Integracija beskontaktnog 3D mjernog uređaja za mjerenje hrapavosti

Čuvalo, Matea

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Matea Čuvalo

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:
Doc. dr. sc. Miho Klaić, dipl. ing.

Student:
Matea Čuvalo

Zagreb, 2023.

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

Zahvaljujem mentoru doc. dr. sc. Mihi Klaiću te prof. dr. sc. Tomislavu Staroveškom na ustupljenoj opremi, stručnoj pomoći i savjetima tokom izrade diplomskg rada.

Također zahvaljujem svojoj obitelji, prijateljima i kolegama koji su me podržavali i pomagali mi tijekom studiranja te prilikom izrade ovog diplomskog rada.

Matea Čuvalo

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

Sveučilište u Zagrebu Fakultet strojarstva i brodogradnje	
Datum:	Prilog:
Klasa: 602 – 04 / 23 – 6 / 1	
Ur. broj: 15 - 1703 - 23 -	

DIPLOMSKI ZADATAK

Student: **MATEA ČUVALO** Mat. br.: 0035208162

Naslov rada na hrvatskom jeziku: **Integracija beskontaktnog 3D mjernog uređaja za mjerenje hrapavosti**

Naslov rada na engleskom jeziku: **Integration of optical 3D measurement instrument for roughness measuring**

Opis zadatka:

Na Katedri za alatne strojeve Fakulteta strojarstva i brodogradnje u tijeku je provođenje projekta ARCOPS čiji je cilj razvoj sustava za izravni i posredni nadzor procesa brušenja i poliranja primjenom industrijskih robota. U sklopu provedenih projektnih aktivnosti nabavljen je industrijski robot IRB4600 (ABB, Švicarska) koji je opremljen s optičkim mjernim uređajem tipa IF-SensorR25 (Alicona Imaging GmbH, Austrija) s ciljem realizacije sustava za izravno određivanje stupnja istrošenosti alata i kvalitete obrađene površine. Shodno navedenom u ovom diplomskom radu je potrebno:
1. Opisati ispitni postav.
2. Istražiti mogućnosti automatskog vođenja predmetnog uređaja kroz dostupnu programsku podršku i pripadajuće biblioteke.
3. Aktivno sudjelovati u izradi programske podrške kojom će se omogućiti upravljanje predmetnim mjernim uređajem izravno iz upravljačkog sustava robota putem računalne mreže (na osnovi TCP/IP protokola).
4. Napraviti probna mjerenja na zadanoj ispitnoj površini.
5. Komentirati dobivene rezultate i dati zaključke.

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

Zadatak zadan: 19. siječnja 2023.

Rok predaje rada: 23. ožujka 2023.

Predviđeni datum obrane: 27. ožujka do 31. ožujka 2023.

Zadatak zadao: doc. dr. sc. Miho Klaić

Predsjednica Povjerenstva: prof. dr. sc. Biserka Runje

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Mjerna jedinica	Opis oznake
X	mm	Pozicija u X osi
Y	mm	Pozicija u Y osi
Z	mm	Pozicija u Z osi

POPIS KRATICA

Kratica	**Opis**
3D	Trodimenzionalan
LED	*Light-Emitting Diode* – Svjetleća dioda
TCP	*Transmission Control Protocol* – Transportni protokol
IP	*Internet Protocol* – Interenet Protokol
TCP	*Tool Central Point* – Vrh alata robota
GUI	*Graphical User Interface* – Grafičko korisničko sučelje
RI	*Remoting Interface* – Udaljeno sučelje

SAŽETAK

U ovom diplomskom radu opisan je sustav integracije beskontatnog mjernog uređaja Alicona IF-SensorR25 s industrijskim robotom ABB IRB4600. Dan je uvid i objašnjenje programskog koda razvijenog za realizaciju komunikacije pomoću računalne mreže temeljenje na TCP/IP protokolu između mjernog uređaja i upravljačkog sustava robota. Programska podrška koja se izvršava na mjernom računalu pisana je u programskom jeziku Python, dok je kod na upravljačkoj jedinici robota napisan u programskom jeziku RAPID. Proveden je proces mjerenja te su dobiveni mjereni rezultati prokomentirani u zaključnom dijelu rada.

Ključne riječi: Alicona IF-SensorR25, integracija, ABB IRB4600 robot

SUMMARY

This thesis describes the integration process of 3D optical measurement instrument Alicona IF-SensorR25 with the ABB IRB4600 industrial robot. Thesis also gives an overview and explanation of the program code developed for the communication between measuring system and the robot controller based on the TCP/IP protocol. The software on the measuring computer is written in the Python programming language, while the code on the ABB controller is written in the RAPID programming language. The measurement process was performed, and the obtained results were commented on the final part of the paper.

Key words: Alicona IF-SensorR25, integration, ABB IRB4600 robot

1. UVOD

Trošenje alata je nezaobilazan fenomen koji je općenito prisutan kod svih postupaka obrade odvajanjem čestica. Proces trošenja izrazito je nelinearan, a dijelom i stohaističke prirode s izravnim utjecam na kvalitetu obrađene površine. Stoga je razvoj obradnih sustava dobrim dijelom usmjeren razvoju sustava za nadzor alata i obradnih procesa.

Direktnim metodama nadzora stanja obradnih procesa, značajke alata i obrađene površine dobivaju se izravnim mjerenjem, zbog čega je obradni proces potrebno često prekidati. Indirektne metode, za razliku od direktnih, mogu dati informaciju o stanju alata i površine u stvarnom vremenu tijekom obrade, koristeći različite vrste senzora kao što su senzori sila, jakosti struje ili snage pogonskih motora i vibracija, aukustične emisije i buke.

Česti prekidi u procesu, a time i nepotrebni troškovi, posljedica su prijevremenog mijenjanja brusnog alata pri procesu brušenja. Uslijed niza mehaničkih, toplinskih, kemijskih utjecaja prisutnih za vrijeme obrade dolazi do intenzivnog trošenja brusnog papira, što negativno utječe na kvalitetu obrađivane površine. Pravovremenom izmjenom brusnog alata postiže se bolja kvaliteta površine i ekonomičnost sustava. Stoga je cilj projekta ARCOPS („Autonomni robotski sustav za brušenje i karakterizaciju površina tankostjenih kompozitnih proizvoda“), koji se provodi na Katedri za alatne strojeve Fakulteta strojarstva i brodogradnje, razviti sustav za izravni i posredni nadzor procesa brušenja i poliranja primjenom industrijskih robota, čime bi se značajno podigla razina autonomnosti sustava.

Robotska ćelija (Slika 1) opremljena je sa više vrsta senzora (uređajem za preciznu regulaciju sile, senzorom vibracija, akustične emisije i buke, termovizijskom kamerom i beskontaktnim mjernim uređajem) pomoću kojih će se snimati signali sila, vibracija, akustične emisije i buke, toplinski trag brusnog alata na obrađenoj površini te stupanj istrošenosti brusnog alata i hrapavosti određene površine, kako bi se utvrdili parametri obrade pomoću kojih će se razviti model izravnog i posrednog nadzornog sustava.

Slika 1. Robotska ćelija ispitnog postava projekta ARCOPS

U cilju realizacije sustava za izravno određivanje stupnja istrošenosti alata i kvalitete površine u sklopu projektnih aktivnosti nabavljen je optički beskontaktni mjerni uređaj tipa IF-SensorR25 (Alicona Imaging GmbH, Austrija) te će u ovom diplomskom radu biti opisana njegova integracija s upravljačkim sustavom robota tipa IRB4600 (ABB, Švicarska).

U drugom poglavlju rada opisan je ispitni postav. Navedene su tehničke karakteristike i objašnjen je princip rada mjernog uređaja te je opisan robotski sustav. U trećem poglavlju sažeto je opisana računalna mreža temeljena na TCP/IP protokolu. Četvrto poglavlje opisuje aktivnosti koje su provedene u cilju integracije mjernog uređaja s upravljačkim sustavom robota i programski kod potreban za realizaciju komunikacije temeljene na TCP/IP protokolu. Na kraju rada provedena su probna mjerenja na zadanoj ispitnoj površini i prokomentirani su rezultati uz dani zaključak rada.

2. OPREMA ISPITNOG POSTAVA

Robotska ćelija ispitnog postava sastoji se od dva industrijska robota od kojih je prvi opremljen glavnim prigonom za brušenje dok je drugi opremljen beskontaktnim senzorom za mjerenje parametara hrapavosti. Mjerni uređaj IF-sensorR25 (zajedno s ATOS 5x skenerom) nalazi se unutar konstrukcije koja je ugrađena na prihvatnicu robota IRB4600 (Slika 2). Nakon izvođenja određenih zadataka uređaj se dovodi u komoru s predtlakom kako bi se zaštitili od prašine i ostalih vanjskih utjecaja koji su prisutni tijekom obrade.

U ovom poglavlju, prvo će se opisati mjerni uređaj i proces mjerena istim, zatim upravljački sustav robota za automatsko vođenje mjernog uređaja.

Slika 2. Prikaz mjernog uređaja montiranog na robot i spremnika

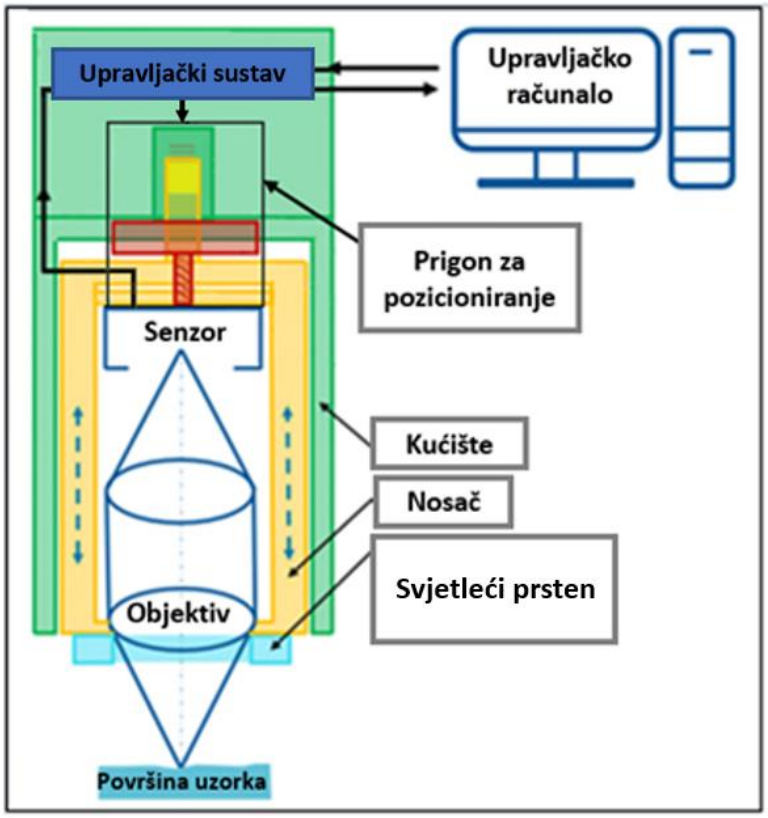
2.1. Optički mjerni uređaj Alicona IF-SensorR25

Alicona IF-SensorR25 (Slika 3) je beskontaktni optički 3D mjerni instrument za automatizirano mjerenje oblika i hrapavosti u proizvodnji. Pomoću uređaja mogu se mjeriti razne površinske karakteristike kao što su srednje aritmetičko odstupanje profila *Ra*, srednja visina neravnina *Rz*, odstupanje srednjeg korijena profila *Rq*, aritmetičko odstupanje površine *Sa*, srednja visina neravnina površine *Sz*, odstupanje srednjeg korijena površine *Sq* i druge. Senzor je moguće integrirati u proizvodnu liniju te proizvodne sustave. [1]

Slika 3. Alicona IF-SensorR25 mjerni uređaj [1]

2.1.1. Princip rada i tehničke specifikacije mjernog uređaja

Princip rada uređaja Alicona IF-SensorR25 temelji se na tehnologiji koja pomicanjem žarišta utvrđuje dubinu površine (engl. *focus variation/depth from focus*). Metoda digitalizacije (3D skeniranja) ispitne površine provodi se pomicanjem optičkog sustava u smjeru paralelnim s optičkom osi optičkog sustava pomoću preciznog prigona za pozicioniranje. Pojednostavljena struktura uobičajenog uređaja koji se temelji na navedenoj tehnologiji prikazana je slikom 4.



Slika 4. Shema FV uređaja [2]

Pri procesu skeniranja koriste se mikroskopski objektivi s fiksnom žarišnom duljinom i ograničenom dubinom vidnog polja. Pomicanjem optičkog sustava različiti dijelovi površine dovode se u žarište, te senzor snima slike u razmacima definiranim vertikalnom rezolucijom, odnosno dubinskom oštrinom korištenog objektiva. Nakon skeniranja primjenjuju se algoritmi za obradu snimljenog žarišnog niza (engl. *focal stack*).slika kako bi se dobio 3D topografski skup podataka na temelju promjene fokusa. Algoritam mjernog uređaja automatski pretvara sva pojedinačna mjerenja žarišnog niza slika u zajednički koordinatni sustav te stvara oblak

točaka površine objekta visoke rezolucije. Rezolucija je definirana vidnim poljem, optičkom rezolucijom objektiva i senzora te gustoćom snimljenih mjernih točaka, odnosno količinom površinskih detalja.

Uređaj pri pozicioniranju/fokusiranju koristi lasersku zraku valne duljine 635 nm, a za osvjetljavanje površine koristi se LED svjetleći prsten (engl. *ring light*), koji se sastoji od 24 segmenta te se nalazi na obodu objektiva. [1] Pomoću svjetlećeg prstena mogu se osvijetliti površine s izraženim nagibom. Smanjenje refleksije površine moguće je upotrebom polarizatora koji se montira na objektiv.

Uređajem je moguće mjeriti površinske karakteristike u mikrometarskom području. Površine kojima se fokus tijekom skeniranja ne razlikuje, kao što je transparentni uzorak, nije moguće mjeriti. [1] Optički sustav može biti opremljen objektivima različitih uvećanja omogućujući mjerenja s različitom rezolucijom i radnom području (vidnim poljem). Korištenjem objektiva s većim povećanjem smanjuje se dubina vidnog polja što izravno utječe na poboljšanje vertikalne rezolucije, no ujedno se smanjuje i lateralno mjerno područje. U ovom radu korišten je objektiv uvećanja 10x. Tehničke specifikacije mjernog uređaja prikazane su Tablicom 1.

Tablica 1. Tehničke specifikacije Alicona IF-SensorR25 mjernog uređaja [1]

Raspon pozicioniranja (Z)		25 mm (mot.)							
Uvećanje objektiva		10x	20x	50x	2xSX	5xAX	10xAX	20xAX	50xSX
Radna udaljenost	mm	17.5	16	10.1	34	34	33.5	20	13
Lateralno mjerno područje (X,Y)	mm	2	1	0.4	10	3.61	2	1	0.4
(X x Y)	mm^2	4	1	0.16	100	13.03	4	1	0.16
Vertikalna rezolucija	nm	100	50	20	3500	460	130	70	45
Točnost koraka (1mm)	%	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Min. mjerljiva hrapavost (Ra)	µm	0.3	0.15	0.08	-	-	0.45	0.25	0.15
Min. mjerljiva hrapavost (Sa)	µm	0.15	0.075	0.05	-	-	0.25	0.1	0.08
Min. mjerljivi polumjer	µm	5	3	2	20	10	5	3	2

2.1.2. Proces mjerenja uređajem Alicona IF-sensor-R25

Proces mjerenja izvršava se korištenjem programskog paketa instaliranog na računalu. Operativni programski paket mjernog uređaja naziva *MeasureSuite* sastoji se od više modula. Služi za obradu izmjerenih 3D skupova podataka i za izvođenje različitih mjerenja nad tim podacima (npr. izračun parametara hrapavosti, parametrara oblika, parametara trošenja). [3]

Modul korišten u ovom radu zove se *LaboratoryMeasurementModule*. Koristi se za pokretanje mjernog ciklusa i za upravljanje uređajem putem grafičkog korisničkog sučelja (GUI) ili udaljenog upravljačkog sučelja (RI). Predmetno sučelje omogućuje upravljanje mjernim uređajem putem drugog programa putrem računalne mreže. Program se može pokrenuti na istom računalu ili s drugog, udaljenog računala. [3]

Unutar modula potrebno je definirati uvećanje objektiva, gornju i donju granicu skeniranja, namjestiti odgovarajuću svjetlinu (engl. *brightness*) i kontrast (engl. *contrast*) (odnosno ekspoziciju senzora) te vertikalnu i lateralnu rezoluciju. Određivanje svjetline i kontrasta olakšano je upotrebom histograma. Histogram je dijagramski prikaz funkcije intenziteta svjetline koja daje broj piksela za svaku vrijednost svjetline u promatranom području.

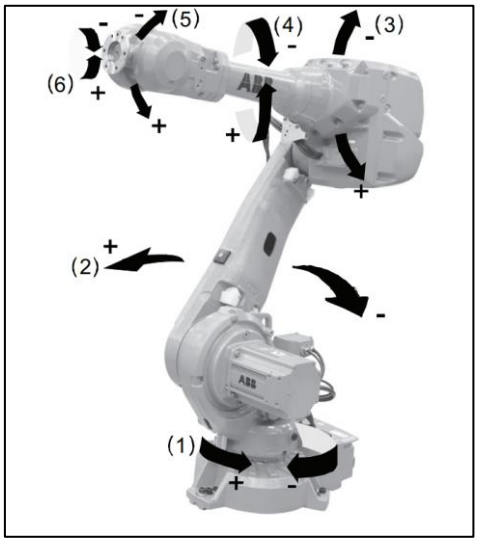
Također je moguće i upravljati odnosno uključiti ili isključiti polarizator (ukoliko je isti ugrađen) te segmente svjetlećeg prstena. Nakon definiranja svih potrebnih parametara mjerenje se izvršava naredbom *start single measurement*. Rezultati se nakon mjerenja pohranjuju u bazu podataka.

Obzirom da je na ovaj način moguće skenirati površinu veličine $2 \times 2\ \mathrm{mm}^2$ (korištenjem objektiva uvećanja 10xAX) potrebno je razviti sustav kojim će se omogućiti automatski sustav skeniranja površine. Robotski sustav kojim je ostvareno vođenje mjernog uređaja opisan je dalje u radu.

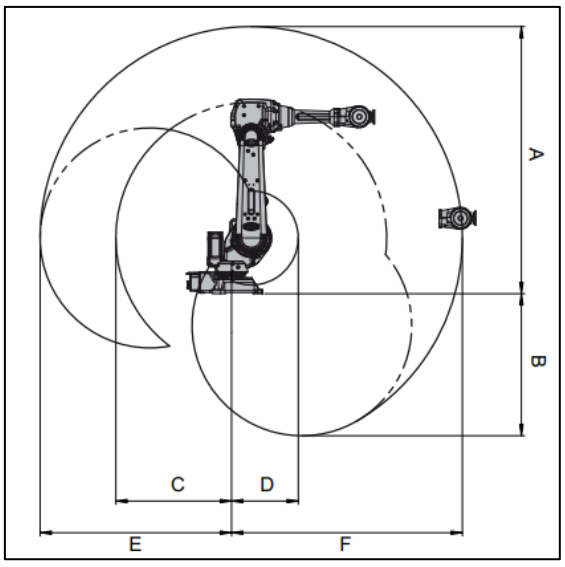
2.2. Robotski sustav

2.2.1. Robot IRB 4600-40/2.55

Automatsko vođenje mjernog uređaja je ostvareno pomoću industrijskog robota tvrtke ABB, model IRB 4600-40/2.55. Model IRB 4600-40/2.55 je 6-osni robot (Slika 5) čija nosivost (ukupna masa koju robot može prenijeti) iznosi 40kg, a dohvat 2.55m. Radni prostor robota je prikazan slikom 6, gdje je A = 2872 mm, B = 1735mm, C = 1393mm, D = 680mm, E = 2202 mm, F = 2552mm. [4]



Slika 5. Prikaz stupnjeva slobode gibanja robota model IRB4600 [4]



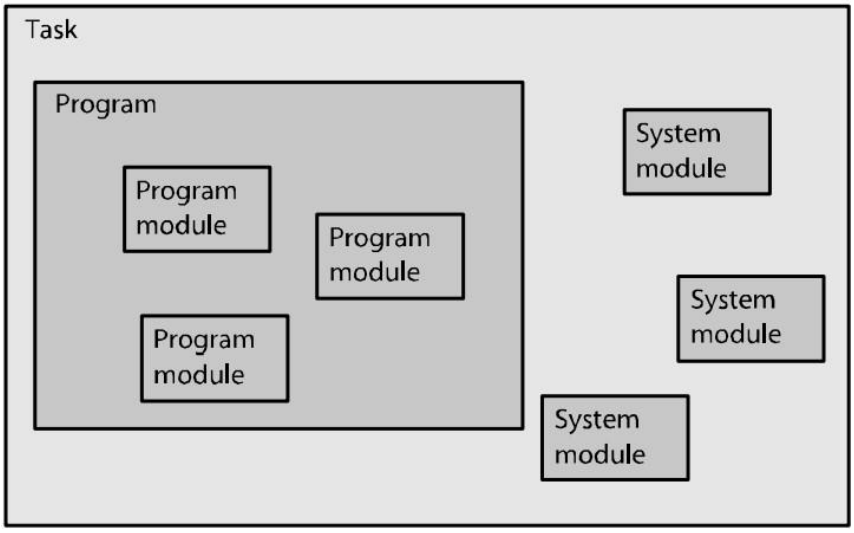
Slika 6. Prikaz radnog prostora robota IRB4600 [4]

2.2.2. Programski jezik RAPID

Upravljačka jedinica robota IRC5 je spojena s operatorskim panelom (Slika 7) preko kojeg se vrši upravljanje robotom. Programiranje robota je moguće ostvariti pisanjem koda izravno u operatorski panel ili preko računala korištenjem RobotStudio softvera.

Slika 7. Upravljačka jedinica IRC5 spojena sa operatorskim panelom

Upravljačke jedinice ABB robota koriste RAPID programski jezik za vođenje robota. Programski kod u RAPID-u je definiran unutar modula koji mogu biti sistemski ili programski (Slika 8). Sistemski modul je modul koji je uvijek prisutan u programskoj memoriji, a najčešće sadrži funkcije, procedure i varijable (tipove podataka) specifično prilagođene potrebama ćelije. Programski moduli predstavljaju programe za neke određene procese, primjerice skeniranje, lakiranje, zavarivanje i slično. Moduli se pridružuju zadacima (engl. *Task*), svaki ABB kontroler ima minimalno jedan zadatak koji se ujedno koristi za gibanje robota. [5] Zavisno o licencama, upravljački sustav ABB robota (IRC5) podržava i više zadataka koji se mogu izvršavati paralelno.



Slika 8. Struktura modula unutar programskog jezika RAPID [6]

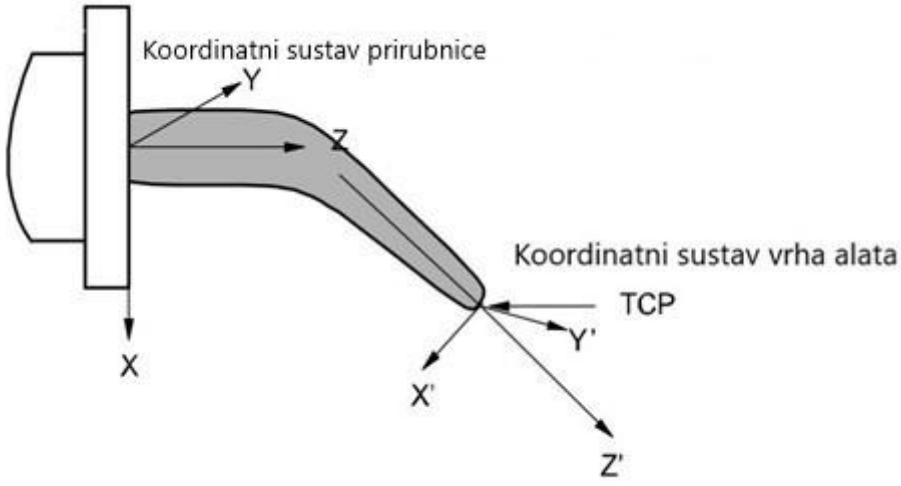
Unutar modula se definiraju rutine, odnosno procedure i funkcije. Funkcije za razliku od procedura vraćaju vrijednost/podatak određenog tipa nakon izvršenja koji se može pridružiti određenoj varijabli. Procedure se definiraju izrazom *PROC*, a funkcije sa izrazom *FUNC*. [5]

Osim uobičajenih varijabli definiranih izrazom *VAR*, u programskom jeziku RAPID moguće je definirati konstante izrazom *CONST* i trajne varijable (engl. *persistent variable*) izrazom *PERS*. Trajne varijable trajno spremaju vrijednosti zadnjeg unosa i nakon zaustavljanja i ponovnog pokretanja programa. Konstantama se dodijeljena vrijednost ne može mijenjati tijekom izvođenja programa. Varijablama je potrebno definirati i vrstu podataka. Vrste podataka korištene u ovom radu su: numerički podaci (definirani izrazom *num*) koji mogu biti i cijeli i decimalni brojevi, tekstualni niz (definirani izrazom *string*) s najviše 80 znakova i logičke varijable (definirani izrazom *bool*) koje mogu imati vrijednosti istinito (*TRUE*) ili neistinito (*FALSE*). [5]

Gibanje središta alata robota može biti linearno ili kružno. Naredbom *MoveL* izvršava se linearno gibanje, odnosno središte alata giba se po ravnoj liniji od početne do završne točke. *MoveJ* naredba izvršava gibanje središta alata od točke do točke u koordinatnom sustavu obratka (engl. *work object*). Sve osi završavaju s gibanjem u isto vrijeme, najkraćom putanjom konstantnom brzinom s obzirom na kutove zakreta zglobova. [6]

2.2.3. Definiranje alata

Podaci alata spremaju se u trajne varijable i definirani su tipom podatka *tooldata*. Podaci kojima se definira *tooldata* su *robhold*, *tframe* i *tload*. Kada je alat montiran na robot, vrijednost *robhold* je potrebno postaviti u vrijednost *TRUE*. *Tframe* definira poziciju i orijentaciju koordinatnog sustava alata u koordinatnom sustavu prirubnice robota (engl. *wrist coordinate system*) (Slika 8). Masa i težište alata te opterećenje preko kojeg upravljački sustav robota prilagođava pokretanje zglobova robota definirani su podatkom *tload*. [7] Definirani alat skenera spremljen je kao globalna varijabla u sistemskom modulu. Ovako spremljena varijabla omogućuje korištenje podatka alata u bilo kojem drugom modulu.

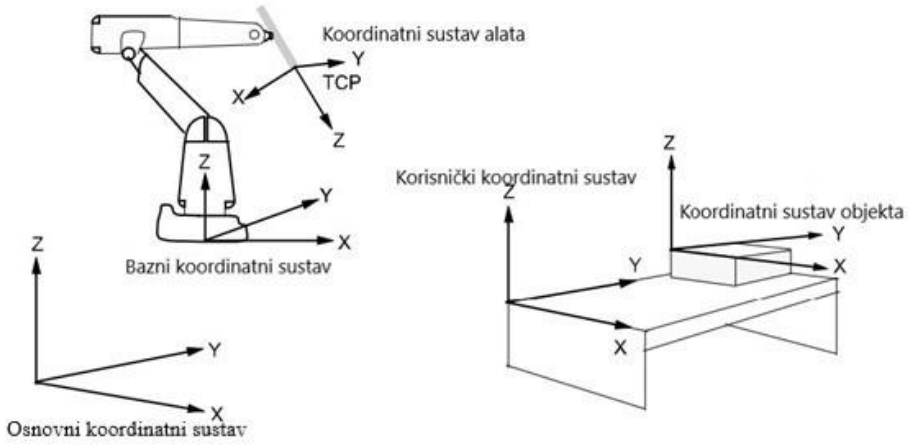


Slika 9. Koordinatni sustav vrha alata [6]

2.2.4. Koordinatni sustavi

Prilikom izrade programa za ABB robote potrebno je poznavati sljedeće koordinatne sustave (Slika 10):

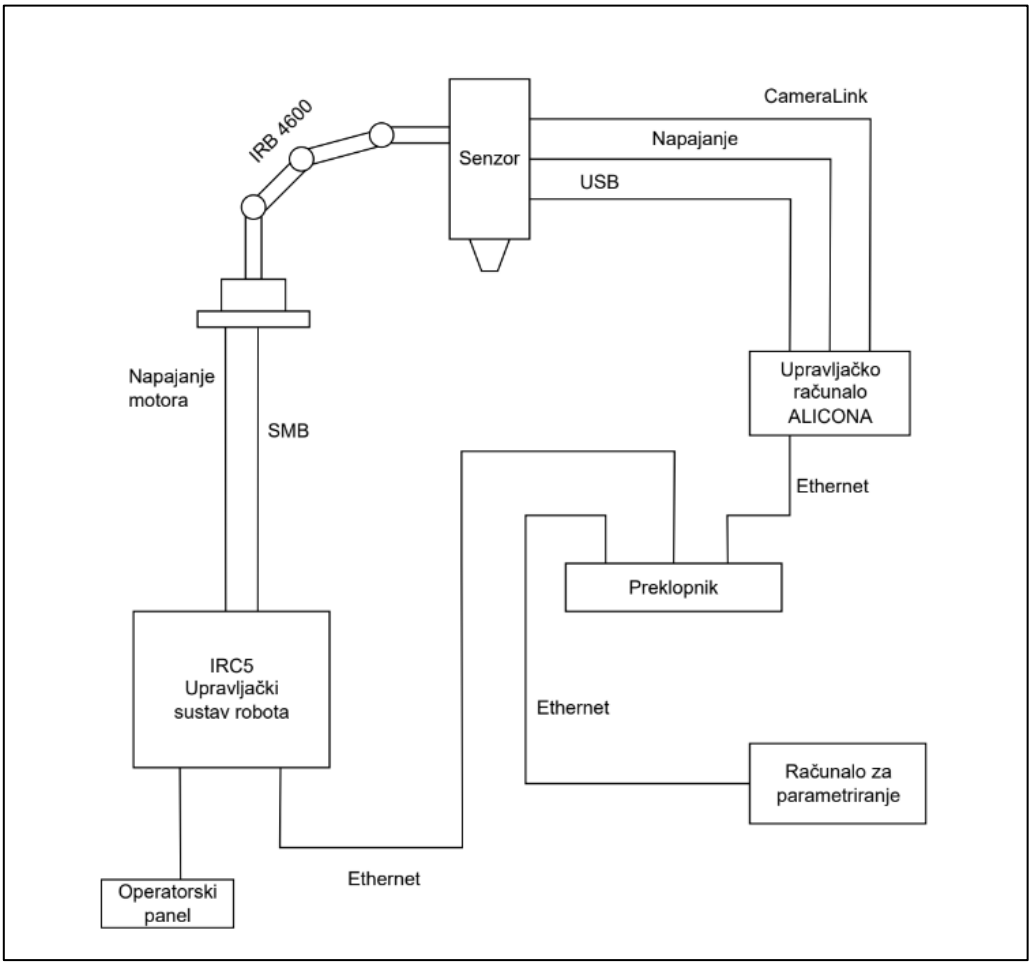
- Osnovni koordinatni sustav (engl. *World coordinate system*): definira cijelu robotsku ćeliju, svi ostali koordinatni sustavi definirani su u odnosu na ovaj koordinatni sustav
- Bazni koordinatni sustav (engl. *Base coordinate system*): smješten je, odnosno fiksiran za bazu robota. Njegova se pozicija i orijentacija određuje u odnosu na osnovni koordinatni sustav
- Koordinatni sustav alata TCP (engl. *Tool Center Point*): definira središnju točku alata. Za svaki robot moguće je definirati više koordinatnih sustava alata. Definira se u odnosu na bazni koordinatni sustav.
- Koordinatni sustav objekta (engl. *Object coordinate system*): pozicija i orijentacija definirana mu je u korisničkom koordinatnom sustavu koji je određen s obzirom na osnovni koordinatni sustav. Ovim koordinatnim sustavom omogućena je brza prilagodba cijelog robotskog programa na novi položaj objekta
- Korisnički koordinatni sustav (engl. *User coordinate system*): u ovom koordinatnom sustavu određuje se pozicija i orijentacija koordinatnog sustava objekta, a definiran je u odnosu na koordinatni sustav objekta
-



Slika 10. Koordinatni sustavi [5]

2.3. Povezivanje opreme ispitnog postava

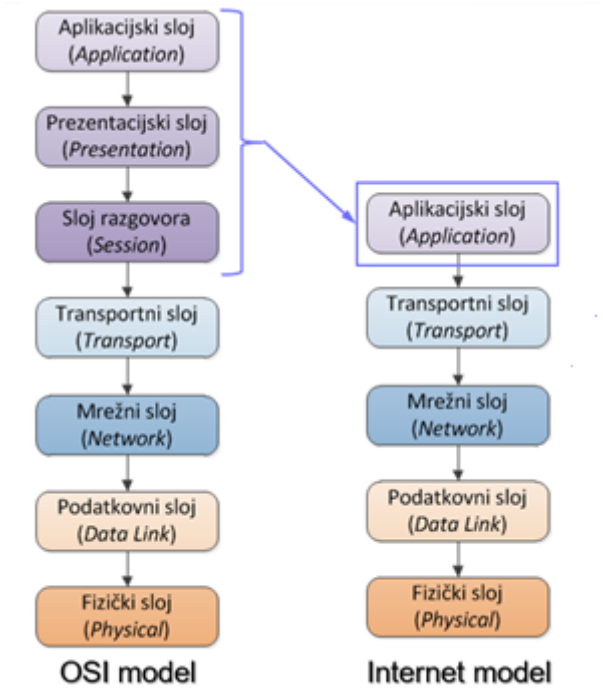
Uvjet ostvarivanja automatizacije procesa mjerenja je komunikacija između mjernog uređaja i upravljačkog sustava robota. Kako bi komunikacija bila moguća potrebno je uspostaviti mrežnu povezanost između upravljačkog računala mjernog uređaja i upravljačke jedinice robota. Navedeni uređaji spojeni su na preklopnik (engl. *switch*) ethernet kablom (Slika 10) koji omogućuje prijenos podataka. Računalna mreža pomoću koje se ostvaruje komunikacija biti će ukratko objašnjena u idućem poglavlju.



Slika 11. Povezivanje opreme ispitnog postava

3. RAČUNALNA MREŽA TEMELJENA NA TCP/IP PROTOKOLU

Računalna mreža omogućuje komunikaciju mrežnih korisnika, a slijed komunikacije se razdvaja na slojeve koji obavljaju pojedine procese. Iako je OSI (engl. *Open Systems Interconnection Basic Reference Model*) model definiran sa 7 slojeva, u internet komunikaciji je uobičajeno razmatrati samo 5 slojeva (Slika 12). Svaki sloj opisuje skup povezanih funkcija koje omogućuju jedan dio računalne komunikacije i ima neki oblik pakiranja podataka PDU (engl. *Protokol Data Unit*). Slojevi unutar jednog modela komuniciraju samo s prvim slojem iznad i prvim slojem ispod sebe. Svaki od modela u osnovi predstavlja jedan komunikacijski uređaj. [8]



Slika 12. Prikaz OSI i Internet modela komunikacije [8]

Razmjena podataka između kontrolera robota i softvera mjernog uređaja ostvarena je preko računalne mreže unutar aplikacijskog sloja točnije sloja razgovora (engl. *session*) koji se oslanja na TCP (engl. *Transmission Control Protocol*) protokol prijenosnog sloja te IP (engl. *Internet Protocol*) protokol mrežnog sloja. Jedinice za prijenos podataka IP protokolom nazivaju se paketi, dok se jedinice za prijenos podataka TCP protokolom nazivaju segmenti. [8]

Zadatak IP je da dostavlja pakete, od pošiljatelja do primatelja, na osnovi IP adrese. Svako sučelje računala prema mreži ima svoju jedinstvenu IP adresu (engl. *internet protocol address*) definiranu 32-bitnim brojem. [9]

TCP se brine za uspostavljanje i prekidanje veze te potvrđivanje primljenih segmenata i njihov pravilan poredak. Svaka TCP konekcija između dva računala je identificirana parom (izvorišni port, odredišni port). Mrežno sučelje računala podijeljeno je na 65535 ulaza, odnosno portova identificiranih 16-bitnim brojem. Broj porta označava određeni proces ili vrstu mrežne usluge, dok je prvih 1024 brojeva rezervirano za sustavne portove. Korištenjem koncepta portova (mrežnih vrata) omogućeno je da se na jednom sučelju može ostvariti više istovremenih veza prema više različitih aplikacija. [10]

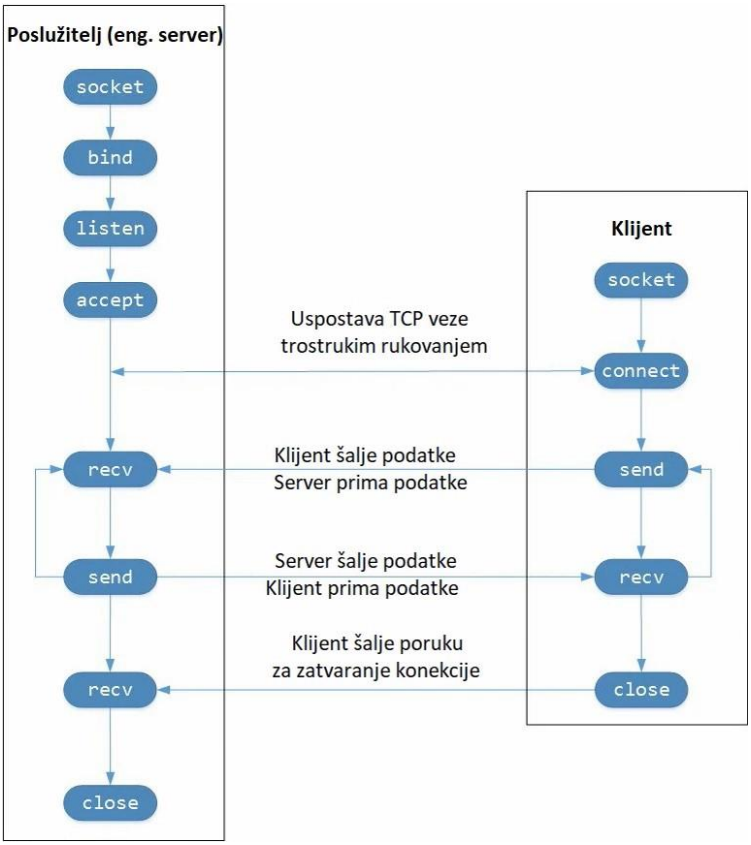
4. RAZVIJANJE KOMUNIKACIJSKOG MODELA KLIJENT – POSLUŽITELJ

Kako bi se razvio sustav automatskog mjerenja potrebno je razviti programski kod kojim će se uspostaviti komunikacija između upravljačkog računala mjernog uređaja i upravljačkog sustava robota. Programski kod na strani računala izrađen je u programskom jeziku Python, dok je kod na upravljačkog sustava robota napisan u programskom jeziku RAPID. Python podržava objektno orijentirano programiranje koje koristi objekte temeljene na podacima i funkcionalnosti. Posjeduje jednostavnu sintaksu te je programski jezik otvorenog koda (engl. *Open source*), što znači da se mogu dodavati dijelovi drugih kodova odnosno biblioteka. Zbog navedenih karakteristika Python 3.9. odabran je za izradu programa u ovom diplomskom radu.

Oba programska jezika podržavaju modul pristupne točke (engl. *socket*) te varijable i funkcije pomoću kojih je uspostavljena komunikacija temeljena na TCP/IP protokolu. Slika 13 prikazuje shemu uspostavljanja *socket* komunikacije.

Socket objekt služi kao komunikacijska veza između poslužiteljskog procesa i klijentskog procesa. Klijenti i poslužitelji su aplikacije ili procesi koji se izvode lokalno na jednom računalu ili udaljeno kroz računalnu mrežu. Klijent zahtjeva uslugu, šalje zahtjev poslužitelju i čeka odgovor. Dok poslužitelj nudi usluge, prima i obrađuje dolazne zahtjeve te šalje odgovor klijentima. [12]

Poslužitelj predstavlja program izrađen u programskom jeziku Python koji se izvršava na mjernom računalu, dok klijent predstavlja program izrađen u programskom jeziku RAPID koji se izvršava na upravljačkom računalu robota.



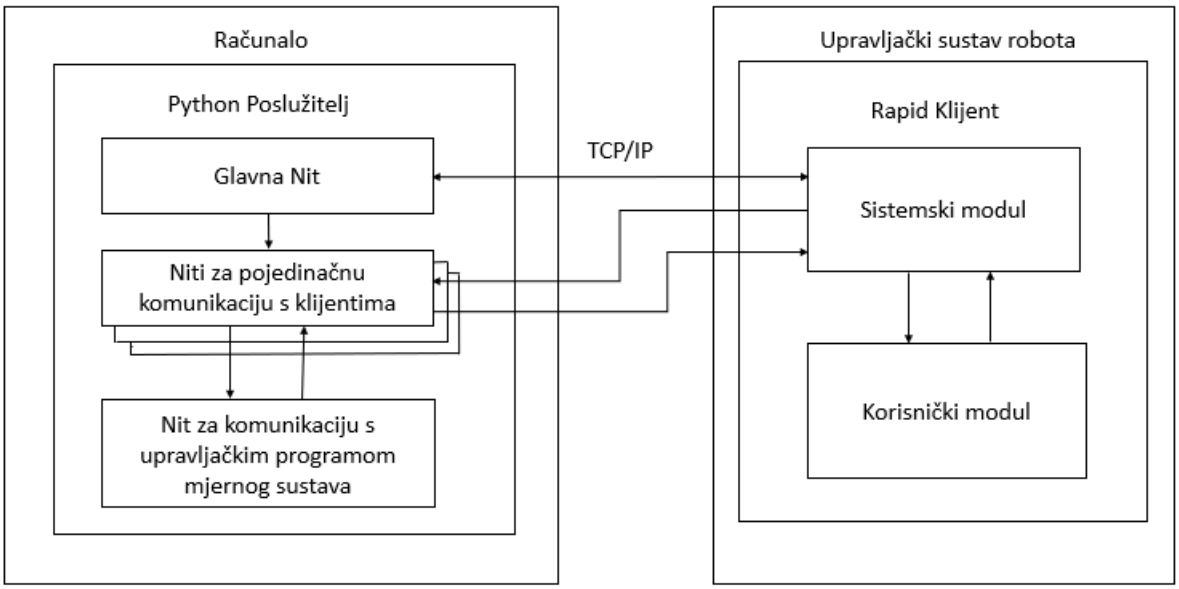
Slika 13. Shema uspostavljanja socket komunikacije [11]

Klijenti i poslužitelji komuniciraju slanjem i primanjem tekstualnih nizova (engl. *string*) kroz svoje *socket* veze. Poslužitelj može prihvaćati i obrađivati nekoliko klijenata istovremeno tako što svakome klijentu pridružuje posebnu nit (engl. *thread*). Nit je najmanja jedinica koja se može izvršiti u operacijskom sustavu kao dio računalnog programa, odnosno niz takvih instrukcija unutar programa koje se mogu izvršiti neovisno o drugom kodu. [10]

4.1. Struktura programa razvijenog modela klijent-poslužitelj

Unutar *socket*-a definirano je vrijeme (engl. *timeout*) unutar kojeg se ,ako nema razmijene podataka, veza s klijentom prekida te se uspostavlja ponovna konekcija. Na taj način se osigurava prekid izvršavanja programa u slučaju da dođe do greške izvođenja programa klijenta ili do nenadanog prekida veze.

Mjernom uređaju za izvršavanje pojedinih radnji treba kraći (npr. zadavanje imena direktorija u koji se pohranjuju rezultati mjerenja) ili dulji (npr. izvođenje automatskog fokusiranja) vremenski period te postoji mogućnost da se tijekom izvođenja operacija duljeg vremenskog perioda od zadanog isteka veze, prekida veza s klijentom. Kako bi se takva nepovoljna situacija izbjegla program na strani poslužitelja podijeljen je na tri niti kojima se omogućava kontinuirana komunikacija između poslužitelja i klijenta. Struktura programa prikazana je slikom 14.



Slika 14. Prikaz komunikacijskog modela klijent-poslužitelj

Glavna nit (engl. *Socket Main Thread*) pokreće pojedinačne niti (engl. *Socket Client Thread*) za komunikaciju s klijentima u trenutku uspostavljanja veze. Nit za komunikaciju s upravljačkim programom mjernog sustava nazvana je *Socket Alicona Thread*.

Unutar programskog koda klijenta definirana su dva modula, sistemski i korisnički. U sistemskom modulu definirane su procedure i funkcije za povezivanje i komunikaciju s poslužiteljem odnosno s mjernim uređajem (u radu nazvanog *srmLibAliconaSM*). Veza se uspostavlja putem korisničkog modula (u radu nazvanog *srmTestAuxR2*) koji koristi funkcije iz sistemskog modula, ali ujedno i pozicionira mjerni uređaj na pozicije obratka.

4.2. Programska podrška poslužitelja

Na početku skripte u kojoj je definiran kod poslužitelja, naredbom *import* se prvo učitavaju svi potrebni moduli, odnosno njihovi objekti, konstante, metode i iznimke (Slika 15). Modul *socket* služi za kreiranje *socket* sučelja. Modul *sys* omogućuje pristup parametrima i funkcijama specifičnim za sustav. Modul *time* omogućuje koristiti vrijeme unutar programa, npr. zaustavljanje/pauziranje izvršavanja. Modul *threading* služi definiranju niti i upravljanju njima, dok modul *queue* omogućuje implementaciju redova koji slijede FIFO (engl. *first in first out*) metodu umetanja i brisanja podataka unutar reda. Modul *PyAlicona* je zapravo skripta koja sadrži programski kod izrađen s ciljem upravljanja softverom mjernog uređaja preko udaljenog sučelja te zbog velikog opsega programskog koda neće biti prikazan u ovom radu. Njegovim uključivanjem omogućuje se pristup svim metodama za upravljanje mjernim uređajem.

```
import socket
import sys
import time
import threading
import queue

from PyAlicona import PyAlicona
```

Slika 15. Slika uključenih modula unutar skripte

Nakon uključivanja svih potrebnih modula definira se klasa (engl. *class*) nazvana *ABBAliconaSocketServ* te konstruktor klase pomoću metode *__init__()* (Slika 16). Unutar konstruktora definirani su objekti, argumenti, varijable, konstante te metode potrebne za upravljanje mjernim uređajem te definiranje *socket* poslužitelja i izvršavanje naredbi zadanih od strane klijenta, odnosno robota. Zadane vrijednosti mogu se promijeniti tijekom inicijalizacije objekta.

Varijable za evidentiranje (engl. *logging*) služe za sustavno praćenje procesa unutar programa. Varijable *socket* poslužitelja definiraju IP adresu poslužitelja (*srvBindIP*) te port na kojem poslužitelj osluškuje veze (*srvPort*). Vrijednost IP adrese "0.0.0.0" označava sve IPv4 adrese na uređaju povezanom na mrežu. Definirana je i vremenska varijabla s*rvTimeout* čijim se istekom prekida veza s klijentom ako nije bilo razmijene podataka te vrijeme dozvoljenog kašnjenja odgovora servera *srvRespDelay*. Parametar *srvMaxClient* označava maksimalan dozvoljeni broj klijenata, a *srvSocket* predstavlja objekt inicijaliziran u metodi *starSocketServ()* čijim se pokretanjem izvršava spajanje s klijentom, odnosno robotom.

```
class ABBAliconaSocketServ(object):

    def __init__(self, srvBindIP="0.0.0.0", srvPort=20025, srvTimeout=10, srvMaxClients=5, srvRespDelay=0.3, alHost="localhost", alPort=8086, autoStart=True, verb=0):

        ##############################################################################
        # Logging/output related objects/parameters
        ##############################################################################
        self.Verbose = verb                             # Verbose level
        self.printLock = threading.Lock()               # Lock object for console logging

        ##############################################################################
        # Socket server related objects/parameters
        ##############################################################################
        self.srvBindIP = srvBindIP                      # IP address to bind to
        self.srvPort = srvPort                          # Listening port
        self.srvTimeout = srvTimeout                    # Connection timeout
        self.srvMaxClients = srvMaxClients              # Maximum number of allowed clients
        self.srvRespDelay = srvRespDelay                # Server reply delay (fine adj. thread queue / sync)
        self.srvSocket = None                           # Server socket object (initialised in startSocketServ())

        self.srvRUN = False                             # Flag - socket server server is running
        self.srvLastRESPONSE=b''                        # Last server response

        self.SocketServerThread = None                  # Reference to server main thread (initialised in startSocketServ())
        self.clientThreadCounter=0                      # Client thread counter (total nr. of threads started)
        self.lstSrvClientThreads = list()               # List of references to running client threads
```

Slika 16. Prikaz konstruktora i varijabli klase

Varijable koji služe za definiranje IP adrese i porta udaljenog sučelja (*alHost* i *alPort*) prikazane su slikom 17. Za svako stanje izvršenja definiran je prefiks poruke koja se šalje klijentu. Prefiks ACK označava uspješno izvođenje naredbe dok prefiks BSY predstavlja stanje zauzetosti poslužitelja s izvođenjem trenutne naredbe. U slučaju greške prilikom izvođenja naredbe šalje se poruka s prefiksom ERR ili s prefiksom ERR-RI-NOT-

CONNECTED u slučaju greške spajanja s udaljenim sučeljem mjernog uređaja. Stanje poslužitelja inicijalno je postavljeno u zauzeto stanje (prefiks *BSY*).

Objektom *alinQueue* definira se red (engl. *queue*) metodom modula *queue* te se varijablom definira *maxsize* definira broj podataka unutar reda koji u ovom slučaju iznosi 1. Podaci poslani sa strane klijenta spremaju se u red iz kojeg poslije podatak preuzima *AliconaClientThread* odnosno dio programa koji izvršava određene naredbe na mjernom uređaju.

Pomoću metode *Lock* modula *threading* zaključavaju se varijable u koje pojedina nit upisuje podatke, odnosno sprječava se preuzimanje podatka ostalim nitma. Unutar konstruktora definirana je varijabla *autostart* te ako je njena vrijednost istinita (engl. *true*) izvršava se metoda *startAll()*.

```
# Alicona client (remoting interface) related objects/parameters
#################################################################################
self.alHost = alHost                          # Alicona server IP
self.alPort = alPort                          # Alicona server port
self.al = None                                # reference to PyAlicona object (remoting interface)
self.alRUN = False                            # Flag - alicona client (remoting interface) is running
self.alBUSY = False                           # Flag - alicona client (remoting interface) is busy

self.alRespPrefixACK = b'ACK'                 # Response prefix in case of sucsesfull CMD
self.alRespPrefixBSY = b'BSY'                 # Response prefix in case of busy executing CMD
self.alRespPrefixERR = b'ERR'                 # Response prefix in case of CMD error
self.alRespPrefixNC  = b'ERR-RI-NOT-CONNECTED' # Response prefix in case of Alicona client not connected

self.alClientThread = None                    # Reference to Alicona client thread
self.alInQueue = queue.Queue(maxsize=1)       # Queue used to feed data to alicona client  thread
self.alClientLock = threading.Lock()          # Lock object for alicona client responses

#################################################################################
#Automatic startup of Alicona client / Socket server threads
#################################################################################
if autoStart:
    self.startAll()
```

Slika 17 Prikaz varijabli konstruktora klase

Metoda *startAll()* (Slika 18) pokreće metodu uspostavljanja veze s udaljenim sučeljem mjernog uređaja *startAliconaClient()*, zatim metodu *startSocketServ()* u kojoj se stvara pristupna točka odnosno *socket* objekt. Unutar klase definirana je i metoda *stopAll()* koja poziva metode za zatvaranje pristupne točke *socket* poslužitelja *stopSocketServ()* te metode za zaustavljanje konekcije s udaljenim sučeljem *stopAliconaClient()*.

```
def startAll(self):
    self.startAliconaClient()
    self.startSocketServ()

def stopAll(self):
    self.stopSocketServ()
    self.stopAliconaClient()
```

Slika 18. Prikaz metoda *startAll()* i *stopAll()*

Metodom *startAliconaClient()* (Slika 19) uspostavlja se konekcija s udaljenim sučeljem softvera mjernog uređaja putem *PyAlicona* skripte. Također je definirana i pokrenuta nit *AliconaClientThread* koja izvršava metodu *__ProcessAlRq()*. Nit se definira uvođenjem *thread* objekta pomoću modula *Threading*. Argumenti *Thread* objekta su metoda koji želimo da nit izvrši postavljena kao cilj/meta (engl. *target*) i naziv same niti (engl. *name*). Metodom *start* započinje se izvršavanje niti. Metoda *self.__iprint* služi za praćenje izvođenja metoda u standardnom ispisu, definirana je na kraju skripte te će biti prikazana poslije u radu.

```
def startAliconaClient(self):
    if not self.alRUN:
        self.__iprint("Starting alicona client ...")

        with self.alClientLock:
            self.alRUN = True

        if self.al is None:
            self.al = PyAlicona(host=self.alHost, port=self.alPort, verb=self.Verbose)
        else:
            self.__iprint("WARNING: alicona client seems to be already connected...")

        if isinstance(self.alClientThread, threading.Thread) and (self.alClientThread.is_alive()):
            self.__iprint("WARNING: alicona client thread seems to be already running...")
        else:
            self.alClientThread = threading.Thread(target=self.__ProcessAlRq, name="AliconaClientThread")
            self.alClientThread.start()
    else:
        self.__iprint("Alicona client already started ...")
```

Slika 19. Prikaz metode *startAliconaClient()*

Metoda *stopAliconaClient()* (Slika 20) zaustavlja dretvu *AliconaClientThread* tako da se red prvo prazni pomoću naredbe *clear* te se u red postavlja varijabla *none*. Naredbom *mutex* ostalim se nitima sprječava dohvaćanje podataka iz reda. Naredbom *join* privremeno se pauzira izvršavanje programa do trenutka kada je nit AliconaClientThread u potpunosti izvršena.

```
    def stopAliconaClient(self):
        self.__iprint("Stopping alicona client ...")

        with self.alClientLock:
            self.alRUN = False

        with self.alInQueue.mutex:
            self.alInQueue.queue.clear()
        self.alInQueue.put(None)

        if self.alClientThread is not None:
            self.alClientThread.join()
            self.__iprint("Stopped alicona client ...")
            try:
                del(self.al)
            except Exception as e:
                pass
            self.al = None
```

Slika 20. Prikaz metode *stopAliconaClient()*

Unutar metode *startSocketServ()* (Slika 20) stvara se objekt pristupne točke koji se dodjeljuje varijabli *srvSocket*. Argument *socket.AF_INET* označava vrstu pristupne točke koja koristi IPv4 kao adresni prostor, dok se argumentom *socket.SOCK_STREAM* definira tip pristupne točke za komunikaciju putem TCP protokola. Naredba *bind()* veže socket objekt na definiranu IP adresu (“0.0.0.0”) i port (20025). Također je definirana i pokrenuta nit *SocketServerMainThread* koji izvršava metodu *__runSocketServ()*.

```
    def startSocketServ(self):
        if not self.srvRUN:
            self.__iprint("Starting socket server ...")
            self.srvSocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.srvSocket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
            self.srvSocket.bind((self.srvBindIP, self.srvPort))
            self.srvRUN = True
            self.SocketServerThread = threading.Thread(target=self.__runSocketServ, name="SocketServerMainThread")
            self.SocketServerThread.start()
        else:
            self.__iprint("Socket server already running ...")
```

Slika 21. Prikaz metode *startSocketServ()*

Metodom *stopSocketServer()* (Slika 21) zaustavljaju se sve niti koje su uspostavljene za komunikaciju sa klijentima te nit *SocketServerMainThread()*. *Socket* objekt odnosno pristupna točka zatvara se naredbom *close()* i uništava pomoću naredbe *del()*.

```
def stopSocketServ(self):
    self.__iprint("Stopping socket server ...")
    self.srvRUN = False
    if self.Verbose >= 1:
        self.__iprint("\tWaiting for client threads to join ...")
    for _SocketClientThread in self.lstSrvClientThreads:
        if self.Verbose >= 1:
            self.__iprint("\t\t Joining thread %s ..." % (_SocketClientThread.name))
        _SocketClientThread.join()

    if self.SocketServerThread is not None:
        if self.Verbose >= 1:
            self.__iprint("\tWaiting for server thread to join ...")

        try:
            socket.socket(socket.AF_INET,socket.SOCK_STREAM).connect(("127.0.0.1", self.srvPort))
        except Exception as e:
            pass

        self.SocketServerThread.join()

    if self.Verbose >= 1:
        self.__iprint("\tClosing socket ...")

    if self.srvSocket is not None:
        self.srvSocket.close()

    del (self.srvSocket)
    self.srvSocket = None
    self.__iprint("Socket server stopped ...")
```

Slika 22. Prikaz metode *stopSocketServ()*

Unutar metode *__runSocketServ()* pomoću naredbe *listen()* osluškuju se veze klijenata s definiranom varijablom unutar konstruktora *srvMaxClients* koja označava maksimalan dopušten broj klijenata koji se mogu povezati s poslužiteljem. Također se prihvaćaju konekcije *socket* naredbom *accept()* koja vraća dvije vrijednosti, odnosno *socket* objekt koji služi za primanje i slanje podataka klijentu pridružen varijabli *_conn* i adresu klijenta pridruženu varijabli *_addr*. Unutar metode definirano je i dodjeljivanje nove niti *SocketClientThread* svakom povezanom klijentu. Broj niti klijenta sprema se u listu. Ako

jednom klijentu treba više vremena za primanje ili obradu podataka, poslužitelj ne mora čekati već može komunicirati s drugim klijentima koristeći zasebne niti za svakog klijenta.

```
def __runSocketServ(self):
    self.srvSocket.listen(self.srvMaxClients)
    self.lstSrvClientThreads = list()

    while self.srvRUN:
        self.__iprint("Waiting for connection...")

        #Lista (samo) živih threadova
        self.lstSrvClientThreads = [_thread for _thread in self.lstSrvClientThreads if _thread.is_alive()]
        self.clientThreadCounter += 1

        _conn, _addr = self.srvSocket.accept()
        _SocketClientThread = threading.Thread(target=self.__ProcessClRq, args=(_conn, _addr,), name="SocketClientThread%s"
        _SocketClientThread.start()
        self.lstSrvClientThreads.append(_SocketClientThread)
```

Slika 23. Prikaz metode *__runSocketServ()*

Unutar metode *__ProcessClRq()* (Slika 24 i Slika 25) zaprimaju se i šalju podaci klijentu. Varijabla *_data* zaprima podatke pomoću metode *socket* metode *recv()* te je definirano čitanje podataka u serijama od 1024 bajta. Argument *srvTimeout* funkcije *settimeout()* definira vremensko ograničenje za primanje podataka od klijenta definiranog unutar konstruktora klase na vrijednost od 10 sekundi. Ako nije došlo do isteka vremena i primljeni podaci nisu prazni, poruka se sprema pomoću naredbe *put()* u red (engl. *queue*) inicijaliziranog u konstruktoru klase. Ograničavanjem veličine reda omogućeno je sprječavanje izvršavanja nove operacije s mjernim uređajem za vrijeme dok mjerni uređaj već izvršava neku zadanu operaciju.

Unutar metode definirana je i varijabla *_retData* kojoj se pridružuje vrijednost varijable *srvLastRESPONSE* odnosno odgovor niti *AliconaClientThread* o statusu izvršenja funkcije na mjernom uređaju koji se šalje klijentu *socket* metodom *sendall*. Odgovori se zaključavaju odnosno sprječavaju da ga ostale niti promijene pomoću naredbe *with lock* objekta dostupnog unutar modula *threading*.

```
    def __ProcessClRq(self, conn, addr):
        _retData = b''
        #conn.settimeout(self.srvTimeout)
        #self._iprint("Connected from %s (%s)" % (addr, conn))
        self.__iprint("Connected from %s" % str(addr))

        while self.srvRUN:
            if self.Verbose >= 2:
                self.__iprint(" ")

            try:
                conn.settimeout(self.srvTimeout)
                _data = conn.recv(1024)

                if not _data:
                    #self.__iprint("Connection lost from %s (%s) - BLANK DATA" % (addr, conn))
                    self.__iprint("Connection lost from %s  (BLANK DATA)" % (str(addr)))
                    break

                else:
                    if self.Verbose >= 1:
                        self.__iprint("Received from %s : '%s'" % (addr, _data))

                    # If Alicona client is not running,
                    # response should be NC error without any further processing
                    if self.al is None or self.alRUN == False:
                        _retData = b"%b" % (self.alRespPrefixNC)
```

Slika 24. Prikaz metode *__ProcessClRq()*

```
                except queue.Full:
                    if self.Verbose >= 2:
                        self.__iprint("AliconaClient Queue is FULL, replying last response")
                    _retData = self.srvLastRESPONSE
                else:
                    if self.Verbose >= 2:
                        self.__iprint("AliconaClient Queue is OK, command sent...")

                    if self.srvLastRESPONSE == b'':
                        if self.Verbose >= 2:
                            self.__iprint("Response is blank, waiting 0.5s")
                        time.sleep(0.5)
                    else:
                        time.sleep(self.srvRespDelay)

                    with self.alClientLock:
                        _retData = self.srvLastRESPONSE


            if self.Verbose >= 1:
                self.__iprint("Sending response to %s : '%s'" % (addr, _retData.decode()))
            conn.sendall(_retData)

        except socket.timeout:
            self.__iprint("\nERROR: Socket timeout from %s (%s)" % (addr, conn))
            break
```

Slika 25. Prikaz metode *__ProcessClRq()*

Nit *Alicona Client Thread* izvršava metodu *__ProcessAlRq()* (Slika 26 i Slika 27) u kojoj preuzima podatke iz reda naredbom *get()*. Preuzeti podatak dodijeljuje se varijabli *_cmd*, čija se vrijednost provjerava višestrukim uvjetnim grananjem unutar petlje definirane u metodi *__ProcessAlRq()*. Obzirom na vrijednost varijable izvodi se odgovarajući blok naredbi. Ako red primi NULL petlja se prekida. Budući da se funkcija izvršava u zasebnoj niti, ta nit će se prekinuti. U slučaju da varijabla primi vrijednost SRQ, izvršava se brzi odgovor o statusu izvođenja zadane metode. Ostale poruke klijenta označavaju pozivanje određenih metoda PyAlicona modula, odnosno izvođenje određenih naredbi na mjernom uređaju pomoću udaljenog sučelja. Primjerice ako zaprimljena poruka glasi SFN, poziva se metoda *setFolderName()* kojom se izvršava zadavanje naziva direktorija u kojeg se pohranjuju rezultati mjerenja.

Tip podataka koji se šalje naredbom *sendall()* i prima naredbom *recv()* je bajtovni zapis (engl. *bytes*). Stoga je poruku koja se šalje iz znakovnog niza (engl. *string*) potrebno kodirati u bajtovni zapis naredbom *encode()*, dok se naredbom *decode()* omogućava kodiranje iz bajtovnog zapisa u niz znakova u slučaju zaprimanja poruka.

```
def __ProcessAlRq(self):
    while self.alRUN:
        if self.Verbose >= 2:
            self.__iprint("Waiting for Queue...")

        _cmd = self.alInQueue.get()

        if self.Verbose >= 3:
            self.__iprint("-" * 20)


        # Ako queue primi NULL, petlja se prekida
        # (buduci da se funkcija izvrsava u zasebon thread-u, taj thread ce se prekinuti)
        if _cmd is None:
            if self.Verbose >= 1:
                self.__iprint("Received NULL in queue, terminating...")
            with self.alClientLock:
                self.alRUN = False
            break

        # SRQ command does not naredba ne koristi niti jednu funkciju iz Alicona namespace-a i sluzi
        # za brzi odgovor o statusu Alicona klijenta
        elif (_cmd == b'SRQ'):
            self.__iprint("STATUS REQUEST")
            _response = self.srvLastRESPONSE
```

Slika 26. Prikaz metode *__ProcessAlRq()*

```
        # Sve ostale funckije ispod pozivaju funkcije povezane s Alicona namespace-om;
        # za njih se moze ocekivati veci ili manji delay u izvsravanju.
        else:

            # Inicijalno se odgovor postavlja u BUSY stanje...
            _response = b"%b-%b" % (self.alRespPrefixBSY, _cmd)

            #Lock critical vars using thread lock
            with self.alClientLock:
                self.alBUSY = True
                self.srvLastRESPONSE = _response

            if (_cmd == b'SMPRQ'):
                # StartMeasurementProcedure
                self.__iprint("START MEASUREMENT PROCEDURE REQUEST")
                self.al.setDefaultParams()

            elif (_cmd.decode()[0:3] == 'SFN'):
                data_decoded = _cmd.decode().replace("SFN", "")
                self.__iprint("SET FOLDER NAME: %s" % (data_decoded))
                self.al.setFolderName(data_decoded)
                _response = b"%b-%b" % (self.alRespPrefixACK, _cmd)

            elif (_cmd.decode()[0:3] == 'SPN'):
                data_decoded = _cmd.decode().replace("SPN", "")
                self.__iprint("SET PROJECT NAME: %s" % (data_decoded))
                self.al.setProjcetName(data_decoded)
                _response = b"%b-%b" % (self.alRespPrefixACK, _cmd)
```

Slika 27. Prikaz metode *__ProcessAlRq()*

Na kraju izvršenja svake funkcije unutar metode *__ProcessAlRq()* (Slika 28) odgovor *_response* se zaključava te pridružuje varijabli *srvLastRESPONSE*. Nakon izvršenja varijablu preuzima nit *SocketClientThread* te se odgovor šalje klijentu.

```
        with self.alClientLock:
            self.srvLastRESPONSE = _response
            self.alBUSY = False

        self.__iprint("RESPONSE: %s" % self.srvLastRESPONSE.decode())
```

Slika 28. Prikaz metode *__ProcessAlRq()*

Na kraju programa definiran je destruktor klase *__del__()* (koji služi zaustavljanju programa pozivajući metodu *stopAll()*), metoda *__iprint* čijim se pozivom ispsuje status izvođenja programa te metoda *printThreads()* koja ispisuje postojeće niti (Slika 29).

```
    def __del__(self):
        self.__iprint("Destroying object...")
        self.stopAll()
        self.__iprint("Object destroyed...")
        #del (self.al)

    def __iprint(self, strData=""):
        with self.printLock:
            print("%s.%s: %s" % (self.__class__.__name__, sys._getframe(1).f_code.co_name, strData))

    def printThreads(self):
        self.__iprint("--------------| RUNNING THREADS: |--------------")
        for _thread in threading.enumerate():
            self.__iprint(" \t %s" % (_thread.name))
        if len(self.lstSrvClientThreads) > 0:
            self.__iprint("-------| SOCKET SERVER CLIENT THREADS: |--------")
            for _thread in self.lstSrvClientThreads:
                self.__iprint(" \t %s" % (_thread.name))
        self.__iprint("-----------------------------------------------")
```

Slika 29. Prikaz metoda *__del__(), __iprint() i printThreads()*

4.3. Programska podrška klijenta

U sistemskom modulu nazvanom *srmLibAliconaSM.sys* definirani su objekti odnosno varijable potrebne za definiranje pristupne točke odnosno *socket*-a (Slika 30) te procedure za komunikaciju i slanje naredbi poslužitelju. Varijabla *sdAlClient*, kojojm se omogućuje stvaranje pristupne spojne točke, spajanje na poslužitelja i prekid veze s poslužiteljem, definirana je vrstom podataka *socketdev* (engl. *socket device*). Varijabla *ssAlClient* definirana vrstom podataka *socketstatus* služi za provjeru statusa spajanja i kontrolu tijeka programa. IP adresa poslužitelja definiranana je varijablom *strAlHostIP* tipa *string*, dok je port poslužitelja definiran varijablom *numAlHostPort* tipa *num*. Nakon definiranja varijabli pristupne točke (engl. *socket*), pokretanjem modula, pristupna točka se automatski uspostavlja i spaja s poslužiteljem.

T_ROB1/srmLibAliconaSM

```
1   MODULE srmLibAliconaSM(SYSMODULE)
2
3       VAR socketdev sdAlClient;
4       VAR socketstatus ssAlClient;
5       !VAR string strAlClientRcv;
6       VAR string strAlHostIP:= "192.168.2.141";
7       VAR num numAlHostPort := 20025;
8
```

Slika 30. Prikaz definiranja pristupne točke klijenta

Procedure i funkcije unutar ovog modula mogu se podijeliti na komunakcijske i izvršne. Komunikacijskim procedurama smatraju se one čijim se izvršavanjem uspostavlja ili prekida veza s poslužiteljem te provjerava stanje veze. Komunikacijskom funkcijom primaju se i šalju podaci poslužitelju. Izvršne funkcije namijenjene su za definiranje podataka odnosno parametara koji se šalju poslužitelju. Dalje u radu, prvo će biti objašnjen programski kod pomoću kojeg su definirane komunikacijske procedure i funkcije te nakon toga kod vezan za izvršne procedure.

4.3.1. Komunikacijske procedure

4.3.1.1. Procedura *alConnect()*

Budući da se na opisani način definiranja *socket* varijabli unutar sistemskog modula, klijent odnosno program koji se izvršava na upravljačkom sustavu robota automatski spaja s poslužiteljem, u korisničkom modulu nije potrebno pozivati dodatnu proceduru za povezivanje. No, procedurom *alConnect()*, prikazanom slikom 31 moguće je ubrzati spajanje te provjeriti status spajanja. Argumenti procedure *alConnect()* su IP adresa i port poslužitelja te broj pokušaja ponovnog povezivanja. Pomoću funkcije *SocketGetStatus()* provjerava se status spajanja. U slučaju da je pristupna točka zatvorena ili nije u jednom od sljedećih navedenih stanja: stvorena, u stanju slušanja ili povezana, izvršava se pozivanje procedure *alReconnect()*. Status pristupne točke ispisuje se na operatorski panel naredbom *TPWrite()*.

```
PROC alConnect(\string Host, \num Port, \num RecconectionAttempts, \bool Verbose )
    VAR bool bRet := FALSE;
    VAR string strAlHost;
    VAR num numAlPort;
    VAR num numRecconections;          !Number of recconection attempts
    VAR bool bVerb;                 !Enable/disable verbose logging to TP

    IF Present (Host) THEN strAlHost := Host; ELSE strAlHost := strAlHostIP; ENDIF
    IF Present (Port) THEN numAlPort := Port; ELSE numAlPort := numAlHostPort; ENDIF
    IF Present (RecconectionAttempts) THEN numRecconections := RecconectionAttempts; ELSE numRecconections := 1; ENDIF
    IF Present (Verbose) THEN bVerb := Verbose; ELSE bVerb := FALSE; ENDIF

    ssAlClient := SocketGetStatus(sdAlClient);
    TEST ssAlClient
        CASE SOCKET_CREATED:
            if (bVerb =TRUE) THEN TPWrite "AlConnect: SOCKET_CREATED; skipping connection..."; ENDIF

        CASE SOCKET_CLOSED:
            if (bVerb =TRUE) THEN TPWrite "AlConnect: SOCKET_CLOSED; (re)connecting..."; ENDIF
            alReconnect \Host:= strAlHost, \Port:= numAlPort, \RecconectionAttempts:=numRecconections, \Verbose:= bVerb;

        CASE SOCKET_BOUND:
            if (bVerb =TRUE) THEN TPWrite "AlConnect: SOCKET_BOUND; invalid fo rclient"; ENDIF

        CASE SOCKET_LISTENING:
            if (bVerb =TRUE) THEN TPWrite "AlConnect: SOCKET_LISTENING; invalid for client"; ENDIF
                !ErrWrite "alConnect()", "Socket in listening mode!", \RL2:="Socket should be CLOSED or CONNECTED";
                !Break;
                !SystemStopAction \StopBlock;

        CASE SOCKET_CONNECTED:
            if (bVerb =TRUE) THEN TPWrite "AlConnect: SOCKET_CONNECTED; skipping (re-connection)..."; ENDIF
            bRet := TRUE;

        DEFAULT:
            if (bVerb =TRUE) THEN TPWrite "AlConnect: UNKNOWN SOCKET STAT"; ENDIF
            alReconnect \Host:= strAlHost, \Port:= numAlPort, \RecconectionAttempts:=numRecconections, \Verbose:= bVerb;

    ENDTEST

ENDPROC
```

Slika 31. Prikaz procedure *alConnect()*

4.3.1.2. Procedura *alReconnect()*

Unutar procedure *alReconnect()* (Slika 32) prvo se naredbom *SocketClose* prekida veza s poslužiteljem. Nakon prekida veze naredbom *SocketCreate* stvara se novi objekt pristupne točke, a zatim se naredbom *SocketConnect* ponovno uspostavlja veza s poslužiteljem. Atributima naredbe *SocketConnect* definira se IP adresa (*strAlHost*) i port (*numAlPort*) poslužitelja te maksimalno vrijeme čekanja za uspostavu veze (*Time*) koje iznosi 2 sekunde.

```
PROC alReconnect(\string Host, \num Port, \num RecconectionAttempts, \bool Verbose )
    VAR bool bRet := FALSE;
    VAR string strAlHost;
    VAR num numAlPort;
    VAR num numAaxRecconections;          !Number of recconection attempts
    VAR bool bVerb;                       !Enable/disable verbose logging to TP
    VAR num numRecconectionAttempt := 0;

    IF Present (Host) THEN strAlHost := Host; ELSE strAlHost := strAlHostIP; ENDIF
    IF Present (Port) THEN numAlPort := Port; ELSE numAlPort := numAlHostPort; ENDIF
    IF Present (RecconectionAttempts) THEN numAaxRecconections := RecconectionAttempts; ELSE numAaxRecconections := 1; ENDIF
    IF Present (Verbose) THEN bVerb := Verbose; ELSE bVerb := FALSE; ENDIF

    SocketClose sdAlClient;
    SocketCreate sdAlClient;
    SocketConnect sdAlClient, strAlHost, numAlPort, \Time:=2;
```

Slika 32 Prikaz procedure *alReconnect()*

Ako je uspostava veze prihvaćena ili odbijena, nakon 2 sekunde podiže se iznimka *ERR_SOCK_TIMEOUT* koju je potrebno obraditi (Slika 33). Ako iznimka nije obrađena zaustavlja se izvođenje povezivanja. Varijabla *numAaxRecconection* predstavlja broj pokušaja ponovnog spajanja. Vrijednost varijable je 1, osim ako nije definirana neka druga vrijednost prilikom pozivanja procedure atributom *RecconectionAttempts*. Vrijednost varijable *numAaxRecconection* je 0. Upravljanje iznimkom *ERR_SOCK_TIMEOUT* izvedeno je pomoću sustavne varijable *ERRNO* (Slika 33). Ako je *ERRNO= ERR_SOCK_TIMEOUT* i ako je broj pokušaja ponovnog spajanja (*numAaxRecconection*) veći od varijable *numRecconectionAttempt*, povećava se vrijednost varijable *numAaxRecconection* za 1 te se naredbom *RETRY* izlazi iz bloka upravljanja pogreškom i pokušava se novo uspostavljanje veze s poslužiteljem.

```
        IF ERRNO = ERR_SOCK_TIMEOUT THEN
            IF numRecconectionAttempt < numAaxRecconections THEN
                if (bVerb =TRUE) THEN TPWrite "alReconnect: ERR_SOCK_TIMEOUT; ATTEMPT [" + NumToStr(numRecconectionAttempt,0) + "/" + NumToStr(numAaxRecconections,0)  +"]
                WaitTime 1;
                numRecconectionAttempt := numRecconectionAttempt + 1;
                RETRY;
            ELSE
                !BIN CASE OF ANY OTHER ERROR
                ErrWrite "alReconnect()", "Unable to establish connection to the Alicona server", \RL2:="Please check if Alicona server is running";
                Break;
                SystemStopAction \StopBlock;
                RAISE;
            ENDIF
        ELSE
            !BIN CASE OF ANY OTHER ERROR
            ErrWrite "alReconnect()", "Unable to establish connection to the Alicona server", \RL2:="Please check if Alicona server is running";
            Break;
            SystemStopAction \StopBlock;

        ENDIF

ENDPROC
```

Slika 33. Prikaz procedure *alReconnect()*

4.3.1.3. Procedura *alDisconnect()*

U proceduri *alDisconnect* (Slika 34) pomoću naredbe *SocketClose* pristupna točka se zatvara te se prekida veza sa poslužiteljem.

```
PROC alDisconnect(\bool Verbose)
    VAR bool bVerb;                 !Enable/disable verbose logging to TP
    IF Present (Verbose) THEN bVerb := Verbose; ELSE bVerb := FALSE; ENDIF
    SocketClose sdAlClient;
    IF (bVerb =TRUE) THEN TPWrite "alDisconnect: DISCONNECTED"; ENDIF
ENDPROC
```

Slika 34. Prikaz procedure *alDisconnect()*

4.3.1.4. Funkcija *alSendRcvMsg()*

Funkcija *alSendRcvMsg()* (Slika 35) služi za slanje i primanje podataka poslužitelja. Prilikom pozivanja funkcije potrebno je definirati varijablu *strMsg2Send* koja predstavlja poruku odnosno naredbu koja se šalje poslužitelju. Ostali argumenti funkcije su opcionalni i ne moraju se definirati prilikom pozivanja funkcije osim u slučaju zadavanja novih vrijednosti.

```
FUNC string alSendRcvMsg(string strMsg2Send, \num rcvTimeout, \num bsyRetrysMax, \num bsyReryDelay, \bool bVerbose)
    VAR bool bVerb := FALSE;
    VAR string strMsg2Receive := "";
    VAR string strExpctedACKMsg := "";
    VAR string strExpctedBSYMsg := "";
    VAR string strExpctedERRMsg := "";
    VAR string strExpctedERRNCMsg := "ERR-RI-NOT-CONNECTED";

    VAR num numTimeout := 2;
    VAR num numBsyRetrysMax :=100;
    VAR num numBsyReryDelay := 0.5;
    VAR bool msgOK := FALSE;

    VAR bool bRunLoop := TRUE;

    IF Present(bVerbose) THEN bVerb := bVerbose; ELSE bVerb := FALSE; ENDIF
    IF Present(rcvTimeout) THEN numTimeout := rcvTimeout; ENDIF
    IF Present(bsyRetrysMax) THEN numBsyRetrysMax := bsyRetrysMax; ENDIF
    IF Present(bsyReryDelay) THEN numBsyReryDelay := bsyReryDelay; ENDIF
```

Slika 35. Prikaz funkcije *alSendRcvMsg()*

Očekivane poruke poslužitelja definirane su pomoću tri varijable (Slika 36). Očekivana poruka je zapravo ista naredba odnosno *string* koji je poslan poslužitelju uz prefiks koji označava stanje izvršenja naredbe: ACK u slučaju uspješnog izvršenja, BSY u slučaju da izvršenje još traje te ERR u slučaju neuspješnog izvršenja. Slanje poruka poslužitelju izvršava se pomoću naredbe *SocketSend()*. Podaci koji se mogu slati naredbom *SocketSend()* mogu biti tipa *string, rawbytes ili array of byte*. Za kodiranje jednog znaka podatka tipa *string* koristi se jedan bajt. Veličina poruke koja se šalje je maksimalno 80 bajta. [7] Argument *strMsg2Receive* naredbe *SocketRecieve()* predstavlja varijablu koja sprema primljene podatke. *ReadNoOfBytes* argument čija je definirana vrijednost 80, označava količinu bajtova koju *SocketRecieve()* može primiti. Unutar petlje naredbom *SocketRecieve()* primaju se poruke od poslužitelja te se izvršava određeni blok naredbi. Ako je primljena poruka očekivana poruka s prefiksom *ACK*, poruka se ispisuje na operatorski panel te se prekida petlja. U slučaju da je primljena poruka s prefiksom *BSY*, šalje se poruka s naredbom *SRQ* za brzi odgovor o statusu poslužitelja te se naredbom *WaitTime()* određuje čekanje od 0,5 sekundi. Na kraju funkcije *alSendRcvMsg()* naredbom *RETURN* funkcija vraća vrijednost *strMsg2Recieve* koja se ispisuje na operatorskom panelu.

```
strExpctedACKMsg := "ACK-" + strMsg2Send;
strExpctedBSYMsg := "BSY-" + strMsg2Send;
strExpctedERRMsg := "ERR-" + strMsg2Send;

SocketSend sdAlClient \Str:=strMsg2Send;
IF (bVerb =TRUE) THEN TPWrite "alSendRcvMsg: TX:" + strMsg2Send; ENDIF
WHILE (bRunLoop) DO
    SocketReceive sdAlClient \Str:=strMsg2Receive, \ReadNoOfBytes:=80, \Time:=numTimeout;
    IF (bVerb =TRUE) THEN TPWrite "alSendRcvMsg: RX:" + strMsg2Receive; ENDIF

    IF strMsg2Receive = strExpctedACKMsg THEN
        IF (bVerb =TRUE) THEN TPWrite "alSendRcvMsg: RX ACK:" + strMsg2Receive; ENDIF
        bRunLoop := FALSE;

    ELSEIF strMsg2Receive = strExpctedBSYMsg THEN
        SocketSend sdAlClient \Str:="SRQ";
        IF (bVerb =TRUE) THEN TPWrite "alSendRcvMsg: RX BSY -> TX: SRQ"; ENDIF
        WaitTime 0.5;

    ELSE
        !KOd isppd moze biti greska ili ACK od neke druge komende
        !Ovaj kod svakako razraditi....
        SocketSend sdAlClient \Str:=strMsg2Send;
        IF (bVerb =TRUE) THEN TPWrite "alSendRcvMsg: RX DIFFERENT-ACK/ERR -> TX:" + strMsg2Send; ENDIF
        WaitTime 0.5;

    ENDIF


ENDWHILE
```

Slika 36. Prikaz funkcije *alSendRcvMsg()*

Argument *Time* čija je vrijednost određena varijablom *numTimeout*, definira maksimalno vrijeme čekanja za primanje podataka. Ako definirano vrijeme istekne, podiže se iznimka *ERR_SOCK_TIMEOUT* koju je potrebno obraditi (Slika 37). Upravljanje *ERR_SOCK_TIMEOUT* iznimkom riješeno je na način da se u slučaju pojave ove iznimke poziva procedura *atReconnect()* za ponovno spajanje s poslužiteljem.

```
ERROR
    IF ERRNO=ERR_SOCK_TIMEOUT THEN
        IF (bVerb =TRUE) THEN TPWrite "SM ERR; TX:" + strMsg2Send + "; RX:" + strMsg2Receive; ENDIF
        alReconnect;
        RETRY;
        ErrWrite "alSendRcvMsg:", "ERR_SOCK_TIMEOUT to the Alicona server", \RL2:="Please check if Alicona server is running";
        Break;
        SystemStopAction \StopBlock;
        RAISE;

    ELSEIF ERRNO=ERR_SOCK_CLOSED THEN
        IF (bVerb =TRUE) THEN TPWrite "SM ERR; TX:" + strMsg2Send + "; RX:" + strMsg2Receive; ENDIF
        alReconnect;
        RETRY;
        ErrWrite "alSendRcvMsg:", "ERR_SOCK_CLOSED to the Alicona server", \RL2:="Please check if Alicona server is running";
        Break;
        SystemStopAction \StopBlock;
        RAISE;
```

Slika 37. Prikaz funkcije *alSendRcvMsg()*

4.3.2. Izvršne procedure

U svakoj izvršnoj proceduri definira se različita poruka odnosno string vezan za pozivanje određene funkcije skeniranja koji se poslužitelju šalje prethodno opisanom funkcijom *alSendRcvMsg()*. Tablica 2 prikazuje sve izrađene procedure te stringove koje šalju poslužitelju te metode unutar programskog koda poslužitelja koje se temeljem preuzetog stringa izvršavaju.

Tablica 2. Nazivi izvršnih procedura, stringovi koji se šalju i metode poslužitelja koje pokreću

NAZIV PROCEDURA	STRING	METODA POSLUŽITELJA
alSetFolderName()	*SFN*	*setFolderName()*
alSetProjectName()	*SPN*	*setProjcetName()*
alScanStoreSurfaceImage()	*SCSTI*	*scanStoreSurfaceImage()*
alScanAndStoreSurface()	*SCSTS*	*scanStoreSurface()*
alScanStoreSurfaceVCImage()	*SCSTVI*	*scanStoreSurfaceVignettingCorrectionImage()*
alConAutoExsposure()	*CAEX*	*conAutoExposure()*
alMoveToLastFocalPt()	*MVLFP*	*moveToLastFocusPt()*
alIterativeAutoFocus()	*IAF*	*iterativeAutoFocus()*
	SUSPR	*setUpperScanPosRel()*
	SLSPR	*setLowerScanPosRel()*
alSendCurrentTCP()	*STCTX*	*setWTMTransX()*
	STCTY	*setWTMTransY()*
	STCTZ	*setWTMTransZ()*
	STCRX	*setWTMRotX()*
	STCRY	*setWTMRotY()*
	STCRZ	*setWTMRotZ()*

Automatske funkcije, kao što je automatsko namještanje ekspozicije senzora (*alConAutoExsposure*) (Slika 38) i pozicioniranje uređaja u zadnju zabilježenu žarišnu točku (*alMoveToLastFocalPt()*) imaju definiran samo *string* (*strMsgSend*) koji pozivajući funkciju *alSendRcvMsg()* šalju poslužitelju koji temeljem stringa izvršavaju zadanu naredbu. Vrijednost koju vraća funckija *atSendRcvMsg()* sprema se u varijablu *strMsgRecv*.

```
PROC alConAutoExsposure(\bool Verbose)
    VAR bool bVerb;
    VAR string strMsgSend := "CAEX";
    VAR string strMsgRecv := "";
    IF Present (Verbose) THEN bVerb := Verbose; ENDIF

    strMsgSend := strMsgSend;
    strMsgRecv := alSendRcvMsg(strMsgSend, \bVerbose:=bVerb);
ENDPROC
```

Slika 38. Prikaz procedure *alConAutoExsposure()*

Procedure *alSetFolderName()* (Slika 39), *alSetProjectName(), alScanStoreSurfaceImage(), alScanAndStoreSurface(), alScanStoreSurfaceVCImage()* uz string koji označava izvođenje naredbe šalju i string, koji definira naziv projekta, direktorija, slike ili 3D modela mjerne površine te naziv slike ispravka vinetiranja, potrebnog kao ulazni argument određenih funkcija definiranih unutar poslužitelja. Programski kod navedenih procedura definiran je na isti način te je kao primjer prikazana procedura *alSetFolderName()*.

```
!Set folder name in alicona server
PROC alSetFolderName(string FolderName, \bool Verbose)
    VAR bool bVerb;                 !Enable/disable verbose logging to TP
    VAR string strMsgSend := "";
    VAR string strMsgRecv := "";
    IF Present (Verbose) THEN bVerb := Verbose; ELSE bVerb := FALSE; ENDIF

    strMsgSend :="SFN" + FolderName;
    strMsgRecv := alSendRcvMsg(strMsgSend, \bVerbose:=bVerb);
ENDPROC
```

Slika 39. Prikaz procedure *alSetFolderName()*

Unutar procedure *alIterativeAutoFocus()* (Slika 40) zadaje se string koji označava izvođenje automatskog fokusiranja uređaja te se zadaje gornja i donja granica skeniranja. Prije izvođenja fokusiranja korištenjem *InPos* argumenta naredbe *WaitRob,* robot mora doći u zadanu poziciju prije nastavka izvođenja programa. Argument *ZeroSpeed* iste naredbe zaustavlja robot i sve njegove osi prije nastavka izvođenja programa. Naredbom *WaitTime* definirano je vrijeme smirivanja robota nakon pozicioniranja što znači da će robot prije slanja zahtjeva za izvršavanje fokusiranja i zadavanje granica skeniranja pričekati još 1 sekundu kako bi mjerenje uzorka bilo što točnije. Granice skeniranja (*UpperPosRel* i *LowerPosRel*) argumenti su procedure te ih je naredbom *NumToStr()* potrebno pretvoriti u string kako bi se mogli poslati poslužitelju.

```
PROC alIterativeAutoFocus(\num settleTime, \num UpperPosRel, \num LowerPosRel, \bool Verbose)
    VAR bool bVerb := FALSE;
    VAR num numSettleTime := 5.0;
    VAR num  numUpperPosRel := 0.5;
    VAR num  numLowerPosRel := -0.5;
    VAR string strMsgSend := "";
    VAR string strMsgRecv := "";


    IF Present (settleTime) THEN numSettleTime := settleTime; ENDIF
    IF Present (UpperPosRel) THEN numUpperPosRel := UpperPosRel; ENDIF
    IF Present (UpperPosRel) THEN numLowerPosRel := LowerPosRel; ENDIF
    IF Present (Verbose) THEN bVerb := Verbose; ENDIF

    !Cekanje da robot dodje u poziciju i da se smiri
    WaitRob \InPos;
    WaitRob \ZeroSpeed;
    WaitTime numSettleTime;

    strMsgSend :="IAF";
    strMsgRecv := alSendRcvMsg(strMsgSend, \bVerbose:=bVerb);

    strMsgSend :="SUSPR" + NumToStr(numUpperPosRel,6);
    strMsgRecv := alSendRcvMsg(strMsgSend, \bVerbose:=bVerb);

    strMsgSend :="SLSPR" + NumToStr(numLowerPosRel,6);
    strMsgRecv := alSendRcvMsg(strMsgSend, \bVerbose:=bVerb);

ENDPROC
```

Slika 40. Prikaz procedure *alIterativeAutoFocus()*

Unutar procedure *alSendCurrentTCP()* (Slika 41) dohvaćaju se TCP koordinate alata pomoću naredbe *CRobT()* te se pridružuju varijabli ptCurrentPos tipa podataka *robtarget*. Nakon smirivanja robota šalju se koordinate translacijskih x,y,z osi i vrijednosti Eulerovih kutova rotacijskih osi u koordinatnom sustavu objekta. U programskom kodu na strani računala

definirana je jedinična matrica WTM kojoj se pridružuju vrijednosti TCP koordinata kako bi se unutar upravljačkog softvera mjernog uređaja definirao položaj i orijentacija mjernog uređaja s obzirom na koordinatni sustav objekta.

```
!Send active TCP data to alicona server
PROC alSendCurrentTCP(\num settleTime, \bool swapXY, \bool Verbose)
    VAR string strMsgSend := "";
    VAR string strMsgRecv := "";
    VAR robtarget ptCurrentPos;
    VAR num numSettleTime := 5.0;
    VAR bool bRet := FALSE;
    VAR bool bVerb :=FALSE;                !Enable/disable verbose logging to TP
    VAR bool bSwapXY := FALSE;

    IF Present (settleTime) THEN numSettleTime := settleTime; ENDIF
    IF Present (Verbose) THEN bVerb := Verbose; ENDIF
    IF Present (swapXY) THEN bSwapXY := swapXY; ENDIF


    !Cekanje da robot dodje u poziciju i da se smiri
    WaitRob \InPos;
    WaitRob \ZeroSpeed;
    WaitTime numSettleTime;

    !Dohvacanje pozicije
    ptCurrentPos := CRobT();


    !Slanje koordinate tcp-a
    IF NOT bSwapXY THEN

        strMsgSend :="STCTX" + NumToStr(ptCurrentPos.trans.x,3);
        strMsgRecv := alSendRcvMsg(strMsgSend, \bVerbose:=bVerb);

        strMsgSend :="STCTY" + NumToStr(ptCurrentPos.trans.y,3);
        strMsgRecv := alSendRcvMsg(strMsgSend, \bVerbose:=bVerb);

    ELSE

        strMsgSend :="STCTX" + NumToStr(ptCurrentPos.trans.y,3);
        strMsgRecv := alSendRcvMsg(strMsgSend, \bVerbose:=bVerb);

        strMsgSend :="STCTY" + NumToStr(ptCurrentPos.trans.x,3);
        strMsgRecv := alSendRcvMsg(strMsgSend, \bVerbose:=bVerb);

    ENDIF


        strMsgSend :="STCTZ" + NumToStr(ptCurrentPos.trans.z,3);
     strMsgRecv := alSendRcvMsg(strMsgSend, \bVerbose:=bVerb);

   IF NOT bSwapXY THEN

   strMsgSend :="STCRX" + NumToStr(EulerZYX(\X, ptCurrentPos.rot),6);
   strMsgRecv := alSendRcvMsg(strMsgSend, \bVerbose:=bVerb);

   strMsgSend :="STCRY" + NumToStr(EulerZYX(\Y, ptCurrentPos.rot),6);
   strMsgRecv := alSendRcvMsg(strMsgSend, \bVerbose:=bVerb);

   ELSE

       strMsgSend :="STCRX" + NumToStr(EulerZYX(\Y, ptCurrentPos.rot),6);
       strMsgRecv := alSendRcvMsg(strMsgSend, \bVerbose:=bVerb);

       strMsgSend :="STCRY" + NumToStr(EulerZYX(\X, ptCurrentPos.rot),6);
       strMsgRecv := alSendRcvMsg(strMsgSend, \bVerbose:=bVerb);

   ENDIF


   strMsgSend :="STCRZ" + NumToStr(EulerZYX(\Z, ptCurrentPos.rot),6);
   strMsgRecv := alSendRcvMsg(strMsgSend, \bVerbose:=bVerb);

ENDPROC
```

Slika 41. Prikaz procedure *alSendCurrentTCP()*

U proceduri *alSendTCPScanAndStoreSurface()* (Slika 42) prvo se poziva procedura kojom se šalju TCP koordinate, a zatim procedura koja šalje naredbu za skeniranje i pohranjivanje mjerenih podataka površine.

Pozivom procedure *alSetDBdata()*, prvo se izvršava procedura kojom se zadaje naziv direktorija *alSetFolderName()*, a zatim procedura kojom se zadaje naziv projekta *alSetProjectName()* u koje će se pohraniti rezultati mjerenja. Navedene procedure definirane su kako bi se pojednostavnio i skratio programski kod čijim će se pozivanjem izvršiti mjerenje površine (navedena rutina bit će opisana u idućem poglavlju).

```
PROC alSendTCPScanAndStoreSurface(\string SurfName, \num settleTime, \bool Verbose)
    VAR num numSettleTime := 5.0;
    VAR bool bVerb := FALSE;                !Enable/disable verbose logging to TP

    IF Present (Verbose) THEN bVerb := Verbose;  ENDIF

    IF Present (settleTime) THEN
        alSendCurrentTCP \settleTime:=settleTime, \Verbose:=bVerb;
    ELSE
        alSendCurrentTCP \Verbose:=bVerb;
    ENDIF

    IF Present (SurfName) THEN
        alScanAndStoreSurface \SurfName:=SurfName, \Verbose:=bVerb;
    ELSE
        alScanAndStoreSurface \Verbose:=bVerb;
    ENDIF

ENDPROC
```

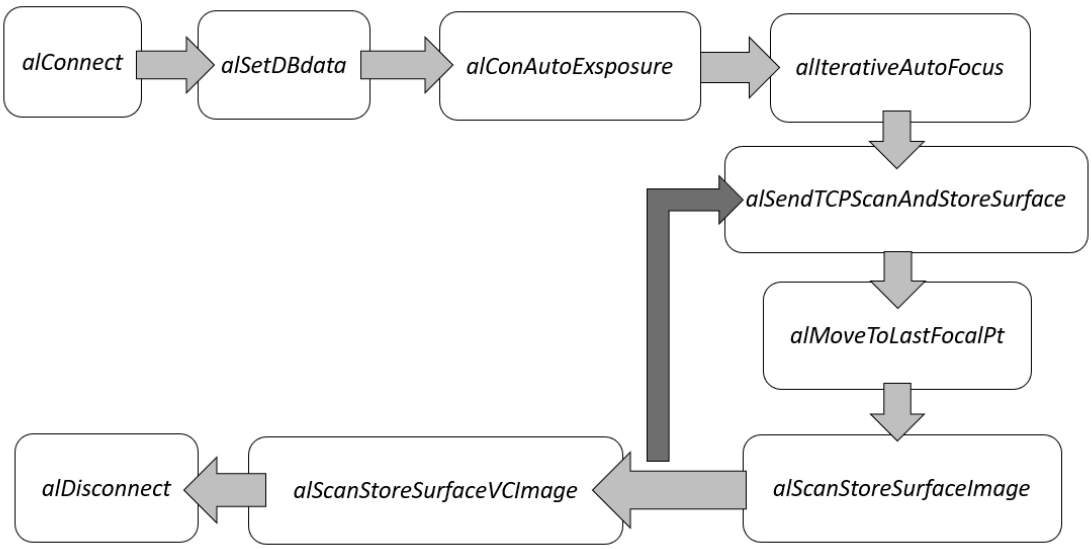
Slika 42. Prikaz procedure *alSendTCPScanAndStoreSurface()*

5. AUTOMATIZIRANI PROCES MJERENJA

Kako bi se postupak skeniranja mogao izvršiti izravno iz korisničkog programa robota potrebno je definirati rutinu odnosno program skeniranja koji će pozivanjem komunikacijskih i izvršnih procedura slijediti iduće korake:

1. povezivanje s računalom, odnosno poslužiteljem
2. zadavanje naziva direktorija i projekta u koje će se spremati rezultati mjerenja
3. zadavanje automatskog namještanja ekspozicije senzora
4. fokusiranje mjernog uređaja te zadavanje raspona skeniranja po Z osi
5. slanje trenutnih koordinata TCP-a (engl. *Tool center point*) i slanje zahtjeva za skeniranje i pohranjivanje modela površine
6. dovođenje uređaja u prethodnu spremljenu žarišnu točku
7. slanje zahtjeva za skeniranje i pohranjivanje slike površine
8. pomicanje robota te ponavljanje koraka 5,6, i 7
   Ovaj korak se ponavlja ovisno o veličini skenirane površine. Površina se skenira dio po dio te se spremaju njene slike i podatci koji se na kraju spajaju u cjelokupni 3D model površine.
9. skeniranje i pohranjivanje slike ispravka vinjetiranja
10. prekidanje veze s poslužiteljem.

Dijagram toka pozivanja procedura kojima se izvršavaju navedeni koraci prikazan je slikom 43.



Slika 43. Dijagram toka pozivanja procedura

Procedura *AliconaTestRun4()* definirana je unutar korisničkog modula *srmTestAuxR2.mod* (Slika 44) gdje su na početku modula definirani korisnički koordinatni sustav i koordinatni sustav objekta tipom podataka *wobjdata.*

```
MODULE srmTestAuxR2(SYSMODULE)
    CONST robtarget ptAl50xR2TestScanPos:=[[-1191.51,-1113.17,724.37],[0.0698555,0.698292,0.0605938,-0.709814],[0,0,-3,6],[9E+09,
    !TASK PERS wobjdata wobj_AlTest:=[FALSE,TRUE,"",[[1862.26,-764.318,815.244],[0.70711,0.0,0.0,-0.70711]],[[0,0,0],[1,0,0,0]]];
    TASK PERS wobjdata wobj_AlTest:=[FALSE,TRUE,"",[[1862.26,-764.318,815.244],[0.70711,0.0,0.0,0.70711]],[[0,0,0],[1,0,0,0]]];
```

Slika 44. Definiranje koordinatnog sustava

Na početku procedure definirane su varijable tipa *num* koje označavaju pozicije i posmake po x i y osi te varijable tipa *string* potrebne za zadavanje naziva i spremanje slika i modela skeniranih površina (Slika 45). Naredbom *AccSet* postavlja se akceleracija gibanja robota, dok se naredbama *MoveJ* i *MoveL* uređaj dovodi u položaj za skeniranje. Ako nema nikakve greške u kodu, rutina nastavlja izvršavati odnosno pozivati funkcije kojima se uspostavlja konekcija s poslužiteljem i slanje podataka poslužitelju. Argumenti izvršnih i komunikacijskih procedura koji se zadaju u programu su ime foldera, projekta, modela površine i slike površine te raspon skeniranja po Z osi i vrijeme smirivanja robota između pojedinih operacija.

```
PROC AliconaTestRun4()
    VAR num posXmin := -29;
    VAR num posXmax := -27;
    VAR num posYmin := 25;
    VAR num posYmax := 27;
    VAR num stepPosX := 1.0;
    VAR num stepPosY := 1.0;

    VAR num pX :=0;
    VAR num py :=0;
    VAR string surfNamePrefix := "A";
    VAR string imageNamePrefix := "E";
    VAR string surfName :="";
    VAR string imageName := "";

    VAR BOOL Debug := FALSE;


    AccSet 10,20 \FinePointRamp:=5;
    MoveJ [[-15.00,30.00,50.00],[1,0,0,0],[-1,-3,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v200, fine, T2_Al10AX\WObj:=wobj_AlTest;


    MoveL [[posXmin,posYmin,0.00],[1,0,0,0],[-1,-3,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v50, fine, T2_Al10AX\WObj:=wobj_AlTest;

    IF NOT Debug THEN
        alConnect \Verbose:=TRUE;
        alSetDBData "TEMP", "REMOTING3", \Verbose:=TRUE;
        alConAutoExsposure \Verbose:=TRUE;

        alIterativeAutoFocus \settleTime:=5.0, \UpperPosRel:=0.8, \LowerPosRel:=-0.8;
```

Slika 45. Procedura *AliconaTestRun4()*

Definiranom petljom unutar rutine (Slika 46) provodi se skeniranje površine u 9 točaka, sa posmakom od 1mm.

```
    FOR i FROM posXmin TO posXmax STEP stepPosX DO
        pY := 0;
        FOR j FROM posYmin TO posYmax STEP stepPosY DO
            surfName := surfNamePrefix + "_" + numToStr(pX,0) + "_" + NumToStr(pY,0);
            imageName := imageNamePrefix + "_" + numToStr(pX,0) + "_" + NumToStr(pY,0);
            TPWrite "X="+ numToStr(i,3) + ", Y=" + NumToStr(j,3) + ", "+ surfName;

            MoveL [[i,j,0.00],[1,0,0,0],[-1,-3,2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]], v20, fine, T2_Al10AX\WObj:=wobj_AlTest;

            IF NOT Debug THEN
                alSendTCPScanAndStoreSurface \SurfName:=surfName, \settleTime:=5.0, \Verbose:=TRUE;
                alMoveToLastFocalPt;
                alScanStoreSurfaceImage \SurfName:=imageName, \Verbose:=TRUE;
            ELSE
                UIMsgBox "Please adjust focus position and make test scan X="+ numToStr(i,3) + ", Y=" + NumToStr(j,3) + ", "+ surfName;
            ENDIF

            pY := pY + 1;
        ENDFOR
        pX := pX + 1;
    ENDFOR
```

Slika 46. Prikaz procedure *AliconaTestRun4()*

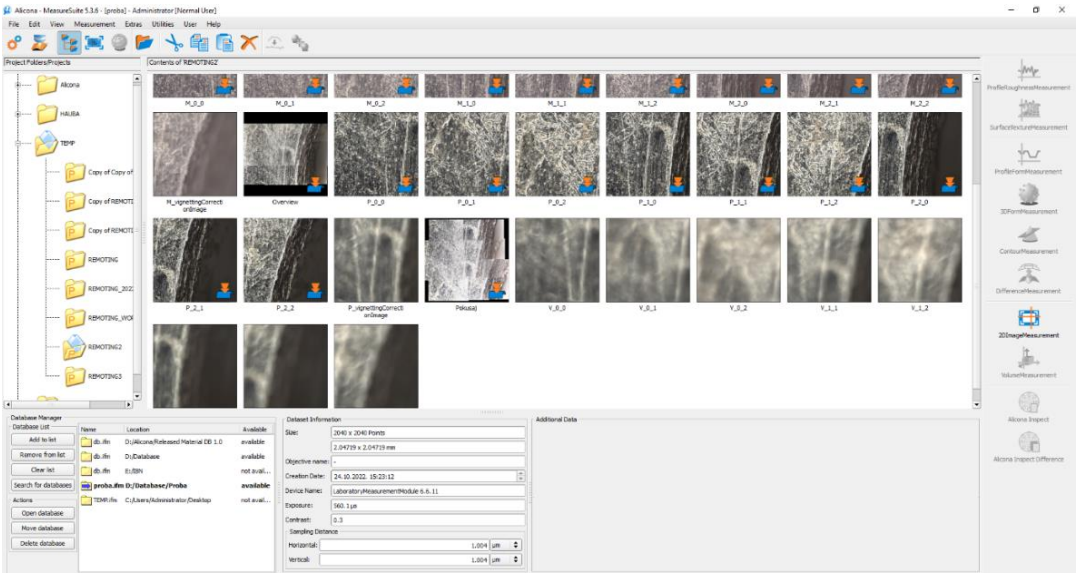
6. TESTIRANJE RADA SUSTAVA

Proces mjerenja provodi se pokretanjem poslužitelja na mjernom računalu, zatim pozivom rutine *AliconaTestRun4()* (opisane u prošlom poglavlju) na operatorskom panelu robota. Vođenje robota, odnosno pokretanje pripremljenog korisničkog programa, izvršeno je u ručnom načinu rada. Na operatorskom panelu se redom odabiru naredbe *Program Editor-Debug-PP to Routine-AliconatestRun4()* te se pritiskom tipke *play* pokreće izvođenje rutine. Tijekom izvođenja rutine potrebno je držati sigurnosno tipkalo na operatorskom panelu čijim se otpuštanjem prekida gibanje robota odnosno rad motora te se prekida izvođenje programa i komunikacija s poslužiteljem. Nakon pokretanja poslužitelja i uspostave konekcije s upravljačkim sustavom robota na mjernom računalu unutar Python sučelja ispisuju se metode koje se izvršavaju, poruke koje šalje klijent te odgovori poslužitelja (Slika 47).

```
ABBAliconaSocketServ.startAliconaClient: Starting alicona client ...
ABBAliconaSocketServ.startSocketServ: Starting socket server ...
COMMANDS:
   [h] - print this message
   [p] - print threads
   [rs/ss] - start/stop SocketServer
   [ra/sa] - start/stop Alicona Client
   Any other key terminates program
   COMMAND:ABBAliconaSocketServ.__runSocketServ: Waiting for connection...
ABBAliconaSocketServ.__ProcessClRq: Connected from ('192.168.2.107', 56091)
ABBAliconaSocketServ.__runSocketServ: Waiting for connection...
ABBAliconaSocketServ.__ProcessClRq: Received from ('192.168.2.107', 56091) : 'b'SFNTEMP"
ABBAliconaSocketServ.__ProcessAlRq: SET FOLDER NAME: TEMP
ABBAliconaSocketServ.__ProcessAlRq: RESPONSE: ACK-SFNTEMP
ABBAliconaSocketServ.__ProcessClRq: Sending response to ('192.168.2.107', 56091) : 'ACK-SFNTEMP'
ABBAliconaSocketServ.__ProcessClRq: Received from ('192.168.2.107', 56091) : 'b'SPNREMOTING"
ABBAliconaSocketServ.__ProcessAlRq: SET PROJECT NAME: REMOTING
ABBAliconaSocketServ.__ProcessAlRq: RESPONSE: ACK-SPNREMOTING
ABBAliconaSocketServ.__ProcessClRq: Sending response to ('192.168.2.107', 56091) : 'ACK-SPNREMOTING'
ABBAliconaSocketServ.__ProcessClRq: Received from ('192.168.2.107', 56091) : 'b'IAF"
ABBAliconaSocketServ.__ProcessAlRq: ITERATIVE AUTOFOCUS
ABBAliconaSocketServ.__ProcessClRq: Sending response to ('192.168.2.107', 56091) : 'BSY-IAF'
ABBAliconaSocketServ.__ProcessClRq: Received from ('192.168.2.107', 56091) : 'b'SRQ"
ABBAliconaSocketServ.__ProcessClRq: Sending response to ('192.168.2.107', 56091) : 'BSY-IAF'
```

Slika 47. Ispis poslužitelja unutar Python sučelja

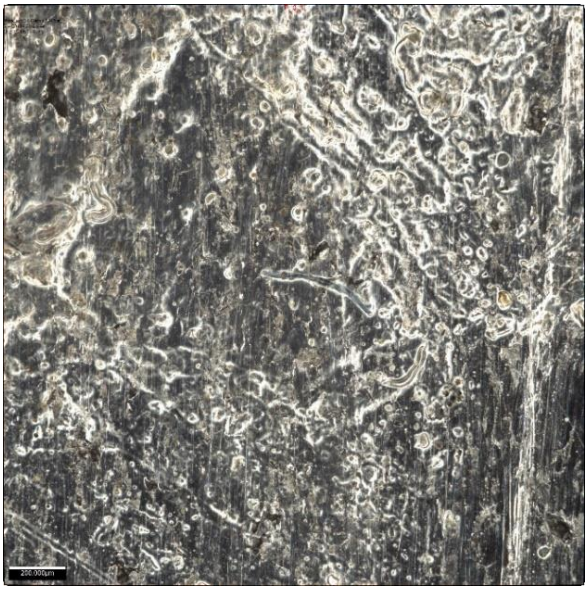
Primarni zadatak rada, uspostavljanje komunikacije između mjernog uređaja i robota, te izvođenje programa skeniranja uspješno je izvršen. Nakon skeniranja rezultati mjerenja spremljeni su u bazu podataka *LabaratoryMeasurement* modula (Slika 48).



Slika 48. Slika grafičkog korisničkog sučelja *LabaratoryMeasurement* modula

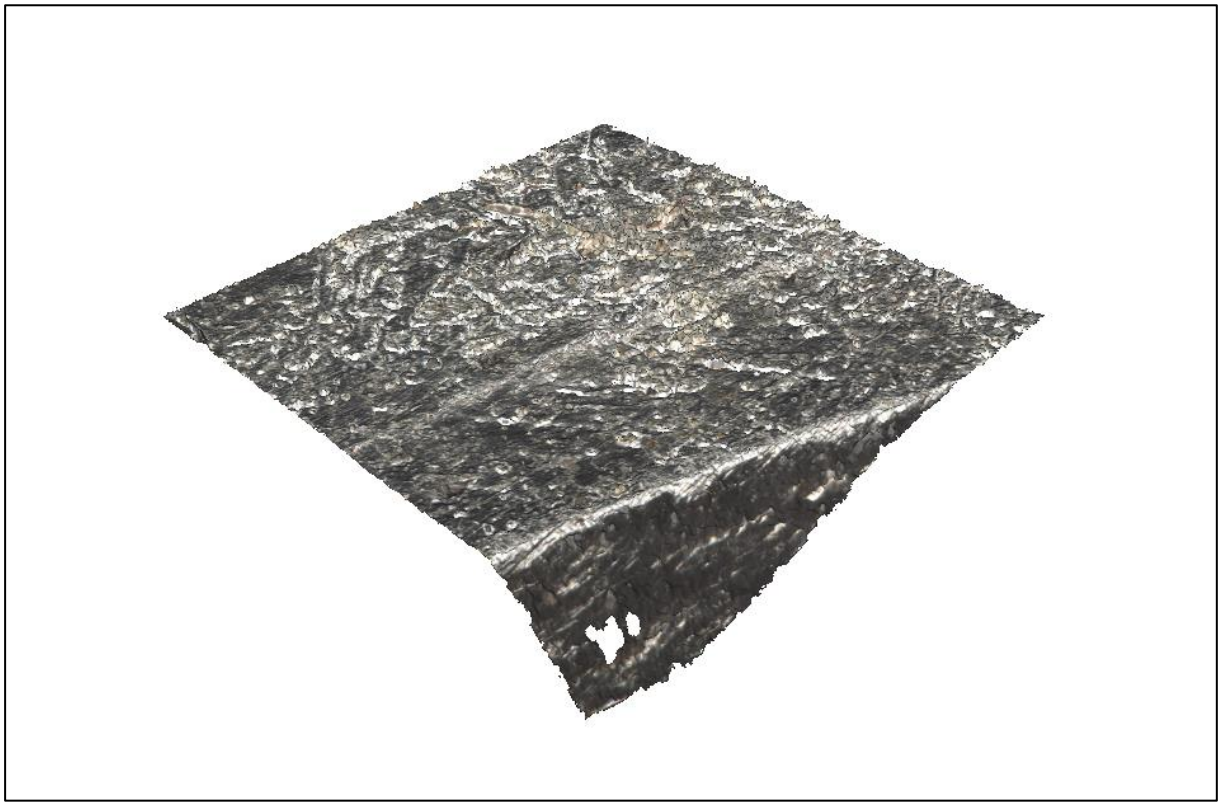
6.1. Rezultati mjerenja

Topologija i tekstura pojedinačnih skeniranih uzoraka površine gdje je ispitna površina ravna, jasno su vidljive (Slika 49 i Slika 50), dok na mjestima gdje je ispitna površina zakrivljena, odnosno ima skošenje, postoje odstupanja uzrokovana nedostatkom mjernih točaka odnosno podataka (Slika 51).



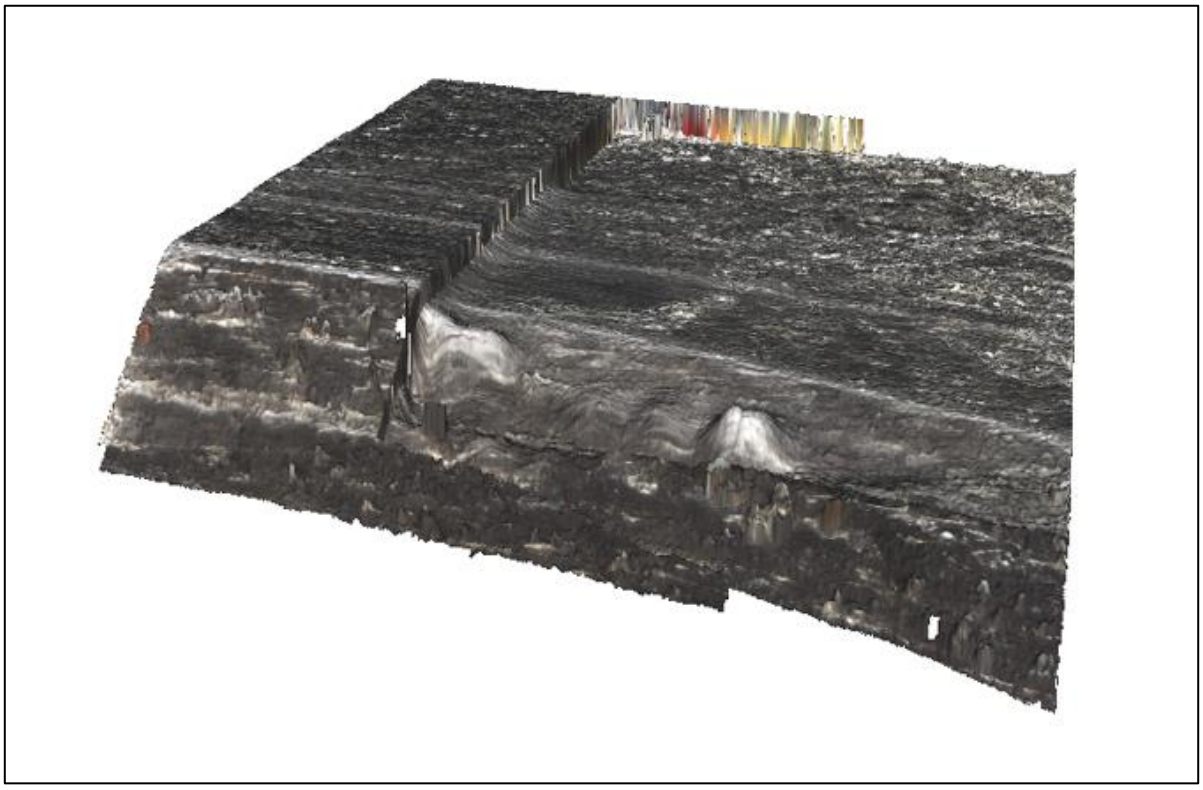
Slika 49. 2D prikaz pojedinačnog mjernog modela površine

Slika 50. 3D prikaz pojedinače skenirane površine



Slika 51. 3D prikaz pojedinačne skenirane površine

Testiranje rada sustava provedeno je na način da se ispitna površina skenirala po regijama površine 2 x 2 mm^2 na 9 pozicija i to na način da se susjedne površine preklapaju 50%. Nakon što je robot pomaknuo mjerni uređaj u svih 9 pozicija te je izvršeno skeniranje svih pojedinačnih regija razmatrane površine, na upravljačkom računalu poziva se skripta u kojoj su definirane metode pomoću koje se izvršava spajanje svih oblaka točaka. Slika 53 prikazuje cijelu skeniranu površinu.

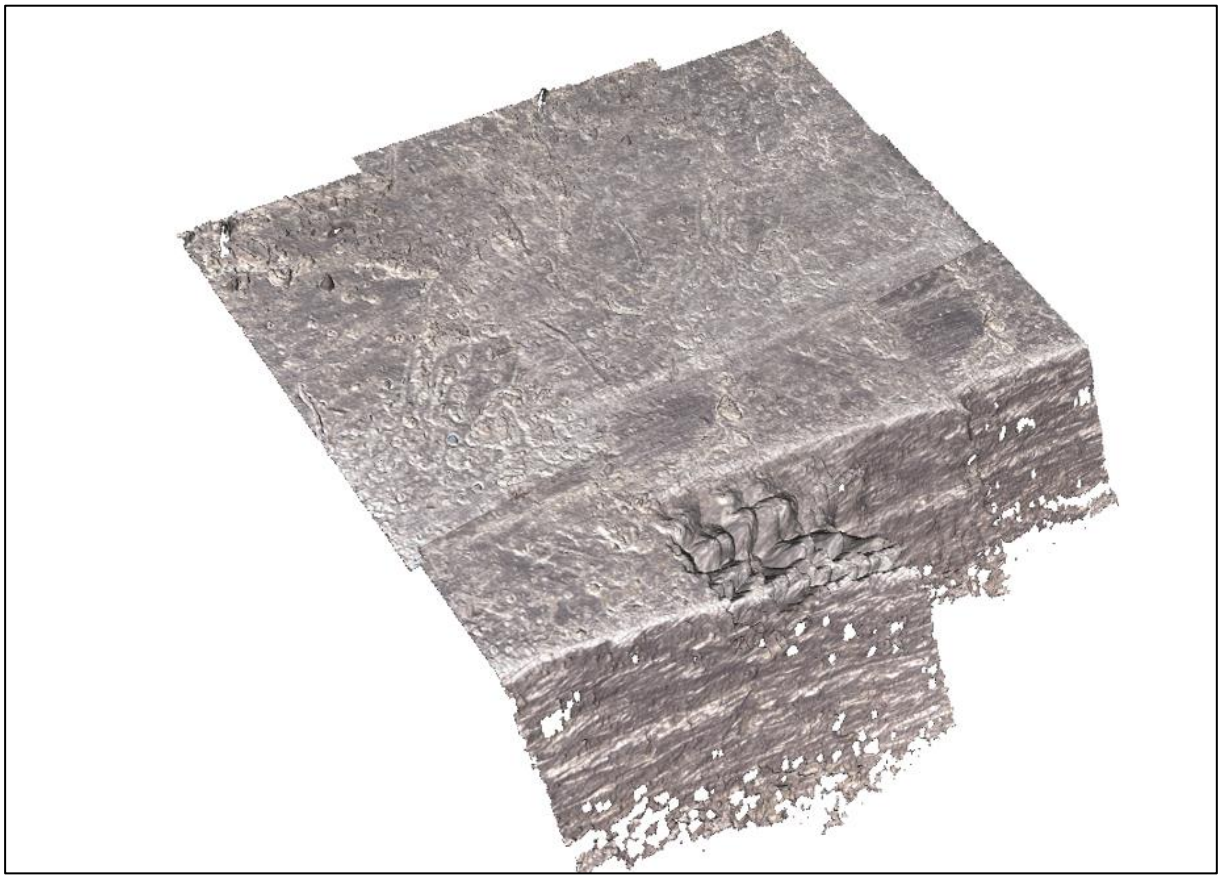


Slika 52. Prikaz cijelog mjernog površinskog modela

Uočljivo je da mjerni podaci nisu uspješno povezani, pogotovo spajanje modela po Z osi. Razlog tome mogu biti vibracije uzrokovane pomicanjem robota što utječe na točnost i ponovljivost pozicioniranja robota te neprilagođenost parametara potrebnih za spajanje površinskih modela.

Prije ponovnog mjerenja, prilagođeni su parametri u kodu za spajanje svih mjerenih površina na strani upravljačkog računala mjernog uređaja. Također, na strani robota, odnosno u rutini (*Aliconatestrun4()*) gdje se pozivaju funkcije i zadaju parametri skeniranja povećano je

vrijeme čekanja smirivanja robota između pojedinih operacija skeniranja kako bi robot nakon pomicanja stabilizirao čime bi se poboljšala kvaliteta skeniranja. U prvom mjerenju vrijeme čekanja smirivanja robota iznosilo je 0.3 sekundi dok u drugom mjerenju iznosi 0.5 sekundi. Rezultat drugog mjerenja prikazan je slikom 54.



Slika 53. Rezultat drugog skeniranja i spajanja površina

Rezultat je puno bolji od prethodnog, no i dalje postoje razlike odstupanja kod spajanja pojedinih oblaka točaka. U obzir treba uzeti i grešku pozicioniranja robota na koju se ne može utjecati. Također, na rubu skenirane ispitne površine nalazi se skošenje na kojem su vidljive najveće greške u spajanju oblaka točaka.

7. ZAKLJUČAK

U ovom radu opisan je sustav integracije beskontaktnog mjernog uređaja Alicona IF Sensor R25 s upravljačkom jedinicom robota ABB IRB4600. Razvijen je upravljački program koji omogućuje automatsko mjerenje ispitnog uzorka, što je ujedno bio zadatak ovog diplomskog rada.

U sklopu integracije bilo je potrebno uspostaviti komunikaciju između mjernog uređaja i robota. Komunikacijski model klijent-poslužitelj implementiran je pomoću modula pristupne točke (engl. *socket*) koji direktno koristi funkcionalnost transportnog sloja računalne mreže temeljene na TCP/IP protokolu. Klijent predstavlja program na strani upravljačkog sustava robota napisanog u programskom jeziku RAPID, dok poslužitelj predstavlja program implementiranog unutar upravljačkog računala mjernog uređaja pomoću programskog jezika Python. Razvijeni komunikacijski protokol odvija se na način da klijent šalje zahtjeve za povezivanje i izvršenje naredbi mjerenja poslužitelju. Uz naredbe definirani su i prefiksi poruka kojima se provjera stanje izvršenja pojedinih zahtjeva i ostvaruje kontinuirana komunikacija klijenta i poslužitelja, odnosno robota i mjernog uređaja.

Upravljački program testiran je izvođenjem probnog mjerenja ispitne površine te su prikazani dobiveni mjereni rezultati koje je naknadno moguće analizirati i uređivati pomoću softvera mjernog uređaja. Uspješno je izvršena uspostava komunikacije i razmjena podataka između mjernog uređaja i robota kojima se vrši automatsko mjerenje.

Softver mjernog uređaja nudi mnoštvo opcija i metoda kojima bi se prilagodili parametri analize i spajanja mjernih podataka. Stoga se postojeće rješenje, odnosno upravljački program nastao u okviru ovog diplomskog rada, može nastaviti razvijati i nadograđivati u vidu daljnje automatizacije analize mjernih podataka.

LITERATURA

[1] The magazine about Alicona-metrology: https://www.alicona.com/magazine-focus-variation-2020/, Pristupljeno rujan 2022..

[2] Santoso, T., Syam, Wahyudin P., Darukumalli, On-machine focus variation measurement for micro-scale hybrid surface texture machining. Int J Adv Manuf Technol 109, 2353–2364 (2020). https://doi.org/10.1007/s00170-020-05767-z

[3] RemotingInterface_6.6.12_Manual_EN.pdf., Preuzeto u rujnu 2022.

[4] ABB Product specification - IRB 4600, pristupljeno studeni 2022.

[5] ABB Operating manual, Introduction to RAPID, RobotWare 5.0, pristupljeno studeni 2022.

[6] Technical reference manual - RAPID overview - ABB, pristupljeno studeni 2022

[7] Technical reference manual RAPID Instructions, Functions and Data types, pristupljeno studeni 2022.

[8] Staroveški, T. Predavanje CNC upravljački sustavi,Fakultet strojarstva i brodogradnje

[9] https://www.phy.pmf.unizg.hr/~dandroic/nastava/ramr/poglavlje_3.pdf , Prisupljeno prosinac 2022.

[10] https://www.fer.unizg.hr/_download/repository/Rassus-2013_knjiga_v_1_0.pdf

[11] https://realpython.com/python-sockets/, Pristupljeno rujan 2022.

[12] https://vojni.unizg.hr/_download/repository/6_pred_OsnoveMreznogProgramiranja.pdf.

PRILOZI

I. CD-R disk