Integracija beskontaktnog uređaja za digitalizaciju oblika i mjerenje topologije površine

Požgaj, Mislav

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mislav Požgaj

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor*:*

Dr. sc. Tomislav Staroveški, dipl. ing.

Student:

Mislav Požgaj

Zagreb, 2023.

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

Zahvaljujem mentoru dr. sc. Tomislavu Staroveškom na ustupljenoj opremi, stručnoj pomoći i savjetima tijekom izrade rada. Zahvaljujem i ostalim djelatnicima Katedre za alatne strojeve na pomoći i savjetima tijekom izrade rada.

Također zahvaljujem obitelji i prijateljima koji su me podržavali tijekom cijelog razdoblja studija.

Mislav Požgaj

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za diplomske ispite studija strojarstva za smjerove:
Proizvodno inženjerstvo, inženjerstvo materijala, industrijsko inženjerstvo i menadžment, mehatronika i robotika, autonomni sustavi i računalna inteligencija

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

DIPLOMSKI ZADATAK

Student: **Mislav Požgaj** JMBAG: 0035213874

Naslov rada na hrvatskom jeziku: **Integracija beskontaktnog uređaja za digitalizaciju oblika i mjerenje topologije površine**

Naslov rada na engleskom jeziku: **Integration of optical 3D scanning instrument**

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 optički mjerni uređaj tipa ATOS 5x (GOM GmbH, Njemačka), koji je ugrađen na industrijski robot IRB4600 (ABB, Švicarska) u cilju realizacije sustava za digitalizaciju obrađenih površina ispitnih uzoraka. Idući korak u realizaciji ispitnog postava je integracija mjernog uređaja s upravljačkim sustavom robota.

Stoga je u radu je potrebno:

1. Opisati optički mjerni uređaj ATOS 5x s osvrtom na programsku podršku za upravljanje predmetnim mjernim uređajem izravno iz upravljačkog sustava robota, koja se trenutno razvija u sklopu projektnih aktivnosti.
2. Nadograditi postojeću programsku podršku za upravljanje predmetnim mjernim uređajem podrškom za automatsku kalibraciju više različitih mjernih volumena.
3. Predložiti idejno konstrukcijsko rješenje naprave za prihvat kalibracijskih ploča i izraditi 3D model naprave s pripadajućim sklopnim i radioničkim crtežima. Naprava treba biti prilagođena postojećoj konstrukciji robotske ćelije tako da se može ugraditi na lokaciju iz koje će biti moguće provoditi umjeravanje više različitih mjernih volumena uz minimalnu mogućnost kolizije.
4. Napraviti simulaciju procesa umjeravanja primjenom simulacijskog softvera RoboDK i/ili RobotStudio.
5. Dati zaključke rada.

U radu je potrebno navesti eventualno dobivenu pomoć pri izradi rada, a u popisu literature navesti svu korištenu literaturu.

Zadatak zadan:	Datum predaje rada:	Predviđeni datumi obrane:
4. svibnja 2023.	6. srpnja 2023.	17. – 21. srpnja 2023.

Zadatak zadao:

Izv.prof. dr. sc. Tomislav Staroveški

Predsjednik Povjerenstva:

Prof. dr. sc. Ivica Garašić

Sadržaj

Popis slika

Popis tablica

Popis oznaka:

Oznaka	Mjerna jedinica	Opis
x	*mm*	pomak u x smjeru
y	*mm*	pomak u y smjeru
z	*mm*	pomak u z smjeru
u	°	rotacija oko x osi
v	°	rotacija oko y osi
w	°	rotacija oko z os

Popis kratica

DLP *Digital Light Processing* – digitalna obrada svjetla

2D dvodimenzionalni

3D trodimenzionalni

TCP *Transmission Control Protocol* – transportni protokol

IP *Internet Protocol* – Internet protokol

TCP *Tool Center Point* – vrh alata robota

Sažetak

U ovom je radu opisan proces integracije beskontaktnog uređaja za digitalizaciju oblika i mjerenje topologije površine, Atos 5X, s robotom ABB IRB4600. Prikazano je programsko rješenje za komunikaciju mjernog uređaja s upravljačkim sustavom robota pomoću računalne mreže temeljene na TCP/IP protokolu. Programski kod koji se izvršava na mjernom uređaju napisan je u programskom jeziku Python, dok je programski kod koji se izvršava na robotu napisan u programskom jeziku RAPID. U drugom dijelu rada prikazano je predloženo rješenje automatizacije procesa kalibracije mjernog uređaja Atos 5X. Predloženo je mjesto za pozicioniranje kalibracijskih ploča različitih veličina unutar postojeće robotske ćelije, a izrađeno je i idejno konstrukcijsko rješenje naprave za prihvat kalibracijskih ploča. Izrađene su simulacije u dvjema aplikacijama, RoboDK i RobotStudio, kojima je izrađena analiza dohvatljivosti robota i testiranje programskog koda. Na kraju rada prokomentirani su dobiveni rezultati programskog rješenja i doneseni zaključci .

Ključne riječi: Atos 5X, integracija, kalibracija, ABB IRC5

Summary

This paper describes the integration of the GOM Atos 5X measurement device with the ABB IRB 4600 robot. An overview of the software solution developed for communication between the measurment device and the robot using a computer network based on the TCP/IP protocol is given. The program code to be run on the measurement device was written in the Python programming language, while the program code to be run on the robot was written in the RAPID programming language. In the second part of the paper, the proposed solution for automating the calibration process of the Atos 5X measuring device is presented. A place for positioning calibration plates of various sizes within the existing robotic cell is presented, and a conceptual design of a holding device for the calibration plates is proposed. A simulation is performed in two applications, RoboDK and RobotStudio, which were used for a reachabillity analysis and testing of the program code. Finally, the obtained results of the software solution and the conclusions reached are discussed.

Keywords: Atos 5X, integration, calibration, ABB IRC5

1. Uvod

Rad je izrađen na opremi nabavljenoj u sklopu projekta ARCOPS – „Autonomni robotski sustav za brušenje i karakterizaciju površina tankostijenih kompozitnih proizvoda. Cilj projekta je razvoj robotske obradne ćelije za brušenje i poliranje tankostijenih kompozitnih struktura s mogućnošću izravnog i posrednog nadzora procesa. U sklopu projekta razvijena je robotska ćelija opremljena senzorima za praćenje velikog broja parametara procesa potrebnih za razvoj modela izravnog nadzornog sustava. Jedan od ugrađenih elemenata za izravno praćenje procesa je optički mjerni uređaj Atos 5X (eng. *Advanced Topometric Optical Sensor*).

Zadatak je ovog rada razvoj programske podrške za automatizaciju procesa kalibracije optičkog mjernog uređaja Atos 5X. Automatizacija procesa ostvarena je razvojem programskog koda pomoću kojeg se uspostavlja komunikacija između mjernog uređaja Atos 5X i upravljačkog sustava robota ABB IRB4600. Programsko rješenje izrađeno je pomoću programskih jezika Python i RAPID. Usto je određena pozicija kalibracijskih ploča unutar postojeće robotske ćelije nizom simulacija pozicioniranja, a izrađen je i koncept naprave za prihvat kalibracijskih ploča. Konačno, izrađena je simulacija dobivenog rješenja kako bi se provjerila njegova funkcionalnost.

Slika 1 prikazuje mjerni uređaj Atos 5X u robotskoj ćeliji ARCOPS.

Slika 1 Atos 5X unutar ćelije ARCOPS

2. Optički mjerni uređaj Atos 5X

2.1. O optičkom uređaju Atos 5X

Atos 5X je 3D optički mjerni uređaj koji koristi tehnologiju strukturiranog svjetla za digitalizaciju razmatranog objekta. Mjerna glava uređaja Atos 5X sastoji se od dvije kamere visoke rezolucije (12 mega piksela) i DLP (eng. *Digital Light Processing*) projektora. Kao izvor svijetlosti se koristi digitalni projektor s laserskim izvorom valne duljine 440 *nm* do 470 *nm,* koji generira strukturirani svjetlosni uzorak. Plavo svjetlo ima jednu od najkraćih valnih duljina te je plavo svjetlo rijetko prisutno u okolišu. Radi toga je postupak digitalizacije plavim svjetlom manje osjetljiv na utjecaj okolišne rasvjete.

Dostupni mjerni volumeni navedenog uređaja su: 320 x 240 x 240 mm, 500 x 370 x 370 mm, 700 x 530 x 530 mm i 1000 x 750 x 750 mm[1], a isti se zamjenjuju ručno, mijenjanjem para objektiva kamera i objektiva projektora. Slika 2 prikazuje optički mjerni uređaj Atos 5X.

Slika 2 Optički mjerni uređaj Atos 5X[1]

2.2. Kalibracija mjernog uređaja

U nastavku je opisan postupak kalibracije mjernog uređaja Atos 5X. Prilikom promjene mjernog volumena potrebno je provesti novu kalibraciju te je preporučljivo kalibraciju provesti prije svakog skeniranja radi utjecaja temperature na mjerenja.

Na slici 3 vidljivi su dijelovi objektiva kamera i projektora koji određuju mjerni volumen mjernog uređaja Atos 5X.

Slika 3 Objektiv mjernog volumena MV500[1]

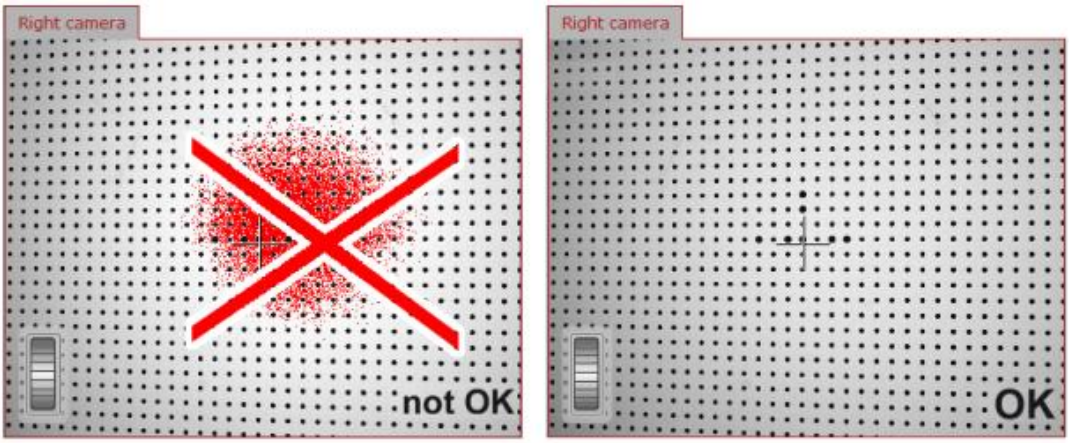
Dijelovi su:

1. objektiv lijeve kamere
2. objektiv projektora
3. objektiv desne kamere
4. prsten za podešavanje otvora blende
5. prsten za podešavanje žarišne duljine
6. navoj za pričvršćivanje na kućište mjerne glave uređaja Atos 5X
7. naziv objektiva.

Za kalibraciju se koriste kalibracijske ploče. Kalibracijska ploča sastoji se od točaka u rasteru pomoću kojih softver u tijeku kalibracije kompenzira distorziju objektiva kamera. Kako bi se kalibracija mogla provesti, potrebno je zadovoljiti nekoliko uvjeta. Neki su od tih uvjeta puštanje da se mjerni uređaj zagrije nekoliko minuta kako bi se postigla radna temperatura te pravilno postavljanje objektiva. Slika 4 prikazuje kalibracijsku ploču za mjerni volumen MV500.

Slika 4 Kalibracijska ploča MV500[1]

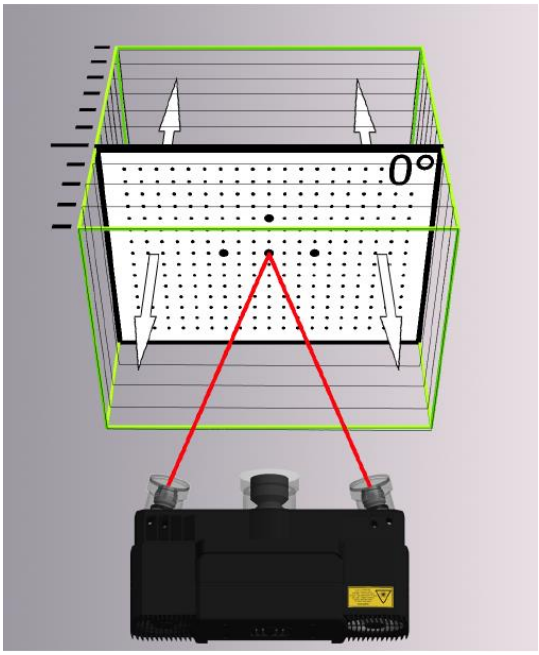
Prvi je korak u kalibraciji mjernog volumena određivanje temperature kalibracijske ploče kako bi se u kalibraciju uračunao utjecaj temperature. Vrijeme ekspozicije softver postavlja automatski uz mogućnost ručne korekcije. Kako je prikazano na slici 5, softver prikazuje ispravnu razinu ekspozicije. Crvene regije na slici označavaju preveliko vrijeme ekspozicije objektiva.



Slika 5 Namještanje vremena ekspozicije objektiva[1]

Jednom kad se namjesti temperatura kalibracijske ploče, softver korisnika vodi kroz proces kalibracije koji se sastoji od osamnaest pozicija u koje se mora dovesti mjerna glava. Prvih dvanaest pozicija kalibracije sastoji se od udaljavanja, odnosno približavanja mjerne glave u softverski određenim intervalima (37 *mm*) u odnosu na kalibracijsku ploču. U prvom koraku potrebno je postaviti žarišta objektiva kamera u centar kalibracijske ploče, što olakšavaju dva

lasera na uređaju koji se sijeku na mjestu žarišta oba objektiva kamera. Drugi korak je udaljavanje mjerne glave za 220 *mm* od kalibracijske ploče. Slijedećih deset koraka se sastoji od približavanja mjerne glave kalibracijskoj ploči u intervalima od 37 *mm*. U zadnjih šest koraka kalibracije mjerna glava se treba orijentirati zakrenuta naspram kalibracijske ploče. Slika 6 prikazuje prvi korak kalibracije mjernog uređaja.



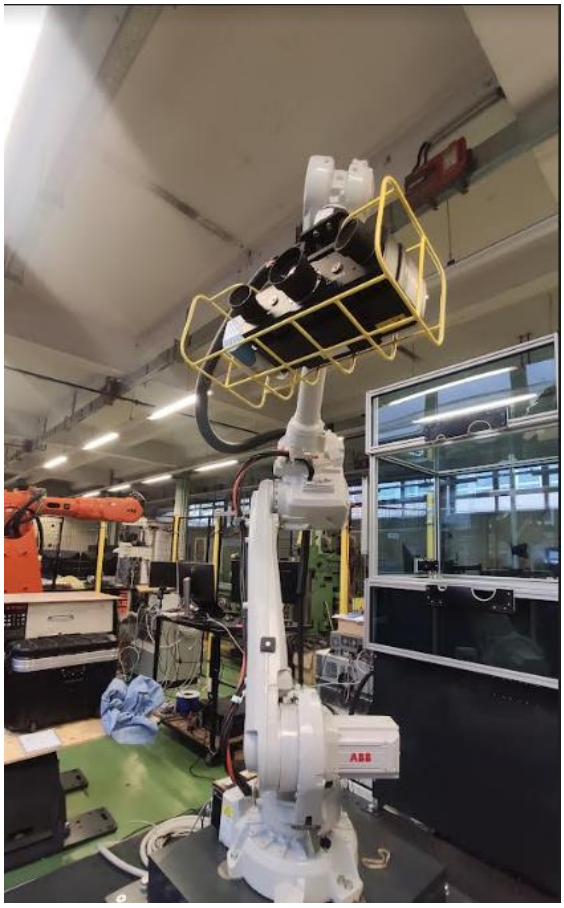
Slika 6 Prikaz prvog koraka kalibracije[1]

Na kraju kalibracije softver obavještava korisnika o uspješnosti kalibracije te sprema podatke o kalibraciji kako bi se mogli koristiti u mjernim protokolima.

3. Robotski sustav ABB IRB4600

3.1. Industrijski robot ABB IRB 4600-40/2.55

Za automatsko vođenje mjernog uređaja Atos 5X korišten je industrijski robot ABB IRB 4600-40/2.55. ABB IRB 4600-40/2.55 je industrijski robot sa šest stupnjeva slobode gibanja nosivosti 40 kg i dohvata 2,55 m. Robot s uređajem Atos 5X prikazan je na slici 7.



Slika 7 ABB IRB4600 s mjernim uređajem Atos 5X

Kao upravljački sustav robota koristi se ABB IRC5, koji pruža napredne mogućnosti upravljanja robotom uključujući funkcije programiranja pomoću programskog jezika RAPID. Kontroler IRC5 modularnog je tipa, što ga čini prilagodljivim različitim vrstama integracije i primjene [2]. Slika 8 prikazuje kontroler IRC5.

Slika 8 Kontroler IRC5[3]

3.2. Koordinatni sustavi robota

Koordinatni sustavi definirani u softveru kontrolera robotskog sustava služe za programiranje robota. Osnovni koordinatni sustav (eng. *World coordinate system*) služi za definiciju cijele robotske ćelije i svi se ostali koordinatni sustavi definiraju u odnosu na njega. Bazni je koordinatni sustav (eng. *Base coordinate system*) koordinatni sustav baze robota, odnosno on je fiksiran na bazu samog robota, a definiran je s obzirom na osnovni koordinatni sustav. Koordinatni sustav alata TCP (eng. *Tool Center Point*) definira središnju točku alata [4]. U svakom robotu moguće je definirati više TCP-a, a oni se definiraju s obzirom na nulti koordinatni sustav alata koji je definiran na prirubnici robota. Za potrebe ovog rada vrh alata definiran je u sjecištu žarišta objektiva kamera mjerne glave. To omogućava programiranje kretnji robota tijekom kalibracije jer su sve točke kalibracije definirane pomoću udaljenosti ili kuta nagiba mjerne glave od kalibracijske ploče. Također, tako definiran vrh alata korišten je za programiranje kretnji robota tijekom skeniranja. Uz navedeno postoje koordinatni sustav objekta (eng. *Object coordinate system)* i korisnički koordinatni sustav (eng. *User coordinate system).* Pomoću njih moguće je brzo prilagoditi cijeli program promjenama položaja objekta. Kalibracijska ploča u ovom slučaju predstavlja jedan korisnički koordinatni sustav. Prilikom promjene pozicije kalibracijske ploče ne treba se mijenjati cijeli program, nego je dovoljno zamijeniti korisnički koordinatni sustav.

4. Automatizacija procesa kalibracije

4.1. Računalna mreža temeljena na TCP/IP paketu protokola

Razmjena podataka između robotskog upravljačkog sustava i softvera mjernog uređaja Atos 5X ostvarena je preko računalne mreže. Računalna mreža omogućuje komunikaciju i razmjenu podataka između korisnika na mreži. Slijed komunikacije se razdvaja na slojeve koji obavljaju određene procese. U internet komunikaciji uglavnom se razmatra pet slojeva, a to su: aplikacijski sloj, transportni sloj, mrežni sloj, podatkovni sloj i fizički sloj. Svaki sloj opisuje skup povezanih funkcija koje omogućuju jedan dio računalne komunikacije.

Upravljački sustav robota i softver mjernog uređaja razmjenjuju podatke unutar aplikacijskog sloja, unutar sloja razgovora (eng. *session*). Sloj razgovora u ovom radu koristi TCP (eng. *Transmission Control Protocol*) protokol prijenosnog sloja i IP (eng. *Internet protocol*) protokol mrežnog sloja. Jedinice prijenosa podataka pomoću TCP protokola nazivaju se segmenti, a jedinice za prijenos podataka kod IP protokola nazivaju se paketi.

Zadatak TCP protokola je uspostavljanje i prekid veze. Usto potvrđuje valjanost primljenih segmenata i njihov redoslijed. Veza između dva računala definirana je parom priključaka (eng. *port*) (izvorišni i odredišni port). Port je numerička oznaka koja identificira određenu aplikaciju ili uslugu koja komunicira putem mreže na određenom računalu. Portovi se koriste kako bi se omogućilo višestruko korištenje iste IP adrese za različite vrste komunikacije. Portovi su 16-bitni brojevi, a njihov broj se kreće od 1 do 65535. Portovi se dijele u tri glavne kategorije: poznati portovi (1-1023), registrirani portovi (1024-49151) i dinamički/privremeni portovi (49152-65535).

Zadatak IP protokola je dostavljanje paketa od pošiljatelja do primatelja na osnovi IP adrese. IP adresa je jedinstvena numerička adresa koja se dodjeljuje uređajima u računalnoj mreži kako bi se omogućila identifikacija i adresiranje. IP adresa se sastoji od 32-bitnog binarnog broja koji se obično prikazuje u obliku četiri broja odvojenih točkama (npr. 192.168.0.1).

TCP/IP protokol se sastoji od tri faze.

Prva je faza TCP/IP protokola uspostavljanje veze između dviju računalnih mreža. Taj se proces naziva trostruko rukovanje (eng. *three-way handshake*). Prvo računalo (pošiljatelj) šalje sinkronizacijski paket (eng. *SYN*) drugom računalu (primatelju). Primatelj odgovara paketom *SYN-ACK*, čime potvrđuje pošiljatelju da je primio paket *SYN*. Na kraju pošiljatelj primatelju šalje

paket *ACK* kako bi potvrdio da je primio odgovor *SYN-ACK*. Kad primatelj prihvati paket *ACK*, veza je uspostavljena [5].

Druga je faza prijenos podataka. Jednom kad se uspostavi veza, pošiljatelj podatke šalje segmentirano (ovisno o duljini samog segmenta), a primatelj ih sastavlja i provjerava njihovu ispravnost. Svaki segment podataka ima svoj redni broj kako bi se osigurao ispravan redoslijed slanja i primanja. Svaki segment također sadržava sumu kontrolnih znamenki (eng. *checksum*) koja se koristi za provjeru cjelovitosti podataka [5].

Treća je i posljednja faza zatvaranje veze. Ovaj se proces sastoji od četverostrukog rukovanja (eng. *four-way handshake*). Prvo pošiljatelj šalje paket *FIN*, čime traži prekid veze. Primatelj mu odgovara paketom *ACK*. Nakon toga primatelj pošiljatelju šalje paket *FIN*, a pošiljatelj odgovara paketom *ACK*, čime se veza zatvara. Ako dođe do greške u razmjeni podataka i vrijeme čekanja na slanje ili primanje paketa pređe dozvoljeno vrijeme, veza se automatski prekida te dolazi do pokušaja ponovne uspostave veze [5].

4.2. Korišteni moduli programskog jezika Python

Python je programski jezik koji 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. Podržava modul pristupne točke (engl. socket). Radi toga je programski jezik Python odabran za izradu ovog rada. Usto GOM softver pomoću kojeg je moguća automatizacija rada mjernog uređaja podržava korištenje programskog jezika Python. Radi toga je kod koji radi na strani mjernog uređaja pisan u programskom jeziku Python.

Python *socket* ugrađeni je modul koji omogućava razvoj mrežne aplikacije za komunikaciju i razmjenu podataka preko mreže. Klijenti i poslužitelji su aplikacije ili procesi koji se izvode lokalno na jednom računalu ili udaljeno kroz računalnu mrežu. Klijent šalje zahtjev za uslugom poslužitelju i čeka odgovor. Poslužitelj prima i obrađuje dolazne zahtjeve klijenta te šalje odgovor klijentima. Metoda rada modula sastoji se od triju koraka: stvaranja *socketa*, povezivanja *socketa* s određenom adresom i portom te uspostave veze [6]. *Socket* predstavlja krajnje točke komunikacije između dva programa u računalnoj mreži. Klijenti i poslužitelji komuniciraju slanjem i primanjem tekstualnih nizova (engl. string) kroz svoje *socket* veze.

Threading je ugrađeni modul koji omogućava izradu programa s više niti izvršavanja. Nit (eng. *thread*) je put izvršavanja koji može izvršiti programski kod neovisno o drugim nitima koje se izvršavaju u isto vrijeme [7]. Niti se koriste za istovremeno izvršavanje različitih dijelova koda,

što može poboljšati performanse programa. Niti se često koriste u situacijama u kojima se trebaju obaviti dugotrajni zadatci. Važno je napomenuti da rad s nitima zahtijeva oprez jer se mogu pojaviti problemi s konkurentnim pristupom zajedničkim resursima (eng. *race conditions*) i drugi problemi koji mogu dovesti do neispravnog ponašanja programa.

4.3. Programski jezik RAPID

RAPID je programski jezik koji se koristi za programiranje ABB robota. RAPID sadržava sistemske i programske module unutar kojih se definira programski kod. Sistemski se moduli uglavnom koriste za definiranje zajedničkih podataka i rutina specifično prilagođenim za rad ćelije koji se ne mijenjaju često, kao što su podaci o alatu. Programski moduli uglavnom služe za definiranje procesa kao što su kalibracija, skeniranje, itd. Svi se moduli pridružuju zadatcima (eng. *task*) i svaki kontroler ima barem jedan zadatak koji upravlja kretanjem robota.

Procedura je blok koda koji se može izvoditi više puta i sastoji se od niza naredbi. U programskom jeziku RAPID procedure se definiraju pomoću ključne riječi "PROC", nakon koje slijedi naziv procedure [8]. Nakon naziva procedure navode se ulazni argumenti. Nakon toga slijedi blok koda koji čini proceduru.

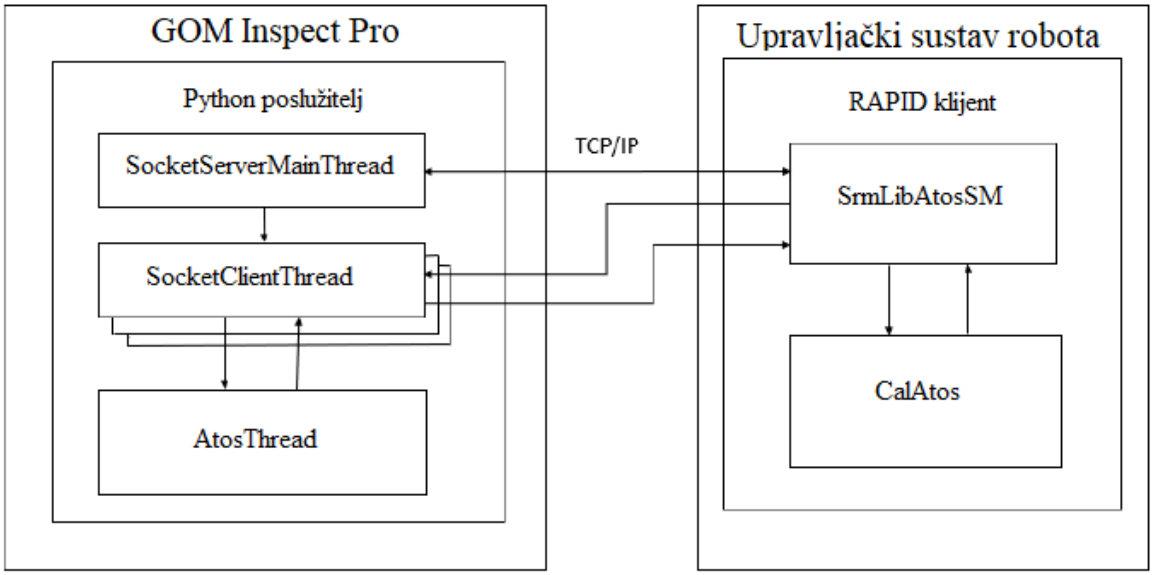
Funkcija se od procedure razlikuje samo u tome što ona vraća vrijednost. Funkcije u programskom jeziku RAPID definiraju se pomoću ključne riječi "FUNC", nakon koje slijedi tip podatka koji vraća i naziv funkcije. Nakon naziva funkcije navode se ulazni argumenti. Nakon toga slijedi blok koda koji čini funkciju. Funkcija vraća vrijednost pomoću ključne riječi "RETURN" [8].

Varijable se definiraju pomoću ključne riječi "VAR", nakon koje slijedi tip podataka varijable i njezin naziv. Varijabla može biti lokalna ili globalna. Lokalne se varijable definiraju unutar funkcije ili procedure i vrijede samo unutar te funkcije ili procedure. Globalne se varijable definiraju izvan funkcija i procedura i mogu se koristiti u cijelom programu [8].

Konstanta je vrijednost koja se ne može mijenjati tijekom izvođenja programa. Konstante se definiraju pomoću ključne riječi "CONST", nakon koje slijedi naziv konstante i njezina vrijednost [8].

5. Komunikacijski model poslužitelj-klijent

Kako je rečeno u prošlom poglavlju, za uspostavu komunikacije koriste se *socketi*. Sami su *socketi* definirani tako da imaju ograničeno vrijeme neaktivnosti (eng. *timeout*) te se, ako u tom vremenu nema izmjene podataka, veza između poslužitelja i klijenta zatvara. Mjernom uređaju za izvršavanje pojedinih radnji treba dulji (npr. inicijalizacija mjerne glave) ili kraći (npr. provjera mjernog volumena) vremenski period te postoji mogućnost da se tijekom izvođenja operacija duljeg vremenskog perioda od zadanog isteka veze, prekida veza s klijentom. Kako bi se izbjegli navedeni problemi, program poslužitelja podijeljen je na tri niti koje omogućavaju kontinuiranu komunikaciju poslužitelja i klijenta te istovremeno izvođenje potrebnih naredbi. Navedene su niti: glavna nit (u radu nazvana SocketServerMainThread), nit za pojedinačnu komunikaciju s klijentima (u radu nazvana SocketClientThread) i nit za komunikaciju s mjernim uređajem (u radu nazvana AtosThread). Slika 9 prikazuje strukturu programa.



Slika 9 Struktura komunikacijskog modela

Prema danoj strukturi vidljivo je da SocketServerMainThread predstavlja glavnu nit koja pokreće preostale niti, SocketClientThread i AtosThread. Na strani RAPID-a postoje dva modula, a to su SrmLibAtosSM i CalAtos. SrmLibAtosSM sistemski je modul koji sadržava procedure i funkcije povezane s uspostavljanjem veze klijenta i poslužitelja te njihovom komunikacijom. Programski modul CalAtos koristi funkcije definirane u modulu SrmLibAtosSM kako bi uspostavio vezu, a usto se sam koristi za pozicioniranje robota u potrebne točke kalibracije.

U nastavku ovog poglavlja bit će objašnjeni svi spomenuti dijelovi komunikacijskog modela, a prikazat će se i razvijeni programski kod.

5.1. PyAtos modul

PyAtos je modul izrađen za potrebe ovog rada u jeziku Python i programiran je korištenjem biblioteka dostupnih u sklopu softvera GOM Inspect Pro. Modul PyAtos sadrži sve naredbe potrebne za provedbu procesa kalibracije. Sve naredbe dostupne unutar modula *gom* dobivaju se snimanjem izvršavanja naredbi (eng. *macro*). *Macro* je automatski generirani Python kod kojim se omogućuje olakšano pisanje programske podrške te brža upotreba često korištenih naredbi. Također se mogu implementirati u korisničke programe te se na taj način postiže automatizacija određenih programskih procesa (kalibracije, inicijalizacije senzora i dr.).

```
import sys
import gom
import time
```

Slika 10 Uključeni moduli unutar modula *PyAtos*

Na početku programa naredbom *import* učitavaju se potrebni moduli kako je prikazano na slici 10. Modul *sys* dozvoljava pristup parametrima i funkcijama specifičnima za sustav. Modul *time* omogućava korištenje funkcija povezanih s vremenom (npr. pauziranje programa na određeno vrijeme). Modul *gom* specifičan je za aplikaciju GOM Inspect Pro. Modul gom sadržava sve funkcije i parametre potrebne za automatizirano upravljanje mjernim sustavom ATOS 5X.

```
class PyAtos(object):

    def __init__(self, verb = 0):
        self.Verbose = verb
```

Slika 11 Definiranje konstruktora klase PyAtos

Sljedeći je korak definiranje klase (eng. *class*), koja je u ovom slučaju nazvana PyAtos, i konstruktora klase *__init__()* kako je prikazano na slici 11. Konstruktor klase u Pythonu je specijalna metoda koja se koristi za inicijalizaciju objekata klase. U ovom slučaju konstruktor prima samo jedan argument *verb* koji ima zadanu vrijednost 0. Konstruktor klase postavlja vrijednost instance varijable *Verbose* na vrijednost argumenta *verb* koji je prenesen u konstruktoru. Korištenjem instance varijable *Verbose* u ostalim metodama klase kontrolira se razina detalja informacija koje se ispisuju u programu. Na primjer, ako je vrijednost instance varijable *Verbose*

jednaka 0, ispisuju se samo osnovne informacije, dok se za veće vrijednosti ispisuju detaljnije informacije.

```
def sensorini(self):
    gom.script.sys.check_system_configuration (stop_on_problem=True)
    gom.script.sys.initialize_server (mode='init')
    gom.script.atos.switch_laser(enable=True)
    gom.script.atos.switch_projector_light(enable=True)
    gom.script.atos.wait_for_sensor_warmup ()
    self._iprint("Sensor Initialized")
```

Slika 12 Metoda sensorini()

Nadalje, potrebno je definirati sve metode potrebne za izvođenje procesa kalibracije. Prva je potrebna metoda *sensorini()* (slika 12). Ova metoda poziva nekoliko funkcija iz biblioteke *gom.script* kako bi inicijalizirala senzor. Funkcija *gom.script.sys.check_system_configuration()* provjerava je li senzor kamera i projektora pravilno postavljen prije inicijalizacije. Sljedeća funkcija *gom.script.sys.initialize_server()* postavlja senzor u inicijalizirano stanje. Funkcije *gom.script.atos.switch_laser()* i *gom.script.atos.switch_projector_light()* služe za uključivanje laserskih pokazivača i projektora mjernog uređaja. Funkcija *gom.script.atos.wait_for_sensor_warmup()* čeka da istekne vrijeme zagrijavanja senzora, koje je u prosjeku između 10 i 15 minuta. Kako se proces kalibracije ne može izvršiti ako senzor nije zagrijan, ovom se funkcijom provjerava može li se kalibracija pokrenuti. Nakon inicijalizacije metoda ispisuje poruku „Sensor Initialized“ korištenjem metode *iprint*. *iprint* metoda koristi ugrađenu funkciju *print* za ispis poruke na konzolu. Funkcija prima jedan argument – „strData“ – koji se prenosi prilikom poziva. Metoda koristi naredbu *print* kako bi ispisala informacije o klasi i nazivu trenutne metode.

Ova je metoda definirana s podcrtom ispred imena kako bi se naznačilo da je to privatna metoda klase, tj. da nije namijenjena izravnom korištenju izvan klase. Ona je definirana na kraju programskog koda, ali kako se koristi u svim metodama, objašnjena je u ovom dijelu rada.

```
def checkvol(self):
    self.gommv = gom.app.get ('sys_sensor_configuration.scan_measuring_volume.name')
    if self.gommv == "MV1000 (1000x750x750)":
        self.gommv = self.gommv[0:6]
        self._iprint(self.gommv)
    else:
        self.gommv = self.gommv[0:5]
        self._iprint(self.gommv)
```

Slika 13 Metoda checkvol()

Svrha metode *checkvol()*, prikazane na slici 13, provjera je pokretanja ispravnog programa kalibracije. Funkcijom *gom.app.get()* učitava se naziv instaliranog senzora, koji se uspoređuje s nazivom koji je poslao klijent. Ako se oni podudaraju, kalibracija se može nastaviti.

```
def startcal(self, temp):
    self.temp = temp

    gom.script.calibration.start_calibration (
        calibration_sequence='stand',
        extended_measuring_volume_depth_for_reference_points=False,
        instructions=True,
        max_residual_edge_point_adjustment=0.4,
        max_residual_gray_value_adjustment=0.06,
        min_ellipse_radius=3.0,
        minimum_ellipse_contrast=20.0,
        movement_check_threshold=0.2,
        reference_point_identification_method='gray_value_adjustment',
        temperature=self.temp,
        use_calibration_object_from_sensor=True)

    self._iprint("Set temperature to %s C" % str(self.temp))
```

Slika 14 Metoda startcal()

Metoda *startcal()* prima argument „temp“ kojim je određena temperatura kalibracijske ploče kao jedan od važnijih uvjeta za kvalitetnu kalibraciju. *gom.script.calibration.start_calibration()*ugrađena je funkcija koja se koristi za postavljanje svih parametara potrebnih za kalibraciju. Funkcija *start_calibration* ima mnogo opcija za postavljanje parametara kalibracije. U te se opcije, među ostalima, ubrajaju:

- *calibration_sequence*: tip kalibracijskog postupka koji se želi pokrenuti
- *max_residual_edge_point_adjustment*: maksimalna dopuštena greška prilikom podešavanja rubnih točaka
- *max_residual_gray_value_adjustment*: maksimalna dopuštena greška prilikom podešavanja sivih vrijednosti
- *movement_check_threshold*: prag za provjeru pokreta mjerne glave tijekom bilježenja pozicije
- *reference_point_identification_method*: metoda identifikacije referentne točke
- *temperature*: temperatura senzora.

Metoda je prikazana na slici 14.

```
def recordposition(self):
    gom.script.calibration.snap_image (exposure_time=0.001633,exposure_time_correction=True)

    time.sleep(10)
    self._iprint("Position recorded")
```

Slika 15 Metoda recordposition()

Metoda *recordposition()* (slika 15) sadržava jednu ugrađenu *gom* funkciju pomoću koje se snima pozicija kalibracijske ploče. U njoj se namještaju vrijednosti vremena ekspozicije, a to se vrijeme ispravlja ovisno o trenutnim uvjetima. Na kraju funkcija ispisuje poruku „Position recorded“, čime se označava da je pozicija uspješno zabilježena.

```
def endcal(self):
    gom.script.calibration.compute_calibration ()
    gom.script.calibration.save_calibration ()
    gom.script.calibration.exit_calibration ()
    time.sleep(2)
    self._iprint("Ending cal")
```

Slika 16 Metoda endcal()

Metoda *endcal()* prikazana je na slici 16. Ova metoda iz biblioteke *gom.script.calibration* poziva tri funkcije koje se koriste za završetak postupka kalibracije:

- *compute_calibration*: izračunava rezultate kalibracije
- *save_calibration*: sprema kalibraciju u memoriju
- *exit_calibration*: završava postupak kalibracije.

Nakon toga metoda pauzira izvođenje programa na dvije sekunde pomoću funkcije *time.sleep* i na kraju ispisuje poruku „Ending calibration“ koristeći metodu *_iprint*.

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


if __name__ == "__main__":
    at = PyAtos(verb=1)

    exit(0)
```

Slika 17 Metoda _iprint() i izravno pokretanje

Na slici 17 prikazan je dio koda koji sadržava prethodno objašnjenu metodu *_iprint()* i blok za izravno pokretanje programa. Kad se ova skripta pokrene, prvo se provjerava je li varijabla „name“ jednaka varijabli „main“. Ako jest, to znači da se modul izvršava izravno, umjesto da se uvozi iz

drugog modula ili programa, stoga se stvara nova instanca klase PyAtos s argumentom *verb*=1, koja se dodjeljuje varijabli „at“.

5.2. Implementacija poslužitelja u programskom jeziku Python

SocketServer je modul programskog jezika Python napisan za potrebe ovog rada kako bi se omogućila komunikacija mjernog sustava s upravljačkim sustavom robota. Modulom SocketServer definiran je kod poslužitelja. Kao i kod modula PyAtos, na početku se pomoću naredbe *import* učitavaju svi potrebni moduli.

```
import socket
import sys
import threading
import queue
import time
from tkinter import *
from tkinter import messagebox
from PyAtosT import PyAtosT
```

Slika 18 Učitani moduli programa SocketServer

Modul *socket* opisan je u prethodnom poglavlju, u kojem je spomenuto da služi uspostavi komunikacije poslužitelja i klijenta pomoću TCP/IP protokola. Modul *threading* služi za kreiranje niti unutar programa i upravljanje njima. Modul *queue* služi za implementaciju redova prema metodi FIFO (eng. *First In First Out*) što znači da prvi element koji se doda u red je i prvi element koji se čita iz reda. Modul *Tkinter* omogućava izradu jednostavnog grafičkog korisničkog sučelja za upravljanje programom. Ta je funkcionalnost potrebna jer ukoliko se Python skripta izvršava u okruženju GOM Inspect Pro, nije moguće imati korisničku interakciju s programom pomoću konzole. Posljednji modul koji se učitava prethodno je objašnjeni modul PyAtos. Moduli *sys* i *time* također su prethodno objašnjeni. Slika 18 prikazuje sve učitane module.

```
class AtosSocketServer(object):
    def __init__(self, srvBindIP="0.0.0.0", srvPort=30000, srvTimeout=40, srvMaxClients=5, srvRespDelay=0.3,
                 autoStart=True, verb=0):

        self.Verbose = verb
        self.printLock = threading.Lock()



        self.srvBindIP = srvBindIP
        self.srvPort = srvPort
        self.srvTimeout = srvTimeout
        self.srvMaxClients = srvMaxClients
        self.srvRespDelay = srvRespDelay
        self.srvSocket = None

        self.srvRUN = False
        self.srvLastRESPONSE = b''

        self.SocketServerThread = None
        self.clientThreadCounter = 0
        self.lstSrvClientThreads = list()



        self.at = None
        self.atRUN = False
        self.atBUSY = False

        self.atRespPrefixACK = b'ACK'
        self.atRespPrefixBSY = b'BSY'
        self.atRespPrefixERR = b'ERR'
        self.atRespPrefixNC = b'ERR-RI-NOT-CONNECTED'

        self.atClientThread = None
        self.atInQueue = queue.Queue(maxsize=1)
        self.atClientLock = threading.Lock()

        self.stateID = "IDLE"

        if autoStart:
            self.startAll()
```

Slika 19 Klasa AtosSocketServer i konstruktor klase

Nakon što su moduli učitani, definira se klasa nazvana AtosSocketServer i konstruktor klase *__init__()* prema slici 19. U sklopu konstruktora definirani su argumenti *srvBindIP*, *srvPort*, *srvTimeout*, *srvMaxClients*, *srvRespDelay*, *autoStart* i *verb*. Argument *srvBindIP* koristi se za definiranje IP adrese poslužitelja koja definira IP adresu na kojoj *socket* osluškuje za vezu, dok argument *srvPort* postavlja port na kojem poslužitelj osluškuje za vezu. Vrijednost IP adrese „0.0.0.0“ označava da poslužitelj osluškuje sve postavljene IPv4 adrese računala na kojem se programski kod izvršava. *srvTimeout* definira vrijeme prekida veze (eng. *timeout*) u slučaju neaktivnosti veze, a *srvMaxClients* definira maksimalan mogući broj povezanih klijenata. *srvRespDelay* definira vrijeme kašnjenja odgovora poslužitelja. Također, u sklopu konstruktora definirana je vrijednost instance varijable *Verbose*, koja služi za ispis detalja izvršavanja programa. *printLock = threading.Lock()* inicijalizira objekt klase *Lock* koji se koristi za sinkronizaciju pristupa dijelu koda u kojem se koriste naredbe *print*. U višenitnom programiranju, kad više niti pokušava izvršiti naredbu *print()* u isto vrijeme, može doći do grešaka jer se tekst ispisa može preklapati, a ispisi iz različitih niti mogu se miješati. Zbog toga je potrebno osigurati da samo jedna

nit može pristupiti naredbi *print()* u isto vrijeme, a to se postiže pomoću objekta klase *Lock*. Prilikom pozivanja naredbe *print()*, nit pokušava zaključati objekt klase *Lock* pozivom metode *acquire()*. Ako objekt već ima zaključanu nit, nit koja pokušava zaključati objekt čeka dok se taj objekt ne otključa, odnosno dok prva nit ne pozove metodu *release()*. Tek tad druga nit može zaključati objekt i izvršiti naredbu *print()*. Ako je objekt otključan nit ga zaključava i izvršava naredbu *print()*. Tako se osigurava da tekst ispisa neće biti isprepleten i da će svaki ispis pojedine niti biti jasno vidljiv. *SocketServerThread* nit je koja osluškuje dolazne veze klijenata. *lstSrvClientThreads* lista je svih niti koje se trenutno izvršavaju za svakog klijenta koji je povezan na poslužitelj. *at* je instanca klase PyAtos koja se koristi za obradu naredbi koje klijenti šalju na poslužitelj. *atInQueue* red je čekanja u kojem se pohranjuju naredbe koje treba obraditi.

Za svako stanje procesa 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 upravljačkim sustavom robota. Stanje poslužitelja inicijalno je postavljeno u zauzeto stanje (prefiks BSY). Ovi se atributi kasnije koriste prilikom slanja odgovora klijentu nakon obrade njegova zahtjeva. *atClientThread* varijabla je koja će sadržavati nit koja predstavlja klijenta spojenog na poslužitelj. Objektom *atInQueue* definira se red metodom modula queue te se varijablom *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 *atClientThread* odnosno dio programa koji izvršava određene naredbe na mjernom uređaju. *stateID* predstavlja varijablu stanja koja govori u kojem se stanju poslužitelj trenutno nalazi. Pomoću te varijable prati se redoslijed odvijanja programa te se točno određuje iz kojeg se stanja može prijeći u sljedeće. Posljednja linija pokreće metodu *startAll()*. Ta metoda pokreće glavnu nit poslužitelja, a time i cijeli program.

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

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

Slika 20 Metode startAll() i stopAll()

Metode prikazane na slici 20 koriste se za pozivanje metoda *startSocketServ()* i *stopSocketServ()*.

```
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()
        self.atRUN = True
        self.at = PyAtosT(verb=self.Verbose)
        self.atClientThread = threading.Thread(target=self.__ProcessAtRq, name="AtosThread")
        self.atClientThread.start()
    else:
        self.__iprint("Socket server already running ...")
        messagebox.showinfo("Server", "Socket server already running...")
```

Slika 21 Metoda startSocketServ()

Kod prikazan na slici 21 sadržava metodu *startSocketServ()*. Metodom *startSocketServ()* uspostavlja se konekcija s upravljačkim sustavom robota. Unutar metode startSocketServ() 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. Konačno, poziva se metoda *bind* za *socket* koja povezuje *socket* s adresom *srvBindIP* i priključkom *srvPort*. Ako su svi koraci uspješno izvedeni, *srvRUN* postavlja se na „TRUE“. Nakon toga je definirana i pokrenuta nit *SocketServerThread* koja izvršava metodu *_runSocketServ*. *atRUN* usto se postavlja na „TRUE“.

Ako je *srvRUN* već postavljena na „TRUE“, metoda će ispisati poruku „Socket server already running ...“ i korisniku prikazati odgovarajuću poruku upozorenja. Time se osigurava da se poslužitelj ne pokreće kada on već radi kako ne bi došlo do nepotrebnog ponovnog pokretanja programa.

```
def stopSocketServ(self):
    # TODO: Prevent shutdown if socket server is allready stopped
    self.__iprint("Stopping socket server ...")
    self.srvRUN = False
    self.stateID = "IDLE"
    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()

    self.__iprint(("\tWaiting for atos thread to join..."))
    with self.atClientLock:
        self.atRUN = False

    with self.atInQueue.mutex:
        self.atInQueue.queue.clear()
    self.atInQueue.put(None)
    if self.atClientThread is not None:
        self.atClientThread.join()
        try:
            del (self.at)
        except Exception as e:
            pass
        self.at = None

    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 Metoda stopSocketServ()

Prema kodu prikazanom na slici 22, metoda *stopSocketServ()* služi za zaustavljanje instance *SocketServer*. Metoda *stopSocketServ()* prvo postavlja *srvRUN* na „FALSE“, čime se signalizira da se server treba zaustaviti, a *stateID* na „IDLE“ kako bi se stanje sačuvalo za ponovno uključivanje poslužitelja. Zaustavljaju se sve niti koje su uspostavljene za komunikaciju s klijentima te nit *SocketServerThread()*. Metoda zatim postavlja *atRUN* na „FALSE“ i briše sve objekte iz reda čekanja *atinQueue*. Na kraju se objekt *socket* zatvara naredbom *close()* i veza s klijentom se prekida.

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

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

        # Lista (samo) zivih 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" % (str(self.clientThreadCounter)))
        _SocketClientThread.start()
        self.lstSrvClientThreads.append(_SocketClientThread)
```

Slika 23 Metoda _runSocketServ()

Metoda prikazana na slici 23 definira ponašanje *socket* poslužitelja. Na početku se *socket* konfigurira tako da sluša dolazne veze, a pritom je definiran maksimalni broj mogućih klijenata. Inicijalizira se prazna lista koja će sadržavati sve niti klijenata koji su se povezali na poslužitelj. Pokreće se petlja koja se izvršava sve dok je poslužitelj aktivan. Filtriraju se niti koje su završile s radom. Kad se klijent poveže, vraćaju se dva argumenta: argument *_conn*, koji predstavlja vezu klijenta s poslužiteljem (koristi se za slanje i primanje podataka između klijenta i poslužitelja), i argument *_addr*, koji predstavlja adresu klijenta. Stvara se nova nit koja obrađuje vezu novog klijenta. Povezivanjem klijenta pokreće se metoda *_ProcessClRq()*. Na kraju se nit dodaje na popis svih niti *lstSrvClientThreads()*.

```
def __ProcessClRq(self, conn, addr):
    _retData = b''

    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  (BLANK DATA)" % (str(addr)))
                break

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


                if self.at is None or self.atRUN == False:
                    _retData = b"%b" % (self.atRespPrefixNC)

                else:
                    try:
                        self.atInQueue.put(_data, block=False)


                    except queue.Full:
                        if self.Verbose >= 2:
                            self.__iprint("Atos Queue is FULL, replying last response")
                        _retData = self.srvLastRESPONSE
                    else:
                        if self.Verbose >= 2:
                            self.__iprint("Atos 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.atClientLock:
                            _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
        # Timeout occurred, do things

        if self.Verbose >= 2:
            self.__iprint("=" * 20)
```

Slika 24 Metoda _ProcessClRq()

Metoda *_ProccesClRq()* prikazana na slici 24 koristi se za zaprimanje i slanje podataka klijentu. U varijablu *_data* pomoću metode *recv()* zaprimaju se podatci do veličine 1024 bajta. Vremensko ograničenje zaprimanja podataka od klijenta definirano je u sklopu konstruktora. U ovom je slučaju postavljeno na 40 sekundi zbog dugog kretanja robota između pozicija. Ako se podatci

prime unutar definiranog vremena, naredbom *put()* stavljaju se u red, koji je inicijaliziran u konstruktoru. Veličina reda ograničena je na jedan kako se tijekom obrade određene naredbe ne bi mogle zaprimati dodatne.

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

        _cmd = self.atInQueue.get()

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

        if _cmd is None:
            if self.Verbose >= 1:
                self.__iprint("Received NULL in queue, terminating...")
            with self.atClientLock:
                self.atRUN = False
            break


        elif (_cmd == b'SRQ'):
            self.__iprint("STATUS REQUEST")

            _response = self.srvLastRESPONSE

        else:

            _response = b"%b-%b" % (self.atRespPrefixBSY, _cmd)

            # Lock critical vars using thread lock
            with self.atClientLock:
                self.atBUSY = True
                self.srvLastRESPONSE = _response

            if (_cmd == b'IAS'):
                data_decoded = _cmd.decode()
                self.__iprint("Waiting for sensor to initialize...")
                self.at.sensorini()
                _response = b"%b-%b" % (self.atRespPrefixACK, _cmd)

            elif (_cmd.decode()[0:3] == "CMV"):
                if self.stateID == "IDLE":
                    self.stateID = "CAL.CMV"
                    self.__iprint("Checking measure volume...")
                    data_decoded = _cmd.decode().replace("CMV", "")
                    self.at.checkvol()
                    mv = self.at.gommv
                    self.__iprint(mv)
                    if mv == data_decoded:
                        _response = b"%b-%b" % (self.atRespPrefixACK, _cmd)
                    else:
                        _response = b"%b-%b" % (self.atRespPrefixERR, b'WRONG MEASURE VOLUME')
                else:
                    _response = b"%b-%b" % (self.atRespPrefixERR, b'WRONG STATE')
```

Slika 25 Metoda __ProcessAtRq()

Metodu *__ProcessAtRq()* izvršava *AtosThread* (slike 25 i 26). U sklopu te metode podatci iz reda preuzimaju se pomoću naredbe *get()*. Učitani podatak dodjeljuje se varijabli *_cmd*. Vrijednost te varijable provjerava se kroz petlju sve dok je *atRUN* jednak „TRUE“ kroz niz uvjeta. Ako *_cmd* nema vrijednost, odnosno ako primi vrijednost „NONE“, petlja se prekida i ta se nit zaustavlja, odnosno dolazi do prekida veze poslužitelja i klijenta. U slučaju da se primi vrijednost SRQ,

metoda vraća odgovor o izvršavanju posljednje naredbe. U tijeku izvođenja metode modula PyAtos, ova metoda klijentu odgovara prefiksom *BSY*, čime se veza drži aktivnom.

```
elif (_cmd.decode()[0:3] == "SCP"):
    if self.stateID == "CAL.CMV":
        self.stateID = "CAL.SCP"
        self.i = 1
        data_decoded = _cmd.decode().replace("SCP", "")
        self.__iprint("Starting calibration")
        self.at.startcal(data_decoded[0:5], data_decoded[6:10])
        # self.at.startcal()
        _response = b"%b-%b" % (self.atRespPrefixACK, _cmd)
    else:
        _response = b"%b-%b" % (self.atRespPrefixERR, b'WRONG STATE')
elif (_cmd == b'POS'):
    if self.stateID[0:7] == "CAL.POS" or self.stateID == "CAL.SCP":
        data_decoded = _cmd.decode()
        self.stateID = "CAL.POS{}".format(self.i)
        self.__iprint("Recording position{}".format(self.i))
        self.at.recordposition()
        self.i += 1
        _response = b"%b-%b" % (self.atRespPrefixACK, _cmd)
    else:
        _response = b"%b-%b" % (self.atRespPrefixERR, b'WRONG STATE')
elif (_cmd == b'ECP'):
    if self.stateID == "CAL.POS18":
        self.stateID = "CAL.ECP"
        data_decoded = _cmd.decode()
        self.__iprint("Ending calibration")
        self.at.endcal()
        _response = b"%b-%b" % (self.atRespPrefixACK, _cmd)
        self.stateID = "IDLE"
    else:
        _response = b"%b-%b" % (self.atRespPrefixERR, b'WRONG STATE')


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

Slika 26 Metoda __ProcessAtRq()

Ako se primi vrijednost „IAS“, „CMV“, „SCP“, „POS“ ili „ECP“ pokreću se pojedine metode definirane u sklopu modula PyAtos. Na primjer, ako se primi vrijednost „IAS“, metoda inicijalizira metodu definiranu unutar modula PyAtos koja se koristi za inicijalizaciju mjerne glave. Usto se pomoću vrijednosti argumenta *stateID* provjerava nalazi li se program u odgovarajućem stanju. Ako je dopušten prelazak u novo stanje, program se nastavlja i postavlja se nova vrijednost atributa *stateID*. U suprotnom se slučaju šalje odgovor prefiksa *ERR*, koji označava grešku stanja. Kad je metoda modula PyAtos uspješno izvršena, klijentu se šalje odgovor prefiksa *ACK*, što označava da je mjerni uređaj spreman za izvršavanje nove naredbe. Prema ovom su primjeru izrađene sve funkcije potrebne za provedbu kalibracije.

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

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):
    thread = list()
    self.__iprint("--------------| RUNNING THREADS: |--------------")
    thread.append("| RUNNING THREADS: |")
    for _thread in threading.enumerate():
        self.__iprint(" \t %s" % (_thread.name))
        thread.append(_thread.name)
    if len(self.lstSrvClientThreads) > 0:
        self.__iprint("-------| SOCKET SERVER CLIENT THREADS: |--------")
        thread.append("SOCKET SERVER CLIENT THREADS:")
        for _thread in self.lstSrvClientThreads:
            self.__iprint(" \t %s" % (_thread.name))
            thread.append(_thread.name)
    self.__iprint("-----------------------------------------------")
    thread = [str(x) for x in thread]
    msg = '--------- '.join(thread)
    messagebox.showinfo("Running threads", msg)

def printState(self):
    self.__iprint("--------------| CURRENT STATE: |--------------")
    self.__iprint(self.stateID)
    messagebox.showinfo("Current state", self.stateID)

def stop(self):
    messagebox.showinfo("Server", "Server stopped...")
```

Slika 27 Ostale metode poslužitelja

Prva metoda prikazana na slici 27 metoda je destruktora koja služi za zaustavljanje programa pozivanjem metode *stopAll()*. Druga je metoda opisana prilikom opisa modula PyAtos i ovdje ima istu ulogu. Metoda *printThreads()* koristi se za ispis postojećih niti u sklopu programa, koje se prikazuju u *messageboxu* grafičkog sučelja *Tkinter* kao novi skočni prozor. Na taj način moguće je praćenje trenutno aktivnih niti u programu. Također, definirane su metode koje služe za ispis trenutnog stanja programa definiranog u *stateID* i metoda za zaustavljanje poslužitelja pomoću grafičkog sučelja.

Posljednji dio koda poslužitelja koristi se za izravno pokretanje programa i u sklopu njega se pomoću modula *Tkinter* inicijalizira grafičko korisničko sučelje za interakciju s programom. Kod je prikazan na slici 28.

```
if __name__ == "__main__":

    sserv = AtosSocketServer(srvMaxClients=2, verb=1)

    try:

        root = Tk()
        root.title("Server")
        root.geometry("200x180")
        exit_button = Button(root, text="Stop program", command=root.destroy)
        start_server_button = Button(root, text="Start server", command=sserv.startSocketServ)
        stop_server_button = Button(root, text="Stop server", command=lambda: [sserv.stopSocketServ(), sserv.stop()])
        print_thread_button = Button(root, text="Print threads", command=sserv.printThreads)
        print_state_button = Button(root, text="Print states", command=sserv.printState)
        start_server_button.pack(pady=5)
        stop_server_button.pack(pady=5)
        print_thread_button.pack(pady=5)
        print_state_button.pack(pady=5)
        exit_button.pack(pady=5)
        root.mainloop()
    except KeyboardInterrupt:
        print("TERMINATING MAIN DUE TO KEYBOARD INTERRUPT")

    sserv.stopSocketServ()

    # del sserv
    print("REMOVED")
```

Slika 28 Izravno pokretanje ServerSocketa

5.3. Implementacija klijenta u programskom jeziku RAPID

Kod prikazan u ovom poglavlju je izrađen na upravljačkom sustavu robota te predstavlja klijenta u modelu poslužitelj-klijent.

SrmLibAtosSM predstavlja sistemski modul u kojem su definirani svi objekti i varijable potrebni za definiranje pristupne točke. U njemu su također definirane sve procedure za povezivanje i slanje naredbi poslužitelju.

```
MODULE srmLibAtosSM(SYSMODULE)

    VAR socketdev sdAtClient;
    VAR socketstatus ssAtClient;
    VAR string strAtHostIP:="192.168.142.10";
    VAR num numAtHostPort:=30000;
```

Slika 29 Definiranje varijabli klijenta

Varijabla *sdAtClient* definirana je tipom podataka *socketdev*, koja omogućava stvaranje pristupne točke i spajanje na poslužitelj. Varijabla *ssAtClient* omogućava provjeru statusa spajanja i kontrolu tijeka programa. Varijablom *strAtHost* tipa *string,* definirana je IP adresa poslužitelja, dok je varijablom *numAtHostPort,* tipa *num*, definiran port poslužitelja. Kod je prikazan na slici 29.

Prvo je definirana procedura *atConnect()* prikazana na slici 30. Uloga je te procedure provjera statusa veze klijenta s poslužiteljem. Procedura *atConnect()* prima četiri opcionalna argumenta, a to su *Host, Port, RecconnectionAttempts* i *Verbose*. Pomoću varijable *ssAtClient* ta procedura

provjerava trenutni status pristupne točke i ako dođe do prekida veze, pokreće proceduru *atRecconnect* radi ponovne uspostave veze. Ta procedura također ispisuje status pristupne točke na privjesak za učenje. Privjesak za učenje predstavlja operatorski panel upravljačkog sustava robota.

```
PROC atConnect(\string Host,\num Port,\num RecconectionAttempts,\bool Verbose)
    VAR bool bRet:=FALSE;
    VAR string stratHost;
    VAR num numAtPort;
    VAR num numRecconections;
    VAR bool bVerb;

    IF Present(Host) THEN strAtHost:=Host;
    ELSE strAtHost:=strAtHostIP;
    ENDIF
    IF Present(Port) THEN numAtPort:=Port;
    ELSE numAtPort:=numAtHostPort;
    ENDIF
    IF Present(RecconectionAttempts) THEN numRecconections:=RecconectionAttempts;
    ELSE numRecconections:=1;
    ENDIF
    IF Present(Verbose) THEN bVerb:=Verbose;
    ELSE bVerb:=FALSE;
    ENDIF
    ssAtClient:=SocketGetStatus(sdAtClient);
    TEST ssAtClient
    CASE SOCKET_CREATED:
        if (bVerb=TRUE) THEN TPWrite "AtConnect: SOCKET_CREATED; skipping connection...";
        ENDIF
    CASE SOCKET_CLOSED:
        if (bVerb=TRUE) THEN TPWrite "AtConnect: SOCKET_CLOSED; (re)connecting...";
        ENDIF
        atReconnect\Host:=strAtHost,\Port:=numAtPort,\RecconectionAttempts:=numRecconections,\Verbose:=bVerb;
    CASE SOCKET_BOUND: if (bVerb=TRUE) THEN
            TPWrite "AtConnect: SOCKET_BOUND; invalid fo rclient";
        ENDIF
    CASE SOCKET_LISTENING:
        if (bVerb=TRUE) THEN TPWrite "AtConnect: SOCKET_LISTENING; invalid for client";
        ENDIF
    CASE SOCKET_CONNECTED:
        if (bVerb=TRUE) THEN TPWrite "AtConnect: SOCKET_CONNECTED; skipping (re-connection)...";
        ENDIF
        bRet:=TRUE;
    DEFAULT:
        if (bVerb=TRUE) THEN TPWrite "AtConnect: UNKNOWN SOCKET STAT";
        ENDIF
        atReconnect\Host:=strAtHost,\Port:=numAtPort,\RecconectionAttempts:=numRecconections,\Verbose:=bVerb;
    ENDTEST
ENDPROC
```

Slika 30 Procedura atConnect

Sljedeća je definirana procedura *atReconnect*, koja je prikazana na slici 31. U sklopu te procedure prvo se prekida veza s poslužiteljem, a nakon toga se kreira nova pristupna točka radi uspostave nove veze s poslužiteljem. Definiraju se IP adresa i port te maksimalno vrijeme čekanja za uspostavu veze, koje u ovom slučaju iznosi dvije sekunde. Ako se veza ne uspostavi u roku od dvije sekunde ili ako se veza odbije, program će generirati iznimku *ERR_SOCK_TIMEOUT*, koju je potrebno obraditi. Broj pokušaja ponovnog spajanja zadan je varijablom *numAaxRecconection*, koja je početno postavljena na jedan, osim ako korisnik ne navede drugu vrijednost prilikom

pozivanja procedure korištenjem atributa *RecconectionAttempts*. Upravljanje iznimkom *ERR_SOCK_TIMEOUT* izvedeno je pomoću sustavne varijable *ERRNO*. Ako se generira iznimka *ERR_SOCK_TIMEOUT*, a broj pokušaja ponovnog spajanja (*numAaxRecconection*) premaši vrijednost *numRecconectionAttempt*, vrijednost *numAaxRecconection* povećava se za jedan te se program pokušava ponovno povezati s poslužiteljem koristeći naredbu *RETRY*.

```
PROC atReconnect(\string Host,\num Port,\num RecconectionAttempts,\bool Verbose)
    VAR bool bRet:=FALSE;
    VAR string strAtHost;
    VAR num numAtPort;
    VAR num numAaxRecconections;
    VAR bool bVerb;
    VAR num numRecconectionAttempt:=0;

    IF Present(Host) THEN strAtHost:=Host;
    ELSE strAtHost:=strAtHostIP;
    ENDIF
    IF Present(Port) THEN numAtPort:=Port;
    ELSE numAtPort:=numAtHostPort;
    ENDIF
    IF Present(RecconectionAttempts) THEN numAaxRecconections:=RecconectionAttempts;
    ELSE numAaxRecconections:=1;
    ENDIF
    IF Present(Verbose) THEN bVerb:=Verbose;
    ELSE bVerb:=FALSE;
    ENDIF
    SocketClose sdAtClient;
    SocketCreate sdAtClient;
    SocketConnect sdAtClient,strAtHost,numAtPort,\Time:=2;

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

Slika 31 Procedura atReconnect

Kako bi se prekinula veza s poslužiteljem, definirana je procedura *atDisconnect*. Naredba *SocketClose* koristi se za zatvaranje pristupne točke, nakon kojeg se ispisuje obavijest o prekidu veze. Kod procedure prikazan je na slici 32.

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

Slika 32 Procedura atDisconnect

Funkcija *atSendRcvMsg()* se koristi za slanje i primanje podataka poslužitelja.

```
FUNC string atSendRcvMsg(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
    strExpctedACKMsg:="ACK-"+strMsg2Send;
    strExpctedBSYMsg:="BSY-"+strMsg2Send;
    strExpctedERRMsg:="ERR-"+strMsg2Send;

    SocketSend sdAtClient\Str:=strMsg2Send;
    IF (bVerb=TRUE) THEN TPWrite "atSendRcvMsg: TX:"+strMsg2Send;
    ENDIF
```

Slika 33 Funkcija atSendRcvMsg

Ima nekoliko opcionalnih argumenata koje nije potrebno definirati tijekom poziva funkcija. Iznimka je argument *strMsg2Send*, koji predstavlja naredbu koja se šalje poslužitelju. Poruke primljene od poslužitelja definirane su pomoću triju varijabli. Poruke primljene od poslužitelja jednake su poslanoj naredbi uz dodan prefiks koji označava status izvršavanja naredbe. Prefiksi mogu biti *ACK*, *BSY* i *ERR* (jednaki kao što su definirani u programskom kodu poslužitelja). Prefiks *ACK* označava uspješno izvršenu naredbu, prefiks *BSY* označava da se prethodna naredba još uvijek izvršava, dok prefiks *ERR* označava da je došlo do greške u izvođenju programa. Naredba *SocketSend()*služi za slanje poruka poslužitelju. Veličina poruke može biti maksimalno 80 bajta [4]. Naredba *SocketRecieve()* služi za primanje poruka od poslužitelja. Unutar funkcije vrti se petlja koja, sve dok se prima poruka prefiksa *BSY*, šalje poruku koja sadržava naredbu SRQ (*Status Request*) kako bi dobila informaciju o statusu programa poslužitelja. Nakon primanja prefiksa *ACK* petlja se prekida, a na privjesak za učenje (operatorski panel upravljačkog sustava robota) ispisuje se obavijest o obavljenoj naredbi. Argument *Time*, koji poprima vrijednost varijable *numTimeout*, služi za definiranje maksimalnog vremena čekanja za primanje podataka. Ako vrijeme istekne, podiže se iznimka *ERR_SOCK_TIMEOUT*, nakon koje se poziva procedura *atReconnect*. Slike 33 i 34 prikazuju opisanu funkciju.

```
    WHILE (bRunLoop) DO
        SocketReceive sdAtClient\Str:=strMsg2Receive,\Time:=numTimeout;
        IF (bVerb=TRUE) THEN TPWrite "atSendRcvMsg: RX:"+strMsg2Receive;
        ENDIF
        IF strMsg2Receive=strExpctedACKMsg THEN
            IF (bVerb=TRUE) THEN TPWrite "atSendRcvMsg: RX ACK:"+strMsg2Receive;
            ENDIF
            bRunLoop:=FALSE;
        ELSEIF strMsg2Receive=strExpctedBSYMsg THEN SocketSend sdAtClient\Str:="SRQ";
            IF (bVerb=TRUE) THEN TPWrite "atSendRcvMsg: RX BSY -> TX: SRQ";
            ENDIF
            WaitTime 0.5;
        ELSE SocketSend sdAtClient\Str:=strMsg2Send;
            IF (bVerb=TRUE) THEN TPWrite "atSendRcvMsg: RX DIFFERENT-ACK/ERR -> TX:"+strMsg2Send;
            ENDIF
            WaitTime 0.5;
        ENDIF
    ENDWHILE
ERROR
    IF ERRNO=ERR_SOCK_TIMEOUT THEN
        IF (bVerb=TRUE) THEN TPWrite "SM ERR; TX:"+strMsg2Send+"; RX:"+strMsg2Receive;
        ENDIF
        atReconnect;
        RETRY;
        ErrWrite "atSendRcvMsg:","ERR_SOCK_TIMEOUT to the Atos server",\RL2:="Please check if Atos 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
        atReconnect;
        RETRY;
        ErrWrite "atSendRcvMsg:","ERR_SOCK_CLOSED to the Atos server",\RL2:="Please check if Atos server is running";
        Break;
        SystemStopAction\StopBlock;
        RAISE ;
    ELSE
        ! No error recovery handling
        IF (bVerb=TRUE) THEN TPWrite "atSendRcvMsg: SM OK; TX:"+strMsg2Send+"; RX:"+strMsg2Receive;
        ENDIF
    ENDIF
    RETURN strMsg2Receive;
ENDFUNC
```

Slika 34 Funkcija atSendRcvMsg

Ovom funkcijom opisane su sve funkcije i procedure potrebne za komunikaciju s poslužiteljem. Preostalo je još definirati funkcije povezane sa slanjem pojedinih naredbi potrebnih za provedbu procesa kalibracije mjernog uređaja.

Prva je definirana procedura *atSensorInit*, koja šalje poruku „IAS“. Tom se naredbom u kodu poslužitelja pokreće inicijalizacija senzora.

Sljedeća je definirana procedura *atCheckVol*, koja šalje poruku „CMV“ kojoj je pridružena vrijednost varijable *MV*. Ona se koristi kako bi se provjerilo je li pokrenuta kalibracija pravog mjernog volumena mjernog uređaja.

Procedura *atStartCalibration* koristi se za slanje naredbe „SCP“ kojoj je pridružena varijabla *temp*. Njome se definira temperatura kalibracijske ploče prije početka kalibracije. Prilikom primanja naredbe „SCP“, pokreće se kalibracija mjernog uređaja.

Nakon toga je definirana procedura *atPosition*, koja šalje poruku „POS“, čime se poslužitelju javlja da se robot nalazi u određenoj poziciji za kalibraciju.

Posljednja je procedura potrebna za proces kalibracije procedura *atEndCalibration*. Ona šalje poruku „ECP“, čime se završava proces kalibracije.

Sve prethodne procedure tijekom izvođenja ispisuju podatke za praćenje programa na privjesak za učenje i prikazane su na slici 35.

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

    strMsgSend :="IAS";
    strMsgRecv := atSendRcvMsg(strMsgSend, \bVerbose:=bVerb);
ENDPROC

PROC atCheckVol(\bool Verbose,string MV)
    VAR bool bVerb;
    VAR string strMsgSend := "";
    VAR string strMsgRecv := "";
    IF Present (Verbose) THEN bVerb := Verbose; ELSE bVerb := FALSE; ENDIF

    strMsgSend :="CMV"+MV;
    strMsgRecv := atSendRcvMsg(strMsgSend, \bVerbose:=bVerb);
ENDPROC

PROC atStartCalibration(string Temp, \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 :="SCP" + Temp;
    strMsgRecv := atSendRcvMsg(strMsgSend, \bVerbose:=bVerb);
ENDPROC

PROC atPosition(\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 :="POS";
    strMsgRecv := atSendRcvMsg(strMsgSend, \bVerbose:=bVerb);
ENDPROC

PROC atEndCalibration(\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 :="ECP";
    strMsgRecv := atSendRcvMsg(strMsgSend, \bVerbose:=bVerb);
ENDPROC
```

Slika 35 Procedure za izvršavanje naredbi

5.4. Veza procedura klijenta i metoda poslužitelja

Radi lakšeg razumijevanja strukture programa, u tablici 1 prikazana je veza između procedura klijenta, poslanih naredbi i metoda poslužitelja.

Metoda poslužitelja	*String*	Procedura klijenta
at.sensorini	IAS	*atSensorInit*
at.gommv	CMV	*atCheckVol*
at.startcal	SCP	*atStartCalibration*
at.recordposition	POS	*atPosition*
at.endcal	ECP	*atEndCalibration*

Tablica 1 Veza procedura klijenta i metoda poslužitelja

5.5. *Programski modul za izvršavanje kalibracije*

Zadnji je korak programskog rješenja upotreba programskog jezika RAPID za izradu programskog modula *CalAtos*, koji poziva prethodno definirane procedure i funkcije potrebne za povezivanje s poslužiteljem i procedure potrebne za izvršavanje procesa kalibracije. Modul *CalAtos* usto sadržava naredbe potrebne za kretanje robota kako bi se postigle tražene pozicije za kalibraciju.

```
MODULE calAtos

    !Nul-tocke kalibracijskih panela
    TASK PERS wobjdata wobj_AtosCalMv500:=[FALSE,TRUE,"",[[510,-1360.25,2150],[0.46175,-0.88701,0,0]],[[0,0,0],[1,0,0,0]]];

    !Nul tovke skenera
    PERS tooldata T5_ATmv500:=[TRUE,[[-957.393,5.628,331.827],[0,0.70711,0,-0.70711]],[37.63,[39.76,-91.7,227.43],[1,0,0,0],3.638,4.855,1.723]];


    !Pozicije kalibracije
    LOCAL PERS robtarget p110:=[[0.0,0.0,0.0],[2.61749E-05,0.70708,-0.707133,-2.69235E-05],[-2,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p120:=[[0.0,0.0,220.0],[1.16599E-05,0.707074,-0.70714,-3.80194E-05],[-2,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p130:=[[0.0,0.0,185.0],[3.13533E-06,0.707071,-0.707143,-4.53423E-05],[-2,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p140:=[[0.0,0.0,148.0],[1.46009E-05,-0.707063,0.70715,5.54427E-05],[-2,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p150:=[[0.0,0.0,111.0],[2.54204E-05,-0.707058,0.707156,7.74553E-05],[-2,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p160:=[[0.0,0.0,74.0],[3.89588E-05,-0.707053,0.707161,9.26159E-05],[-2,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p170:=[[0.0,0.0,37.0],[3.76525E-05,-0.707053,0.707161,9.32927E-05],[-2,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p180:=[[0.0,0.0,0.0],[3.88246E-05,-0.707053,0.707161,9.4176E-05],[-2,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p190:=[[0.0,0.0,-37.0],[5.69641E-05,-0.707046,0.707168,0.00011129],[-2,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p200:=[[0.0,0.0,-74.0],[5.90449E-05,-0.707045,0.707168,0.000109488],[-2,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p210:=[[0.0,0.0,-111.0],[6.27733E-05,-0.707046,0.707168,0.000110852],[-2,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p220:=[[0.0,0.0,-148.0],[6.1861E-05,-0.707045,0.707168,0.000108715],[-2,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p230:=[[0.0,0.0,0.0],[0.141405,0.692757,-0.692833,-0.141684],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p240:=[[0.0,0.0,0.0],[0.14922,-0.691114,0.691299,-0.148996],[-2,-1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p250:=[[0.07,0.08,0.67],[0.166188,-0.672118,0.712422,0.114426],[-1,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p260:=[[0.04,0.11,0.03],[0.00112773,-0.975654,-0.00640964,0.219217],[-1,-1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p270:=[[0.0,0.0,0.0],[0.175543,-0.675158,-0.706675,-0.118141],[-2,-3,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p280:=[[2.41,3.74,1.58],[0.0210418,-0.983138,0.00080313,0.181649],[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p290:=[[0.05,0.08,-0.10],[0.280282,0.641265,0.662398,-0.267301],[-1,-1,-3,3],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p300:=[[-0.02,0.02,0.14],[0.366768,0.004151,0.930302,-0.00161209],[-2,-1,-2,3],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p310:=[[3.71,3.28,2.60],[0.0038002,-0.976318,-0.00643702,0.216212],[-1,0,-3,3],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p320:=[[-53.86,-243.14,1134.23],[0.470373,0.0123346,-0.282857,-0.835817],[-1,-1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p330:=[[930.99,171.51,1271.61],[0.256995,0.0126359,-0.401312,-0.879058],[-1,-1,-2,2],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p340:=[[348.59,-428.39,-718.80],[0.280156,0.301428,-0.910786,0.0335176],[-1,-1,-4,2],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p350:=[[-420.07,-64.07,-336.69],[0.00538213,0.750966,0.620619,-0.225506],[-1,-1,1,3],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p360:=[[563.12,-19.99,407.87],[0.165985,-0.408748,-0.544045,-0.713715],[-1,-1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p370:=[[-428.46,-785.12,602.29],[0.467434,0.750372,0.306692,-0.352684],[-1,0,-3,3],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p380:=[[-679.98,-800.84,624.81],[0.431568,0.577655,0.535254,-0.439963],[-2,0,-3,3],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p390:=[[-428.42,-785.11,602.41],[0.467491,0.750328,0.306689,-0.352707],[-1,0,-3,3],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p400:=[[1533.80,-315.97,715.90],[0.101748,0.515734,0.409004,0.74591],[-1,0,-3,1],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    LOCAL PERS robtarget p410:=[[-287.02,-816.98,648.05],[0.468754,0.821398,0.156042,-0.285],[-1,0,-3,3],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
```

Slika 36 Korisnički koordinatni sustav, TCP alata i točke kalibracije

Na početku modula definiran je korisnički koordinatni sustav koji je povezan na centar kalibracijske ploče naziva *wobj_AtosCalMv500* i TCP alata naziva *T5_ATmv500*. S obzirom na definirani koordinatni sustav i vrh alata određene su točke kalibracije, a usto su dodane međutočke

u koje se robot mora pozicionirati zbog limita zglobova robota. Taj dio koda prikazan je na slici 36.

Definirana je procedura *calAtosMv500* koja započinje pozivanjem prije definirane rutine *OpenBox* razvijene tijekom razvoja projekta ARCOPS. Rutina *OpenBox* služi za otvaranje vrata zaštitnog spremnika mjerne glave. *OpenBox* rutinu potrebno je pozvati kako bi se u procesu kalibracije dobio veći radni prostor za kretnje robota. Zatim se poziva procedura *atConnect* kako bi se uspostavila veza s poslužiteljem. Nakon toga se pozivaju procedure *atSensorInit*, *atCheckVol* i *atStartCalibration*. Nakon početka kalibracije robot se pomiče u prvu točku kalibracije, gdje se pozivaju naredbe *WaitRob\InPos* i *WaitRob\ZeroSpeed*, čija je uloga zaustaviti izvršavanje programa sve dok robot nije u točnoj poziciji i zaustaviti njegovo kretanje prije nastavka programa. Pozivom procedure *atPosition* snima se trenutna točka kalibracije. Nakon toga se ponavljaju prethodna dva koraka dok se ne zabilježe sve pozicije kalibracije. Na kraju se poziva procedura *atEndCalibration* za završetak kalibracije pa se pomoću procedure *atDisconnect* prekida veza s poslužiteljem. Slika 37 prikazuje kod procedure *calAtosMv500*. Zbog duljine i repetitivnosti samog koda, slika 37 prikazuje prvih nekoliko linija, u kojima se pozivaju početne procedure i posljednji dio, u kojem se završava proces.

```
PROC calAtosMv500()
    OpenBox;
    atConnect\Verbose:=TRUE;
    atSensorInit\Verbose:=TRUE;
    atCheckVol\Verbose:=TRUE;
    atStartCalibration\Verbose:=TRUE;
    MoveJ p110, v100, z50, T5_ATmv500\WObj:=wobj_AtosCalMv500;
    WaitRob\InPos;
    WaitRob\ZeroSpeed;
    atPosition\Verbose:=TRUE;
    MoveJ p120, v100, z50, T5_ATmv500\WObj:=wobj_AtosCalMv500;
    WaitRob\InPos;
    WaitRob\ZeroSpeed;
    atPosition\Verbose:=TRUE;
    MoveJ p130, v100, z50, T5_ATmv500\WObj:=wobj_AtosCalMv500;
    WaitRob\InPos;
    WaitRob\ZeroSpeed;
    atPosition\Verbose:=TRUE;
    MoveJ p140, v100, z50, T5_ATmv500\WObj:=wobj_AtosCalMv500;
    WaitRob\InPos;
    WaitRob\ZeroSpeed;
    atPosition\Verbose:=TRUE;
    atEndCalibration\Verbose:=TRUE;
    atDisconnect\Verbose:=TRUE;
    MoveJ p390, v100, z50, T5_ATmv500\WObj:=wobj_AtosCalMv500;
    MoveJ p410, v100, z50, T5_ATmv500\WObj:=wobj_AtosCalMv500;
    MoveJ p400, v100, z50, T5_ATmv500\WObj:=wobj_AtosCalMv500;

ENDPROC
```

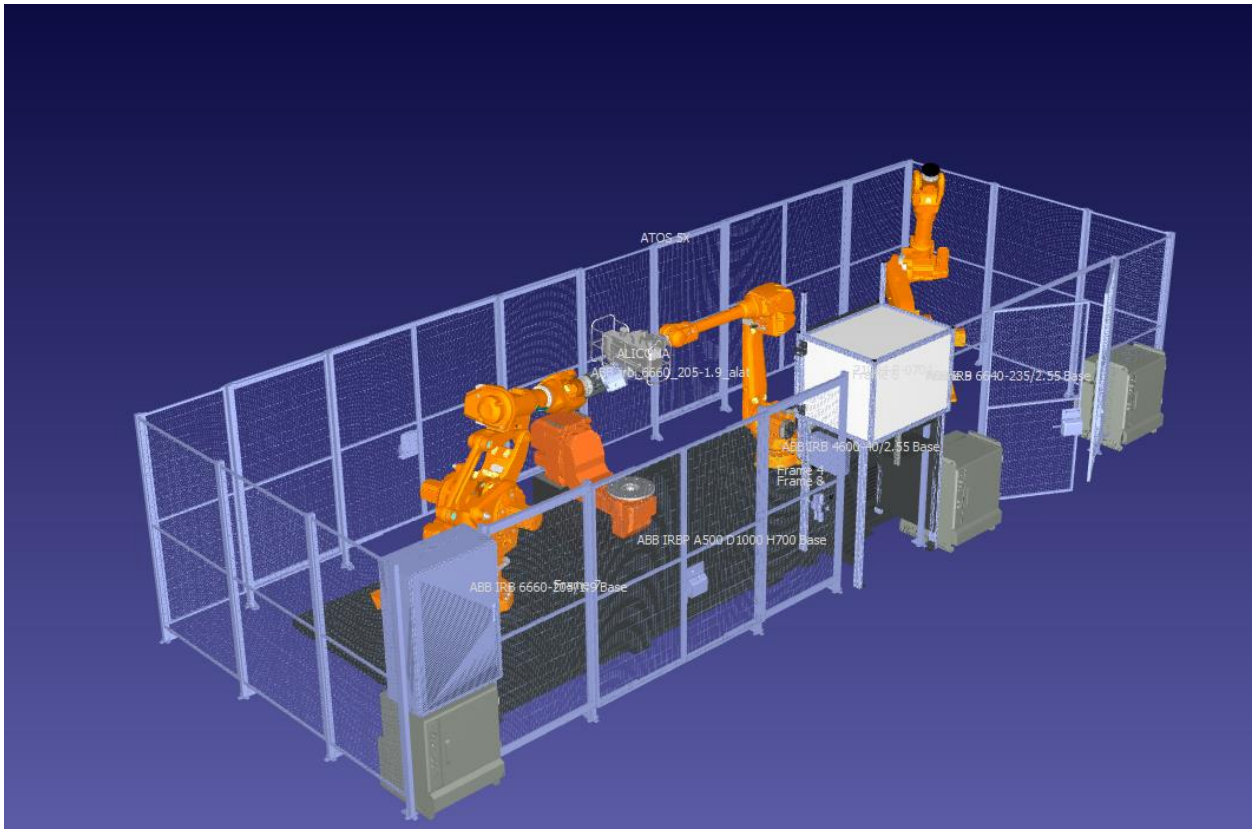
Slika 37 Procedura calAtosMv500

6. Konstrukcijsko rješenje naprave za prihvat kalibracijskih ploča i simulacija procesa kalibracije

Kako bi se automatizirao proces kalibracije, potrebno je pronaći poziciju postavljanja kalibracijskih ploča unutar robotske ćelije u kojoj mjerni robot može zauzeti sve pozicije potrebne za izvršavanje kalibracije bez kolizije s okolinom. Postupak kalibracije se provodi na način da se mjerna glava mora orijentirati i pozicionirati naspram kalibracijske ploče u ukupno 18 točaka. Zbog toga je, kao i zbog veličine robotske ćelije, broj takvih mjesta pozicioniranja kalibracijske ploče ograničen. Kako bi se odredile moguće pozicije kalibracije, korištena je aplikacija RoboDK (RoboDK je softver za simulaciju industrijskih robota i njihovo programiranje[9]).

6.1. Moguće pozicije kalibracije

Pomoću softvera RoboDK kreiran je model robotske ćelije ARCOPS kako bi bilo moguće postići vjernu simulaciju i iskoristiti softver za traženje odgovarajućih pozicija kalibracijske ploče. Robotska ćelija sastoji se od nepomičnih elemenata (npr. zaštitne ograde, elektroormara, postolja robota…) i pomičnih elemenata (tri industrijska robota, okretnog stola i okretno-nagibnog stola). Usto je radi simulacije procesa na robot dodan model mjernog uređaja Atos 5X. Slika 38 prikazuje robotsku ćeliju modeliranu u softveru RoboDK.



Slika 38 Ćelija ARCOPS u okružju softvera RoboDK

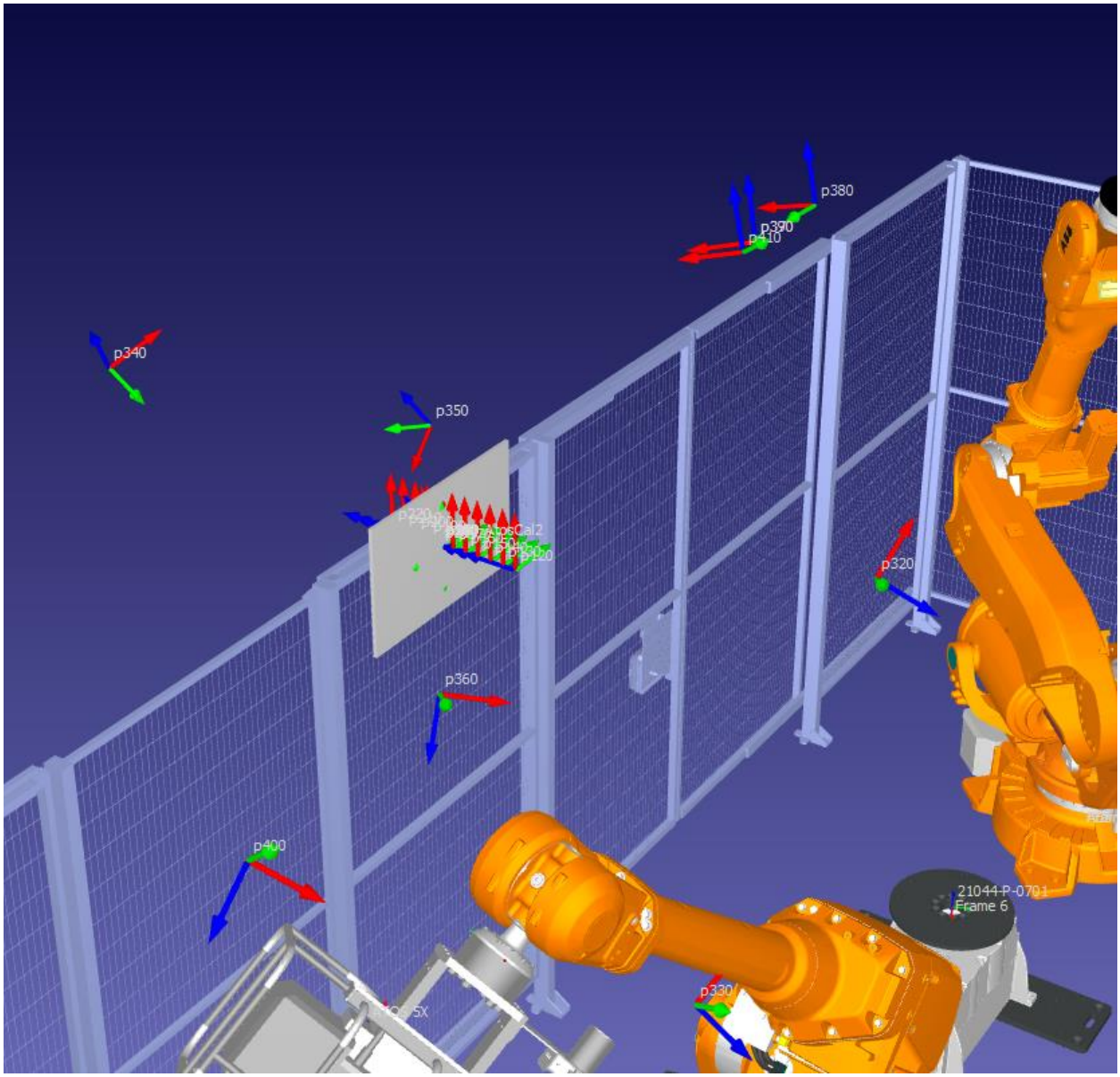
Prvi je korak u određivanju pozicije kalibracijske ploče definiranje pozicija u koje se Atos 5X mora postaviti da bi se izvršila kalibracija. Pozicija korisničkog koordinatnog sustava, u radu nazvanog *wobjAtosCal*, postavljena je u centar kalibracijske ploče, a usto je definiran koordinatni sustav alata Atos 5X. Alat je definiran na stvarnom robotu te su vrijednosti pozicije zapisane u programu. Tablica 2 prikazuje koordinate osamnaest točaka kalibracije s obzirom na definirani koordinatni sustav (*wobjAtosCal*) i definirani alat.

[*mm*]	X	Y	Z	u	v	w
Pozicija 1	0,00	0,00	0,00	180,00	0,00	-90,00
Pozicija 2	0,00	0,00	220,0	180,00	0,00	-90,00
Pozicija 3	0,00	0,00	185,00	180,00	0,00	-90,00
Pozicija 4	0,00	0,00	148,00	180,00	0,00	-90,00
Pozicija 5	0,00	0,00	111,00	180,00	0,00	-90,00
Pozicija 6	0,00	0,00	74,00	180,00	0,00	-90,00
Pozicija 7	0,00	0,00	37,00	180,00	0,00	-90,00
Pozicija 8	0,00	0,00	0,00	180,00	0,00	-90,00
Pozicija 9	0,00	0,00	-37,00	180,00	0,00	-90,00
Pozicija 10	0,00	0,00	-74,00	180,00	0,00	-90,00
Pozicija 11	0,00	0,00	-111,00	180,00	0,00	-90,00
Pozicija 12	0,00	0,00	-148,00	180,00	0,00	-90,00
Pozicija 13	0,00	0,00	0,00	155,00	0,00	-90,00
Pozicija 14	0,00	0,00	0,00	-155,00	0,00	-90,00
Pozicija 15	0,00	0,00	0,00	-180,00	22,00	-90,00
Pozicija 16	0,00	0,00	0,00	-180,00	25,00	0,00
Pozicija 17	0,00	0,00	0,00	90,00	45,00	180,00
Pozicija 18	0,00	0,00	0,00	180,00	45,00	180,00

Tablica 2 Koordinate točaka kalibracije

Jednom kad su definirane koordinate svih pozicija mjerne glave naspram kalibracijske ploče, one su u softveru RoboDK definirane kao točke u koje se robot mora pozicionirati. U tom se softveru mogu definirati korisnički koordinatni sustavi pa je definiran koordinatni sustav *wobjAtosCal*, koji predstavlja centar kalibracijske ploče. Prethodno definirane točke dodane su u taj novostvoreni koordinatni sustav. Time je omogućeno pomicanje samo koordinatnog sustava čime se sve točke pomiču zajedno s njim. Prednost toga je brže provjeravanje više pozicija nego kad bi se svaka

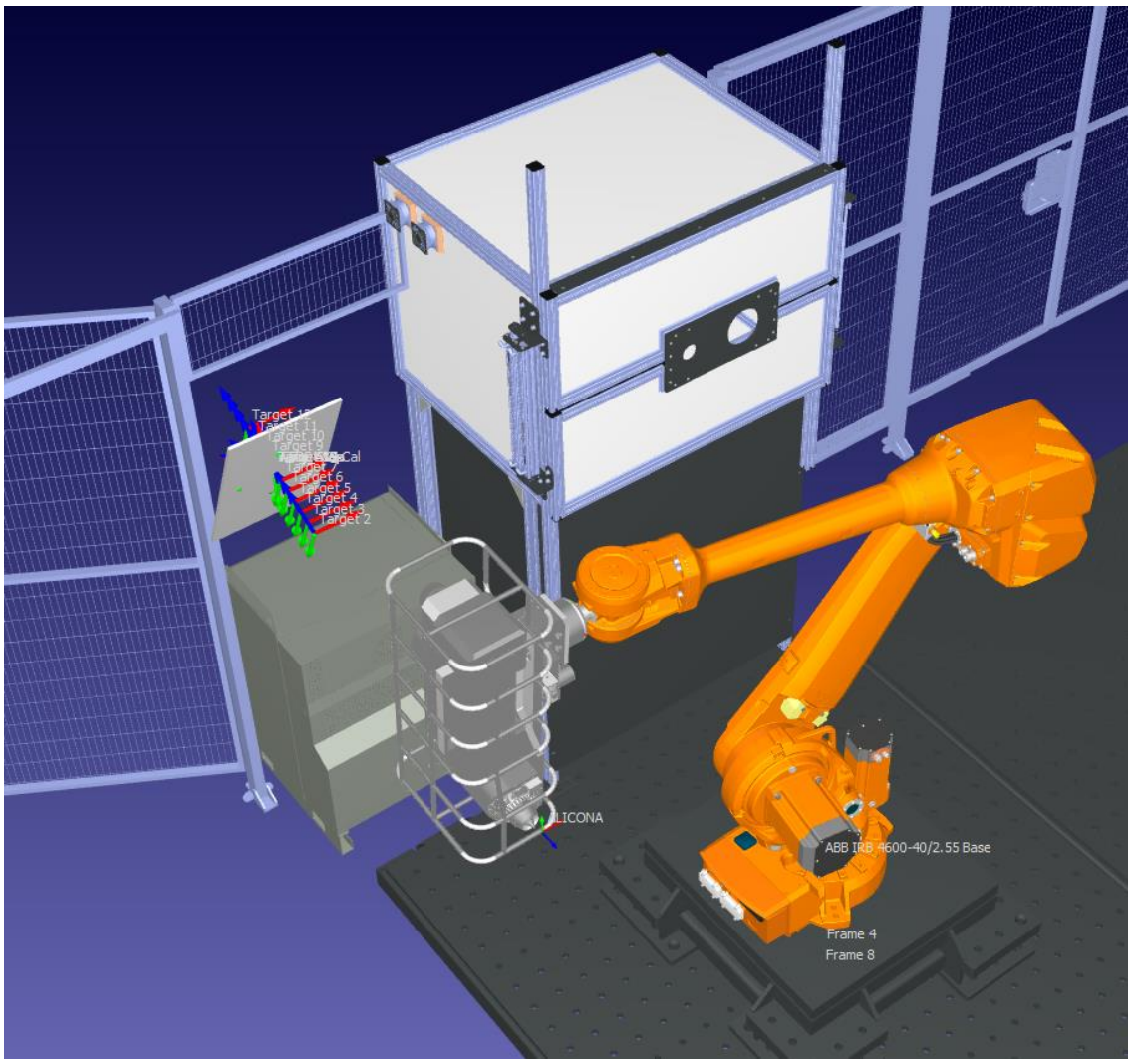
točka morala pomicati zasebno. Također osigurano je zadržavanje pozicija točaka. Kako bi se lakše vizualizirala kalibracijska ploča, usto je kreiran njezin 3D model. Slika 39 prikazuje definirani koordinatni sustav *wobjAtosCal* i relativne pozicije nužne za postupak kalibracije.



Slika 39 Koordinatni sustav wobjAtosCal s točkama za kalibraciju

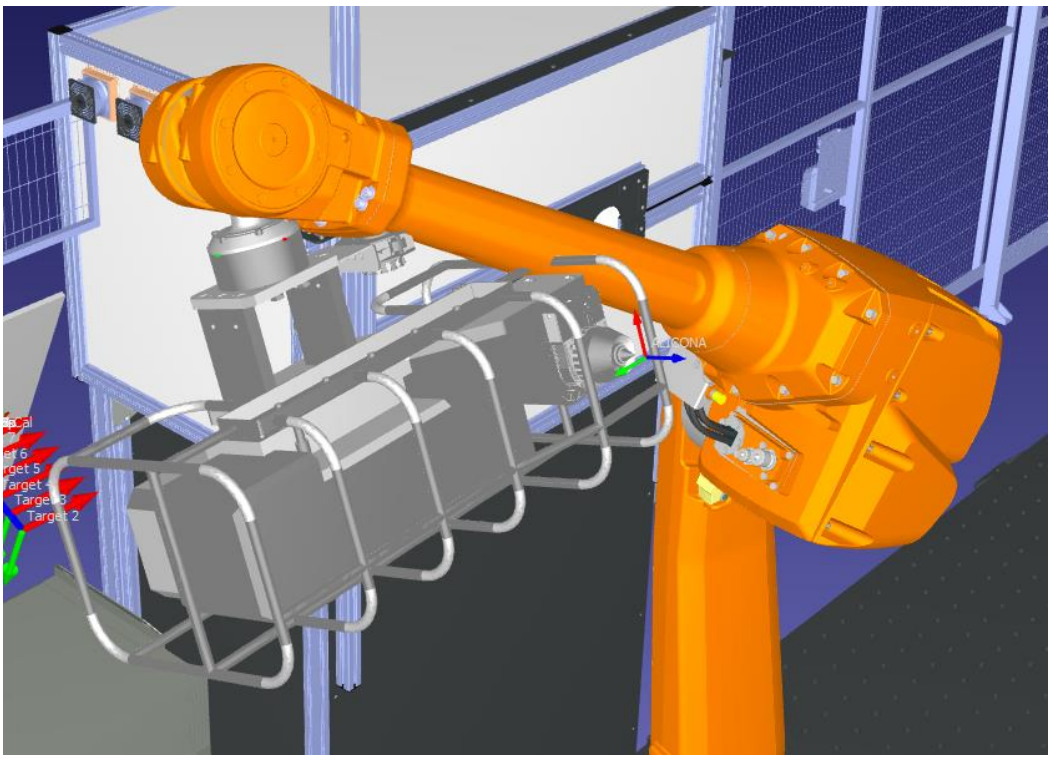
Traženje pozicije kalibracijske ploče svodi se na pomicanje koordinatnog sustava *wobjAtosCal* unutar robotske ćelije i provjeravanje valjanosti pozicije robota. Usto je potrebno omogućiti neometan rad ostatka ćelije bez obzira na proces kalibracije.

Prva pozicija iz koje je prema softveru moguće pozicionirati mjernu glavu u sve pozicije kalibracije prikazana je na slici 40.



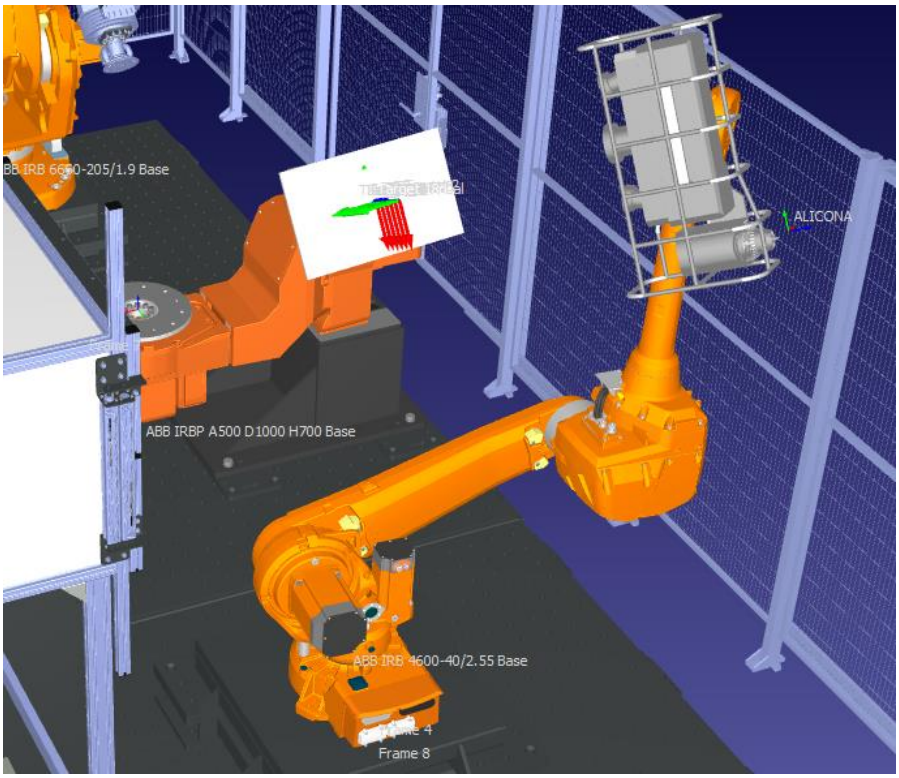
Slika 40 Prva pronađena pozicija kalibracijske ploče

Uvjet da pozicija kalibracijske ploče ne ometa rad ostatak robotske ćelije zadovoljen je. Međutim, provjerom pozicija robota u svim točkama tijekom kalibracije javlja se greška na nekoliko pozicija. U točki osamnaest mjerna glava dolazi u koliziju s robotom i iako Atos 5X dopušta odstupanje od 3 stupnja, pozicija se ne može ispraviti da osamnaesta pozicija kalibracije bude dostupna. Slika 41 prikazuje poziciju robota u osamnaestoj točki kalibracije. Usto u točki sedamnaest robot dolazi u koliziju s konstrukcijom ćelije, no ta se kolizija može riješiti pomicanjem točke za 2 stupnja, pri čemu ona ostaje unutar dozvoljenog odstupanja mjernog uređaja Atos 5X.



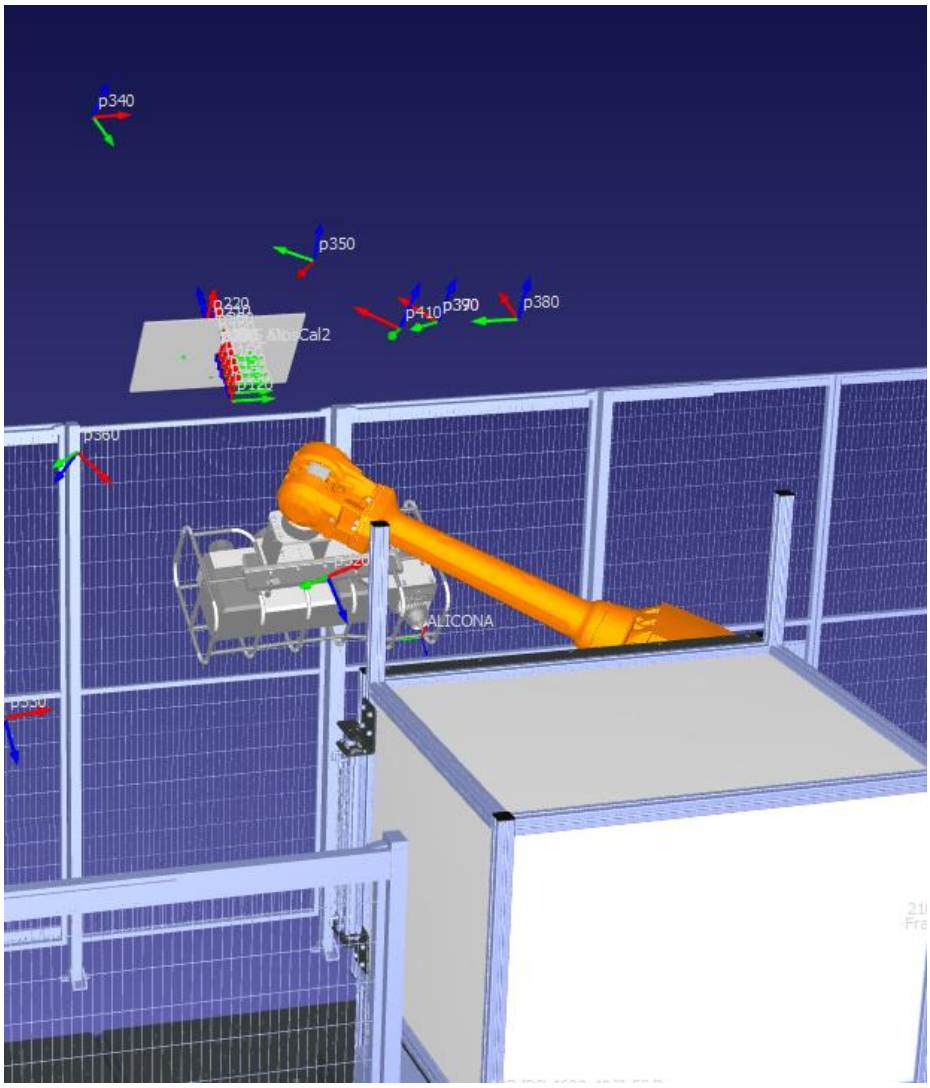
Slika 41 Greška pozicije u točki osamnaest

Sljedeća pronađena pozicija prikazana je na slici 42. U toj poziciju robot ne dolazi u nedozvoljene pozicije, ali kalibracijska se ploča nije mogla tako pozicionirati jer bi nosač na koji bi se trebala postaviti ometao robot u ostalim operacijama koje treba obavljati u automatskom radu ćelije.



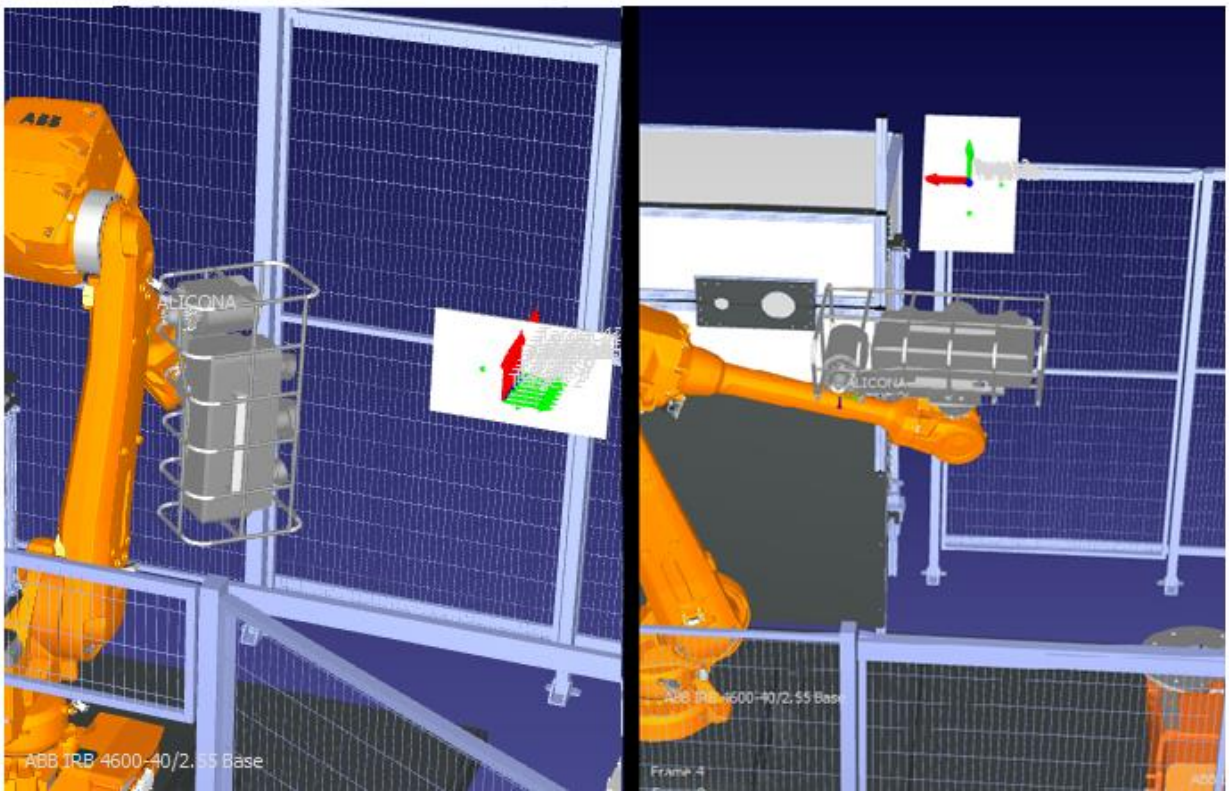
Slika 42 Druga pronađena pozicija kalibracijske ploče

Treća pronađena pozicija kalibracijske ploče zadovoljava sve uvjete koji su potrebni da bi se proces kalibracije automatizirao. Pozicija ne ometa rad ostatka ćelije i robot može neometano doći u sve točke kalibracije. Zbog toga je upravo ta pozicija odabrana kao konačna pozicija kalibracijske ploče. Prikazana je na slici 43. Konačna je pozicija središta kalibracijskih ploča, svih mjernih volumena, s obzirom na bazni koordinatni sustav robota: x = 510,00 *mm*, y = -1360,250 *mm*, z = 2150 *mm*, u = 0,00˚, v = 0,00˚ i w = -125,00˚.



Slika 43 Konačna pozicija kalibracijske ploče

Uz navedene pozicije kalibracijske ploče pronađeno je još nekoliko pozicija kalibracijskih ploča koje ne ometaju rad ćelije, ali je moguće ostvariti samo šesnaest od osamnaest pozicija kalibracije. Međutim, kad bi se kalibracijska ploča postavila u neku od tih pozicija, proces kalibracije ne bi bio potpuno automatiziran jer bi se kalibracijska ploča u posljednje dvije pozicije morala ručno zamicati, stoga su te pozicije odbačene. Dvije su takve pozicije prikazane su na slici 44.



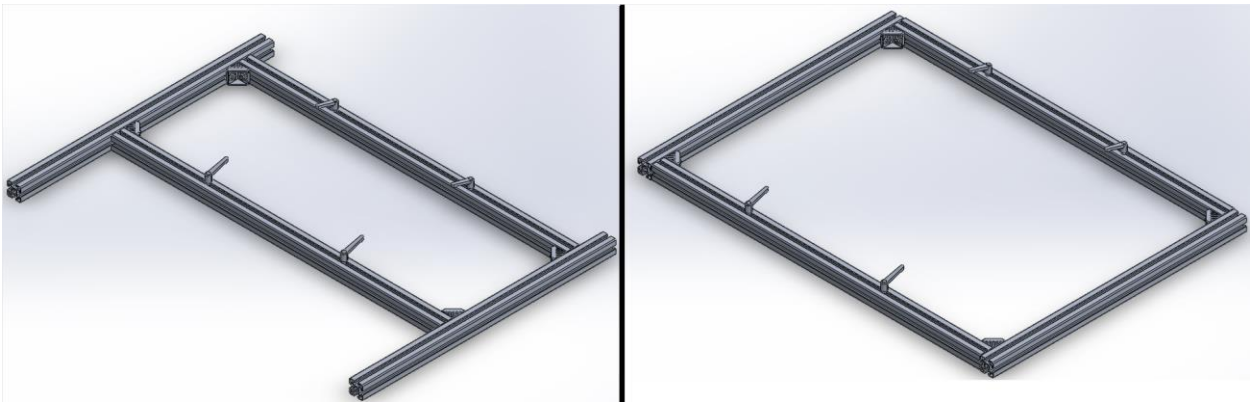
Slika 44 Dvije pozicije poluautomatske kalibracije

Na objema bi prikazanim pozicijama bilo potrebno jedno ručno okretanje tijekom kalibracije, za koje bi bio zadužen operater ćelije. Kalibracija bi se mogla izvršiti i na jednom od okretnih stolova unutar ćelije, ali time bi se narušila funkcionalnost ostatka ćelije tijekom kalibracije, a kalibracijska bi se ploča trebala ručno postaviti prije i ukloniti poslije procesa kalibracije.

6.2. Konstrukcijsko rješenje naprave za prihvat kalibracijskih ploča

Konstrukcijsko rješenje naprave za prihvat kalibracijskih ploča modelirano je nakon što je određena potrebna pozicija kalibracijskih ploča unutar robotske ćelije. Koncept konstrukcijskog rješenja omogućuje prihvat kalibracijskih ploča različitih dimenzija te njihovo pozicioniranje unutar robotske ćelije. Konstrukcija se sastoji od standardnih aluminijskih profila dimenzija 45x45 *mm* i elemenata za povezivanje profila. Koncept konstrukcije je modeliran pomoću programskog paketa SolidWorks.

Slika 45 prikazuje dio naprave za prihvat kalibracijskih ploča. Lijeva slika prikazuje konfiguraciju za kalibracijsku ploču MV500, a desna za kalibracijsku ploču MV1000. Kako je vidljivo na slici, razmicanjem aluminijskih profila ili njihovim približavanjem omogućen je prihvat kalibracijskih ploča različitih dimenzija. Dio za prihvat kalibracijskih ploča također sadržava četiri poluge na koje se oslanjaju kalibracijske ploče kako bi one ostale nepomične tijekom mijenjanja nagiba naprave.



Slika 45 Dvije konfiguracije dijela za prihvat kalibracijskih ploča

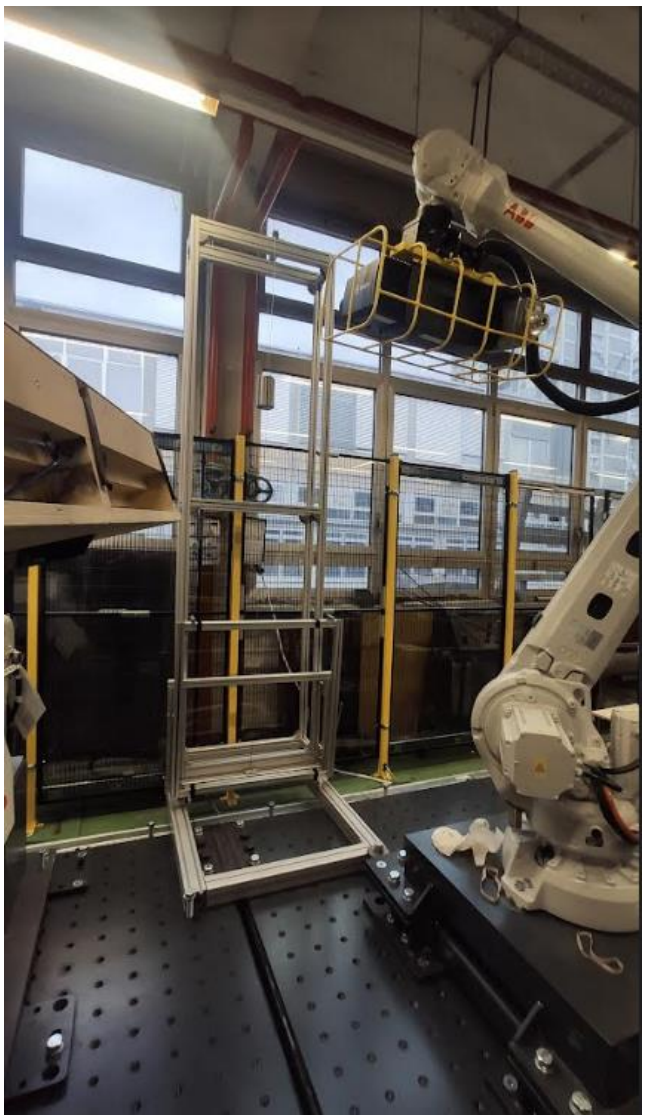
Slika 46 prikazuje dio naprave koji omogućuje podizanje kalibracijskih ploča na željenu visinu. Također omogućuje montiranje naprave na postolje robotske ćelije.

Slika 46 Dio koncepta naprave za prihvat ploča

Cjelovita naprava prikazana je na slici 47. Sastoji se od prethodno prikazanih pozicija te veznih elemenata koji omogućuju pomicanje kalibracijskih ploča po visini te osiguravanje potrebnog nagiba.

Slika 47 Naprava za prihvat kalibracijskih ploča

Sama naprava izrađena je na Katedri za alatne strojeve Fakulteta strojarstva i brodogradnje u Zagrebu. Montirana je u robotsku ćeliju i prikazana na slici 48. Usto je na napravu dodan protuuteg koji omogućava lakše podizanje kalibracijskih ploča većih dimenzija na zahtijevanu visinu. Tehnička dokumentacija se nalazi u prilogu rada.

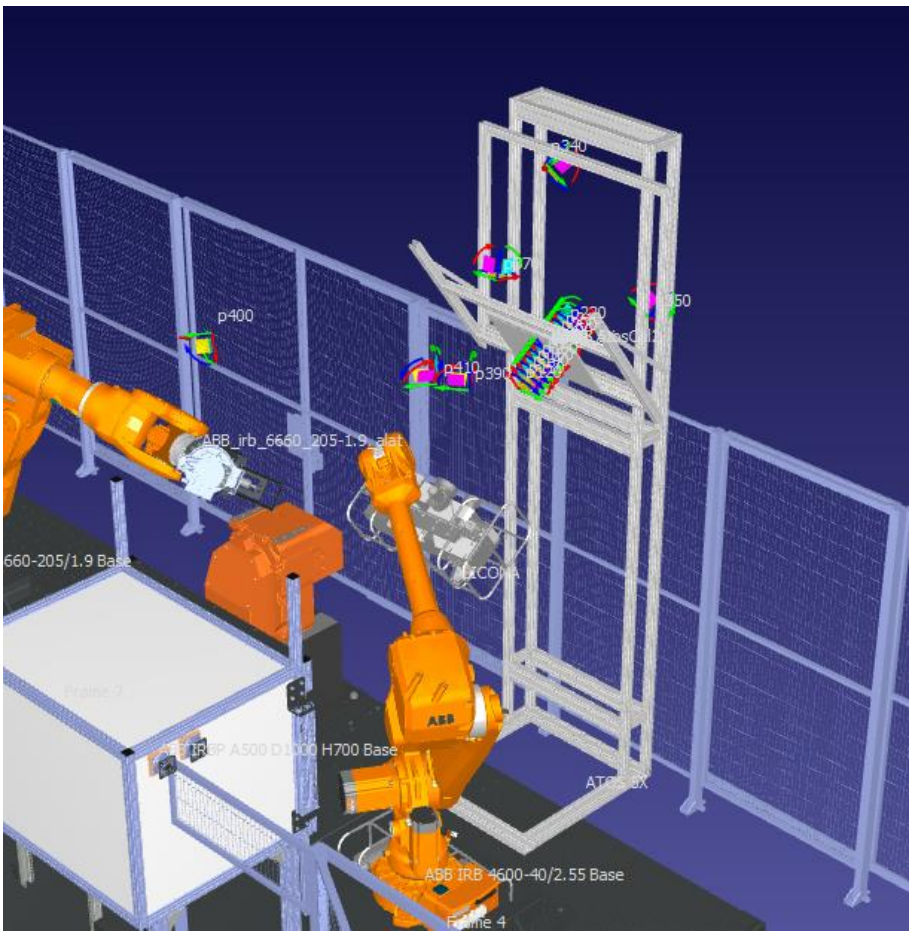


Slika 48 Montirana naprava za prihvat kalibracijskih ploča

6.3. Simulacija gibanja s međutočkama pomoću softvera RoboDK

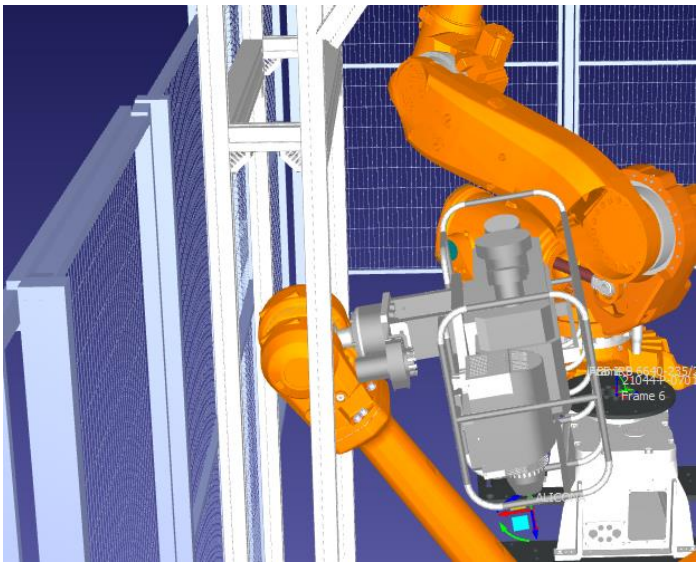
U prethodno spomenutu robotsku ćeliju u sklopu softvera RoboDK dodana je naprava za pozicioniranje. Postavljena je na predviđeno mjesto, a zatim je izrađen program u koji su dodane međutočke potrebne za izvođenje procesa kalibracije. Međutočke su određene pomicanjem robota u dodatne točke između dvije zahtijevane pozicije kako bi se izbjegla kolizija robota s ostatkom robotske ćelije.

Slika 49 prikazuje početnu poziciju robota tijekom programa automatskog kalibriranja. Programi kalibracije različitih kalibracijskih ploča razlikuju se samo po tome koji se koordinatni sustav alata koristi, a u ovom se radu prikazuje program koji je izrađen za kalibracijsku ploču MV500. Program se izvršava bez kolizija s ostalim dijelovima ćelije, a robot može doći u sve tražene pozicije.



Slika 49 Početna pozicija programa automatske kalibracije

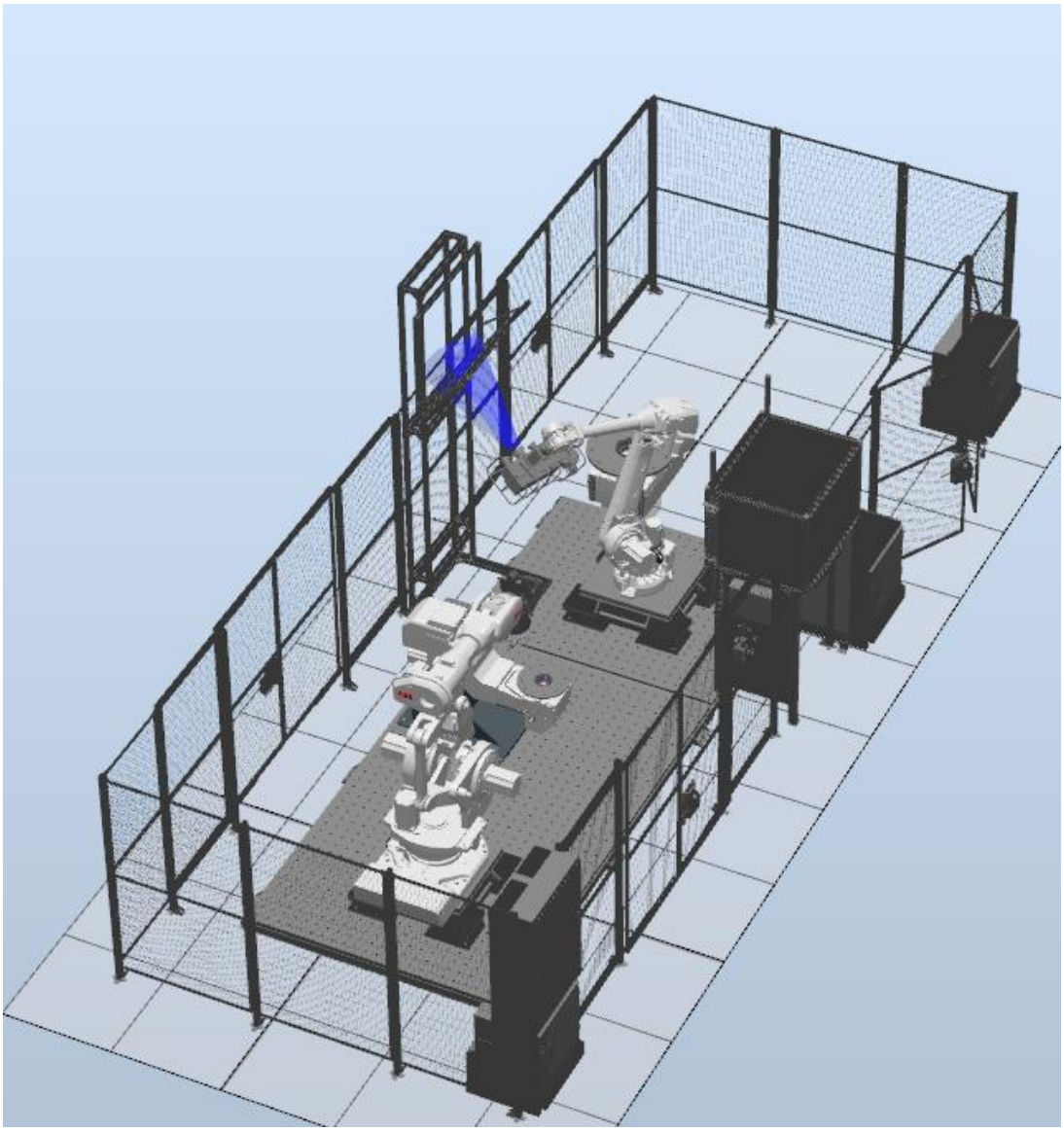
Tijekom kalibracije dodano je pet međutočaka kako bi se maksimalno smanjila mogućnost kolizije. U jednu od nepogodnih pozicija robot dolazi u sedamnaestoj točki kalibracije, kad robot se približava rubu ćelije. Zbog toga je brzina dolaska u tu poziciju smanjena kako bi se u slučaju greške programa potencijalna šteta maksimalno smanjila. Pozicija je prikazana na slici 50.



Slika 50 Pozicija robota u sedamnaestoj točki kalibraciji

7. Simulacija rada izrađenog programskog rješenja i kretnji robota pomoću aplikacije ABB RobotStudio

Iako je simulacija inicijalno izrađena u okružju softvera RoboDK, ona ne može kreirati potpuno vjernu simulaciju stvarnog robota. Radi dodatne provjere u aplikaciji ABB RobotStudio kreirana je nova robotska ćelija, modelirana prema robotskoj ćeliji ARCOPS [10]. ABB RobotStudio omogućava stvaranje kopije stvarnog robota i simulira postojeće kontrolere robota. Time se omogućava stvaranje tzv. „digitalnog blizanca“ stvarnog robota. Dobivena simulacija predstavlja stvarne kretnje robota. Kreirana robotska ćelija sa svim elementima potrebnim za simulaciju prikazana je na slici 51.



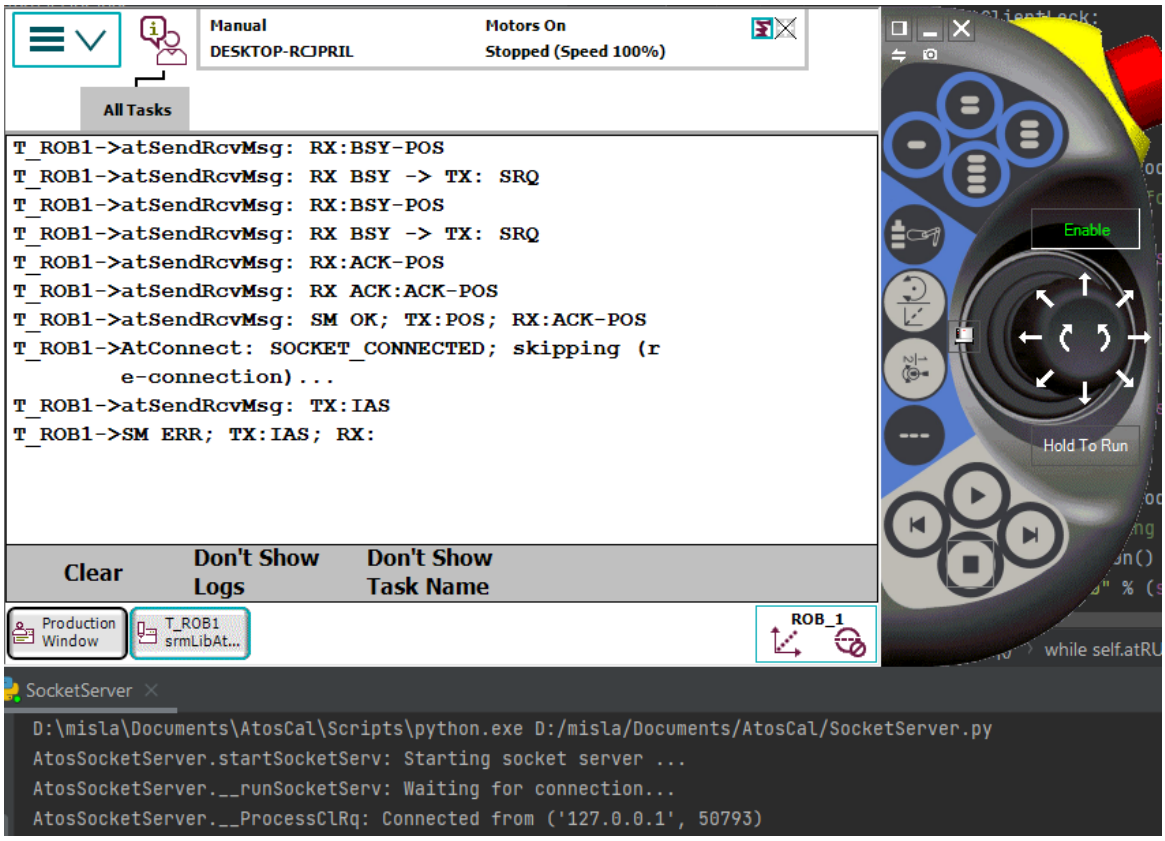
Slika 51 Robotska ćelija u aplikaciji ABB RobotStudio

Radi vizualizacije mjernog volumena na alat je dodana simulacija rada projektora mjernog uređaja Atos 5X.

U izradi simulacije korišteni su programsko rješenje opisano u 5. poglavlju, programski modul prikazan u potpoglavlju 5.5. (*calAtos*) i točke dobivene pomoću aplikacije RoboDK.

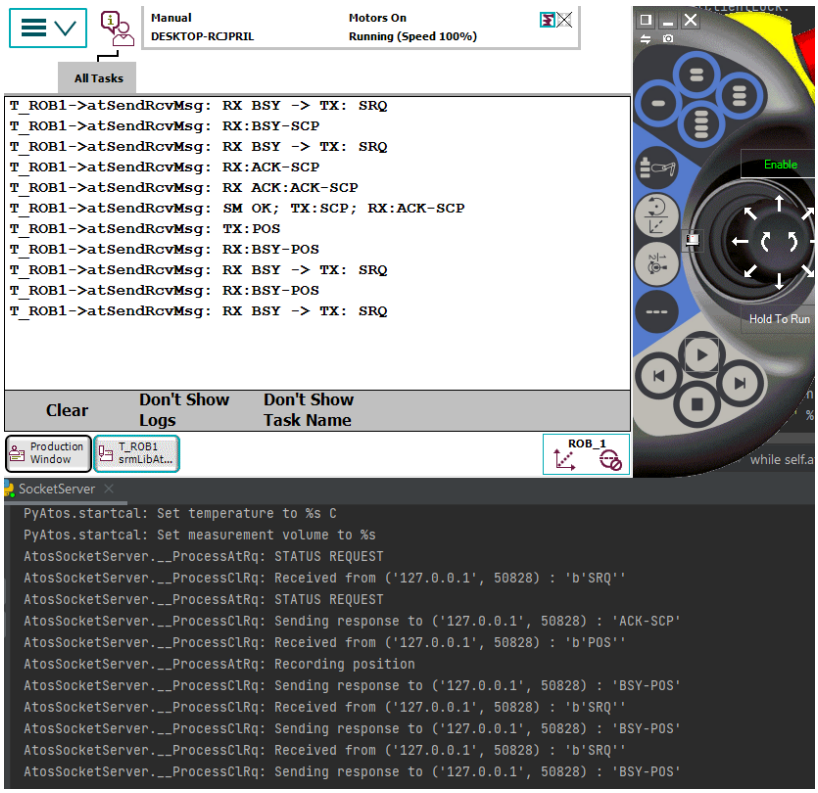
Modul *PyAtos* je prilagođen kako bi se mogao izvršavati bez korištenja mjernog sustava Atos 5X. Sadrži sve opisane metode, ali one ne izvršavaju naredbe GOM softvera nego čekaju pet sekundi i ispisuju koja se metoda izvršava.

Prilikom pokretanja programa poslužitelja pojavljuje se prozor koji omogućava upravljanje izvršavanjem programa i ispisuje se poruka da je poslužitelj spreman za povezivanje. Odabirom procedure *calAtosMv500* pomoću simuliranog privjeska za učenje robota pokreće se povezivanje upravljačkog sustava robota s poslužiteljem. Povezivanje se vrši pozivanjem prethodno objašnjene procedure *atConnect*. Slika 52 prikazuje zaslon privjeska za učenje i konzole Python interpretera prilikom povezivanja.



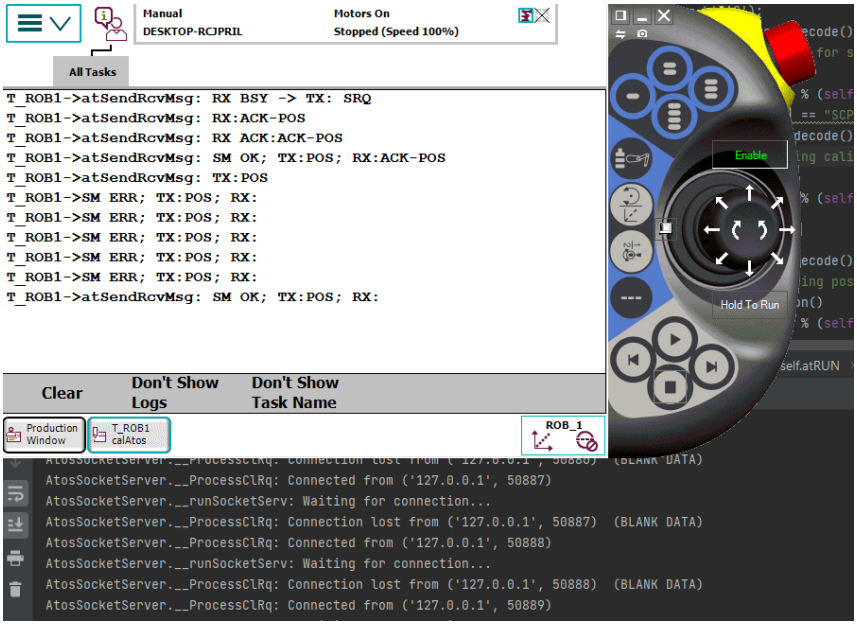
Slika 52 Uspostavljanje veze poslužitelja i klijenta

Kako je opisano u 5. poglavlju, poslužitelj i klijent konstantno komuniciraju te ovisno o prefiksu dobivenih poruka odlučuju o sljedećoj radnji. Izmjena poruka poslužitelja i klijenta prikazana je na slici 53.



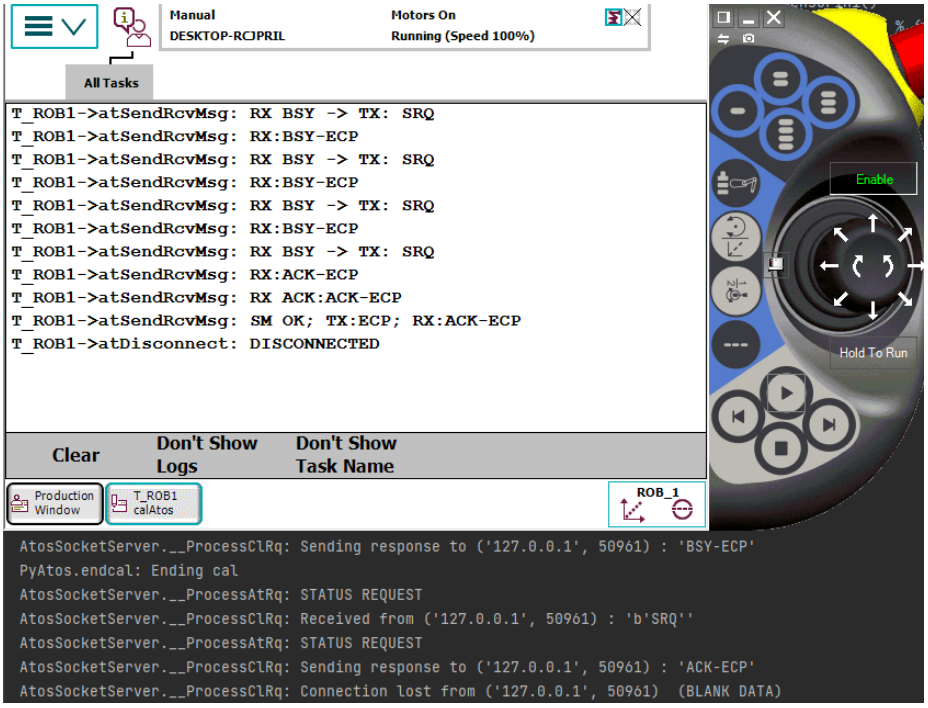
Slika 53 Izmjena podataka između poslužitelja i klijenta

Usporedno s tom komunikacijom robot se pomiče u tražene točke kalibracije. Ako dođe do prekida veze zbog predugog čekanja poslužitelja na zahtjev klijenta, pokreće se procedura *atReconnect.* Procedura *atReconnect* pokušava ponovno uspostaviti vezu klijenta s poslužiteljem kako bi se program mogao nastaviti izvršavati. Vrijeme kretanja robota do određenih pozicija nadmašuje definirano vrijeme neaktivnosti komunikacije poslužitelja i klijenta te radi toga dolazi do prekida veze. Proces ponovnog uspostavljanja veze prikazan je na slici 54.



Slika 54 Ponovna uspostava veze poslužitelja i klijenta

Na kraju kalibracije dolazi do prekida veze poslužitelja i klijenta, a obavijest o tome ispisuje se na konzoli Python interpretera i na zaslonu privjeska za učenje. Slika 55 prikazuje obavijest o prekidu veze.



Slika 55 Obavijest o prekidu veze

Tijekom izvođenja programa poslužitelja moguć je ispis svih aktivnih niti i trenutnog stanja izvršavanja programa na konzoli te ispis u grafičko sučelje. Prikaz ispisa stanja i aktivnih niti prikazan je na slici 56.

```
AtosSocketServer.state:      IDLE
AtosSocketServer.printThreads: --------------| RUNNING THREADS: |--------------
AtosSocketServer.printThreads:       MainThread
AtosSocketServer.printThreads:       SocketServerMainThread
AtosSocketServer.printThreads:       AtosThread
AtosSocketServer.printThreads: -------| SOCKET SERVER CLIENT THREADS: |--------
AtosSocketServer.printThreads:       SocketClientThread28
AtosSocketServer.printThreads: ------------------------------------------------
```

Slika 56 Aktivno stanje i aktivne niti

Klikom na gumb *Stop program* završava se izvođenje programa poslužitelja.

U simulaciji su promijenjene dvije međutočke dobivene pomoću aplikacije RoboDK. Time je dobivena simulacija koja je provedena bez kolizije robota s bilo kojim dijelom ćelije.

8. Zaključak

U ovom je radu opisan sustav integracije mjernog uređaja ATOS 5X proizvođača GOM s robotskim sustavom ABB IRB4600. Razvijen je upravljački program koji omogućuje automatsku kalibraciju mjernog uređaja, što je bio zadatak ovog diplomskog rada.

U sklopu integracije bilo je potrebno uspostaviti komunikaciju između mjernog uređaja i upravljačkog sustava robota. Komunikacijski model poslužitelj-klijent implementiran je pomoću modula pristupne točke (eng. *socket*) koji koristi funkcionalnost transportnog sloja računalne mreže temeljene na TCP/IP protokolu. Poslužitelj predstavlja program implementiran unutar upravljačkog računala mjernog uređaja napisanog u programskom jeziku Python. Klijent predstavlja program na upravljačkom sustavu robota napisanog u programskom jeziku RAPID. Razvijeni model funkcionira na način da klijent šalje zahtjeve poslužitelju za povezivanje i izvršavanje naredbi na mjernom uređaju. Pri tome je osigurana je konstanta komunikacija između poslužitelja i klijenta te mogućnost ponovne uspostave veze u slučaju prekida. Također, definirani su prefiksi poruka kojima se provjerava stanje izvršavanja pojedinih zahtjeva čime se ostvaruje kontinuirana komunikacija između klijenta i poslužitelja.

Točke kalibracije unesene su u aplikaciju RoboDK pomoću koje su određene moguće pozicije kalibracijskih ploča unutar robotske ćelije. Kad je dobivena pozicija kalibracijskih ploča unutar robotske ćelije, izrađen je koncept konstrukcije naprave za prihvat ploča popraćen dokumentacijom koja je potrebna za njezinu izradu.

Simulacija procesa automatske kalibracije izrađena je u dvama aplikacijama, RoboDK i RobotStudio. RoboDK omogućava bržu izradu simulacije, ali je skloniji greškama jer ne koristi stvarni kontroler robota. RobotStudio daje realnu simulaciju stvarnog procesa jer koristi stvaran kontroler robota. Razlika između simulacija vidljiva je u različitim kretnjama u pojedine točke prilikom simulacije procesa. Simuliran je proces uspostavljanja veze poslužitelja i klijenta te je provjerena funkcionalnost izrađenog modela. Potvrđeno je da je moguće ponovno uspostaviti vezu i nastaviti program ukoliko dođe do prekida veze te prekinuti rad poslužitelja.

Sljedeći je korak provjera dobivenog rješenja u stvarnim uvjetima uz potencijalno optimiziranje putanje robota tijekom kalibracije. Potrebno je provjeriti stvarnu funkcionalnost koncepta naprave i napraviti modifikacije ovisno o dobivenim rezultatima. Usto je potrebno proširiti funkcionalnost programskog rješenja.

Literatura

[1] GOM, User Manual Hardware ATOS 5X, 2021.

[2] ABB Product specification – IRB 4600, pristupljeno 11. travnja 2023.

[3] https://new.abb.com/products/3HAC020536-014/irc5-controller,pristupljeno 11. travnja 2023.

[4]ABB Operating manual, Introduction to RAPID, pristupljeno 13. travnja 2023.

[5] Parziale, L., Liu, W., Matthews, C., Rosselot, N., Davis, C., Forrester, J., & Britt, D. T. (2006). TCP/IP tutorial and technical overview.

[6] https://docs.python.org/3/library/socket.html, pristupljeno 13. travnja 2023.

[7] https://docs.python.org/3/library/threading.html, pristupljeno 13. travnja 2023.

[8]Technical reference manual RAPID Instructions, Functions and Data types, pristupljeno 15. travnja 2023.

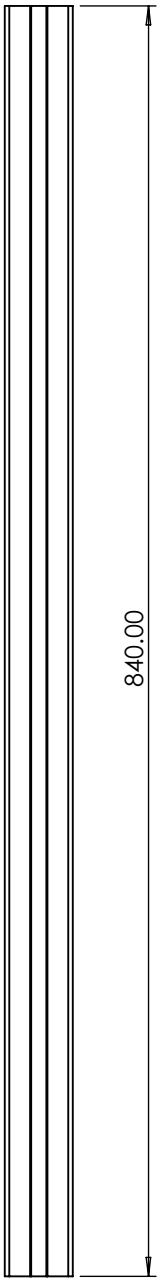
[9] https://robodk.com/about, pristupljeno 06. svibnja 2023.

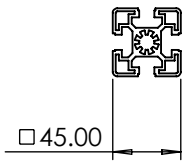
[10] https://developercenter.robotstudio.com/api/robotstudio/api/index.html, pristupljeno 08. svibnja 2023.

Prilozi:

I. CD-R

II. Tehnička dokumentacija





	Datum	Ime i prezime	Potpis
Projektirao	06.05.2023	Mislav Požgaj	
Razradio	06.05.2023	Mislav Požgaj	
Crtao	06.05.2023	Mislav Požgaj	
Pregledao			

FSB Zagreb

Objekt:

Objekt broj:

R. N. broj:

Napomena:

Kopija

Materijal: EN AW – Al MgSi

Masa: 1.68kg

Mjerilo originala

1:5

Naziv: 45x45x840 aluminijski profil

Pozicija: 1

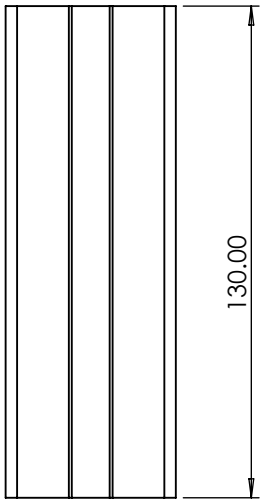
Format: A4

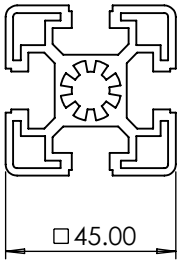
Listova: 1

Crtež broj: 2023-001

List: 1

Design by CADLab

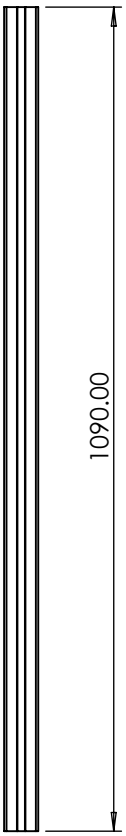


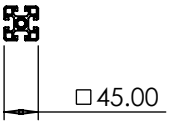


	Datum	Ime i prezime	Potpis
Projektirao	06.05.2023	Mislav Požgaj	
Razradio	06.05.2023	Mislav Požgaj	
Crtao	06.05.2023	Mislav Požgaj	
Pregledao			

FSB Zagreb

Objekt:	Objekt broj:
	R. N. broj:
Napomena:	Kopija
Materijal: ENAW–Al MgSi	Masa: 0.26kg
Mjerilo originala 1:2	Naziv: 45x45x130 aluminijski profil
Pozicija: 2	Format: A4
Listova: 1	List: 1
Crtež broj: 2023-002	



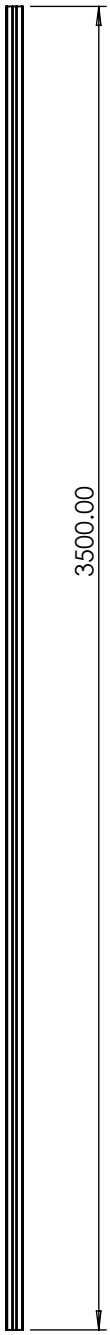


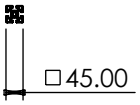
	Datum	Ime i prezime	Potpis
Projektirao	06.05.2023	Mislav Požgaj	
Razradio	06.05.2023	Mislav Požgaj	
Crtao	06.05.2023	Mislav Požgaj	
Pregledao			

FSB Zagreb

Objekt:	Objekt broj:
	R. N. broj:
Napomena:	Kopija
Materijal: ENAW–Al MgSi	Masa: 2.16kg
Mjerilo originala 1:10	Naziv: 45x45x1090 aluminijski profil
Pozicija: 3	Format: A4
Listova: 1	List: 1
Crtež broj: 2023-003	

Design by CADLab



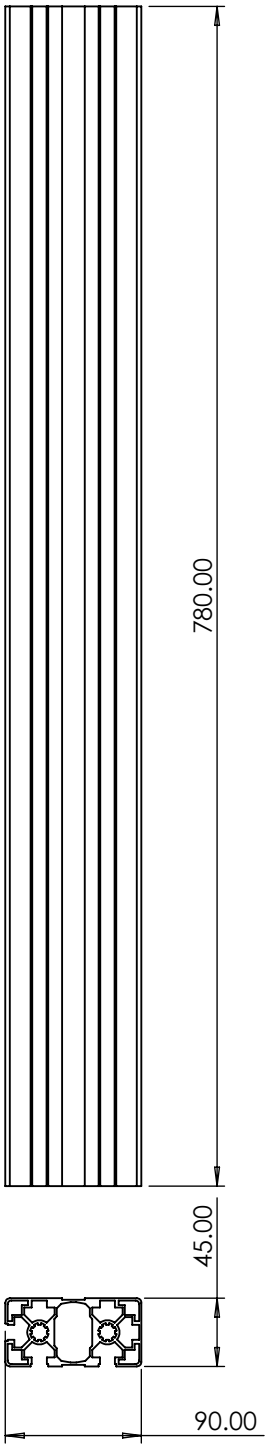


	Datum	Ime i prezime	Potpis
Projektirao	06.05.2023	Mislav Požgaj	
Razradio	06.05.2023	Mislav Požgaj	
Crtao	06.05.2023	Mislav Požgaj	
Pregledao			

FSB Zagreb

Objekt:	Objekt broj:
	R. N. broj:
Napomena:	Kopija
Materijal: ENAW–Al MgSi	Masa: 6.99kg
Mjerilo originala 1:5	Naziv: 45x45x3500 aluminijski profil
Pozicija: 4	Format: A4
Listova: 1	List: 1
Crtež broj: 2023-004	

Design by CADLab

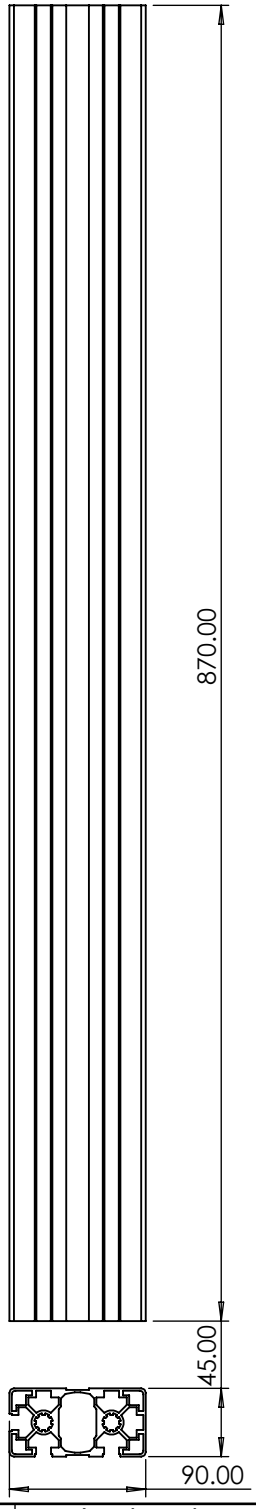


	Datum	Ime i prezime	Potpis
Projektirao	06.05.2023	Mislav Požgaj	
Razradio	06.05.2023	Mislav Požgaj	
Crtao	06.05.2023	Mislav Požgaj	
Pregledao			

FSB Zagreb

Objekt:	Objekt broj:
	R. N. broj:
Napomena:	Kopija
Materijal: ENAW–Al MgSi	Masa: 2.45kg
Mjerilo originala 1:5	Naziv: 45x90x780 aluminijski profil
Pozicija: 5	Format: A4
Listova: 1	List: 1
Crtež broj: 2023-005	

Design by CADLab

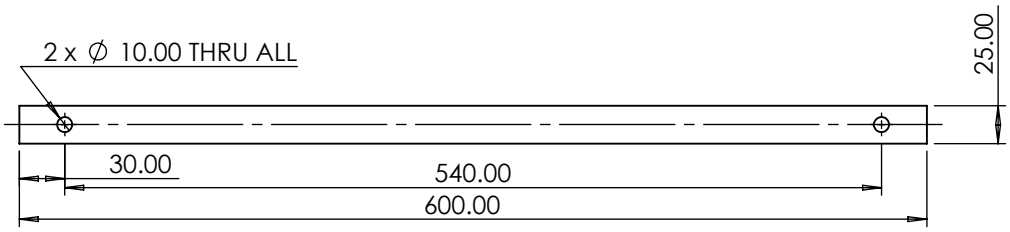


	Datum	Ime i prezime	Potpis
Projektirao	06.05.2023	Mislav Požgaj	
Razradio	06.05.2023	Mislav Požgaj	
Crtao	06.05.2023	Mislav Požgaj	
Pregledao			

FSB Zagreb

Objekt:	Objekt broj:
	R. N. broj:
Napomena:	Kopija
Materijal: ENAW–Al MgSi	Masa: 2.73kg
Mjerilo originala 1:5	Naziv: 45x90x870 aluminijski profil
Pozicija: 6	Format: A4
Listova: 1	List: 1
Crtež broj: 2023-006	

Design by CADLab

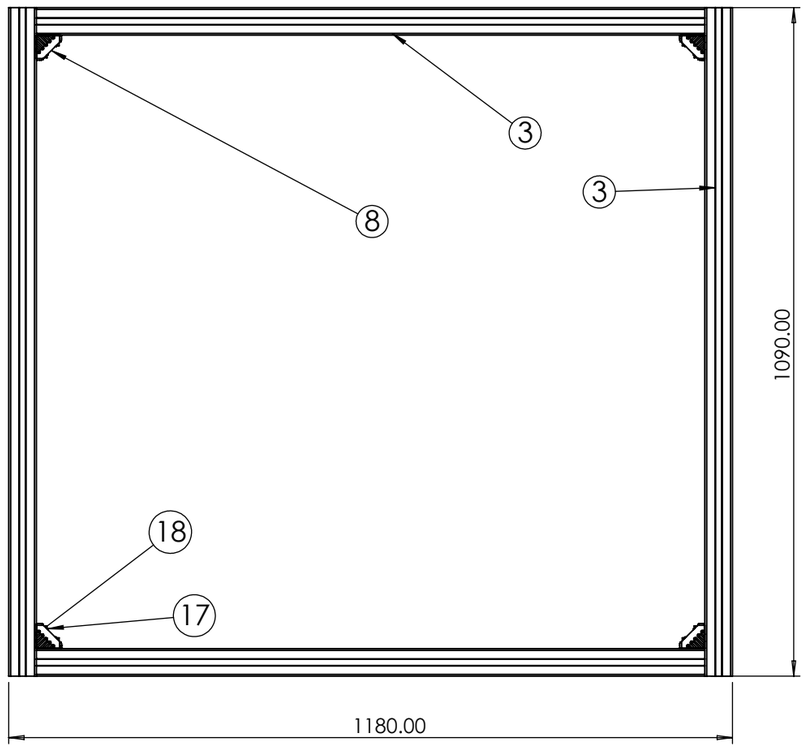


	Datum	Ime i prezime	Potpis
Projektirao	06.05.2023	Mislav Požgaj	
Razradio	06.05.2023	Mislav Požgaj	
Crtao	06.05.2023	Mislav Požgaj	
Pregledao			

FSB Zagreb

Objekt:	Objekt broj:
	R. N. broj:
Napomena: Nekotirana debljina je 3 mm.	Kopija
Materijal: ENAW–Al MgSi	Masa: 0.12kg
Mjerilo originala 1:5	Naziv: Ploča za osiguravanje nagiba
Pozicija: 7	Format: A4
Listova: 1	List: 1
Crtež broj: 2023-007	

Design by CADLab



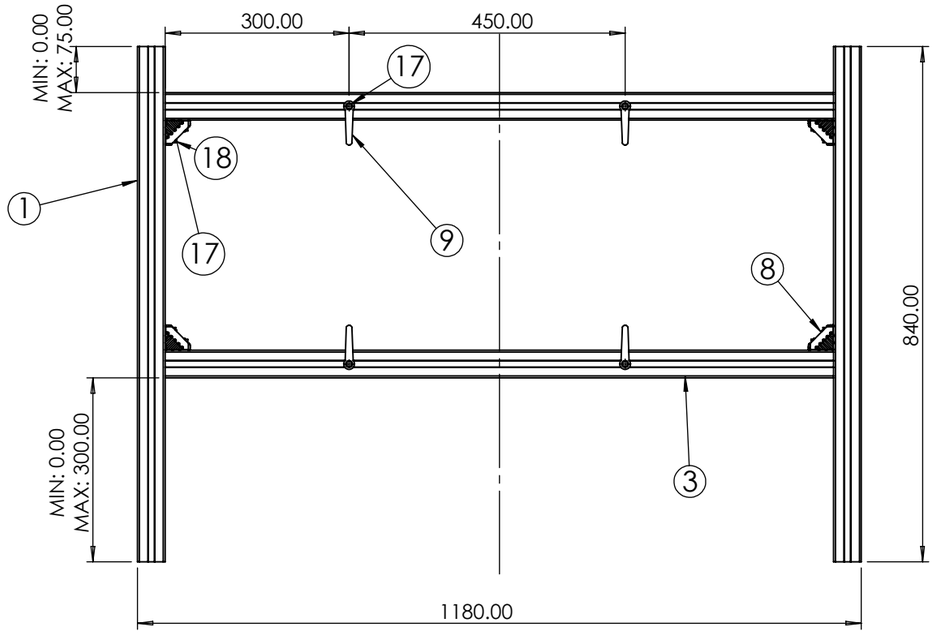
18	M8 MATICA	48	ISO 4032	5	Bosch-Rexroth	0.01 kg
17	T-VIJAK 8, M8X25	48	DIN 7992	6.8	Bosch-Rexroth	0.01 kg
8	45X45 KUTNIK	4	3842523558	ENAW–Al MgSi	Bosch-Rexroth	0.11 kg
3	45x45x1090 ALU PROFIL	4	2023-003	ENAW–Al MgSi	Bosch-Rexroth	2.16 kg
Poz.	Naziv dijela	Kom.	Crtež broj Norma	Materijal	Sirove dimenzije Proizvođač	Masa

Broj naziva - code		Datum	Ime i prezime	Potpis	
	Projektirao	06.05.2023	Mislav Požgaj		FSB Zagreb
	Razradio	06.05.2023	Mislav Požgaj		
	Crtao	06.05.2023	Mislav Požgaj		
	Pregledao				
	Mentor		Tomislav Staroveški		

ISO - tolerancije	Objekt:	Objekt broj:
		R. N. broj:
	Napomena:	Kopija
	Materijal: ENAW–Al MgSi Masa: 9.09kg	
	Mjerilo originala 1:10	Naziv: Sklop klizača nosača
		Pozicija: 11
		Format: A3
		Listova: 1
	Crtež broj: 2023-A-008	List: 1

Design by CADLab

0 10 20 30 40 50 60 70 80 90 100



Poz.	Naziv dijela	Kom.	Crtež broj Norma	Materijal	Sirove dimenzije Proizvođač	Masa
18	M8 MATICA	8	ISO 4032	5	Bosch-Rexroth	0.01 kg
17	T-VIJAK 8, M8X25	12	DIN 7992	6.8	Bosch-Rexroth	0.01 kg
9	POLUGA PROFILA	4	GN300-63-M8-55-SW	ENAW–Al MgSi	Bosch-Rexroth	0.03 kg
8	45X45 KUTNIK	4	3842523558	ENAW–Al MgSi	Bosch-Rexroth	0.07 kg
3	45x45x1090 ALU PROFIL	2	2023-003	ENAW–Al MgSi	Bosch-Rexroth	2.16 kg
1	45x45x840 ALU PROFIL	2	2023-001	ENAW–Al MgSi	Bosch-Rexroth	1.68 kg

Broj naziva - code		Datum	Ime i prezime	Potpis
	Projektirao	06.05.2023	Mislav Požgaj	
	Razradio	06.05.2023	Mislav Požgaj	
	Crtao	06.05.2023	Mislav Požgaj	
	Pregledao			
	Mentor		Tomislav Staroveški	

FSB Zagreb

ISO - tolerancije

Objekt:

Objekt broj:

R. N. broj:

Napomena:

Kopija

Materijal: ENAW–Al MgSi

Masa: 8.23kg

Mjerilo originala 1:10

Naziv: Sklop nosača kalibracijskih panela

Pozicija: 12

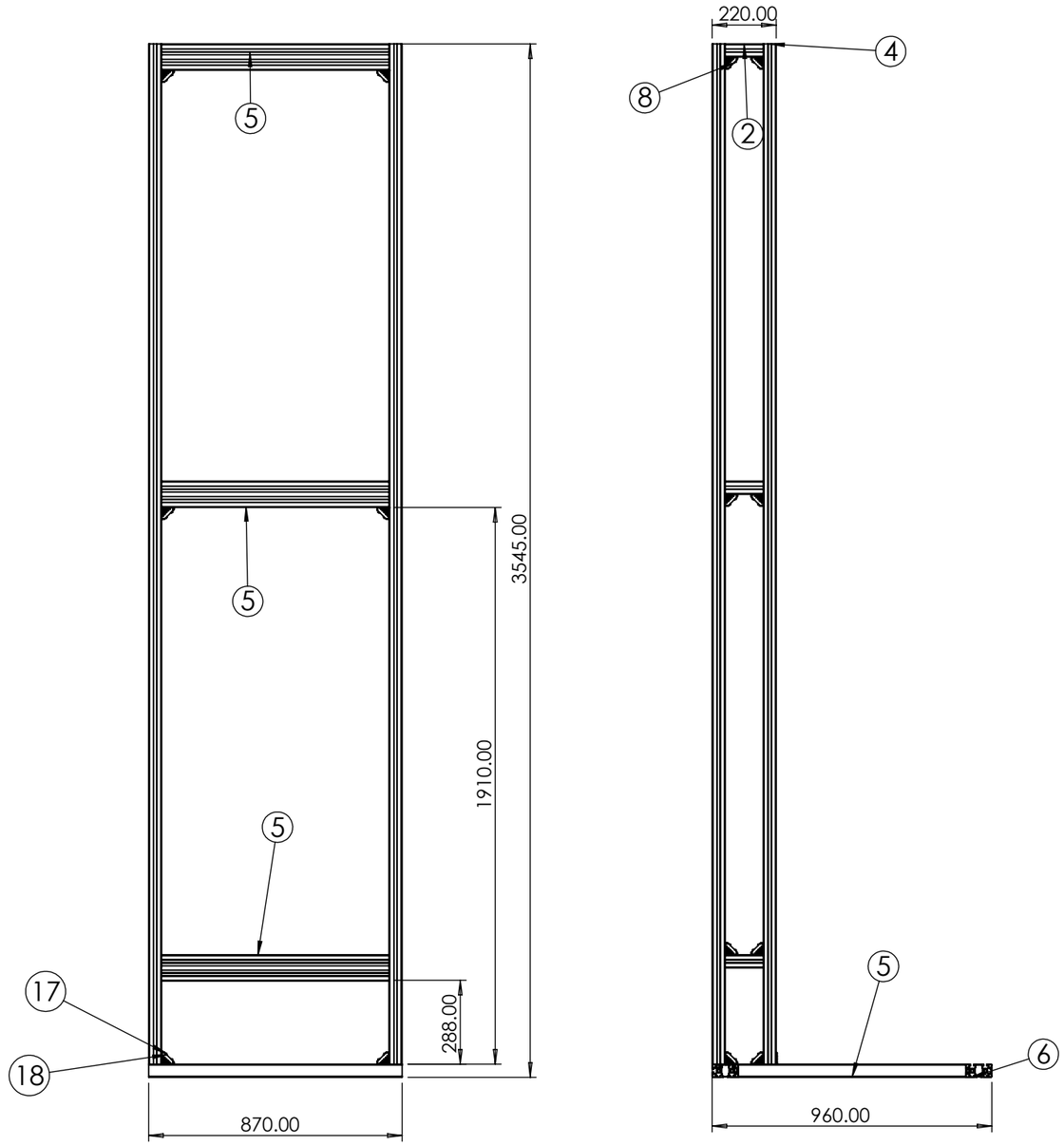
Format: A3

Listova: 1

Crtež broj: 2023-A-009

List: 1

Design by CADLab



Poz.	Naziv dijela	Kom.	Crtež broj Norma	Materijal	Sirove dimenzije Proizvođač	Masa
18	M8 MATICA	48	ISO 4032	5	Bosch-Rexroth	0.01 kg
17	T-VIJAK 8, M8X25	48	DIN 7992	6.8	Bosch-Rexroth	0.01 kg
8	45X45 KUTNIK	24	3842523558	ENAW–Al MgSi	Bosch-Rexroth	0.07 kg
6	45x45x870 ALU PROFIL	2	2023-006	ENAW–Al MgSi	Bosch-Rexroth	2.73 kg
5	45x45x780 ALU PROFIL	8	2023-005	ENAW–Al MgSi	Bosch-Rexroth	2.45 kg
4	45x45x3500 ALU PROFIL	4	2023-004	ENAW–Al MgSi	Bosch-Rexroth	6.99 kg
2	45x45x130 ALU PROFIL	6	2023-002	ENAW–Al MgSi	Bosch-Rexroth	0.26 kg

Broj naziva - code

	Datum	Ime i prezime	Potpis
Projektirao	06.05.2023	Mislav Požgaj	
Razradio	06.05.2023	Mislav Požgaj	
Crtao	06.05.2023	Mislav Požgaj	
Pregledao			
Mentor		Tomislav Staroveški	

FSB Zagreb

ISO - tolerancije

Objekt:

Objekt broj:

R. N. broj:

Napomena:

Kopija

Materijal: ENAW–Al MgSi

Masa: 57.54kg

Mjerilo originala

1:20

Naziv: Sklop glavnog nosača

Pozicija: 13

Format: A3

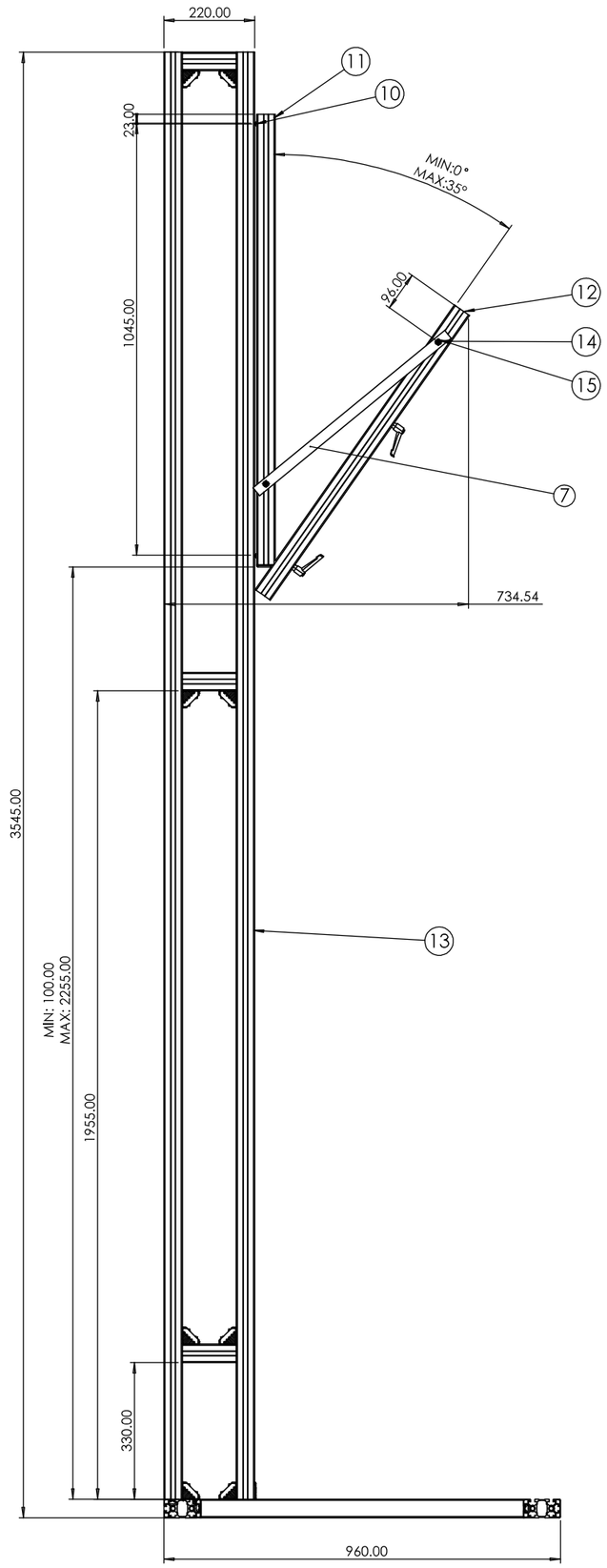
Listova: 1

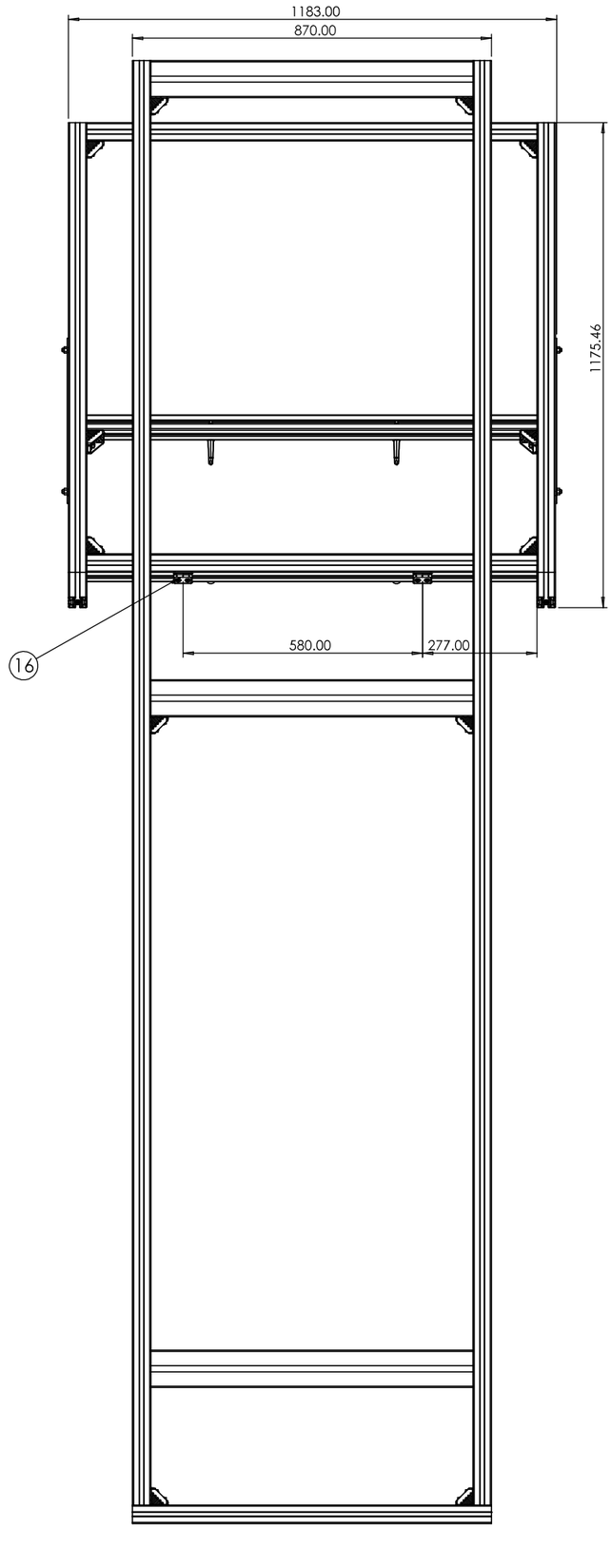
Crtež broj: 2023-A-0010

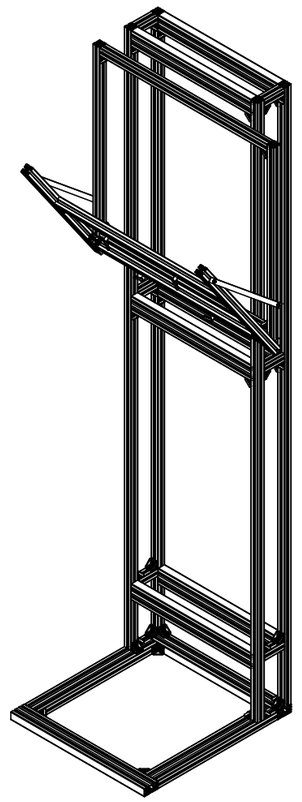
List: 1

Design by CADLab

0 10 20 30 40 50 60 70 80 90 100







MJERILO:
1:20

Poz.	Naziv dijela	Kom.	Crtež broj / Norma	Materijal	Sirove dimenzije / Proizvođač	Masa
16	ŠARKA ZA OSIGURAVANJE NAGIBA	2	AL 45/45	ENAW–Al MgSi	Bosch-Rexroth	0.1 kg
15	MATICA M8	4	ISO 4032	5	ISO 4032	0.02 kg
14	T-BOLT M8X20	4	3842528715	6.8	Bosch-Rexroth	0.01 kg
13	SKLOP GLAVNOG NOSAČA	1	2023-010	ENAW–Al MgSi	3545X870X960	57.54kg
12	SKLOP NOSAČA KALIBRACIJSKIH PANELA	1	2023-009	ENAW–Al MgSi	1180x840x45	8.23 kg
11	SKLOP KLIZAČA NOSAČA	1	2023-008	ENAW–Al MgSi	1180X1090X45	9.09 kg
10	PARALELNI SPOJNIK PROFILA	4	3842542737	ENAW–Al MgSi	Bosch-Rexroth	0.02 kg
7	PLOČA ZA OSIGURAVANJE NAGIBA	2	2023-007	ENAW–Al MgSi	600X25X3	0.12 kg

Broj naziva - code		Datum	Ime i prezime	Potpis
	Projektirao	06.05.2023	Mislav Požgaj	
	Razradio	06.05.2023	Mislav Požgaj	
	Crtao	06.05.2023	Mislav Požgaj	
	Pregledao			
	Mentor		Tomislav Staroveški	

FSB Zagreb

ISO - tolerancije	Objekt:	Objekt broj:
		R. N. broj:
	Napomena:	Kopija
	Materijal:	Masa: 75.15kg
Mjerilo originala 1:20	Naziv: NOSAČ KALIBRACIJSKIH PANELA	Pozicija:
	Crtež broj: 2023-A-011	Format: A3
		Listova: 1
		List: 1