koju koristi samo UR softver
Mod robota	8	Način rada robota
Modovi zglobova	48	Način rada zglobova
Sigurnosni mod	8	Sigurnosni način rada
-	48	Vrijednost koju koristi samo UR softver
$\mathbf{T\ddot{C}P}$	24	x, y i z vrijednosti ubrzanja alata
-	48	Vrijednost koju koristi samo UR softver
Brzina skaliranja	8	Brzina skaliranja trajektorije
$\mathbf{p}$	8	Količina gibanja
-	8	Vrijednost koju koristi samo UR softver

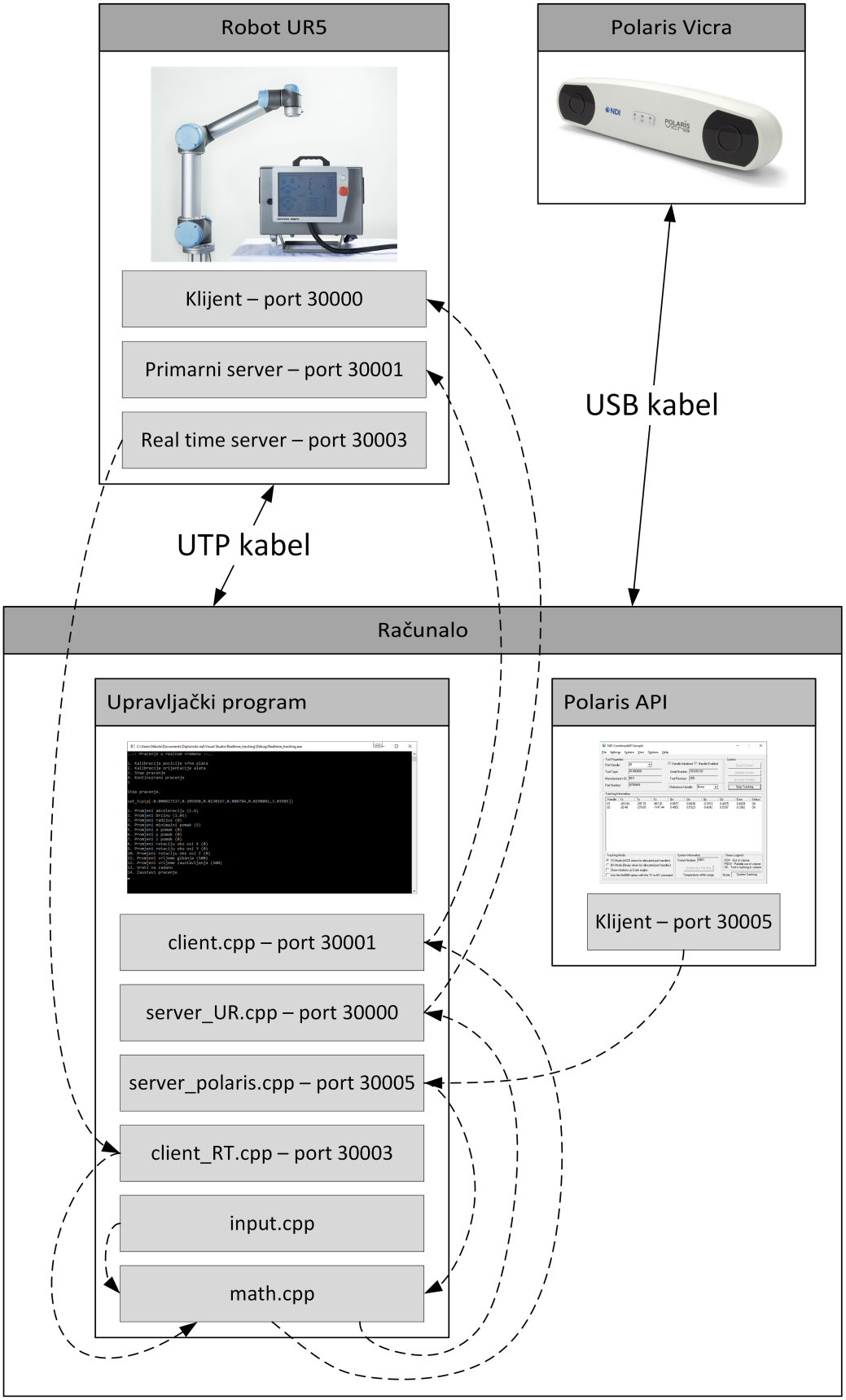
-	8	Vrijednost koju koristi samo UR softver
$\mathbf{V_m}$	8	Glavni napon
$\mathbf{V_r}$	8	Napon robota
$\mathbf{I_r}$	8	Struja robota
$\mathbf{V_a}$	48	Trenutni naponi u zglobovima
Digitalni izlazi	8	Trenutno stanje digitalnih izlaza u bitovima
Stanje programa	8	Trenutno stanje programa
Ukupno	**1060**	

5. UPRAVLJAČKI PROGRAM ZA PRAĆENJE

Upravljački program izrađen je u programskom paketu Visual Studio 2015. Napisan je u programskom jeziku C++. Kompletan kôd priložen je u prilogu A. U programu je korištena knjižnica za linearnu algebru Eigen koja bitno pojednostavljuje rad s matricama, vektorima i transformacijama. Eigen je besplatan i moguće ga je skinuti sa službene web stranice. Njegov direktorij potrebno je smjestiti negdje na računalu, a u programskom paketu Visual Studio locirati taj direktorij pod C++ postavkama projekta. Upravljački program sastoji se od glavnog programa i šest threadova koji se izvršavaju paralelno. Svaki od tih programa detaljno je objašnjen u nastavku. Početno sučelje upravljačkog programa prikazano je na slici 5.1, a shema cijelog sustava na slici 5.2.



Slika 5.1: Početno sučelje upravljačkog programa



Slika 5.2: Shema sustava

5.1. Polaris API dodatak

Unutar programa Polaris API, opisanog u poglavlju 3.3, dodan je programski kôd za TCP klijent. Pritiskom na „*Start tracking*" klijent se spaja na poslužitelj (server) koji je pokrenut u upravljačkom programu na portu 30005 i šalje mu dvije poruke u tekstualnom obliku. Prva poruka šalje položaj i orijentaciju od alata M1, a druga od alata M2. Njihov oblik je:

- M1(x, y, z, q_w, q_x, q_y, q_z) ili M1_(OOV)
- M2(x, y, z, q_w, q_x, q_y, q_z) ili M2_(OOV)

gdje x, y i z predstavljaju položaj ishodišta svakog alata u koordinatnom sustavu kamere, a q_w, q_x, q_y i q_z njegovu orijentaciju u kvaternionima. Ukoliko je alat izvan mjernog područja sustava tada se šalje poruka M1_(OOV), odnosno M2_(OOV). Brzina kojom se šalju poruke iznosi 20Hz, kolko iznosi i maksimalna brzina ažuriranja sustava Polaris Vicra. Kada se pritisne „*Stop tracking*" tada i klijent prestaje raditi, tj. odspaja se od poslužitelja.

5.2. Glavni program (main.cpp)

Main.cpp je glavni program, on pokreće i drži pokrenutim svih šest programa. Svaki od programa deklariran je naredbom „void naziv_programa(void *P);" Njegov kôd priložen je u prilogu A.1.

5.3. Komunikacija sa Polarisom (server_polaris.cpp)

Ovaj program pokreće TCP poslužitelj (server) na portu 30005. Na njega se spaja Polaris API i šalje mu prije navedene poruke. U dijelu gdje se primaju poruke dodana je uvjetna (if) petlja koja razlikuje poruke ovisno o njihovom početku („M1" ili „M2"). Svaka od tih poruka zatim se obrađuje na način da se prvo brišu prva tri („M1(" ili „M2(") i zadnji znak („)"), a preostale vrijednosti spremaju se u vektor „Receive_Data_M1", odnosno „Receive_Data_M2" koji se sastoji od sedam članova . Iz tih vektora zatim se deklariraju varijable M1_x, M1_y, M1_z, M1_qw, M1_qx, M1_qy i M1_qz, odnosno M2_x,

M2_y, M2_z, M2_qw, M2_qx, M2_qy i M2_qz. Sve varijable su globalne i kasnije se koriste u programu math.cpp. Cijeli taj postupak potreban je kako bi se iz tekstualne poruke izvukle numeričke vrijednosti. Ukoliko je alat izvan mjernog područja sustava tada klijent šalje M1_(OOV), odnosno M2_(OOV), a server deklarira booleon operator M1_OOV, odnosno M2_OOV kao istinit.

5.4. Primanje podataka sa robota (client_RT.cpp)

Ovaj TCP klijent spaja se na robotov Real time server koji je pokrenut na portu 30003 i opisan u poglavlju 4.3.3. Sa tog servera poruke pristižu brzinom od 125Hz. Sadržaj poruke opisan je u tablici 4.3. Ovaj thread iz cijele poruke izvlači samo trenutne koordinate alata u baznom koordinatnom sustavu robota. Veličina cijele poruke je 1060 bajta, a trenutne koordinate zapisane su od 444. do 492. bajta. Svaka numerička vrijednost zapisana je u double preciznosti prema IEEE 754 standardu. Veličina svake vrijednosti iznosi 8 bajta. Poruka se prima u obliku polja znakova gdje je svaki znak zapisan kao jedan bajt. Iz tog polja izvlači se po 8 znakova od člana 444. do 492. Svaki znak zatim se pretvara u binarni oblik koji se sastoji od osam nula ili jedinica (8 bitova). Binarni zapis od 8 znakova zatim se spaja u string koji se sastoji od 64 nula ili jedinica (64 bita). Ovaj 64 bitni zapis je ustvari numerička vrijednost zapisana u double preciznosti prema IEEE 754 standardu. Naredbom „bitstring_to_double()" dobiva se numerička vrijednost. Isti taj postupak ponovljen je za svih šest članova koji prikazuju poziciju i orijentaciju alata. Ti članovi deklarirani su kao globalne varijable X_in, Y_in, Z_in, Rx_in, Ry_in i Rz_in. X_in, Y_in i Z_in predstavljaju poziciju vrha alata u baznom koordinatnom sustavu robota, a Rx_in, Ry_in i Rz_in orijentaciju alata u Axis–Angle zapisu opisanom u poglavlju 2.4.

5.5. Matematika vezana za praćenje (math.cpp)

U ovom programu izvodi se sva matematika vezana za praćenje i neke dodatne stvari. Program je podijeljen na četiri razine:

1. Kalibracija pozicije vrha alata
2. Kalibracija orijentacije alata

3. Step praćenje
4. Kontinuirano praćenje

Na početnom sučelju upravljačkog programa moguće je odabrati željenu razinu za ivođenje, a svaka od njih detaljno je opisana u sljedećim odjeljcima.

5.5.1. 1. Kalibracija pozicije vrha alata

Kalibracija pozicije vrha alata služi za precizno određivanje koordinata vrha alata u koordinatnom sustavu prihvatnice robota. Vrh alata robota korisnik mora ručno dovesti u istu točku u prostoru sa 4 (minimalno moraju biti tri) različite konfiguracije robota. Nakon što korisnik dovede vrh robota u željenu točku potrebno je pritisnuti Enter kako bi se spremile vrijednosti. Za svaku konfiguraciju sprema se pozicija i orijentacija vrha alata u njoj. Pozicija se sprema u trodimenzionalni vektor t_Pi, a orijentacija u matricu rotacije R_Pi ($i = 1 \dots 4$). Koordinate vrha alata izračunavaju se prema izrazu 2.38, u ovom slučaju je:

$$\mathbf{A} = \begin{bmatrix} \mathbf{R_2} - \mathbf{R_1} \\ \mathbf{R_3} - \mathbf{R_2} \\ \mathbf{R_4} - \mathbf{R_3} \end{bmatrix} \tag{5.1}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{t_1} - \mathbf{t_2} \\ \mathbf{t_2} - \mathbf{t_3} \\ \mathbf{t_3} - \mathbf{t_4} \end{bmatrix} \tag{5.2}$$

Nakon izračuna X, Y i Z koordinate vrha alata zapisane su u trodimenzionalnom vektoru $\mathbf{X_t}$. Te vrijednosti također se zapisuju u datoteku „TCP_Position.txt”. Ukoliko postoji, ta datoteka se učitava na početku programa te je ovu kalibraciju potrebno obaviti samo jednom.

5.5.2. 2. Kalibracija orijentacije alata

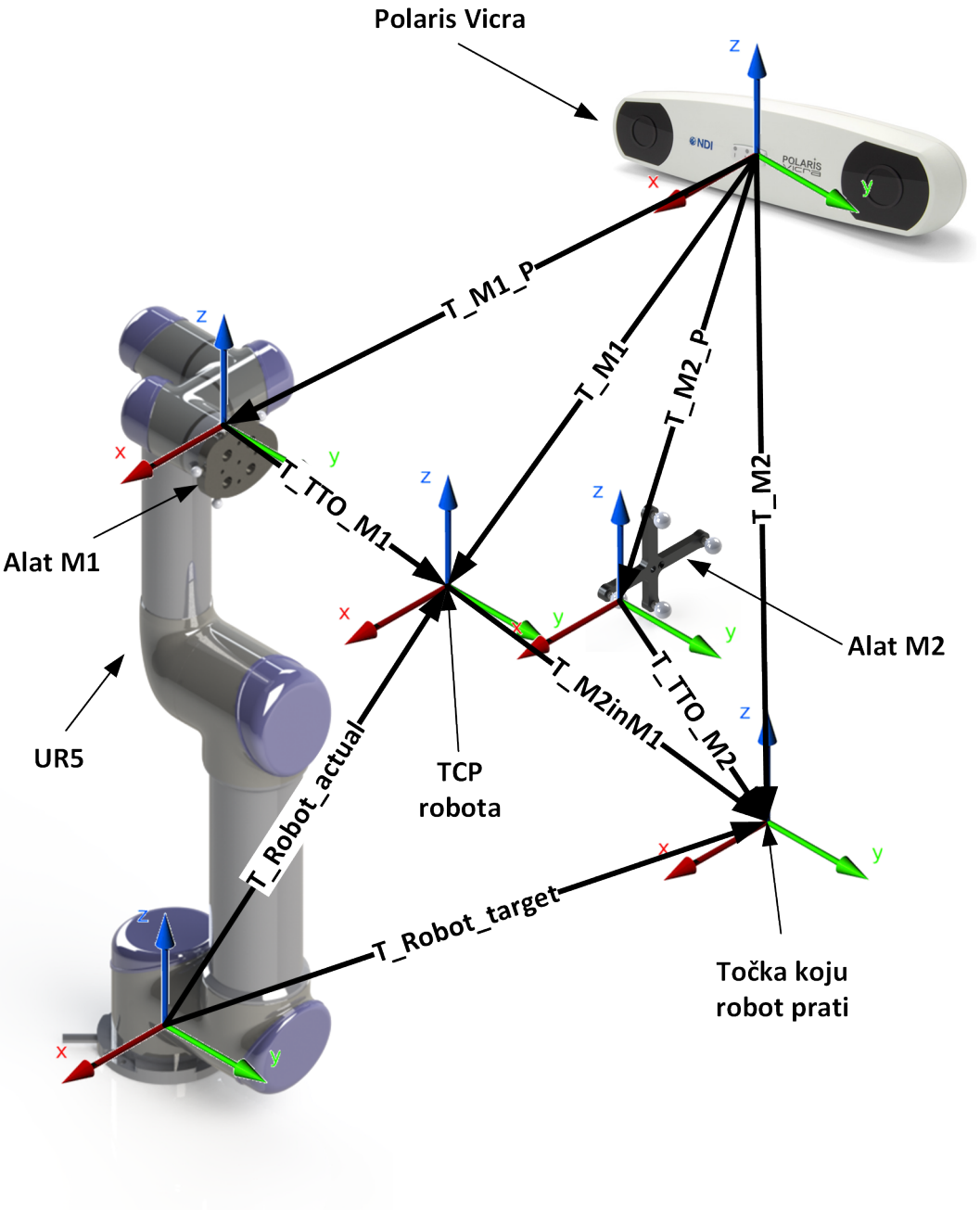
Ovom kalibracijom dobiva se da stereovizijski sustav i robot orijentaciju alata vide jednako. Bez toga bi praćenje bilo nemoguće. Postupak je potpuno automatiziran. Jedini uvjet je da je obavljena kalibracija pozicije vrha alata. Robotu se šalju naredbe za pomak u tri unaprijed određene točke (T_0, T_1 i T_2) u prostoru. Pozicija svake točke sprema se

u koordinatom sustavu robota u trodimenzionalni vektor Ti_R i koordinatnom sustavu Polarisa u trodimenzionalni vektor Ti_P, ($i = 0 \dots 2$). Točka T_0 određuje ishodište koordinatnog sustava, točke T_0 i T_1 smjer X–osi, a točke T_0 i T_2 smjer lažne Y–osi. Vektor koji određuje X-os dobiva se oduzimanjem normaliziranih vektora $\mathbf{T_1}$ i $\mathbf{T_0}$: $\mathbf{X} = |\mathbf{T_1} - \mathbf{T_0}|$. Vektor koji određuje lažnu Y-os dobiva se oduzimanjem normaliziranih vektora $\mathbf{T_2}$ i $\mathbf{T_0}$: $\mathbf{Y_l} = |\mathbf{T_2} - \mathbf{T_0}|$. Vektor koji određuje Z-os dobiva se vektorskim produktom normaliziranih vektora $\mathbf{Y_l}$ i $\mathbf{X}$: $\mathbf{Z} = |\mathbf{Y_l} \times \mathbf{X}|$. I na kraju, vektor koji određuje Y-os dobiva se vektorskim produktom normaliziranih vektora $\mathbf{X}$ i $\mathbf{Z}$: $\mathbf{Y} = |\mathbf{X} \times \mathbf{Z}|$. Vektori $\mathbf{X}$, $\mathbf{Y}$ i $\mathbf{Z}$ određuju koordinatni sustav u prostoru i pomoću njih se određuje matrica rotacije $\mathbf{R}$ kao što je prikazano u izrazu 2.2. Pozicija i orijentacija koordinatnog sustava definirana je matricom homogene transformacije $\mathbf{T}$ koja je objašnjena u poglavlju 2.5 gdje je vektor $\mathbf{t}$ pozicija točke T_0 zapisana kao vektor $\mathbf{T_0}$. Ovim postupkom definira se isti koordinatni sustav u prostoru, ali iz dva različita ishodišta. Jedno je u bazi robota, a drugo u bazi stereovizijske kamere. Sljedeći korak je kreiranje matrice homogene transformacije za točku u kojoj je trenutno vrh alata robota u prostoru. Jedna matrica definira se u robotovom, a druga u Polarisovom baznom koordinatnom sustavu. Zatim se prema izrazu 2.33 izračuna matrica homogene transformacije prije spomenute točke u koordinatnim sustavima robota i Polarisa koji su prethodno definirani. Zadnji korak je množenje tih dviju matrica i time se dobiva matrica rotacije $\mathbf{R_{RP}}$ koja je veza između robota i Polarisa. Iz te matrice izvuku se Axis–Angle vrijednosti prema formulama 2.28, 2.29, 2.30 i 2.31. Te vrijednosti rotacija zajedno sa vrijednostima pozicije vrha alata definiraju TCP (eng. *Tool Center Point*) robota. Naredba kojom se definira TCP na robotu glasi „set_tcp(p[X_TCP, Y_TCP, Z_TCP, Rx_TCP, Ry_TCP, Rz_TCP])" i njome je definirana pozicija i orijentacija vrha alata M1 na robotu. Navedene vrijednosti zapisuju se i u datoteku „TCP_Orientation.txt" kako bi prilikom sljedećeg pokretanja programa bilo moguće preskočiti ovaj korak i odmah započeti praćenje.

5.5.3. 3. Step praćenje

Step praćenje je praćenje u kojem se robot jedan interval giba, a jedan zaustavlja. Trajanje svakog intervala moguće je mijenjati u toku praćenja. Bez zaustavljanja robot jako trza te je ovo praćenje morala biti izvedeno sa zaustavljanjem. Trzaji se događaju iz razloga što

se robotu šalje nova točka u koju se mora gibati prije nego što je došao u prethodnu. Prije početka praćenja program šalje robotu položaj i orijentaciju vrha alata – TCP (eng. *Tool Center Point*) koji je pomaknut iz ishodišta alata M1 za pomak koji je određen matricom transformacije T_TTO_M1. Sve matrice transformacija potrebne za praćenje prikazane su na slici 5.3.



Slika 5.3: Matrice transformacija potrebne za praćenje

U nastavku slijedi objašnjenje svake od njih:

- $\mathbf{T_{M1_P}}$ – pozicija i orijentacija alata M1 u baznom koordinatnom sustavu Polarisa

- $\mathbf{T_{TTO_{M1}}}$ – pomak iz ishodišta alata M1 u vrh robota (TCP)
- $\mathbf{T_{M1}}$ – pozicija i orijentacija vrha robota u baznom koordinatnom sustavu Polarisa

$$\mathbf{T_{M1}} = \mathbf{T_{M1_P}} \cdot \mathbf{T_{TTO_{M1}}} \tag{5.3}$$

- $\mathbf{T_{M2_P}}$ – pozicija i orijentacija alata M2 u baznom koordinatnom sustavu Polarisa
- $\mathbf{T_{TTO_{M2}}}$ – pomak iz ishodišta alata M2 u točku za praćenje
- $\mathbf{T_{M2}}$ – pozicija i orijentacija točke za praćenje u baznom koordinatnom sustavu Polarisa

$$\mathbf{T_{M2}} = \mathbf{T_{M2_P}} \cdot \mathbf{T_{TTO_{M2}}} \tag{5.4}$$

- $\mathbf{T_{Robot_{actual}}}$ – pozicija i orijentacija vrha robota (TCP) u baznom koordinatnom sustavu robota
- $\mathbf{T_{M2inM1}}$ – pozicija i orijentacija točke za praćenje u koordinatnom sustavu vrha robota

$$\mathbf{T_{M2inM1}} = \mathbf{T_{M1}}^{-1} \cdot \mathbf{T_{M2}} \tag{5.5}$$

- $\mathbf{T_{Robot_{target}}}$ – pozicija i orijentacija točke za praćenje u baznom koordinatnom sustavu robota

$$\mathbf{T_{Robot_{target}}} = \mathbf{T_{Robot_{actual}}} \cdot \mathbf{T_{M2inM1}} \tag{5.6}$$

Svi izračuni vezani za praćenje izvršavaju se unutar while petlje. Ciljni položaj robota definiran je matricom transformacije $\mathbf{T_{Robot_{target}}}$. Iz te matrice izvlače se vrijednosti pozicije i orijentacije vrha robota u točki koja se prati. Zatim se robotu šalje naredba za gibanje u tu točku. Unutar while petlje računa se još i euklidska udaljenost između točke u kojoj je vrh robota i točke za praćenje. Ukoliko je ta udaljenost manja od željene robot se ne giba. Također, robot se ne giba ni u slučajevima kada je jedan od alata van mjernog volumena stereovizijske kamere. Brzina kojom upravljački program računa ciljni položaj robota jednaka je brzini ažuriranja sustava Polaris Vicra i ona iznosi 20Hz.

5.5.4. 4. Kontinuirano praćenje

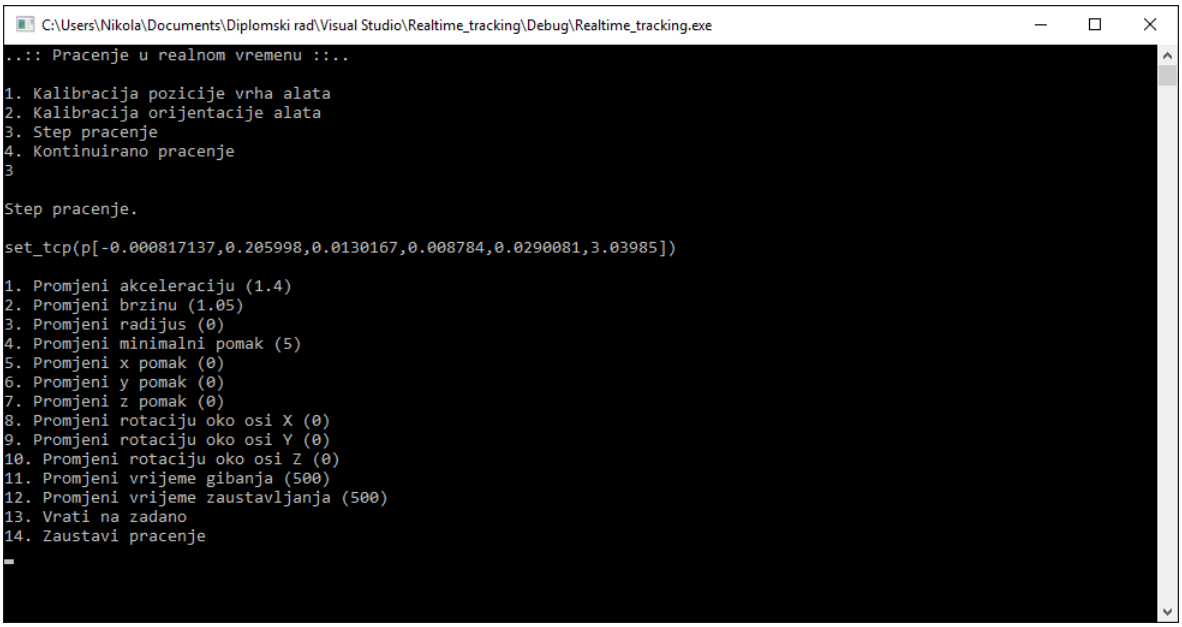
Za razliku od step praćenja, ovdje se praćenje izvodi bez periodičkog zaustavljanja. Razlika je u tome što se ovdje robot kao klijent spaja na server unutar upravljačkog programa. Taj server robotu šalje samo numeričke vrijednosti pozicije, orijentacije, akceleracije i brzine, a ne cijelu naredbu za gibanje. Naredba za gibanje izvršava se unutar programa koji je pokrenut na robotu. Na taj način su spriječeni trzaji jer robot uvijek naredbu gibanja izvrši do kraja, tj. tek kad dođe u točku kreće u sljedeću. Iz razloga što ovdje nema periodičkog zaustavljanja ovo praćenje je brže, ali nestabilnije. Nestabilnost uzrokuje primijenjeni robot UR5 i njegovi algoritmi sigurnosti.

5.6. Parametri praćenja (input.cpp)

Ovaj program služi za upravljanje parametrima praćenja. U nastavku su navedene i opisane varijable koje je moguće mijenjati:

- a – ubrzanje zglobova robota
- v – brzina zglobova robota
- r – radijus zaobilaženje točke
- MinDist – minimalni pomak alata M2 za koji se robot neće micati
- x_offset – pomak alata M1 u smjeru osi X
- y_offset – pomak alata M1 u smjeru osi Y
- z_offset – pomak alata M1 u smjeru osi Z
- roll_deg – rotacija oko osi X u °
- pitch_deg – rotacija oko osi Y u °
- yaw_deg – rotacija oko osi Z u °
- sleep1 – vrijeme gibanja robota u step praćenju
- sleep2 – vrijeme zaustavljanja robota u step praćenju

Ovdje je također moguće vratiti varijable na početne vrijednosti i zaustaviti praćenje. Zaustavljanje se odvija tako da se globalni booleon operator *tracking* postavi kao neistinit (false) što automatski prekida while petlju unutar programa math.cpp. Izgled sučelja za unos parametara prikazan je na slici 5.4.



Slika 5.4: Sučelje za unos parametara praćenja

5.7. Slanje naredbi robotu (client.cpp)

Client.cpp je TCP klijent koji se spaja na robotov primarni server na portu 30001. On mu šalje pojedinačne naredbe ili u slučaju kontinuiranog praćenja cijeli program. Naredbe i program napisani su u programskom jeziku robota koji se naziva URScript. Naredbe se šalju kada je booleon operator *sending* postavljen kao istinit. U slučaju step praćenja, tj. kada je booleon operator *step_motion* postavljen kao istinit, šalje se i naredba „stopj(a)" gdje je a iznos deceleracije zglobova. Nakon svake naredbe postavljen je tajmer čije je vrijeme deklarirano varijablom sleep1, odnosno sleep2.

5.8. Slanje tekstualne poruke robotu (server_UR.cpp)

Ovaj poslužitelj (server) koristi se samo ukoliko korisnik odabere „4. Kontinuirano praćenje". U tom slučaju je na robotu program koji se kao klijent spaja na taj poslužitelj. Komunikacija se odvija tako da poslužitelj robotu (klijentu) šalje poruku oblika „(0/1, X,

Y, Z, Rx, Ry, Rz, a, v, r)". Kada je prvi znak 0 robot miruje, a kada je 1 se giba. X, Y i Z su koordinate točke u baznom koordinatnom sustavu robota u koju mora doći vrh alata robota, Rx, Ry i Rz je orijentacija alata u toj točki u Axis–angle zapisu, a je iznos akceleracije zglobova, v je iznos ubrzanja zglobova, a r je radijus zaobilaženja točke.

6. ZAKLJUČAK

U okviru diplomskog zadatka razvijen je upravljački program koji omogućuje relativno vođenje alata robotske ruke u realnom vremenu, koristeći informacije stereovizijskog sustava. Upravljački program napisan je u C++ programskom jeziku i sastoji se od šest programa koji se izvršavaju istovremeno. Cijeli sustav upravljanja sastoji se od robota UR5, stereovizijskog sustava Polaris Vicra i računala. Stabilna i brza komunikacija između članova sustava odvija se putem TCP protokola. Samo praćenje izvodi se na način da upravljački program kontinuirano prikuplja informacije stereovizijskog sustava i robota, na temelju tih podataka izračunava matricu transformacije koja definira ciljni položaj i orijentaciju vrha robota u prostoru, te na kraju izračunati položaj i orijentaciju šalje robotu koji tada započinje s gibanjem. Brzina kojom upravljački program računa ciljni položaj vrha robota iznosi 20Hz, a definirana je brzinom ažuriranja stereovizijskog sustava Polaris Vicra. Stereovizijski sustav osigurava i točnost pozicioniranja u ciljnom položaju i ona je unutar jednog milimetra. Najsporiji član u sustavu je, očekivano, robot. On bi se mogao ubrzati integriranjem regulacije za upravljanje brzinom i akceleracijom gibanja. Međutim, kod primijenjenog robota UR5 to nije bilo moguće iz razloga što samo jedna izvršna naredba može upravljati gibanjem, brzinom i akceleracijom te nije moguće ubrzavati ili usporavati gibanje u toku. Razvijeni upravljački program moguće je primijeniti na bilo kojem industrijskom robotu uz prilagodbu samih naredbi koje se šalju robotu. Primjena razvijenog sustava, između ostalog, moguća je i u medicini i to u području transkranijalne magnetske stimulacije (TMS) gdje je bitna preciznost i kontinuirana komparativna korekcija položaja magneta u odnosu na glavu pacijenta. Buduće istraživanje ovog područja može se posvetiti upravo primjeni u TMS-u uz implementaciju regulacije i senzora sile te sprječavanje kolizije sa praćenim objektom.

LITERATURA

[1] Asimov, I.: Runaround, Street & Smith, United States, 1942.

[2] World Robotics – Industrial Robots 2015 – Executive Summary, http://www.worldrobotics.org/uploads/media/Executive_Summary__WR_2015.pdf

[3] Golub, A. A., Tkacheva, N. A.: ROBOTS IN MEDICINE, Siberian Federal University, 2011.

[4] Gerhardus, D.: Robot-assisted surgery: the future is here, Journal of Healthcare Management, 48(4), 242, 2003.

[5] Rogić, M.: Prilog metodologiji prepoznavanja neurofizioloških biljega u mišićima grkljana izazvanih transkranijalnom magnetskom stimulacijom (TMS) motoričkih područja kore mozga za govor, Sveučilište u Splitu, Medicinski fakultet, Split, 2012.

[6] Rješavanje direktnog i inverznog kinematičkog problema pomoću dvojnih kvaterniona, https://www.fer.unizg.hr/_download/repository/Kvaternioni.pdf

[7] Doleček, V., Karabegović, I.: Robotika, Tehnički fakultet Bihać, Bihać, 2002.

[8] Yin, S., Ren, Y., Zhu, J., Yang, S., Ye, S.: A Vision-Based Self-Calibration Method for Robotic Visual Inspection Systems, Sensors, stranice 16571–16572, 2013.

[9] Benić, Z., Šimunić, N.: Stereovizijski sustav podržan OpenCV bibliotekom, Zbornik Veleučilišta u Karlovcu, 2013.

[10] TCP, https://hr.wikipedia.org/wiki/TCP

[11] TCP protokol, http://mreze.layer-x.com/s040100-0.html

[12] Wiles, A. D., Thompson, D. G., Frantz, D. D.: Accuracy assessment and interpretation for optical tracking systems, in Medical Imaging 2004, stranice 421–432, International Society for Optics and Photonics, 2004

[13] Kufieta, K.: Force Estimation in Robotic Manipulators: Modeling, Simulation and Experiments, NTNU Norwegian University of Science and Technology, 2014.

PRILOZI

A. Programski kôd

B. Tehnička dokumentacija

C. CD-R disc

A. Programski kôd

A.1. main.cpp

```
#include <stdio.h>
#include <windows.h>
#include <process.h>

using namespace std;
void server_polaris(void *P);
void server_UR(void *P);
void client(void *P);
void client_RT(void *P);
void math(void *P);
void input(void *P);

int main() {
	printf("..:: Pracenje u realnom vremenu ::..\n\n");

	HANDLE hThreads[6];
	hThreads[0] = (HANDLE)_beginthread(server_polaris, 0, (void*)0);
	hThreads[1] = (HANDLE)_beginthread(server_UR, 0, (void*)0);
	hThreads[2] = (HANDLE)_beginthread(client, 0, (void*)0);
	hThreads[3] = (HANDLE)_beginthread(client_RT, 0, (void*)0);
	hThreads[4] = (HANDLE)_beginthread(math, 0, (void*)0);
	hThreads[5] = (HANDLE)_beginthread(input, 0, (void*)0);

	WaitForMultipleObjects(6, hThreads, TRUE, INFINITE);
}
```

A.2. server_polaris.cpp

```
#define WIN32_LEAN_AND_MEAN

#include <WinSock2.h>
#include <WS2tcpip.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <string>
#include <sstream>

// link with Ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

#define DEFAULT_PORT "30005"
#define DEFAULT_BUFFER_LENGTH     512

using namespace std;

double M1_x, M1_y, M1_z, M1_qw, M1_qx, M1_qy, M1_qz;
double M2_x, M2_y, M2_z, M2_qw, M2_qx, M2_qy, M2_qz;
double Receive_Data_M1[7];
double Receive_Data_M2[7];
int i;
string msgstr;
string delimiter = ",";
size_t pos;
string number;
bool M1_OOV, M2_OOV;

void server_polaris(void *P) {
	WSADATA wsaData;

	// Initialize Winsock
	int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
	if (iResult != 0) {
		printf("server_polaris: WSAStartup failed: %d\n", iResult);
		return;
	}

	struct addrinfo     *result = NULL,
		hints;

	ZeroMemory(&hints, sizeof(hints));
	hints.ai_family = AF_INET;        // Internet address family is unspecified so
that either an IPv6 or IPv4 address can be returned
	hints.ai_socktype = SOCK_STREAM;  // Requests the socket type to be a stream
socket for the TCP protocol
	hints.ai_protocol = IPPROTO_TCP;
	hints.ai_flags = AI_PASSIVE;

	// Resolve the local address and port to be used by the server
	iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result);
	if (iResult != 0) {
		printf("server_polaris: getaddrinfo failed: %d\n", iResult);
		WSACleanup();
		return;
	}
```

```
	SOCKET ListenSocket = INVALID_SOCKET;

	// Create a SOCKET for the server to listen for client connections
	ListenSocket = socket(result->ai_family, result->ai_socktype, result-
>ai_protocol);

	if (ListenSocket == INVALID_SOCKET) {
		printf("server_polaris: Error at socket(): %d\n", WSAGetLastError());
		freeaddrinfo(result);
		WSACleanup();
		return;
	}

	// Setup the TCP listening socket
	iResult = bind(ListenSocket, result->ai_addr, (int)result->ai_addrlen);

	if (iResult == SOCKET_ERROR) {
		printf("server_polaris: bind failed: %d", WSAGetLastError());
		freeaddrinfo(result);
		closesocket(ListenSocket);
		WSACleanup();
		return;
	}

	freeaddrinfo(result);

	// To listen on a socket
	if (listen(ListenSocket, SOMAXCONN) == SOCKET_ERROR) {
		printf("server_polaris: listen failed: %d\n", WSAGetLastError());
		closesocket(ListenSocket);
		WSACleanup();
		return;
	}

	SOCKET ClientSocket;

	ClientSocket = INVALID_SOCKET;

	// Accept a client socket
	ClientSocket = accept(ListenSocket, NULL, NULL);

	if (ClientSocket == INVALID_SOCKET) {
		printf("server_polaris: accept failed: %d\n", WSAGetLastError());
		closesocket(ListenSocket);
		WSACleanup();
		return;
	}

	char recvbuf[DEFAULT_BUFFER_LENGTH];
	//int iSendResult;

	// reveice until the client shutdown the connection
	do {
		iResult = recv(ClientSocket, recvbuf, DEFAULT_BUFFER_LENGTH, 0);
		if (iResult > 0) {
			char msg[DEFAULT_BUFFER_LENGTH];
			memset(&msg, 0, sizeof(msg));
			strncpy_s(msg, recvbuf, iResult);
```

```
		//printf("Received: %s\n", msg);

		if (msg[1] == '1' && msg[2] == '(') {
			M1_OOV = FALSE;
			i = 0;
			msgstr = msg;
			msgstr.erase(strlen(msg) - 2, 1);
			msgstr.erase(0, 3);
			msgstr.append(",");
			pos = 0;

			while ((pos = msgstr.find(delimiter)) != string::npos) {
				number = msgstr.substr(0, pos);
				Receive_Data_M1[i] = stod(number);
				msgstr.erase(0, pos + delimiter.length());
				i++;
			}

			M1_x = Receive_Data_M1[0];
			M1_y = Receive_Data_M1[1];
			M1_z = Receive_Data_M1[2];
			M1_qw = Receive_Data_M1[3];
			M1_qx = Receive_Data_M1[4];
			M1_qy = Receive_Data_M1[5];
			M1_qz = Receive_Data_M1[6];
		}
		else if (msg[1] == '1' && msg[2] == '_') M1_OOV = TRUE;

		if (msg[1] == '2' && msg[2] == '(') {
			M2_OOV = FALSE;
			i = 0;
			msgstr = msg;
			msgstr.erase(strlen(msg) - 2, 1);
			msgstr.erase(0, 3);
			msgstr.append(",");
			pos = 0;

			while ((pos = msgstr.find(delimiter)) != string::npos) {
				number = msgstr.substr(0, pos);
				Receive_Data_M2[i] = stod(number);
				msgstr.erase(0, pos + delimiter.length());
				i++;
			}

			M2_x = Receive_Data_M2[0];
			M2_y = Receive_Data_M2[1];
			M2_z = Receive_Data_M2[2];
			M2_qw = Receive_Data_M2[3];
			M2_qx = Receive_Data_M2[4];
			M2_qy = Receive_Data_M2[5];
			M2_qz = Receive_Data_M2[6];
		}
		else if (msg[1] == '2' && msg[2] == '_') M2_OOV = TRUE;
	}
	else if (iResult == 0) {
		printf("server_polaris: Connection closed\n");
	}
	else {
		printf("server_polaris: recv failed: %d\n", WSAGetLastError());
		closesocket(ClientSocket);
```

```
                    WSACleanup();
                    return;
                }
        } while (iResult > 0);

        // Free the resouces
        closesocket(ListenSocket);
        WSACleanup();

        getchar();
        return;
}
```

A.3. client_RT.cpp

```
#define WIN32_LEAN_AND_MEAN

#include <WinSock2.h>
#include <WS2tcpip.h>
#include <string>
#include <sstream>
#include <iostream>

// link with Ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

#define DEFAULT_PORT "30003"
//#define DEFAULT_BUFFER_LENGTH   812 //UR 1.8
//#define DEFAULT_BUFFER_LENGTH   1044 //UR 3.1
#define DEFAULT_BUFFER_LENGTH     1060 //UR 3.2

using namespace std;
double X_in, Y_in, Z_in, Rx_in, Ry_in, Rz_in;

double bitstring_to_double(const string& s) {
	unsigned long long x = 0;
	for (string::const_iterator it = s.begin(); it != s.end(); ++it) {
		x = (x << 1) + (*it - '0');
	}
	double d;
	memcpy(&d, &x, 8);
	return d;
}

class Client_RT {
public:
	Client_RT(char* servername) {
		szServerName = "192.168.0.8"; //Robot IP
		ConnectSocket = INVALID_SOCKET;
	}

	bool Start() {
		WSADATA wsaData;

		// Initialize Winsock
		int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
		if (iResult != 0) {
			printf("client_RT: WSAStartup failed: %d\n", iResult);
			return false;
		}

		struct addrinfo      *result = NULL,
			*ptr = NULL,
			hints;

		ZeroMemory(&hints, sizeof(hints));
		hints.ai_family = AF_UNSPEC;
		hints.ai_socktype = SOCK_STREAM;
		hints.ai_protocol = IPPROTO_TCP;

		// Resolve the server address and port
		iResult = getaddrinfo(szServerName, DEFAULT_PORT, &hints, &result);
```

```
		if (iResult != 0) {
			printf("client_RT: getaddrinfo failed: %d\n", iResult);
			WSACleanup();
			return false;
		}

		ptr = result;

		// Create a SOCKET for connecting to server
		ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype, ptr-
>ai_protocol);

		if (ConnectSocket == INVALID_SOCKET) {
			printf("client_RT: Error at socket(): %d\n", WSAGetLastError());
			freeaddrinfo(result);
			WSACleanup();
			return false;
		}

		// Connect to server
		iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);

		if (iResult == SOCKET_ERROR) {
			closesocket(ConnectSocket);
			ConnectSocket = INVALID_SOCKET;
		}

		freeaddrinfo(result);

		if (ConnectSocket == INVALID_SOCKET) {
			printf("client_RT: Unable to connect to server!\n");
			WSACleanup();
			return false;
		}

		return true;
	};

	// Free the resouces
	void Stop() {
		int iResult = shutdown(ConnectSocket, SD_SEND);

		if (iResult == SOCKET_ERROR) {
			printf("client_RT: shutdown failed: %d\n", WSAGetLastError());
		}

		closesocket(ConnectSocket);
		WSACleanup();
	};

	// Send message to server
	bool Send(char* szMsg) {
		int iResult = send(ConnectSocket, szMsg, strlen(szMsg), 0);

		if (iResult == SOCKET_ERROR) {
			printf("sclient_RT: end failed: %d\n", WSAGetLastError());
			Stop();
			return false;
		}
```

```
		return true;
	};

	// Receive message from server
	bool Recv() {
		char recvbuf[DEFAULT_BUFFER_LENGTH];
		int iResult = recv(ConnectSocket, recvbuf, DEFAULT_BUFFER_LENGTH, 0);

		int i, recv_int, pos_b;
		char recv_bit[9], recv_bit1[9];
		string Xs, Ys, Zs, Rxs, Rys, Rzs;
		stringstream Xss, Yss, Zss, Rxss, Ryss, Rzss;

		do {
			iResult = recv(ConnectSocket, recvbuf, DEFAULT_BUFFER_LENGTH, 0);
			if (iResult > 0) {
				//printf("Bytes received: %d\n", iResult);

				//pos_b = 588; //UR1.8
				pos_b = 444; //UR3.2

				//X
				pos_b = pos_b;
				for (i = pos_b; i < pos_b + 8; i++) {
					if ((int)recvbuf[i] < 0) recv_int = recvbuf[i] +
256;
					else recv_int = recvbuf[i];
					_itoa_s(recv_int, recv_bit, 2);
					if (strlen(recv_bit) == 8) strcpy_s(recv_bit1,
recv_bit);
					else if (strlen(recv_bit) == 7) {
						strcpy_s(recv_bit1, "0");
						strncat_s(recv_bit1, recv_bit, 7);
					}
					else if (strlen(recv_bit) == 6) {
						strcpy_s(recv_bit1, "00");
						strncat_s(recv_bit1, recv_bit, 6);
					}
					else if (strlen(recv_bit) == 5) {
						strcpy_s(recv_bit1, "000");
						strncat_s(recv_bit1, recv_bit, 5);
					}
					else if (strlen(recv_bit) == 4) {
						strcpy_s(recv_bit1, "0000");
						strncat_s(recv_bit1, recv_bit, 4);
					}
					else if (strlen(recv_bit) == 3) {
						strcpy_s(recv_bit1, "00000");
						strncat_s(recv_bit1, recv_bit, 3);
					}
					else if (strlen(recv_bit) == 2) {
						strcpy_s(recv_bit1, "000000");
						strncat_s(recv_bit1, recv_bit, 2);
					}
					else if (strlen(recv_bit) == 1) {
						strcpy_s(recv_bit1, "0000000");
						strncat_s(recv_bit1, recv_bit, 1);
					}
					Xss << recv_bit1;
				}
```

```
                        Xs = Xss.str();
                        //cout << "X= " << Xs << "\n";

                        //Y
                        pos_b = pos_b + 8;
                        for (i = pos_b; i < pos_b + 8; i++) {
                              if ((int)recvbuf[i] < 0) recv_int = recvbuf[i] +
256;
                              else recv_int = recvbuf[i];
                              _itoa_s(recv_int, recv_bit, 2);
                              if (strlen(recv_bit) == 8) strcpy_s(recv_bit1,
recv_bit);
                              else if (strlen(recv_bit) == 7) {
                                    strcpy_s(recv_bit1, "0");
                                    strncat_s(recv_bit1, recv_bit, 7);
                              }
                              else if (strlen(recv_bit) == 6) {
                                    strcpy_s(recv_bit1, "00");
                                    strncat_s(recv_bit1, recv_bit, 6);
                              }
                              else if (strlen(recv_bit) == 5) {
                                    strcpy_s(recv_bit1, "000");
                                    strncat_s(recv_bit1, recv_bit, 5);
                              }
                              else if (strlen(recv_bit) == 4) {
                                    strcpy_s(recv_bit1, "0000");
                                    strncat_s(recv_bit1, recv_bit, 4);
                              }
                              else if (strlen(recv_bit) == 3) {
                                    strcpy_s(recv_bit1, "00000");
                                    strncat_s(recv_bit1, recv_bit, 3);
                              }
                              else if (strlen(recv_bit) == 2) {
                                    strcpy_s(recv_bit1, "000000");
                                    strncat_s(recv_bit1, recv_bit, 2);
                              }
                              else if (strlen(recv_bit) == 1) {
                                    strcpy_s(recv_bit1, "0000000");
                                    strncat_s(recv_bit1, recv_bit, 1);
                              }
                              Yss << recv_bit1;
                        }
                        Ys = Yss.str();
                        //cout << "Y= " << Ys << "\n";

                        //Z
                        pos_b = pos_b + 8;
                        for (i = pos_b; i < pos_b + 8; i++) {
                              if ((int)recvbuf[i] < 0) recv_int = recvbuf[i] +
256;
                              else recv_int = recvbuf[i];
                              _itoa_s(recv_int, recv_bit, 2);
                              if (strlen(recv_bit) == 8) strcpy_s(recv_bit1,
recv_bit);
                              else if (strlen(recv_bit) == 7) {
                                    strcpy_s(recv_bit1, "0");
                                    strncat_s(recv_bit1, recv_bit, 7);
                              }
                              else if (strlen(recv_bit) == 6) {
                                    strcpy_s(recv_bit1, "00");
```

```
                                        strncat_s(recv_bit1, recv_bit, 6);
                                }
                                else if (strlen(recv_bit) == 5) {
                                        strcpy_s(recv_bit1, "000");
                                        strncat_s(recv_bit1, recv_bit, 5);
                                }
                                else if (strlen(recv_bit) == 4) {
                                        strcpy_s(recv_bit1, "0000");
                                        strncat_s(recv_bit1, recv_bit, 4);
                                }
                                else if (strlen(recv_bit) == 3) {
                                        strcpy_s(recv_bit1, "00000");
                                        strncat_s(recv_bit1, recv_bit, 3);
                                }
                                else if (strlen(recv_bit) == 2) {
                                        strcpy_s(recv_bit1, "000000");
                                        strncat_s(recv_bit1, recv_bit, 2);
                                }
                                else if (strlen(recv_bit) == 1) {
                                        strcpy_s(recv_bit1, "0000000");
                                        strncat_s(recv_bit1, recv_bit, 1);
                                }
                                Zss << recv_bit1;
                        }
                        Zs = Zss.str();
                        //cout << "Z= " << Zs << "\n";

                        //Rx
                        pos_b = pos_b + 8;
                        for (i = pos_b; i < pos_b + 8; i++) {
                                if ((int)recvbuf[i] < 0) recv_int = recvbuf[i] +
256;
                                else recv_int = recvbuf[i];
                                _itoa_s(recv_int, recv_bit, 2);
                                if (strlen(recv_bit) == 8) strcpy_s(recv_bit1,
recv_bit);
                                else if (strlen(recv_bit) == 7) {
                                        strcpy_s(recv_bit1, "0");
                                        strncat_s(recv_bit1, recv_bit, 7);
                                }
                                else if (strlen(recv_bit) == 6) {
                                        strcpy_s(recv_bit1, "00");
                                        strncat_s(recv_bit1, recv_bit, 6);
                                }
                                else if (strlen(recv_bit) == 5) {
                                        strcpy_s(recv_bit1, "000");
                                        strncat_s(recv_bit1, recv_bit, 5);
                                }
                                else if (strlen(recv_bit) == 4) {
                                        strcpy_s(recv_bit1, "0000");
                                        strncat_s(recv_bit1, recv_bit, 4);
                                }
                                else if (strlen(recv_bit) == 3) {
                                        strcpy_s(recv_bit1, "00000");
                                        strncat_s(recv_bit1, recv_bit, 3);
                                }
                                else if (strlen(recv_bit) == 2) {
                                        strcpy_s(recv_bit1, "000000");
                                        strncat_s(recv_bit1, recv_bit, 2);
                                }
```

```
                        else if (strlen(recv_bit) == 1) {
                                strcpy_s(recv_bit1, "0000000");
                                strncat_s(recv_bit1, recv_bit, 1);
                        }
                        Rxss << recv_bit1;
                }
                Rxs = Rxss.str();
                //cout << "Rx= " << Rxs << "\n";

                //Ry
                pos_b = pos_b + 8;
                for (i = pos_b; i < pos_b + 8; i++) {
                        if ((int)recvbuf[i] < 0) recv_int = recvbuf[i] +
256;
                        else recv_int = recvbuf[i];
                        _itoa_s(recv_int, recv_bit, 2);
                        if (strlen(recv_bit) == 8) strcpy_s(recv_bit1,
recv_bit);
                        else if (strlen(recv_bit) == 7) {
                                strcpy_s(recv_bit1, "0");
                                strncat_s(recv_bit1, recv_bit, 7);
                        }
                        else if (strlen(recv_bit) == 6) {
                                strcpy_s(recv_bit1, "00");
                                strncat_s(recv_bit1, recv_bit, 6);
                        }
                        else if (strlen(recv_bit) == 5) {
                                strcpy_s(recv_bit1, "000");
                                strncat_s(recv_bit1, recv_bit, 5);
                        }
                        else if (strlen(recv_bit) == 4) {
                                strcpy_s(recv_bit1, "0000");
                                strncat_s(recv_bit1, recv_bit, 4);
                        }
                        else if (strlen(recv_bit) == 3) {
                                strcpy_s(recv_bit1, "00000");
                                strncat_s(recv_bit1, recv_bit, 3);
                        }
                        else if (strlen(recv_bit) == 2) {
                                strcpy_s(recv_bit1, "000000");
                                strncat_s(recv_bit1, recv_bit, 2);
                        }
                        else if (strlen(recv_bit) == 1) {
                                strcpy_s(recv_bit1, "0000000");
                                strncat_s(recv_bit1, recv_bit, 1);
                        }
                        Ryss << recv_bit1;
                }
                Rys = Ryss.str();
                //cout << "Ry= " << Rys << "\n";

                //Rz
                pos_b = pos_b + 8;
                for (i = pos_b; i < pos_b + 8; i++) {
                        if ((int)recvbuf[i] < 0) recv_int = recvbuf[i] +
256;
                        else recv_int = recvbuf[i];
                        _itoa_s(recv_int, recv_bit, 2);
                        if (strlen(recv_bit) == 8) strcpy_s(recv_bit1,
recv_bit);
```

```
                        else if (strlen(recv_bit) == 7) {
                                strcpy_s(recv_bit1, "0");
                                strncat_s(recv_bit1, recv_bit, 7);
                        }
                        else if (strlen(recv_bit) == 6) {
                                strcpy_s(recv_bit1, "00");
                                strncat_s(recv_bit1, recv_bit, 6);
                        }
                        else if (strlen(recv_bit) == 5) {
                                strcpy_s(recv_bit1, "000");
                                strncat_s(recv_bit1, recv_bit, 5);
                        }
                        else if (strlen(recv_bit) == 4) {
                                strcpy_s(recv_bit1, "0000");
                                strncat_s(recv_bit1, recv_bit, 4);
                        }
                        else if (strlen(recv_bit) == 3) {
                                strcpy_s(recv_bit1, "00000");
                                strncat_s(recv_bit1, recv_bit, 3);
                        }
                        else if (strlen(recv_bit) == 2) {
                                strcpy_s(recv_bit1, "000000");
                                strncat_s(recv_bit1, recv_bit, 2);
                        }
                        else if (strlen(recv_bit) == 1) {
                                strcpy_s(recv_bit1, "0000000");
                                strncat_s(recv_bit1, recv_bit, 1);
                        }
                        Rzss << recv_bit1;
                }
                Rzs = Rzss.str();
                //cout << "Rz= " << Rzs << "\n";

                X_in = bitstring_to_double(Xs) * 1000;
                //cout << "X= " << X_in << '      ';

                Y_in = bitstring_to_double(Ys) * 1000;
                //cout << "Y= " << Y_in << '      ';

                Z_in = bitstring_to_double(Zs) * 1000;
                //cout << "Z= " << Z_in << '      ';

                Rx_in = bitstring_to_double(Rxs);
                //cout << "Rx= " << Rx_in << '    ';

                Ry_in = bitstring_to_double(Rys);
                //cout << "Ry= " << Ry_in << '    ';

                Rz_in = bitstring_to_double(Rzs);
                //cout << "Rz= " << Rz_in << '\n';

                Xss.str(""); //clear Xss
                Yss.str(""); //clear Yss
                Zss.str(""); //clear Zss
                Rxss.str(""); //clear Rxss
                Ryss.str(""); //clear Ryss
                Rzss.str(""); //clear Rzss
        }
        else if (iResult == 0)
                printf("client_RT: Connection closed\n");
```

```
                    else
                           printf("client_RT: recv failed with error: %d\n",
WSAGetLastError());

             } while (iResult > 0);

             return false;
       }

private:
       char* szServerName;
       SOCKET ConnectSocket;
};

void client_RT(void *P)
{
       Client_RT client("127.0.0.1");

       if (!client.Start())
             return;

       while (true) {
             client.Recv();
       }

       client.Stop();

       getchar();
       return;
}
```

A.4. math.cpp

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <windows.h>
#include <Eigen/Dense>
#include <Eigen/Geometry>

using namespace std;
using namespace Eigen;

bool sending(FALSE);
bool step_motion(FALSE);
bool tracking(FALSE);
bool step_tracking(FALSE);
string math_selection;
Vector3d RxRyRz_in;
double theta_in;
Matrix3d R_P1, R_P2, R_P3, R_P4;
Vector3d t_P1, t_P2, t_P3, t_P4;
MatrixXd A(9, 3);
VectorXd B(9);
Vector3d TCP_P;
double X_TCP, Y_TCP, Z_TCP;
fstream TCP_Position;
extern int sleep1;
stringstream set_tcp;
string sendstr;
Vector3d T0_R, T1_R, T2_R, X_R, Y_R, Z_R;
Vector3d T0_P, T1_P, T2_P, X_P, Y_P, Z_P;
extern double X_in, Y_in, Z_in, Rx_in, Ry_in, Rz_in;
Matrix4d T_Robot_CS, T_Polaris_M1_CS;
Matrix3d R_Robot_in;
Matrix4d T_Robot_actual;
Matrix4d T_M1_P, T_M1;
Quaterniond M1_q, M2_q;
extern double M1_x, M1_y, M1_z, M1_qw, M1_qx, M1_qy, M1_qz;
extern double M2_x, M2_y, M2_z, M2_qw, M2_qx, M2_qy, M2_qz;
Matrix3d R_M1, R_M2;
Matrix4d T_Robot;
Matrix4d T_Polaris;
Matrix3d R_RP;
double theta_TCP, Rx_TCP, Ry_TCP, Rz_TCP;
fstream TCP_Orientation;
stringstream program;
extern double x_offset, y_offset, z_offset;
Matrix4d T_TTO_M1, T_TTO_M2;
Vector3d RxRyRz_TCP;
Matrix3d R_TCP;
Matrix4d T_TCP;
extern double roll, pitch, yaw;
Matrix4d R_rotM2;
Matrix4d T_M2_P, T_M2;
double E;
bool MinMove;
extern double MinDist;
Matrix4d T_M2inM1;
```

```
Matrix4d T_Robot_target;
Matrix3d R_Robot_out;
double theta_out, X_out, Y_out, Z_out, Rx_out, Ry_out, Rz_out;
stringstream movejss;
extern double a, v, r;
extern bool M1_OOV, M2_OOV;
stringstream sendbufss;
string sendbufs;
char *sendbuf;
double X_out1, Y_out1, Z_out1, Rx_out1, Ry_out1, Rz_out1;

void PressEnterToContinue() {
	int c;
	printf("Pritisni ENTER za nastavak...\n");
	fflush(stdout);
	do c = getchar(); while ((c != '\n') && (c != EOF));
}

void math(void *P) {
start:
	Sleep(500);
	sending = FALSE;
	step_motion = FALSE;
	tracking = FALSE;
	step_tracking = FALSE;

	cout << "1. Kalibracija pozicije vrha alata\n";
	cout << "2. Kalibracija orijentacije alata\n";
	cout << "3. Step pracenje\n";
	cout << "4. Kontinuirano pracenje\n";
start1:
	getline(cin, math_selection);

	if (math_selection.compare("1") == 0) {
		cout << "\nKalibracija pozicije vrha alata.\n\n";
		goto Position;
	}
	else if (math_selection.compare("2") == 0) {
		cout << "\nKalibracija orijentacije alata.\n\n";
		goto Orientation;
	}
	else if (math_selection.compare("3") == 0) {
		cout << "\nStep pracenje.\n\n";
		step_tracking = TRUE;
		goto Tracking;
	}
	else if (math_selection.compare("4") == 0) {
		cout << "\nKontinuirano pracenje.\n\n";
		goto Tracking;
	}
	else if (math_selection.compare("") == 0) {
		goto start1;
	}
	else {
		cout << "\nKrivi unos.\n\n";
		goto start1;
	}

Position:
	cout << "Pomakni robot u polozaj P1!\n";
```

```
        PressEnterToContinue();
        RxRyRz_in << Rx_in, Ry_in, Rz_in;
        theta_in = sqrt(pow(Rx_in, 2) + pow(Ry_in, 2) + pow(Rz_in, 2));
        R_P1 = AngleAxisd(theta_in, RxRyRz_in.normalized()); //AxisAngle to DCM
        cout << "R_P1= " << R_P1 << "\n";
        t_P1 << X_in, Y_in, Z_in;
        cout << "t_P1= " << t_P1 << "\n";

        cout << "\nPomakni robot u polozaj P2!\n";
        PressEnterToContinue();
        RxRyRz_in << Rx_in, Ry_in, Rz_in;
        theta_in = sqrt(pow(Rx_in, 2) + pow(Ry_in, 2) + pow(Rz_in, 2));
        R_P2 = AngleAxisd(theta_in, RxRyRz_in.normalized()); //AxisAngle to DCM
        cout << "R_P2= " << R_P2 << "\n";
        t_P2 << X_in, Y_in, Z_in;
        cout << "t_P2= " << t_P2 << "\n";

        cout << "\nPomakni robot u polozaj P3!\n";
        PressEnterToContinue();
        RxRyRz_in << Rx_in, Ry_in, Rz_in;
        theta_in = sqrt(pow(Rx_in, 2) + pow(Ry_in, 2) + pow(Rz_in, 2));
        R_P3 = AngleAxisd(theta_in, RxRyRz_in.normalized()); //AxisAngle to DCM
        cout << "R_P3= " << R_P3 << "\n";
        t_P3 << X_in, Y_in, Z_in;
        cout << "t_P3= " << t_P3 << "\n";

        cout << "\nPomakni robot u polozaj P4!\n";
        PressEnterToContinue();
        RxRyRz_in << Rx_in, Ry_in, Rz_in;
        theta_in = sqrt(pow(Rx_in, 2) + pow(Ry_in, 2) + pow(Rz_in, 2));
        R_P4 = AngleAxisd(theta_in, RxRyRz_in.normalized()); //AxisAngle to DCM
        cout << "R_P4= " << R_P4 << "\n";
        t_P4 << X_in, Y_in, Z_in;
        cout << "t_P4= " << t_P4 << "\n";

        A << (R_P2 - R_P1), (R_P3 - R_P2), (R_P4 - R_P3);
        cout << "\nA= " << A << "\n";

        B << (t_P1 - t_P2), (t_P2 - t_P3), (t_P3 - t_P4);
        cout << "\nB= " << B << "\n";

        TCP_P = (((A.transpose()*A).inverse())*A.transpose())*B;
        cout << "\nTCP_P= " << TCP_P << "\n";
        PressEnterToContinue();

        X_TCP = TCP_P(0) / 1000;
        Y_TCP = TCP_P(1) / 1000;
        Z_TCP = TCP_P(2) / 1000;

        sending = TRUE;

        set_tcp.str("");
        set_tcp << "set_tcp(p[" << X_TCP << "," << Y_TCP << "," << Z_TCP <<
",0,0,0])\n";
        sendstr = set_tcp.str();
        Sleep(sleep1 + 100);
        cout << sendstr << "\n";

        TCP_Position.open("TCP_Position.txt", fstream::out);
        if (TCP_Position.is_open()) {
```

```
			TCP_Position << X_TCP << "\n" << Y_TCP << "\n" << Z_TCP;
			TCP_Position.close();
		}
		else cout << "Unable to open file TCP_Position.txt\n\n";

		goto start;

Orientation:
		sending = TRUE;

		TCP_Position.open("TCP_Position.txt", fstream::in);
		if (TCP_Position.is_open()) {
			TCP_Position >> X_TCP >> Y_TCP >> Z_TCP;
			TCP_Position.close();
		}
		else {
			cout << "Unable to open file TCP_Position.txt\n\n";
			goto start;
		}

		Sleep(2000);
		sendstr = "set_tcp(p[0, 0, 0, 0, 0, 0])\n";
		Sleep(sleep1 + 100);

		step_motion = TRUE;

		sendstr = "movej(p[-0.5, -0.1, -0.2, 0.2, -0.3, 1.5], a=1.4, v=1.05)\n"; //T0
		cout << "Pomicanje u T0...\n";
		do {} while (((pow(-0.5 - (X_in / 1000), 2)) > 0.000001));
		Sleep(1000);
		T0_R << X_in, Y_in, Z_in;
		T0_P << M1_x, M1_y, M1_z;

		sendstr = "movej(p[-0.5, -0.1, -0.1, 0.2, -0.3, 1.5], a=1.4, v=1.05)\n"; //T1
		cout << "Pomicanje u T1...\n";
		do {} while (((pow(-0.1 - (Z_in / 1000), 2)) > 0.000001));
		Sleep(1000);
		T1_R << X_in, Y_in, Z_in;
		T1_P << M1_x, M1_y, M1_z;

		sendstr = "movej(p[-0.45, -0.1, -0.15, 0.2, -0.3, 1.5], a=1.4, v=1.05)\n"; //T2
		cout << "Pomicanje u T2...\n";
		do {} while (((pow(-0.45 - (X_in / 1000), 2)) > 0.000001));
		Sleep(1000);
		T2_R << X_in, Y_in, Z_in;
		T2_P << M1_x, M1_y, M1_z;

		step_motion = FALSE;

		//Kordinatni sustav robota
		X_R = (T1_R - T0_R).normalized();
		Y_R = (T2_R - T0_R).normalized();
		Z_R = (Y_R.cross(X_R)).normalized();
		Y_R = (X_R.cross(Z_R)).normalized();
		T_Robot_CS << X_R(0), Y_R(0), Z_R(0), T0_R(0), X_R(1), Y_R(1), Z_R(1), T0_R(1),
X_R(2), Y_R(2), Z_R(2), T0_R(2), 0, 0, 0, 1;

		//Kordinatni sustav Polarisa
		X_P = (T1_P - T0_P).normalized();
		Y_P = (T2_P - T0_P).normalized();
```

```
		Z_P = (Y_P.cross(X_P)).normalized();
		Y_P = (X_P.cross(Z_P)).normalized();
		T_Polaris_M1_CS << X_P(0), Y_P(0), Z_P(0), T0_P(0), X_P(1), Y_P(1), Z_P(1),
T0_P(1), X_P(2), Y_P(2), Z_P(2), T0_P(2), 0, 0, 0, 1;

		//Trenutna pozicija i orijentacija vrha alata u baznom koordinatnom sustavu
robota
		RxRyRz_in << Rx_in, Ry_in, Rz_in;
		theta_in = sqrt(pow(Rx_in, 2) + pow(Ry_in, 2) + pow(Rz_in, 2));
		R_Robot_in = AngleAxisd(theta_in, RxRyRz_in.normalized()); //AxisAngle to DCM
		T_Robot_actual << R_Robot_in(0, 0), R_Robot_in(0, 1), R_Robot_in(0, 2), X_in,
R_Robot_in(1, 0), R_Robot_in(1, 1), R_Robot_in(1, 2), Y_in, R_Robot_in(2, 0),
R_Robot_in(2, 1), R_Robot_in(2, 2), Z_in, 0, 0, 0, 1;

		//Trenutna pozicija i orijentacija alata M1 u baznom koordinatnom sustavu
Polarisa
		M1_q.w() = M1_qw;
		M1_q.x() = M1_qx;
		M1_q.y() = M1_qy;
		M1_q.z() = M1_qz;
		R_M1 = M1_q.normalized().toRotationMatrix(); //Quaternion to DCM
		T_M1 << R_M1(0, 0), R_M1(0, 1), R_M1(0, 2), M1_x, R_M1(1, 0), R_M1(1, 1),
R_M1(1, 2), M1_y, R_M1(2, 0), R_M1(2, 1), R_M1(2, 2), M1_z, 0, 0, 0, 1;

		//Trenutna pozicija i orijentacija vrha alata robota u koordinatnom sustavu
robota
		T_Robot = T_Robot_CS.inverse()*T_Robot_actual;

		//Trenutna pozicija i orijentacija alata M1 u koordinatnom sustavu Polarisa
		T_Polaris = T_Polaris_M1_CS.inverse()*T_M1;

		//Matrica rotacije - veza između robota i Polarisa
		R_RP = (T_Robot.inverse().block<3, 3>(0, 0))*T_Polaris.block<3, 3>(0, 0);

		//DCM to AxisAngle
		theta_TCP = acos((R_RP(0, 0) + R_RP(1, 1) + R_RP(2, 2) - 1) / 2);
		Rx_TCP = (R_RP(2, 1) - R_RP(1, 2)) / (2 * sin(theta_TCP))*theta_TCP;
		Ry_TCP = (R_RP(0, 2) - R_RP(2, 0)) / (2 * sin(theta_TCP))*theta_TCP;
		Rz_TCP = (R_RP(1, 0) - R_RP(0, 1)) / (2 * sin(theta_TCP))*theta_TCP;

		//Postavljanje robotovog TCP-a
		set_tcp.str("");
		set_tcp << "set_tcp(p[" << X_TCP << "," << Y_TCP << "," << Z_TCP << "," <<
Rx_TCP << "," << Ry_TCP << "," << Rz_TCP << "])\n";
		sendstr = set_tcp.str();
		Sleep(sleep1 + 100);
		cout << sendstr << "\n";

		TCP_Orientation.open("TCP_Orientation.txt", fstream::out);
		if (TCP_Orientation.is_open()) {
			TCP_Orientation << Rx_TCP << "\n" << Ry_TCP << "\n" << Rz_TCP;
			TCP_Orientation.close();
		}
		else cout << "Unable to open file TCP_Orientation.txt\n\n";

		goto start;

Tracking:
		TCP_Position.open("TCP_Position.txt", fstream::in);
		if (TCP_Position.is_open()) {
```

```
			TCP_Position >> X_TCP >> Y_TCP >> Z_TCP;
			TCP_Position.close();
		}
		else cout << "Unable to open file TCP_Position.txt\n\n";

		TCP_Orientation.open("TCP_Orientation.txt", fstream::in);
		if (TCP_Orientation.is_open()) {
			TCP_Orientation >> Rx_TCP >> Ry_TCP >> Rz_TCP;
			TCP_Orientation.close();
		}
		else cout << "Unable to open file TCP_Orientation.txt\n\n";

		T_TTO_M1 << 1, 0, 0, -30, 0, 1, 0, -260, 0, 0, 1, 40, 0, 0, 0, 1;

		RxRyRz_TCP << Rx_TCP, Ry_TCP, Rz_TCP;
		theta_TCP = sqrt(pow(Rx_TCP, 2) + pow(Ry_TCP, 2) + pow(Rz_TCP, 2));
		R_TCP = AngleAxisd(theta_TCP, RxRyRz_TCP.normalized()); //AxisAngle to DCM
		T_TCP << R_TCP(0, 0), R_TCP(0, 1), R_TCP(0, 2), X_TCP * 1000, R_TCP(1, 0),
R_TCP(1, 1), R_TCP(1, 2), Y_TCP * 1000, R_TCP(2, 0), R_TCP(2, 1), R_TCP(2, 2), Z_TCP *
1000, 0, 0, 0, 1;

		T_TCP = T_TCP*T_TTO_M1;

		X_TCP = T_TCP(0, 3) / 1000;
		Y_TCP = T_TCP(1, 3) / 1000;
		Z_TCP = T_TCP(2, 3) / 1000;

		//DCM to AxisAngle
		theta_TCP = acos((T_TCP(0, 0) + T_TCP(1, 1) + T_TCP(2, 2) - 1) / 2);
		Rx_TCP = (T_TCP(2, 1) - T_TCP(1, 2)) / (2 * sin(theta_TCP))*theta_TCP;
		Ry_TCP = (T_TCP(0, 2) - T_TCP(2, 0)) / (2 * sin(theta_TCP))*theta_TCP;
		Rz_TCP = (T_TCP(1, 0) - T_TCP(0, 1)) / (2 * sin(theta_TCP))*theta_TCP;

		sending = TRUE;

		if (step_tracking == TRUE) {
			set_tcp.str("");
			set_tcp << "set_tcp(p[" << X_TCP << "," << Y_TCP << "," << Z_TCP << ","
<< Rx_TCP << "," << Ry_TCP << "," << Rz_TCP << "])\n";
			sendstr = set_tcp.str();
			Sleep(sleep1 + 100);
			cout << sendstr << "\n";

			step_motion = TRUE;

			goto Start_Tracking;
		}

		program.str("");
		program << "def nikola_diplomski() :\n";
		program << "  set_standard_analog_input_domain(0, 1)\n";
		program << "  set_standard_analog_input_domain(1, 1)\n";
		program << "  set_tool_analog_input_domain(0, 1)\n";
		program << "  set_tool_analog_input_domain(1, 1)\n";
		program << "  set_analog_outputdomain(0, 0)\n";
		program << "  set_analog_outputdomain(1, 0)\n";
		program << "  set_tool_voltage(0)\n";
		program << "  set_standard_digital_input_action(0, \"default\")\n";
		program << "  set_standard_digital_input_action(1, \"default\")\n";
		program << "  set_standard_digital_input_action(2, \"default\")\n";
```

```
        program << "  set_standard_digital_input_action(3, \"default\")\n";
        program << "  set_standard_digital_input_action(4, \"default\")\n";
        program << "  set_standard_digital_input_action(5, \"default\")\n";
        program << "  set_standard_digital_input_action(6, \"default\")\n";
        program << "  set_standard_digital_input_action(7, \"default\")\n";
        program << "  set_tool_digital_input_action(0, \"default\")\n";
        program << "  set_tool_digital_input_action(1, \"default\")\n";
        program << "  set_tcp(p[" << X_TCP << ", " << Y_TCP << ", " << Z_TCP << ", " <<
Rx_TCP << ", " << Ry_TCP << ", " << Rz_TCP << "])\n";
        program << "  set_payload(0.4)\n";
        program << "  set_gravity([6.943788591251896, -6.943788591251898,
6.013015783813504E-16])\n";
        program << "  $ 1 \"BeforeStart\"\n";
        program << "  $ 2 \"Receive_Data:=[0,0,0,0,0,0,0,0,0,0,0,0]\"\n";
        program << "  global Receive_Data = [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]\n";
        program << "  $ 3 \"a:=1.4\"\n";
        program << "  global a = 1.4\n";
        program << "  $ 4 \"v:=1.05\"\n";
        program << "  global v = 1.05\n";
        program << "  $ 5 \"r:=0\"\n";
        program << "  global r = 0\n";
        program << "  $ 6 \"P1:=get_actual_tcp_pose()\"\n";
        program << "  global P1 = get_actual_tcp_pose()\n";
        program << "  $ 7 \"socket_open('192.168.0.100', 30000)\"\n";
        program << "  socket_open(\"192.168.0.100\", 30000)\n";
        program << "  $ 8 \"Wait: 2.0\"\n";
        program << "  sleep(2.0)\n";
        program << "  $ 27 \"Thread_2\"\n";
        program << "  thread Thread_2() :\n";
        program << "  while (True) :\n";
        program << "         socket_send_byte(1)\n";
        program << "         sleep(0.1)\n";
        program << "  end\n";
        program << "  end\n";
        program << "         threadId_Thread_2 = run Thread_2()\n";
        program << "         $ 16 \"Thread_1\"\n";
        program << "         thread Thread_1() :\n";
        program << "         while (True) :\n";
        program << "                Receive_Data = socket_read_ascii_float(10)\n";
        program << "                global Pointer = 0\n";
        program << "                sleep(0.01)\n";
        program << "                while (Pointer + 4<Receive_Data[0]) :\n";
        program << "                       P1[Pointer] = Receive_Data[Pointer + 2]\n";
        program << "                       global Pointer = Pointer + 1\n";
        program << "                end\n";
        program << "                       global a = Receive_Data[8]\n";
        program << "                       global v = Receive_Data[9]\n";
        program << "                       global r = Receive_Data[10]\n";
        program << "         end\n";
        program << "         end\n";
        program << "                       threadId_Thread_1 = run Thread_1()\n";
        program << "                       while (True) :\n";
        program << "                              $ 9 \"Robot Program\"\n";
        program << "                              $ 10 \"stopj(a/4)\"\n";
        program << "                              stopj(a/4)\n";
        program << "                              $ 11 \"Wait: 0.01\"\n";
        program << "                              sleep(0.01)\n";
        program << "                              $ 12 \"Loop Receive_Data[1]>0\"\n";
        program << "                              thread Thread_while_13() :\n";
        program << "                              while (True) :\n";
```

```
	program << "                                        $ 13 \"movej(P1, a=a, v=v,
r=r)\"\n";
	program << "                                        movej(P1, a = a, v = v, r =
r)\n";
	program << "                                        $ 14 \"stopj(a/4)\"\n";
	program << "                                        stopj(a/4)\n";
	program << "                                        $ 15 \"Wait: 0.01\"\n";
	program << "                                        sleep(0.01)\n";
	program << "                                end\n";
	program << "                                end\n";
	program << "                                        if (Receive_Data[1]>0) :\n";
	program << "                                                global thread_handler_13
= run Thread_while_13()\n";
	program << "                                                while (Receive_Data[1]>0)
:\n";
	program << "                                                        sync()\n";
	program << "                                                end\n";
	program << "                                                        kill
thread_handler_13\n";
	program << "                        end\n";
	program << "        end\n";
	program << "end\n";

	sendstr = program.str();
	Sleep(sleep1 + 100);

	sending = FALSE;

Start_Tracking:

	tracking = TRUE;

	do {
		T_TTO_M2 << 1, 0, 0, x_offset, 0, 1, 0, 28.59 + y_offset, 0, 0, 1, 41.02
+ z_offset, 0, 0, 0, 1;

		R_rotM2 << cos(pitch)*cos(yaw), cos(roll)*sin(yaw) +
sin(roll)*sin(pitch)*cos(yaw), sin(roll)*sin(yaw) - cos(roll)*sin(pitch)*cos(yaw), 0,
-cos(pitch)*sin(yaw), cos(roll)*cos(yaw) - sin(roll)*sin(pitch)*sin(yaw),
sin(roll)*cos(yaw) + cos(roll)*sin(pitch)*sin(yaw), 0, sin(pitch), -
sin(roll)*cos(pitch), cos(roll)*cos(pitch), 0, 0, 0, 0, 1;

		M1_q.w() = M1_qw;
		M1_q.x() = M1_qx;
		M1_q.y() = M1_qy;
		M1_q.z() = M1_qz;
		R_M1 = M1_q.normalized().toRotationMatrix(); //Quaternion to DCM
		T_M1_P << R_M1(0, 0), R_M1(0, 1), R_M1(0, 2), M1_x, R_M1(1, 0), R_M1(1,
1), R_M1(1, 2), M1_y, R_M1(2, 0), R_M1(2, 1), R_M1(2, 2), M1_z, 0, 0, 0, 1;
		T_M1 = T_M1_P*T_TTO_M1;

		M2_q.w() = M2_qw;
		M2_q.x() = M2_qx;
		M2_q.y() = M2_qy;
		M2_q.z() = M2_qz;
		R_M2 = M2_q.normalized().toRotationMatrix(); //Quaternion to DCM
		T_M2_P << R_M2(0, 0), R_M2(0, 1), R_M2(0, 2), M2_x, R_M2(1, 0), R_M2(1,
1), R_M2(1, 2), M2_y, R_M2(2, 0), R_M2(2, 1), R_M2(2, 2), M2_z, 0, 0, 0, 1;
		T_M2 = T_M2_P*T_TTO_M2*R_rotM2;
```

```
			E = sqrt(pow(T_M2(0, 3) - T_M1(0, 3), 2) + pow(T_M2(1, 3) - T_M1(1, 3),
2) + pow(T_M2(2, 3) - T_M1(2, 3), 2)); //Euklidska udaljenost
			//cout << "Euklidska udaljenost= " << E << "mm \n";
			if (E > MinDist) { MinMove = FALSE; }
			else { MinMove = TRUE; }

			T_M2inM1 = T_M1.inverse()*T_M2;

			RxRyRz_in << Rx_in, Ry_in, Rz_in;
			theta_in = sqrt(pow(Rx_in, 2) + pow(Ry_in, 2) + pow(Rz_in, 2));
			R_Robot_in = AngleAxisd(theta_in, RxRyRz_in.normalized()); //AxisAngle to
DCM

			T_Robot_actual << R_Robot_in(0, 0), R_Robot_in(0, 1), R_Robot_in(0, 2),
X_in, R_Robot_in(1, 0), R_Robot_in(1, 1), R_Robot_in(1, 2), Y_in, R_Robot_in(2, 0),
R_Robot_in(2, 1), R_Robot_in(2, 2), Z_in, 0, 0, 0, 1;

			T_Robot_target = T_Robot_actual*T_M2inM1;

			X_out = T_Robot_target(0, 3) / 1000;
			Y_out = T_Robot_target(1, 3) / 1000;
			Z_out = T_Robot_target(2, 3) / 1000;

			R_Robot_out = T_Robot_target.block<3, 3>(0, 0);
			theta_out = acos((R_Robot_out(0, 0) + R_Robot_out(1, 1) + R_Robot_out(2,
2) - 1) / 2);
			Rx_out = (R_Robot_out(2, 1) - R_Robot_out(1, 2)) / (2 *
sin(theta_out))*theta_out;
			Ry_out = (R_Robot_out(0, 2) - R_Robot_out(2, 0)) / (2 *
sin(theta_out))*theta_out;
			Rz_out = (R_Robot_out(1, 0) - R_Robot_out(0, 1)) / (2 *
sin(theta_out))*theta_out;

			if (M1_OOV == FALSE && M2_OOV == FALSE && MinMove == FALSE) {
				if (step_motion == FALSE) {
					sendbufss.str("");
					//sendbufss= " 0/1 X Y Z Rx Ry Rz a v r "
					sendbufss << "(1," << X_out << "," << Y_out << "," << Z_out
<< "," << Rx_out << "," << Ry_out << "," << Rz_out << "," << a << "," << v << "," << r
<< ")";
					sendbufs = sendbufss.str();
					//cout << sendbufs << "\n";

					sendbuf = (char*)sendbufs.c_str();

					X_out1 = X_out;
					Y_out1 = Y_out;
					Z_out1 = Z_out;
					Rx_out1 = Rx_out;
					Ry_out1 = Ry_out;
					Rz_out1 = Rz_out;
				}
				else {
					movejss.str("");
					movejss << "movej(p[" << X_out << "," << Y_out << "," <<
Z_out << "," << Rx_out << "," << Ry_out << "," << Rz_out << "], a=" << a << ", v=" <<
v << ", r=" << r << ")\n";
					sendstr = movejss.str();
					//cout << sendstr << "\n";
				}
```

```
			}
			else if (M1_OOV == TRUE || M2_OOV == TRUE || MinMove == TRUE) {
				if (step_motion == FALSE) {
					sendbufss.str("");
					sendbufss << "(0," << X_out1 << "," << Y_out1 << "," <<
Z_out1 << "," << Rx_out1 << "," << Ry_out1 << "," << Rz_out1 << "," << a << "," << v
<< "," << r << ")";
					sendbufs = sendbufss.str();
					sendbuf = (char*)sendbufs.c_str();
				}
				else {
					sendstr = "";
				}
				//if (M1_OOV == TRUE) { printf("M1: Izvan mjernog volumena.\n"); }
				//if (M2_OOV == TRUE) { printf("M2: Izvan mjernog volumena.\n"); }
				//if (MinMove == TRUE) { printf("Minimalni pomak.\n"); }
			}
		} while (tracking == TRUE);

		goto start;

		return;
}
```

A.5. input.cpp

```
#include <iostream>
#include <windows.h>
#include <string>
#include <Eigen/Dense>
#include <Eigen/Geometry>

using namespace std;
using namespace Eigen;

int sleep1, sleep2;
double a, v, r;
double MinDist;
extern bool tracking;
double x_offset, y_offset, z_offset;
double roll, pitch, yaw;
extern Matrix4d R_rotM2;
stringstream stopjss;
string stopj;
string input_selection;
double pi = 3.14159265;
double roll_deg, pitch_deg, yaw_deg;

void input(void *P) {
reset:
	a = 1.4;
	v = 1.05;
	r = 0;
	MinDist = 5;
	x_offset = 0;
	y_offset = 0;
	z_offset = 0;
	roll_deg = 0;
	pitch_deg = 0;
	yaw_deg = 0;
	roll = 0;
	pitch = 0;
	yaw = 0;
	sleep1 = 500;
	sleep2 = 500;

	stopjss.str("");
	stopjss << "stopj(" << a << ")\n";
	stopj = stopjss.str();

start:
	do {} while (tracking == FALSE);

	cout << "1. Promjeni akceleraciju (" << a << ")\n";
	cout << "2. Promjeni brzinu (" << v << ")\n";
	cout << "3. Promjeni radijus (" << r << ")\n";
	cout << "4. Promjeni minimalni pomak (" << MinDist << ")\n";
	cout << "5. Promjeni x pomak (" << x_offset << ")\n";
	cout << "6. Promjeni y pomak (" << y_offset << ")\n";
	cout << "7. Promjeni z pomak (" << z_offset << ")\n";
	cout << "8. Promjeni rotaciju oko osi X (" << roll_deg << ")\n";
	cout << "9. Promjeni rotaciju oko osi Y (" << pitch_deg << ")\n";
	cout << "10. Promjeni rotaciju oko osi Z (" << yaw_deg << ")\n";
```

```
	cout << "11. Promjeni vrijeme gibanja (" << sleep1 << ")\n";
	cout << "12. Promjeni vrijeme zaustavljanja (" << sleep2 << ")\n";
	cout << "13. Vrati na zadano\n";
	cout << "14. Zaustavi pracenje\n";
	cin >> input_selection;

start1:
	if (input_selection.compare("1") == 0) {
		cout << "Unesi akceleraciju (" << a << ") (zadano=1.4): "; cin >> a;
		stopjss.str("");
		stopjss << "stopj(" << a << ")\n";
		stopj = stopjss.str();
		cout << "\n";
		goto start;
	}
	else if (input_selection.compare("2") == 0) {
		cout << "Unesi brzinu (" << v << ") (zadano=1.05): "; cin >> v;
		cout << "\n";
		goto start;
	}
	else if (input_selection.compare("3") == 0) {
		cout << "Unesi radijus (" << r << ") (zadano=0): "; cin >> r;
		cout << "\n";
		goto start;
	}
	else if (input_selection.compare("4") == 0) {
		cout << "Unesi minimalni pomak (" << MinDist << ") (zadano=5): "; cin >>
MinDist;
		cout << "\n";
		goto start;
	}
	else if (input_selection.compare("5") == 0) {
		cout << "Unesi x pomak (" << x_offset << ") (zadano=0): "; cin >>
x_offset;
		cout << "\n";
		goto start;
	}
	else if (input_selection.compare("6") == 0) {
		cout << "Unesi y pomak(" << y_offset << ") (zadano=0): "; cin >>
y_offset;
		cout << "\n";
		goto start;
	}
	else if (input_selection.compare("7") == 0) {
		cout << "Unesi z pomak (" << z_offset << ") (zadano=0): "; cin >>
z_offset;
		cout << "\n";
		goto start;
	}
	else if (input_selection.compare("8") == 0) {
		cout << "Unesi rotaciju oko osi X (" << roll_deg << ") (zadano=0): "; cin
>> roll_deg;
		roll = -roll_deg*(pi / 180);
		cout << "\nR_M2=\n" << R_rotM2 << "\n\n";
		goto start;
	}
	else if (input_selection.compare("9") == 0) {
		cout << "Unesi rotaciju oko osi Y (" << pitch_deg << ") (zadano=0): ";
cin >> pitch_deg;
		pitch = -pitch_deg*(pi / 180);
```

```
			cout << "\nR_M2=\n" << R_rotM2 << "\n\n";
			goto start;
		}
		else if (input_selection.compare("10") == 0) {
			cout << "Unesi rotaciju oko osi Z (" << yaw_deg << ") (zadano=0): "; cin
>> yaw_deg;
			yaw = -yaw_deg*(pi / 180);
			cout << "\nR_M2=\n" << R_rotM2 << "\n\n";
			goto start;
		}
		else if (input_selection.compare("11") == 0) {
			cout << "Unesi vrijeme gibanja(" << sleep1 << ") (zadano=500): "; cin >>
sleep1;
			cout << "\n";
			goto start;
		}
		else if (input_selection.compare("12") == 0) {
			cout << "Unesi vrijeme zaustavljanja(" << sleep2 << ") (zadano=500): ";
cin >> sleep2;
			cout << "\n";
			goto start;
		}
		else if (input_selection.compare("13") == 0) {
			cout << "Postavljene zadane vrijednosti.\n\n";
			goto reset;
		}
		else if (input_selection.compare("14") == 0) {
			tracking = FALSE;
			cout << "\nPracenje je zaustavljeno.\n\n";
		}
		else if (input_selection.compare("") == 0) {
			goto start1;
		}
		else {
			cout << "\nKrivi unos.\n\n";
			goto start1;
		}

		goto start;

		return;
}
```

A.6. client.cpp

```
#define WIN32_LEAN_AND_MEAN

#include <WinSock2.h>
#include <WS2tcpip.h>
#include <iostream>

// link with Ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

#define DEFAULT_PORT "30001"
#define DEFAULT_BUFFER_LENGTH     512

using namespace std;
extern string sendstr, stopj;
extern int sleep1, sleep2;
extern bool sending, step_motion;

class Client {
public:
	Client(char* servername) {
		szServerName = "192.168.0.8"; //Robot IP
		ConnectSocket = INVALID_SOCKET;
	}

	bool Start() {
		WSADATA wsaData;

		// Initialize Winsock
		int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
		if (iResult != 0) {
			printf("client: WSAStartup failed: %d\n", iResult);
			return false;
		}

		struct addrinfo     *result = NULL,
			*ptr = NULL,
			hints;

		ZeroMemory(&hints, sizeof(hints));
		hints.ai_family = AF_UNSPEC;
		hints.ai_socktype = SOCK_STREAM;
		hints.ai_protocol = IPPROTO_TCP;

		// Resolve the server address and port
		iResult = getaddrinfo(szServerName, DEFAULT_PORT, &hints, &result);
		if (iResult != 0) {
			printf("client: getaddrinfo failed: %d\n", iResult);
			WSACleanup();
			return false;
		}

		ptr = result;

		// Create a SOCKET for connecting to server
		ConnectSocket = socket(ptr->ai_family, ptr->ai_socktype, ptr-
>ai_protocol);
```

```
		if (ConnectSocket == INVALID_SOCKET) {
			printf("client: Error at socket(): %d\n", WSAGetLastError());
			freeaddrinfo(result);
			WSACleanup();
			return false;
		}

		// Connect to server
		iResult = connect(ConnectSocket, ptr->ai_addr, (int)ptr->ai_addrlen);

		if (iResult == SOCKET_ERROR) {
			closesocket(ConnectSocket);
			ConnectSocket = INVALID_SOCKET;
		}

		freeaddrinfo(result);

		if (ConnectSocket == INVALID_SOCKET) {
			printf("client: Unable to connect to server!\n");
			WSACleanup();
			return false;
		}

		return true;
	};

	// Free the resouces
	void Stop() {
		int iResult = shutdown(ConnectSocket, SD_SEND);

		if (iResult == SOCKET_ERROR) {
			printf("client: shutdown failed: %d\n", WSAGetLastError());
		}

		closesocket(ConnectSocket);
		WSACleanup();
	};

	// Send message to server
	bool Send(char* szMsg) {
		int iResult = send(ConnectSocket, szMsg, strlen(szMsg), 0);

		if (iResult == SOCKET_ERROR) {
			printf("client: send failed: %d\n", WSAGetLastError());
			Stop();
			return false;
		}

		return true;
	};

	// Receive message from server
	bool Recv() {
		char recvbuf[DEFAULT_BUFFER_LENGTH];
		int iResult = recv(ConnectSocket, recvbuf, DEFAULT_BUFFER_LENGTH, 0);

		if (iResult > 0) {
			char msg[DEFAULT_BUFFER_LENGTH];
			memset(&msg, 0, sizeof(msg));
			strncpy_s(msg, recvbuf, iResult);
```

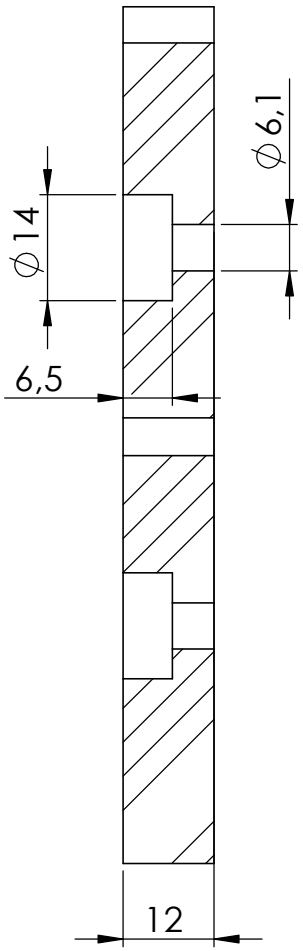
```
                        printf("client: Received: %s\n", msg);
                        cout << "client: ConnectSocket=" << ConnectSocket;
                        return true;
                }

                return false;
        }

private:
        char* szServerName;
        SOCKET ConnectSocket;
};

void client(void *P) {
        Client client("127.0.0.1");

        if (!client.Start())
                return;

        while (true) {
                do {} while (sending == FALSE);
                client.Send((char*)sendstr.c_str());
                Sleep(sleep1);
                if (step_motion == TRUE) {
                        client.Send((char*)stopj.c_str());
                        Sleep(sleep2);
                }
        }

        client.Stop();

        getchar();
        return;
}
```

A.7. server_UR.cpp

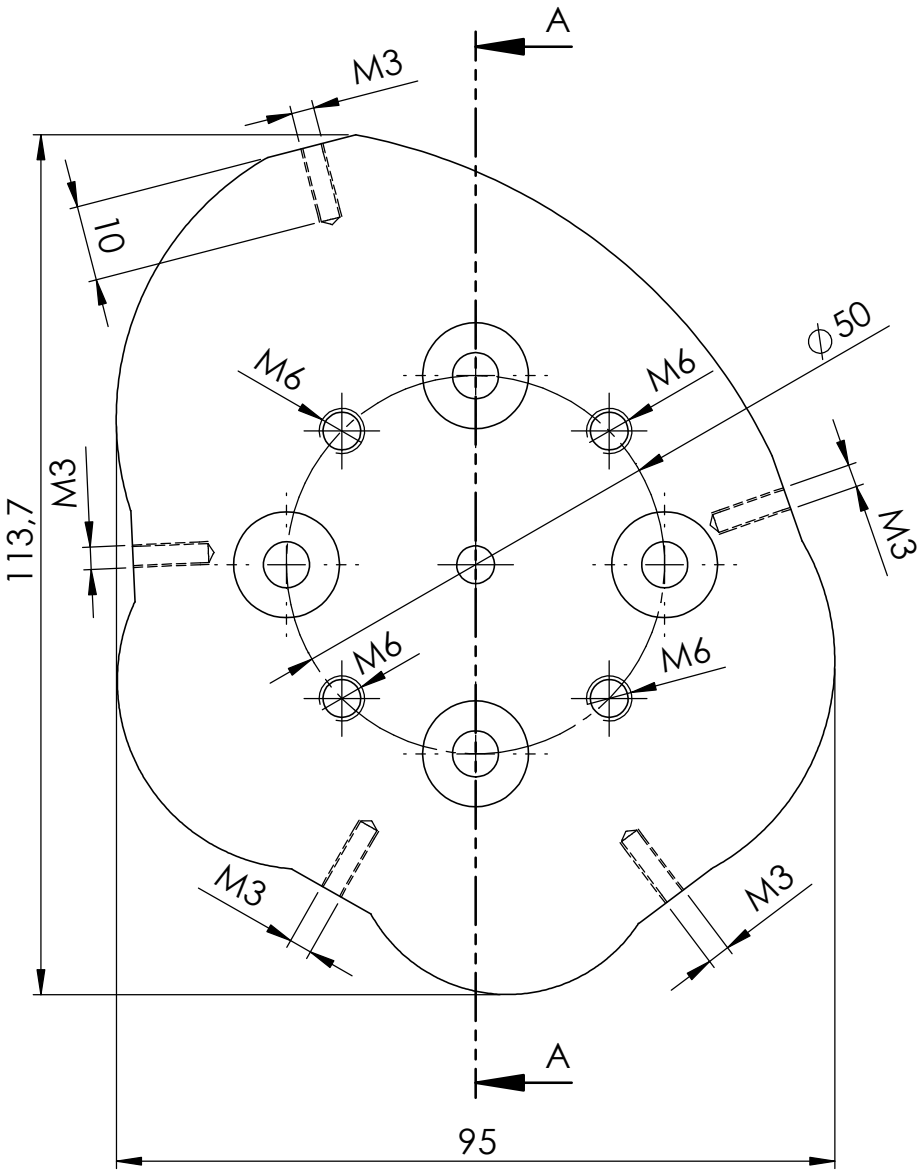
```
#define WIN32_LEAN_AND_MEAN

#include <WinSock2.h>
#include <WS2tcpip.h>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <string>
#include <sstream>

// link with Ws2_32.lib
#pragma comment(lib, "Ws2_32.lib")

#define DEFAULT_PORT "30000"
#define DEFAULT_BUFFER_LENGTH     512

using namespace std;
extern char* sendbuf;

void server_UR(void *P) {
	WSADATA wsaData;

	// Initialize Winsock
	int iResult = WSAStartup(MAKEWORD(2, 2), &wsaData);
	if (iResult != 0) {
		printf("server_UR: WSAStartup failed: %d\n", iResult);
		return;
	}

	struct addrinfo	*result = NULL,
		hints;

	ZeroMemory(&hints, sizeof(hints));
	hints.ai_family = AF_INET;		// Internet address family is unspecified so
that either an IPv6 or IPv4 address can be returned
	hints.ai_socktype = SOCK_STREAM;  // Requests the socket type to be a stream
socket for the TCP protocol
	hints.ai_protocol = IPPROTO_TCP;
	hints.ai_flags = AI_PASSIVE;

	// Resolve the local address and port to be used by the server
	iResult = getaddrinfo(NULL, DEFAULT_PORT, &hints, &result);
	if (iResult != 0) {
		printf("server_UR: getaddrinfo failed: %d\n", iResult);
		WSACleanup();
		return;
	}

	SOCKET ListenSocket = INVALID_SOCKET;

	// Create a SOCKET for the server to listen for client connections
	ListenSocket = socket(result->ai_family, result->ai_socktype, result-
>ai_protocol);

	if (ListenSocket == INVALID_SOCKET) {
		printf("server_UR: Error at socket(): %d\n", WSAGetLastError());
		freeaddrinfo(result);
```

```
		WSACleanup();
		return;
	}

	// Setup the TCP listening socket
	iResult = bind(ListenSocket, result->ai_addr, (int)result->ai_addrlen);

	if (iResult == SOCKET_ERROR) {
		printf("server_UR: bind failed: %d", WSAGetLastError());
		freeaddrinfo(result);
		closesocket(ListenSocket);
		WSACleanup();
		return;
	}

	freeaddrinfo(result);

	// To listen on a socket
	if (listen(ListenSocket, SOMAXCONN) == SOCKET_ERROR) {
		printf("server_UR: listen failed: %d\n", WSAGetLastError());
		closesocket(ListenSocket);
		WSACleanup();
		return;
	}

	SOCKET ClientSocket;

	ClientSocket = INVALID_SOCKET;

	// Accept a client socket
	ClientSocket = accept(ListenSocket, NULL, NULL);

	if (ClientSocket == INVALID_SOCKET) {
		printf("server_UR: accept failed: %d\n", WSAGetLastError());
		closesocket(ListenSocket);
		WSACleanup();
		return;
	}

	char recvbuf[DEFAULT_BUFFER_LENGTH];
	int iSendResult;

	// reveice until the client shutdown the connection
	do {
		iResult = recv(ClientSocket, recvbuf, DEFAULT_BUFFER_LENGTH, 0);

		if (iResult > 0) {
			iSendResult = send(ClientSocket, sendbuf, (int)strlen(sendbuf),
0);
			//printf("Data sent: %s\n", sendbuf);

			if (iSendResult == SOCKET_ERROR) {
			printf("server_UR: send failed: %d\n", WSAGetLastError());
			closesocket(ClientSocket);
			WSACleanup();
			return;
			}

			//printf("Bytes sent: %ld\n", iSendResult);
		}
```

```
		else if (iResult == 0) {
			printf("server_UR: Connection closed\n");
		}
		else {
			printf("server_UR: recv failed: %d\n", WSAGetLastError());
			closesocket(ClientSocket);
			WSACleanup();
			return;
		}
	} while (iResult > 0);

	// Free the resouces
	closesocket(ListenSocket);
	WSACleanup();

	getchar();
	return;
}
```



Presjek A-A



	Datum	Ime i prezime	Potpis	FSB Zagreb
Projektirao	20.11.2015.	Nikola Kirić		
Razradio	20.11.2015.	Nikola Kirić		
Crtao	20.11.2015.	Nikola Kirić		
Pregledao	20.11.2015.	Filip Šuligoj		

Objekt: Alat M1		Objekt broj:
		R. N. broj:
Napomena:		Kopija
Materijal: S235JRG2	Masa: 0.7 kg	

Mjerilo originala 1:1	Naziv: Alat M1	Pozicija: 1	Format: A4
			Listova: 1
	Crtež broj: M1-001-2015		List: 1

Design by CADLab