Sustav za automatsko zavarivanje matica u sklopu postojećeg postava

Jurendić, Domagoj

Master's thesis / Diplomski rad

2022

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

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:235:954502

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Domagoj Jurendić

Zagreb, 2022.g.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentori: Student:

doc. dr. sc. Tomislav Stipančić Domagoj Jurendić

Zagreb, 2022.g.

Zagreb, 2022.g.Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu.

Zahvaljujem mentoru doc. dr. sc. Tomislavu Stipančiću na pomoći i savjetima pruženim u toku izrade ovoga rada.

Zahvaljujem se i cijeloj svojoj obitelji, koja mi je kroz godine studiranja bila podrška i oslonac u radu.

Također zahvaljujem se i kolegama i prijateljima koji su uvijek bili motivacija i podrška u zajedničkom studiju.

Domagoj Jurendić

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

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

DIPLOMSKI ZADATAK

Student: **DOMAGOJ JURENDIĆ** Mat. br.: 0035209079

Naslov rada na hrvatskom jeziku: **Sustav za automatsko zavarivanje matica u sklopu postojećeg postava**

Naslov rada na engleskom jeziku: **Automatic nut welding system within the existing installation**

Opis zadatka:

Proizvodni program tvrtke KonigMetall sadrži stotine proizvoda, koji se uglavnom izrađuju od hladno oblikovanih prešanih proizvoda od lima, a koji su namijenjeni automobilskoj/kamionskoj industriji.
U sklopu proizvodnog pogona nalazi se odjel zavarivanja gdje se na već gotove otpreske zavaruju matice na mjestima gdje je potrebno postupkom elektrootpornog projekcijskog varenja, što za radnika predstavlja monoton i po zdravlje škodljiv posao (iskre uslijed varenja, dimni plinovi i sl.).
U ovom radu potrebno je predložiti idejno rješenje za automatizaciju proizvodnog procesa zavarivanja matica pazeći pritom na prilagodljivost i kompatibilnost rješenja konceptualnom rješenju s obzirom na opširan proizvodni program gdje ponuđeno rješenje treba biti kompatibilno s trenutnim proizvodnim sustavom.

Rad treba sadržavati:

- pregled i analizu trenutnog proizvodnog procesa na odjelu zavarivanja, kako bi se uočile mogućnosti automatizacije procesa,
- simulacije, programe, nacrte i modele koji će vizualizirati automatizacijsko rješenje,
- korisničko (HMI) sučelje za upravljanje stanicom za zavarivanje matica,
- idejno rješenje vizijskog sustava za kontrolu gotovih proizvoda nakon postupka zavarivanja.

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

Zadatak zadan:
5. svibnja 2022.

Rok predaje rada:
7. srpnja 2022.

Predviđeni datum obrane:
18. srpnja do 22. srpnja 2022.

Zadatak zadao:
doc. dr. sc. Tomislav Stipančić

Predsjednica Povjerenstva:
prof. dr. sc. Biserka Runje

SADRŽAJ

POPIS SLIKA

POPIS TABLICA

POPIS OZNAKA

Oznaka	Mjerna jedinica	Opis oznake
M	Kg	- masa
g	m/s^2	- akceleracija sile teže
a	m/s^2	- akceleracija
S	-	- faktor sigurnosti
F_G	N	- sila stezanja prihvatnice
μ	-	- koeficijent trenja
m_d	kg	- maksimalna masa dijela/platine
m_p	kg	- ukupna masa prihvatnice
α	-	- parametri krivulje

SAŽETAK

U ovom diplomskom radu obrađena je konceptualna ideja i razrada automatizacije procesa varenja matica na prethodno hladno prešane proizvode, u cilju unaprjeđenja i modernizacije proizvodnog procesa tvrtke KM-Kovnica d.o.o. KM-Kovnica slovi za najveću hrvatsku tvornicu za proizvodnju dijelova od metala za autoindustriju i druge visokotehnološke branše. Tvrtka koja je osnovana 2013.g. i dio je KÖNIG METALL grupacije koja iza sebe ima više desetljetno iskustvo u proizvodnji dijelova od metala i obradi metala.

Svrha ovog rada jest predstaviti rješenje za robotizaciju i modernizaciju proizvodnog procesa u odjeli varenja, točnije jednom njegovom segmentu koji se bavi varenjem matica postupkom elektrootpornog projekcijskog varenja, a koje se u trenutačnom postavu obavlja u potpunosti ručno na sveukupno desetak strojeva s tom namjenom. Osnovni cilj ove automatizacije je postizanje visoke razine kvalitete završnog proizvoda, te postizanje ujednačene kvalitete, zatim modernizacija i smanjenje broja potrebnih radnika na fizički i psihički zahtjevnim radnim mjestima.

Za modeliranje potrebnih dijelova i elemenata robotske stanice poslužio je softverski paket za izradu tehničke dokumentacije i 3D modeliranje SolidWorks 2017. Za simuliranje postrojenja i analizu ciklusa korišten je softverski paket RoboDK. Stanicom, njenim elementima kao i samim robotom upravlja PLC u kombinaciji s HMI korisničkim sučeljem koje je programirano unutar Siemens-ovog softverskog paketa TIA Portal v15.1, dok je komunikacija između PLC-a i same robotske stanice, obzirom da se radi o simulacijskom okuženju, izvedena i isprogramirana unutar Python-a.

U sklopu same robotske stanice integrirana je stanica za vizijsku kontrolu zavarenosti matica, gdje se prethodno zavaren komad snima kamerom, te se provjerava ispravnost odrađenog postupka. Navedena vizijska kontrola također je izvedena Python programskim jezikom.

Ključne riječi: robotizacija, automatizacija, varenje matica

SUMMARY

In this diploma paper we deal with the conceptual idea and elaboration of automation of the process of welding nuts on previously cold-pressed products, in order to improve and modernize the production process of the company KM-Kovnica d.o.o. KM-Kovnica is known as the largest Croatian factory for the production of metal parts for the automotive industry and other high-tech industries. The company, which was founded in 2013. and is a part of the

KÖNIG METALL Group, which has decades of experience in the production of metal parts and metal processing.

The purpose of this paper is to present a solution for robotization and modernization of the production process in the welding department, more precisely one of its segments dealing with welding of nuts by electric resistance projection welding, which is currently performed entirely by hand on a total of ten machines. The main goal of this automation is to achieve a high level of quality of the final product, aswell as to achieve uniform quality, then modernization and reduction of the number of workers needed in this physically and mentally demanding jobs.

The software package for the preperation of technical documentation and 3D modeling SolidWorks 2017 was used to model the necessary parts and elements of the robotic station. The software package RoboDK was used to simulate the plant and analyze the cycle. The station, its elements and the robot itself are controlled by a PLC in combination with a HMI user interface programmed within Siemens' Tia Portal v15.1 software package, while communication between the PLC and the robotic station itself, since it is a simulation environment, is executed and programmed within Python.

As part of the robotic station itself, there is an integrated station for visual inspection of the welded parts, where the previously welded piece is recorded with a camera, and the correctness of the performed procedure is checked. The mentioned vision inspection was also performed in the Python programming language.

Key words: robotization, automatization, nut welding

1. UVOD

U sklopu ovog diplomskog rada obrađena je ideja automatiziranja procesa elektrootpornog projekcijskog varenja matica na hladno prešane dijelove. Dijelovi se najprije iz lima prešaju na hidrauličkim ili ekscentar prešama gdje radnik u prešu stavlja materijal(komad lima) te se ovisno o vrsti proizvoda u jednom ili više koraka dobiva završni proizvod prešanja. Po završetku procesa prešanja komada, isti se šalju na odjel varenja gdje se prema specifikaciji proizvoda vari određeni tip matica na za to predviđena mjesta.

Tvrtka KM-Kovnica d.o.o u svom proizvodnom programu ima više stotina različitih dijelova za proizvodnju većina od čega je namijenjena za automobilsku i kamionsku industriju. Tvrtka je 2013.godine ustanovljena u industrijskoj zoni Pisarovina, malom mjestu oko 30 kilometara južno od Zagreba. S obzirom na širok proizvodni program tvrtka mora imati fleksibilnu i dinamičnu strukturu proizvodnje koja joj omogućuje mjesečnu proizvodnju oko 630 različitih tipova proizvoda i mjesečni output od oko 16 000 tona gotovih proizvoda od metala.

Trenutačni proizvodni procesi u tvrtki odvijaju se u većem dijelu manualno/ručno, gdje jedan ili više radnika opslužuje jedan stroj bilo to preša od 630 tona pritiska ili stroj za elektrootporno projekcijsko varenje. Tvrtka već nekoliko godina ulaže veliki napor i novac u strojeve i automatizaciju, tako su instalirane dvije robotske stanice za MIG varenje, dvije automatske ekscentar preše od 400 tona pritiska i slično.

Uz sve navedeno u okviru ovog rada dakle potrebno je dati rješenje koje će odgovarati zahtjevima ovakvog proizvodnog pogona, rješenje mora biti fleksibilno i otvoreno za implementaciju proizvodnje čim više različitih proizvoda. U radu će se prikazati ideja i simulacija predloženog rješenja, u čiju svrhu su korišteni programski paketi SolidWorks 2017, RoboDK, TIA Portal v15.1 te Python.

1.1.KÖNIG METALL CROATIA

König Metall Hrvatska sinonim je za dvije hrvatske sestrinske tvrtke KM-Kovnica te KM-Alati, koje zajedno djeluju u okviru König Metall grupe koje se više od stotinu godina bavi obradom metalnih limova, cijevi i tubi prema zahtjevima kupaca i to na ukupno osam lokacija diljem svijeta, a sve u smjeru metalurške, elektroničke, automobilske te industrije prigušivača i ispušnih sustava.

Ono po čemu se grupacija kao cjelina ističe jest njena fleksibilnost, koja omogućuje veoma traženu i zahtjevnu 'just-in-time' proizvodnju, na kojoj se danas zasnivaju gotovo sve velike industrije, poglavito ona automobilska. Upravo ovaj tip proizvodnje u današnje vrijeme zahtjeva od konkurentnih poduzeća visoku razinu fleksibilnosti kako bi bili u mogućnosti u pravo vrijeme isporučiti zahtijevane količine proizvoda, što predstavlja veliki problem već u samoj organizaciji proizvodnje na tradicionalni način, a također predstavlja i veliki problem u automatizaciji iste.

U okviru König Metall Hrvatska dakle nalaze se dvije sestrinske tvrtke KM-Kovnica, i KM-Alati. Tvrtka KM-Kovnica bavi se primarno tlačnom obradom limova od metala, u vidu čega se koristi strojni park koji se sastoji od:

1. 18 hidrauličkih preša
2. 4 ekscentrične preše
3. 3 automatske ekscentrične preše

Slika 1. Hidrauličke preše[1]

snaga između 63 i 630 tona. Osim toga uz postupke prešanja s do 630 tona sile, tvrtka se bavi i tehnikama zavarivanja s raznovrsnim strojevima(elektrootporna zavarivanja, točkasta

zavarivanja, MIG varenja – ručna i robotizirana), zatim i postupcima savijanja na strojevima za savijanje limova te bušenja.

Slika 2. Dio izložbenog asortimana tvrtke KM-Kovnica

Tradicionalni načini proizvodnje kod kojih se šarže proizvoda prema narudžbi klijenta u manjim serijama i količinama proizvode tako da radnici na predviđenim radnim mjestima s točno propisanim koracima i operacijama vrše ručno svaki propisani korak u današnjem svijetu moderne proizvodnje postaju zastarjeli i neefikasni načini proizvodnje. Međutim u određenim primjerima takav tip proizvodnje i dalje je poželjan i primjenjiv s obzirom na mogućnost brze prilagodbe radnih operacija, brze izmjene radnih stanica i proizvodnih postupaka. No činjenica jest da danas, takav način proizvodnje živi još jedino u društvima koja su u mogućnosti pružiti jeftinu radnu snagu, što je svjedoci smo u našem okruženju sve teže isporučiti s obzirom na rast cijena rada i života.

Proizvodnja dijelova na ovakav tradicionalni način – ručno, obavljajući korak po korak naravno za radnike predstavlja monoton i zamarajući rad, no uz pomoć automatizacije i

modernizacije proizvodnje takva proizvodnja postaje dakako konkurentnija, brža i efikasnija. Problemi se javljaju prilikom (pokušaja) primjene automatizacijskih rješenja koja bi se u obliku masovne proizvodnje dijelova pokazala adekvatnim, na tip proizvodnje koji se odlikuje manjim šaržama proizvoda, uz veliki broj proizvoda u proizvodnom asortimanu.

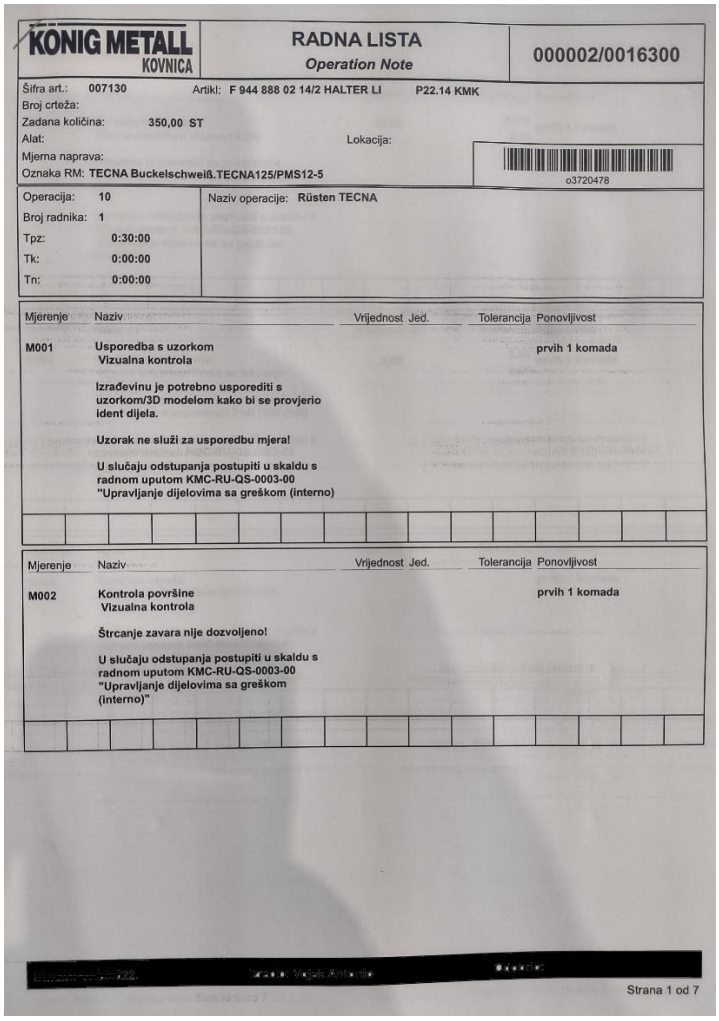
Robotski i automatizacijski sustavi često su percipirani kao nefleksibilna rješenja, teška za prilagodbu drugačijim proizvodnim procesima, pa se obično izbjegava pokrenuti prvi korak u automatizaciji procesa proizvodnje, a to jest nacrt i izrada idejnog rješenja automatizacije. Očigledno je da zbog načina proizvodnje s kojim se ovdje susrećemo bilo kakvo rješenje u sebi mora ukomponirati mogućnost brze, efikasne i jednostavne promjene procesa proizvodnje.

Ovaj rad dati će prijedlog izvedbe automatske robotske stanice za posluživanje stroja za varenje matica, uz pažnju na čim jednostavniju izmjenu proizvoda na stroju, uz minimalne operacije prilagodbe stanice.

2. IZGLED POSTOJEĆE RADNE STANICE

Na odjelu varenja, točnije na dijelu koji se bavi isključivo varenjem matica nalazi se 10-ak strojeva za elektrootporno – bradavičasto zavarivanje matica, na kojima radi po jedan radnik, u tri smjene. U cilju davanja kvalitetnog rješenja automatizacije ovog procesa najprije ćemo razmotriti način i slijed operacija kojima se trenutno izvršava posao. Ovisno o tipu proizvoda koji se vari, svaki radnik prije početka rada na stroju dobiva radni nalog, na kojem se nalaze podatci o samom proizvodu koji će se raditi, naziv radne operacije koje obavlja, te također druge informacije vezane uz potreban materijal, postupke koje je potrebno provoditi tijekom rada, poput vizualnih kontrola proizvoda. Također dobiva i informacije o vrsti pakiranja koja je potrebna za završene proizvode.

Dok što se tiče samog načina kako i što variti, radnik s obzirom na proizvod ima na raspolaganju kratke upute sa slikovnim prikazom kako variti, koje pozicije treba variti, kako držati komad u procesu varenja i slično.



KONIG METALL KOVNICA

RADNA LISTA
Operation Note

000002/0016300

Šifra art.: 007130 Artikl: F 944 888 02 14/2 HALTER LI P22.14 KMK
Broj crteža:
Zadana količina: 350,00 ST
Alat: Lokacija:
Mjerna naprava:
Oznaka RM: TECNA Buckelschweiß.TECNA125/PMS12-5

o3720478

Operacija: 10
Broj radnika: 1
Tpz: 0:30:00
Tk: 0:00:00
Tn: 0:00:00

Naziv operacije: Rüsten TECNA

Mjerenje	Naziv	Vrijednost	Jed.	Tolerancija	Ponovljivost
M001	Usporedba s uzorkom Vizualna kontrola Izrađevinu je potrebno usporediti s uzorkom/3D modelom kako bi se provjerio ident dijela. Uzorak ne služi za usporedbu mjera! U slučaju odstupanja postupiti u skaldu s radnom uputom KMC-RU-QS-0003-00 "Upravljanje dijelovima sa greškom (interno)				prvih 1 komada

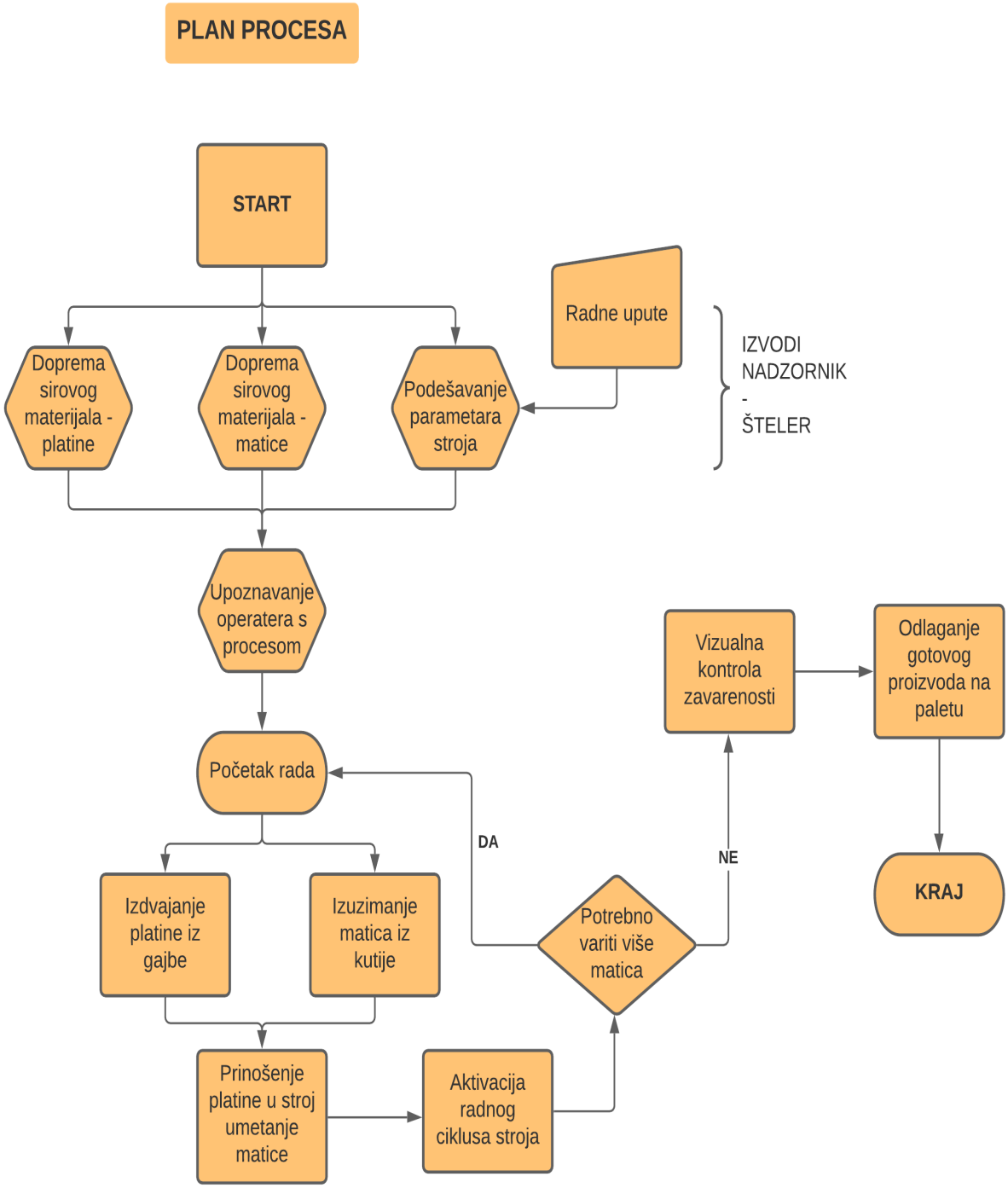
Mjerenje	Naziv	Vrijednost	Jed.	Tolerancija	Ponovljivost
M002	Kontrola površine Vizualna kontrola Štrcanje zavara nije dozvoljeno! U slučaju odstupanja postupiti u skaldu s radnom uputom KMC-RU-QS-0003-00 "Upravljanje dijelovima sa greškom (interno)"				prvih 1 komada

[redacted]

Strana 1 od 7

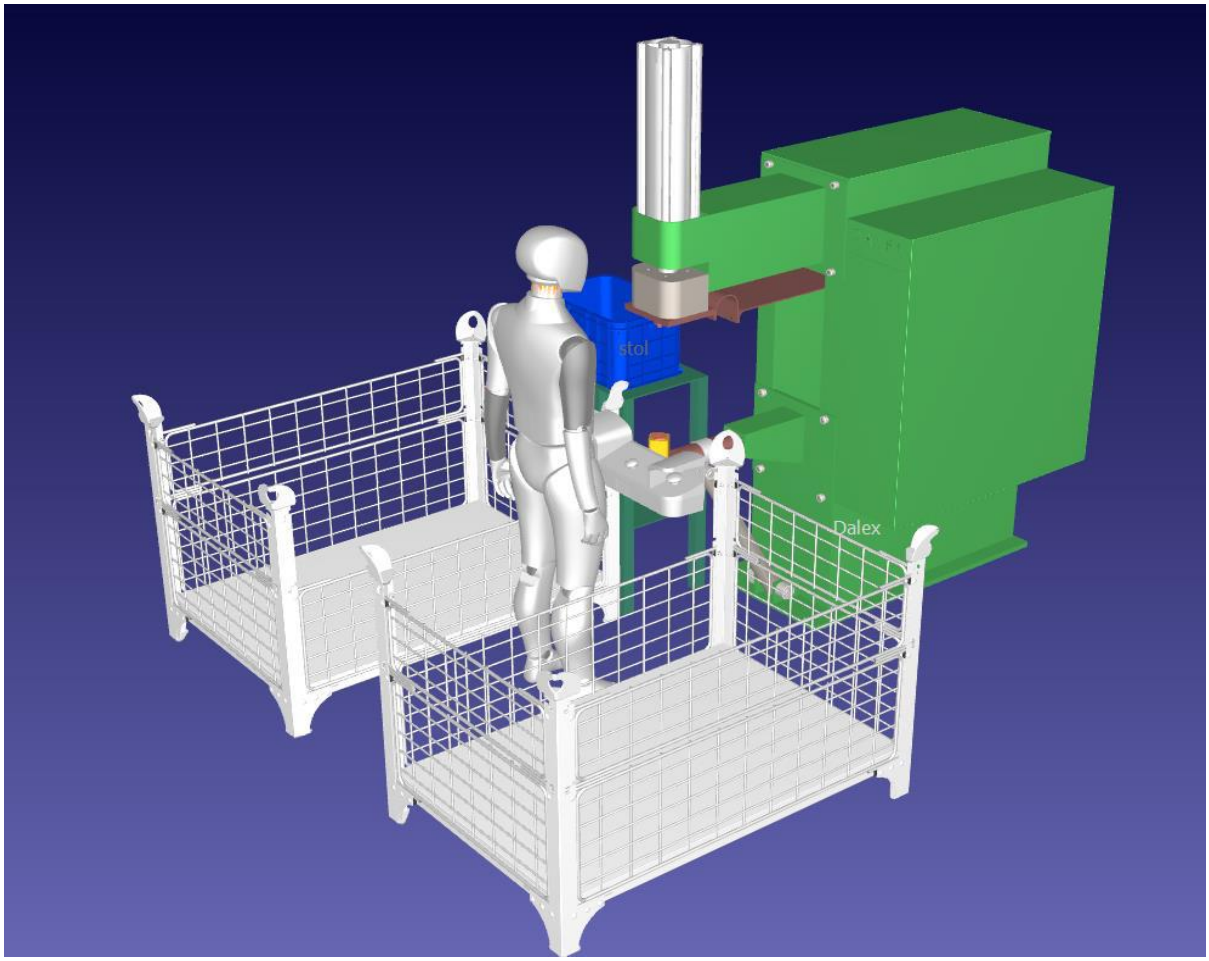
Slika 3. Primjer radnog naloga/liste

Osim radnika na strojevima za varenje, kao što je slučaj i u ostatku tvrtke bez obzira na odjel, postoje predradnici/nadzornici(tzv. Šteleri), koji vode timove od po nekoliko radnika/strojeva), koji su zaduženi za uštelavanje i pripremu stroja za rad, dobavu i pripremu materijala koji se obično nalazi u gajbama, i naravno uvođenje i pojašnjavanje radnog procesa radniku. Na slici ispod možemo ukratko vidjeti

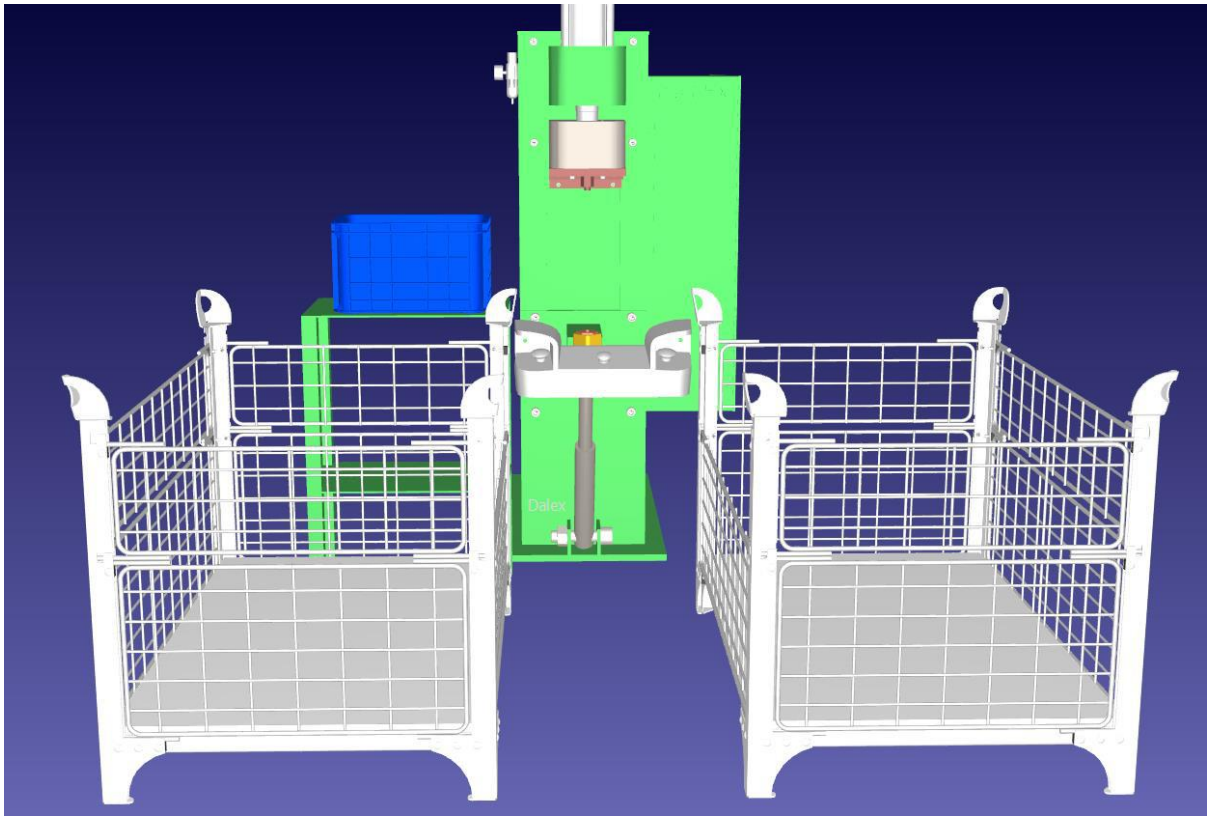


Slika 4. Plan procesa

U nastavku možemo vidjeti 3D model trenutne radne stanice za varenje matica, kao što se može vidjeti s lijeve i desne strane radnika nalaze se gajbe u kojima se nalazi materijal ili se u nju odlažu gotovi dijelovi, dok u kutijici pokraj samog stroja radnik ima matice potrebne za varenje.



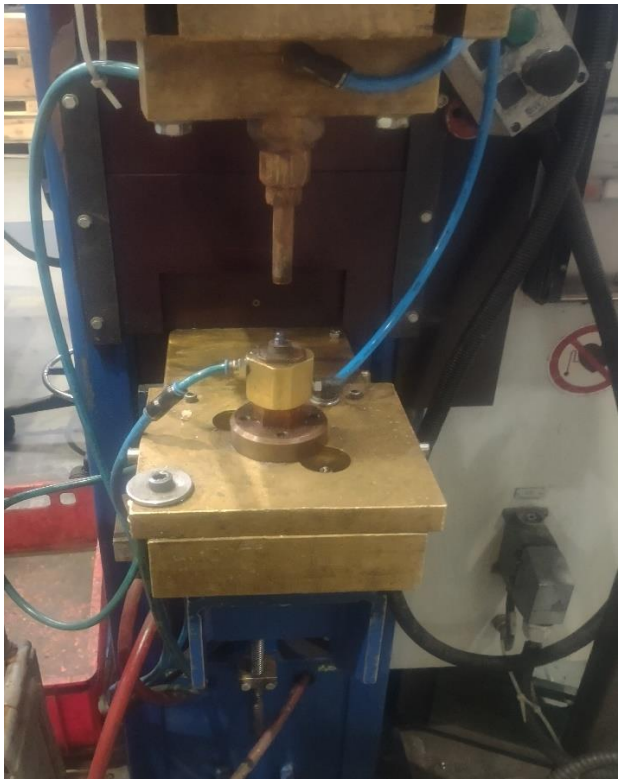
Slika 5. Radnik u radnoj stanici



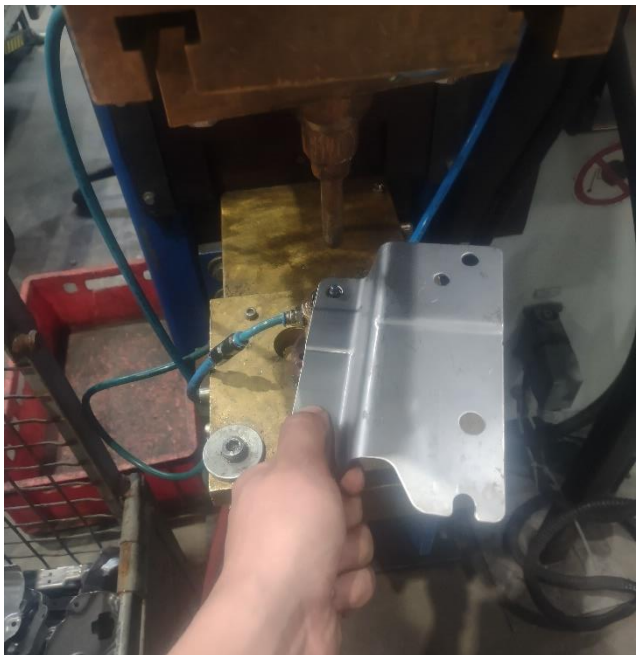
Slika 6. Radna stanica

Na gornjoj slici vidimo trenutni izgled radnog mjesta. Radnik iz gajbe s njegove lijeve strane vadi platine, u plavoj kutiji na radnom stolu nalaze se matice za varenje, radnik prinese platinu na donju elektrodu stroja za varenje, drugom rukom na vrh elektrode postavlja maticu, te potom pritiskom na gumb aktivira radni ciklus stroja. Gornja elektroda stroja se spusti i stisne maticu i platinu, potom se aktivira struja varenja koje je podešena parametrima samog stroja. Po završetku varenja elektroda se diže u početnu poziciju, operater vari sljedeću maticu ako je potrebna, dok gotove komade odlaže u gajbu desno, te se ciklus ponavlja.

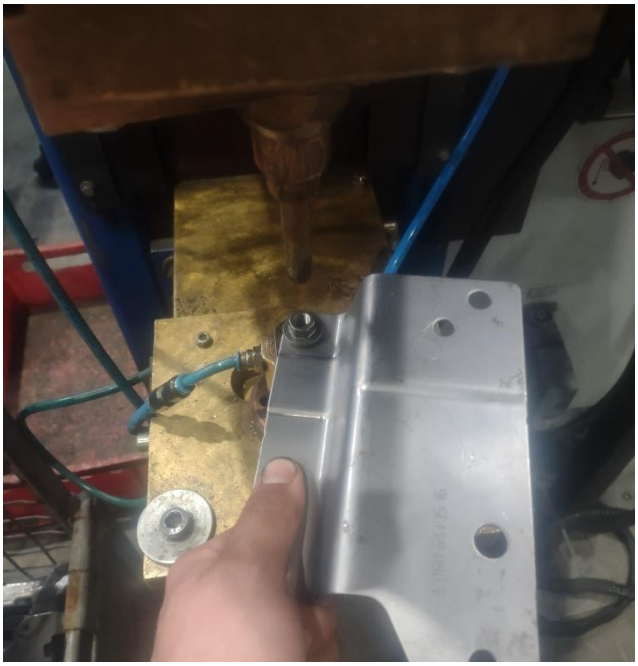
U nastavku možemo vidjeti primjer procesa varenja jedne matice na za to predviđeno mjesto.



Slika 7. Izgled elektrode za varenje



Slika 8. Postavljanje platine na elektrodu



Slika 9. Postavljanje matice na platinu

Slika 10. Aktivacija radnog ciklusa stroja za varenje

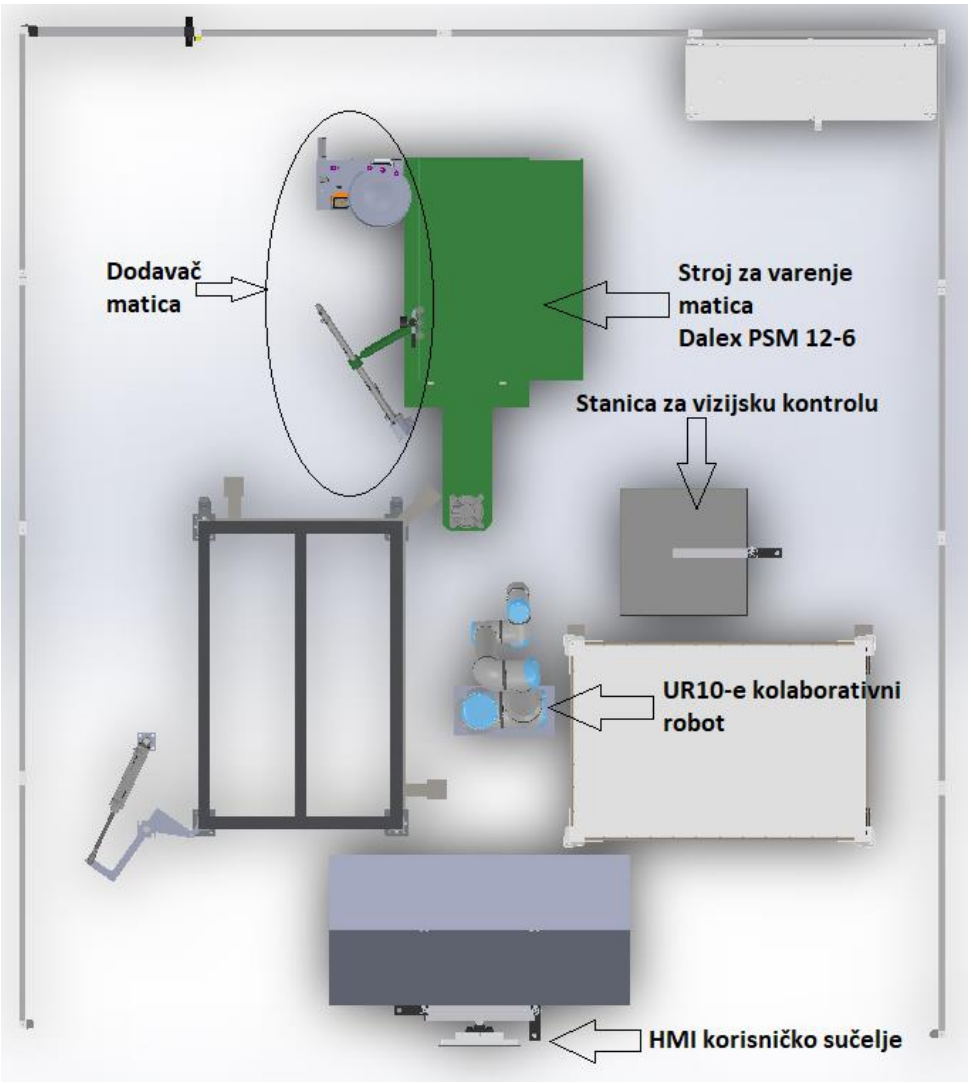
Na slici ispod možemo vidjeti izgled matice za elektrootporno bradavičasto varenje, matica sa svoje donje strane ima 3 bradavice raspoređene po obodu same matice, koje se pri puštenoj struji varenja rastale, te se utope u osnovni materijal, te je time varenje dovršeno.

Slika 11. Matica za bradavičasto varenje

Ovaj postupak pokušat ćemo čim bolje automatizirati, te ćemo pokušati dati rješenje za čitav ciklus koji obavlja radnik, vidljiv na slici 4.

3. STANICA ZA AUTOMATSKO ZAVARIVANJE MATICA

Pri izradi ovog idejnog rješenja robotske stanice za automatsko varenje matica, u vidu treba imati čim lakšu i jednostavniju implementaciju u već postojećem pogonu. Jedan od ciljeva jest i unutar nove automatske stanice iskoristiti čim više opreme koja se već u pogonu koristi, kako bi samu stanicu bilo čim lakše adaptirati različitim potrebama pogona. Također u svrhu uštede treba paziti na čim manju potrebu za izradom specijalnih dijelova ili narudžba i kupovina istih. Također, obzirom da je sam pogon poprilično prostorno popunjen, te su trenutne radne stanice poprilično kompaktne i male, treba paziti da novo rješenje zauzima čim manje prostora, kako bi se što više očuvao trenutni izgled i raspored pogona, kako bi se izbjegla nepotrebna seljenja i preraspodjele unutar pogona.



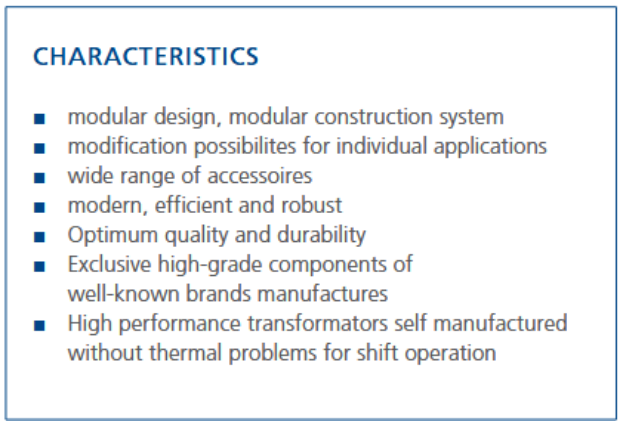
Slika 12. Gornji pogled stanice za automatsko varenje matica

U sljedećim pod poglavljima prikazana je korištena opremu, koja se nalazi unutar stanice za automatsko varenje matica.

3.1. Stroj za varenje Dalex PMS 12-6

Dalex je tvrtka koje se bavi tehnologijama otporničkog varenja, tvrtka je stacionirana u mjestu Wissen, Njemačka, u svom portfelju nude dugoročna i efikasna rješenja za točkasta varenja, šavna varenja, šavna varenja folija, potom varenja mreža, MAG i laserska varenja, te nama najinteresantnije bradavičasto varenje.

Modeli serije PMS mogu se primjenjivati u širokom spektru zadataka, obzirom na svoju prilagodljivost i pouzdanost. Ono što krasi ovaj model strojeva jest njihova modularnost koja nam omogućuje lakšu primjenu u specifičnim zadatcima.



Slika 13. Karakteristike PMS modela[3]

3.1.1. Metode varenja

PMS modeli omogućuju nam tri tipa varenja koja se odabiru jednostavnom izmjenom alata/elektroda na stroju, te naravno izmjenom parametara stroja.

Svako od ovih tipova varenja specifično je po svojoj izvedbi i namjeni, a to su:

1. Točkasto varenje

Dijelovi su pri varenju pritisnuti jedan uz drugoga, potom na mjestima gdje se dijelovi spajaju dolazi do zagrijavanja materijala koji se počinje taliti uslijed struje varenja. Ovisno o podešenim parametrima stroja struja varenja teče određeni broj perioda, te se potom gasi. Materijal se hladi i dolazi do spoja materijala

Slika 14. Točkasto varenje [3]

2. Bradavičasto varenje

   Radi se o podvrsti točkastoga varenja, kod kojeg se može raditi istodobno ili o jedno ili o više točkastom varenju ovisno o broju izbočina/bradavica koje se nalaze na spoju dijelova, kroz koje se pušta struja varenja. Po puštanju struje varenja, bradavice se tale i 'utapaju' u osnovni materijal te tako dolazi do spoja.

Slika 15. Bradavičasto varenje [3]

3. Šavno varenje

   Pri šavnom varenju potrebno je na stroju za varenje postaviti elektrode u obliku diska, koje varene dijelove pritišću s obiju strana, te se istovremeno pomiču. Preko samih diskova također ovisno o parametrima varenja, protječe struja varenja neprekidno, ili pak u ciklusima s prekidima. Nastali zavareni spoj tako može biti ili kontinuiran ili pak neprekidan niz preklopljenih točkastih zavara(vodonepropustan var) ili niz međusobno odmaknutih točkastih zavara(isprekidano varenje)

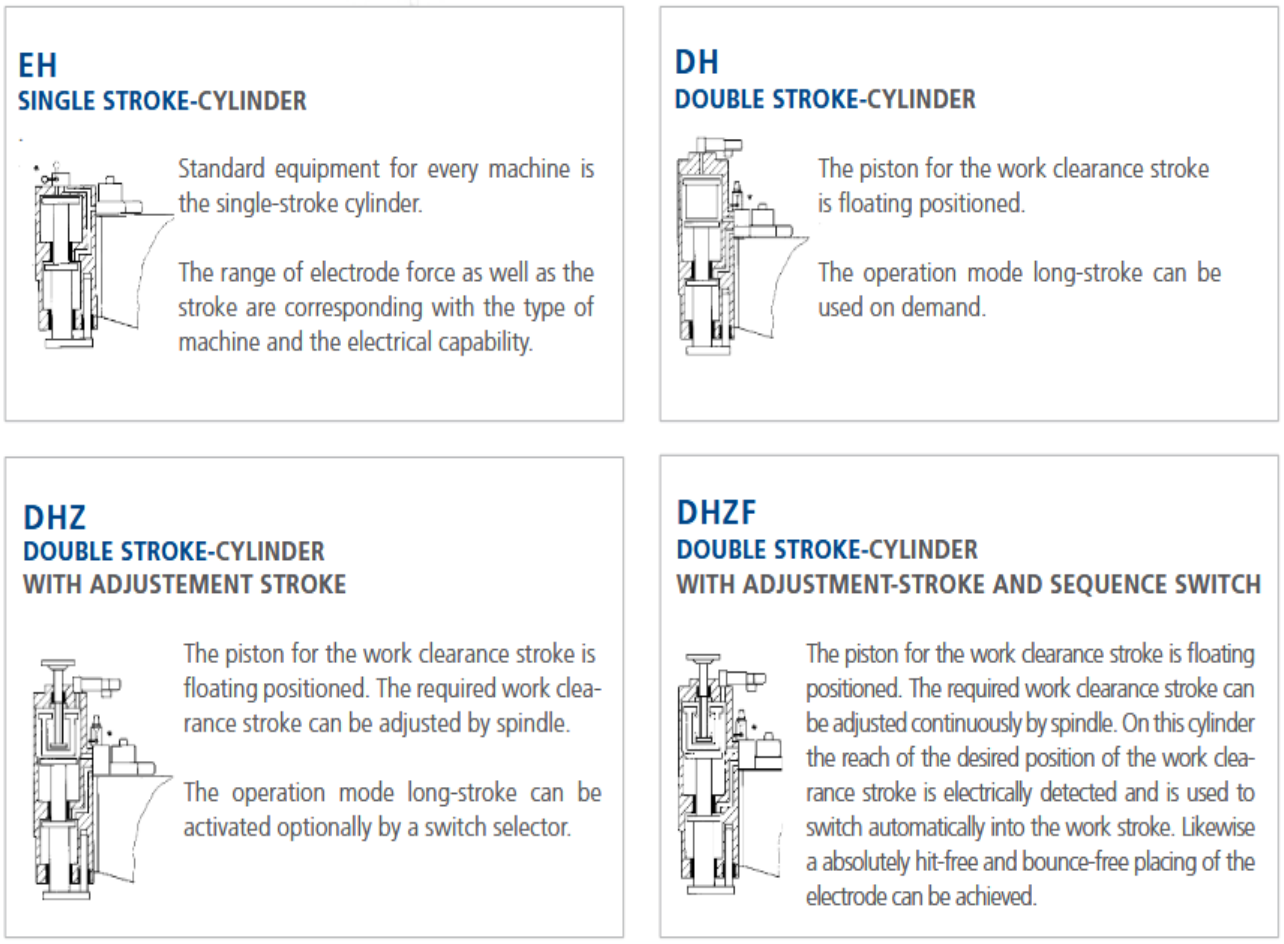
Slika 16. Šavno varenje [3]

Slika 17. Dalex PMS 12-6[3]

Pneumatski cilindar koji pritišće gornju elektrodu stroja na materijal jest tandemski pneumatski cilindar koji uz pomoć dvije zračne komore ima mogućnost ostvarivanja relativno velike sile pritiskanja, bez obzira na manji promjer klipa cilindra. U odabiru imamo različite izvedbe pneumatskih cilindara:

1. Jednoradni cilindar oznake EH
2. Dvoradni cilindar oznake DH
3. Dvoradni cilindar s podešavajućim hodom

4. Dvoradni cilindar s podešavajućim hodom i sekvencijskim prekidačem



Slika 18. Izvedbe pneumatskih cilindara za pritiskanje elektroda [3]

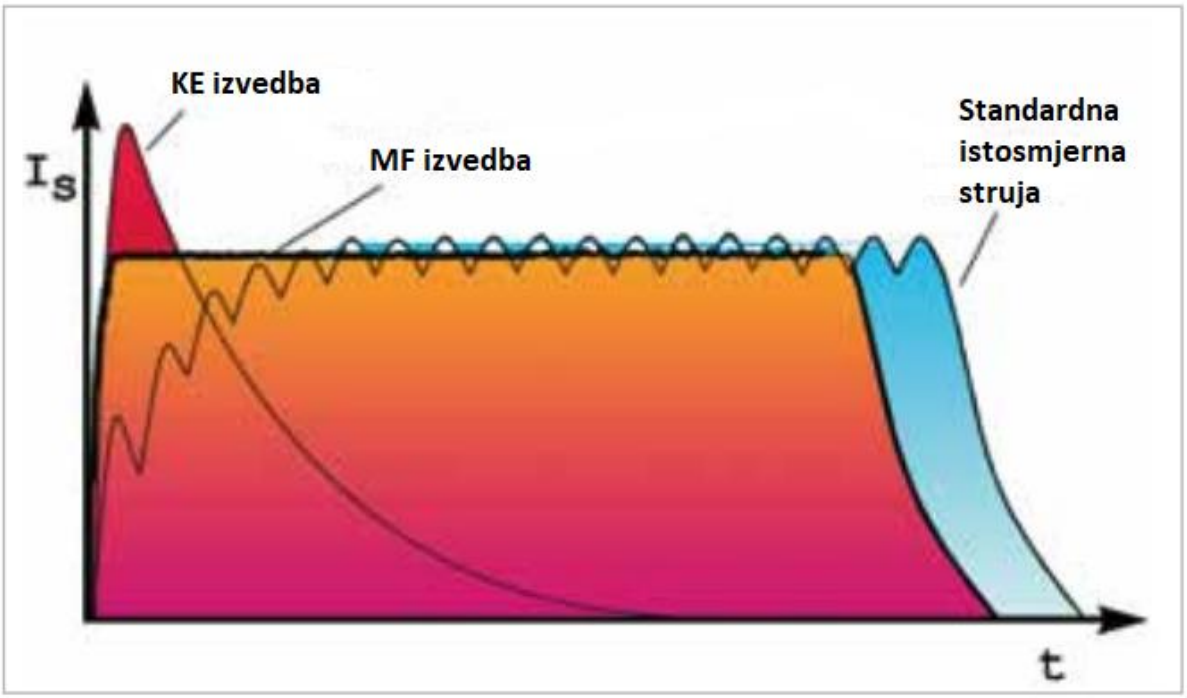
Za našu primjenu odabran je standardni jednoradni cilindar oznake EH, obzirom da nemamo potrebu za brzim i sporim hodom cilindra, dapače odgovara nam maksimalna brzina spuštanja elektrode do materijala, čime se u popriličnoj mjeri smanjuje vremenski period ciklusa varenja jedne matice.

Podjela strojeva s obzirom na način generiranja napona tj. struje varenja jest sljedeća:

1. Izvedba s trofaznim izmjeničnim 400V transformatorom koji na sekundarnom namotu generira napon od 5-9V istovremeno podižući struju varenja za isti faktor.
2. Izvedba s trofaznom istosmjernom strujom, koja koristi tri precizna adaptivna transformatora, koji na svojim izlazima imaju ispravljače koji skupa generiraju

prekidani istosmjerni napon, dok je odnos struje i napona sličan kao kod izvedbe br.1

3. MF(*eng. Medium frequency)* izvedba koja implementira frekvencijski pretvarač. Ova izvedba radi tako da frekvencijski pretvarač ulazni napon transformatora diže na oko 1000Hz, dok se na sekundaru transformatora nalazi vodom hlađen ispravljač tako da se na elektrodama dobiva istosmjerna struja varenja.
4. KE(*eng. Capacitor discharge*) izvedba koja koristi energiju iz prethodno napunjenog kondenzatora, koje prije dolaska na same elektrode stroja prolazi niz tiristora i transformatora za varenje, čija je odlika relativno visok napon sekundara u praznom hodu transformatora, te jako brz rast struje sekundara uslijed iznenadnog pražnjenja kondenzatora.



Slika 19. Usporedba krivulja struje obzirom na izvedbu[3]

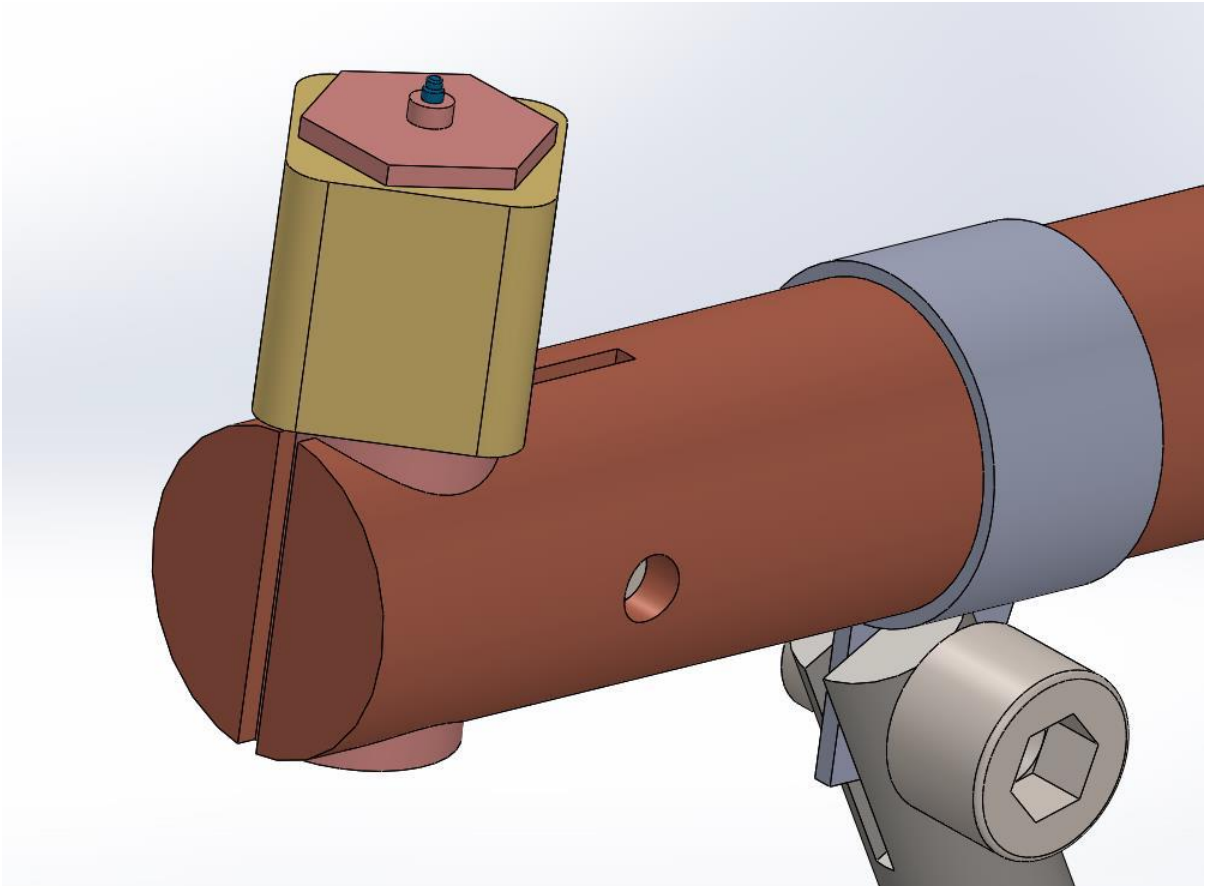
Slika ispod prikazuje vremenski ciklus unutar kojeg je snimljena struja varenja, kao i napon na transformatoru, te se jasno može vidjeti obrnuto proporcionalna veza napona i struje varenja, također možemo primijetiti kako je unutar vremenskog perioda od svega nekoliko milisekundi propuštena struja od otprilike *700A*

Slika 20. Snimljeni ciklus struje varenja [20]

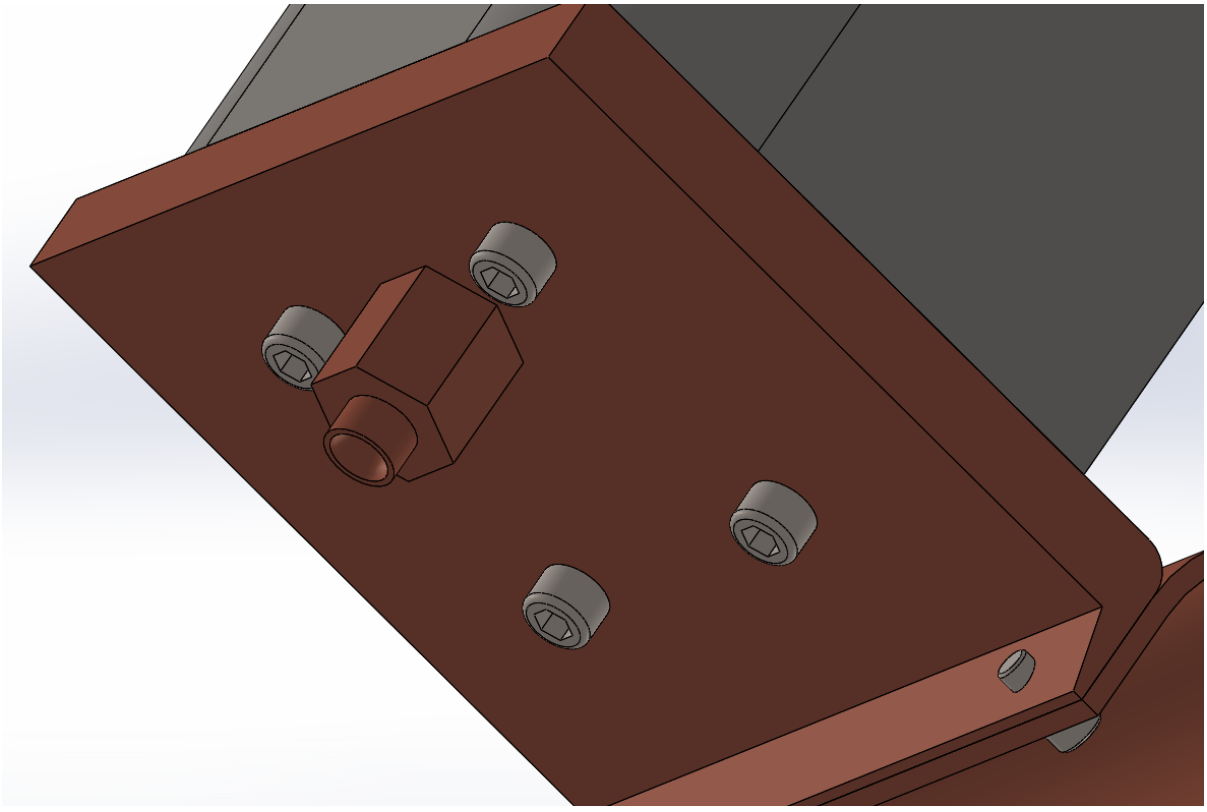
U našem slučaju odabran je stroj Dalex PMS 12-6 namijenjen točkastom i bradavičastom varenju, u MF izvedbi s frekvencijskim pretvaračem koja nam nudi preciznu i pouzdanu regulaciju struje varenja, ključno za ostvarivanje kvalitetnih i ujednačenih varova koji zadovoljavaju specifikacije kupca.

S obzirom na to da se sam ciklus varenja odvija poprilično brzo, u principu oko 2-3s, moguće je poprilično brzo ponovno pokrenuti ciklus, što uvelike ovisi o hlađenju elektroda i transformatora. Naime stroj mora biti spojen na sustav hlađenja(može biti zaseban ili centralni sustav hlađenja), ovisno o samom hlađenju stroj je u mogućnosti svakih nekoliko sekundi ulaziti u ciklus varenja, jedina granica dakle jest pregrijavanje nakon čega se stroj automatski zaustavlja.

U našem primjeru izvedbe stanice za automatsko varenje matica, varit će se kako je vidljivo i na slici *br11.* matice metričkog M8 navoja, s krunom, te se prema tome na stroj za varenje montiraju gornja i donja elektroda za tu namjenu. Na slikama ispod možemo vidjeti CAD modele gornje i donje elektrode stroja za ovakvu maticu.



Slika 21. CAD model donje elektrode



Slika 22. CAD model gornje elektrode

Na slikama ispod prikazana su i objašnjena podešavanja parametara stroja za varenje matica. S obzirom da je lista parametara poprilično opširna, prikazani su parametri od ključne važnosti koji se najčešće mijenjaju pri izmjeni tipa proizvoda koji se vari, a koji ovisi o vrsti materijala koji se vari, debljini osnovnog materijala odnosno lima, veličini i tipu matice koja se vari itd.

Slika 23. Podešavanje VHZ parametra

Parametar VHZ (*njem. Vorhaltzeit)* odnosi se na vrijeme zadržavanja, ovo vrijeme odnosi se na period premošćivanja vremena zatvaranja alata za zavarivanje(gornja i donja elektroda) i sigurnog stvaranja pritiska u donjoj poziciji cilindra (ostvarena dovoljna sila pritiska na maticu i komad). Ovaj parametar podešen je na 30 perioda.

Slika 24. Podešavanje SAZ parametra

Parametar SAZ (*njem. Anstiegzeit)* odnosi se na vrijeme koje mora proteći pri podizanju cilindra u gornju poziciju i pokretanja novog ciklusa, te je ono podešeno na 0 perioda, tako da je u teoriji moguće odmah po podizanju cilindra u gornju točku pokrenuti novi ciklus.

Slika 25. Podešavanje I parametra

Parametar I (*njem. Schweißzeit*), odnosi se na jakost struje koja vari materijal, te se podešava u promilima(od 1-999‰), te smo ga podesili na 540‰, kako bi se zadovoljili kriteriji vara (moment držanja, proboj vara i sl.).

Slika 26. Podešavanje SZ parametra

Parametar SZ (*njem. Stromzeit*) odnosi se na vrijeme unutar kojeg se pušta struja varenja kroz elektrode, te je podešen na 15 perioda.

Slika 27. Podešavanje NHZ parametra

Parametar NHZ (*njem. Nachhaltzeit*) određuje vrijeme zadržavanja potrebno da se bradavičasti var ohladi prije nego što se elektrode otvore(podizanje cilindra), te je podešen na 25 perioda.

Slika 28. Podešavanje IMP parametra

Parametra IMP (*njem. Impulse Anzahl*) određuje koliko se strujnih impulsa izvršava tijekom kontrole/testa rada, te je podešen na minimum – 1, radi skraćivanja ciklusa s obzirom da nema potrebe pri svakom varu provoditi testiranje.

Slika 29. Podešavanje PZ parametra

Parametar PZ (*njem. Pausenzeit*) odnosi se na trajanje pauze između svakog impulsa struje varenja, te je podešen na minimum odnosno 0 perioda, čime u principu ostvarujemo kontinuiranu struju varenja, bez prekida.

4. DODAVAČ MATICA

U svrhu upucavanja matica na donju elektrodu, nakon što je robot u stroj prinio komad konstruiran je uređaj koji se sastoji od vibracijske zdjele čija je primarna svrha skladištenje i razvrstavanje odnosno pravilna orijentacija matica, potom imamo linearnu vibracijsku stazu koju koristimo kako bi ubrzali hod matice od izlaza vibracijske zdjele do mjesta za upucavanje u savitljivu vinil cijev pravokutnog oblika. Cilindar za upucavanje matica puni vodilicu i vinil cijev maticama, sve dok matice ne dođu do glave samog dodavača koji se nalazi fiksno montiran na stroj za varenje. S te pozicije matice na vrh donje elektrode upucava pneumatski cilindar izrazito brzim i dugačkim hodom, dok za dodatno osiguranje od opadanja, vrh klipa cilindra za upucavanje je izrađen od magnetičnog materijala koji dodatno pridržava maticu na relaciji od glave dodavača do vrha donje elektrode.

4.1. Vibracijska zdjela

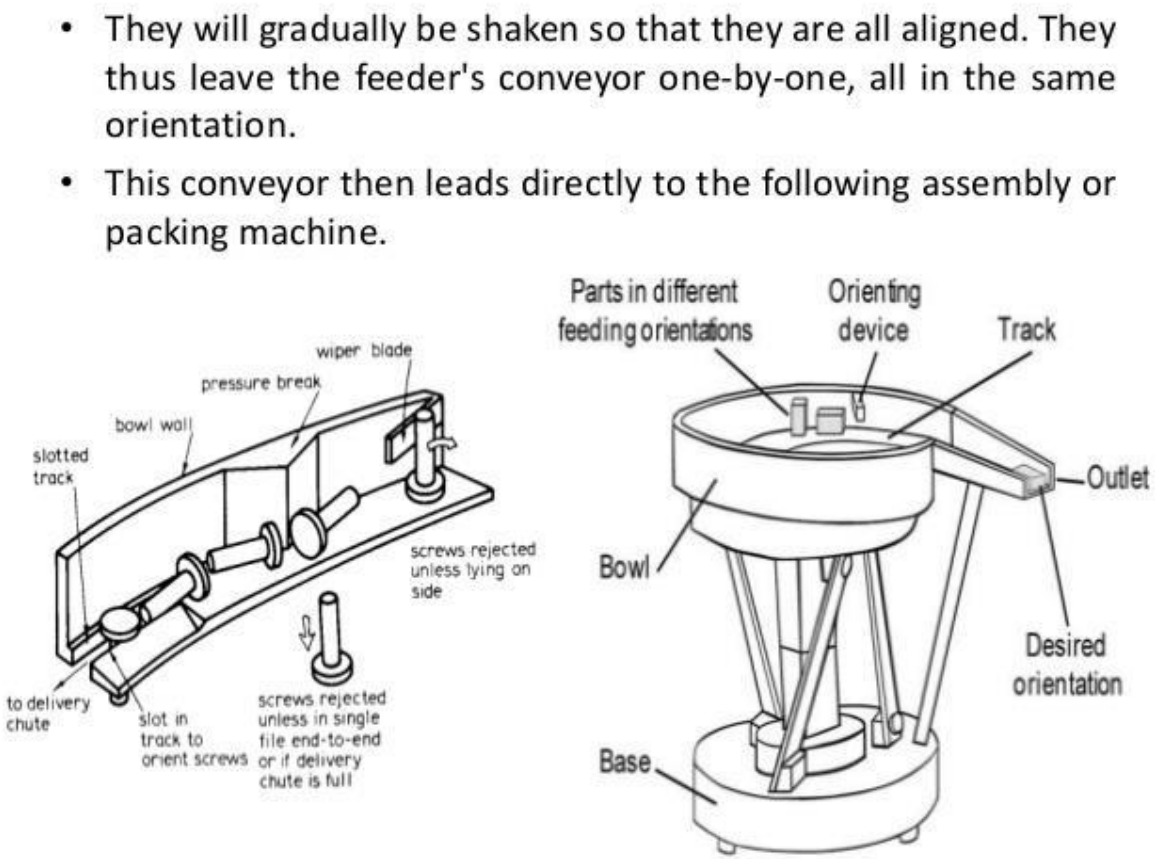
Vibracijske zdjele uobičajeni su uređaji koji se koriste za sortiranje i dodavanje uglavnom sitnijih komponenti poput u našem primjeru matica. Ovi uređaji koriste se u situacijama kada je nesortiranu skupinu dijelova potrebno dodavati u stroj jednu po jednu, u točno određenoj orijentaciji. Vibracijske zdjele pokazale su se izrazito robusnim i pouzdanim uređajima u različitim industrijskim primjenama s obzirom na to da sadrže mali broj pokretnih dijelova, dok su zdjele obično izrađene od nehrđajućeg čelika koji je za veliku većinu primjena dovoljno tvrd te stoga otporan na trošenje tijekom rada. U osnovi vibracijske zdjele sastoje se od dva glavna dijela:

1. Pogon – vibrator koji obično dolazi s pripadajućom upravljačkom jedinicom za regulaciju vibracija, dok su neke od klasičnih vrsta pogona: piezoelektrični, magnetski ili pneumatski
2. Dio za vođenje – u osnovi može biti samo zdjela(kružno vođenje) ili vodilica(linearno vođenje).

Zdjele se projektiraju prema zapremnini dijelova koje bi trebale biti u mogućnosti skladištiti, te prema obliku same staze koje služe za izdvajanje, orijentiranje i vođenje komoda u nizu. Vodilice s druge strane izvedene su kao kanali za transport već

orijentiranih dijelova, te mogu poslužiti kao spremnik već orijentiranih komada koji su spremni za daljnju uporabu.

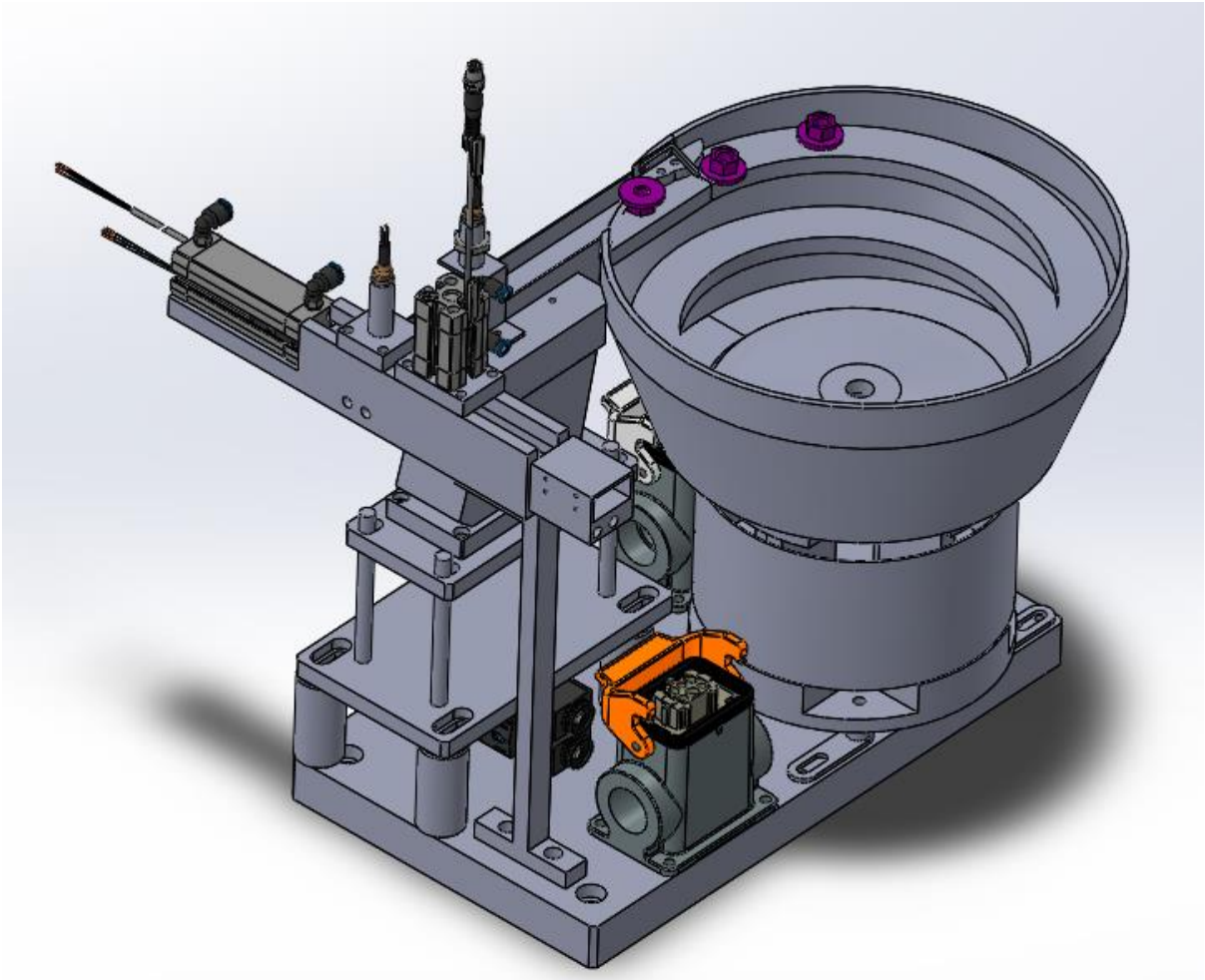
Vibracijske zdjele primjenjuju mehaničke karakteristike dijelova, njihov oblik, masu, centar mase i centar volumena kako bi pravilno orijentirali dijelove. Tako pri vibraciji zdjele dolazi do postepenog međusobnog poravnavanja dijelova, koji putuju uz stazu zdjele.



Slika 30. Princip djelovanja vibracijske zdjele [21]

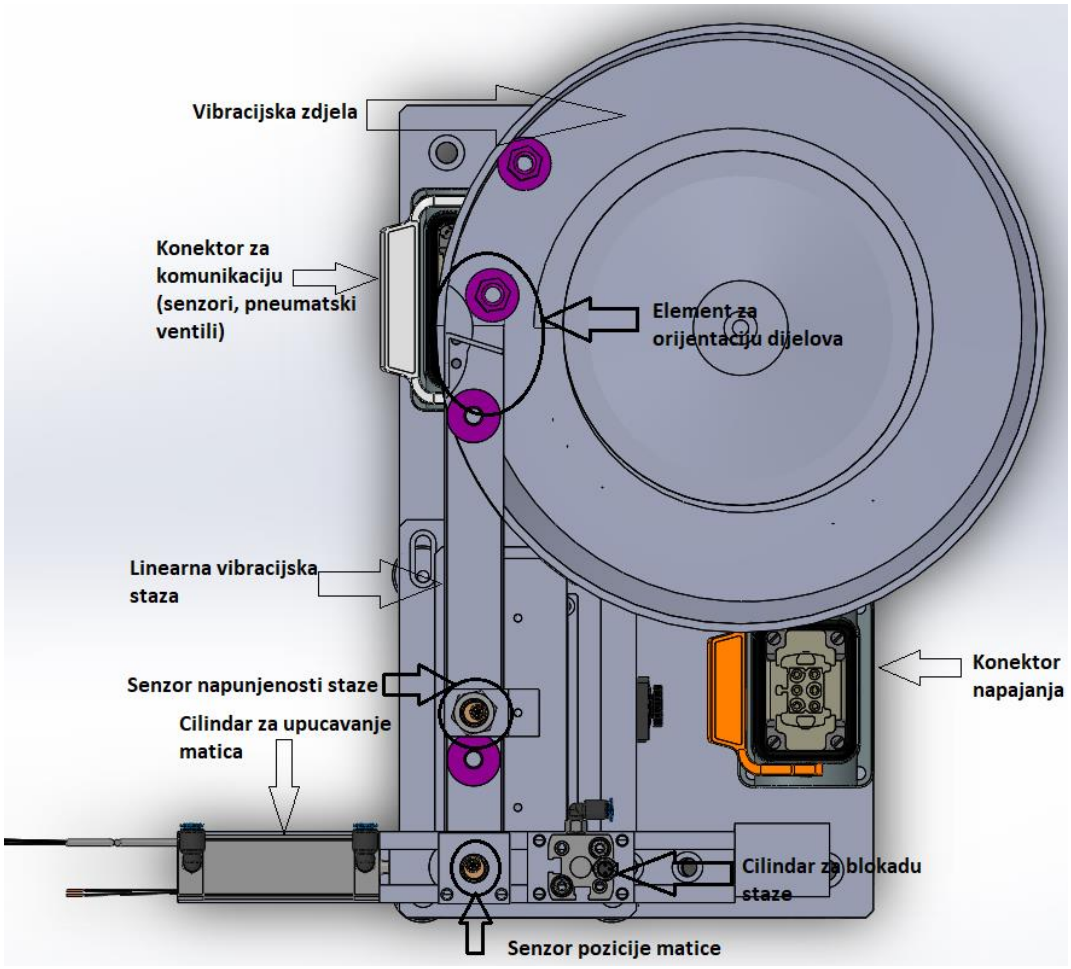
Kao što je rečeno orijentiranje dijelova u najvećem dijelu ovisi o njihovom izgledu i poziciji centra mase s obzirom na centar volumena predmeta, stoga vibracijske zdjele osobito dobro rade primjerice s vijcima, ili općenito asimetričnim dijelovima. Stoga su i staze zdjele konstruiranje posebno za svaki proizvod, iako je sami princip djelovanja isti za sve.

U slučaju da dio prođe čitav put staze do samog izlaznog kanala pogrešno orijentiran, ispred samog izlaza iz vibracijske zdjele nalazi se specifično konstruiran element za orijentaciju dijelova, koji će pogrešno orijentirane dijelove ili vratiti nazad u zdjelu, prebaciti ih u drugi kanal, ili će ih na neki način prevrtanjem ili slično prebaciti u ispravnu orijentaciju.

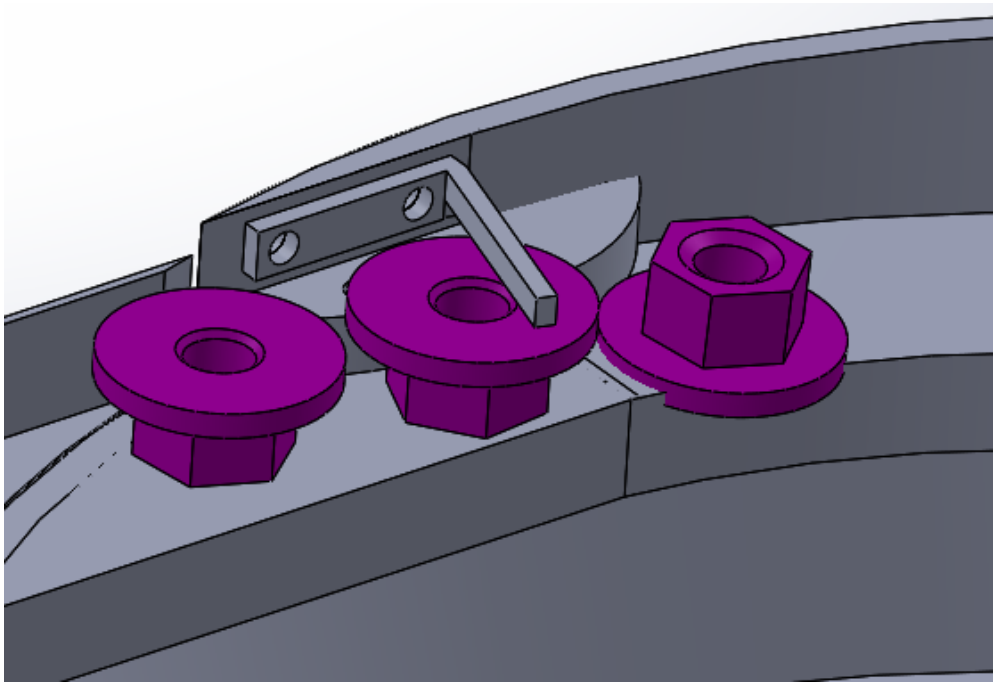


Slika 31. Vibracijska zdjela

Na slici iznad vidi se konstruirani sklop vibracijske zdjele s linearnom vibracijskom stazom, kao i elementima za upucavanje matica u savitljivu vodilicu od vinila, konstruiranu specifično za našu namjenu upucavanja matica. Na slici ispod prikazani su osnovni elementi vibrododavača.



Slika 32. Elementi vibrododavača



Slika 33. Element za orijentaciju dijelova

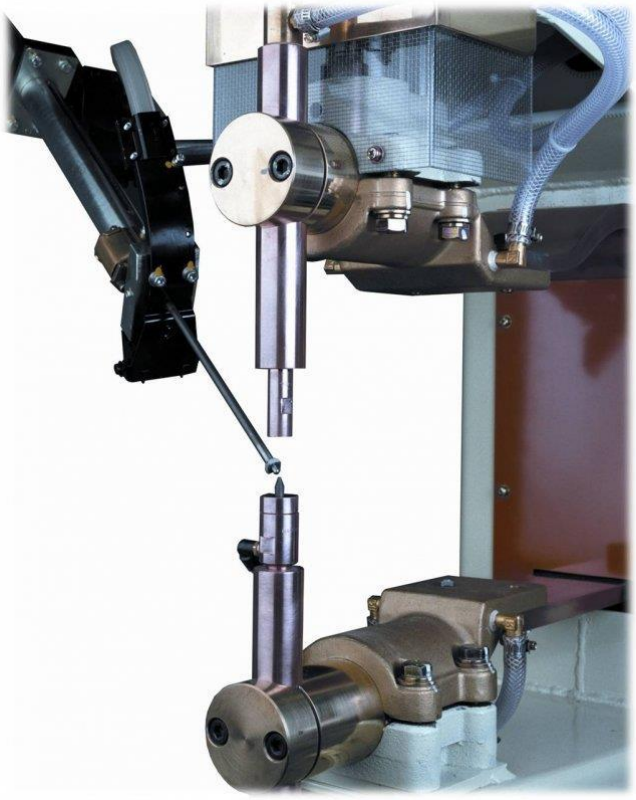
U tablici ispod nalazi se popis iskorištenih komponenti za vibrododavač:

Tablica 1. Popis dijelova vibrododavača

VIBRODODAVAČ			
Poz.	**Naziv**	**Tip**	**Kom.**
1	Vibracijska zdjela	ENSO	1
2	Linearna vibracijska staza	ENSO	1
3	Cilindar za upucavanje matica	Festo ADN-12-50-I-P-A	1
4	Senzor pozicije cilindra 3	Festo SDAS-MHS-M40-1L_PNLK-PN	2
5	Cilindar za blokadu staze	Festo ADN-12-10-I-P-A	1
6	Senzor pozicije cilindra 5	Festo SMT-8M-A-PS-24V-E-0,3-M8D	1
7	Senzor napunjenosti staze	Sick IMS12-04BPSVU2S	1
8	Senzor pozicije matice	Sick IMS12-04BPSVU2S	1
9	Elektro-pneumatski razvodnik	Festo VUVS-L20-M32C-MD-Q6-F7-1C1	2
10	Konektor za napajanje/komunikaciju	Harting Han 6ES-Press-HMC-F	2
11	Kućište konektora	Harting Han 6B-HMC-HSM2-SL-M25	2

4.2. Dodavač

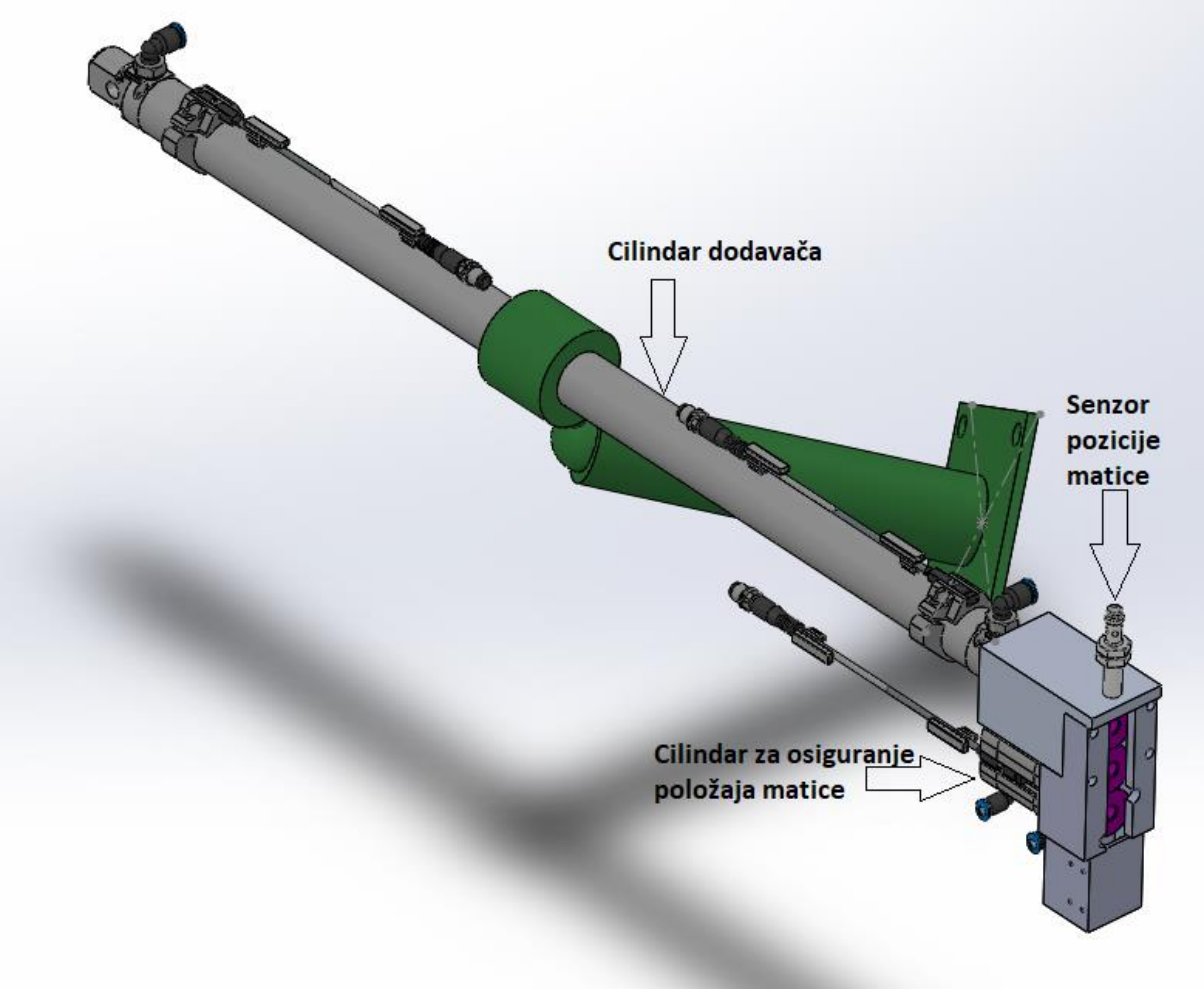
S obzirom na to da matice ne mogu po vodilici dolaziti direktno do vrha donje elektrode stroja za varenje, jer bi pri varenju vodilica dolazila u kontakt s elektrodom, matice iz vibrododavača putem savitljive vinil cijevi pravokutnog oblika(sprječavanje zakretanja matica unutar cijevi) odlaze do glave pneumatskog cilindra dodavača. Matice kroz cijev gura pneumatski cilindar za upucavanje matica, dok pneumatski cilindar za blokiranje staze blokira stazu pri svakom povratnom hodu cilindra za upucavanje, kako matice iz cijevi uslijed gravitacije ne bi mogle krenuti u suprotnome smjeru.



Slika 34. Ideja dodavača matica [6]

Po dolasku matice u glavu cilindra dodavača, što detektira senzor pozicije u glavi dodavača(vidljivo na slici 35.) aktivira se radni hod cilindra dodavača, koji jednim brzim potezom dolazi do vrha donje elektrode, te matica pada na centrirajući stožac donje elektrode, a cilindar dodavača vraća se u početni položaj. Taj izrazito brzi hod do vrha donje elektrode onemogućuje pad matice s vrha klipa, obzirom da je inercijske sile drže na klipu. Vrh klipa cilindra dodavača, izrađen je od magnetičnog materijala, što dodatno osigurava maticu od opadanja.

Slika ispod prikazuje konstruiranu glavu dodavača s cilindrom dodavača i popratnim elementima, koji je konstruiran specifično za našu primjenu.

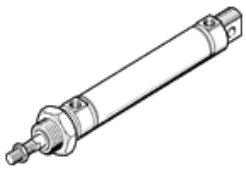


Slika 35. Dodavač matica

U tablici ispod nalazi se popis dijelova dodavača:

Tablica 2. Popis dijelova dodavača

DODAVAČ			
Poz.	**Naziv**	**Tip**	**Kom.**
1	Cilindar dodavača	Festo DSNU-25-500-PPV-A-K3	1
2	Senzor pozicije cilindra 1	Festo SMT-8M-A-24V-E-0.3-M8D	2
3	Cilindar za osiguranje pozicije matice	Festo ADN-12-10-I-P-A	1
4	Senzor cilindra 3	Festo SMT-8M-A-PS-24V-E-0,3-M8D	1
5	Senzor pozicije matice	Sick IME08	1



Tehnički podaci

Svojstvo	Vrijednost
Datum isporuke	→ Pogled
Hod	500 mm
Promjer klipa	25 mm
Navoj klipnjače	M10x1,25
Prigušivanje	P: elastični prigušni prsteni/ploče obostrano
Položaj ugradnje	proizvoljno
Odgovara normi	CETOP RP 52 P ISO 6432
Kraj klipnjače	Vanjski navoj
Konstruktivna struktura	Klip Klipnjača Cijev cilindra
Prepoznavanje pozicije	za beskontaktni prekidač
Varijante	jednostrana klipnjača
Operating pressure MPa	0,1 ... 1 MPa
Pogonski tlak	1 ... 10 bar
Način funkcioniranja	dvoradno
Maritime classification	see certificate
Pogonski medij	Komprimirani zrak prema ISO 8573-1:2010 [7:4:4]
Uputa o mediju pogona i upravljanja	Nauljeni pogon moguć (u daljnjem pogonu potreban)
Klasa korozione otpornosti KBK	2 - umjerena otpornost na koroziju
Temperatura okoline	-20 ... 80 °C
Energija naleta u krajnjim položajima	0,3 J
Theoretical force at 0.6 MPa (6 bar, 87 psi), retracting	247,4 N
Theoretical force at 0.6 MPa (6 bar, 87 psi), advance	294,5 N
Pokretna masa kod hoda 0 mm	71 g
Prirast težine po 10 mm hoda	11 g
Osnovna težina kod hoda 0 mm	238 g
Prirast pokretne mase po 10 mm hoda	6 g
Vrsta pričvršćenja	s priborom
Pneumatski priključak	G1/8
Materijal - napomena	RoHS sukladno
Material cover	Aluminijska legura za gnječenje bezbojno eloksirano
Material seals	NBR TPE-U(PU)
Material piston rod	visokolegirani čelik, nehrđajući
Material cylinder barrel	visokolegirani čelik, nehrđajući

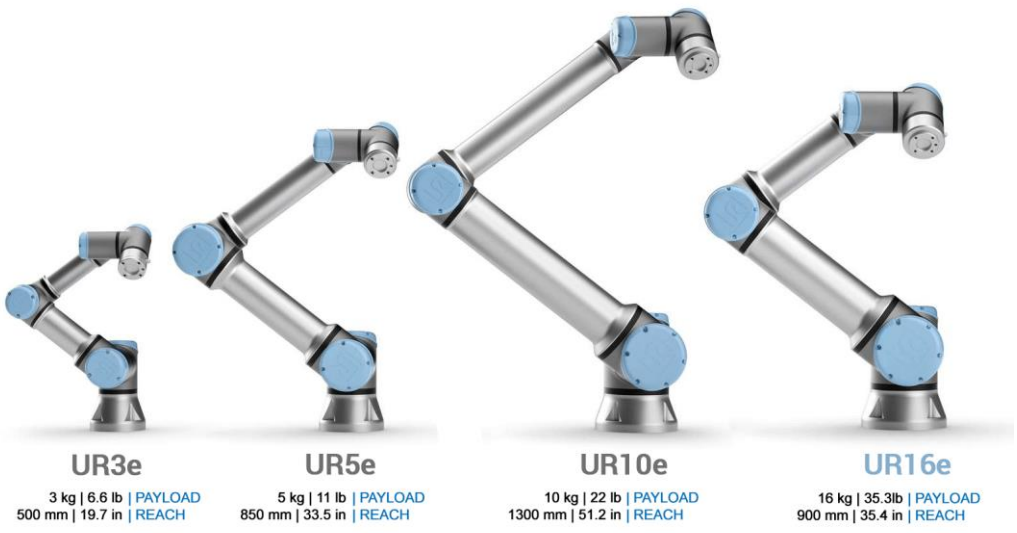
Slika 36. Tehnička specifikacija pneumatskog cilindra dodavača[8]

5. ROBOTSKA RUKA UR10e

Proizvođač kolaborativnih robotskih ruku Universal Robots danska je tvrtka smještena u Odesi, osnovana 2005.g. kada su uz suradnju sa Sveučilištem u Odesi došli do zaključka da je industrija dominirana velikim, teškim i nezgrapnim robotima. Stoga su uložili sredstva u razvoj robotske tehnologije dostupne malom i srednjem poduzetništvu. Trenutno u svom portfelju nude četiri vrste robota, prvi od kojih je razvijen i pušten na tržište još 2008.g., takozvani UR5, dok je četiri godine nakon lansirana varijanta većeg dosega i nosivosti UR10. 2015.g. predstavljen je stolni robot malih dimenzija, UR3 te najnoviji robot visoke nosivosti UR16e, koji je u mogućnosti nositi terete do 16kg. Svi UR roboti imaju šest stupnjeva slobode gibanja, izrazito su male težine u usporedbi sa ekvivalentnim(ukoliko postoje) robotima, a zahvaljujući korištenoj kolaborativnoj tehnologiji u mogućnosti su raditi uz osoblje i radnike bez potrebe ograđivanja, zahvaljujući rezultatima procjene rizika koji su bili izrazito povoljni.

Ove robote odlikuje činjenica da izrazito dobro funkcioniraju u uvjetima fleksibilne proizvodnje, poput one našeg tipa, čime nude povećanje proizvodnosti sustava, poboljšanu i ujednačenu kvalitetu proizvoda, te na posljetku povećanje kapaciteta proizvodnje i konkurentnost na tržištu.

Sigurnosne postavke zadnjih generacija robota mogu se podešavati i prilagoditi svakom pojedinom rješenju. Robotska ruka ima funkciju dva režima rada sigurnosnih funkcija, normalni režim rada i reducirani režim rada, koji se mogu mijenjati i tijekom samog rada robotske ruke.



Slika 37. Universal Robots [7]

Pri odabiru robotske ruke ulogu je igralo nekoliko faktora. Najprije sama konfiguracija ruke sa 6 stupnjeva slobode gibanja, koja je kod proizvođača Universal Robots izrazito povoljna za našu primjenu obzirom je na samom vrhu ruke kompaktno složeno 3 od 6 stupnjeva slobode, što nam daje odlične performanse pri odabiru načina izdvajanja dijelova iz paleta. Također ovo nam omogućuje rad u uskim prostorima uz lagano odabiranje najpovoljnije orijentacije ruke koja garantira izbjegavanje svih prepreka.

Također s obzirom da je jedan od ciljeva dizajna čim više sačuvati izgled radne stanice koja se sastoji od dvije gajbe veličine paleta, bitno je da robot ima dovoljan doseg kako bi mogao robot dosegnuti i najudaljenije predmete s paleta dimenzija *1000mm*1200mm*, što robot UR10e zadovoljava s obzirom na doseg od *1300mm*. Također prednost ovog robota je i njegova težina od svega *33.5kg*, pa stoga nisu potrebne nezgrapne i teške konstrukcije kako bi se robot montirao na postolje, čime ponovno štedimo prostor.

Nosivost UR10e robota kako samo ime govori jest *10kg* što je u našoj primjeni sasvim dovoljno obzirom da ga koristimo na platini koja ne teži više od četvrt kilograma. Također veliki plus je i u samom aluminijskom kućištu robota koje je odlično pri varenju, gdje se neizbježno javljaju frcanja iskri, pri čemu aluminijsko kućište neće trpjeti veća oštećenja uz minimalnu zaštitu ruke vatrootpornim materijalom.

Slika 38. Privjesak za učenje [7]

Ponovljivost UR10e robota specificirana je na $\pm 0.1mm$ što dakako zadovoljava potrebe namjene kod posluživanja stroja za varenje matica, spomenimo ovdje i činjenicu da se točnost pozicioniranja platine na donjoj elektrodi stroja za varenje osigurava i mehaničkim putem. Platina i matica se na donjoj elektrodi centriraju pomoću centrirajućih prstena i štiftova.

UR10e technical details

Performance

Power consumption	Approx. 350 W using a typical program
Collaboration operation	17 advanced adjustable safety functions incl. elbow monitoring. Remote Control according to ISO 10218
Certifications	EN ISO 13849-1, Cat.3, PL d, and EN ISO 10218-1

F/T Sensor - Force, x-y-z		F/T Sensor - Torque, x-y-z	
Range	100 N	Range	10 Nm
Resolution	2.0 N	Resolution	0.02 Nm
Accuracy	5.5 N	Accuracy	0.60 Nm

Ambient temperature range	0-50°C
Humidity	90% RH (non-condensing)

Specification

Payload	10 kg / 22 lbs
Reach	1300 mm / 51.2 in
Degrees of freedom	6 rotating joints DOF
Programming	Polyscope graphical user interface on 12 inch touchscreen with mounting

Movement

Pose Repeatability	+/- 0.05 mm, with payload, per ISO 9283	
Axis movement robot arm	Working range	Maximum speed
Base	± 360°	±120°/Sec.
Shoulder	± 360°	±120°/Sec.
Elbow	± 360°	±180°/Sec.
Wrist 1	± 360°	±180°/Sec.
Wrist 2	± 360°	±180°/Sec.
Wrist 3	± 360°	±180°/Sec.
Typical TCP speed		1 m/Sec./ 39.4 in/Sec.

Control box

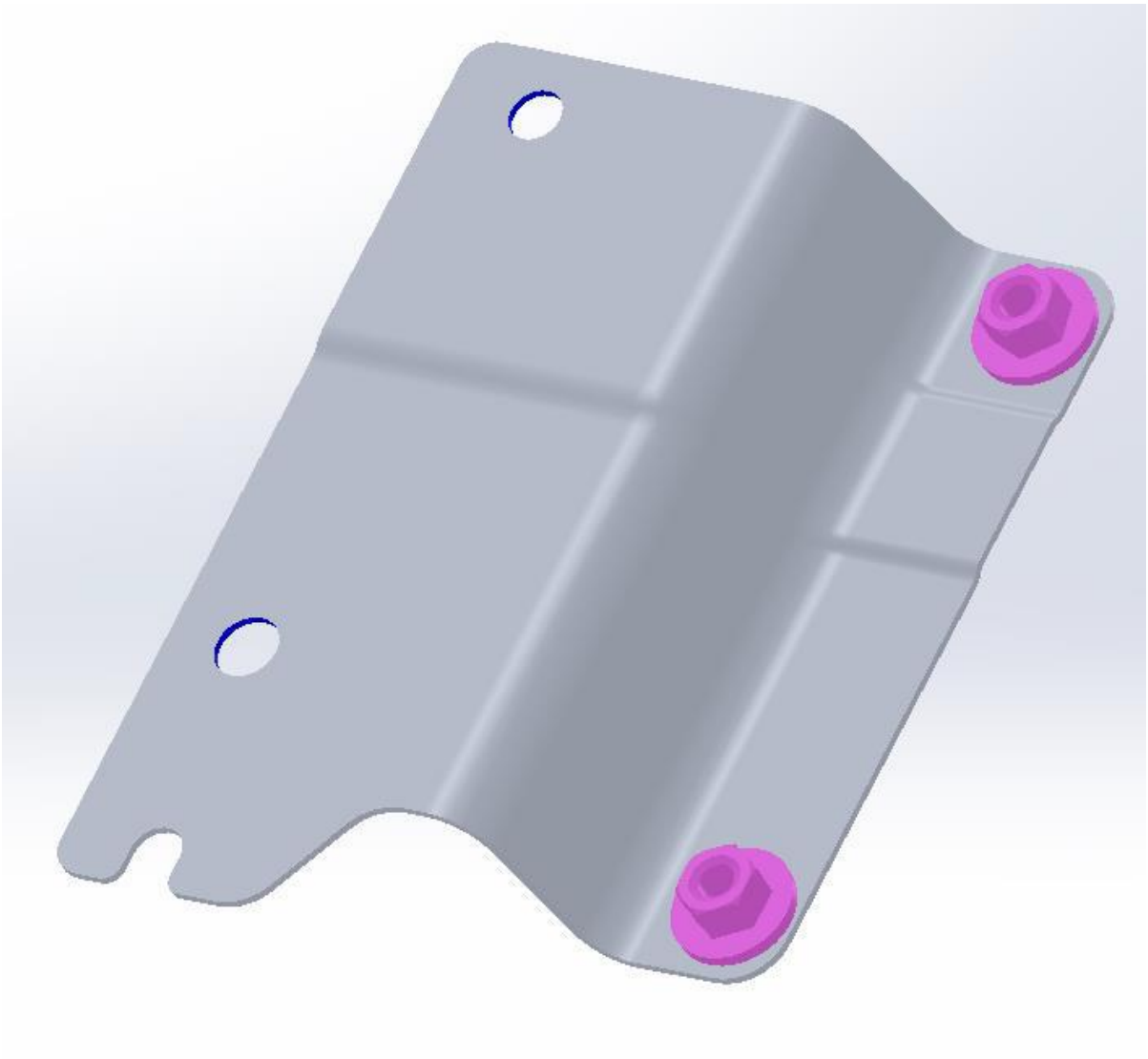
Performance

IP classification	IP44
ISO Class Cleanroom	6
Ambient temperature range	0-50°C
I/O Ports	Digital in 16 Digital out 16 Analog in 2 Analog out 2 500 Hz control, 4 separated high speed quadrature digital inputs
I/O power supply	24V 2A
Communication	Control frequency: 500 Hz ModbusTCP: 500 Hz signal frequency ProfiNet and EthernetIP: 500 Hz signal frequency USB ports: 1 USB 2.0, 1 USB 3.0
Power source	100-240 VAC, 47-440Hz
Humidity	90% RH (non-condensing)

Slika 39. Specifikacija UR10e robotske ruke [7]

6. PRIHVATNICA

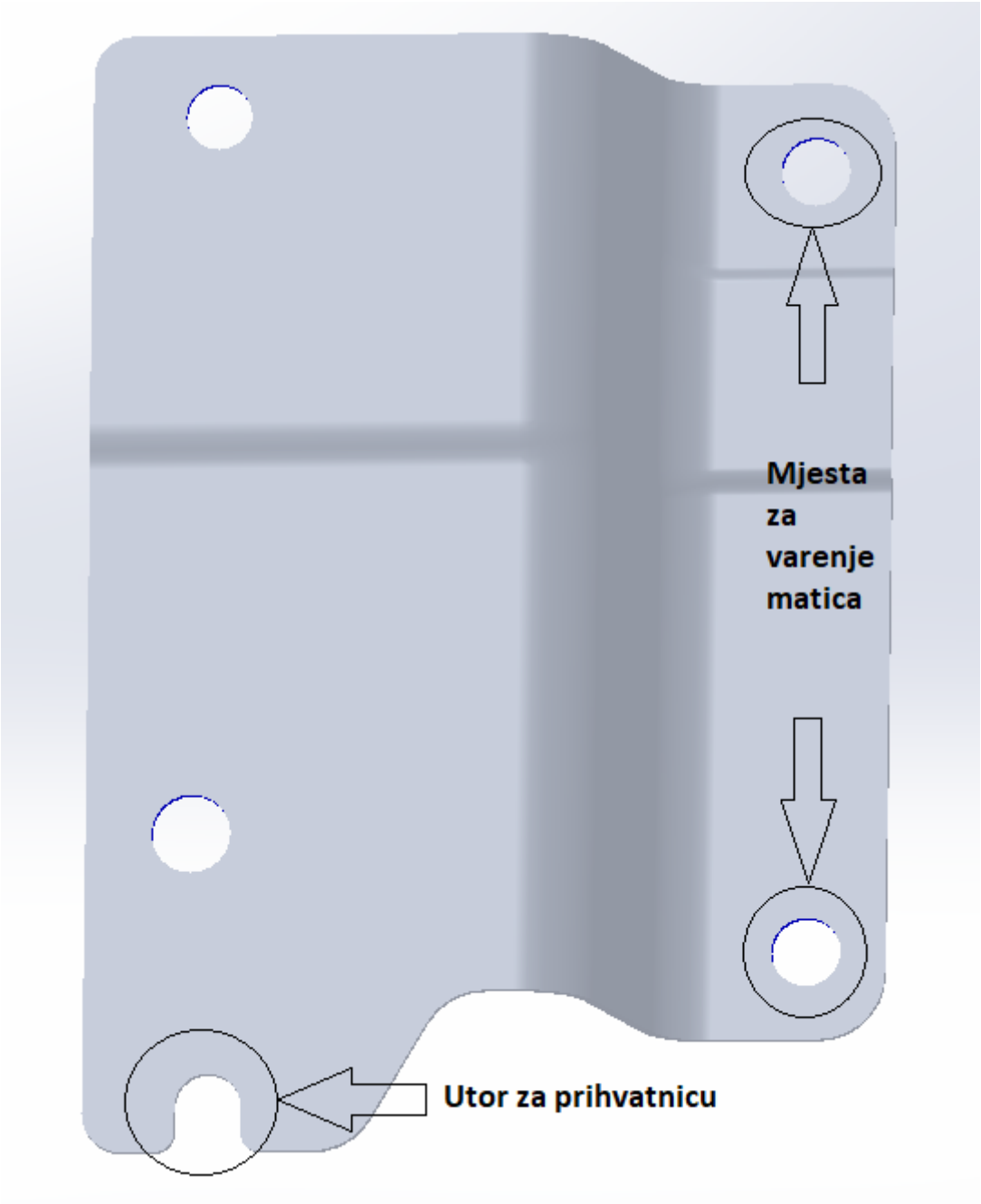
U sklopu ovog diplomskog rada izvedena je simulacija automatizacije varenja za jednu platinu odnosno tip proizvoda. U tu svrhu konstruirana je robotska dvoprstna prihvatnica koja se temelji na mehanizmu hvatanja silom. Kako je vidljivo na slici ispod na ovaj tip proizvoda potrebno je variti dvije matice na za to predviđena mjesta, matice su kako je vidljivo i na slici *11.* metričkoga navoja M8 s obrubom na kojem se s donje strane nalaze tri bradavice koje se pri varenju utapaju u osnovni materijal platine.



Slika 40. Platina

Slika ispod prikazuje istu platinu, bez matica, tako da su vidljivi kružni otvori na kojima se matice vare, također označen je i utor na lijevoj strani platine koji je iskorišten za

hvatanje platine robotskom prihvatnicom, koja na sebi ima ekvivalentnu izbočinu kojom osiguravamo pravilno hvatanje i orijentaciju platine na samoj robotskoj prihvatnici.



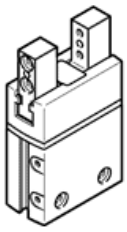
Slika 41. Pozicije na platini

Za robotsku prihvatnicu odabrana je dvoradna pneumatska hvataljka za koja su dizajnirana dva konusna prsta za hvatanje platine iz paleta. Dvoradna paralelna pneumatska hvataljka proizvođača Festo DHSP-16-A-NC nudi hod po prihvatnoj čeljusti od *5mm*, konstrukcijski je izvedena preko polužnog mehanizma i proces gibanja prihvatnih čeljusti je stoga prisilan. Također jedna od bitnih informacija jest maksimalna sila hvatanja prihvatnice koja iznosi *150N* pri pneumatskom tlaku od *8bar*.

Paralelna prihvatnica
DHPS-16-A-NC

Broj artikla: 1254045

FESTO



Opći uvjeti primjene
Tehnički podaci

Tehnički podaci

Svojstvo	Vrijednost
Datum isporuke	→ Pogled
Veličina	16
Hod po prihvatnoj čeljusti	5 mm
Maks. točnost zamjene	<= 0,2 mm
Maks. kutna zračnost prihvatnih čeljusti ax, ay	< 0,5 deg
Maks. zračnost prihvatnih čeljusti Sz	< 0,02 mm
Simetrija rotacije	<= 0,2 mm
Točnost ponavljanja, prihvatnica	< 0,02 mm
Broj čeljusti prihvatnice	2
Položaj ugradnje	proizvoljno
Način funkcioniranja	dvoradno
Funkcija prihvatnice	Paralelno
Osiguranje prihvatne sile	kod zatvaranja
Konstruktivna struktura	Poluga prisilno vođen proces gibanja
Vodilica	Klizna vodilica
Prepoznavanje pozicije	za beskontaktni prekidač
Pogonski tlak	4 ... 8 bar
Maks. radna frekvencija prihvatnice	3 Hz
Min. opening time at 0.6 MPa (6 bar, 87 psi)	48 ms
Min. closing time at 0.6 MPa (6 bar, 87 psi)	37 ms
Maks. masa po eksternom prihvatnom prstu	150 g
Pogonski medij	Komprimirani zrak prema ISO 8573-1:2010 [7:4:4]
Uputa o mediju pogona i upravljanja	Nauljeni pogon moguć (u daljnjem pogonu potreban)
Klasa korozione otpornosti KBK	1 - niska otpornost na koroziju
Temperatura okoline	5 ... 60 °C
Moment tromosti masa	0,468 kgcm2
Maks. sila na prihvatnoj čeljusti Fz, statička	150 N
Maks. moment na prihvatnoj čeljusti Mx statički	8 Nm
Maks. moment na prihvatnoj čeljusti My statički	8 Nm
Maks. moment na prihvatnoj čeljusti Mz statički	8 Nm
Interval podmazivanja elemenata vođenja	10 Mio SP
Težina proizvoda	188 g
Vrsta pričvršćenja	Unutarnji navoj i utor za centriranje s prolaznim provrtom i centrirnim tuljkom po izboru:
Pneumatski priključak	M3
Materijal - napomena	RoHS sukladno
Material cover cap	PA
Material housing	Aluminijska legura za gnječenje tvrdo eloksirano
Material gripper jaws	visokolegirani čelik, nehrđajući

Slika 42. Specifikacija dvoradne pneumatske hvataljke [8]

Za odabranu pneumatsku hvataljku proveden je proračun nosivosti prema [13], uz gore navedene karakteristike prihvatnice. Potrebna sila stezanja prihvatnice određena je prema**(1)** .

$$F_G = \frac{m \cdot (g + a)}{\mu} \cdot S \quad \textbf{(1)}$$

Gdje je:

F_G,	N,	Sila stezanja prihvatnice
m,	kg,	masa koju prihvatnica hvata(masa platine + masa prstiju prihvatnice)
g,	m/s^2,	akceleracija sile teže
a,	m/s^2,	akceleracija(uzeto 0)
S,		sigurnosni faktor
μ,		koeficijent trenja (čelik-čelik, suho) – 0.74

Ako gore navedenu jednadžbu napišemo tako da izrazimo masu koju prihvatnica hvata *m* dobivamo jednadžbu **(2)**

$$m = \frac{F_G \cdot \mu}{(g + a) \cdot S} = \frac{150N \cdot 0.74}{10m/s^2 \cdot 2} = 5.55kg \quad \textbf{(2)}$$

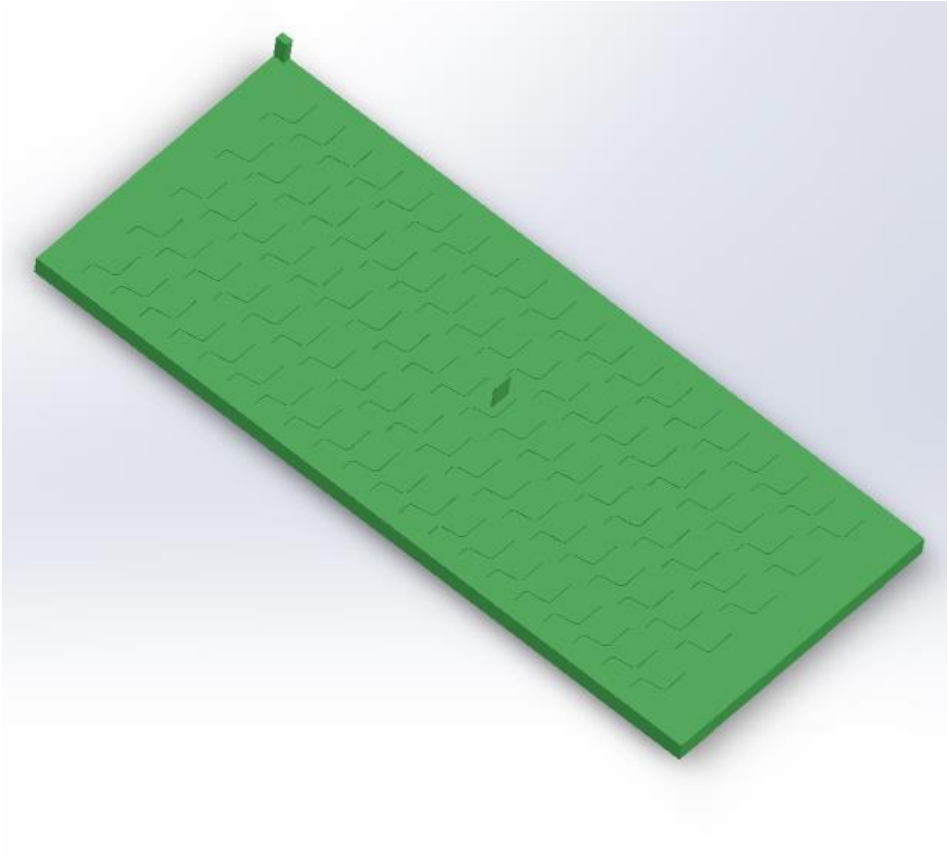
Što znači da ukupna masa koju prihvatnica može podići jest 5.55kg. Uzmemo li potom u obzir da je suma masa dva prsta na pneumatskoj prihvatnici *0,07kg* dolazimo do zaključka da ova prihvatnica može podići masu prema **(3)**

$$m_d = m - m_p = 5.55kg - 0.07kg = 5.48kg \quad \textbf{(3)}$$

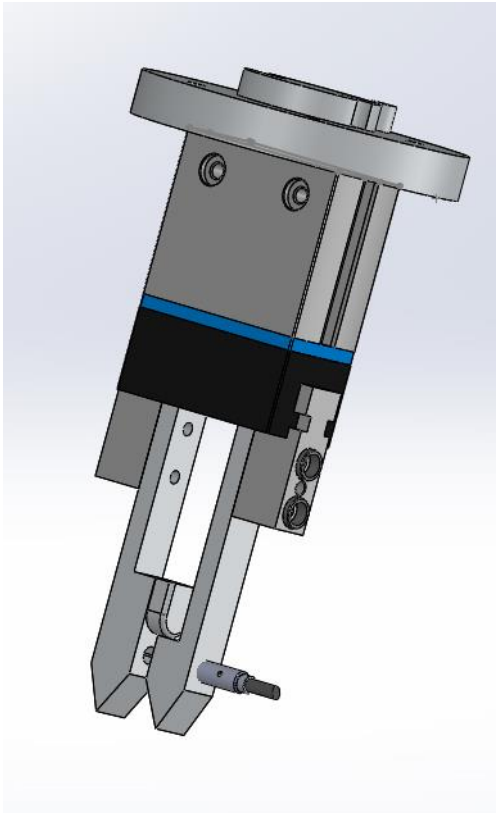
Gdje je

m_d,	kg,	maksimalna masa dijela/platine
m,	kg,	ukupna masa koju prihvatnica može podići
m_p,	kg,	ukupna masa prstiju prihvatnice – *0.07kg*

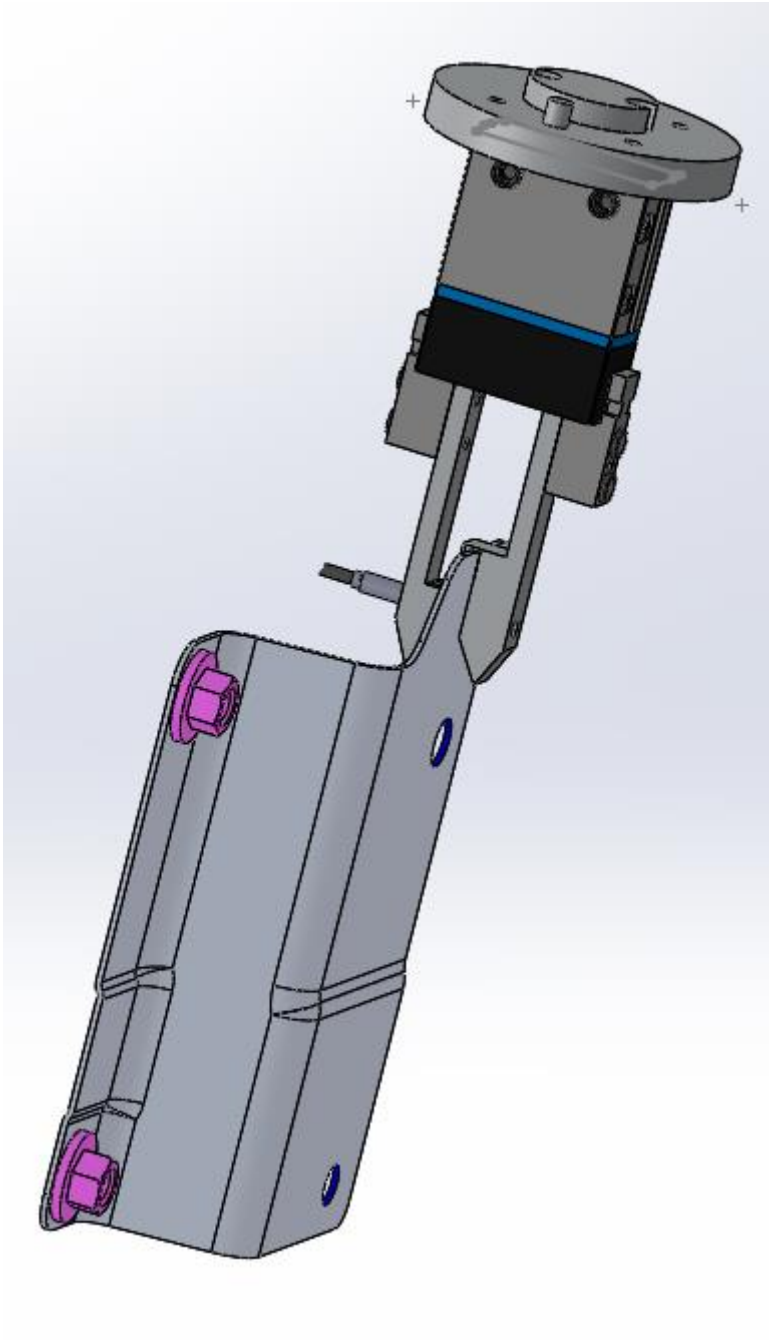
Što znači da prihvatnica bez problema može podići platinu sa zavarenim maticama prikazanu na slici *40* koja teži svega *0.27kg*. Također masa palete prikazane na slici *43.* ispod iznosi *5.2kg* što je također unutar dozvoljenog raspona.



Slika 43. Paleta



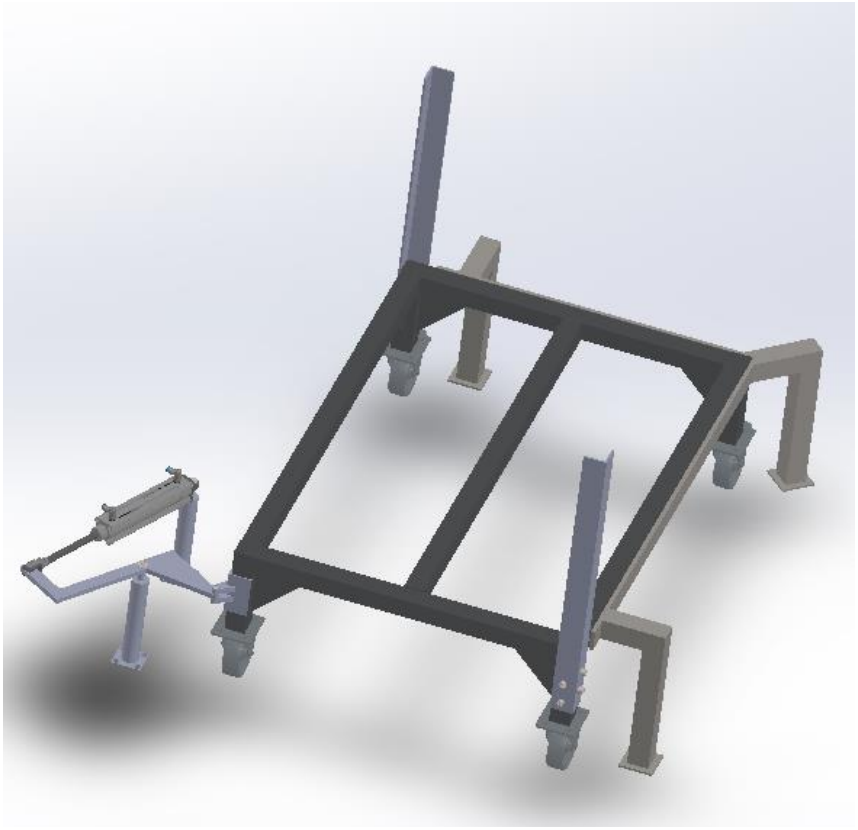
Slika 44. Prihvatnica s adapterom i senzorom



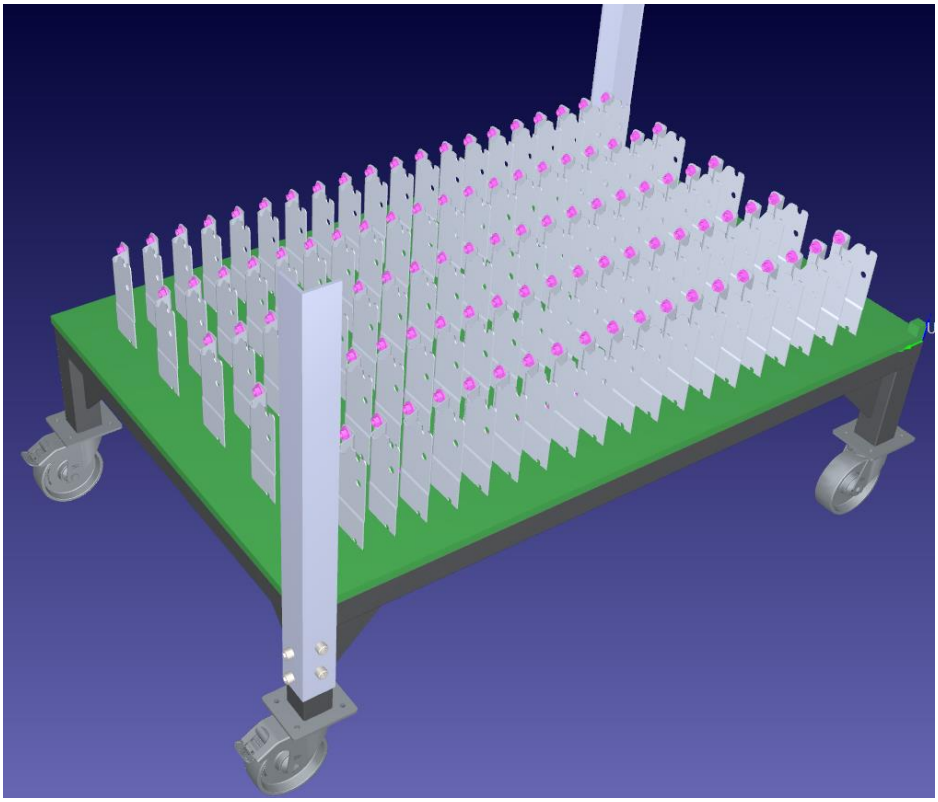
Slika 45. Prihvatnica s platinom

Paleta prikazana na slici *43.* sadrži 100 utora za postavljanje platina raspoređenih u 5 redova s po 20 mjesta za platine. Paleta na sredini ima izbočenje koje omogućuje hvatanje robotskom prihvatnicom. Kada se paleta isprazni, robot hvata paletu, podiže ju i ostavlja na za to predviđeno mjesto. Palete s platinama se nalaze na specijalno izrađenim kolicima koja se postavljaju u stanicu za automatsko varenje na za njih predviđeno mjesto s lijeve strane robota, pozicija kolica osigurana je graničnikom na koji naliježu kolica te mehanizmom koji

uz pomoć pneumatskog cilindra sigurno pritišće kolica na graničnik, osiguravajući time kolica od pomicanja.



Slika 46. Kolica s mehanizmom za zaključavanje

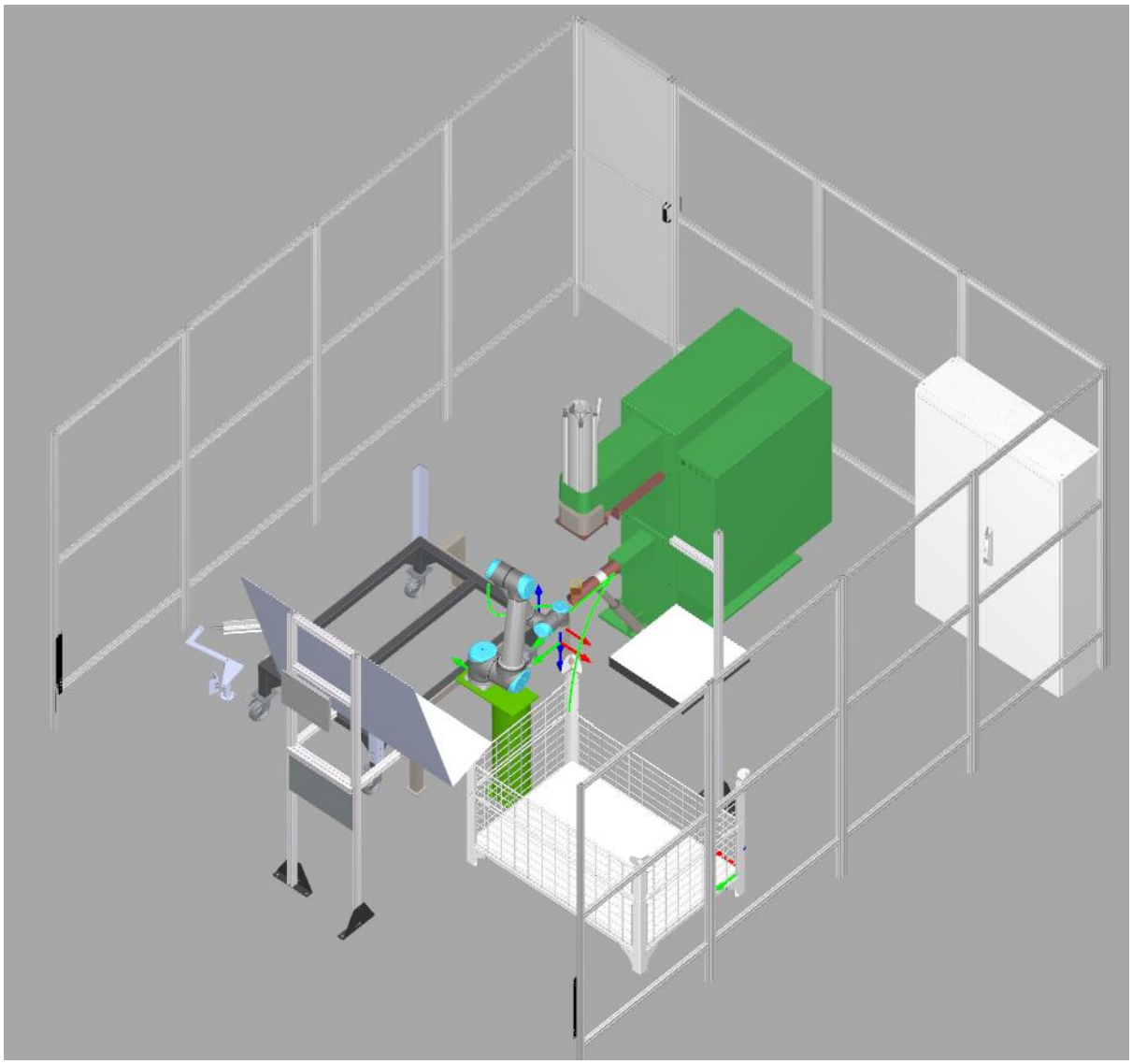


Slika 47. Kolica popunjena jednom paletom s platinama

7. ROBODK

Kako bi izveli najjednostavnije rješenje automatske stanice za varenje matica kreirano je HMI korisničko sučelje za upravljanje robotskom stanicom, te kako bi čim bolje vizualizirali rješenje koje bi se primijenilo u konačnoj izvedbi.

Simulacija rada stanice izvedena je u simulacijskom programu RoboDK, ovaj program omogućuje nam simulaciju robota u industrijskim okruženjima što nam daje priliku izvući iz robota i stanice sve mogućnosti. Program nudi intuitivno korisničko sučelje koje, ukoliko se radi o jednostavnijim zadatcima, ne zahtjeva čak ni znanja programiranja. Ipak obzirom da u ovom radu simuliramo čitavu robotsku stanicu sa popratnim mehanizmima, a kojom upravlja PLC, uz HMI korisničko sučelje, potrebna su znanja u Python programiranju, programiranju PLC te izradi HMI sučelja.

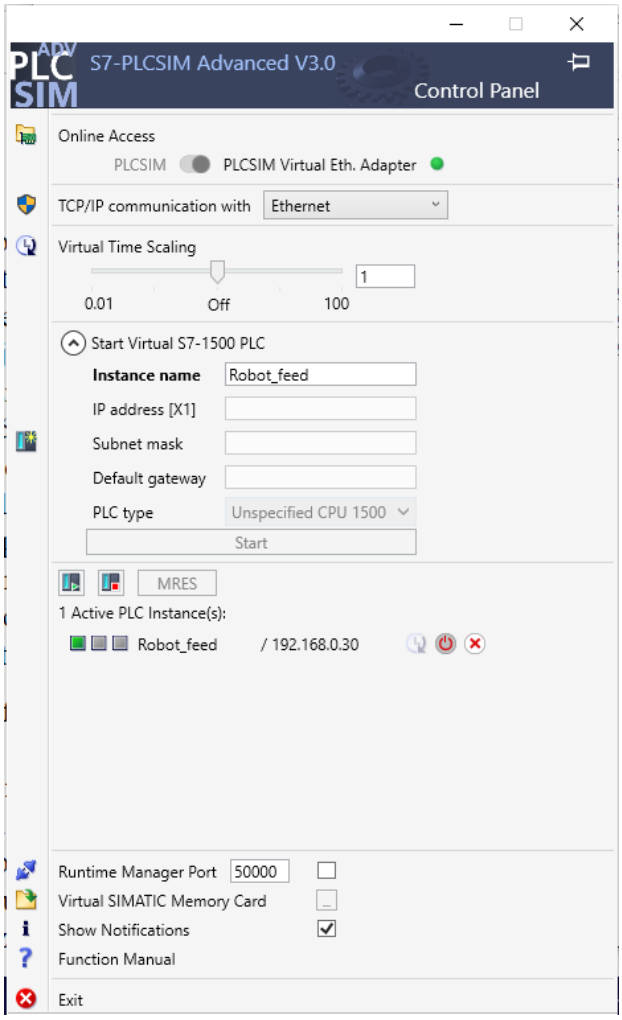


Slika 48. Izgled robotske stanice za varenje matica u RoboDK

Programsko rješenje dakle sastoji se od nekoliko glavnih dijelova koji su potrebni za simulaciju rada stanice:

1. Python kod za dvosmjernu komunikaciju RoboDK – PLC
2. Python kod za učitavanje dijelova u robotsku stanicu
3. PLC kod za upravljanje robotskom stanicom većinom pisan u SCL programskom jeziku uz povremeno korištenje LAD dijagrama
4. HMI korisničko sučelje programirano u TIA Portal sučelju
5. Python kod za vizijsku kontrolu dijelova

Za izvedbu ovih programa korišteno je nekoliko softverskih i simulacijskih paketa. Sama simulacija stanice dakle se izvodi u RoboDK programskom paketu, dok je PLC i HMI programiran unutar Siemens-ovog TIA Portal softverskog paketa, verzije 15.1. Obzirom da je unutar ovog rada izvedena simulacija, tako se i PLC simulira uz pomoć S7-PLCSIM Advanced V3.0 programa za simulaciju PLC-a.



Slika 49. S7-PLCSIM Advanced

HMI sučelje simulira se unutar WinCC RT sučelja, te se prikazuje na simulacijskom računalu. Kao što je vidljivo i na slici iznad odabran je PLC S7-1500 oznake CPU 1516-3 PN/DP, PLC koji se nalazi na vrhu Siemens-ove linije PLC-a. Ovaj PLC odabran je iz nekoliko razloga, glavni od kojih je da ovaj PLC apsolutno zadovoljava sve potrebe vezane uz procesorsku moć i brzinu potrebnu za izvođenje programa za upravljanje robotskom stanicom. Također PLC ima tri PROFInet porta za komunikaciju što je povoljno za našu primjenu obzirom da izbjegavamo korištenje komunikacijskog switcha, dok su nam potrebni portovi za komunikaciju sa HMI uređajem, robotom te računalom na kojem je pokrenut program za vizijsku kontrolu dijelova.

Slika 50. S7-1500 [11]

HMI sučelje programirano je za TP1500 Comfort panel, koji nam sa 15" ekranom osjetljivim na dodir nudi veliku slobodu pri dizajnu korisničkog sučelja.

Slika 51. TP1500 Comfort panel [12]

7.1. Komunikacija RoboDK – PLC

Za izvedbu PROFInet komunikacije između simulacijskog programa RoboDK i PLC-a izveden je Python kod u sklopu RoboDK stanice koji na PLC šalje informacije o stanjima pojedinih senzora, kao i informacije o stanju robota, koordinatama na kojima se robot nalazi s obzirom na odabrani referentni koordinatni sustav, dok PLC u suprotnome smjeru šalje naredbe za aktivaciju pojedinih mehanizama, aktivaciju ciklusa stroja za varenje matica, upravljanje dodavačem matica, te ciljne koordinate za robota.

Za izvedbu komunikacije korištena je snap7 Python biblioteka za komunikaciju. Radi se o open source biblioteci za Ethernet komunikaciju sa Siemens-ovim S7 PLC-ima. Glavne značajke biblioteke jesu:

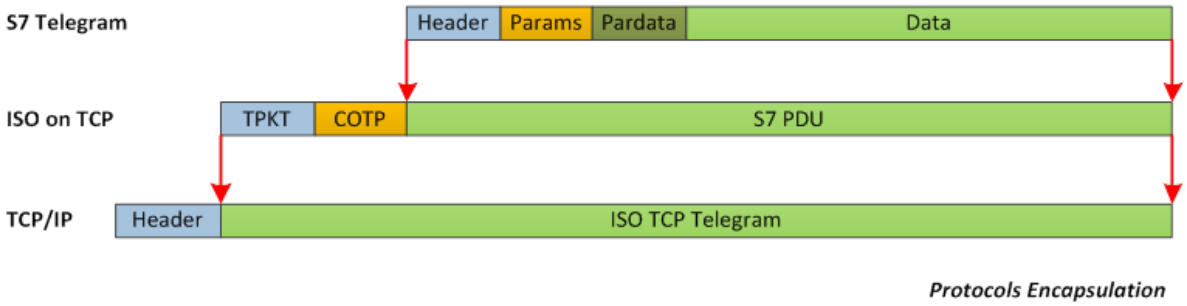
1. Podrška za 32-bitnu ili 64-bitnu arhitekturu
2. Podrška za više jezgrenu arhitekturu CPU-a
3. Neovisna o platformi, trenutno podržana za Windows, Linux, BSD, Oracle Solaris 11, Apple OSX
4. Neovisna o bibliotekama treće strane/nepoznatih autora, nema potrebe za posebnim instalacijama i konfiguracijama
5. Dvije vrste prijenosa podataka: klasični sinkroni i asinkroni

S7 protokol je temelj Siemens-ove komunikacije, njegova Ethernet primjena zasniva se na ISO TCP (RFC1006) koja je prema svom dizajnu blokovski orijentirana. Svaki blok ove komunikacije zove se PDU (*eng. Protocol Data Unit*), te njegova maksimalna duljina ovisi o CP-u te se određuje prilikom same konekcije. S7 protokol je funkcijski orijentiran ili komandno orijentiran, tj. svaki prijenos podataka sadrži komandu ili odgovor na komandu. Ako komanda svojom veličinom ne stane unutar PDU-a, mora biti razdvojena na više naknadnih PDU-ova.

Svaka naredba sastoji se od:

1. Zaglavlja
2. Seta parametara
3. Podataka o parametrima
4. Podatkovni blok

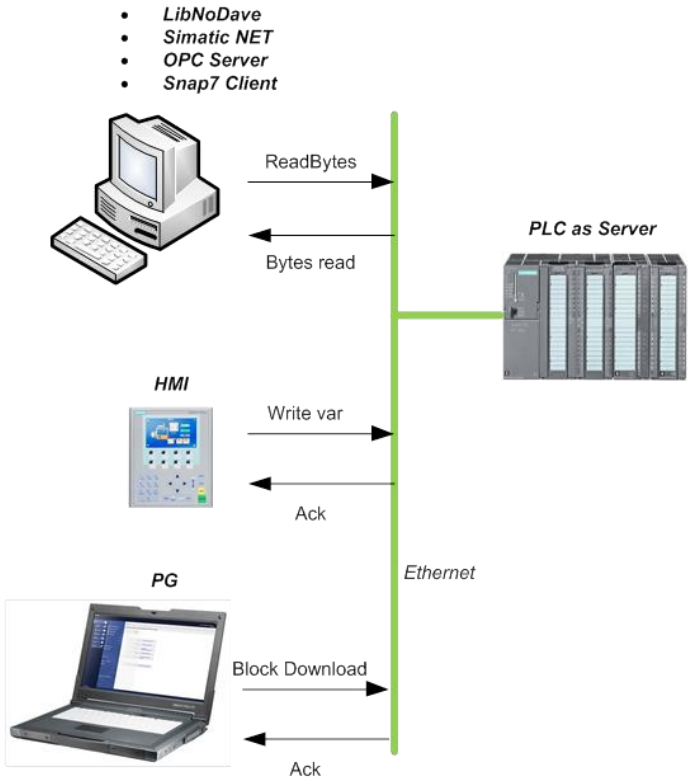
Prva dva elementa naredbi uvijek su prisutna dok su ostali elementi opcionalni.



Slika 52. Inkapsulacijski protokol [14]

S7 komunikacija sastoji se od nekoliko kategorija naredbi:

1. Pisanje/čitanje podataka
2. Cikličko pisanje/čitanje podataka
3. Informacije o direktoriju
4. Sistemske informacije
5. Pomicanje blokova
6. Upravljanje PLC-om
7. Datum i vrijeme
8. Sigurnost
9. Programiranje.



Slika 53. Primjer topografije komunikacije s PLC-om [14]

U nastavku ćemo proći kroz Python kod za komunikaciju RoboDK s PLC-om. Najprije uvozimo sve potrebne biblioteke u Python skriptu:

```
from robolink import *    # RoboDK API
from robodk import *      # Robot toolbox
RDK = Robolink()
import snap7              #biblioteka za komunikaciju s PLC
from snap7 import util
from Parts_load import *  #Skripta za inicijalizaciju i učitavanje dijelova u RoboDK
from vision_control import *  #Skripta za vizijsku kontrolu dijelova
import numpy as np
```

Slika 54. Uvoz biblioteka

S obzirom da je PLC podešen na adresu '192.168.0.30', te se nalazi u redu 0 na poziciji 1. Ovo su svi parametri koji su nam potrebni za uspostavu komunikacije s PLC-om, što činimo sa sljedećim kodom:

```
IP_ADDRESS = '192.168.0.30'
RACK = 0                        #Parametri komunikacijskog kanala
SLOT = 1
plc = snap7.client.Client()
plc.connect(IP_ADDRESS,RACK,SLOT)   #Uspostava komunikacijske veze
Connect_status = plc.get_connected()
print('Status konekcije', Connect_status)

Initialize()                        #Funkcija za inicijalizaciju stanice
```

Slika 55. PLC komunikacija

Nakon uspostavljene konekcije potrebno je provesti inicijalizaciju robotske stanice u RoboDK, povezati sve objekte unutar stanice s Python skriptom, učitati referentne koordinatne sustave, provesti inicijalizaciju ciljnih točaka robota, učitati putanje robota za kretanje po stanici i slično. Također učitavamo file-ove 3D modela platine i palete kako bi ih potom mogli po potrebi pozicionirati unutar stanice.

```
############################
####FUNKCIJA ZA INICIJALIZACIJU
############################
def Initialize():
    Main_frame = RDK.Item('Robot_feed')

    ##FRAMES
    global  Load_frame, Unload_frame
    Load_frame = RDK.Item('Load_frame')
    if not Load_frame.Valid():
        Load_frame = RDK.AddFrame('Load_frame', Main_frame)
        Load_frame.setPose(transl(-1784,1406,158))

    Unload_frame = RDK.Item('Unload_frame')
    if not Unload_frame.Valid():
        Unload_frame = RDK.AddFrame('Unload_frame', Main_frame)
        Unload_frame.setPose(transl(66,916,369.73))

     ########
    weld = [[-160,995.5,818.2,90,0,90],[-326,998.86,818.2,90,0,90],[-214.5,927,809,90,-90,0],[-128,959,809,90,-90,0]]
    weld_target_1 = [-160,995.5,818.2,90,0,90] #np
    weld_target_1_approach = [-160, 995.5, 850, 90, 0, 90] #np
    weld_approach = [[-160, 995.5, 850, 90, 0, 90],[-326,998.86,850,90,0,90],[-214.5,927,850,90,-90,0],[-128,959,850,90,-90,0]]
    weld_target_2 = [-326,998.86,818.2,90,0,90]#np
    weld_target_2_approach = [-326,998.86,850,90,0,90]#np
     ############################

    paleta_files = [r'C:\Users\jogam\OneDrive\Desktop\Diplomski_rad\RoboDK\parts\paleta.step', r'C:\Users\jogam\OneDrive\Desktop\Diplomski_rad\RoboDK\parts\paleta2.step']
    paleta_q = [100,36,0]
    paleta_dimension = [[5,20],[6,6] ]
    part_files = [r'C:\Users\jogam\OneDrive\Desktop\Diplomski_rad\RoboDK\parts\Part.step', r'C:\Users\jogam\OneDrive\Desktop\Diplomski_rad\RoboDK\parts\Part2.stp']
    part_name = ['part1', 'part2']
    part_offset = [[76.5,101,169.5,-90,0,-90],[84,172,43,0,0,-90]]
    part_axis_offset = [[150,55,176.5],[125,185,52]]
    part_targets = [[76.5,101,154,-360,0,180],[84.5,172,39,-102.75,-90,-77.3]]
    ###############
    return
```

Slika 56. Inicijalizacija stanice

Također unutar Python skripte imamo funkcije za izvršavanje određenih naredbi robota, funkcije *Attach()* i *Dettach()* služe za aktivaciju i deaktivaciju robotske hvataljke, funkcija *Reset_wait_status()* resetira PLC varijablu *wait_status* koja služi za pauziranje izvođenja programa dok nisu izvršeni određeni uvjeti poput dolaska robota u ciljnu točku i slično. Funkcije *Go_to_target()* i *Go_to_target_L()* aktiviraju takozvane *Joint* ili *Linear* kretnje robota u ciljnu točku. Dok naredba robota za *Joint* kretnju izvodi putanju robota paraboličnog oblika ovisnog o samoj ciljnoj točci, *Linear* kretnja osim što prisiljava robot na pravocrtnu linearnu kretnju, također je i osjetljiva na singularnosti robota i limite pojedinih zglobova robota.

```
def Attach():
    tool.AttachClosest()
    s.gripper_status = 1
    Reset_wait_status()
def Dettach(ActiveFrame):
    tool.DetachAll(ActiveFrame)
    s.gripper_status = 0
    Reset_wait_status()
def Reset_wait_status():
    DB_number = 1
    Start_address = 0
    Size = 11
    DB_toggle = plc.db_read(DB_number,Start_address,Size)
    util.set_bool(DB_toggle,10,2,False)
    plc.db_write(DB_number,Start_address,DB_toggle)
def Go_to_target():
    DB_number = 2
    Start_address = 0
    Size = 48
    DB_robot_coordinates = plc.db_read(DB_number, Start_address, Size)
    s.TCP1 = util.get_real(DB_robot_coordinates,24)
    s.TCP1 = round(s.TCP1,2)
    s.TCP2 = util.get_real(DB_robot_coordinates, 28)
    s.TCP2 = round(s.TCP2,2)
    s.TCP3 = util.get_real(DB_robot_coordinates, 32)
    s.TCP3 = round(s.TCP3,2)
    s.TCP4 = util.get_real(DB_robot_coordinates, 36)
    s.TCP4 = round(s.TCP4,2)
    s.TCP5 = util.get_real(DB_robot_coordinates, 40)
    s.TCP5 = round(s.TCP5,2)
    s.TCP6 = util.get_real(DB_robot_coordinates, 44)
    s.TCP6 = round(s.TCP6,2)
    s.TCP = [s.TCP1,s.TCP2,s.TCP3,s.TCP4,s.TCP5,s.TCP6]
    target = KUKA_2_Pose(s.TCP)
    frame_pose = robot.PoseFrame()
    robot_Joints = robot.Joints()
    target_joints = robot.SolveIK(target, robot_Joints, tool_pose, frame_pose)
    if len(target_joints.list()) < num_dofs:
        s.moveJ_error = True
        DB_communication_number = 1
        Start_address = 0
        Size = 15
        DB_toggle = plc.db_read(DB_communication_number,Start_address,Size)
        util.set_bool(DB_toggle,14,0,s.moveJ_error)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
    elif s.wait_status==False:
        Toggle_Frames()
        robot.MoveJ(target)
        Reset_wait_status()
```

Slika 57. Funkcije robota(1.dio)

```
def Go_to_target_L():
    DB_number = 2
    Start_address = 0
    Size = 48

    DB_robot_coordinates = plc.db_read(DB_number, Start_address, Size)

    s.TCP1 = util.get_real(DB_robot_coordinates,24)
    s.TCP1 = round(s.TCP1,2)
    s.TCP2 = util.get_real(DB_robot_coordinates, 28)
    s.TCP2 = round(s.TCP2,2)
    s.TCP3 = util.get_real(DB_robot_coordinates, 32)
    s.TCP3 = round(s.TCP3,2)
    s.TCP4 = util.get_real(DB_robot_coordinates, 36)
    s.TCP4 = round(s.TCP4,2)
    s.TCP5 = util.get_real(DB_robot_coordinates, 40)
    s.TCP5 = round(s.TCP5,2)
    s.TCP6 = util.get_real(DB_robot_coordinates, 44)
    s.TCP6 = round(s.TCP6,2)
    s.TCP = [s.TCP1,s.TCP2,s.TCP3,s.TCP4,s.TCP5,s.TCP6]
    target = KUKA_2_Pose(s.TCP)
    Toggle_Frames()
    robot.MoveL(target)
    Reset_wait_status()
```

Slika 58. Funkcije robota(2.dio)

Funkcija *Comm()* služi za slanje stanja senzora prema PLC-u, također služi i za slanje podataka o trenutnim koordinatama robota na PLC. Ove radnje izvodimo naredbama *snap7* biblioteke *db_write()* kojima su ulazne varijable broj Dana blocka u koji se upisuju podatci, početna pozicija od koje se upisuju podatci u DB, te sama poruka.

```
def Comm():
    DB_number = 1
    Start_address = 0
    Size = 2

    DB_communication_in = plc.db_read(DB_number,Start_address,Size)
    util.set_bool(DB_communication_in,0,0,s.dalex_home_s)
    util.set_bool(DB_communication_in,0,1,s.dalex_out_s)
    util.set_bool(DB_communication_in,0,2,s.dodavac_in_s)
    util.set_bool(DB_communication_in,0,3,s.dodavac_out_s)
    util.set_bool(DB_communication_in,0,4,s.block_in_s)
    util.set_bool(DB_communication_in,0,5,s.hold_out_s)
    util.set_bool(DB_communication_in,0,6,s.push_in_s)
    util.set_bool(DB_communication_in,0,7,s.push_out_s)
    util.set_bool(DB_communication_in,1,0,s.lock_open_s)
    util.set_bool(DB_communication_in,1,1,s.lock_closed_s)

    plc.db_write(DB_number,Start_address,DB_communication_in)

    TCP_coord = robot.Pose()
    TCP_coord = Pose_2_KUKA(TCP_coord)
    TCP_coord = ['%.2f' % elem for elem in TCP_coord]

    DB_number = 2
    Start_address = 0
    Size = 48

    DB_robot_coordinates = plc.db_read(DB_number, Start_address, Size)
    util.set_real(DB_robot_coordinates,0, float(TCP_coord[0]))
    util.set_real(DB_robot_coordinates,4, float(TCP_coord[1]))
    util.set_real(DB_robot_coordinates,8, float(TCP_coord[2]))
    util.set_real(DB_robot_coordinates,12, float(TCP_coord[3]))
    util.set_real(DB_robot_coordinates,16, float(TCP_coord[4]))
    util.set_real(DB_robot_coordinates,20, float(TCP_coord[5]))

    plc.db_write(DB_number, Start_address, DB_robot_coordinates)
```

Slika 59. *Comm()* funkcija

Simulacija rada pozicijskih senzora pojedinih cilindara što na dodavaču što na senzoru za varenje izvodi se unutar klasnog objekta *IO()* gdje se provjeravaju trenutne pozicije pojedinih mehanizama RoboDK te se prema tome aktiviraju ili deaktiviraju pozicijski senzori

```
def __call__(self):
    if dalex.Pose() == Dalex_Home.Pose():
        self.dalex_home_s = True
        self.dalex_out_s = False
    elif dalex.Pose() == Dalex_Out.Pose():
        self.dalex_home_s = False
        self.dalex_out_s = True
    else:
        self.dalex_home_s = False
        self.dalex_out_s = False

    if dodavac.Pose() == Dodavac_In.Pose():
        self.dodavac_in_s = True
        self.dodavac_out_s = False
    elif dodavac.Pose() == Dodavac_Out.Pose():
        self.dodavac_in_s = False
        self.dodavac_out_s = True
    else:
        self.dodavac_in_s = False
        self.dodavac_out_s = False

    if blocking_mechanism.Pose() == Block_In.Pose():
        self.block_in_s = True
    else:
        self.block_in_s = False

    if holding_piston.Pose() == Hold_Out.Pose():
        self.hold_out_s = True
    else:
        self.hold_out_s = False

    if push_piston.Pose() == Push_In.Pose():
        self.push_in_s = True
        self.push_out_s = False
    elif push_piston.Pose() == Push_Out.Pose():
        self.push_in_s = False
        self.push_out_s = True
    else:
        self.push_in_s = False
        self.push_out_s = False

    if locking_mechanism_open.Visible() == True:
        self.lock_open_s = True
        self.lock_closed_s = False
    elif locking_mechanism_closed.Visible() == True:
        self.lock_open_s = False
        self.lock_closed_s = True
    else:
        self.lock_open_s = False
        self.lock_closed_s = False
```

Slika 60. Simulacija pozicijskih senzora

Funkcija *Toggle_mechanisms()* služi za čitanje naredbi s PLC, što u principu znači čitanje pojedinih varijabli unutar data blockova PLC-a prema kojima se potom aktiviraju pojedini mehanizmi, primjerice cilindar dodavača, ili se aktivira radni ciklus stroja za varenje matica i slično. Podatke s PLC-a također čitamo naredbom iz *snap7* biblioteke *db_read()* čiji su ulazni parametri broj data blocka koji čitamo, početna adresa od koje čitamo dana block, i veličinu podatka u bajtovima koju želimo pročitati, iz čega potom naredbama poput *get_bool(),* izvlačimo iz poruke željene podatke, bitove, realne brojeve i slično.

```
def Toggle_mechanisms():
    DB_communication_number = 1
    Start_address = 0
    Size = 13
    #toggling mechanisms
    DB_toggle = plc.db_read(DB_communication_number,Start_address,Size)
    dalex_toggle = util.get_bool(DB_toggle,1,2)
    dodavac_toggle = util.get_bool(DB_toggle,1,3)
    block_toggle = util.get_bool(DB_toggle,1,4)
    hold_toggle = util.get_bool(DB_toggle,1,5)
    push_toggle = util.get_bool(DB_toggle,1,6)
    lock_toggle = util.get_bool(DB_toggle,1,7)
    if dalex_toggle == True:
        dalex.MoveL(Dalex_Out)
    elif dalex_toggle == False:
        dalex.MoveL(Dalex_Home)
    if dodavac_toggle == True:
        dodavac.MoveL(Dodavac_Out)
    elif dodavac_toggle == False:
        dodavac.MoveL(Dodavac_In)
    if block_toggle == True:
        blocking_mechanism.MoveL(Block_Out)
    elif block_toggle == False:
        blocking_mechanism.MoveL(Block_In)
    if hold_toggle == True:
        holding_piston.MoveL(Hold_Out)
    elif hold_toggle == False:
        holding_piston.MoveL(Hold_In)
    if push_toggle == True:
        push_piston.MoveL(Push_Out)
    elif push_toggle == False:
        push_piston.MoveL(Push_In)
    if lock_toggle == True:
        locking_mechanism_closed.setVisible(True, False)
        locking_mechanism_open.setVisible(False, False)
    elif lock_toggle == False:
        locking_mechanism_closed.setVisible(False, False)
        locking_mechanism_open.setVisible(True, False)
    #toggling reference frames
    Main_frame_toggle = util.get_bool(DB_toggle,2,0)
    Unload_frame_toggle = util.get_bool(DB_toggle,2,1)
    Load_frame_toggle = util.get_bool(DB_toggle,2,2)
    Curve_frame_toggle = util.get_bool(DB_toggle,2,3)
    UR10_frame_toggle = util.get_bool(DB_toggle,2,4)
    Feeder_frame_toggle = util.get_bool(DB_toggle,2,5)
    Dodavac_frame_toggle = util.get_bool(DB_toggle,2,6)
    Dalex_frame_toggle = util.get_bool(DB_toggle,2,7)
    Block_frame_toggle = util.get_bool(DB_toggle,3,0)
    if Main_frame_toggle == True:
        robot.setPoseFrame(Dalex_frame)
        s.ActiveFrame = Dalex_frame
        util.set_bool(DB_toggle,2,0,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
    elif Unload_frame_toggle == True:
        robot.setPoseFrame(Unload_frame)
        util.set_bool(DB_toggle,2,1,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
        s.ActiveFrame = Unload_frame
    elif Load_frame_toggle == True:
        robot.setPoseFrame(Load_frame)
        util.set_bool(DB_toggle,2,2,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
        s.ActiveFrame = Load_frame
```

Slika 61. Funkcija *Toggle_mechanisms() - 1.dio*

```
    elif UR10_frame_toggle == True:
        robot.setPoseFrame(UR10_frame)
        util.set_bool(DB_toggle,2,4,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
        s.ActiveFrame = UR10_frame
    elif Feeder_frame_toggle == True:
        robot.setPoseFrame(Feeder_frame)
        util.set_bool(DB_toggle,2,5,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
        s.ActiveFrame = Feeder_frame
    elif Dodavac_frame_toggle == True:
        robot.setPoseFrame(Dodavac_frame)
        util.set_bool(DB_toggle,2,6,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
        s.ActiveFrame = Dodavac_frame
    elif Dalex_frame_toggle == True:
        robot.setPoseFrame(Dalex_frame)
        util.set_bool(DB_toggle,2,7,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
        s.ActiveFrame = Dalex_frame
    elif Block_frame_toggle == True:
        robot.setPoseFrame(Block_frame)
        util.set_bool(DB_toggle,3,0,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
        s.ActiveFrame = Block_frame
    #Toggling gripper
    s.gripper_toggle = util.get_bool(DB_toggle,3,7)
    if s.gripper_toggle == True and s.gripper_status == 0:
        Attach()
    if s.gripper_toggle == False and s.gripper_status == 1:
        Dettach(s.ActiveFrame)
    #Toggle parts load
    s.load_parts = util.get_bool(DB_toggle,4,0)
    part_quantity = util.get_int(DB_toggle,6)
    number_of_part = util.get_int(DB_toggle,8)
    s.number_of_part = number_of_part
    if s.load_parts == True:
        target_unload, paleta, rows, columns, paleta_q, paleta_pick = Insert_parts_in_station(number_of_part, part_quantity)
        Write_target_list(target_unload, paleta, rows, columns, paleta_q, paleta_pick)
        util.set_bool(DB_toggle,4,0,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
    #Reset station bool
    s.reset_station = util.get_bool(DB_toggle,10,0)
    if s.reset_station == True:
        Reset_station()
        robot.setPoseFrame(Unload_frame)
        util.set_bool(DB_toggle,10,0,False)
        plc.db_write(DB_communication_number,Start_address, DB_toggle)
    #Toggle UR10e go to target
    s.robot_goto_toggle = util.get_bool(DB_toggle,10,1)
    s.wait_status = util.get_bool(DB_toggle,10,2)
    if s.robot_goto_toggle == True:
        Go_to_target()
        util.set_bool(DB_toggle,10,1,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
    s.robot_home = util.get_bool(DB_toggle,10,3)
```

Slika 62. Funkcija *Toggle_mechanisms()* - 2.dio

```
    if s.robot_home == True:
        robot.MoveJ(Home_position)
        util.set_bool(DB_toggle,10,3,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
    s.robot_goto_toggle_L = util.get_bool(DB_toggle,10,4)
    if s.robot_goto_toggle_L == True:
        Go_to_target_L()
        util.set_bool(DB_toggle,10,4,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)

    Toggle_Frames()

    ##Vision control
    s.start_vision_control = util.get_bool(DB_toggle,10,5)
    if s.start_vision_control==True:
        s.vision_control_status = proccess_photo()
        util.set_bool(DB_toggle,10,5,False)
        util.set_int(DB_toggle,12,s.vision_control_status)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
        Reset_wait_status()


    return
```

Slika 63. Funkcija *Toggle_mechanisms()* - 3.dio

Funkcija *Toggle_frames()* služi za aktivaciju programa robota za kretanje u prostoru stanice, poput programa za kretanje od kolica s dijelovima do stroja za varenje matica *Load_Dalex*, a prema podatcima pročitanim s PLC-a, dok funkcija *Write_target_list()* upisuje koordinate učitanih dijelova za varenje u data block PLC-a u kojem se nalaze ovi podatci.

```
def Toggle_Frames():
    DB_communication_number = 1
    Start_address = 0
    Size = 11
    DB_toggle = plc.db_read(DB_communication_number,Start_address,Size)
     #Toggling curve follow projects
    Load_Dalex_toggle = util.get_bool(DB_toggle,3,1)
    Dalex_Load_toggle = util.get_bool(DB_toggle,3,2)
    Dalex_Camera_toggle = util.get_bool(DB_toggle,3,3)
    Camera_Dalex_toggle = util.get_bool(DB_toggle,3,4)
    Camera_Unload_toggle = util.get_bool(DB_toggle,3,5)
    Unload_Camera_toggle = util.get_bool(DB_toggle,3,6)
    if Load_Dalex_toggle == True:
        Load_Dalex.RunProgram()
        util.set_bool(DB_toggle,3,1,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
    elif Dalex_Load_toggle == True:
        Dalex_Load.RunProgram()
        util.set_bool(DB_toggle,3,2,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
    elif Dalex_Camera_toggle == True:
        Dalex_Camera.RunProgram()
        util.set_bool(DB_toggle,3,3,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
    elif Camera_Dalex_toggle == True:
        Camera_Dalex.RunProgram()
        util.set_bool(DB_toggle,3,4,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
    elif Camera_Unload_toggle == True:
        Camera_Unload.RunProgram()
        util.set_bool(DB_toggle,3,5,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)
    elif Unload_Camera_toggle == True:
        Unload_Camera.RunProgram()
        util.set_bool(DB_toggle,3,6,False)
        plc.db_write(DB_communication_number,Start_address,DB_toggle)

def Write_target_list(target_unload, paleta, rows, columns, paleta_q, paleta_pick):
    DB_number = 3
    Start_address = 0
    Size = 7200
    DB_target_unload = plc.db_read(DB_number,Start_address,Size)
    DB_number = 7
    Start_address = 0
    Size = 268
    DB_paleta = plc.db_read(DB_number, Start_address, Size)
    DB_target_offset = [480,144]
    for i in range(paleta):
        for j in range(rows[i]):
            for k in range(columns[i]):
                for coord in range(6):
                    util.set_real(DB_target_unload,((i*paleta_q)*24+j*DB_target_offset[s.number_of_part-1]+k*24+coord*4),float(target_unload[i][j][k][coord]))
    for i in range(paleta):
        for coord in range(6):
            util.set_real(DB_paleta,4+i*24+4*coord, float(paleta_pick[i][coord]))
    DB_number = 3
    Start_address = 0
    Size = 7200
    plc.db_write(DB_number, Start_address, DB_target_unload)
    DB_number = 7
    Start_address = 0
    Size = 268
    plc.db_write(DB_number, Start_address, DB_paleta)
```

Slika 64. Funkcije *Toggle_frames()* i *Write_target_list()*

Na kraju je izvedena i funkcija za učitavanje dijelova za varenje u stanicu skupa sa potrebnim brojem paleta za paletizaciju željenog broja dijelova *Insert_parts_in_station()* čije su ulazne varijable broj dijela koji varimo, te količina dijelova. Ova funkcija poziva funkciju iz skripte *Parts_load* pod nazivom *Load_Parts()* koja u stanicu učitava 3D modele dijelova i paleta.

```
def Insert_parts_in_station(number_of_part, part_quantity):
    robot.setPoseFrame(Unload_frame)
    target_unload, paleta, rows, columns, paleta_q, paleta_pick = Load_parts(number_of_part,part_quantity)
    if number_of_part==1:
        tool = RDK.Item('gripper')
        tool_pose = tool.PoseTool()
        robot.setTool(gripper)
        gripper.setVisible(True,True)
        hole_gripper.setVisible(False,False)
    if number_of_part==2:
        tool = RDK.Item('hole_gripper')
        tool_pose = tool.PoseTool()
        robot.setTool(hole_gripper)
        hole_gripper.setVisible(True,True)
        gripper.setVisible(False,False)

    return target_unload, paleta, rows, columns, paleta_q, paleta_pick
```

Slika 65. Funkcija *Insert_parts_in_station()*

Na posljetku program stavljamo u beskonačnu petlju koja neprekidno izvršava svu komunikaciju s PLC-om.

```
while(True):
    s()
    Comm()
    ActiveFrame = Toggle_mechanisms()
```

Slika 66. Beskonačna petlja programa

8. PLC program

PLC program koji se nalazi kao što je rečeno na S7-1500 Siemensovom PLC-u izvršava čitavo upravljanje robotskom stanicom za varenje matica. Program je izveden najvećim dijelom u SCL programskom jeziku dok je glavna struktura programa, njegova beskonačna petlja *Main* pisana u LAD dijagramu. Program se sastoji od dva glavna dijela:

1. Automatski režim rada
2. Ručni režim rada

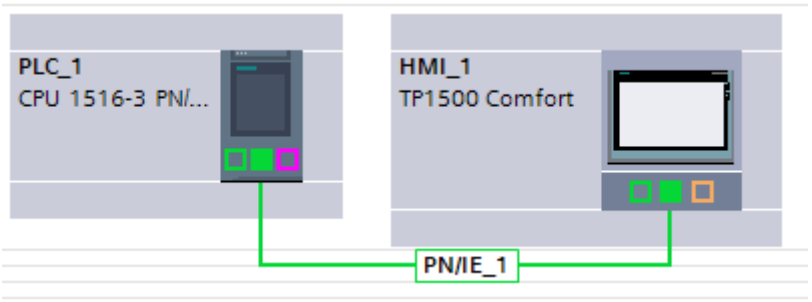
U automatskom režimu rada po učitavanju svih potrebnih dijelova u stanicu, odvija se automatski proces varenja što se sastoji od:

1. Izuzimanje dijela s paleta
2. Prinos dijela u stroj za varenje
3. Upucavanje matice na vrh donje elektrode stroja za varenje
4. Aktivacija ciklusa varenja
5. Prinos gotovog komada na stanicu za vizijsku kontrolu
    a. Ostavljanje komada u gajbu s gotovim dijelovima ili
    b. Korekcija varenja dijela u stroju za varenje te ponovna kontrola
6. Po pražnjenju palete dijelova, robot izuzima čitavu paletu s kolica s dijelovima te ju odlaže na za to predviđeno mjesto

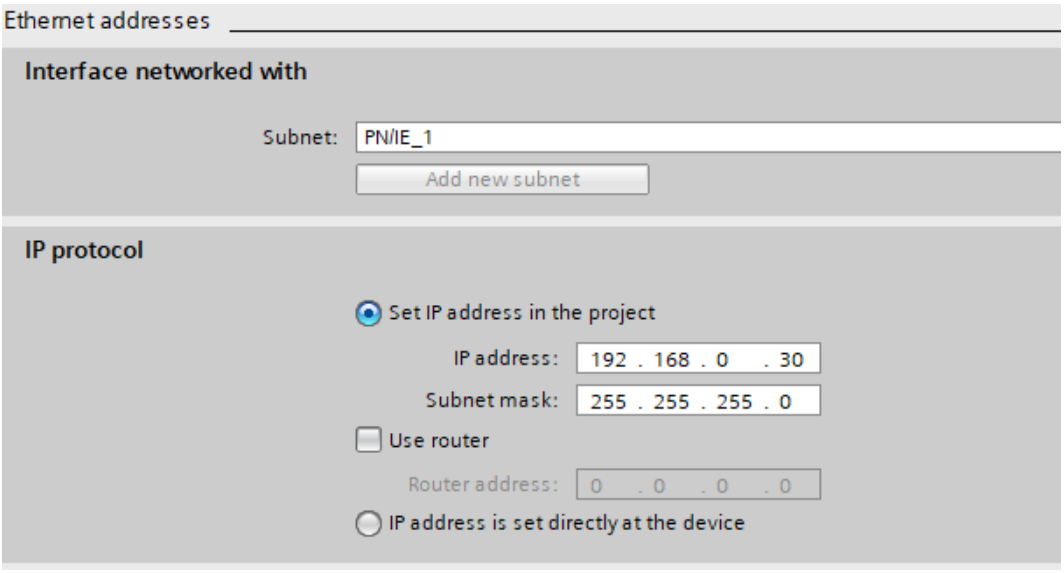
U ručnom režimu rada možemo upravljati stanicom na bilo kakav željeni način, voziti robot u željene pozicije uz pomoć korisničkog sučelja, aktivirati/deaktivirati pojedine mehanizme i tako dalje.

8.1. Konfiguracija komunikacijske mreže PLC

S obzirom da PLC komunicira s Python skriptom te s HMI panelom potrebno je definirati izgled mreže kao i IP adrese PLC-a i HMI-a

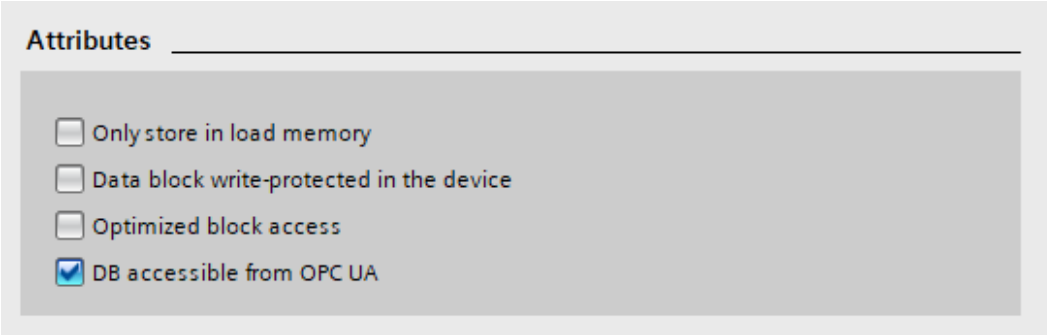


Slika 67. Prikaz PROFInet komunikacije PLC-HMI u TIA Portal-u



Slika 68. Adresne postavke PLC-a

S obzirom da za komunikaciju PLC – Python koristimo *snap7* biblioteku koja funkcionira tako da na točno određene pozicije u Data blocku PLC-a upisuje podatke potrebno je za komunikacijske DB-ove ugasiti optimizaciju memorije kako bi se spriječilo pomicanje varijabli unutar DB-a.



Slika 69. *Optimized block acess*

Access type	Standard block access	Optimized block access
Symbolic	Yes	Yes
Indexed (fields)	Yes	Yes
Slice access	Yes	Yes
AT instruction	Yes	No
Absolute	Yes	No
Indirect (ANY)	Yes	No
Indirect (pointer version)	Yes	Only with symbolic notation
Download without reinitialization	No	Yes (for S7-1200 with firmware V4.0 and higher)

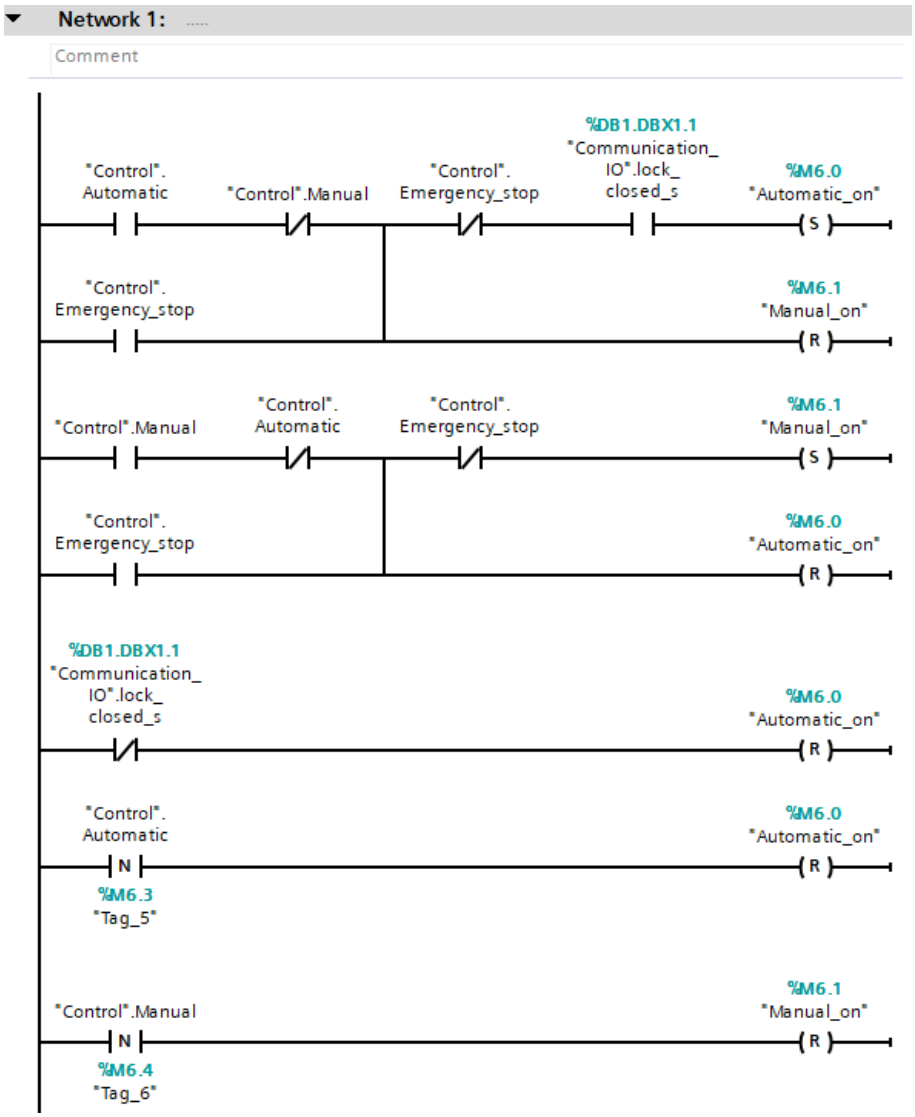
Slika 70. Razlika između optimiziranih i ne optimiziranih DB

8.2.Automatski režim rada

Za aktivaciju automatskog režima rada stanice mora biti zadovoljeno nekoliko uvjeta, bez kojih nije moguće pokrenuti automatski režim a to su:

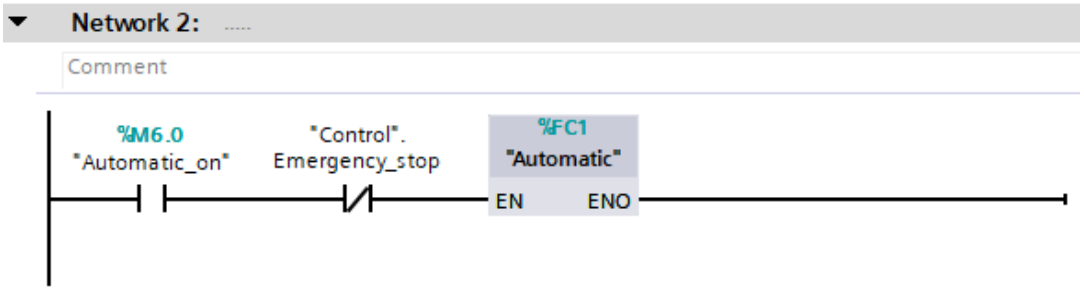
1. Gljiva za isklop u nuždi nije pritisnuta
2. Nije aktiviran ručni režim rada
3. Kolica s dijelovima zaključana su putem mehanizma za zaključavanje

Ako su ova tri uvjeta zadovoljena u *Main* petlji programa logički bit *„Control".Automatic* unutar *Control DB*-a se postavlja u logičku jedinicu. Putem ovog logičkog operatora pali se i gasi automatski režim rada, odnosno ovaj bit 'setira' ili 'resetira' memorijsku varijablu *Automatic_on.*



Slika 71. *Main* petlja, Network 1

Unutar *Networka 1* također se slični uvjeti provjeravaju za aktivaciju ručnog režima rada. Memorijska varijabla *Automatic_on* u networku 2 poziva funkciju automatskog režima rada *Automatic*.



Slika 72. Network 2

Funkcija *Automatic* pisana je SCL programskim jezikom, najviše radi jednostavnosti korištenja funkcija i naredbi vezanih za pojedina uspoređivanja, ili matematičke operacije, a potom i preglednosti samog programa koji bi u nekom drugom obliku vjerojatno bio zamršen i nečitljiv.

Program se izvodi unutar *IF-THEN* uvjeta koji provjerava jesu li svi dijelovi koju su ubačeni u stanicu na kolicima obrađeni. Ukoliko nisu izvodi se program unutar tog uvjeta.

Program je pisan u obliku serije *Case*-ova, koji predstavljaju svaki pojedini korak koji se izvodi u stanici. Ovaj način odabran je radi jednostavnog praćenja i upravljanja, jer dok nisu zadovoljeni uvjeti prethodnog koraka/*Case*-a, slijedeći korak ne može krenuti.

Također ovaj način programiranja omogućuje nam i jednostavnu implementaciju provjere vremenskog ciklusa trajanja jednog koraka, uz pomoć *Timera* možemo provjeravati jeli se korak izvršio u zadanom vremenskom periodu, te ukoliko nije javlja se greška te se prekida automatski režim rada stanice.

Automatski režim rada dakle sastoji se 37 koraka, uz još 11 koraka za odlaganje praznih paleta, te opcionalne korake za korekciju varenja. Taksativno ćemo navesti svaki korak automatskog režima rada, uz nekoliko navedenih primjera koda.

Automatski režim varenja sastoji se od slijedećih koraka, uz pretpostavku da je dodavač spreman za rad, a savitljiva vinil cijev puna matica tako da dodavač nema potrebu upucavati veliki broj matica kako bi napunio vodilice:

1. Izvlačenje cilindra za upucavanje
2. Uvlačenje cilindra za blokiranje staze dodavača
3. Izvlačenje cilindra za blokiranje glave dodavača
4. Izvlačenje cilindra za blokiranje staze dodavača (dodavač spreman za upucavanje)

```
IF "Control".current_part <= "Communication_IO".part_quantity THEN

    CASE "Control".case_number OF
        1:  //Cilindar za upucavanje van
            "Control".Timer_on := TRUE;
            "Communication_IO".push_toggle := TRUE;
            IF "Communication_IO".push_out_s = TRUE THEN
                "Control".case_number := 2;
                "Control".Timer_on := FALSE;
            END_IF;
        2:  //hold in
            "Communication_IO".hold_toggle := False;
            "Control".Timer_on := TRUE;
            IF "Communication_IO".hold_out_s = FALSE THEN
                "Control".case_number := 3;
                "Control".Timer_on := FALSE;
            END_IF;
        3:  //block out
            "Communication_IO".block_toggle := True;
            "Control".Timer_on := TRUE;
            IF "Communication_IO".block_in_s = FALSE THEN
                "Control".case_number := 4;
                "Control".Timer_on := FALSE;
```

Slika 73. Prva tri koraka automatskog režima

5. Aktivacija programa slijeđenja putanje od Dalex-a do kolica s dijelovima
6. Prilazak poziciji za izuzimanje dijelova (odstupanje po z-osi)
7. Prilazak točki izuzimanja komada iz palete
8. Aktivacija robotske hvataljke
9. Vraćanje u poziciju prilaska izuzimanja komada iz palete
10. Aktivacija programa slijeđenja putanje od kolica s dijelovima do Dalexa
11. Prilazak točki varenja prve matice
12. Spuštanje u točku varenja prve matice
13. Izvlačenje i uvlačenje cilindra dodavača (matica na vrhu donje elektrode)
14. Ponovno pokretanje ciklusa dodavača (koraci od 1-4)
15. Aktivacija ciklusa varenja Dalex-a
16. Vraćanje u točku prilaska za varenje prve matice
17. Ponavljanje koraka od 11-16 za drugu maticu
18. Pokretanje programa slijeđenja putanje od Dalexa do stanice za vizijsku kontrolu dijela
19. Pozicioniranje ispod kamere
20. Vizijski pregled dijela prema kojem se odlučuje o vraćanju na korak 11 ili 17, odnosno ukoliko je komad u redu odlazak prema gajbi s gotovim dijelovima
21. Aktivacija programa slijeđenja putanje od stanice za vizijsku kontrolu do gajbe s gotovim dijelivima
22. Deaktivacija robotske hvataljke

23. Vraćanje nazad pred stroj za varenje (Programi slijeđenja od gajbe s gotovim dijelovima do vizijske stanice pa potom do Dalex-a)
24. Ukoliko na paleti ima još dijelova vraćamo se na korak 1

Na slici ispod vidimo primjer funkcionalnosti gibanja robota. U koraku 20 robotu se šalju koordinate prilaska točci varenja matice 1, te se aktivira naredba za *Joint* pomak linijom *„Communication_IO".robot_goto_toggle = TRUE;* te se broj koraka prebacuje na 21. Kako vidimo u koraku 21 njegove naredbe neće se izvršiti sve dok funkcija *Compare_pos,* za uspoređivanje trenutne koordinate robota sa zadanom koordinatom, ne vrati pozitivan rezultat.

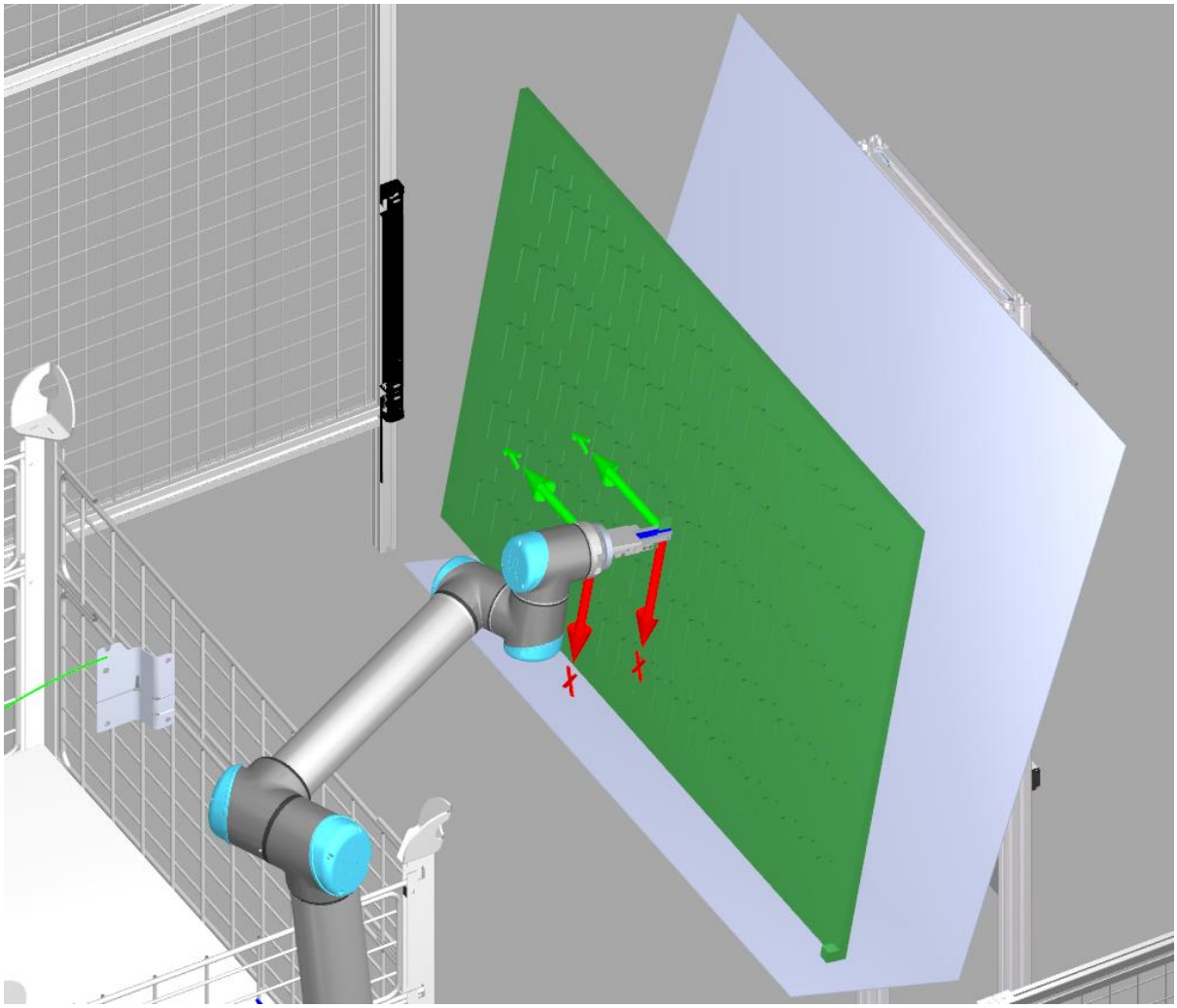
```
20://Back to approach weld 1
    "Control".Timer_on := TRUE;
    IF "Communication_IO".dalex_home_s = TRUE THEN
        "Control".robot_target := "Weld_targets".Weld_approach_1["Communication_IO".number_of_part - 1];
        "Robot_coordinates_IO".Coordinates_out := "Control".robot_target;
        "Communication_IO".robot_goto_toggle := TRUE;
        "Control".case_number := 21;
        "Control".Timer_on := FALSE;
    END_IF;
21://Approach weld target 2
    "Compare_pos"(Compare_1 := "Control".robot_target,
                  Compare_2 := "Robot_coordinates_IO".Coordinates_in,
                  Compare_status => #Comp_status);
    "Control".Timer_on := TRUE;
    IF #Comp_status = TRUE THEN
        #Comp_status := FALSE;
        "Control".robot_target := "Weld_targets".Weld_approach_2["Communication_IO".number_of_part - 1];
        "Robot_coordinates_IO".Coordinates_out := "Control".robot_target;
        "Communication_IO".robot_goto_toggle := TRUE;
        "Control".case_number := 22;
        "Control".Timer_on := FALSE;
    END_IF;
22://Weld target 2
    "Compare_pos"(Compare_1 := "Control".robot_target,
                  Compare_2 := "Robot_coordinates_IO".Coordinates_in,
                  Compare_status => #Comp_status);
    "Control".Timer_on := TRUE;
    IF #Comp_status = TRUE THEN
        #Comp_status := FALSE;
        "Control".robot_target := "Weld_targets".Weld_target_2["Communication_IO".number_of_part - 1];
        "Robot_coordinates_IO".Coordinates_out := "Control".robot_target;
        "Communication_IO".robot_goto_toggle := TRUE;
        "Control".case_number := 23;
        "Control".Timer_on := FALSE;
    END_IF;
```

Slika 74. Koraci robota

Po završetku pražnjenja palete ulazimo u dio programa namijenjen za micanje palete na mjesto za odlaganje praznih paleta. Tu također imamo nekoliko koraka koje izvršava robot a koji se sastoje od:

1. Prilazak točki izuzimanja palete
2. Odlazak u točku izuzimanja palete, u kojoj robot hvata paletu za izbočenje palete napravljeno za tu svrhu

3. Podizanje palete iznad kolica
4. Prilazak točki za spremanje paleta
5. Odlazak u točku za spremanje i otpuštanje robotske hvataljke, pri čemu paleta pada na policu
6. Vraćanje robota u početnu poziciju – *Home_position*



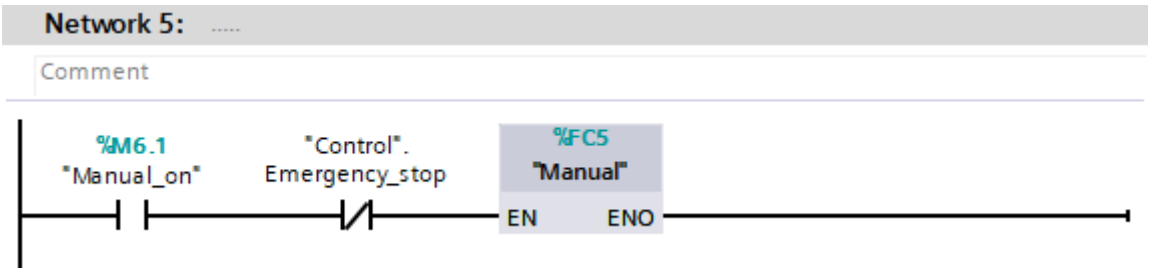
Slika 75. Odlaganje palete

8.3. Ručni režim rada

Kako je prikazano na slici 71. ručni režim rada također se može aktivirati uz zadovoljenje slijedećih uvjeta:

1. Gljiva za isklop u nuždi nije pritisnuta
2. Nije aktiviran automatski režim rada

Pri aktivaciji memorijskog bit-a *Manual_on*, isti bit poziva funkciju *Manual*



Slika 76. Network 5

Ova funkcija također je pisana u SCL programskom jeziku, te nam ona omogućuje ručno upravljanje mehanizmima stanice te robotom. U principu ona prosljeđuje ulaze sa HMI korisničkog sučelja direktno prema mehanizmima i robotu.

```
//Manually control mechanisms from HMI

"Communication_IO".dalex_toggle := "Manual_DB".Dalex;
"Communication_IO".dodavac_toggle := "Manual_DB".Dodavac;
"Communication_IO".block_toggle := "Manual_DB".Block;
"Communication_IO".hold_toggle := "Manual_DB".Hold;
"Communication_IO".push_toggle := "Manual_DB".Push;
"Communication_IO".lock_toggle := "Manual_DB".Lock;
"Communication_IO".gripper_toggle := "Manual_DB".gripper_toggle;
```

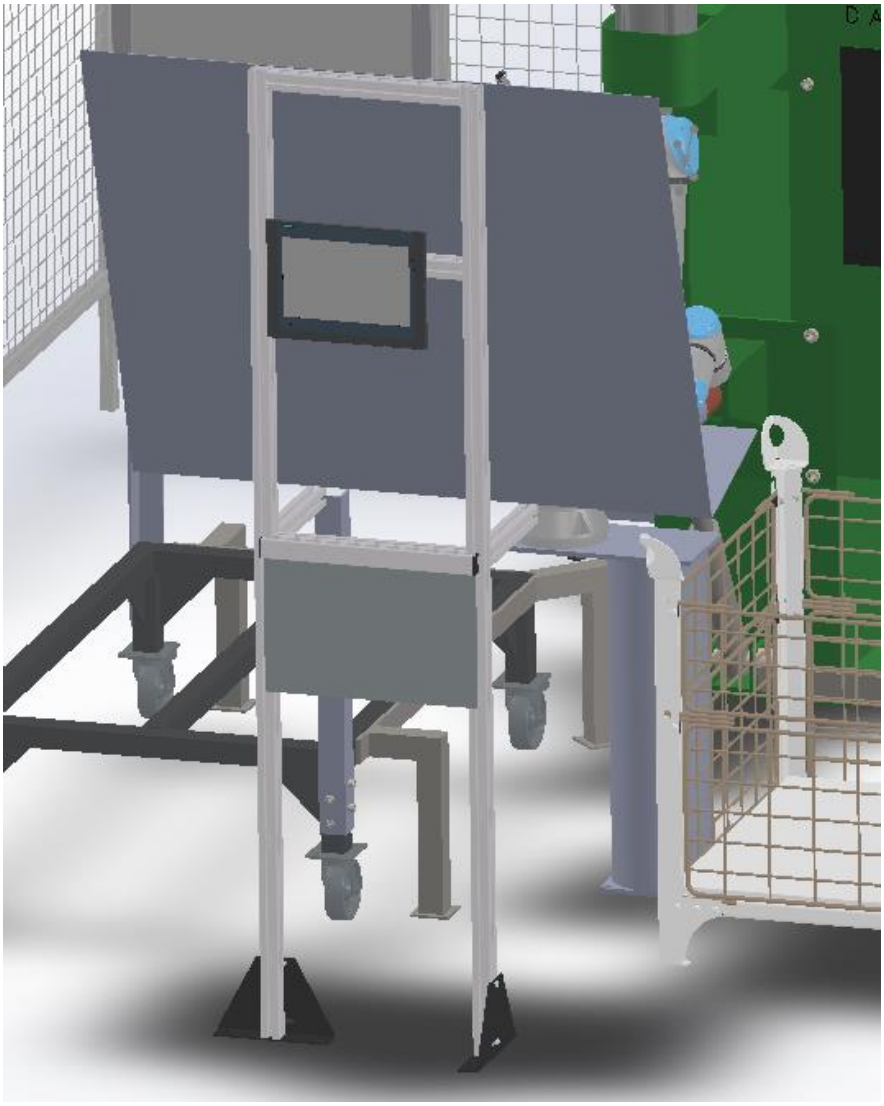
Slika 77. Ručna aktivacija mehanizama unutar *Manual* funkcije

Ova funkcija omogućuje nam pomicanje robota duž njegovih osi, slanje robota u ručno upisanu ciljnu točku i tako dalje, više ćemo vidjeti u poglavlju HMI korisničkog sučelja.

9. HMI KORISNIČKO SUČELJE

Kako je ranije navedeno za ovu namjenu koristimo Siemensov HMI panel TP1500 Comfort. Radi se o 15" panelu s ekranom osjetljivim na dodir, što nam omogućuje veliku slobodu pri dizajniranju sučelja obzirom da prostora za pojedine funkcije ima više nego dovoljno.

Simulacija HMI sučelja odvija se na simulacijskom računalu unutar WinCC RT Advanced programa koji simulira sve funkcije HMI panela, uključujući i samu komunikaciju sa PLC-om



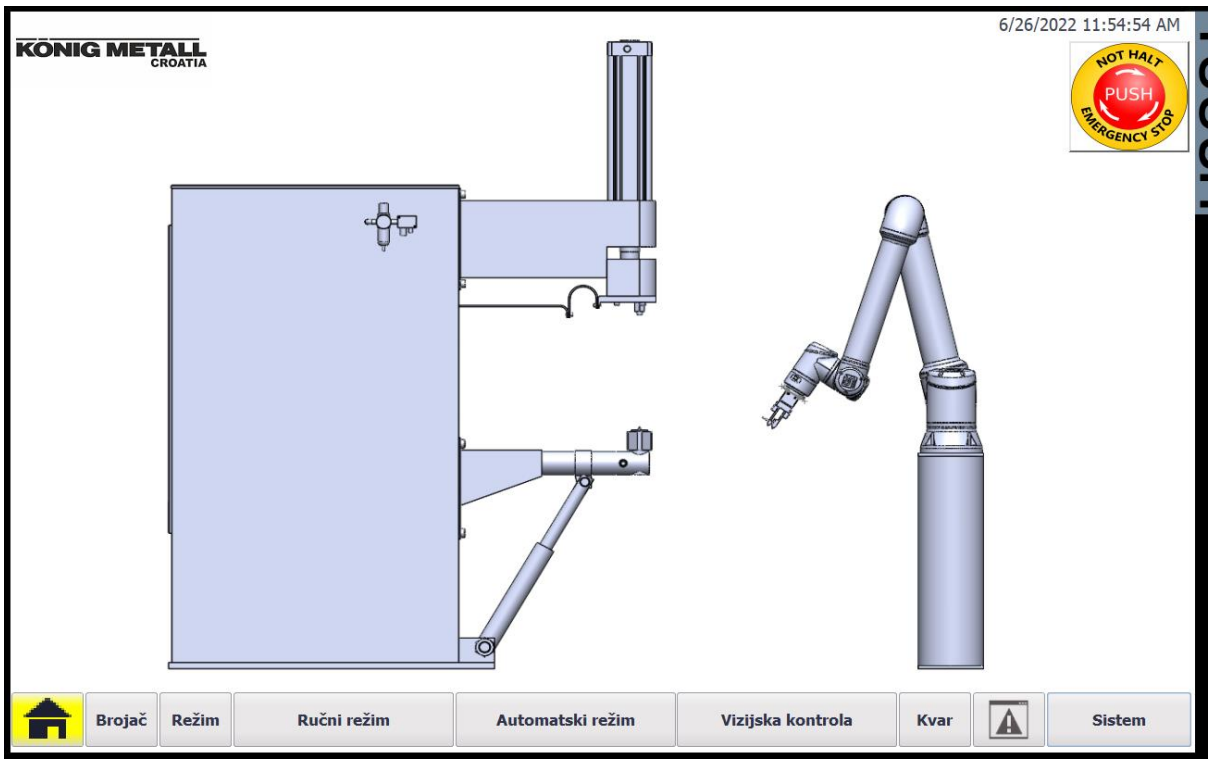
Slika 78. HMI panel na robotskoj stanici za varenje matica

HMI sučelje sastoji se od 10 ekrana/prikaza koji nam omogućuju upravljanje automatskom stanicom za varenje matica:

1. Početni ekran
2. Ekran brojača komada

3. Odabir režima rada stanice(automatski/ručno)
4. Ekran ručnog režima rada
    a. Ekran za upravljanje Dalexom i dodavačem
    b. Ekran za upravljanje vibrododavačem
    c. Ekran za upravljanje UR10e robotom
5. Ekran automatskog režima rada
6. Ekran za prikaz rezultata vizijske kontrole dijelova
7. Ekran za simulaciju kvara na stanici

Krenuti ćemo redom kroz svaki pojedini ekran HMI korisničkog sučelja. Početni ekran ne sadrži u principu nikakve specijalne funkcije, osim gumba gljive za isklop u nuždi.Ovaj ekran pojavljuje se pri paljenju stanice.



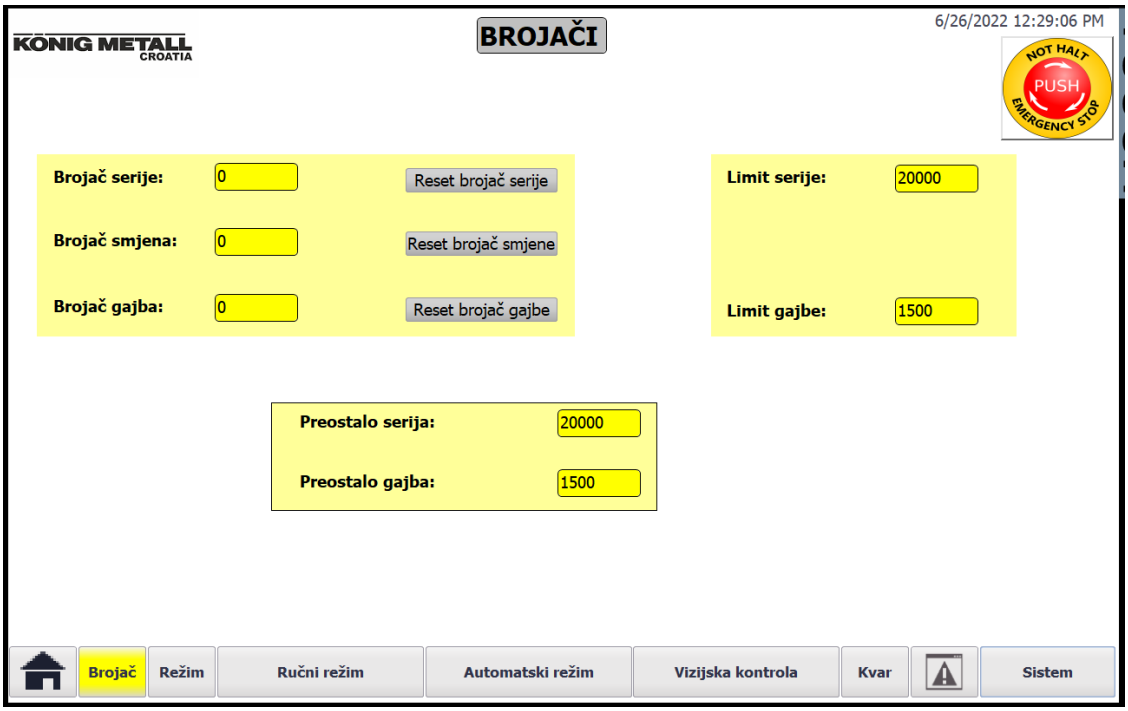
Slika 79. Početni ekran

Ekran brojača sadrži informacije o stanjima ukupno tri brojača:

1. Brojač serije – koliko je dijelova potrebno proizvesti u seriji
2. Brojač smjene- koliko je dijelova proizvedeno u smjeni
3. Brojač gajbe – koliko je dijelova u gajbi s gotovim dijelovima

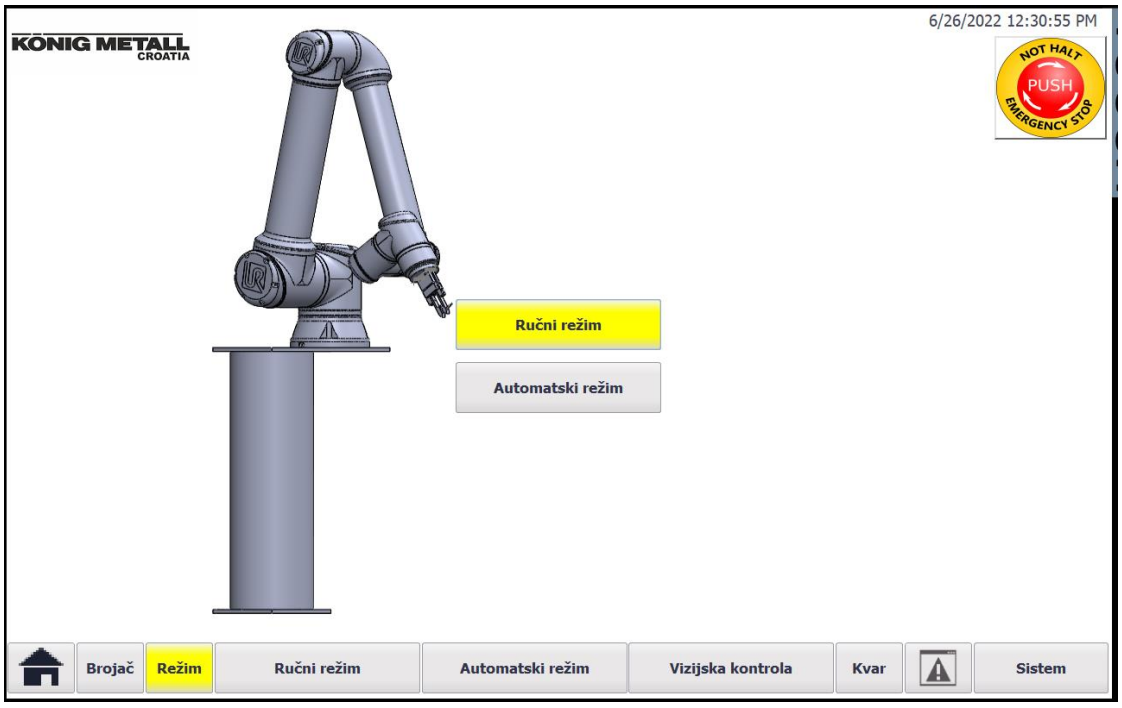
Također možemo vidjeti i informacije o preostalom broju dijelova koje je potrebno proizvesti unutar serije, ili da se napuni gajba s gotovim dijelovima. Na desnoj strani ekrana

podešavamo dva ograničenja brojača, to su ograničenje brojača serije(koliko dijelova treba proizvesti u seriji) i limit brojača gajbe(koliko dijelova se pakira u jednoj gajbi – propisano radnim nalogom)



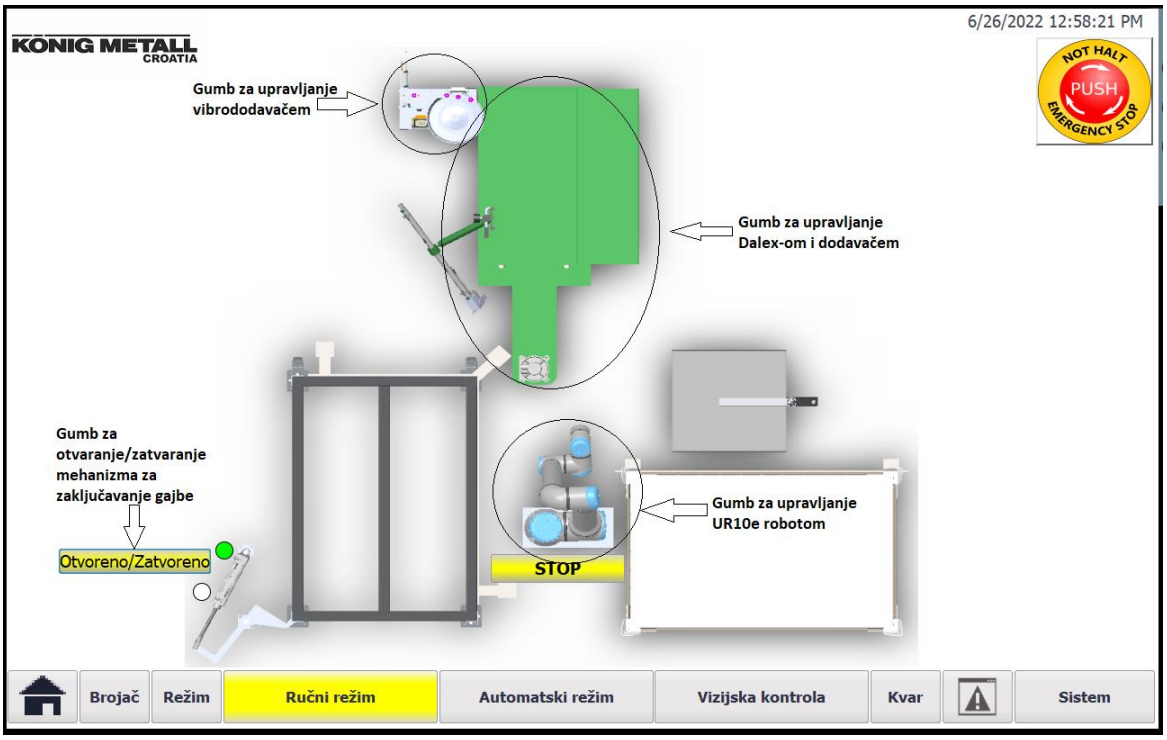
Slika 80. Ekran brojača

Slijedeći ekran jest ekran za odabir režima rada na kojem se u principu nalaze dva gumba za odabir režima rada – automatski ili ručno. Ovisno o odabiru biti će aktivirane pripadajuće funkcije stanice.



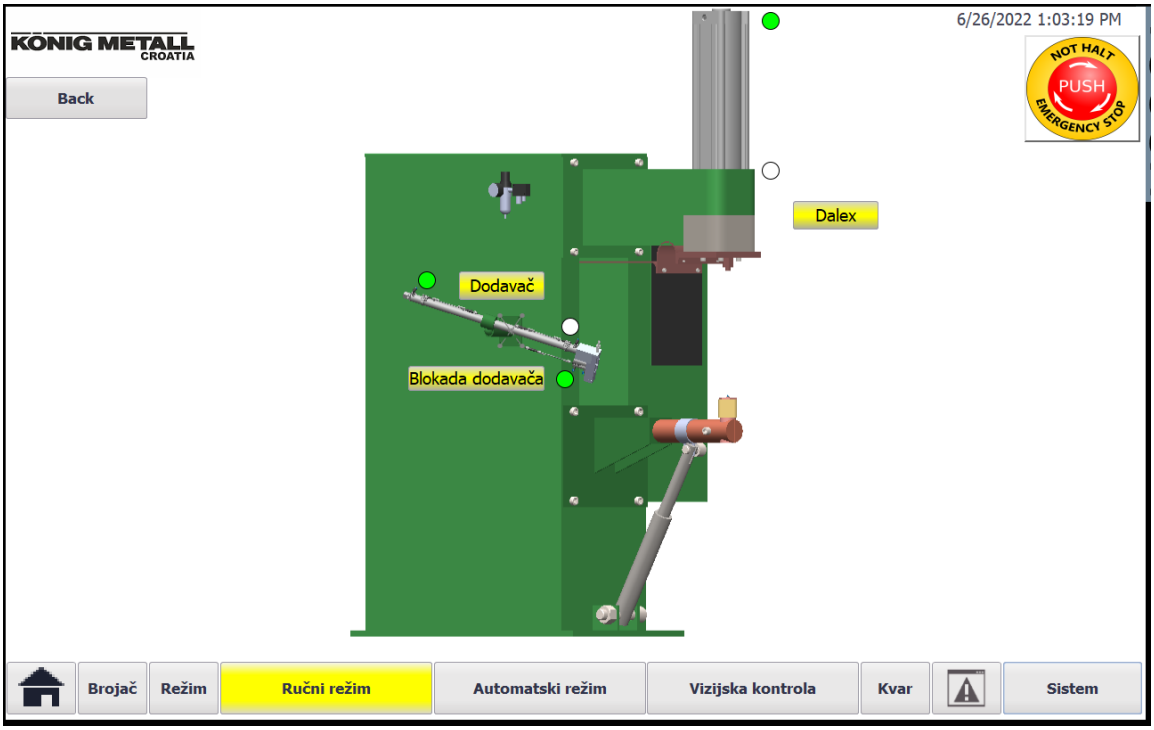
Slika 81. Odabir režima rada

Obzirom da je odabran ručni režim rada, ulaskom u ekran ručnog režima možemo vidjeti topografski prikaz automatske stanice za varenje. Klikom na pojedini stroj/uređaj otvaramo ekran za ručno upravljanje tim strojem/uređajem



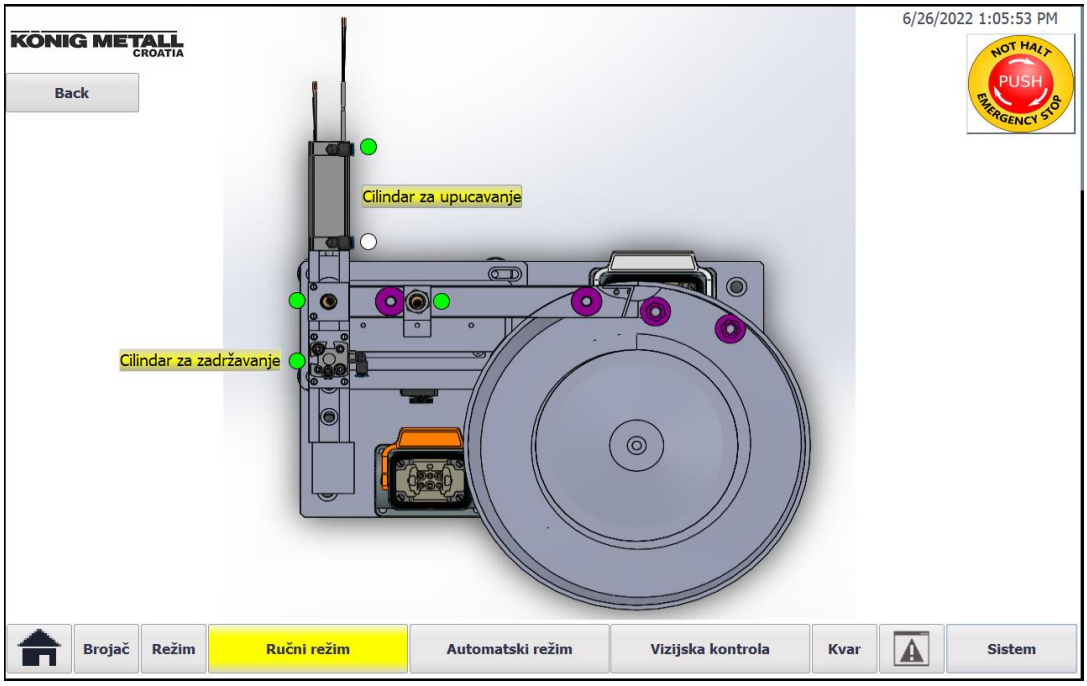
Slika 82. Ekran ručnog režima rada

Odaberemo li Dalex otvara se novi ekran za upravljanje Dalex strojem za varenje matica, cilindrom dodavača, i cilindrom za blokiranje glave dodavača.



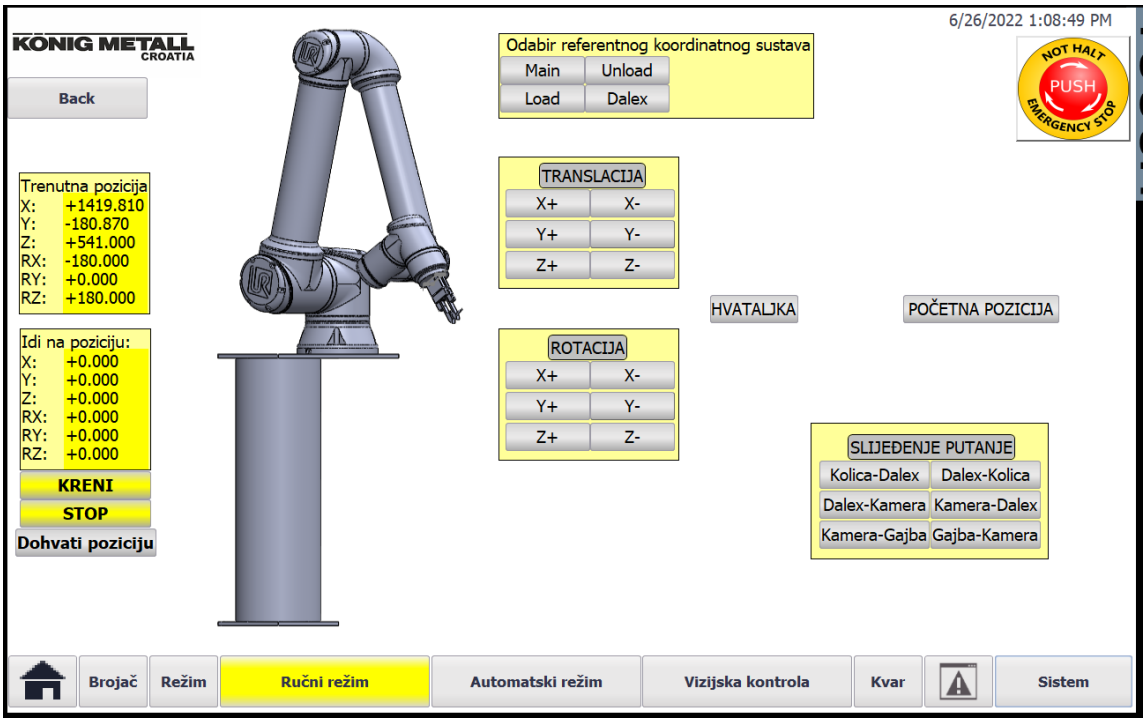
Slika 83. Ekran Dalex-dodavač

Odabirom vibrododavača otvara se ekran za upravljanje cilindrom za upucavanje, te cilindrom za blokadu staze dodavača(cilindar za zadržavanje).



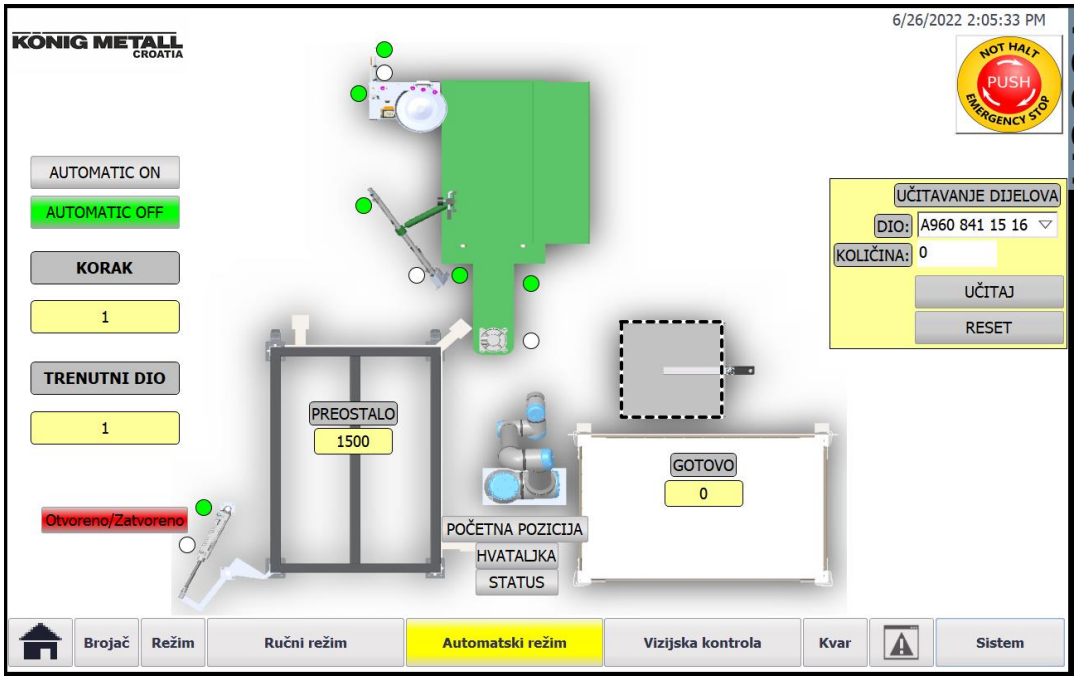
Slika 84. Ekran vibrododavača

Na posljetku u ručnom režimu rada imamo ekran za upravljanje UR10e robotom gdje imamo informacije o poziciji robota, polje za upisivanje ciljne točke robota, te gumbe za pokretanje putanje, ručno voženje robota po osima, odabir referentnog koordinatnog sustava, aktivaciju programa slijeđenja putanja, aktivacija/deaktivacija robotske hvataljke itd.



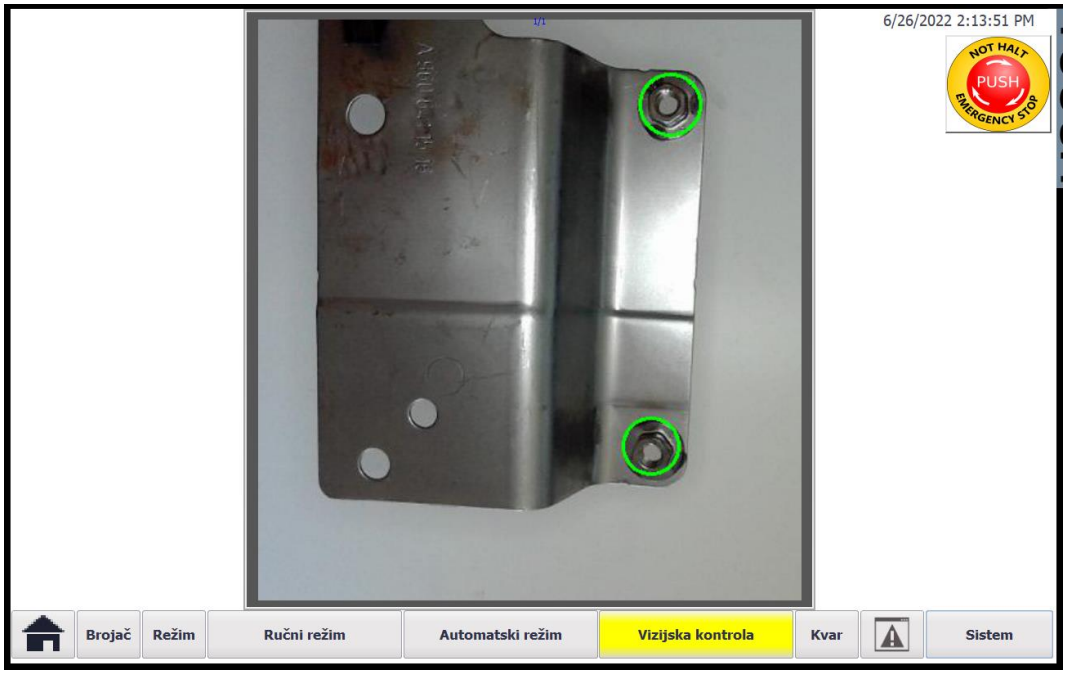
Slika 85. Ekran UR10e robota

Odabirom automatskog režima rada na ekranu režima, omogućuju se funkcije automatskog režima. Na ekranu automatskog režima imamo prikaz trenutnog stanja proizvedenih dijelova, informaciju o koraku koji se trenutno izvodi, te polje za učitavanje dijelova u stanici.



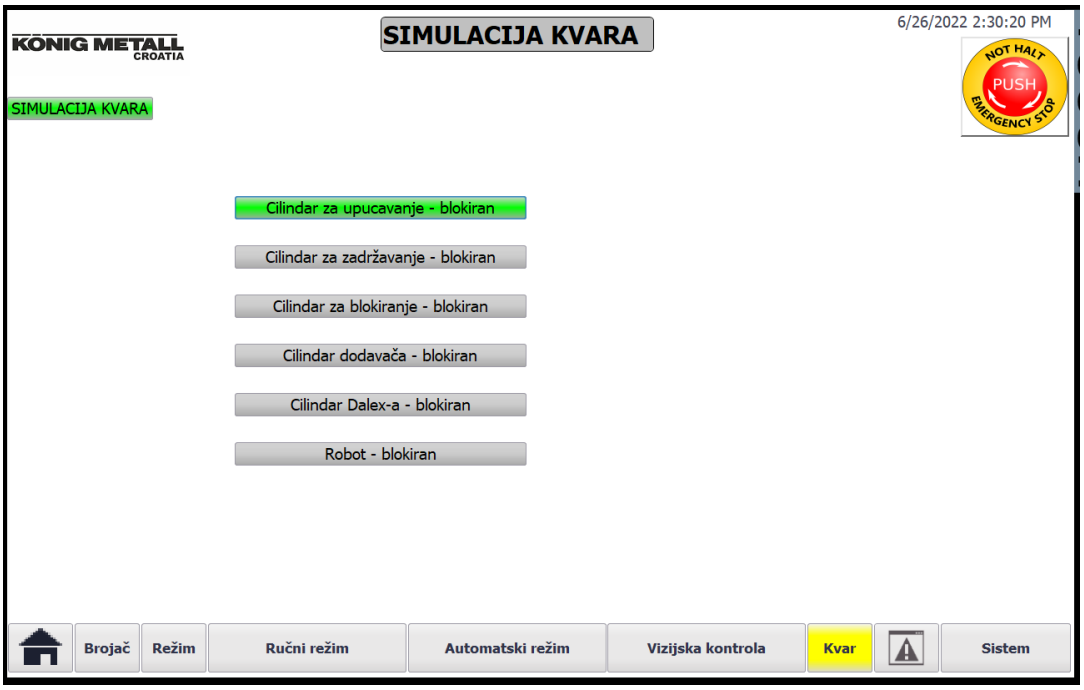
Slika 86. Ekran automatskog režima rada

Ekran vizijske kontrole služi za prikaz detekcije matice na slici s kamere. Prilikom svake vizijske kontrole kamera uzima jedan frame, koji se potom obradi i prikaže na ekranu HMI sučelja.



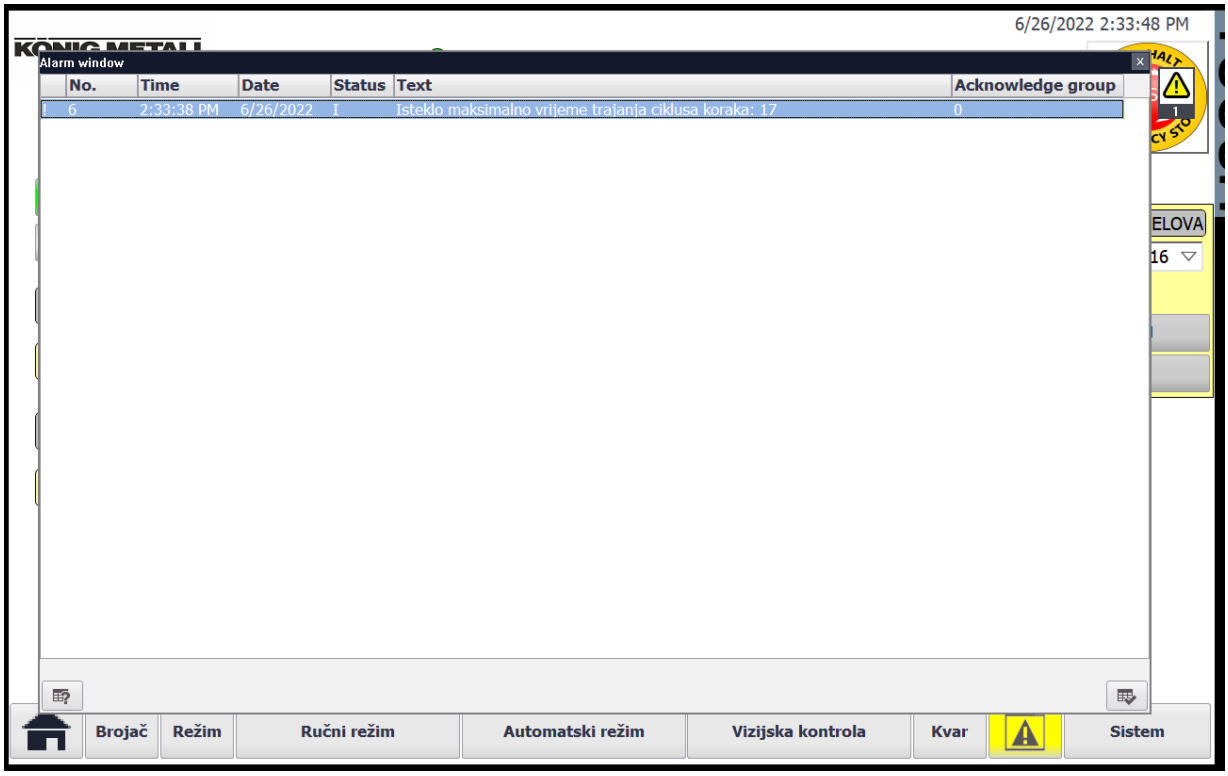
Slika 87. Ekran vizijske kontrole

Na kraju nam ostaje ekran simulacije kvarova stanice, gdje možemo blokirati pojedine mehanizme stanice kako bi simulirali primjerice kvar senzora i slično, nakon čega se javlja greška rada stanice.



Slika 88. Simulacija kvara

Na slici iznad vidimo da je blokiran cilindar za upucavanje vibrododavača, nakon 30 sekundi(maksimalno trajanje ciklusa) javlja se slijedeća greška:



Slika 89. Greška koraka 17

10. STANICA ZA VIZIJSKU KONTROLU

Stanica za vizijsku kontrolu dijelova ukomponirana je u automatskoj stanici za varenje matica. Po završetku varenja dijelova, robot prilazi pred stanicu za vizijsku kontrolu, te se potom pozicionira ispod kamere, a iznad stola stanice za vizijsku kontrolu, koji je bijele boje kako bi se ostvario čim bolji kontrast između kontrolnog komada i pozadine.

U svrhu izrade i testiranja programa za vizijsku kontrolu dijelova iskorištena je web-kamera, Razer Kiyo koja je odabrana iz nekoliko razloga. Kamera ima automatski fokus, komunikaciju preko USB 2.0, 4 megapixelnu rezoluciju(koja je u našem slučaju smanjena na 640*480). Ono što kameru izdvaja od drugih jest vidno polje *FOV* od 81.6°, što je nešto uže vidno polje. Također kamera ima ugrađeno prstenasto osvjetljenje koje može pomoći pri malo lošijim uvjetima svjetla.

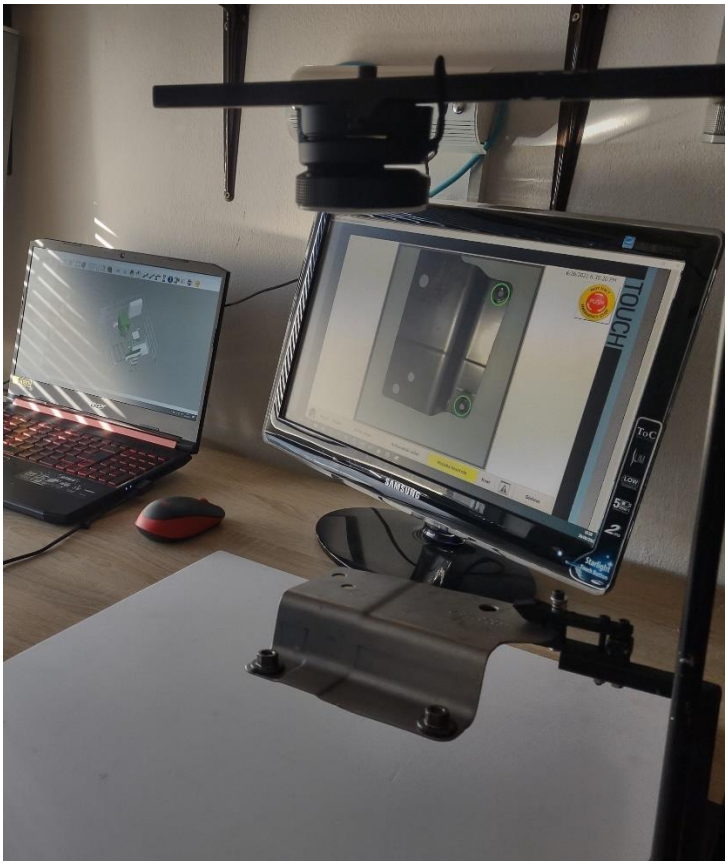
Slika 90. Razer Kiyo kamera [15]

Program vizijske kontrole stanica rađen je unutar Python programskog koda, te su korištene biblioteke poput *OpenCV* biblioteke. Radi se o biblioteci usmjerenoj na računalni vid u stvarnom vremenu, originalno razvijenom od strane Intel-a. Također korištena je i *Pillow* biblioteka koja nam nudi dodatne mogućnosti procesiranja slike.

Svrha stanice za vizijsku kontrolu dijelova jest pregled zavarenog komada, i to u svrhu provjere prisutnosti matice, i provjere pozicije matice, čime možemo doći do zaključka treba li komad ponovno variti, ili je komad u redu te ga se može ostaviti u gajbu s gotovim dijelovima.

U svrhu testiranja programa za vizijsku kontrolu dijelova izrađen je i postav za kameru i dijelove koji simuliraju konstruiranu stanicu za vizijsku kontrolu. Postav se sastoji od nosača kamere i nosača dijelova na kojem se nalaze prsti robotske hvataljke isprintani 3D

printerom, te su sastavljeni na način da jednostavno možemo postaviti i mijenjati komade u robotskoj hvataljci.



Slika 91. Postav za vizijsku kontrolu

Program vizijske kontrole kreće najprije sa aktivacijom kamere i učitavanjem slike s kamere koju potom spremamo u datoteku kako bi jednostavnije i sigurnije mogli vršiti procesiranje same slike.

Čitav program nalazi se unutar Python skripte *Vision_control.py* koji je uvezen u glavnu skriptu simulacijskog programa za komunikaciju. Unutar ove skripte imamo funkciju *take_photo* uz pomoć koje možemo u određenom trenutku, kada se robot pozicionira na potrebnu ciljnu točku, slikati dijelove.

```
#Importing opencv and numpy
import cv2
import numpy as np
from PIL import Image



def take_photo():
    #Connecting to capture device
    Camera = cv2.VideoCapture(1)
    #Get a frame from device
    ret, frame = Camera.read()
    cv2.imwrite('scan\scan.png', frame)
    #Release camera
    Camera.release()
    return frame
```

Slika 92. Funkcija *take_photo()*

Na gornjoj slici vidimo da kamera u varijablu *frame* učita scenu, te se potom ona sprema u folder *scan* pod nazivom *scan.png* te je ta ista varijabla i izlazna varijabla funkcije, u slijedećoj funkciji *proccess_photo()* koja služi za segmentaciju i obradu slike pozivati ćemo gore navedenu funkciju za učitavanje slike kamere, koja se potom obrađuje.

Slika 93. Slika učitana sa kamere

```
def proccess_photo():
    top_edge=10     #Predefiniranje rubova komada na slici
    bottom_edge=10
    right_edge=80
    frame = take_photo()
    frame = frame[:,0:450]
    canny = cv2.Canny(frame, threshold1=80, threshold2=200) #Provlačenje slike kroz Canny filter
    #find left edge of image
    for i in range(430):
        if canny[250,i] == 255:
            left_edge = i;
            break
    #find right edge of image
    for i in reversed(range(430)):  #TRAŽENJE RUBOVA KOMADA NA SLICI
        if canny[250,i]==255:
            right_edge=i;
            break;
    #find top edge
    for i in range(480):
        if canny[i,right_edge-40]==255:
            top_edge=i;
            break;
    #find bottom edge
    for i in reversed(range(480)):
        if canny[i,right_edge-40]==255:
            bottom_edge=i;
            break;

    framed = frame[top_edge-10:bottom_edge+10,right_edge-80:right_edge+10]  #SEGMENTACIJA SLIKE
    frame_size = frame.shape
```

Slika 94. Funkcija *proccess_photo()* - segmentacija slike

Najprije na učitanoj slici s kamere (slika 93.) tražimo lijevi, desni, gornji i donji rub komada kako bi znali točnu lokaciju komada unutar slike. Iako robot dolazi uvijek u istu

poziciju uz propisanu ponovljivost od ±0.1*mm* ipak ovaj postupak nam pruža mogućnost djelomičnog odstupanja od točne pozicije, i daje određenu dozu robusnosti kodu. S nađenom pozicijom komada na slici, iz originalne slike se 'izrezuje' područje od interesa, a to je područje na kojem se nalaze matice, te se ono sprema u varijablu *framed.*

Slika 95. Izrezana slika s područjem od interesa

Tako dobivena slika je u RGB formatu, odnosno u boji je, što nije povoljno za daljnje procesiranje slike tako da sliku prebacujemo u monokromatski takozvani gray format, i provlačimo kroz filter koji zamućuje sliku, kako bi otklonili nesavršenosti i smetnje/šumove sa slike.

Obzirom da nam je zadatak vizijske kontrole detektirati poziciju matica koje imaju kružni obrub, odlučio sam koristiti jednostavnu *HoughCirlces* funkciju za detekciju kružnih oblika na slici. Za ovaj način sam se odlučio iz više razloga, glavni od kojih je činjenica da ova funkcija zadovoljava naše potrebe, i bez problema može detektirati ili maticu, ili ukoliko matice nema otvor na platini gdje je matica trebala biti. Također funkcija daje i dovoljno precizne rezultate, te sam odabrao jednostavno i robusno rješenje koje je lako primjenjivo i na druge vrste proizvoda.

10.1. HoughCircles funkcija

HoughCircles funkcija zasniva se na takozvanoj Hough-ovoj transformaciji koja se koristi za izolaciju značajki određenog oblika unutar slike. Obzirom da sama transformacija zahtjeva da je tražena značajka iskazana u nekom parametriziranom obliku, klasična Hough-ova transformacija se naj češće koristi za detekciju standardnih krivulja poput linija, kružnica elipsa i slično. Generalizirana Hough-ova transformacija može se primijeniti u slučajevima kada jednostavni analitički opis značajke nije moguć, no obzirom na vrlo zahtjevnu računsku kompleksnost ipak ćemo se zadržati na osnovnoj formulaciji. Jedna od glavnih prednosti Hough-ove transformacije (dalje u tekstu HT) jest da je ona poprilično tolerantna vezano uz rupe/nedostatke u obrubima značajki, te je relativno otporna na šum slike.

Glavnja ideja HT-a jest transformacija prostorno raširenih uzoraka u parametarski prostor gdje se uzorci mogu prikazati u prostorno kompaktnijem obliku. Na taj način je kompleksan problem detekcije u prostoru slike sveden na jednostavniji problem detekcije maksimuma u parametarskom prostoru. Princip HT-a objasnit ćemo na primjeru detekcije ravne crte na slici.

Set kolinearnih točaka slike (x,y) prema [19] može se u analitičkom obliku zapisati izrazom:

$$y - mx - n = 0 \quad \textbf{(4)}$$

Gdje su

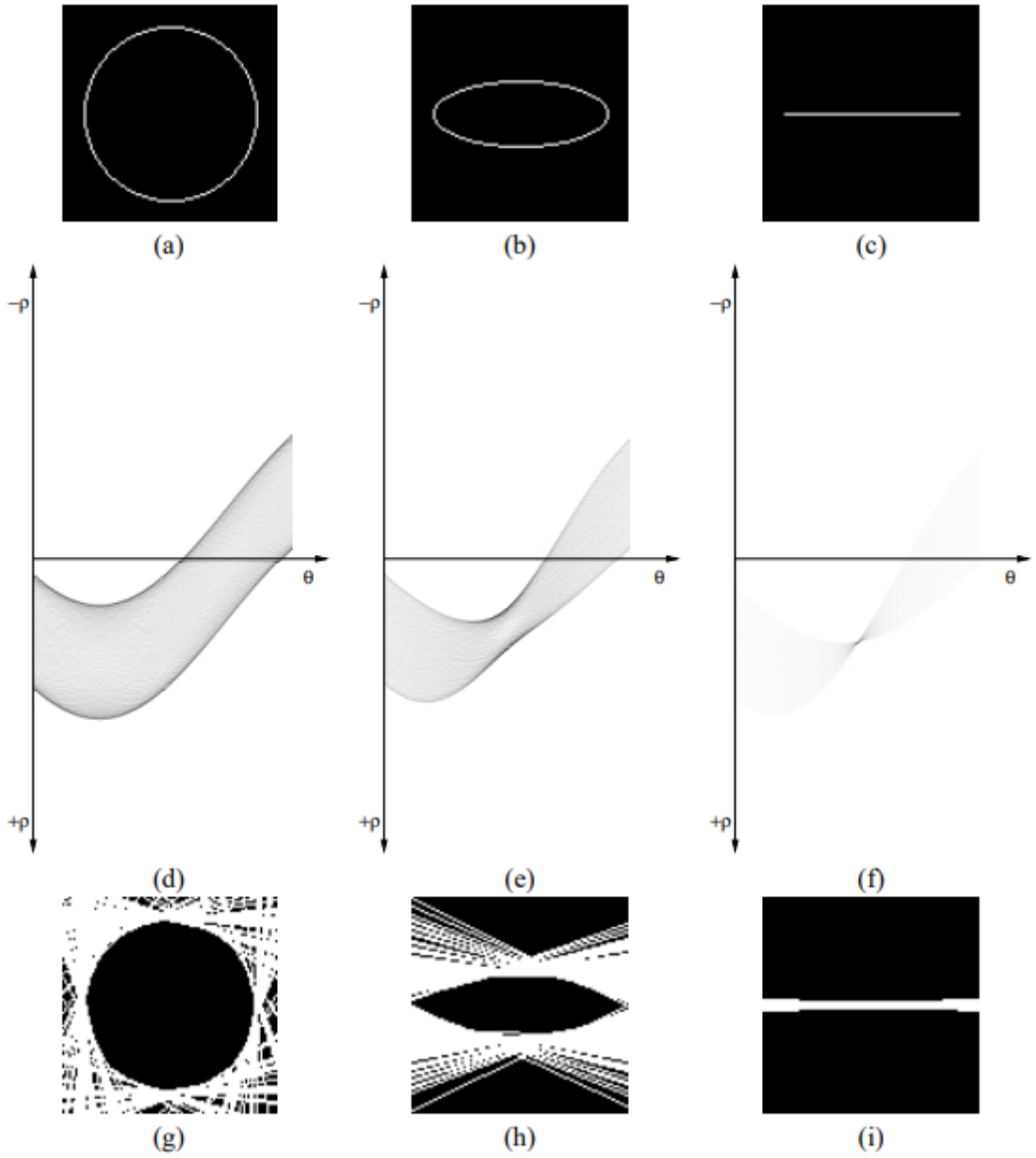
m, nagib

n, sjecište

Ova dva parametra karakteriziraju liniju. Jednadžba (4) mapira sve vrijednosti parametara kombinacije (m,n) prema skupu točaka slike. No, jednadžba (4) može se pročitati i inverzno, tj. ona predstavlja set ravnih linija, od kojih je svaka definirana točkom (m,n) u parametarskom prostoru, od kojih svaka prolazi točkom slike (x,y). Gledajući tako, svaka točka slike (x,y) definira ravnu liniju u parametarskom prostoru što predstavlja sve moguće kombinacije parametara (m,n) koji definiraju linije koje prolaze točkom (x,y). Točke koje su kolinearne u prostoru slike presjecaju se u zajedničkoj točci u parametarskom prostoru, a koordinate ovog sjecišta u parametarskom prostoru određuju ravnu liniju koja spaja sve kolinearne točke u prostoru slike. Time je problem detekcije linije u prostoru slike transformiran u problem traženja sjecišta (maksimuma) u parametarskom prostoru.

Ovaj postupak detekcije ravne linije HT-om jednostavno se može proširiti i na druge parametarski definirane krivulje na slici, ukoliko se točke krivulje na slici mogu opisati s n parametara $\alpha_1, \ldots, \alpha_n$ koje možemo prikazati u analitičkom obliku:

$$f(\alpha_1, \ldots., \alpha_n, x, y) = 0 \quad \textbf{(5)}$$



Slika 96. Houghova transformacija kružnice, elipse i ravne linije [19]

Gornja slika prikazuje postupak detekcije kružnice, elipse i ravne linije u smislu njihovih graničnih tangentnih linija, slike (a), (b) i (c) originalne su slike, (d), (e), i (f) predstavljaju HT originalnih slika, dok su slike (g), (h) i (i) transformacija parametarskog prostora nazad u prostor slike.

Python funkcije *HoughCircles()* zahtjeva nekoliko ulaznih argumenata koji su:

1. Slika na kojoj želimo detektirati kružnice, mora biti u sivim tonovima
2. Metoda koju koristimo za detekciju kružnica
    a. *cv2.HOUGH_STANDARD* – standardna HT
    b. *cv2.HOUGH_PARABOLISTIC* – vjerojatnosna HT, korisna u slučaju prisutnih dugih linearnih segmenata na slici
    c. *cv2.HOUGH_MULTI_SCALE* – više skalna varijanta standardne HT
    d. *cv2.HOUGH_GRADIENT* – temeljena na informaciji o nagibu, prema kojem se u smjeru nagiba širi i akumulatorska matrica
    e. *cv2.GRADIENT_ALT*
3. omjer rezolucije matrice akumulatora i rezolucije slike
4. minimalni razmak između dva centra radijusa kruga
5. u slučaju *HOUGH_GRADIENT* metode koju koristimo ovaj parametar odgovara pragu za primjenu *CannyEdge* detektora
6. u slučaju *HOUGH_GRADIENT* metode koju koristimo ovaj parametar odgovara pragu koji se mora zadovoljit za proglašenje centra kružnice
7. minimalni radijus kružnice
8. maksimalni radijus kružnice

Naravno svi parametri vezani uz dimenzije(primjerice radijus) izraženi su u pikselima.

Algoritam za detekciju kružnice u principu se sastoji od dva koraka:

1. Traženje svih mogućih kandidata za centar kružnice
2. Za svakog kandidata, traži se najbolji radijus

Najprije se dakle slika provlači kroz *CannyEdge* filter za detekciju bridova na slici. Potom se računa gradijent koristeći *Sobel* operator za svaki piksel. Zatim se za svaki piksel brida, povećava matrica akumulatora u oba smjera gradijenta. Potom se u akumulatoru odabiru sve ćelije koje su i lokalni maksimum i iznad određenog praga. Ovako odabrane ćelije akumulatora mogući su kandidati centara kružnice.

Kada smo pronašli kandidate centara, zadatak nam je naći najbolje radijuse kružnica za svaki od kandidata. Očigledno je sada parametarski prostor sveden na 1D. Kako bi se ovo 1D polje popunilo, za svaki od kandidata jednostavno računamo njegovu udaljenost od svih piksela bridova na slici, te povećavamo vrijednost odgovarajuće ćelije akumulatora. Najbolji radijus će biti onaj koji ima najviše glasova(najveća vrijednost ćelije). Ovaj postupak ponavljamo za svakog kandidata centra kružnice , te naposljetku odbacujemo sve kandidate koji nemaju dovoljno 'glasova' ili koji su previše blizu prethodno odabranog centra.

```
    # Convert to grayscale.
    gray = cv2.cvtColor(framed, cv2.COLOR_BGR2GRAY)
    # Blur using 3 * 3 kernel.
    gray_blurred = cv2.blur(gray, (3, 3))
    # Apply Hough transform on the blurred image.
    detected_circles = cv2.HoughCircles(gray_blurred,
                                        cv2.HOUGH_GRADIENT, 1, 200, param1 = 50,
                                        param2 = 30, minRadius = 10, maxRadius = 35)
    # Draw circles that are detected.
    if detected_circles is not None:
        # Convert the circle parameters a, b and r to integers.
        detected_circles = np.uint16(np.around(detected_circles))
        for pt in detected_circles[0, :]:
            a, b, r = pt[0], pt[1], pt[2]
            if r>=20:
                # Draw the circumference of the circle.
                cv2.circle(frame, (right_edge-80+a, top_edge-10+b), r, (0, 255, 0), 2)
            else:
                # Draw the circumference of the circle.
                cv2.circle(frame, (right_edge-80+a, top_edge-10+b), r, (0, 0, 255), 2)
            # cv2.imshow("Detected Circle", frame)
            cv2.imwrite('scan\detected.png', frame)
            image_to_pdf = Image.open(r'scan\detected.png')
            im_to_pdf = image_to_pdf.convert('RGB')
            im_to_pdf.save(r'scan\detected.pdf')
    #Check if all nuts are in position
    if detected_circles is not None:
        detected_elements = detected_circles[0,:,:]
    #print(detected_elements)
    vision_control_status=0
    if detected_circles is None:
        vision_control_status = 99;
    elif detected_elements.shape[0]!=2:
        vision_control_status = 99
    elif detected_elements.shape[0]==2:
        for i in range(2):
            if detected_elements[i,1]>27 and detected_elements[i,1]<58:
                if detected_elements[i,2]>18 and detected_elements[i,2]<30:
                    vision_control_status=vision_control_status+1
            if detected_elements[i,1]>290 and detected_elements[i,1]<330:
                if detected_elements[i,2]>18 and detected_elements[i,2]<30:
                    vision_control_status=vision_control_status+2

    return vision_control_status
```

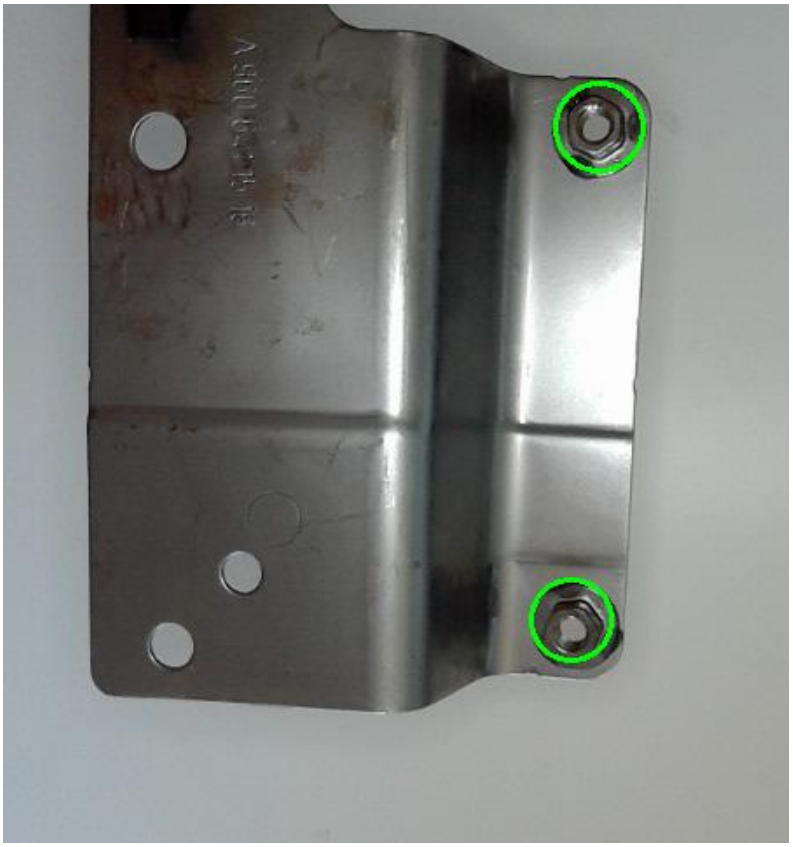
Slika 97. Funkcija *HoughCircles()*

Kako vidimo na slici iznad svi parametri detektiranih kružnica nalaze se u varijabli *detected_circles* (koordinate centra *a* i *b*, te radijus *r*). Naposlijetku je potrebno samo provjeriti nalaze li se detektirane kružnice na traženoj poziciji i koji je njihov radijus(obzirom da je otvor gdje se matica vari mnogo manji od obruba matice), prema čemu se potom donose zaključci o ispravnosti zavarenosti dijela, o čemu nam informaciju daje varijabla *vision_control_status*:

1. *vision_control_status=1* – zavarena samo prva matica
2. *vision_control_status=2* – zavarena samo druga matica
3. *vision_control_status=3* – zavarene sve matice

4. *vision_control_status=99* – greška vizijske kontrole

Slike ispod prikazuju izlazne rezultate detekcije matica na dijelovima.



Slika 98. Vizijska kontrola dijela - zavarene obadvije matice

Slika 99. Vizijska kontrola dijela - zavarena samo druga matica

Slika 100. Vizijska kontrola dijela - zavarena samo prva matica

Obzirom da se pri vizijskoj kontroli dijelova javljaju određena rasipanja u rezultatima parametara detektiranih kružnica, koordinate centra kružnice uzduž x i y osi te samog radijusa kružnice, uvedena su tolerancijska polja koja moraju biti zadovoljena ukoliko se radi o obrubu matice:

1. Tolerancijsko polje y-osi
    i. Za prvu maticu: 27-58 piksela
    ii. Za drugu maticu 290-330 piksela
2. Tolerancijsko polje radijusa obruba matice: 18-30 piksela

Ovaj program testiran je na 100 dobro varenih komada(na trenutnoj ručnoj stanici za varenje) te su rezultati detektiranih kružnica zapisivani za svaki kontrolirani dio, čime smo dobili rasipanja rezultata radijusa od ±4 piksela dok su rasipanja rezultata za y os nešto veća obzirom da se matice pri slikanju komada ne nalaze točno u centru ispod kamere, te je slika nešto izobličena u području matica, tako da su i rezultati malo lošiji, ali opet zadovoljavajući s rasipanjem od ±8 piksela za prvu maticu, dok za drugu maticu imamo rasipanje od ±12 piksela (rezultati se nalaze u tablici u prilogu).

Također veliki doprinos rasipanju rezultata leži i u izvedbi postava za testiranje vizijske kontrole, obzirom da je 3D printana robotska hvataljka poprimila dosta veliku zračnost, te pri ručnom ubacivanju komada ne postižemo uvijek istu poziciju dijela, koji bude nešto zakrenut i položen relativno prema kameri.

ZAKLJUČAK

U ovom diplomskom radu razrađena je konceptualna ideja rješenja za automatizaciju i robotizaciju procesa varenja matica koja se trenutno u sklopu odjela varenja tvrtke KM-Kovnica izvodi ručnom metodom, uz djelovanje odjela 100% kontrole gdje se prema zahtjevima kupaca za određene proizvode provodi ručna vizualna kontrola(ili uz pomoć kontrolnih naprava) svakog pojedinog dijela. Ovim rješenjem osim što smo modernizirali proizvodni proces varenja, ostvaruje se i ubrzanje i ujednačenje kvalitete proizvodnje obzirom da bi stanica mogla raditi 0-24 bez prestanka, uz redovito posluživanje same stanice 'svježim' dijelovima. Također ovim rješenjem uklanjamo potrebu za radnikom na svakom stroju, obzirom da bi jedan radnik mogao poslužiti do dvije ili tri automatske stanice za varenje matica(ovisno o tipu proizvoda i broju matica koji se vari). Također uz implementaciju rješenja vizijske kontrole dijelova, uklanjamo potrebu za ručnom provjerom svakog pojedinog dijela, čime bi se ostvarile značajne uštede ne samo pri smanjenju potrebnog broja radnika, nego najvećim dijelom ujednačenjem i osiguranjem kvalitete proizvoda, gdje ljudska greška ne bi uzrokovala nevaljale dijelove u gajbi s gotovim dijelovima čime se otklanjaju značajni troškovi uzrokovani reklamacijama kupaca na isporučene dijelove.

Ovdje primijenjena znanja za izradu rješenja automatske stanice za varenje stečena su tijekom studiranja na Fakultetu strojarstva i brodogradnje, točnije na smjeru Mehatronika-robotika. U radu je primijenjen širok spektar znanja, od konstrukcijske izvedbe i modeliranja uz pomoć CAD alata *SolidWorsk 2017*, zatim izrade simulacijskog rješenja unutar softverskog paketa *RoboDK*, preko programiranja PLC kontrolera i HMI korisničkog sučelja unutar Siemens-ovog paketa *Tia Portal v15.1,* te na kraju i programiranja u Python programskom jeziku.

Dakako ovdje prikazano rješenje ima potencijal i viziju primjene ne samo na ovdje prikazanom proizvodu, već bi se moglo primijeniti na desetke različitih proizvoda koji se trenutno proizvode u sklopu tvrtke KM-Kovnica, što bi podiglo konkurentnost poduzeća na tržištu i sasvim sigurno otvorilo put prema novim poslovima i implementacijama sličnih rješenja.

U sklopu ovog diplomskog rada naišao sam na hrpu problema u vidu načina izvedbe. Radeći i smišljajući način na koji bi se problem ovog zadatka riješio stekao sam neprocjenjiva znanja i uvid u način izvedbe i problematike unutar industrijskih pogona što će

mi sasvim sigurno dati određeno iskustvo u daljnjem radu što u tvrtki KM-Kovnica, što u budućim angažmanima i poslovima.

LITERATURA

[1] Web stranice tvrtke König Metall Hrvatska, https://km-croatia.com 10.1.2022

[2] Stroj za elektrootpono varenje, Dalex PMS 12-6, web stranica https://www.dalex.de/en/ 10.1.2022

[3] Katalog strojeva za elektrootporno varenje proizvođača Dalex, https://www.dalex.de/fileadmin/user_upload/DALEX_INDUSTRIAL_MACHINES.pdf

[4] Korisnički priručnik upravljačke jedinice stroja za varenje tvrtke Wahlenmeier Schweisstechnik GMBH modeli MPS100/200/300

[5] Sustavi dodavača(vibracijske zdjele i linearne vibracijske staze) ENSO, službena web stranica, https://enso.hr , 05.02.2022.

[6] Vibracijski sustavi dodavača matica Dengensha America, web stranica, https://www.dengensha.com/feeders/nut-feeders/vibratory-nut-feeder/ 05.02.2022.

[7] Kolaborativna robotska ruka UR10e, službene web stranice proizvođača Universal Robots, https://www.universal-robots.com, 05.02.2022.

[8] Pneumatska hvataljka proizvođača Festo, ostale pneumatske komponente, pozicijski senzori, službena web stranica proizvođača, https://www.festo.com/cms/hr_hr/index.htm, 15.02.2022.

[9] Senzori pozicije proizvođača Sick, službena web stranica, https://www.sick.com/at/en/, 15.02.2022.

[10] Industrijski konektori proizvođača Harting, službene web stranice, https://www.harting.com/US/en, 15.02.2022.

[11] Siemens PLC kontroler S7-1500, službena stranica proizvođača, https://mall.industry.siemens.com/mall/en/WW/Catalog/Product/6ES7516-3AN02-0AB0, 10.05.2022.

[12] Siemens HMI panel TP1500 Comfort, službena stranica proizvođača, https://support.industry.siemens.com/cs/pd/186709?pdti=pi&dl=en&lc=en-WW, 15.05.2022.

[13] Proračun sile robotske hvataljke, https://en.iprworldwide.com/calculation-of-gripping-force/, 22.05.2022.

[14] *snap7* biblioteka za PLC ethernet komunikaciju, web stranice, http://snap7.sourceforge.net, 22.05.2022.

[15] Web kamera Razer Kiyo, službene stranice proizvođača, https://www.razer.com/streaming-cameras/razer-kiyo/RZ19-02320100-R3U1, 25.05.2022.

[16] RoboDK uputstva, online priručnik, https://robodk.com/doc/en/Basic-Guide.html , 10.01.2022.

[17] RoboDK forum, web stranica, https://robodk.com/forum/ , 10.01.2022.

[18] web članak o Hough-ovoj transformaciji i python funkciji *HoughCircles()* https://theailearner.com/tag/hough-gradient-method/, 01.06.2022.

[19] Znanstveni rad o Hough-ovoj transformaciji - „The Hough Transform as a tool for image analysis“, Josep Llados, Computer Vision Center – Dept. Informatica. Universitat Autonoma de Barcelona, Ožujak 2003. http://www.cvc.uab.es/~josep/master/hough/hough_a.PDF

[20] Ispitni izvještaj snimanja struje Br. 170911A-MM, tvrtka BELMET 97, 11.09.2017.

[21] Princip rada vibrododavača, web mjesto https://pt.slideshare.net/ilyashussain6/parts-feeder, 27.06.2022.

PRILOZI

a) Python kod komunikacije RoboDK – PLC

b) Python skripta za učitavanje dijelova u RoboDK i proračun pozcija dijelova

c) Python kod vizijske kontrole dijelova

d) *Main* petlja PLC programa

e) SCL kod automatskog režima rada stanice

f) SCL kod ručnog režima rada stanice

g) *Compare_pos* funkcija

h) *HMI* ekrani

i) Tablica rezultata ispitivanja vizijske kontrole dijelova

a)

```
from robolink import *    # RoboDK API
from robodk import *      # Robot toolbox
RDK = Robolink()
import snap7
from snap7 import util
from Parts_load import *
from vision_control import *
import numpy as np

IP_ADDRESS = '192.168.0.30'
RACK = 0
SLOT = 1

plc = snap7.client.Client()
plc.connect(IP_ADDRESS,RACK,SLOT)
Connect status = plc.get connected()
print('Status konekcije', Connect_status)

Initialize()

# Import frames and objects from robodk
# Frames
Main_frame = RDK.Item('Robot_feed')
Load_frame = RDK.Item('Load_frame')
Unload_frame = RDK.Item('Unload_frame')
Curve_frame = RDK.Item('Curve_frame')
UR10_frame = RDK.Item('UR10 Base')
Feeder frame = RDK.Item('Feeder frame')
Dalex_frame = RDK.Item('Dalex Base')
Dodavac_frame = RDK.Item('Dodavac Base')
Block_frame = RDK.Item('Block_frame')

#Curve follow programs
Load_Dalex = RDK.Item('Load-Dalex')
Dalex_Load = RDK.Item('Dalex_Load')
Dalex_Camera = RDK.Item('Dalex-Camera')
Camera_Dalex = RDK.Item('Camera-Dalex')
Camera_Unload = RDK.Item('Camera-Unload')
Unload_Camera = RDK.Item('Unload-Camera')

#Mechanisms and robot; Initial positions
#Robot
robot = RDK.Item('UR10')
num_dofs = len(robot.JointsHome().list())
gripper = RDK.Item('gripper')
hole_gripper = RDK.Item('hole_gripper')
tool = RDK.Item('gripper')
tool_pose = tool.PoseTool()
```

```
hole_gripper.setVisible(False,False)
Home_position = RDK.Item('Home')
robot.MoveJ(Home_position)

#Dalex
dalex = RDK.Item('Dalex')
Dalex_Home = RDK.Item('Dalex_Home')
Dalex_Out = RDK.Item('Dalex_Out')
dalex.MoveL(Dalex_Home)

#Dodavac
dodavac = RDK.Item('Dodavac')
Dodavac_In = RDK.Item('Dodavac_in')
Dodavac_Out = RDK.Item('Dodavac_out')
dodavac.MoveL(Dodavac_In)

#Blocking_mechanism
blocking_mechanism =
RDK.Item('Blocking_mechanism')
Block_In = RDK.Item('Block_in')
Block_Out = RDK.Item('Block_out')
blocking_mechanism.MoveL(Block_In)

#Holding piston
holding_piston = RDK.Item('Holding_piston')
Hold In = RDK.Item('Hold in')
Hold_Out = RDK.Item('Hold_out')
holding_piston.MoveL(Hold_Out)

#Push piston
push_piston = RDK.Item('Push_cylinder')
Push In = RDK.Item('Push in')
Push_Out = RDK.Item('Push_out')
push_piston.MoveL(Push_In)

#Locking mechanism
locking_mechanism_closed =
RDK.Item('Locking mechanism closed')
locking_mechanism_open =
RDK.Item('Locking_mechanism_open')
locking_mechanism_open.setVisible(True,
False)
locking_mechanism_closed.setVisible(False,
False)

#Simulation of position senzors
#Dalex
class IO(object):
    def __init__(self):
        #Senzors
        self.dalex_home_s = True
        self.dalex_out_s = False
        self.dodavac_in_s = True
        self.dodavac_out_s = False
        self.block_in_s = True
        self.hold_out_s = True
        self.push_in_s = True
        self.push_out_s = False
        self.lock_open_s = True
        self.lock_closed_s = False
        #Mechanisms toggle
        self.dalex_toggle = False
        self.dodavac_toggle = False
        self.block_toggle = False
        self.hold_toggle = True
        self.push_toggle = False
        self.lock_toggle = False
        self.gripper_toggle = False
        self.gripper_status = 0
        self.load_parts = False
        self.robot_goto_toggle = False
        self.moveJ_error = False
        self.robot_goto_toggle_L = False
        self.robot_home = False
        self.wait_status = False
        self.reset_station=False
        self.part_number = 0
        #Robot_coordinates_in
        self.TCP1 = 0.0
        self.TCP2 = 0.0
        self.TCP3 = 0.0
        self.TCP4 = 0.0
        self.TCP5 = 0.0
        self.TCP6 = 0.0
        self.TCP = ['','','','','','']
        self.ActiveFrame = Dalex_frame
        #Vision control
        self.start_vision_control = False
        self.vision_control_status = 0

    def __call__(self):
        if dalex.Pose() ==
Dalex_Home.Pose():
            self.dalex_home_s = True
            self.dalex_out_s = False
        elif dalex.Pose() ==
Dalex_Out.Pose():
            self.dalex home s = False
            self.dalex_out_s = True
        else:
            self.dalex_home_s = False
            self.dalex_out_s = False
        if dodavac.Pose() ==
Dodavac In.Pose():
            self.dodavac_in_s = True
            self.dodavac_out_s = False
        elif dodavac.Pose() ==
Dodavac_Out.Pose():
            self.dodavac_in_s = False
            self.dodavac out s = True
        else:
            self.dodavac_in_s = False
            self.dodavac_out_s = False
        if blocking_mechanism.Pose() ==
Block_In.Pose():
            self.block_in_s = True
        else:
            self.block_in_s = False
        if holding_piston.Pose() ==
Hold_Out.Pose():
            self.hold_out_s = True
        else:
            self.hold_out_s = False
        if push_piston.Pose() ==
Push_In.Pose():
            self.push_in_s = True
            self.push_out_s = False
        elif push_piston.Pose() ==
Push_Out.Pose():
            self.push_in_s = False
            self.push_out_s = True
        else:
            self.push_in_s = False
            self.push_out_s = False
        if locking_mechanism_open.Visible()
== True:
            self.lock_open_s = True
            self.lock_closed_s = False
        elif
locking_mechanism_closed.Visible() == True:
            self.lock_open_s = False
            self.lock_closed_s = True
        else:
            self.lock_open_s = False
            self.lock_closed_s = False
s = IO()

def Attach():
    tool.AttachClosest()
```

```
    s.gripper_status = 1
    Reset_wait_status()

def Dettach(ActiveFrame):
    tool.DetachAll(ActiveFrame)
    s.gripper_status = 0
    Reset_wait_status()

def Reset_wait_status():
    DB_number = 1
    Start_address = 0
    Size = 11
    DB_toggle =
plc.db_read(DB_number,Start_address,Size)
    util.set_bool(DB_toggle,10,2,False)

plc.db_write(DB_number,Start_address,DB_tog
gle)

def Go_to_target():
    DB_number = 2
    Start_address = 0
    Size = 48
    DB_robot_coordinates =
plc.db_read(DB_number, Start_address, Size)
    s.TCP1 =
util.get_real(DB_robot_coordinates,24)
    s.TCP1 = round(s.TCP1,2)
    s.TCP2 =
util.get_real(DB_robot_coordinates, 28)
    s.TCP2 = round(s.TCP2,2)
    s.TCP3 =
util.get_real(DB_robot_coordinates, 32)
    s.TCP3 = round(s.TCP3,2)
    s.TCP4 =
util.get_real(DB_robot_coordinates, 36)
    s.TCP4 = round(s.TCP4,2)
    s.TCP5 =
util.get_real(DB_robot_coordinates, 40)
    s.TCP5 = round(s.TCP5,2)
    s.TCP6 =
util.get_real(DB_robot_coordinates, 44)
    s.TCP6 = round(s.TCP6,2)
    s.TCP =
[s.TCP1,s.TCP2,s.TCP3,s.TCP4,s.TCP5,s.TCP6]
    target = KUKA_2_Pose(s.TCP)
    frame_pose = robot.PoseFrame()
    robot_Joints = robot.Joints()
    target_joints = robot.SolveIK(target,
robot_Joints, tool_pose, frame_pose)
    if len(target_joints.list()) <
num_dofs:
        s.moveJ_error = True
        DB_communication_number = 1
        Start_address = 0
        Size = 15
        DB_toggle =
plc.db_read(DB_communication_number,Start_a
ddress,Size)

util.set_bool(DB_toggle,14,0,s.moveJ_error)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)

    elif s.wait_status==False:
        Toggle_Frames()
        robot.MoveJ(target)
        Reset_wait_status()

def Go_to_target_L():
    DB_number = 2
    Start_address = 0
    Size = 48
    DB_robot_coordinates =
plc.db_read(DB_number, Start_address, Size)
    s.TCP1 =
util.get_real(DB_robot_coordinates,24)
    s.TCP1 = round(s.TCP1,2)
    s.TCP2 =
util.get_real(DB_robot_coordinates, 28)
    s.TCP2 = round(s.TCP2,2)
    s.TCP3 =
util.get_real(DB_robot_coordinates, 32)
    s.TCP3 = round(s.TCP3,2)
    s.TCP4 =
util.get_real(DB_robot_coordinates, 36)
    s.TCP4 = round(s.TCP4,2)
    s.TCP5 =
util.get_real(DB_robot_coordinates, 40)
    s.TCP5 = round(s.TCP5,2)
    s.TCP6 =
util.get_real(DB_robot_coordinates, 44)
    s.TCP6 = round(s.TCP6,2)
    s.TCP =
[s.TCP1,s.TCP2,s.TCP3,s.TCP4,s.TCP5,s.TCP6]
    target = KUKA_2_Pose(s.TCP)
    Toggle_Frames()
    robot.MoveL(target)
    Reset_wait_status()

def Comm():
    DB_number = 1
    Start_address = 0
    Size = 2
    DB_communication_in =
plc.db_read(DB_number,Start_address,Size)

util.set_bool(DB_communication_in,0,0,s.dal
ex_home_s)

util.set_bool(DB_communication_in,0,1,s.dal
ex_out_s)

util.set_bool(DB_communication_in,0,2,s.dod
avac_in_s)

util.set_bool(DB_communication_in,0,3,s.dod
avac_out_s)

util.set_bool(DB_communication_in,0,4,s.blo
ck_in_s)

util.set_bool(DB_communication_in,0,5,s.hol
d_out_s)

util.set_bool(DB_communication_in,0,6,s.pus
h_in_s)

util.set_bool(DB_communication_in,0,7,s.pus
h_out_s)

util.set_bool(DB_communication_in,1,0,s.loc
k_open_s)

util.set_bool(DB_communication_in,1,1,s.loc
k_closed_s)

plc.db_write(DB_number,Start_address,DB_com
munication_in)
    TCP_coord = robot.Pose()
    TCP_coord = Pose_2_KUKA(TCP_coord)
    TCP_coord = ['%.2f' % elem for elem in
TCP_coord]
    DB_number = 2
    Start_address = 0
    Size = 48
    DB_robot_coordinates =
plc.db_read(DB_number, Start_address, Size)
    util.set_real(DB_robot_coordinates,0,
float(TCP_coord[0]))
    util.set_real(DB_robot_coordinates,4,
float(TCP_coord[1]))
```

```
    util.set_real(DB_robot_coordinates,8,
float(TCP_coord[2]))
    util.set_real(DB_robot_coordinates,12,
float(TCP_coord[3]))
    util.set_real(DB_robot_coordinates,16,
float(TCP_coord[4]))
    util.set_real(DB_robot_coordinates,20,
float(TCP_coord[5]))
    plc.db_write(DB_number, Start_address,
DB_robot_coordinates)

def Toggle_mechanisms():
    DB_communication_number = 1
    Start_address = 0
    Size = 13
    #toggling mechanisms
    DB_toggle =
plc.db_read(DB_communication_number,Start_a
ddress,Size)
    dalex_toggle =
util.get_bool(DB_toggle,1,2)
    dodavac_toggle =
util.get_bool(DB_toggle,1,3)
    block_toggle =
util.get_bool(DB_toggle,1,4)
    hold_toggle =
util.get_bool(DB_toggle,1,5)
    push_toggle =
util.get_bool(DB_toggle,1,6)
    lock_toggle =
util.get_bool(DB_toggle,1,7)
    if dalex_toggle == True:
        dalex.MoveL(Dalex_Out)
    elif dalex_toggle == False:
        dalex.MoveL(Dalex_Home)
    if dodavac_toggle == True:
        dodavac.MoveL(Dodavac_Out)
    elif dodavac_toggle == False:
        dodavac.MoveL(Dodavac_In)
    if block_toggle == True:
        blocking_mechanism.MoveL(Block_Out)
    elif block_toggle == False:
        blocking_mechanism.MoveL(Block_In)
    if hold_toggle == True:
        holding_piston.MoveL(Hold_Out)
    elif hold_toggle == False:
        holding_piston.MoveL(Hold_In)
    if push_toggle == True:
        push_piston.MoveL(Push_Out)
    elif push_toggle == False:
        push_piston.MoveL(Push_In)
    if lock_toggle == True:

locking_mechanism_closed.setVisible(True,
False)

locking_mechanism_open.setVisible(False,
False)
    elif lock_toggle == False:

locking_mechanism_closed.setVisible(False,
False)

locking_mechanism_open.setVisible(True,
False)

    #toggling reference frames
    Main_frame_toggle =
util.get_bool(DB_toggle,2,0)
    Unload_frame_toggle =
util.get_bool(DB_toggle,2,1)
    Load_frame_toggle =
util.get_bool(DB_toggle,2,2)
    Curve_frame_toggle =
util.get_bool(DB_toggle,2,3)
    UR10_frame_toggle =
util.get_bool(DB_toggle,2,4)
    Feeder_frame_toggle =
util.get_bool(DB_toggle,2,5)
    Dodavac_frame_toggle =
util.get_bool(DB_toggle,2,6)
    Dalex_frame_toggle =
util.get_bool(DB_toggle,2,7)
    Block_frame_toggle =
util.get_bool(DB_toggle,3,0)
    if Main_frame_toggle == True:
        robot.setPoseFrame(Dalex_frame)
        s.ActiveFrame = Dalex_frame
        util.set_bool(DB_toggle,2,0,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
    elif Unload_frame_toggle == True:
        robot.setPoseFrame(Unload_frame)
        util.set_bool(DB_toggle,2,1,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
        s.ActiveFrame = Unload_frame
    elif Load_frame_toggle == True:
        robot.setPoseFrame(Load_frame)
        util.set_bool(DB_toggle,2,2,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
        s.ActiveFrame = Load_frame
    elif Curve_frame_toggle == True:
        robot.setPoseFrame(Curve_frame)
        util.set bool(DB toggle,2,3,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
        s.ActiveFrame = Curve_frame
    elif UR10_frame_toggle == True:
        robot.setPoseFrame(UR10 frame)
        util.set_bool(DB_toggle,2,4,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
        s.ActiveFrame = UR10_frame
    elif Feeder_frame_toggle == True:
        robot.setPoseFrame(Feeder_frame)
        util.set_bool(DB_toggle,2,5,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
        s.ActiveFrame = Feeder_frame
    elif Dodavac_frame_toggle == True:
        robot.setPoseFrame(Dodavac_frame)
        util.set_bool(DB_toggle,2,6,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
        s.ActiveFrame = Dodavac_frame
    elif Dalex_frame_toggle == True:
        robot.setPoseFrame(Dalex_frame)
        util.set_bool(DB_toggle,2,7,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
        s.ActiveFrame = Dalex_frame
    elif Block_frame_toggle == True:
        robot.setPoseFrame(Block_frame)
        util.set_bool(DB_toggle,3,0,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
        s.ActiveFrame = Block_frame

    #Toggling gripper
    s.gripper_toggle =
util.get_bool(DB_toggle,3,7)
    if s.gripper_toggle == True and
s.gripper_status == 0:
```

```
        Attach()
    if s.gripper_toggle == False and
s.gripper_status == 1:
        Dettach(s.ActiveFrame)

    #Toggle parts load
    s.load_parts =
util.get_bool(DB_toggle,4,0)
    part_quantity =
util.get_int(DB_toggle,6)
    number_of_part =
util.get_int(DB_toggle,8)
    s.number_of_part = number_of_part
    if s.load_parts == True:
        target_unload, paleta, rows,
columns, paleta_q, paleta_pick =
Insert_parts_in_station(number_of_part,
part_quantity)
        Write_target_list(target_unload,
paleta, rows, columns, paleta_q,
paleta_pick)
        util.set_bool(DB_toggle,4,0,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
    #Reset station bool
    s.reset_station =
util.get_bool(DB_toggle,10,0)
    if s.reset_station == True:
        Reset_station()
        robot.setPoseFrame(Unload_frame)
        util.set bool(DB toggle,10,0,False)

plc.db_write(DB_communication_number,Start_
address, DB_toggle)
    #Toggle UR10e go to target
    s.robot_goto_toggle =
util.get bool(DB toggle,10,1)
    s.wait_status =
util.get_bool(DB_toggle,10,2)
    if s.robot_goto_toggle == True:
        Go_to_target()
        util.set_bool(DB_toggle,10,1,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
    s.robot_home =
util.get_bool(DB_toggle,10,3)
    if s.robot_home == True:
        robot.MoveJ(Home_position)
        util.set_bool(DB_toggle,10,3,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
    s.robot_goto_toggle_L =
util.get_bool(DB_toggle,10,4)
    if s.robot_goto_toggle_L == True:
        Go_to_target_L()
        util.set_bool(DB_toggle,10,4,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
    Toggle_Frames()

    ##Vision control
    s.start_vision_control =
util.get_bool(DB_toggle,10,5)
    if s.start_vision_control==True:
        s.vision_control_status =
proccess_photo()
        util.set_bool(DB_toggle,10,5,False)

util.set_int(DB_toggle,12,s.vision_control_
status)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
        Reset_wait_status()
    return

def Toggle_Frames():
    DB_communication_number = 1
    Start_address = 0
    Size = 11
    DB_toggle =
plc.db_read(DB_communication_number,Start_a
ddress,Size)
     #Toggling curve follow projects
    Load_Dalex_toggle =
util.get_bool(DB_toggle,3,1)
    Dalex_Load_toggle =
util.get_bool(DB_toggle,3,2)
    Dalex_Camera_toggle =
util.get_bool(DB_toggle,3,3)
    Camera_Dalex_toggle =
util.get_bool(DB_toggle,3,4)
    Camera_Unload_toggle =
util.get_bool(DB_toggle,3,5)
    Unload_Camera_toggle =
util.get_bool(DB_toggle,3,6)
    if Load_Dalex_toggle == True:
        Load_Dalex.RunProgram()
        util.set bool(DB toggle,3,1,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
    elif Dalex_Load_toggle == True:
        Dalex_Load.RunProgram()
        util.set bool(DB toggle,3,2,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
    elif Dalex_Camera_toggle == True:
        Dalex_Camera.RunProgram()
        util.set bool(DB toggle,3,3,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
    elif Camera_Dalex_toggle == True:
        Camera_Dalex.RunProgram()
        util.set_bool(DB_toggle,3,4,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
    elif Camera_Unload_toggle == True:
        Camera_Unload.RunProgram()
        util.set_bool(DB_toggle,3,5,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)
    elif Unload_Camera_toggle == True:
        Unload_Camera.RunProgram()
        util.set_bool(DB_toggle,3,6,False)

plc.db_write(DB_communication_number,Start_
address,DB_toggle)

def Write_target_list(target_unload,
paleta, rows, columns, paleta_q,
paleta_pick):
    DB_number = 3
    Start_address = 0
    Size = 7200
    DB_target_unload =
plc.db_read(DB_number,Start_address,Size)
    DB_number = 7
    Start_address = 0
    Size = 268
    DB_paleta = plc.db_read(DB_number,
Start_address, Size)
    DB_target_offset = [480,144]
    for i in range(paleta):
        for j in range(rows[i]):
            for k in range(columns[i]):
```

```
                for coord in range(6):

util.set_real(DB_target_unload,((i*paleta_q
)*24+j*DB_target_offset[s.number_of_part-
1]+k*24+coord*4),float(target_unload[i][j][
k][coord]))
    for i in range(paleta):
        for coord in range(6):

util.set_real(DB_paleta,4+i*24+4*coord,
float(paleta_pick[i][coord]))
    DB_number = 3
    Start_address = 0
    Size = 7200
    plc.db_write(DB_number, Start_address,
DB_target_unload)
    DB_number = 7
    Start_address = 0
    Size = 268
    plc.db_write(DB_number, Start_address,
DB_paleta)

def Insert_parts_in_station(number_of_part,
part_quantity):
    robot.setPoseFrame(Unload_frame)
    target unload, paleta, rows, columns,
paleta_q, paleta_pick =
Load_parts(number_of_part,part_quantity)
    if number_of_part==1:
        tool = RDK.Item('gripper')
        tool_pose = tool.PoseTool()
        robot.setTool(gripper)
        gripper.setVisible(True,True)

hole_gripper.setVisible(False,False)
    if number_of_part==2:
        tool = RDK.Item('hole_gripper')
        tool pose = tool.PoseTool()
        robot.setTool(hole_gripper)
        hole_gripper.setVisible(True,True)
        gripper.setVisible(False,False)

    return target_unload, paleta, rows,
columns, paleta_q, paleta_pick

while(True):
    s()
    Comm()
    ActiveFrame = Toggle_mechanisms()
```

b)

```
from robolink import *    # RoboDK API
from robodk import *      # Robot toolbox
import numpy as np
RDK = Robolink()
def Initialize():
    Main_frame = RDK.Item('Robot_feed')
    ##FRAMES
    global  Load_frame, Unload_frame
    Load_frame = RDK.Item('Load_frame')
    if not Load_frame.Valid():
        Load_frame =
RDK.AddFrame('Load_frame', Main_frame)
        Load_frame.setPose(transl(-
1784,1406,158))
    Unload_frame = RDK.Item('Unload_frame')
    if not Unload_frame.Valid():
        Unload_frame =
RDK.AddFrame('Unload_frame', Main_frame)

Unload_frame.setPose(transl(66,916,369.73))
     ########
    weld = [[-160,995.5,818.2,90,0,90],[-
326,998.86,818.2,90,0,90],[-
214.5,927,809,90,-90,0],[-128,959,809,90,-
90,0]]
    weld_target_1 = [-
160,995.5,818.2,90,0,90] #np
    weld_target_1_approach = [-160, 995.5,
850, 90, 0, 90] #np
    weld_approach = [[-160, 995.5, 850, 90,
0, 90],[-326,998.86,850,90,0,90],[-
214.5,927,850,90,-90,0],[-128,959,850,90,-
90,0]]
    weld_target_2 = [-
326,998.86,818.2,90,0,90]#np
    weld_target_2_approach = [-
326,998.86,850,90,0,90]#np

    paleta_files =
[r'C:\Users\jogam\OneDrive\Desktop\Diplomsk
i_rad\RoboDK\parts\paleta.step',
r'C:\Users\jogam\OneDrive\Desktop\Diplomski
_rad\RoboDK\parts\paleta2.step']
    paleta_q = [100,36,0]
    paleta_dimension = [[5,20],[6,6] ]
    part_files =
[r'C:\Users\jogam\OneDrive\Desktop\Diplomsk
i_rad\RoboDK\parts\Part.step',
r'C:\Users\jogam\OneDrive\Desktop\Diplomski
_rad\RoboDK\parts\Part2.stp']
    part_name = ['part1', 'part2']
    part_offset = [[76.5,101,169.5,-90,0,-
90],[84,172,43,0,0,-90]]
    part_axis_offset =
[[150,55,176.5],[125,185,52]]
    part_targets = [[76.5,101,154,-
360,0,180],[84.5,172,39,-102.75,-90,-77.3]]
    #Load_parts(1,20)
    #################
    return
############################
###FUNKCIJA ZA LOAD DJELOVA
############################
def Load_parts(part_number, part_quantity):
    global parts_number
    parts_number = part_quantity
    part = part_name[part_number-1]
    global paleta
    paleta =
int(part_quantity/paleta_q[part_number-1])
    if part_quantity%paleta_q[part_number-
1] != 0:
        paleta = paleta+1
    Paleta = ['0' for x in range(paleta)]
    paleta_approach = [0 for x in
range(paleta)]
    paleta_pick = [0 for x in
range(paleta)]
    pall = list("Paleta1")
    y = 1
    paleta_file = paleta_files[part_number-
1]
    for i in range(paleta):
        name = "".join(pall)
        check = RDK.Item(name)
        if not check.Valid():
            Paleta[i] =
RDK.AddFile(paleta_file, Unload_frame)

Paleta[i].setPose(KUKA_2_Pose([382.75,636.5
,35+part_axis_offset[part_number-
1][2]*i,90,0,90]))
            Paleta[i].setName(name)
            Unload_frame.setParam("Tree",
"Collapse")
        if part_number==1:
            paleta_approach[i] =
[382.75,636.5,75+part_axis_offset[part_numb
er-1][2]*i,-180,0,-180]#np
```

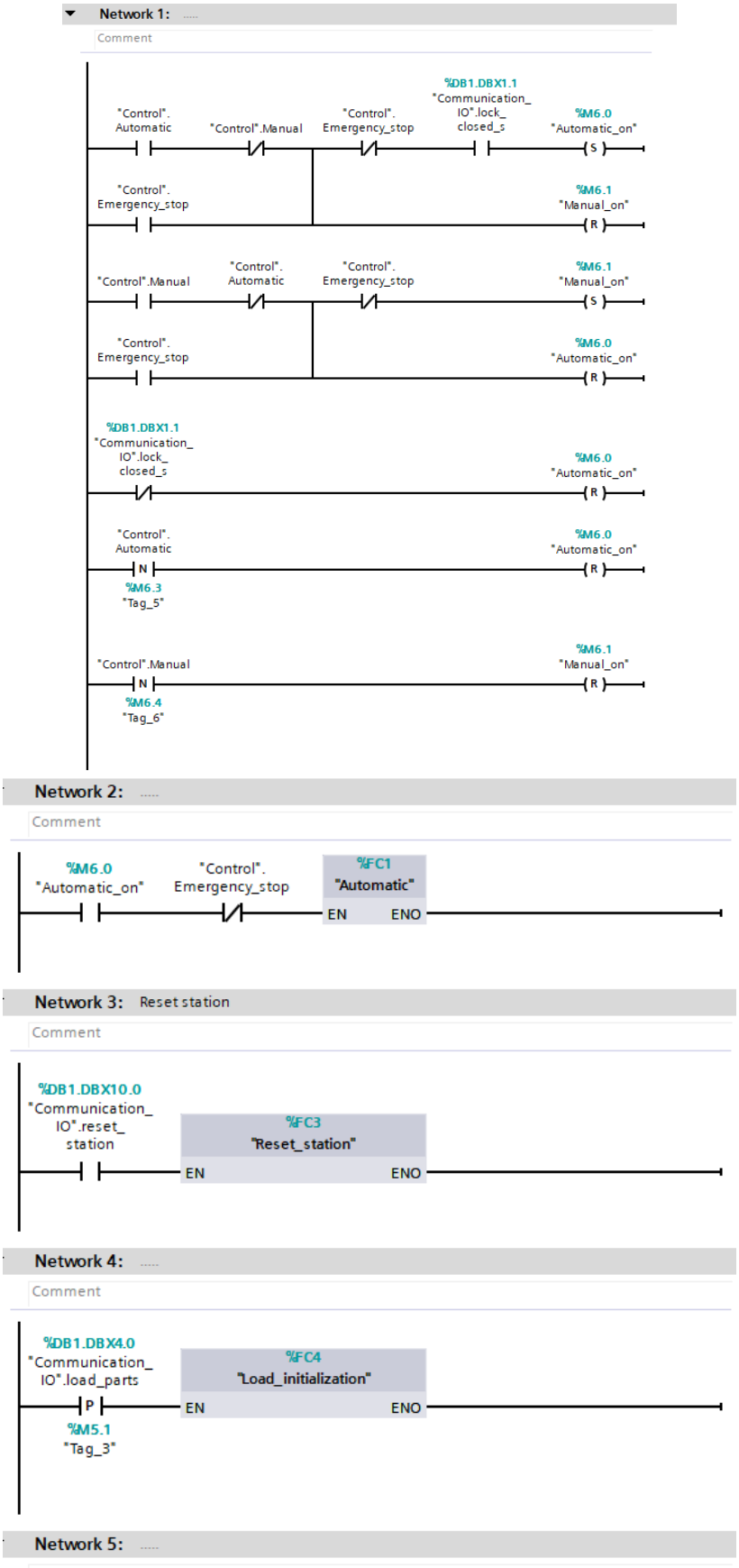
```
            paleta_pick[i] =
[382.75,636.5,25+part_axis_offset[part_numb
er-1][2]*i,-180,0,-180]#np
        if part_number==2:
            paleta_approach[i] =
[382.75,629.5,75+part_axis_offset[part_numb
er-1][2]*i,-180,0,-180]#np
            paleta_pick[i] =
[382.75,629.5,25+part_axis_offset[part_numb
er-1][2]*i,-180,0,-180]#np
        y = y+1
        pall[6] = str(y)
    paleta_store = [-20,570,1140,180,90,0]
    global rows
    global columns
    global Part
    rows = [0 for x in range(paleta)]
    columns = [0 for x in range(paleta)]
    ###Izracun popunjenosti svake palete
    partn = part_quantity
    for i in range(paleta):
        if partn-paleta_q[part_number-1]>0:
            rows[i] =
paleta_dimension[part_number-1][0]
            columns[i] =
paleta dimension[part number-1][1]
            partn = partn-
paleta_q[part_number-1]
        elif partn%paleta_q[part_number-
1]>=0:
            rows[i] = 0
            for j in
range(paleta_dimension[part_number-1][0]):
                if partn-
paleta_dimension[part_number-1][1]>=0:
                    columns[i] =
paleta_dimension[part_number-1][1]
                    rows[i] = rows[i]+1
                    partn = partn-
paleta_dimension[part_number-1][1]
    Part = [[['0' for x in
range(columns[y])] for x in range(rows[y])]
for y in range(paleta)]
    ###Part Loading
    x=list("P1")
    y=1
    for i in range(paleta):
        for j in range(rows[i]):
            for k in range(columns[i]):
                name = "".join(x)
                check = RDK.Item(name)
                if not check.Valid():
                    Part[i][j][k] =
RDK.AddFile(part_files[part_number-1],
Unload_frame)

Part[i][j][k].setPose(KUKA_2_Pose([part_off
set[part_number-
1][0]+(part_axis_offset[part_number-
1][0]*j),

part_offset[part_number-
1][1]+(part_axis_offset[part_number-
1][1]*k),

part_offset[part_number-
1][2]+part_axis_offset[part_number-1][2]*i,

part_offset[part_number-1][3],

part_offset[part_number-1][4],

part_offset[part_number-1][5]]))

Part[i][j][k].setName(name)

Unload_frame.setParam("Tree", "Collapse")
                y = y+1
                x[1] = str(y)
    ###UNLOAD TARGETI
    target_unload = [[[0 for i in
range(columns[y])] for j in range(rows[y])]
for y in range(paleta)]
    approach_unload = [[[0 for i in
range(columns[y])] for j in range(rows[y])]
for y in range(paleta)]
    for i in range(paleta):
        for j in range(rows[i]):
            for k in range(columns[i]):
                target_unload[i][j][k] =
[part_targets[part_number-
1][0]+(part_axis_offset[part_number-
1][0]*j),

part_targets[part_number-
1][1]+(part_axis_offset[part_number-
1][1]*k),

part_targets[part_number-
1][2]+part_axis_offset[part_number-1][2]*i,

part_targets[part_number-1][3],

part_targets[part_number-1][4],

part_targets[part_number-1][5]]#np
                approach_unload[i][j][k] =
target_unload[i][j][k]
    data file = open("data file.txt", "a")
    data_file.write("parts_number={0}
\npaleta={1} \nrows={2} \ncolumns={3}
\napproach_unload={4} \ntarget_unload={5}
\n".format(parts_number, paleta, rows,
columns, approach_unload, target_unload))
    data file.write("paleta approach={0}
\npaleta_pick={1} \npart_number={2}
\npaleta_store={3}".format(paleta_approach,
paleta_pick, part_number, paleta_store))
    data_file.close()
    return target_unload, paleta, rows,
columns,paleta_q[part_number-1],
paleta_pick
def Reset_station():
    Load_frame = RDK.Item('Load_frame')
    Unload_frame = RDK.Item('Unload_frame')
    Load_frame.Delete()
    Unload_frame.Delete()
    Load_frame = RDK.Item('Load_frame')
    if not Load_frame.Valid():
        Load_frame =
RDK.AddFrame('Load_frame', Main_frame)
        Load_frame.setPose(transl(-
1784,1406,158))
    Unload_frame = RDK.Item('Unload_frame')
    if not Unload_frame.Valid():
        Unload_frame =
RDK.AddFrame('Unload_frame', Main_frame)

Unload_frame.setPose(transl(66,916,369.73))
def Close():
    RDK.CloseStation()
```

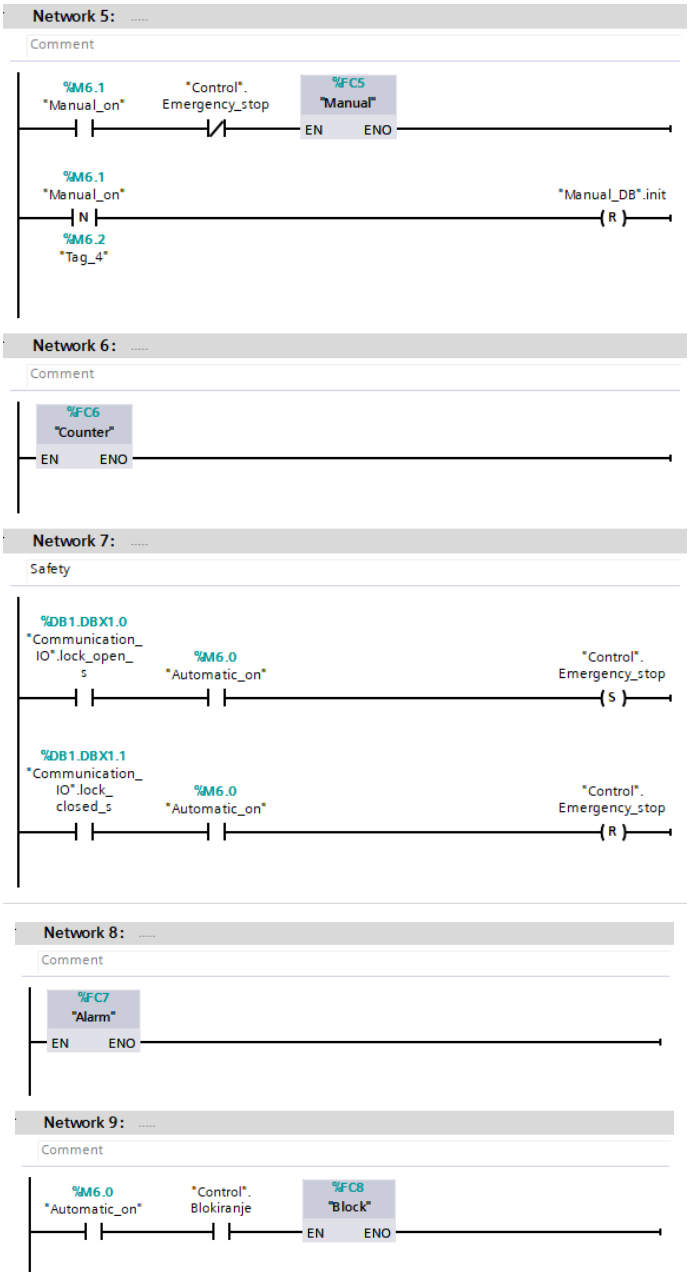
c)

```
#Importing opencv and numpy
import cv2
import numpy as np
from PIL import Image
```

```
def take_photo():
    #Connecting to capture device
    Camera = cv2.VideoCapture(1)
    #Get a frame from device
    ret, frame = Camera.read()
    cv2.imwrite('scan\scan.png', frame)
    #Release camera
    Camera.release()
    return frame

def proccess_photo():
    top_edge=10
    bottom_edge=10
    right_edge=80
    frame = take_photo()
    frame = frame[:,0:450]
    canny = cv2.Canny(frame, threshold1=80,
threshold2=200)
    #find left edge of image
    for i in range(430):
        if canny[250,i] == 255:
            left_edge = i;
            break
    #find right edge of image
    for i in reversed(range(430)):
        if canny[250,i]==255:
            right_edge=i;
            break;
    #find top edge
    for i in range(480):
        if canny[i,right_edge-40]==255:
            top edge=i;
            break;
    #find bottom edge
    for i in reversed(range(480)):
        if canny[i,right_edge-40]==255:
            bottom_edge=i;
            break;
    framed = frame[top_edge-
10:bottom_edge+10,right_edge-
80:right_edge+10]
    frame_size = frame.shape
    cv2.imwrite('scan\framed.png', framed)
    # Convert to grayscale.
    gray = cv2.cvtColor(framed,
cv2.COLOR_BGR2GRAY)
    cv2.imwrite('scan\gray.png', gray)
    # Blur using 3 * 3 kernel.
    gray_blurred = cv2.blur(gray, (3, 3))
    # Apply Hough transform on the blurred
image.
    detected_circles =
cv2.HoughCircles(gray_blurred,

cv2.HOUGH_GRADIENT, 1, 200, param1 = 50,

param2 = 30, minRadius = 10, maxRadius =
35)
    # Draw circles that are detected.
    if detected_circles is not None:

        # Convert the circle parameters a,
b and r to integers.
        detected_circles =
np.uint16(np.around(detected_circles))
        for pt in detected_circles[0, :]:
            a, b, r = pt[0], pt[1], pt[2]
            if r>=20:
                # Draw the circumference of
the circle.
                cv2.circle(frame,
(right_edge-80+a, top_edge-10+b), r, (0,
255, 0), 2)
            else:
                # Draw the circumference of
the circle.
                cv2.circle(frame,
(right_edge-80+a, top_edge-10+b), r, (0, 0,
255), 2)
            # cv2.imshow("Detected Circle",
frame)

cv2.imwrite('scan\detected.png', frame)
            image_to_pdf =
Image.open(r'scan\detected.png')
            im_to_pdf =
image to pdf.convert('RGB')

im_to_pdf.save(r'scan\detected.pdf')

    #Check if all nuts are in position
    if detected_circles is not None:
        detected elements =
detected_circles[0,:,:]
    #print(detected_elements)
    vision_control_status=0
    if detected_circles is None:
        vision_control_status = 99;
    elif detected elements.shape[0]!=2:
        vision_control_status = 99
    elif detected_elements.shape[0]==2:
        for i in range(2):
            if detected_elements[i,1]>27
and detected_elements[i,1]<58:
                if
detected_elements[i,2]>18 and
detected_elements[i,2]<30:

vision_control_status=vision_control_status
+1
            if detected_elements[i,1]>290
and detected_elements[i,1]<330:
                if
detected_elements[i,2]>18 and
detected_elements[i,2]<30:

vision_control_status=vision_control_status
+2

    return vision_control_status
```

d)


Network 1:
Comment
"Control". Automatic
"Control".Manual
"Control". Emergency_stop
%DB1.DBX1.1 "Communication_ IO".lock_ closed_s
%M6.0 "Automatic_on"
S
"Control". Emergency_stop
%M6.1 "Manual_on"
R
"Control".Manual
"Control". Automatic
"Control". Emergency_stop
%M6.1 "Manual_on"
S
"Control". Emergency_stop
%M6.0 "Automatic_on"
R
%DB1.DBX1.1 "Communication_ IO".lock_ closed_s
%M6.0 "Automatic_on"
R
"Control". Automatic
N
%M6.3 "Tag_5"
%M6.0 "Automatic_on"
R
"Control".Manual
N
%M6.4 "Tag_6"
%M6.1 "Manual_on"
R
Network 2:
Comment
%M6.0 "Automatic_on"
"Control". Emergency_stop
%FC1 "Automatic"
EN
ENO
Network 3: Reset station
Comment
%DB1.DBX10.0 "Communication_ IO".reset_ station
%FC3 "Reset_station"
EN
ENO
Network 4:
Comment
%DB1.DBX4.0 "Communication_ IO".load_parts
P
%M5.1 "Tag_3"
%FC4 "Load_initialization"
EN
ENO
Network 5:



e)

```
//Automatic mode of station

IF "Control".current_part <=
"Communication_IO".part_quantity THEN
   // Statement section WHILE
   CASE "Control".case_number OF
      1:  //Push out
         "Control".Timer_on := TRUE;
         "Communication_IO".push_toggle := TRUE;
         IF "Communication_IO".push_out_s = TRUE THEN
            "Control".case_number := 2;
            "Control".Timer_on := FALSE;
         END_IF;
      2:  //hold in
         "Communication_IO".hold_toggle := False;
         "Control".Timer_on := TRUE;
         IF "Communication_IO".hold_out_s = FALSE THEN
            "Control".case_number := 3;
            "Control".Timer_on := FALSE;
         END_IF;
      3:  //block out
         "Communication_IO".block_toggle := True;
         "Control".Timer_on := TRUE;
         IF "Communication_IO".block_in_s = FALSE THEN
            "Control".case_number := 4;
            "Control".Timer_on := FALSE;
         END_IF;
      4:  //hold out
```

```
        "Communication_IO".hold_toggle := True;
        "Control".Timer_on := TRUE;
        IF "Communication_IO".hold_out_s = TRUE THEN
          "Control".case_number := 5;
          "Control".Timer_on := FALSE;
        END_IF;
      5:  //Dalex - Load
        "Communication_IO".Dalex_Load_toggle := TRUE;
        "Control".case_number := 6;
        "Control".Timer_on := TRUE;
      6:
        "Compare_pos"(Compare_1 :=
"Robot_coordinates_IO".Coordinates_in,
                Compare_2 :=
"Robot_coordinates_IO".Load_position,
                Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          "Control".Timer_on := FALSE;
          "Communication_IO".Unload_frame_toggle :=
TRUE;
          //"Control".case_number := 7;
          #Comp_status := False;

          "Control".robot_target :=
"Target_unload".target_unload[("Communication_IO".part_qua
ntity - "Control".current_part)];
          "Control".robot_target[2] += 100;
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "Control".case_number := 8;
          "Control".Timer_on := TRUE;
        END_IF;
      //7://approach unload
        //"Control".robot_target :=
"Target_unload".target_unload[("Communication_IO".part_qua
ntity - "Control".current_part)];
        //"Control".robot_target[2] += 100;
        //"Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
        //"Communication_IO".robot_goto_toggle := TRUE;
        //"Control".case_number := 8;
        //"Control".Timer_on := TRUE;

      8://target unload
        "Compare_pos"(Compare_1 := "Control".robot_target,
                Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          "Control".case_number := 9;
          #Comp_status := FALSE;
          "Control".robot_target :=
"Target_unload".target_unload[("Communication_IO".part_qua
ntity - "Control".current_part)];
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "Control".Timer_on := FALSE;
        END_IF;


      9://attach part
        "Compare_pos"(Compare_1 := "Control".robot_target,
                Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                Compare_status => #Comp_status);
        "Control".Timer_on := TRUE;
        IF #Comp_status = TRUE THEN
          "Control".case_number := 10;
          #Comp_status := FALSE;
          "Communication_IO".wait_status := True;
          "Communication_IO".gripper_toggle := True;
          "Control".Timer_on := FALSE;
        END_IF;

      10:// back to approach unload
        "Control".Timer_on := TRUE;
        IF "Communication_IO".wait_status = FALSE THEN
          "Control".robot_target :=
"Target_unload".target_unload[("Communication_IO".part_qua
ntity - "Control".current_part)];
          "Control".robot_target[2] += 100;
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "Control".case_number := 11;
          "Control".Timer_on := FALSE;
        END_IF;
      11://Load - Dalex
        "Compare_pos"(Compare_1 := "Control".robot_target,
                Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                Compare_status => #Comp_status);
        "Control".Timer_on := TRUE;
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".Load_Dalex_toggle := TRUE;
          "Control".robot_target :=
"Robot_coordinates_IO".Dalex_position;
```

```
            "Control".case_number := 12;
            "Control".Timer_on := FALSE;
         END_IF;
      12://approach weld target 1
         "Compare_pos"(Compare_1 := "Control".robot_target,
                  Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                  Compare_status => #Comp_status);
         "Control".Timer_on := TRUE;
         IF #Comp_status = TRUE THEN
            #Comp_status := FALSE;
            "Communication_IO".Dalex_frame_toggle := TRUE;
            "Control".robot_target :=
"Weld_targets".Weld_approach_1["Communication_IO".numbe
r_of_part - 1];
            "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
            "Communication_IO".robot_goto_toggle := TRUE;
            "Control".case_number := 13;
            "Control".Timer_on := FALSE;
         END_IF;
      13://weld target 1
         "Compare_pos"(Compare_1 := "Control".robot_target,
                  Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                  Compare_status => #Comp_status);
         "Control".Timer_on := TRUE;
         IF #Comp_status = TRUE THEN
            #Comp_status := FALSE;
            "Control".robot_target :=
"Weld_targets".Weld_target_1["Communication_IO".number_o
f_part - 1];
            "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
            "Communication_IO".robot_goto_toggle := TRUE;
            "Control".case_number := 14;
            "Control".Timer_on := FALSE;
         END_IF;
      14://Dodavac_out
         "Compare_pos"(Compare_1 := "Control".robot_target,
                  Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                  Compare_status => #Comp_status);
         "Control".Timer_on := TRUE;
         IF #Comp_status = TRUE THEN
            #Comp_status := False;
            "Communication_IO".dodavac_toggle := TRUE;
            "Control".case_number := 15;
            "Control".Timer_on := FALSE;
         END_IF;
      15://Dodavac in start feeder
         "Control".Timer_on := TRUE;
         IF "Communication_IO".dodavac_out_s = TRUE THEN
            "Communication_IO".dodavac_toggle := FALSE;
            "Communication_IO".push_toggle := FALSE;
            "Communication_IO".block_toggle := False;
            "Control".case_number := 16;
            "Control".Timer_on := FALSE;
         END_IF;
      16://Push the nut in position and proceed
         "Control".Timer_on := TRUE;
         IF "Communication_IO".push_in_s = TRUE THEN
            "Communication_IO".push_toggle := TRUE;
            "Communication_IO".hold_toggle := FALSE;
            "Control".case_number := 17;
            "Control".Timer_on := FALSE;
         END_IF;
      17: // lock the feeder
         "Control".Timer_on := TRUE;
         IF "Communication_IO".push_out_s = TRUE THEN
            "Communication_IO".block_toggle := TRUE;
            "Communication_IO".hold_toggle := TRUE;
            "Control".case_number := 18;
            "Control".Timer_on := FALSE;
         END_IF;

      18://Start DALEX cycle
         "Compare_pos"(Compare_1 := "Control".robot_target,
                  Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                  Compare_status => #Comp_status);
         "Control".Timer_on := TRUE;
         IF #Comp_status = TRUE THEN
            #Comp_status := FALSE;
            "Communication_IO".dalex_toggle := TRUE;
            "Control".case_number := 19;
            "Control".Timer_on := FALSE;
         END_IF;
      19://Return Dalex
         "Control".Timer_on := TRUE;
         IF "Communication_IO".dalex_out_s = TRUE THEN
            "Communication_IO".dalex_toggle := FALSE;
            "Control".case_number := 20;
            "Control".Timer_on := FALSE;
         END_IF;
      20://Back to approach weld 1
         "Control".Timer_on := TRUE;
         IF "Communication_IO".dalex_home_s = TRUE THEN
```

```
            "Control".robot_target :=
"Weld_targets".Weld_approach_1["Communication_IO".numbe
r_of_part - 1];
            "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
            "Communication_IO".robot_goto_toggle := TRUE;
            "Control".case_number := 21;
            "Control".Timer_on := FALSE;
         END_IF;
      21://Approach weld target 2
         "Compare_pos"(Compare_1 := "Control".robot_target,
                  Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                  Compare_status => #Comp_status);
         "Control".Timer_on := TRUE;
         IF #Comp_status = TRUE THEN
            #Comp_status := FALSE;
            "Control".robot_target :=
"Weld_targets".Weld_approach_2["Communication_IO".numbe
r_of_part - 1];
            "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
            "Communication_IO".robot_goto_toggle := TRUE;
            "Control".case_number := 22;
            "Control".Timer_on := FALSE;
         END_IF;
      22://Weld target 2
         "Compare_pos"(Compare_1 := "Control".robot_target,
                  Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                  Compare_status => #Comp_status);
         "Control".Timer_on := TRUE;
         IF #Comp_status = TRUE THEN
            #Comp_status := FALSE;
            "Control".robot_target :=
"Weld_targets".Weld_target_2["Communication_IO".number_o
f_part - 1];
            "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
            "Communication_IO".robot_goto_toggle := TRUE;
            "Control".case_number := 23;
            "Control".Timer_on := FALSE;
         END_IF;
      23://Dodavac_out
         "Compare_pos"(Compare_1 := "Control".robot_target,
                  Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                  Compare_status => #Comp_status);
         "Control".Timer_on := TRUE;
         IF #Comp_status = TRUE THEN
            #Comp_status := False;
            "Communication_IO".dodavac_toggle := TRUE;
            "Control".case_number := 24;
            "Control".Timer_on := FALSE;
         END_IF;
      24://Dodavac in start feeder
         "Control".Timer_on := TRUE;
         IF "Communication_IO".dodavac_out_s = TRUE THEN
            "Communication_IO".dodavac_toggle := FALSE;
            "Communication_IO".push_toggle := FALSE;
            "Communication_IO".block_toggle := False;
            "Control".case_number := 25;
            "Control".Timer_on := FALSE;
         END_IF;
      25://Push the nut in position and proceed
         "Control".Timer_on := TRUE;
         IF "Communication_IO".push_in_s = TRUE THEN
            "Communication_IO".push_toggle := TRUE;
            "Communication_IO".hold_toggle := FALSE;
            "Control".case_number := 26;
            "Control".Timer_on := FALSE;
         END_IF;
      26: // lock the feeder
         "Control".Timer_on := TRUE;
         IF "Communication_IO".push_out_s = TRUE THEN
            "Communication_IO".block_toggle := TRUE;
            "Communication_IO".hold_toggle := TRUE;
            "Control".case_number := 27;
            "Control".Timer_on := FALSE;
         END_IF;
      27://Start second DALEX cycle
         "Compare_pos"(Compare_1 := "Control".robot_target,
                  Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                  Compare_status => #Comp_status);
         "Control".Timer_on := TRUE;
         IF #Comp_status = TRUE THEN
            #Comp_status := FALSE;
            "Communication_IO".dalex_toggle := TRUE;
            "Control".case_number := 28;
            "Control".Timer_on := FALSE;
         END_IF;
      28:
         "Control".Timer_on := TRUE;
         IF "Communication_IO".dalex_out_s = TRUE THEN
            "Communication_IO".dalex_toggle := FALSE;
            "Control".case_number := 29;
            "Control".Timer_on := FALSE;
         END_IF;
      29://Back to approach weld 2
```

```
        "Control".Timer_on := TRUE;
        IF "Communication_IO".dalex_home_s = TRUE THEN
          "Control".robot_target :=
"Weld_targets".Weld_approach_2["Communication_IO".numbe
r_of_part - 1];
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "Control".case_number := 30;
          "Control".Timer_on := FALSE;
        END_IF;
      30://Start Dalex-Camera
        "Compare_pos"(Compare_1 := "Control".robot_target,
                Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                Compare_status => #Comp_status);
        "Control".Timer_on := TRUE;
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".Dalex_Camera_toggle :=
TRUE;
          "Control".robot_target :=
"Robot_coordinates_IO".Camera_position;
          "Control".case_number := 31;
          "Control".Timer_on := FALSE;
        END_IF;
      31://Go to visual control position
        "Compare_pos"(Compare_1 := "Control".robot_target,
                Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                Compare_status => #Comp_status);
        "Control".Timer_on := TRUE;
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".Load_frame_toggle := TRUE;
          "Control".robot_target :=
"Robot_coordinates_IO".Vision_control_pose["Communication
_IO".number_of_part - 1];
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "HMI_tags".detected_visible := FALSE;
          "Control".case_number := 32;
          "Control".Timer_on := FALSE;
        END_IF;
      32://Start vision control
        "Compare_pos"(Compare_1 := "Control".robot_target,
                Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                Compare_status => #Comp_status);
        "Control".Timer_on := TRUE;
        IF #Comp_status = TRUE AND
"Control".Vision_control_error=FALSE THEN
          #Comp_status := FALSE;
          "Communication_IO".wait_status := TRUE;
          //Manually start vision control
          "Communication_IO".start_vision_control := TRUE;
          "Control".case_number := 33;
          "Control".Timer_on := FALSE;
        END_IF;
      33://Check vision control status
        "HMI_tags".vision_control_stauts :=
"Communication_IO".vision_control_status;
        IF "Communication_IO".wait_status = FALSE AND
"Communication_IO".vision_control_status <> 0 THEN
          "HMI_tags".detected_visible := TRUE;
          IF "Communication_IO".vision_control_status = 99
THEN
            "Communication_IO".vision_control_status := 0;
            "Control".Vision_control_error := TRUE;
            "Control".case_number := 32;
          ELSIF "Communication_IO".vision_control_status =
2 THEN
            "Communication_IO".vision_control_status := 0;
            "Control".case_number := 101;
          ELSIF "Communication_IO".vision_control_status =
1 THEN
            "Communication_IO".vision_control_status := 0;
            "Control".case_number := 201;
          ELSIF "Communication_IO".vision_control_status =
3 THEN
            "Communication_IO".vision_control_status := 0;
            "Control".case_number := 34;
          END_IF;
        END_IF;
      34://Start Camera - Unload
        "Compare_pos"(Compare_1 := "Control".robot_target,
                Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                Compare_status => #Comp_status);
        "Control".Timer_on := TRUE;
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".Camera_Unload_toggle :=
TRUE;
          "Control".robot_target :=
"Robot_coordinates_IO".Unload_position;
          "Control".case_number := 35;
          "Control".Timer_on := FALSE;
        END_IF;
```

```
      35://DETACH PART
        "Compare_pos"(Compare_1 := "Control".robot_target,
                Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                Compare_status => #Comp_status);
        "Control".Timer_on := TRUE;
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".wait_status := TRUE;
          "Communication_IO".gripper_toggle := FALSE;
          "Control".case_number := 36;
          "Control".Timer_on := FALSE;
        END_IF;
      36://GOING BACK
        //UNLOAD - CAMERA
        "Control".Timer_on := TRUE;
        IF "Communication_IO".wait_status = FALSE THEN
          "Communication_IO".Unload_Camera_toggle :=
TRUE;
          "Control".robot_target :=
"Robot_coordinates_IO".Camera_position;
          "Control".case_number := 37;
          "Control".Timer_on := FALSE;
        END_IF;
      37://CAMERA - DALEX
        "Compare_pos"(Compare_1 := "Control".robot_target,
                Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                Compare_status => #Comp_status);
        "Control".Timer_on := TRUE;
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".Camera_Dalex_toggle :=
TRUE;
          "Control".robot_target :=
"Robot_coordinates_IO".Dalex_position;
          "Control".case_number := 38;
          "Control".Timer_on := FALSE;
        END_IF;
      38://last CASE
        "Compare_pos"(Compare_1 := "Control".robot_target,
                Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                Compare_status => #Comp_status);
        "Control".Timer_on := TRUE;
        IF #Comp_status = TRUE AND
("Communication_IO".part_quantity - "Control".current_part)
MOD "Paleta".paleta_q = 0 THEN
          #Comp_status := FALSE;
          "Control".case_number := 901;
          "Communication_IO".Dalex_Load_toggle := TRUE;
          "Control".robot_target :=
"Robot_coordinates_IO".Load_position;
          "Control".Timer_on := FALSE;

        ELSIF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Control".current_part += 1;
          "Control".case_number := 1;
          "HMI_tags".Brojac_gajba += 1;
          "HMI_tags".Brojac_serija += 1;
          "HMI_tags".Brojac_smjena += 1;
          "Control".Timer_on := FALSE;
        END_IF;

      101://Go back to Dalex to repeat weld 1
        //Start Camera - Dalex
        "Compare_pos"(Compare_1 := "Control".robot_target,
                Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".Camera_Dalex_toggle :=
TRUE;
          "Control".robot_target :=
"Robot_coordinates_IO".Dalex_position;
          "Control".case_number := 102;
        END_IF;
      102://Go to approach weld 1
        "Compare_pos"(Compare_1 := "Control".robot_target,
                Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".Dalex_frame_toggle := TRUE;
          "Control".robot_target :=
"Weld_targets".Weld_approach_1["Communication_IO".numbe
r_of_part - 1];
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "Control".case_number := 103;
        END_IF;
      103://weld target 1
        "Compare_pos"(Compare_1 := "Control".robot_target,
                Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                Compare_status => #Comp_status);
```

```
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Control".robot_target :=
"Weld_targets".Weld_target_1["Communication_IO".number_o
f_part - 1];
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "Control".case_number := 104;
        END_IF;
      104://Dodavac_out
        "Compare_pos"(Compare_1 := "Control".robot_target,
                      Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := False;
          "Communication_IO".dodavac_toggle := TRUE;
          "Control".case_number := 105;
        END_IF;
      105://Dodavac in start feeder
        IF "Communication_IO".dodavac_out_s = TRUE THEN
          "Communication_IO".dodavac_toggle := FALSE;
          "Communication_IO".push_toggle := FALSE;
          "Communication_IO".block_toggle := False;
          "Control".case_number := 106;
        END_IF;
      106://Push the nut in position and proceed
        IF "Communication_IO".push_in_s = TRUE THEN
          "Communication_IO".push_toggle := TRUE;
          "Communication_IO".hold_toggle := FALSE;
          "Control".case_number := 107;
        END_IF;
      107: // lock the feeder
        IF "Communication_IO".push_out_s = TRUE THEN
          "Communication_IO".block_toggle := TRUE;
          "Communication_IO".hold_toggle := TRUE;
          "Control".case_number := 108;
        END_IF;


      108://Start DALEX cycle
        "Compare_pos"(Compare_1 := "Control".robot_target,
                      Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".dalex_toggle := TRUE;
          "Control".case_number := 109;
        END_IF;
      109://Return Dalex
        IF "Communication_IO".dalex_out_s = TRUE THEN
          "Communication_IO".dalex_toggle := FALSE;
          "Control".case_number := 110;
        END_IF;
      110://Back to approach weld 1
        IF "Communication_IO".dalex_home_s = TRUE THEN
          "Control".robot_target :=
"Weld_targets".Weld_approach_1["Communication_IO".numbe
r_of_part - 1];
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "Control".case_number := 30;
        END_IF;
      201://Go back to Dalex to repeat weld 2
        //Start Camera - Dalex
        "Compare_pos"(Compare_1 := "Control".robot_target,
                      Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".Camera_Dalex_toggle :=
TRUE;
          "Control".robot_target :=
"Robot_coordinates_IO".Dalex_position;
          "Control".case_number := 202;
        END_IF;
      202://Approach weld target 2
        "Compare_pos"(Compare_1 := "Control".robot_target,
                      Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".Dalex_frame_toggle := TRUE;
          "Control".robot_target :=
"Weld_targets".Weld_approach_2["Communication_IO".numbe
r_of_part - 1];
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "Control".case_number := 203;
        END_IF;
      203://Weld target 2
        "Compare_pos"(Compare_1 := "Control".robot_target,
                      Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_status => #Comp_status);
```

```
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Control".robot_target :=
"Weld_targets".Weld_target_2["Communication_IO".number_o
f_part - 1];
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "Control".case_number := 204;
        END_IF;
      204://Dodavac_out
        "Compare_pos"(Compare_1 := "Control".robot_target,
                      Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := False;
          "Communication_IO".dodavac_toggle := TRUE;
          "Control".case_number := 205;
        END_IF;
      205://Dodavac in start feeder
        IF "Communication_IO".dodavac_out_s = TRUE THEN
          "Communication_IO".dodavac_toggle := FALSE;
          "Communication_IO".push_toggle := FALSE;
          "Communication_IO".block_toggle := False;
          "Control".case_number := 206;
        END_IF;
      206://Push the nut in position and proceed
        IF "Communication_IO".push_in_s = TRUE THEN
          "Communication_IO".push_toggle := TRUE;
          "Communication_IO".hold_toggle := FALSE;
          "Control".case_number := 207;
        END_IF;
      207: // lock the feeder
        IF "Communication_IO".push_out_s = TRUE THEN
          "Communication_IO".block_toggle := TRUE;
          "Communication_IO".hold_toggle := TRUE;
          "Control".case_number := 208;
        END_IF;
      208://Start second DALEX cycle
        "Compare_pos"(Compare_1 := "Control".robot_target,
                      Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".dalex_toggle := TRUE;
          "Control".case_number := 209;
        END_IF;
      209:
        IF "Communication_IO".dalex_out_s = TRUE THEN
          "Communication_IO".dalex_toggle := FALSE;
          "Control".case_number := 210;
        END_IF;
      210://Back to approach weld 2
        IF "Communication_IO".dalex_home_s = TRUE THEN
          "Control".robot_target :=
"Weld_targets".Weld_approach_2["Communication_IO".numbe
r_of_part - 1];
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "Control".case_number := 30;
        END_IF;



      901://Pallete removal - approach pallete
        "Compare_pos"(Compare_1 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_2 :=
"Robot_coordinates_IO".Load_position,
                      Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".Unload_frame_toggle :=
TRUE;
          "Control".robot_target :=
"Paleta".paleta_pick["Paleta".broj_paleta - 1];
          "Control".robot_target[2] += 50;
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "Control".case_number := 902;
        END_IF;



      902: //go to pallete target
        "Compare_pos"(Compare_1 := "Control".robot_target,
                      Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Control".robot_target :=
"Paleta".paleta_pick["Paleta".broj_paleta - 1];
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle_L := TRUE;
          "Control".case_number := 903;
        END_IF;
```

```
      903://attach pallete
        "Compare_pos"(Compare_1 := "Control".robot_target,
                      Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".gripper_toggle := TRUE;
          "Communication_IO".wait_status := TRUE;
          "Control".case_number := 904;
        END_IF;
      904:// GO up
        IF "Communication_IO".wait_status = FALSE THEN
          "Control".robot_target[2] := 1000;
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle_L := TRUE;
          "Control".case_number := 905;
        END_IF;
      905:
        "Compare_pos"(Compare_1 := "Control".robot_target,
                      Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Control".robot_target[2] := 1400;
          "Control".robot_target[4] := 90;
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "Control".case_number := 906;
        END_IF;
      906://approach pallete unload position
        "Compare_pos"(Compare_1 := "Control".robot_target,
                      Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".Load_frame_toggle := TRUE;
          "Control".robot_target :=
"Paleta".paleta_approach_unload;
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "Control".case_number := 907;
        END_IF;
      907://goto unload pallete
        "Compare_pos"(Compare_1 := "Control".robot_target,
                      Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Control".robot_target := "Paleta".paleta_unload;
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "Control".case_number := 908;
        END_IF;
      908://Detach pallete
        "Compare_pos"(Compare_1 := "Control".robot_target,
                      Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".gripper_toggle := FALSE;
          "Communication_IO".wait_status := TRUE;
          "Control".case_number := 909;
        END_IF;
      909://Back to approach unload pallete
        IF "Communication_IO".wait_status = FALSE THEN
          "Control".robot_target :=
"Paleta".paleta_approach_unload;
          "Robot_coordinates_IO".Coordinates_out :=
"Control".robot_target;
          "Communication_IO".robot_goto_toggle := TRUE;
          "Control".case_number := 910;
        END_IF;
      910://start load-dalex
        "Compare_pos"(Compare_1 := "Control".robot_target,
                      Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := FALSE;
          "Communication_IO".Load_Dalex_toggle := TRUE;
          "Control".robot_target :=
"Robot_coordinates_IO".Dalex_position;
          "Control".case_number := 911;
        END_IF;
      911://Continue
        "Compare_pos"(Compare_1 := "Control".robot_target,
                      Compare_2 :=
"Robot_coordinates_IO".Coordinates_in,
                      Compare_status => #Comp_status);
        IF #Comp_status = TRUE THEN
          #Comp_status := False;
```

```
            "Control".case_number := 1;
            "Control".current_part += 1;
            "HMI_tags".Brojac_gajba += 1;
            "HMI_tags".Brojac_serija += 1;
            "HMI_tags".Brojac_smjena += 1;
        END_IF;
    END_CASE;
END_IF;
```

f)

```
//Manually control mechanisms from HMI
"Communication_IO".dalex_toggle := "Manual_DB".Dalex;
"Communication_IO".dodavac_toggle :=
"Manual_DB".Dodavac;
"Communication_IO".block_toggle := "Manual_DB".Block;
"Communication_IO".hold_toggle := "Manual_DB".Hold;
"Communication_IO".push_toggle := "Manual_DB".Push;
"Communication_IO".lock_toggle := "Manual_DB".Lock;
"Communication_IO".gripper_toggle :=
"Manual_DB".gripper_toggle;
//Manually start curve follow
"Communication_IO".Load_Dalex_toggle :=
"Manual_DB".Load_Dalex;
"Communication_IO".Dalex_Load_toggle :=
"Manual_DB".Dalex_Load;
"Communication_IO".Dalex_Camera_toggle :=
"Manual_DB".Dalex_Camera;
"Communication_IO".Camera_Dalex_toggle :=
"Manual_DB".Camera_Dalex;
"Communication_IO".Camera_Unload_toggle :=
"Manual_DB".Camera_Unload;
"Communication_IO".Unload_Camera_toggle :=
"Manual_DB".Unload_Camera;
//Manually toggle reference frames
"Communication_IO".Main_frame_toggle :=
"Manual_DB".Main_frame;
"Communication_IO".Unload_frame_toggle :=
"Manual_DB".Unload_frame;
"Communication_IO".Load_frame_toggle :=
"Manual_DB".Load_frame;
"Communication_IO".Curve_frame_toggle :=
"Manual_DB".Curve_frame;
"Communication_IO".UR10_frame_toggle :=
"Manual_DB".UR10_frame;
"Communication_IO".Dodavac_frame_toggle :=
"Manual_DB".Dodavac_frame;
"Communication_IO".Dalex_frame_toggle :=
"Manual_DB".Dalex_frame;
"Communication_IO".Feeder_frame_toggle :=
"Manual_DB".Feeder_frame;
"Communication_IO".Block_frame_toggle :=
"Manual_DB".Block_frame;
//Manually control robot
IF "Manual_DB".init = FALSE THEN
    "Manual_DB".robot_target :=
"Robot_coordinates_IO".Coordinates_in;
    "Manual_DB".init := TRUE;
END_IF;
"Compare_pos"(Compare_1:="Robot_coordinates_IO".Coordin
ates_in,
            Compare_2:="Manual_DB".robot_target,
            Compare_status=>#comp_status);
IF "Manual_DB"."tx+" = TRUE AND #comp_status=TRUE
THEN
    #comp_status := FALSE;
    "Manual_DB".robot_target :=
"Robot_coordinates_IO".Coordinates_in;
    "Manual_DB".robot_target[0] += 50;
    "Robot_coordinates_IO".Coordinates_out
:="Manual_DB".robot_target;
    "Communication_IO".robot_goto_toggle := TRUE;
END_IF;
IF "Manual_DB"."tx-" = TRUE AND #comp_status = TRUE
THEN
    #comp_status := FALSE;
    "Manual_DB".robot_target :=
"Robot_coordinates_IO".Coordinates_in;
    "Manual_DB".robot_target[0] -= 50;
    "Robot_coordinates_IO".Coordinates_out :=
"Manual_DB".robot_target;
    "Communication_IO".robot_goto_toggle := TRUE;
END_IF;
IF "Manual_DB"."ty+" = TRUE AND #comp_status = TRUE
THEN
    #comp_status := FALSE;
    "Manual_DB".robot_target :=
"Robot_coordinates_IO".Coordinates_in;
    "Manual_DB".robot_target[1] += 50;
    "Robot_coordinates_IO".Coordinates_out :=
"Manual_DB".robot_target;
    "Communication_IO".robot_goto_toggle := TRUE;
END_IF;
IF "Manual_DB"."ty-" = TRUE AND #comp_status = TRUE
THEN
    #comp_status := FALSE;
    "Manual_DB".robot_target :=
"Robot_coordinates_IO".Coordinates_in;
    "Manual_DB".robot_target[1] -= 50;
    "Robot_coordinates_IO".Coordinates_out :=
"Manual_DB".robot_target;
```

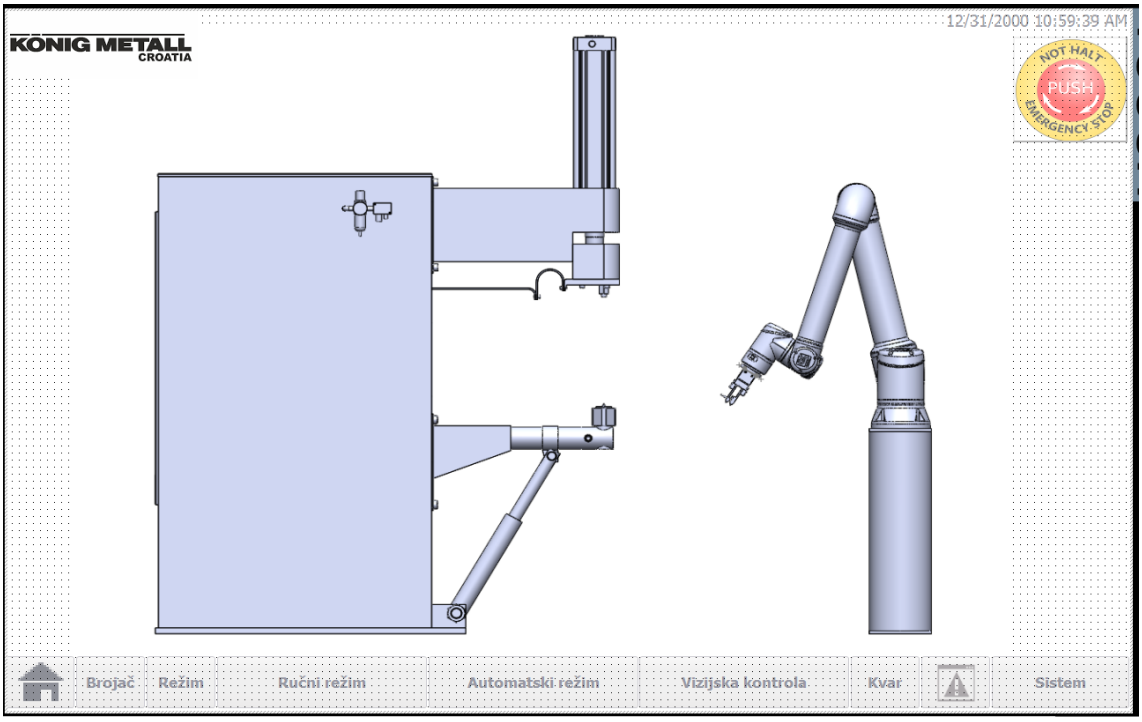
```
   "Communication_IO".robot_goto_toggle := TRUE;
END_IF;
IF "Manual_DB"."tz+" = TRUE AND #comp_status = TRUE
THEN
   #comp_status := FALSE;
   "Manual_DB".robot_target :=
"Robot_coordinates_IO".Coordinates_in;
   "Manual_DB".robot_target[2] += 50;
   "Robot_coordinates_IO".Coordinates_out :=
"Manual_DB".robot_target;
   "Communication_IO".robot_goto_toggle := TRUE;
END_IF;
IF "Manual_DB"."tz-" = TRUE AND #comp_status = TRUE
THEN
   #comp_status := FALSE;
   "Manual_DB".robot_target :=
"Robot_coordinates_IO".Coordinates_in;
   "Manual_DB".robot_target[2] -= 50;
   "Robot_coordinates_IO".Coordinates_out :=
"Manual_DB".robot_target;
   "Communication_IO".robot_goto_toggle := TRUE;
END_IF;
IF "Manual_DB"."rx+" = TRUE AND #comp_status = TRUE
THEN
   #comp_status := FALSE;
   "Manual_DB".robot_target :=
"Robot_coordinates_IO".Coordinates_in;
   "Manual_DB".robot_target[5] += 5;
   "Robot_coordinates_IO".Coordinates_out :=
"Manual_DB".robot_target;
   "Communication_IO".robot_goto_toggle := TRUE;
END_IF;
IF "Manual_DB"."rx-" = TRUE AND #comp_status = TRUE
THEN
   #comp_status := FALSE;
   "Manual_DB".robot_target :=
"Robot_coordinates_IO".Coordinates_in;
   "Manual_DB".robot_target[5] -= 5;
   "Robot_coordinates_IO".Coordinates_out :=
"Manual_DB".robot_target;
   "Communication_IO".robot_goto_toggle := TRUE;
END_IF;
IF "Manual_DB"."ry+" = TRUE AND #comp_status = TRUE
THEN
   #comp_status := FALSE;
   "Manual_DB".robot_target :=
"Robot_coordinates_IO".Coordinates_in;
   "Manual_DB".robot_target[4] += 5;
   "Robot_coordinates_IO".Coordinates_out :=
"Manual_DB".robot_target;
   "Communication_IO".robot_goto_toggle := TRUE;
END_IF;
IF "Manual_DB"."ry-" = TRUE AND #comp_status = TRUE
THEN
   #comp_status := FALSE;
   "Manual_DB".robot_target :=
"Robot_coordinates_IO".Coordinates_in;
   "Manual_DB".robot_target[4] -= 5;
   "Robot_coordinates_IO".Coordinates_out :=
"Manual_DB".robot_target;
   "Communication_IO".robot_goto_toggle := TRUE;
END_IF;
IF "Manual_DB"."rz+" = TRUE AND #comp_status = TRUE
THEN
   #comp_status := FALSE;
   "Manual_DB".robot_target :=
"Robot_coordinates_IO".Coordinates_in;
   "Manual_DB".robot_target[3] += 5;
   "Robot_coordinates_IO".Coordinates_out :=
"Manual_DB".robot_target;
   "Communication_IO".robot_goto_toggle := TRUE;
END_IF;
IF "Manual_DB"."rz-" = TRUE AND #comp_status = TRUE
THEN
   #comp_status := FALSE;
   "Manual_DB".robot_target :=
"Robot_coordinates_IO".Coordinates_in;
   "Manual_DB".robot_target[3] -= 5;
   "Robot_coordinates_IO".Coordinates_out :=
"Manual_DB".robot_target;
   "Communication_IO".robot_goto_toggle := TRUE;
END_IF;
IF "Manual_DB".get_position = TRUE THEN
   "Robot_coordinates_IO".Coordinates_out :=
"Robot_coordinates_IO".Coordinates_in;
END_IF;
```

g)

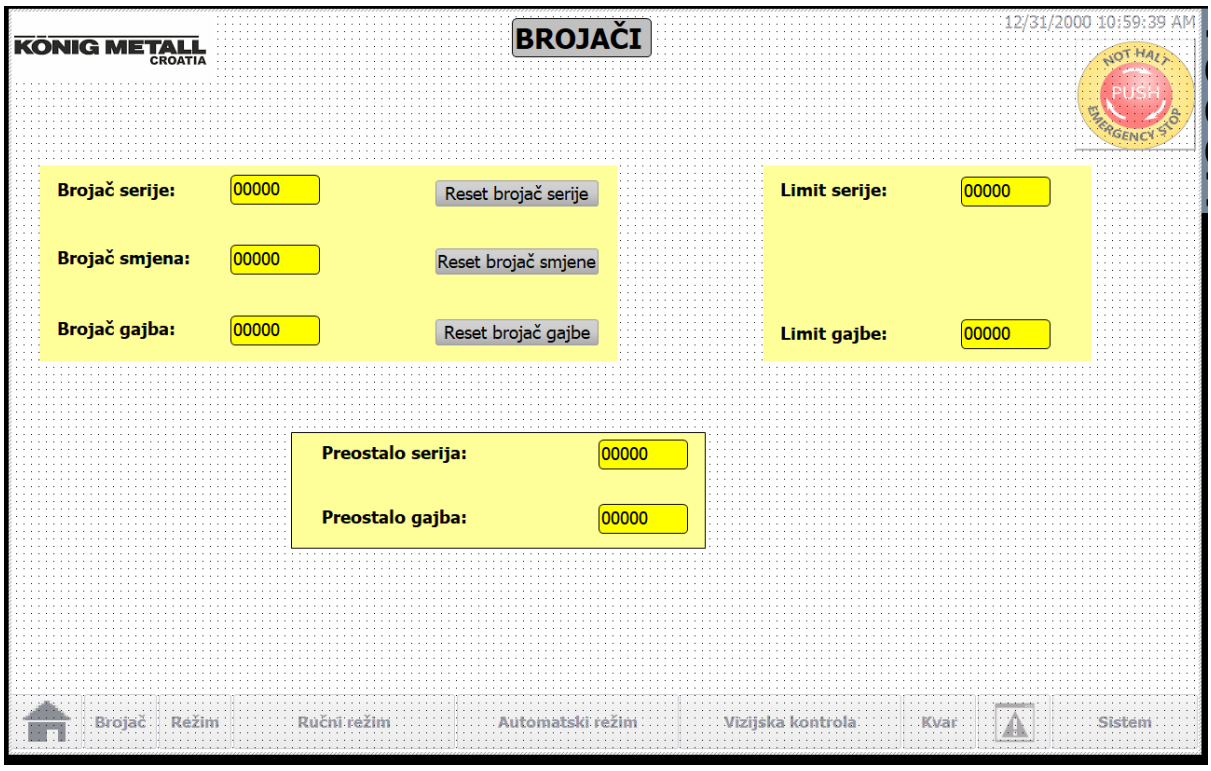
```
//Init temp variable
#Comp1[0] := #Compare_1[0];
#Comp1[1] := #Compare_1[1];
#Comp1[2] := #Compare_1[2];
#Comp2[0] := #Compare_2[0];
#Comp2[1] := #Compare_2[1];
#Comp2[2] := #Compare_2[2];
IF #Comp1 = #Comp2 THEN
   #Compare_status := TRUE;
ELSE
   #Compare_status := FALSE;
END_IF;
"Test".Test_1 := #Compare_status;
```

h)

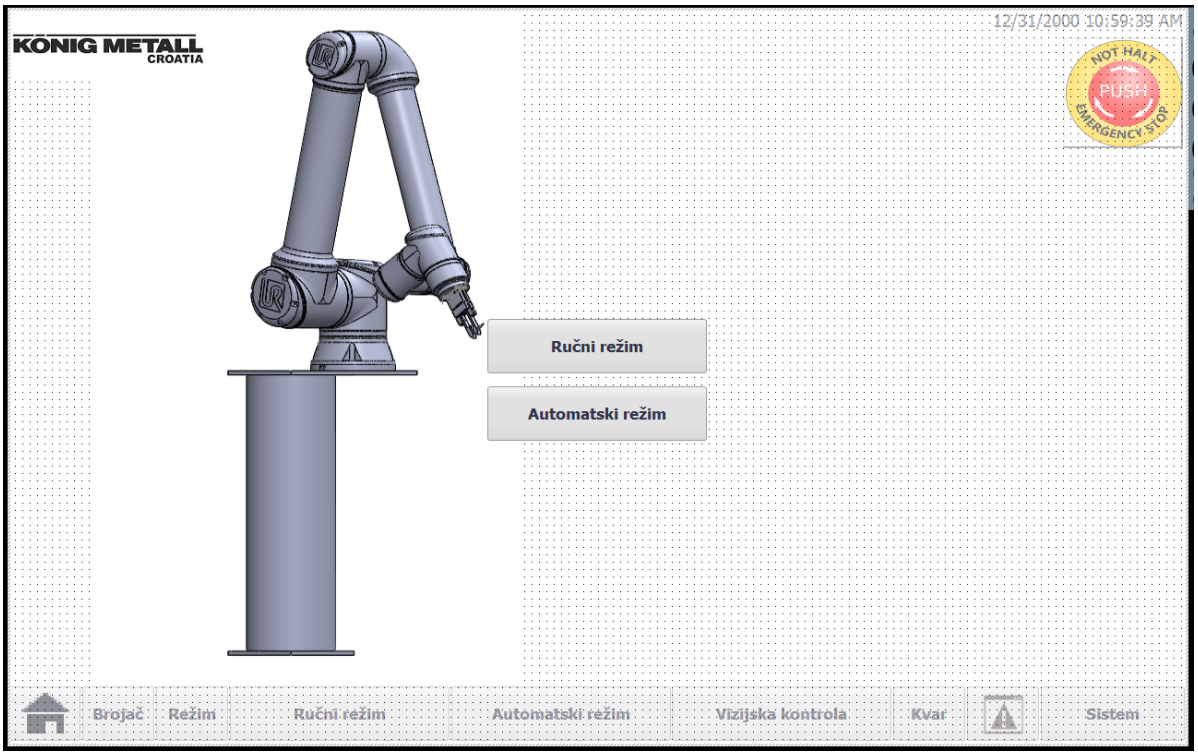
Početni ekran



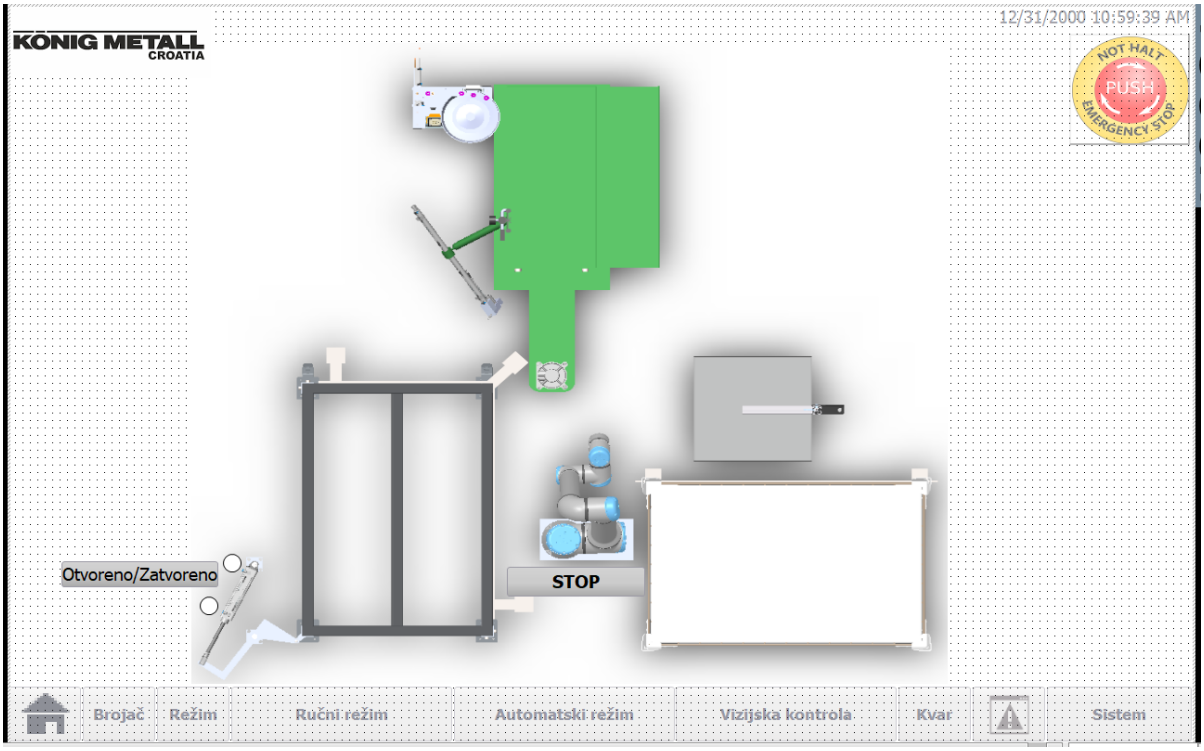
Ekran brojača



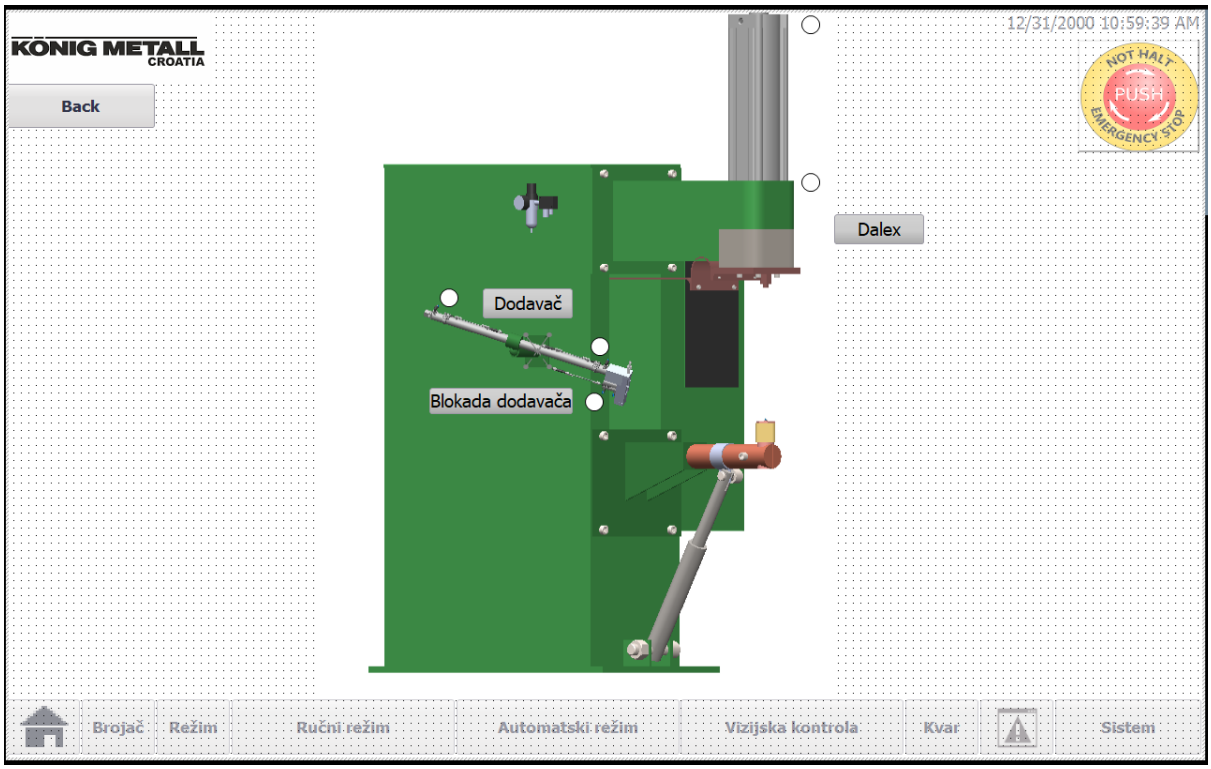
Ekran odabira režima rada



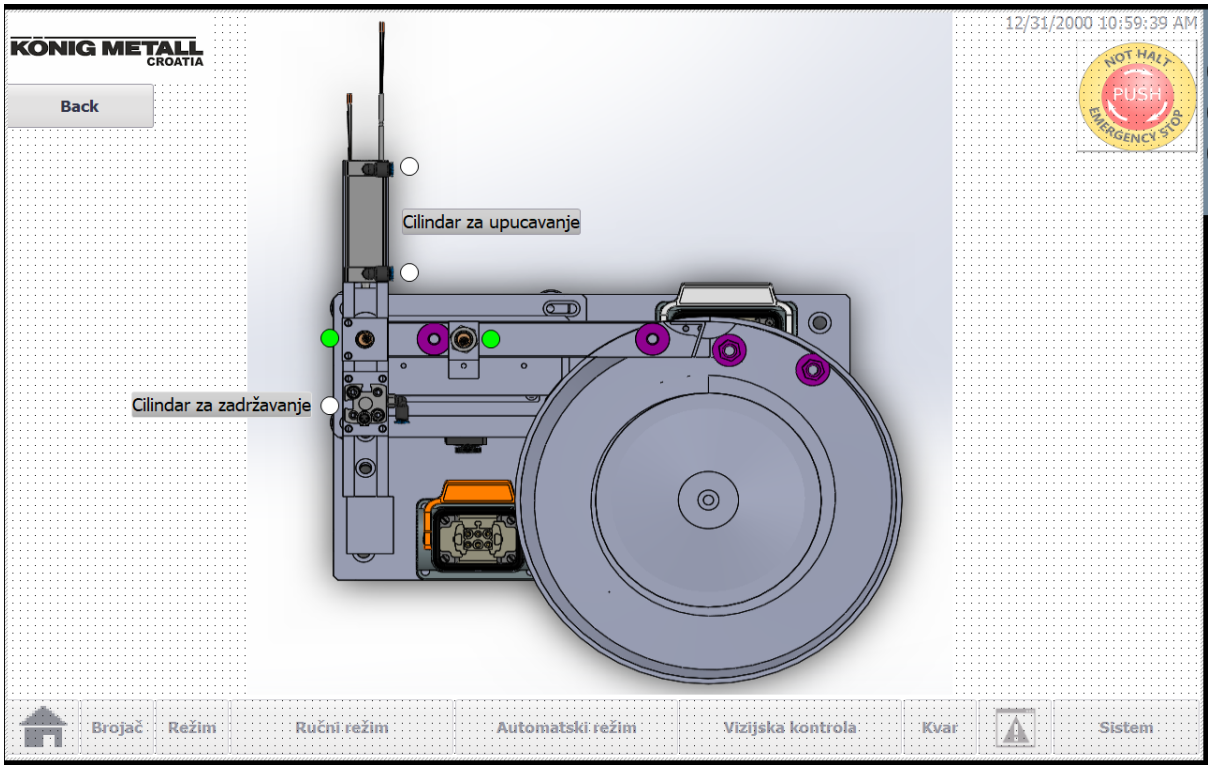
Početni ekran ručnog režima rada



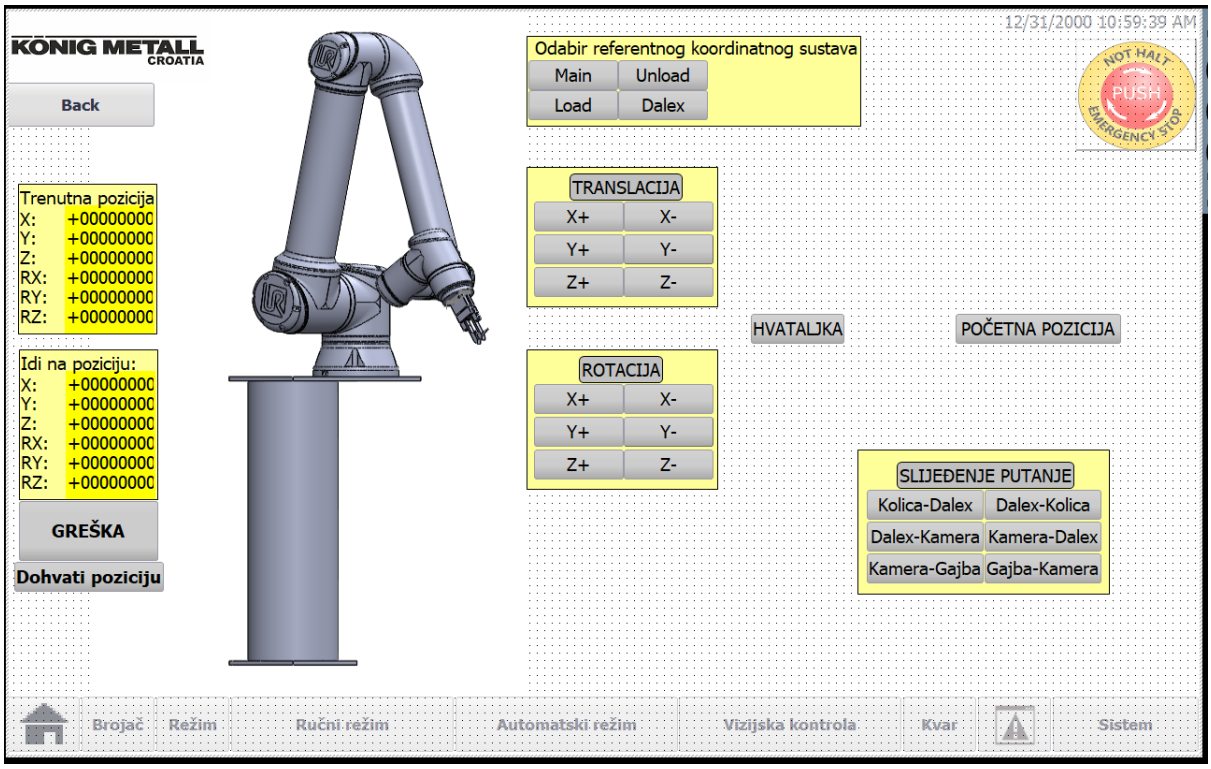
Ekran ručnog režima rada – Dalex



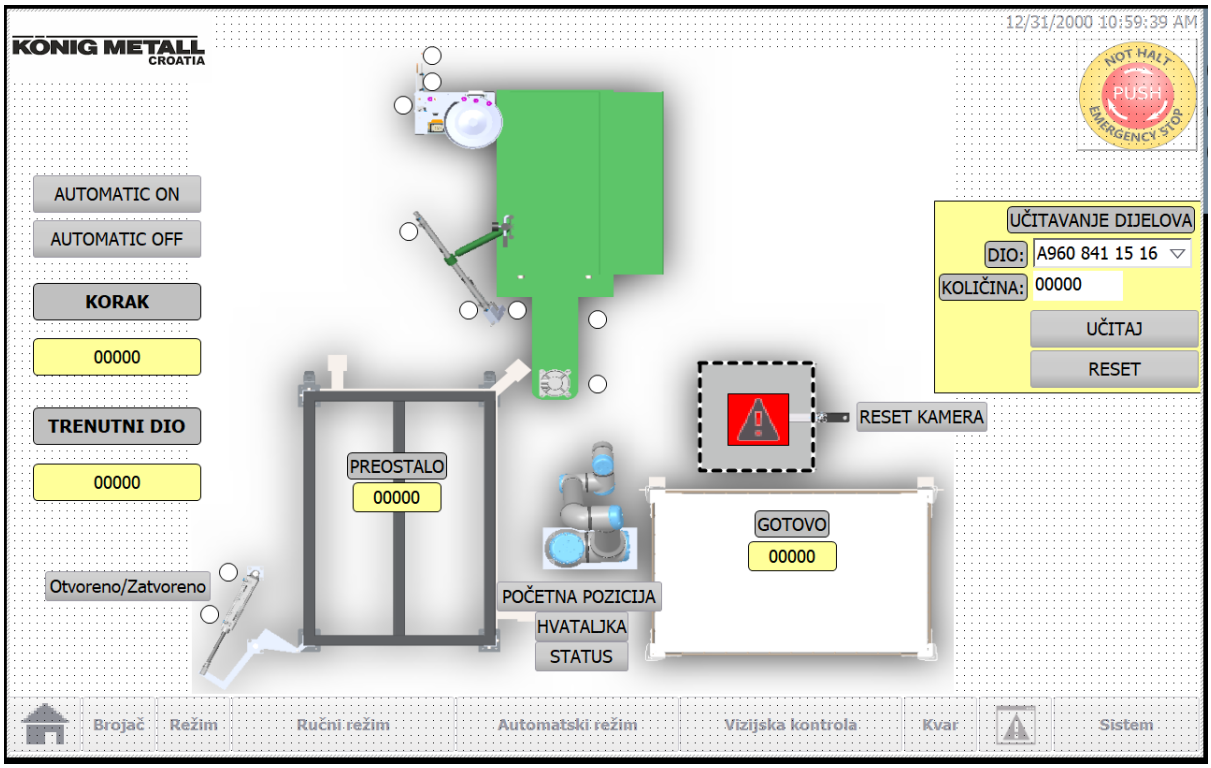
Ekran ručnog režima rada – vibrododavač



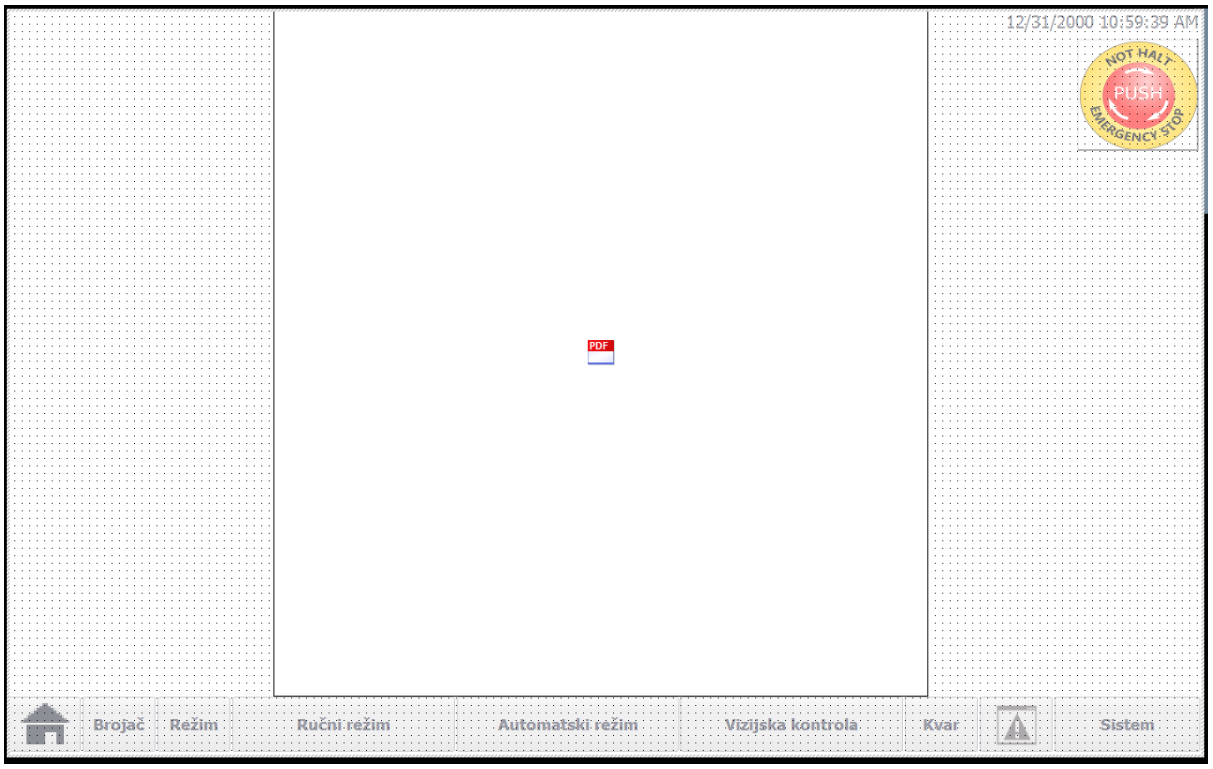
Ekran ručnog režima rada – UR10e



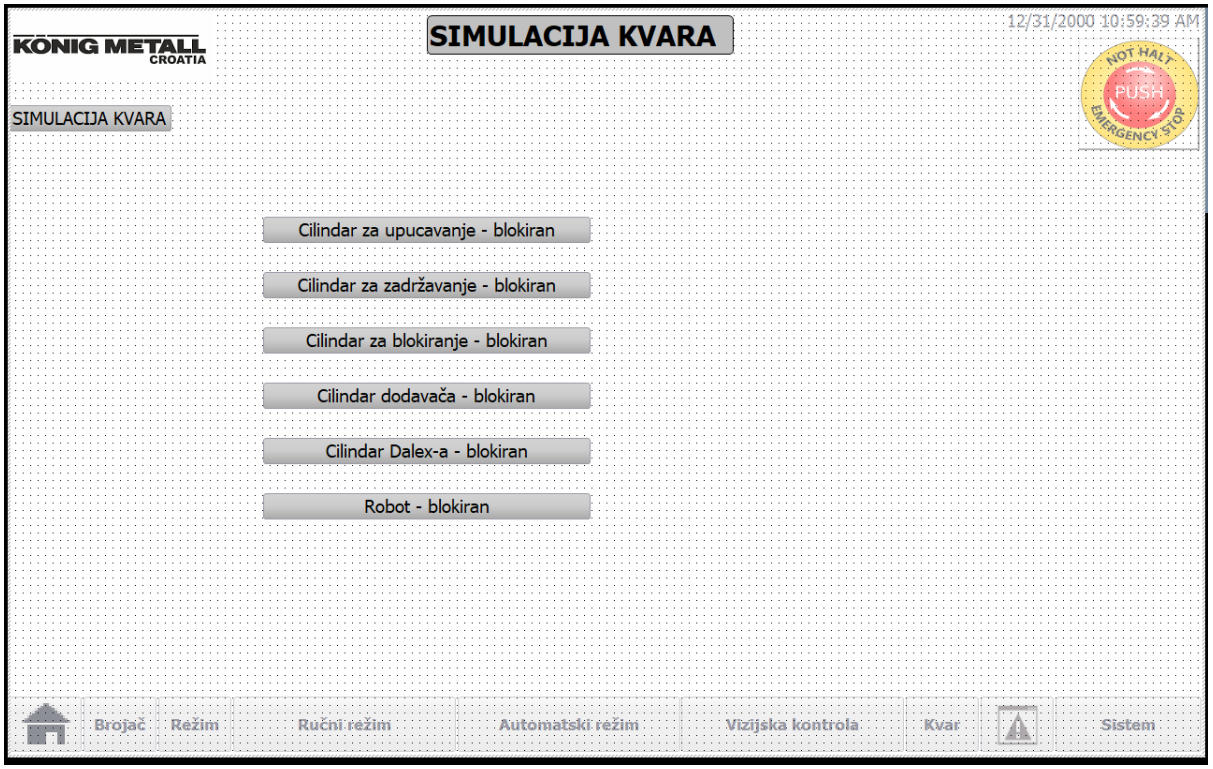
Ekran automatskog režima rada



Ekran prikaza vizijske kontrole



Ekran simulacije kvara



i)

Rn. broj	Rasipanje rezultata radijusa	Rasipanje rezultata y-os - prva matica	Rasipanje rezultata y-os - druga matica
1.	25	311	314
2.	23	311	316
3.	24	313	311
4.	26	309	316
5.	26	305	317
6.	28	306	314
7.	23	307	321
8.	28	311	301
9.	21	306	303
10.	26	302	317
11.	26	299	321
12.	20	303	317
13.	21	311	322
14.	27	302	309
15.	22	306	298
16.	23	308	320
17.	25	312	304
18.	27	312	307
19.	25	299	322
20.	28	301	306
21.	26	305	314
22.	24	304	308
23.	21	307	316
24.	20	309	311
25.	25	309	314
26.	24	300	322
27.	25	299	314
28.	27	309	301
29.	21	308	302
30.	21	304	313
31.	26	312	315
32.	26	309	310
33.	27	302	305
34.	26	307	310
35.	21	312	315
36.	27	307	312
37.	28	302	317
38.	20	307	299
39.	28	300	319

40.	27	303	321
41.	22	307	319
42.	22	307	308
43.	20	303	312
44.	21	310	302
45.	21	307	311
46.	22	310	305
47.	22	303	316
48.	25	309	307
49.	26	298	307
50.	20	298	317
51.	25	298	316
52.	28	311	320
53.	21	307	322
54.	25	306	314
55.	25	298	310
56.	24	304	310
57.	21	308	306
58.	27	309	302
59.	23	306	302
60.	24	311	303
61.	20	305	313
62.	22	312	305
63.	23	312	309
64.	21	301	319
65.	26	301	304
66.	27	297	298
67.	21	301	322
68.	28	309	303
69.	27	308	317
70.	23	304	300
71.	27	312	305
72.	21	308	314
73.	27	309	305
74.	22	303	300
75.	21	297	301
76.	28	297	305
77.	24	297	312
78.	23	310	299
79.	27	297	301
80.	26	298	312
81.	22	304	300
82.	26	308	320
83.	26	311	307

84.	27	298	309
85.	26	298	309
86.	28	308	307
87.	26	302	314
88.	24	300	308
89.	25	310	317
90.	28	311	313
91.	22	313	304
92.	24	301	302
93.	22	306	303
94.	20	304	321
95.	20	305	313
96.	28	299	308
97.	27	301	322
98.	22	306	304
99.	24	311	314
100	24	297	316