SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Damir Mirković

Razvoj i implementacija višerobotskog sustava za novi robotički laboratorij

Zagreb, 2015.

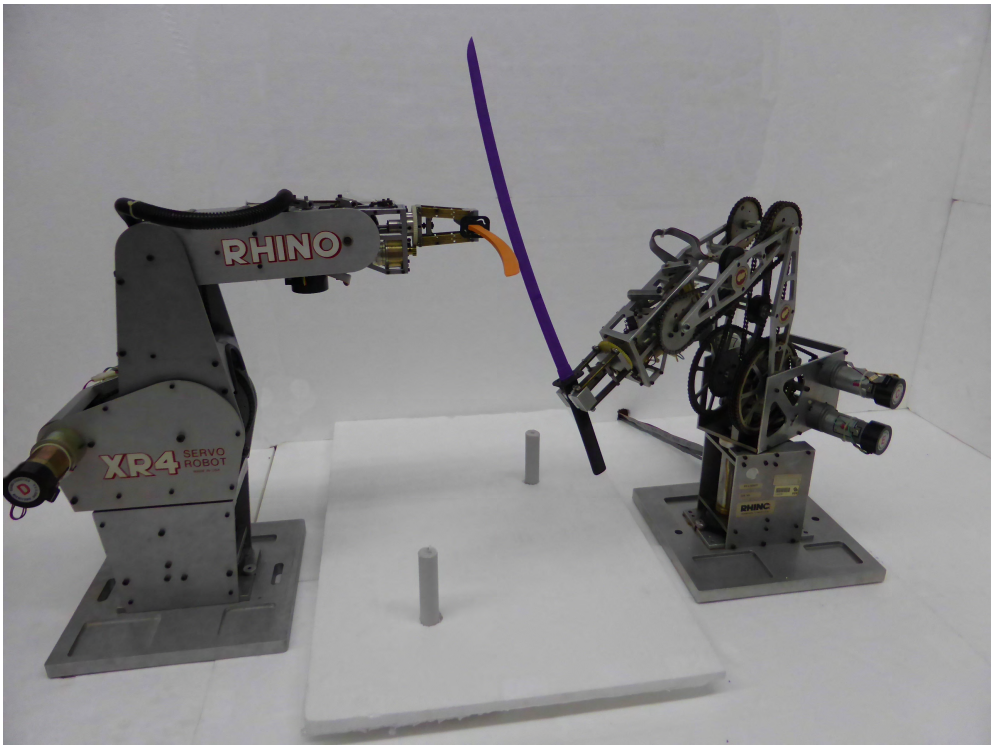
Ovaj rad izrađen je u Laboratoriju za robotiku i inteligentne sustave upravljanja pod vodstvom Prof. dr. sc. Zdenka Kovačića i predan je na natječaj za dodjelu Rektorove nagrade u akademskoj godini 2014./2015.

SADRŽAJ

1. Uvod

Praktičan rad neizostavan je dio obrazovnog procesa studija u području tehničkih znanosti. Izrada ovog rada potaknuta je uočenom potrebom za eksperimentalnom ispitnom platformom usko povezanom s *Matlab* i *Simulink* programskim paketima, alatima u potpunosti integriranima u postojeće programe kolegija Fakulteta elektrotehnike i računarstva. Eksperimentalni sustav pruža mogućnost primjene stečenih teorijskih znanja u poznatom programskom okruženju, na stvarnom sustavu te kritičkog prosuđivanja prednosti i nedostataka pojedinih metoda u realnim uvjetima. Posebna pozornost posvećena je modularnosti i naklonjenost rješenjima otvorenog koda (*Real-time Linux* operacijski sustav, *Simulink real-time target*) te korištenju pouzdane, mogućnostima bogate opreme. Rezultat je sustav koji, uz minimalne izmjene i nadogradnje prati korak dinamične obrazovne sredine, i spreman je prihvatiti nove standarde sklopovske i programske podrške. Primjerice, podrška proizvođača je za korištene *Roboteq* servokontrolere osigurana godinama nakon isporuke te su po obavljenoj nadogradnji mogućnostima gotovo jednaki aktualnoj verziji kontrolera.
Također, pri izradi rada poštovani su suvremeni standardi industrijske automatike (preporuke Laboratorija za razvoj automatike uz korištenje otvorenog koda (*OSADL*) te *CANOpen* komunikacijski protokol) kako bi se osigurala kvaliteta i stabilnost sustava pri ispitivanju naprednih, i često računalno zahtjevnih, upravljačkih algoritama u stvarnom vremenu. Iako izvorno zamišljen kao okosnica laboratorijskih vježbi kolegija vezanih uz robotiku, pokazao se pogodnim i za praktičnu primjenu upravljačkih algoritama obrađenih na kolegijima automatike zbog otvorenog pristupa najnižim strukturama upravljanja, što u komercijalnim rješenjima najčešće nije slučaj. Zamijećeni nedostaci pri korištenju operacijskog sustava *xPC target* tvrtke *Mathworks* ispravljeni su, a mnoge pozitivne odlike implementirane u vlastiti sustav. Kako je sustav s dva robotska manipulatora namijenjen implementaciji algoritama različite razine, za svaku razinu osigurana su kvalitetna i stabilna rješenja kako bi ispitivanje vlastitih algoritama bilo rasterećeno od problema niže razine.

Sve mogućnosti realiziranih struktura upravljanja u stvarnom vremenu pokazane su implementacijom koordiniranog praćenja složenih trajektorija dva robotska manipulatora te je oživljavanjem scene karakteristična za Japan s kraja 13. stoljeća, odmjeravanje vještine i preciznosti baratanja tradicionalnim japanskim mačem, katanom. (Slika **??**).



Slika 1.1: Mačevanje dva *Rhino* robota

1.1. Opći i specifični ciljevi rada

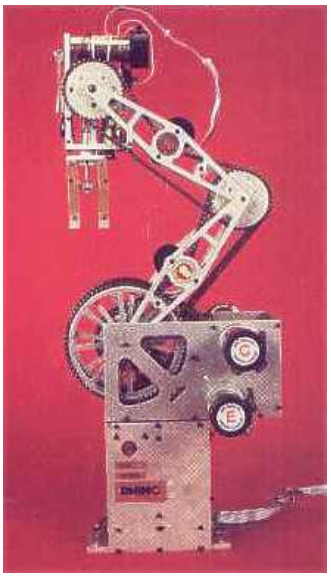
Cilj rada osuvremenjivanje je postojećeg zastarjelog sustava, cjelokupne sklopovske i programske podrške, koja je prilagođena radu sa suvremenim programskim alatima *Matlab* i *Simulink* te proširuje simulacijske mogućnosti samih alata izvođenjem na stvarnom sustavu u stvarnom vremenu. Razvijeni sustav trebao bi omogućiti realizaciju naprednih laboratorijskih vježbi neovisno o korištenoj opremi. U okviru rada, na izrađenom sustavu, ostvareni su praktični elementi nastave robotike na Fakultetu elektrotehnike i računarstva: upravljanje zglobovima, implementacija direktne i inverzne kinematike te praćenje trajektorije. Strukture upravljanja ostvarene su na edukacijskim robotskim manipulatorima, koji, iako za današnje pojmove zastarjeli, posjeduju golem edukacijski potencijal koji nikada nije u potpunosti iskorišten.

2. Materijal i metode

2.1. Sklopovska podrška

2.1.1. Rhino

Rhino XR-3 (Slika 7.4e) i Rhino XR-4 (Slika 7.4f) tvrtke *Rhino Robotics Ltd.* peteroosni su edukacijski roboti s hvataljkom namijenjeni edukaciji studenata sveučilišne razine.



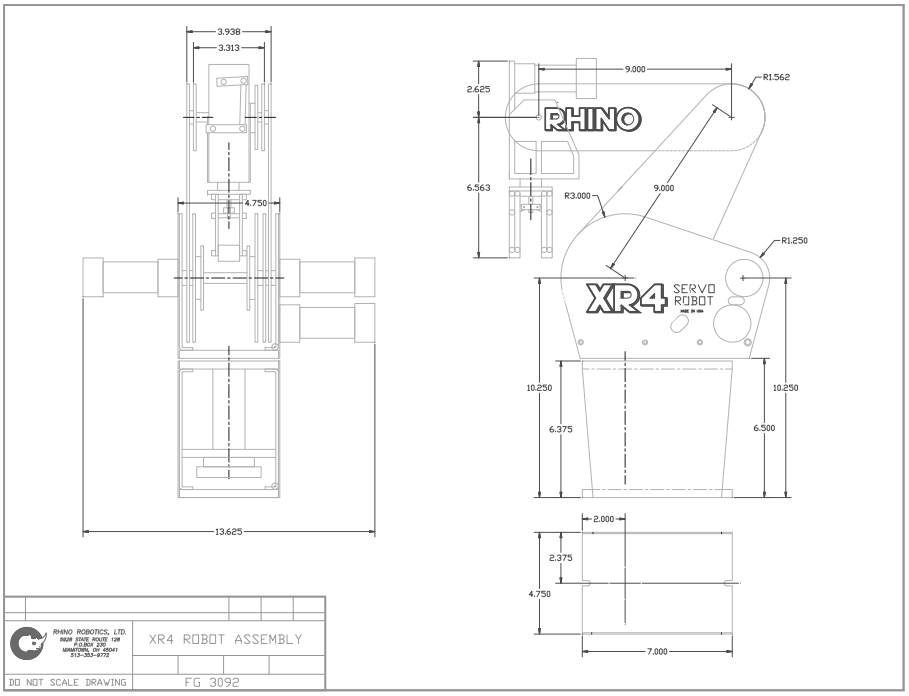
(a) Rhino XR-3

(b) Rhino XR-4

Konstrukcijski aspekti

Rhino XR-3 i *Rhino XR-4* čine baza, tri članka (tijelo, nadlaktica i podlaktica) te alat s hvataljkom, povezani četirima rotacijskim zglobovima (struk, rame, lakat, i zapešće) čime je definirana rotacijska konfiguracija. Radni prostor robota dio je kugle složenog oblika uz maksimalan dohvat u horizontalnoj ravnini od $60.96cm$ (mjereno od osi struka do vrha prstiju alata). Dimenzijski crtež Rhino XR-4 robota 2.2 [4] dostupan je na službenim stranicama *Rhino Robotics Ltd.*, a za pregled crteža potrebno je preuzeti i

instalirati *Autodesk Design Review* preglednik. Jedinice mjere korištene u dimenzijskom crtežu su inči (engl. *inch*).



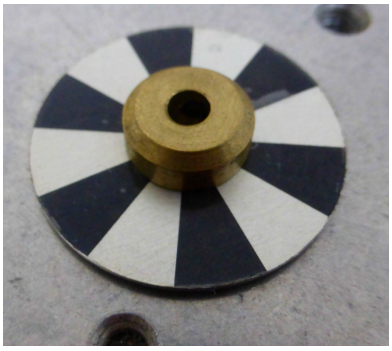
Slika 2.2: Dimenzijski crtež Rhino XR-4 robota

Pogon

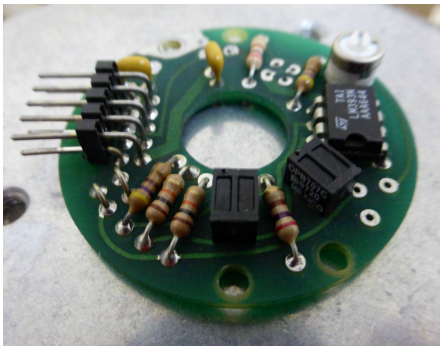
Rhino XR-4 pogoni šest istosmjernih motora tvrtke *Pittman*(četiri $12V$ i dva $24V$). Maksimalna teoretski ostvariva brzina iznosi $40°/s$. [3]

Enkoderi

Na svakom od šest motora Rhino XR-4 robota nalazi se inkrementalni optički enkoder. Optički diskovi na malim motorima (A i B) podijeljeni su na šest segmenata: tri srebrna i tri crna, dok su optički diskovi na velikim motorima (C,D,E,F) podijeljeni na dvanaest segmenata (Slika **??**). Na optičkoj pločici smještena su dva optička senzora (Slika **??**), fazno pomaknuta za $90°$ čime je omogućena detekcija smjera rotacije, što je preduvjet za izračun pozicije, brzine i akceleracije. [3, 9]



(a) Optički disk



(b) Optički senzor

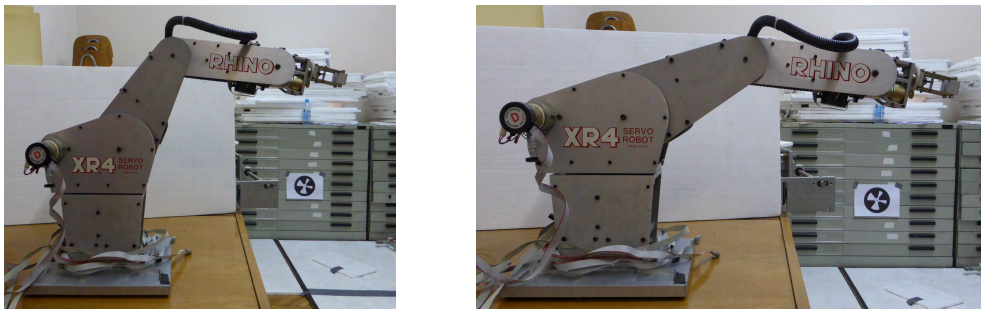
Parametri enkodera dani su tablicom 2.1

Tablica 2.1: Signali na enkoderu prvog motora)

Motor	Zglob	Redukcija	Broj impulsa po okretu
C	Zapešće	524 : 1	12576
D	Lakat	524 : 1	12576
E	Rame	524 : 1	12576
F	Struk B	262 : 1	6288

2.1.2. Prijenos

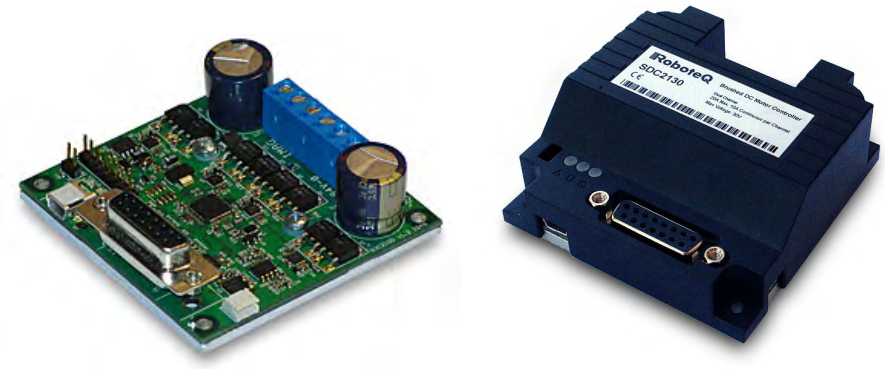
Lančani prijenos robota projektiran je tako da je kut nadlaktice u odnosu na mirni koordinatni sustav baze neovisan o kutu zakreta podlaktice (Slika **??**), iz čega slijedi da se kut zgloba mijenja ovisno u kutu nadređenog članka, neovisno o zakretu motora koji upravlja zglobom. Navedena konstrukcijska specifičnost uvelike otežava upravljanje u prostoru koordinata zglobova.



Slika 2.4: Svojstva prijenosa

2.1.3. Servokontroleri

Servokontroleri *SDC2150N* (Slika 2.5a) i *SDC2160N* (Slika 2.5a) tvrtke *Roboteq* namijenjeni su pretvorbi naredbi primljenih s računala (RS232, USB ili CAN komunikacija) ili mikroračunala u izlazni signal visokog napona ili struje.



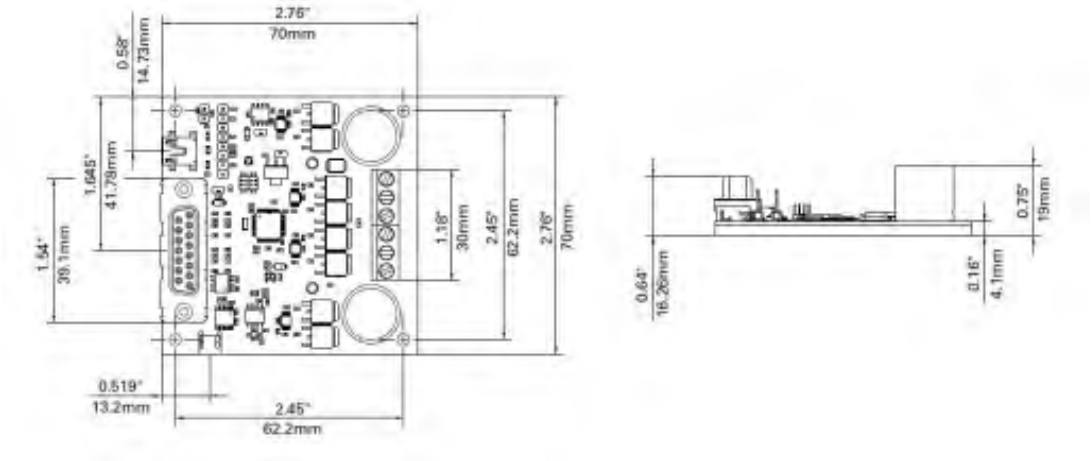
(a) Roboteq SDC2150N **(b)** Roboteq SDC2160N

Tehnički podaci Roboteg SDC2150N i SDC2160N servokontrolera dani su u tablici 5.1.1. [10]

Tablica 2.2: Tehnički podaci servokontrolera

servokontroler	SDC2150N	SDC2160N
maksimalan nereverziran napon napajanja $\mathbf{U_{s,MAX}}$	$55V$	$62V$
minimalan nereverziran napon napajanja $\mathbf{U_{s,min}}$	$10V$	
minimalan reverziran napon napajanja $\mathbf{U_{r,min}}$	$-1V$	
maksimalan izlazni napon $\mathbf{U_{m,MAX}}$	$55V$	$62V$
maksimalan napon digitalnih izlaza $\mathbf{U_{do,MAX}}$	$30V$	$40V$
maksimalan napon digitalnih i analognih ulaza $\mathbf{U_{r,min}}$	$-1V$	
maksimalna radna temperatura $\mathbf{T_{MAX}}$	$85°C$	
minimalna radna temperatura $\mathbf{T_{min}}$	$-40°C$	
maksimalna izlazna struja (do $30s$**)** $\mathbf{I_{MAX}}$	$20A$	
maksimalna kontinuirana izlazna struja $\mathbf{I_{cont}}$	$15A$	
minimalna brzina izvođenja *MicroBasic* programa $\mathbf{es_{MB}}$	$50000lines/s$	

Dimenzijski crtež prikazan je slikom 2.6.



Slika 2.6: Dimenzije kontrolera SDC 21x0N (70 mm x 70 mm x 19 mm)

2.1.4. CAN mrežno sučelje

CAN (*Controller Area Network*) je protokol baziran na

2.1.5. Ciljno računalo

2.2. Programska podrška

2.2.1. Matlab

MATLAB je jezik visoke razine i interaktivno okruženje namijenjeno numeričkim izračunima. Omogućuje analizu podataka, razvoj algoritama i izradu modela i aplikacija. Jezik, alati i ugrađene matematičke funkcije pružaju mogućnost isprobavanja različitih pristupa i brži pronalazak rješenja u odnosu na tradicionalne programske jezike (*C/C++*,*Java*). *Matlab* se koristi u nizu primjena, uključujući obradu signala, obradu slike i videa, razvoj upravljačkih sustava, testiranje i mjerenje, financijske izračune i izračune u biologiji. [5]

2.2.2. Simulink

Simulink je grafički programski alat za modeliranje, simulaciju i analizu dinamičkih sustava. Osnovno sučelje Simulinka je grafički alat temeljen na blokovskim dijagramima te skup blokovskih biblioteka. Omogućuje usku integraciju s ostatkom *MATLAB* okruženja. [7]

2.2.3. 3ds Max

Autodesk 3ds Max profesionalni je 3D računalni grafički alat za izradu 3D animacija, modela, igara i slika. Proizvod je *Autodesk Media and Etertainment*. Pruža velike mogućnosti pri modeliranju, posjeduje fleksibilnu arhitekturu dodataka, a korištenje je ograničeno na *Microsoft Windows* platformu.

2.2.4. Upravljanje u stvarnom vremenu

Upravljanje u stvarnom vremenu podrazumijeva poznavanje vremena uzorkovanja, odnosno postojanje vremenskog roka za izvođenje programskog koda koji odgovara na neku pobudu u sustavu. Kako bi se zadovoljili zahtjevi vremenskih rokova koriste se *real-time* operacijski sustavi (RTOS) za koje je moguće pouzdano odrediti maksimalno vrijeme izvođenja za zadanu okolinu i programski odsječak koji se u toj okolini izvodi.

Real-time operacijski sustavi tipično koriste prioritete. Zahtjev zadatka najvišeg prioriteta prema središnjoj procesnoj jedinici (CPU) uvijek se prihvaća unutar fiksnog

vremenskog intervala od trenutka postavljanja zahtjeva. U takvom *RTOS*-u kašnjenje zadatka ovisi samo o zadacima istog ili višeg prioriteta koji se izvode u okolini, dok svi zadaci nižeg prioriteta mogu biti zanemareni.[1]

Upravljanje u stvarnom vremenu osnova je upravljačkih sustava u kojima je prikupljanje i obrada podataka te osvježavanja izlaznih signala sustava unutar uskog vremenskog okvira od posebnog značaja.

2.2.5. Real-time Linux kernel

Standardne distribucije *Linux kernela* ne osiguravaju ograničena kašnjenja za velik broj operacija. Upotreba *real-time* inačice *Linux kernela* neophodna je kako bi se kašnjenje zadatka najvišeg prioriteta svelo na minimum. Navedena inačica *kernela* minimizira područja u kojima je izvođenje zadatka višeg prioriteta spriječeno od strane *kernela* za vrijeme izvođenja zadataka nižeg prioriteta.

Pri izradi rada korišten je dodatak `CONFIG_PREEMPT_RT` čijom primjenom na izvorni kod *Linux* jezgre nastaje *Real-time* jezgra operacijskog sustava.

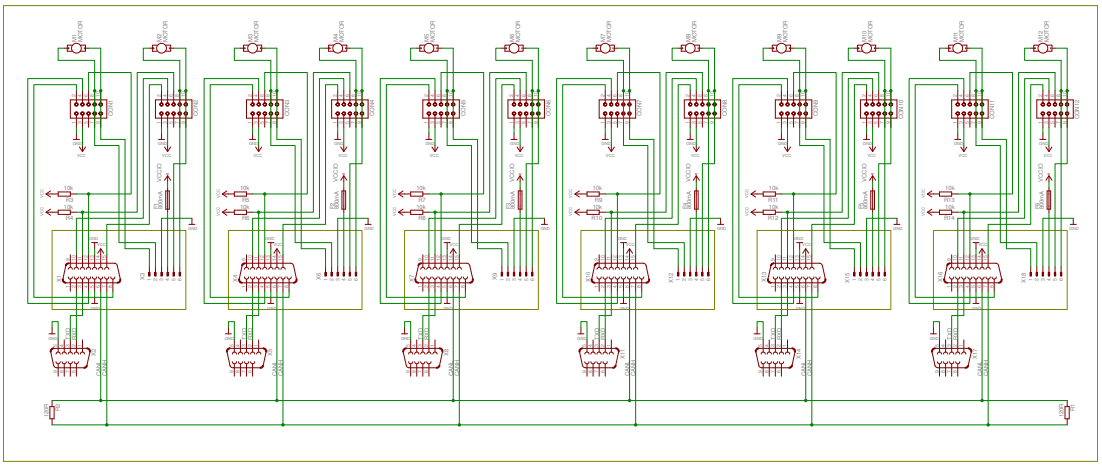
3. Eksperimentalna razvojna platforma

3.1. Roboteq

Roboteq SDC2150N i *SDC2160N* servokontroleri su visokih performansi posebno zanimljivi zbog mogućnosti ostvarivanja vlastitih algoritama upravljanja na samom kontroleru korištenjem *MicroBasic* skripti te podržane komunikacije preko *CAN* protokola.

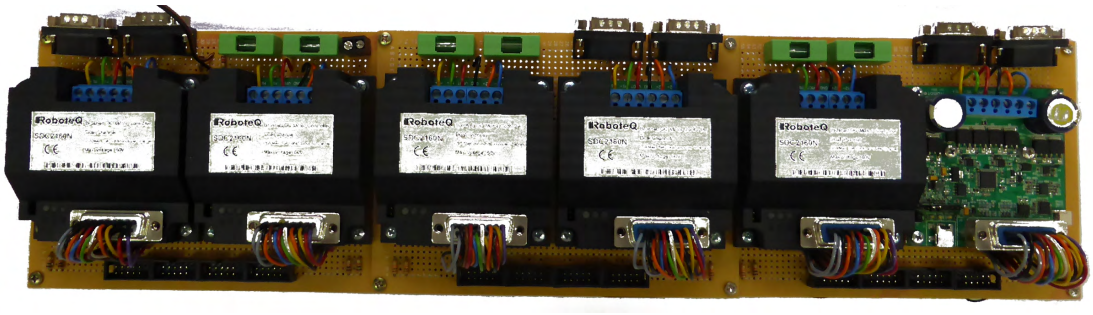
3.1.1. Sklopovska podrška

Kako bi se osigurala pouzdana sklopovska okolina za razvoj sustava, izrađena je modularna podloga kompatibilna s *Roboteq* kontrolerima te *Rhino XR-3* i *Rhino XR-4* robotskim manipulatorima. Na temelju podataka o pinovima *DB15* konektora kontrolera [10] te podataka o pinovima konektora *Mark III* i *Mark IV* [3] kontrolera koji su predviđeni da se na njih spoje *Rhino XR-3* i *Rhino XR-4* robotski manipulatori u programskom paketu *Eagle* napravljena je električna shema modularne podloge prikazana slikom **??**.



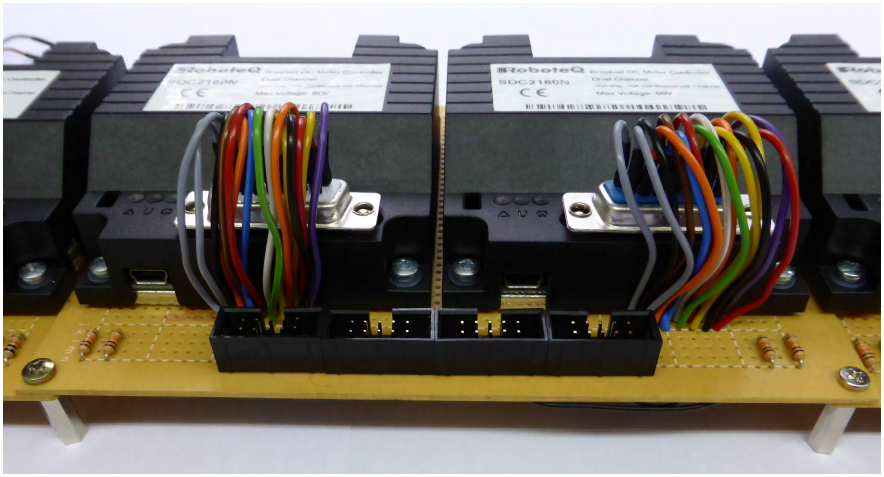
Slika 3.1: Električna shema modularne podloge

Podloga realizirana na temelju električne sheme prikazana je slikom 3.2.

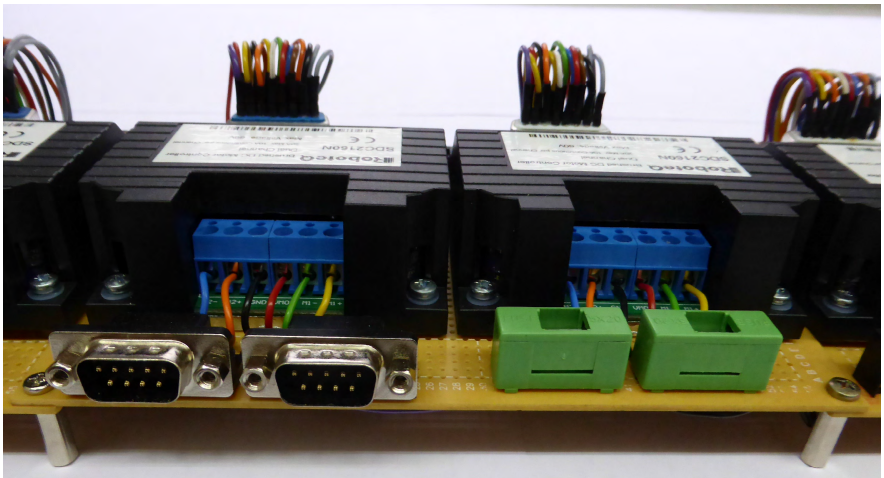


Slika 3.2: Podloga

Pregled konektora prednje i stražnje strane dan je slikama 3.3a i 3.3b.



(a) Konektori prednje strane



(b) Konektori stražnje strane

3.1.2. Konfiguracija

Prije korištenja kontrolera potrebno je omogućiti opcije potrebne za komunikaciju s kontrolerom i upravljanje robotskim manipulatorima.
Sama konfiguracija obavlja se unutar *Roborun+* aplikacije. [11]
Promijenjene postavke unutar sučelja *Roborun+* aplikacije prikazane su slikom **??**.

3.1.3. MicroBasic skripte

Postojeće mogućnosti *Roboteq* kontrolera proširene su korištenjem *MicroBasic* skripti. Na samom kontroleru ostvaren je debouncing filter signala sa sklopki robota, mapiranje unutarnjih varijabli kontrolera na *CANopen* poruke te spremanje *home* pozicije za vrijeme i onemogućavanje okidanja sklopki nakon inicijalizacije robota.

4. Programska platforma

4.1. Real-Time Linux

Preduvjet upravljanju u stvarnom vremenu je ciljno računalo na koje je instaliran *real-time* operacijski sustav.

Odabir operacijskog sustava

Postoji velik broj komercijalno dostupnih *real-time* operacijskih sustava. Razmotreni sustavi, odgovarajućih karakteristika, prikazani su tablicom **??**, gdje su navedene njihove prednosti i nedostaci.

Tablica 4.1: Prednosti i nedostaci *real-time* operacijskih sustava

xPC target	**prednosti**
	Real-time operacijski sustav tvrtke *Mathworks* i kao takav u potpunosti integriran u programske pakete *Matlab* i *Simulink* istog proizvođača Iskustvo u radu s *real-time* sustavima temeljenim na *xPC targetu*
	nedostaci
2-2	Nepostojanje podrške za korišteno *CAN* mrežno sučelje Dostupne upravljačke programe nije moguće preraditi za *xPC target*
QNX	**prednosti**
	Dostupni su upravljački programi za korištenu *IXXAT iPC-I 165/PCI CAN* mrežnu karticu
	nedostaci
	Neslužbena podrška za *Simulink Coder* izvorno podržava isključivo *ARM* arhitekturu te su potrebne znatne izmjene kako bi se se prilagodila *Intel* arhitekturi korištenog ciljnog računala
RT Linux	**prednosti**
	Dostupni su upravljački programi za korištenu *IXXAT iPC-I 165/PCI CAN* mrežnu karticu Neslužbena podrška za *Simulink Coder* izvorno podržava *Intel* arhitekturu korištenog ciljnog računala Otvorena platforma za koju postoje upravljački programi velikog broja komponenti što pruža mogućnost kasnijih izmjena sustava
	nedostaci
	Podrška za *Simulink Coder* izvorno zahtijeva *Linux host* računalo

Od razmotrenih *real-time* operacijskih sustava, *Real-time Linux* u najvećoj mjeri zadovoljava postavljene zahtjeve te ostavlja najviše mogućnosti za buduće primjene realiziranog sustava.

4.1.1. OSADL posljednja stabilna verzija

Open Source Automation Development Lab ustanova je koja promiče upotrebu otvorenog koda u industrijskoj automatici. Kao takav, *OASDL* osigurava servise i proizvode koji pomažu pri implementaciji *softwarea*, širok raspon stručnih znanja iz navedenog područja te certificira proizvode i procese.
OSADL "Latest Stable" `PREEMPT_RT` *real-time Linux kernel* projekt je koji za cilj ima

pružiti informaciju o stabilnosti `CONFIG_PREEMPT_RT` dodatka. Točnije, *"Latest Stable"* verzija `PREEMPT_RT` *Linux* jezgre indicira da, iako podvrgnuta temeljitom testiranju, nisu poznati problemi koji bi spriječili korištenje jezgre u industrijskom okruženju.
Nove verzije *real-time* jezgre kontinuirano su testirane na svim podržanim arhitekturama te se jezgra označava kao "*Latest Stable*" ukoliko

- Jezgra je pokrenuta na svim sustavima bez rušenja sustava u vremenskom trajanju od mjesec dana
- Sve mogućnosti prethodne "*Latest Stable*" jezgre su dostupne
- *real-time* mogućnosti nisu umanjene u usporedbi s prethodnom "*Latest Stable*" jezgrom
- performanse nisu umanjene u usporedbi s prethodnom "*Latest Stable*" jezgrom

U vrijeme izrade rada oznaku "*Latest Stable*" nosila je verzija `3.12` te je preporučena verzija korištena pri izradi. [8]

4.1.2. Instalacija Real-time Linux operacijskog sustava

Real-Time Linux jezgra operacijskog sustava nastaje primjenom `CONFIG_PREEMPT_RT` dodatka (*patch*) na izvorni kod *Linux* jezgre.
`CONFIG_PREEMPT_RT` dodatak i izvorna jezgra kompatibilni su uz uvjet da su glavna verzija dodatka i jezgre jednaki. Kako je odabrana `3.12.40` verzija dodatka, potrebno je koristiti `3.12.40` verziju izvornog koda jezgre. Nakon što je primjena `CONFIG_PREEMPT_RT` dodatka završena, potrebno je prevesti i instalirati modificiranu *Linux* jezgru. Prije samog prevođenja potrebno je podesiti postavke jezgre. Naredbom `make menuconfig` otvara se grafičko sučelje u kojem je moguće interaktivno podesiti opcije koje jezgra podržava. Kako će osnovne postavke karakteristične za sklopovlje sustava biti popunjene na temelju trenutno instaliranog operacijskog sustava, prethodno je na ciljno računalo instalirana distribucija *Ubuntu 14.04 LTS* te je prevođenje izvedeno na računalu na koje će nova jezgra biti instalirana kako bi se izbjegli problemi nekompatibilnosti. Na temelju različitih izvora odabran je skup postavki uz koje je preveden jezgra. Izmjene u odnosu na predefinirane postavke u velikoj mjeri se odnose na omogućavanje te onemogućavanje mogućnosti čuvanja energije i *debuggera* koji su uzročnici visokih latencija u sustavu.
Iako je preporučeno onemogućiti *ACPI* (Advanced Configuration and Power Interface), od verzije dodatka `2.16.18` potrebno ju je omogućiti kako bi se aktivirao *timer* visoke

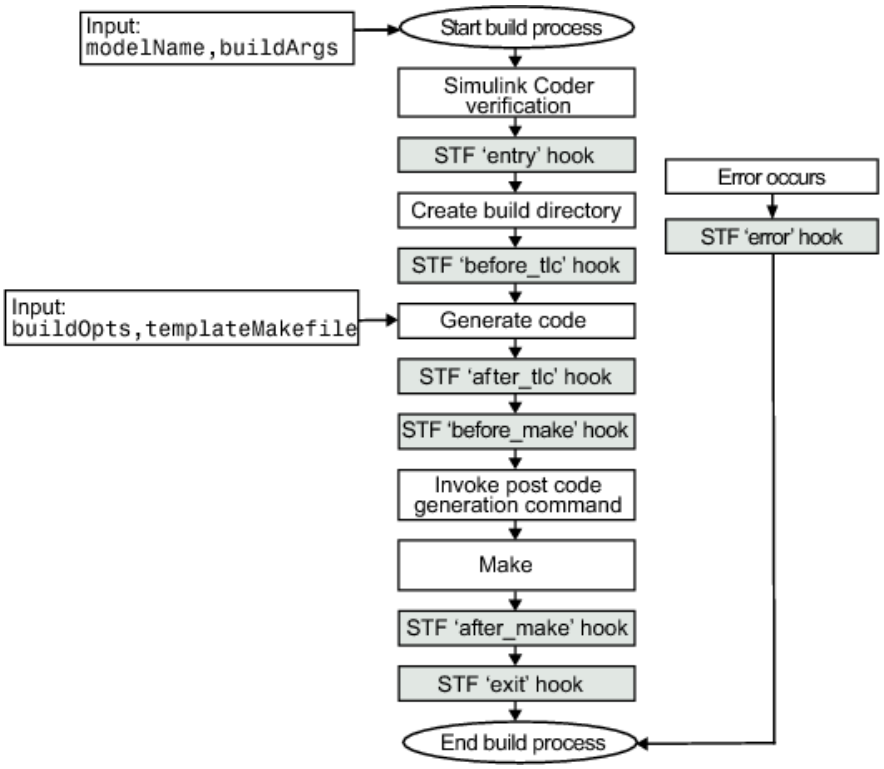
rezolucije. Potom je potrebno onemogućiti sve prisutne podopcije.

4.2. Simulink Linux real-time target

Prevođenje *Simulink* modela i njegovo izvođenje na ciljnom računalu ostvareno je korištenjem *Simulink Codera* i *Linux ERT (embedded real-time) target* za *Simulink Coder*. Rezultat prevođenja je *executable* datoteka koju je moguće pokrenuti na ciljnom računalu.

4.2.1. Prijenos i pokretanje na ciljnom računalu

Iako je u sklopu *Linux ERT targeta* implementirano osvježavanje i prikaz signala u stvarnom vremenu, prijenos prevedenog modela na ciljno računalo i njegovo pokretanje nije omogućeno. Kako bi se u najvećoj mogućoj mjeri pojednostavio i ubrzao postupak razvoja *Simulink* modela do pokrenute simulacije u stvarnom vremenu, korištenjem `STF_make_rtw_hook` [6] funkcije omogućeno je izvođenje akcija koje je potrebno poduzeti po završetku prevođenja kako bi se kod nastao prevođenjem uspješno pokrenuo na ciljnom računalu. Funkcija se poziva u karakterističnim koracima prevođenja (Slika 4.1).



Slika 4.1: Proces prevođenja

Argumenti funkcije `ert_linux_make_rtw_hook` korištene pri izradi rada su `hookMethod` u kojem se pri svakom pozivu funkciji predaje informacija o fazi prevođenja iz koje je pozvana te `modelName` kojim je funkciji predano ime modela koji se prevodi. [dodatak D] Unutar funkcije provjerava se iz kojeg je koraka prevođenja funkcija pozvana te, ukoliko je funkcija pozvana nakon prevođenja (argument `hookMethod` je `after_make`), poziva se funkcija `linuxAfterMakeHook`. [dodatak D] Pozvana funkcija će koristeći *SSH* protokol pristupiti ciljnom računalu kao *root* korisnik te potom poslati *TERM* signal pokrenutim modelima. Prije pokretanja modela njegov identifikacijski broj procesa (PID) zapisan je u datoteku `started` u *Simulink* mapi na ciljnom računalu. Ova sigurnosna mjera sprječava pokretanje novog modela (zadatka najviše razine) ukoliko prethodno pokrenuti model nije prekinuo izvođenje. Ovakva situacija tipična je za slučaj kada se korisnik odspojio u *Simulinku* bez zaustavljanja pokrenutog modela te je pokrenut novi proces prevođenja. Izvođenje više od jednog zadatka najviše razine potencijalno umanjuje mogućnosti izvođenja u stvarnom vremenu. Potom su izbrisani svi prethodno preneseni modeli (svi modeli u mapi `Simulink/models`) te je trenutni model potom prenesen u istu mapu . Nakon što je prenesenoj *executable* datoteci omogućeno izvođenje datoteka je pokrenuta uz opciju `-w`, te će model čekati pokretanje od strane korisnika unutar *Simulink* sučelja na računalu domaćinu. Korištenjem naredbe `nohup` spriječen je završetak pokrenutog procesa prilikom prekidanja *SSH* veze, a preusmjeravanjem sva tri ulazno-izlazna toka izbjegnut je problem koji se često pojavljuje, kada se *SSH* odbija odspojiti kako podaci prema i od pokrenutih procesa ne bi bili izgubljeni. Nadalje, identifikacijski broj pokrenutog procesa (PID) preusmjeren je u `Simulink/started` datoteku kako bi prilikom sljedećeg procesa prevođenja i pokretanja modela, pokrenuti proces mogao biti prekinut.

5. Komunikacija

Komunikacija između ciljnog računala i *Roboteq* kontrolera ostvarena je preko *CAN* sabirnice. korištenjem *CANopen* protokola višeg reda. *Roboteq* kontroleri korišteni pri izradi rada podržavaju četiri načina rada RawCAN, MiniCAN, CANopen i RoboCAN. Pri izradi rada korišten je CANopen protokol

5.1. CAN mrežno sučelje

Korišteno mrežno sučelje *IXXAT iPC-I 165/PCI* omogućava jednokanalnu komunikaciju računala s čvorovima na *CAN* mreži. Kako podrška za korišteno mrežno sučelje nije implementirana unutar programskog paketa Simulink, potrebno je postojeće upravljačke programe prilagoditi svojstvima *Simlink* okruženja.

5.1.1. Upravljački programi

Pri izradi rada korištena je modificirana inačica *BCI* (*Basic Can Interface*) upravljačkog programa (*drivera*), verzije 4.05.15. Izvorni kod *BCI* upravljačkog programa namijenjen je prevođenju prevodiocem *gcc* verzije 2.95.3. Kako bi se upravljački program mogao uspješno prevesti aktualnim *gcc* prevodiocem za korištenu *Linux* 3.12.39 jezgru potrebno je načiniti promjene u izvornom kodu upravljačkog programa. Načinjene promjene prikazane su tablicom **??**.

Tablica 5.1: Promjene izvornog upravljačkog programa

datoteka	can_act.c
originalni kod	vma->vm_flags |= VM_IO | VM_RESERVED;
izmijenjen kod	vma->vm_flags |= VM_IO | (VM_DONTEXPAND | VM_DONTDUMP);
datoteka	can_act.c
originalni kod	static int device_ioctl (struct inode *inode, struct file *file, unsigned int ioctl_number, unsigned long ioctl_param);\end{verbatim}
izmijenjen kod	static int device_ioctl (struct file *file, unsigned int ioctl_number, unsigned long ioctl_param);\end{verbatim}
datoteka	can_act.c
originalni kod	create_proc_read_entry ("dpram", 0, prc_dir, prc_read_dpram, NULL); create_proc_read_entry ("status", 0, prc_dir, prc_read_status, NULL); remove_proc_entry ("status", prc_dir); remove_proc_entry ("dpram", prc_dir); remove_proc_entry (ProcDirName, NULL);

izmijenjen kod	// create_proc_read_entry ("dpram", 0, prc_dir, prc_read_dpram, NULL);
	// create_proc_read_entry ("status", 0, prc_dir, prc_read_status, NULL);
	// remove_proc_entry ("status", prc_dir);
	// remove_proc_entry ("dpram", prc_dir);
	// remove_proc_entry (ProcDirName, NULL);

datoteka	can_act.c
originalni kod	ioctl:device_ioctl,
izmijenjen kod	unlocked_ioctl:device_ioctl,
opis	
datoteka	can_act.c
originalni kod	ioctl:device_ioctl,
izmijenjen kod	unlocked_ioctl:device_ioctl,
datoteka	modprobe.conf
originalni kod	install can0 /sbin/modprobe -o can0 can_act type=IPCI165ISA irq=7 addr=0xd000 major=250 install can1 /sbin/modprobe -o can1 can_act type=IPCI320ISA irq=5 addr=0xd800 major=251 install can2 /sbin/modprobe -o can2 can_act type=PCI major=252
izmijenjen kod	# install can0 /sbin/modprobe -o can0 can_act type=IPCI165ISA irq=7 addr=0xd000 major=250 # install can1 /sbin/modprobe -o can1 can_act type=IPCI320ISA irq=5 addr=0xd800 major=251 # install can2 /sbin/modprobe -o can2 can_act type=PCI major=252 install can0 /sbin/modprobe -o can0 can_act type=PCI
opis	

5.2. CANopen

CANopen protokol implementiran je unutar programskog paketa *Simulink* korištenjem S-funkcija [dodatak B]. S-funkcija poziva funkcije iz modificiranog aplikacijskog programskog sučelje (API) *IXXAT ipc-i 165 PCI* CAN mrežne kartice kako bi se ostvarila komunikacija s upravljačkim programima kartice instaliranim na ciljnom računalu. Implementirani su *TPDO* i *RPDO* tipovi poruka kojima je ostvaren prijenos podataka s *Roboteq* servokontrolera na ciljno računalo i s ciljnog računala na servokontrolere. Napisan je API [dodatak C] u kojemu su implementirane osnovne funkcije namijenjene stvaranju *RPDO* poruka `packMessage` i interpretaciji *TPDO* poruka `unpackMessage`. [dodatak B]

5.2.1. RPDO

Podrška za stvaranje *RPDO* poruka ostvarena je funkcijom `packMessage`. Kako svaka RPDO poruka može osvježiti dvije varijable *Roboteq* kontrolera funkcija kao argument prima pokazivač na poruku koju je potrebno stvoriti te identifikacijske brojeve (*ID*) i vrijednosti dvije varijable kontrolera koje poruka osvježava. Vrijednosti varijabli pretvorene su u niz varijabli tipa `uint8` duljine osam, u kojem su pohranjene vrijednosti obje varijable u heksadecimalnom zapisu. Identifikacijski brojevi varijabli definirani su unutar `rhino.h` zaglavlja te je funkciji dovoljno predati nazive veličina koje je moguće osvježavati (naponi motora robota). Kako je na svakom kontroleru moguće osvježiti dvije veličine, identifikacijski broj čvora (kontrolera) na *CAN* mreži određen je rezultatom, a identifikacijski broj *RPDO* poruke koju je potrebno poslati ostatkom cjelobrojnog dijeljenja identifikacijskog broja poruke s dva, čime je omogućeno dodavanje novih kontrolera ili osvježavanje dodatnog broja varijabli bez značajnijih izmjena postojećeg koda. Uz poznat identifikacijski broj čvora i *RPDO* poruke prema **??** generirano je zaglavlje *CAN* poruke te je funkcijom `BCI_CreateCANMsg` implementiranoj unutar *IXXAT* aplikacijskog programskog sučelja stvorena *CAN* poruka.

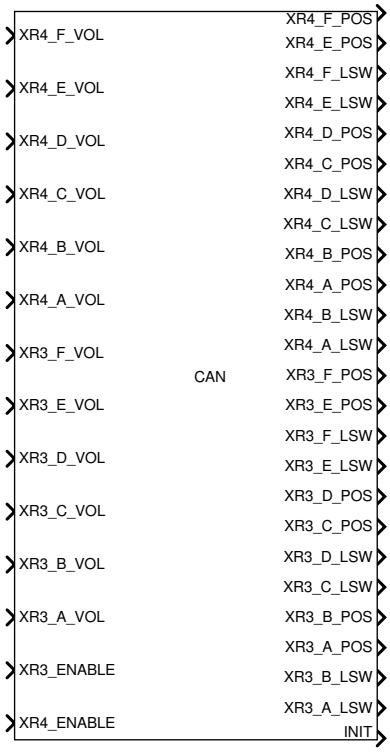
5.2.2. TPDO

Podrška za interpretaciju *TPDO* poruka primljenih s kontrolera ostvarena je funkcijom `packMessage`. Funkcija kao ulazne parametre prima poruku primljenu s *CAN* mrežnog sučelja, te pokazivače na varijable `valueID1` i `valueID2` te `value1` i `value2` preko kojih se programu iz kojeg je funkcija pozvana vraćaju identifikacijski brojevi te vrijednosti varijabli sadržanih u *CAN* poruci. Identifikacijski brojevi primljenih varija-

bli, definirani u `rhino.h` zaglavlju rekonstruirani su na temelju identifikacijskog broja kontrolera (čvora) te broja primljene TPDO poruke. Identifikacijski broj kontrolera i broj *TPDO* poruke moguće je odrediti iz **??**. Zaglavlje poruke u heksadekadskom zapisu je formata

5.2.3. CANopen rječnik

Funkcionalnost *CANopen* rječnika ostvarena je unutar S-funkcije korištenjem diskretnih stanja. Za vrijeme update faze korištenjem funkcije `BCI_ReceiveCanMsg` koja iz spremnika CAN mrežnog sučelja čita najstariju primljenu poruku. Ukoliko je poruka uspješno pročitana funkcija vraća definiranu vrijednost `BCI_OK` što omogućava provjeru trenutnog stanja spremnika *CAN* mrežnog sučelja (ukoliko vraćena vrijednost nije jednaka `BCI_OK`, spremnik je prazan). Poruke će iz spremnika biti čitane sve dok spremnik nije prazan, te će se njihova vrijednost spremati u neko od diskretnih stanja S-funkcije. [13] Na temelju identifikacijskih brojeva varijabli koje je moguće primiti s kontrolera definiranih u `rhino.h` varijable su povezane s diskretnim stanjem. Ovakvim načinom osvježavanja lokalne kopije varijabli ostvarena je funkcija *CANopen* rječnika. S-funkcija omogućuje primanje vrijednosti kuta i sklopke svakog kanala svakog od šest kontrolera te slanje referentnog napona motora (Slika 5.1



Slika 5.1: S-funkcija

Također, ukoliko primanje poruke nije realizirano, unutar lokalnog *CANopen* rječnika biti će posljednja primljena vrijednost čime je osigurana određena razina robusnosti komunikacijskog sustava. Nadalje, iako su frekvencije slanja *TPDO* poruka *Roboteq* kontrolera i frekvencija osvježavanja *CANopen* rječnika jednake ($f_{TPDO,s} = f_{TPDO,R} = 200Hz$), sama komunikacija je asinkrona (slanje i primanje poruke nisu nužno istovremeni). U najnepovoljnijem slučaju, kada je kontroler poruku poslao neposredno nakon što su poruke pročitane iz spremnika *CAN* mrežnog sučelja od strane S-funkcije, poruke će kasniti jedno vrijeme uzorkovanja S-funkcije ($\Delta t_{MAX} = T_S = 5ms$).

5.2.4. Rješavanje problema interpretacije

Kao što je prethodno objašnjeno, zakreti motora robota, uslijed konstrukcijskih aspekata, u općenitom slučaju ne predstavljaju zakrete zglobova robota (Slika **??**). Kako bi se osigurala ispravna interpretacija primljenih kuteva zakreta motora potrebno je kompenzirati utjecaj zakreta prethodnog motora na kut zakreta zglobova *D* i *C*. Kut zakreta zgloba *E* ovisi o zakretu motora *D* dok kut zakreta zgloba *C* ovisi o zakretu motora *E* i *D*. Kompenzacija kuta ostvarena je unutar S-funkcije odgovorne za komunikaciju s kontrolerima te je na izlazu bloka prisutna stvarna vrijednost zakreta kuta zglobova.

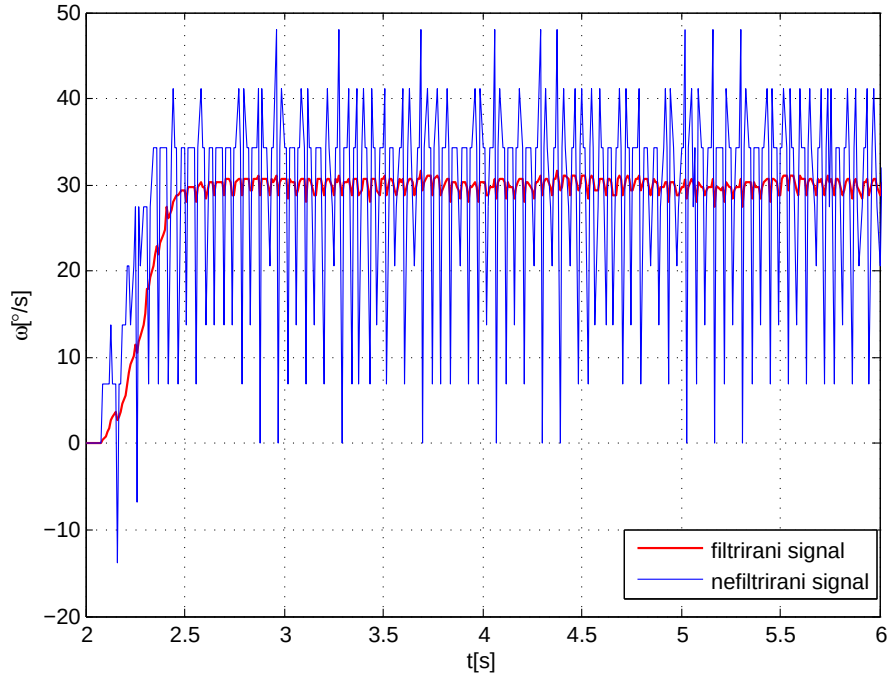
5.2.5. Filtriranje povratnog signala brzine vrtnje

Pri izradi regulatora korišten je filtar za usrednjavanje (diskretni niskopropusni filtar) u povratnoj vezi brzine vrtnje širine 15 uzoraka. Zbog kvantizacije vremenske domene, kao i zbog diskretiziranog izlaza s enkodera, mjerena brzina posjeduje šum. Širina filtra eksperimentalno je odabrana na temelju smanjenja šuma i povećanja kašnjenja koje filtar uzrokuje. Odabrana širina filtra je ona pri kojoj je smanjenje šuma zanemarivo u odnosu na porast kašnjenja signala.
Filtriranje signala u povratnoj vezi podređene petlje upravljanja po brzini vrtnje ostvareno je oduzimanjem trenutne vrijednosti pozicije zgloba od vrijednosti zgloba zakašnjele za 3 uzorka te dijeljenjem s iznosom 3 vremena uzorkovanja što je ekvivalentno traženju aritmetičke sredine posljednjih 3 vrijednosti brzine vrtnje, zbog:

$$\frac{\omega(k)+\omega(k-1)+\omega(k-2)}{3} = \frac{[\theta(k)-\theta(k-1)]+[\theta(k-1)-\theta(k-2)]+[\theta(k-2)-\theta(k-3)]}{3T_s} = \frac{\theta(k)-\theta(k-3)}{3T_s} \tag{5.1}$$

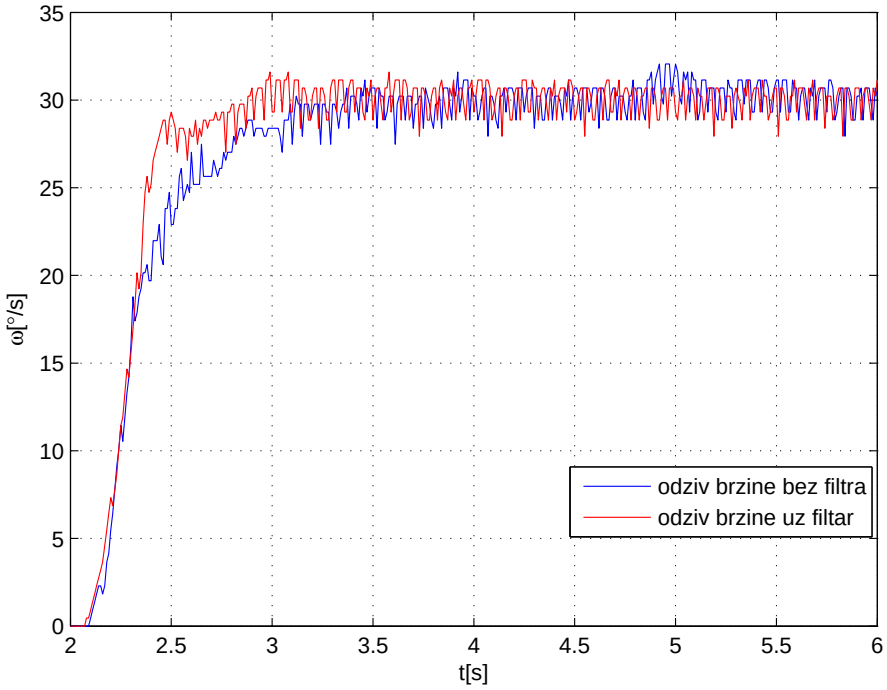
čime je ostvaren filtar za usrednjavanje.

Filtrirani i nefiltrirani signal povratne veze brzine vrtnje prikazani su na slici 5.2.



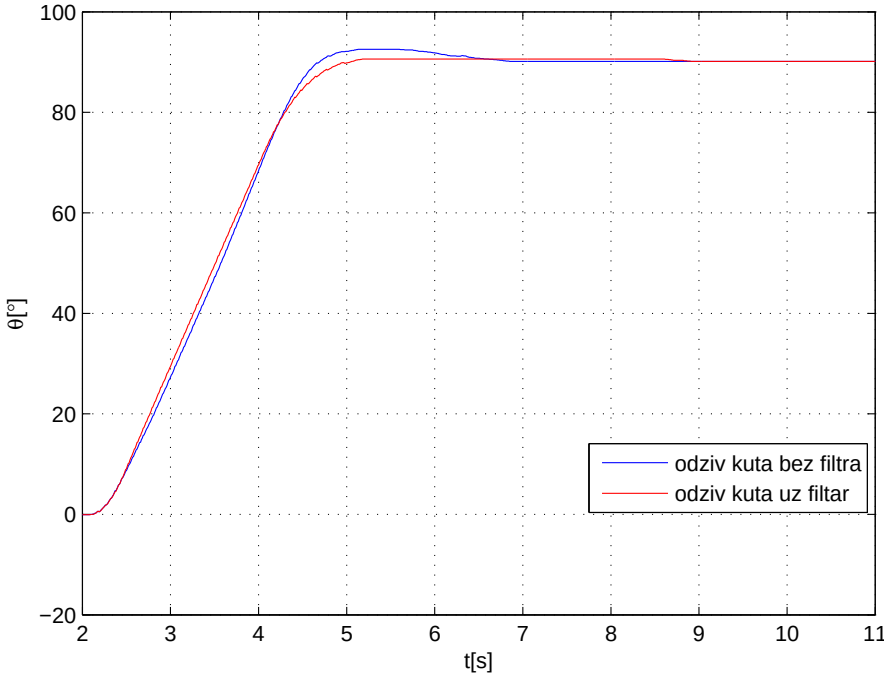
Slika 5.2: Filtrirani signal povratne veze brzine vrtnje

Primjetno je znatno smanjenje šuma signala uz zanemarivo kašnjenje. Odzivi brzine vrtnje i kuta zakreta kao i upravljačke veličine uz filtrirani i nefiltrirani signal prikazane su slikama 5.3, **??** i 5.5.



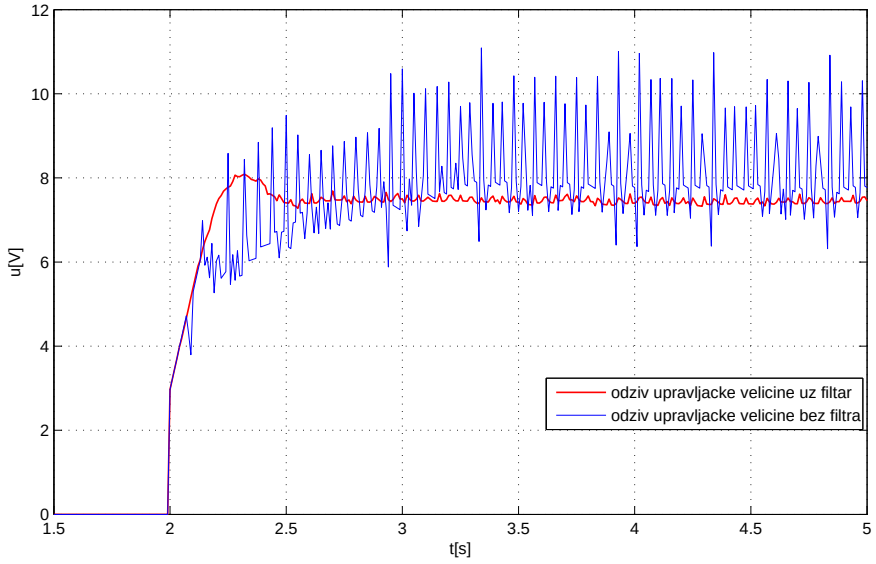
Slika 5.3: Ovisnost odziva brzine vrtnje na skokovitu pobudu o filtriranju povratnog signala

Može se uočiti nešto manje vrijeme porasta kod odziva brzine uz filtriran šum u odnosu na odziv brzine bez filtra.



Slika 5.4: Ovisnost odziva kuta na skokovitu pobudu o filtriranju povratnog signala

Uz prisutnost filtra, nadvišenje u odzivu kuta zakreta postaje zanemarivo.



Slika 5.5: Ovisnost odziva upravljačke veličine na skokovitu pobudu o filtriranju povratnog signala

Nagle promjene upravljačke veličine poželjno je smanjiti kako bi se produljio radni vijek korištene opreme. Uz filtriranje signala povratne veze upravljanja po brzini, smanjeno je nadvišenje i udari upravljačke veličine regulatora čime je podignuta ukupna kvaliteta upravljačkog sustava.

6. Upravljanje niže razine

Upravljanje niže razine podrazumijeva implementaciju svih upravljačkih struktura kojima je osigurano stabilno i kvalitetno upravljanje motorima robota (upravljanje u prostoru koordinata zglobova).

6.1. Regulator

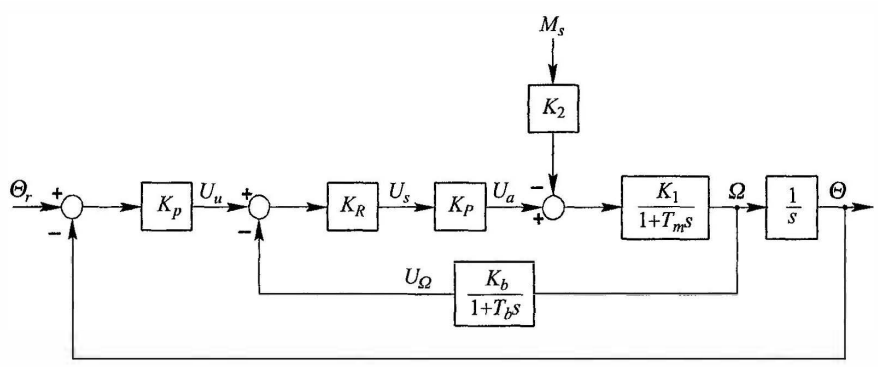
Kako je korištenjem *Roboteq SDC 2150N* i *Roboteq SDC 2160N* servokontrolera ostvareno upravljanje naponom armature motora robota odabran je način upravljanja položajem zgloba robota koji uključuje podređenu regulacijsku petlju brzine gibanja.

6.1.1. Oblik regulatora

Blokovska shema osnovne regulacijske petlje, koja sadrži naponsko pojačalo, prikazana je na slici 6.1. Pri modeliranju istosmjernog motora zanemaren je utjecaj viskoznog trenja ($B = 0$), pa prijenosna funkcija poprima oblik:

$$\frac{\Theta(s)}{U_a(s)} = \frac{K_1}{s(a + T_m s)} \tag{6.1}$$

Moment M_s koji uključuje međuzglobne momente inercije i djelovanje sile teže, djeluje na motor kao poremećajna veličina.



Slika 6.1: Blokovska shema regulacijske petlje položaja uz upravljanje po brzini vrtnje

Podređena petlja brzine vrtnje sadrži proporcionalni regulator opisan prijenosnom funkcijom oblika:

$$\frac{U_s(S)}{U_u(s) - U_\Omega(s)} = K_R \tag{6.2}$$

gdje je: K_R - koeficijent pojačanja regulatora.

Pojačalo daje napon armature proporcionalan ulaznom naponu, tj. izlazu iz regulatora položaja:

$$U_a(s) = K_P U_s(s) \tag{6.3}$$

gdje je: K_P - koeficijent pojačanja naponskog pojačala

Uobičajeno je da se brzina vrtnje mjeri tahogeneratorom ili inkrementalnim davačem impulsa, pri čemu prijenosna funkcija člana povratne veze brzine ima sljedeći oblik:

$$\frac{U_\Omega(s)}{\Omega(s)} = \frac{K_b}{1 + T_b s} \tag{6.4}$$

gdje je: K_b - koeficijent pojačanja člana povratne veze brzine vrtnje, Vs/rad

T_b - vremenska konstanta člana povratne veze brzine vrtnje, s

U tom slučaju prijenosna funkcija zatvorene regulacijske petlje brzine vrtnje u odnosu prema referentnoj veličini ima sljedeći oblik:

$$\frac{\Omega(s)}{U_u(s)} = \frac{K_R K_P K_1}{1 + K_R K_P K_1 K_b} \cdot \frac{1 + T_b s}{\frac{T_m T_b}{1 + K_R K_P K_1 K_b} s^2 + \frac{T_m + T_b}{1 + K_R K_P K_1 K_b} s + 1} \tag{6.5}$$

Slično tome, prijenosna funkcija zatvorene regulacijske petlje brzine vrtnje u odnosu prema poremećajnoj veličini ima oblik:

$$\frac{\Omega(s)}{M_s(s)} = -\frac{K_2}{1 + K_R K_P K_1 K_b} \cdot \frac{1 + T_b s}{\frac{T_m T_b}{1 + K_R K_P K_1 K_b} s^2 + \frac{T_m + T_b}{1 + K_R K_P K_1 K_b} s + 1} \tag{6.6}$$

Ako je opravdano zanemariti djelovanje vremenske konstante člana povratne veze brzine vrtnje ($T_b \ll T_m$), tada prijenosne funkcije (6.5) i (6.6) poprimaju novi oblik:

$$\frac{\Omega(s)}{U_u(s)} = \frac{\alpha K_R K_P K_1}{1 + \alpha T_m s} \tag{6.7}$$

$$\frac{\Omega(s)}{M_s(s)} = -\frac{\alpha K_2}{1 + \alpha T_m s} \tag{6.8}$$

gdje je: $\alpha = 1/(1 + K_R K_P K_1 K_b)$.

Uvođenje povratne veze po brzini vrtnje utjecalo je na smanjenje elektromehaničke vremenske konstante ($\alpha < 1$), a i na smanjenje utjecaja poremećajnog momenta, kao i na smanjenje utjecaja nelinearnosti statičkih karakteristika naponskog pojačala. Zahvaljujući uvođenju proporcionalnog regulatora brzine vrtnje, moguće je podesiti ukupno pojačanje regulacijskog kruga da se ostvari zadano dinamičko ponašanje.
[**?**]

Usporedbom struktura regulacijskih petlji položaja sa strujnim pojačalom i naponskim pojačalom vidi se da zbog drugačijih dinamičkih svojstava regulacijske petlje s naponskim pojačalom više nije potrebno koristiti PD regulator, već je sasvim dovoljan proporcionalni regulator s koeficijentom pojačanja K_R. Jednadžba zatvorenog sustava regulacije položaja koji sadrži podređenu regulacijsku petlju brzine vrtnje ima ovaj oblik:

$$\Theta(s) = \frac{K_o \Theta_r(s) - \alpha K_2 M_s(s)}{\alpha T_m s^2 + s + K_o} \tag{6.9}$$

gdje je: $K_o = \alpha K_P K_R K_p K_1$ - koeficijent pojačanja otvorenog regulacijskog sustava položaja

Karakteristična jednadžba zatvorenog sustava drugog reda opisana je izrazom : $s^2 + 2\xi\omega_n s + \omega_n^2$, gdje su koeficijent prigušenja i prirodna frekvencija neprigušenih oscilacija određeni izrazom:

$$\xi = \frac{1}{2\sqrt{K_o \alpha T_m}} \qquad\qquad \omega_n = \sqrt{\frac{K_o}{\alpha T_m}} \tag{6.10}$$

Promatrajući odziv položaja na skokovitu promjenu referentnog signala pri podešenju koeficijenta prigušenja $\xi = 1$ i uz pretpostavku da je $M_s = 0$, treba uočiti da dobiveno nadvišenje u odzivu iznosi $\sigma_m = 0\%$, u usporedbi s nadvišenjem $\sigma_m = 11\%$, koje je u najpovoljnijem slučaju dobiveno u regulacijskoj petlji položaja s podređenom regulacijom struje. Dakle, uvođenjem povratne veze po brzini vrtnje omogućeno je rješenje problema otklanjanja neželjenog nadvišenja u odzivu. Međutim, ostao je i dalje problem ovisnosti elektromehaničke vremenske konstante T_m o ukupnom momentu inercije J_u, čega je posljedica da je i u regulacijskoj petlji s podređenom regulacijom brzine vrtnje prisutna nepoželjna ovisnost koeficijenta prigušenja o promjenljivom momentu inercije. Osim toga, iz prijenosne funkcije (6.9) vidi se da i ovdje, kao i u slučaju s podređenom regulacijom momenta, postoji pogreška položaja u stacionarnom stanju, koja nastaje zbog djelovanja poremećajnog momenta m_s u obliku momenta sile teže ($M_s(s = 0) = M_g$):

$$E_\Theta(s=0) = \frac{\alpha K_2 M_g}{K_o} \tag{6.11}$$

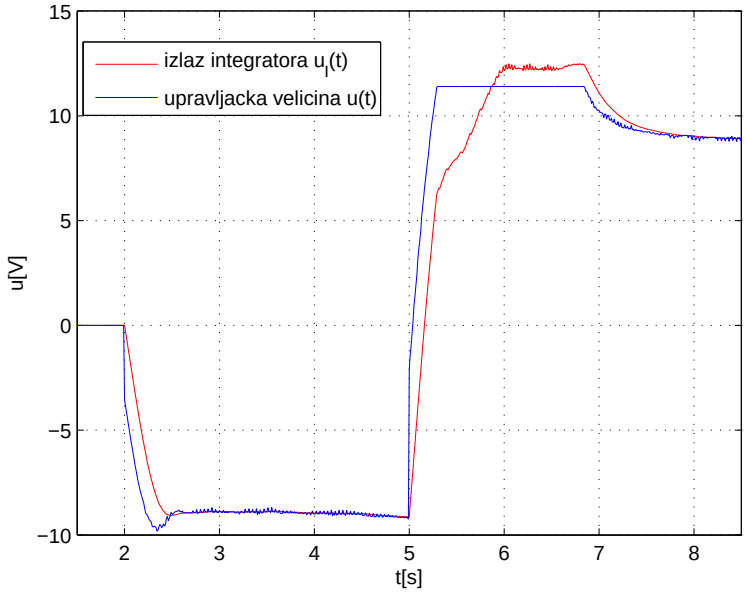
Kako je estimacija momenta konkretnog sustava s više zglobova i članaka kompliciran postupak, dodaje se integralna komponenta u povratnu vezu podređene petlje upravljanja po brzini vrtnje čime je otklonjeno stacionarno regulacijsko odstupanje. [14]

6.1.2. Sprječavanje efekta zaleta

Efekt zaleta

Zalet (engl. *windup*) integratora podrazumijeva prilike kod PID regulatora s povratnom vezom pri nagloj promjeni referentne veličine, pri čemu integralna komponenta akumulira znatnu pogrešku prilikom rasta (zaleta), nastavljajući rasti zbog nepostojanja negativne pogreške što rezultira dodatnim nadvišenjem. Zalet integratora najčešće je posljedica ograničenja fizičkih sustava u odnosu na idealne sustave. Idealan odziv nemoguće je dobiti zbog postojanja zasićenja (engl. *saturation*) (izlaz sustava je ograničen gornjom i donjom granicom) rezultirajući konstantnom pogreškom.

Efekt zaleta pri upravljanju konkretnim robotom Rhino XR-4 za skokovitu promjenu referentne brzine sa $-35°/s$ na $35°/s$ prikazan je slikom 6.2



Slika 6.2: Efekt zaleta

Na slici je vidljivo, da izlazna veličina integratora u_I, nastavlja rasti, iako je upravljačka veličina u ograničena. Kako je upravljačka veličina ograničena zasićenjem, u statičkim

prilikama ovo neće znatno utjecati na sustav, ali u dinamičkim prilikama sustav je tromiji, zbog vremena potrebnog da se izlaz integratora vrati u dozvoljene granice. [15]

Metode sprječavanja efekta zaleta

Neke od metoda rješavanja efekta zaleta (engl. *anti-windup*) su:

- Inicijalizacija integralne komponente regulatora na željenu vrijednost
- Postupno povećanje referentne veličine po rampi
- Onemogućavanje integralne komponente regulatora do povratka kontrolirane veličine procesa u područje kontrole
- Metoda uvjetne integracije kod koje je onemogućena akumulacija ispod ili iznad zadanih granica
- Metoda povratne integracije kojom se integralna komponenta, iako aktivna, drži unutar zadanih granica.

Pri izradi rada korištene su dvije metode sprječavanja efekta zaleta:

a) **Inicijalizacija integralne komponente regulatora na željenu vrijednost**

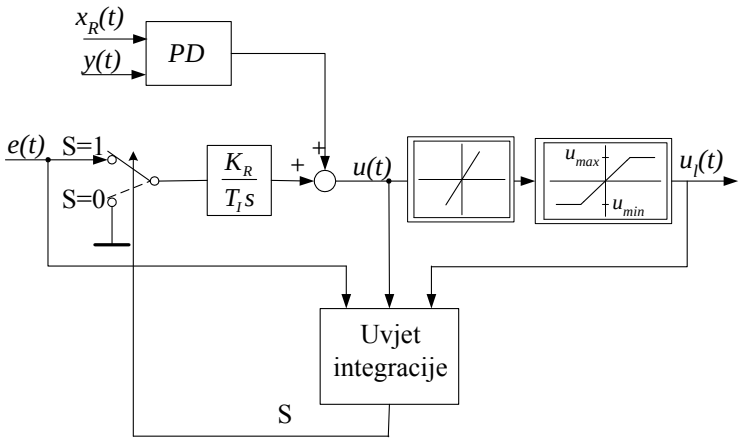
Metoda inicijalizacije integralne komponente na željenu vrijednost korištena je pri inicijalizaciji zgloba F, u trenutku kada zglob robota dosegne krajnju točku brzina mu naglo pada na 0, iako je zadana referentna brzina pozitivna. Od trenutka zaustavljanja do detekcije krajnjeg položaja brzinom okidanom sklopkom potrebno je neko vrijeme za koje integralna komponenta akumulira iznos pogreške te bi po nastavku kretanja prema inicijalnom položaju akumulirana pozitivna pogreška uzrokovala podbačaj. U trenutku okidanja brzinom okidane sklopke izlaz integralne komponente postavlja se na vrijednost 0.

b) **Metoda uvjetne integracije**

Metoda uvjetne integracije temelji se na onemogućavanju integriranja u slučaju kada integracija pridonosi izlasku upravljačke veličine izvan granica zasićenja. Integriranje će pridonositi odmicanju upravljačke veličine samo ukoliko je upravljačka veličina viša od gornje granice zasićenja i pogreška je pozitivna ili ukoliko je upravljačka veličina niža od donje granice zasićenja i pogreška je negativna, odnosno ukoliko vrijedi:

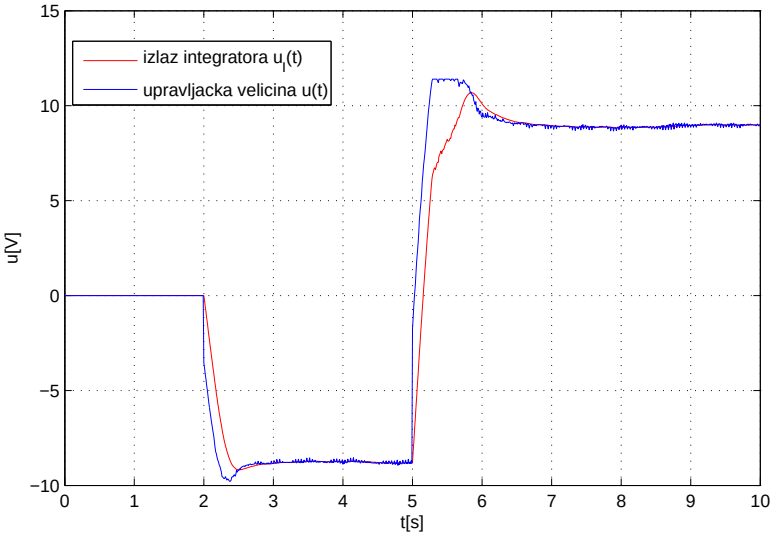
$$(u < u_l \wedge e < 0) \vee (u > u_h \wedge e > 0) \tag{6.12}$$

Idejna shema postupka uvjetne integracije prikazana je slikom 6.3.



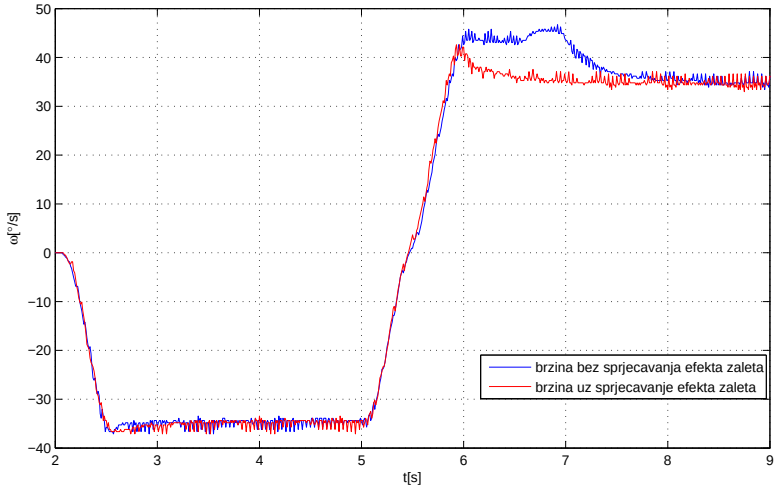
Slika 6.3: Sprječavanje efekta zaleta metodom uvjetnog integriranja

Razlog korištenja navedene metode jednostavna je implementacija algoritma uvjetne integracije. Primjenom metode uvjetnog integriranja, efekt zaleta uspješno je uklonjen (Slika 6.4).



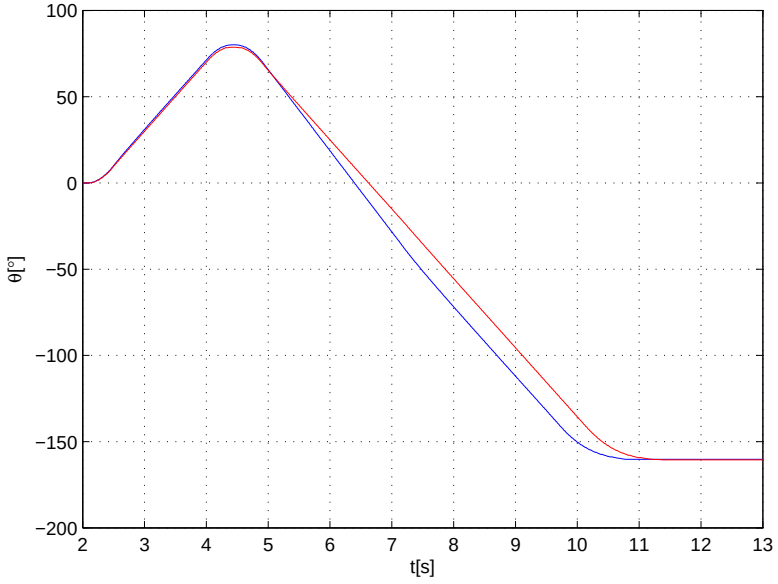
Slika 6.4: Odziv upravljačke veličine uz metodu uvjetnog integriranja

Za razliku od odziva na slici 6.2, uz korištenje metode uvjetnog integriranja, izlaz integratora ne raste izvan dozvoljenih granica, već je u zasićenju rast izlazne veličine integratora uvjetovan ponašanjem upravljačke veličine i pogreške. Utjecaj korištenja metode uvjetnog integriranja na odziv brzine pri nagloj promjeni referentne vrijednosti s $-35°/s$ na $35°/s$ dan je slikom 6.5.



Slika 6.5: Odziv brzine vrtnje uz metodu uvjetnog integriranja

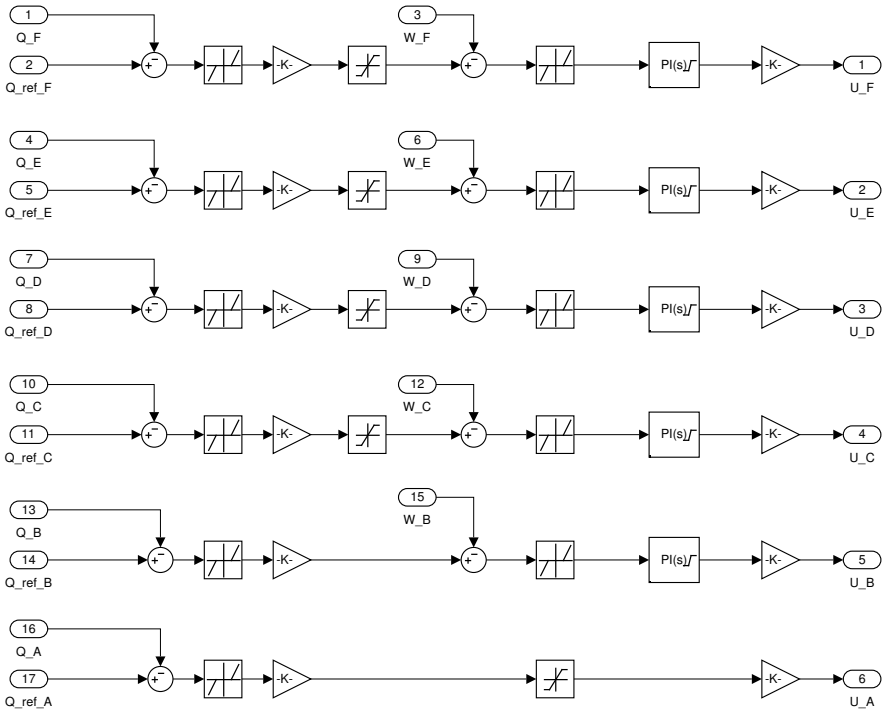
Zbog sprječavanja efekta zaleta metodom uvjetne integracije, smanjeno je nadvišenje pri naglim promjenama brzine, čime su poboljšana dinamička svojstva sustava. Poboljšana dinamička svojstva sustava vidljiva su i pri odzivu kuta na dvije ekstremne skokovite pobude. Za referentnu veličinu skoka na $-160°$, a nedugo potom u $160°$, odziv kuta je brži uz sprječavanje efekta zaleta (Slika 6.6).



Slika 6.6: Odziv kuta zgloba na skokovitu pobudu uz metodu uvjetnog integriranja

Model regulatora

Model strukture regulatora jednog Rhino robota izrađen na temelju pretpostavljenog oblika regulatora, prikazan je slikom 6.7



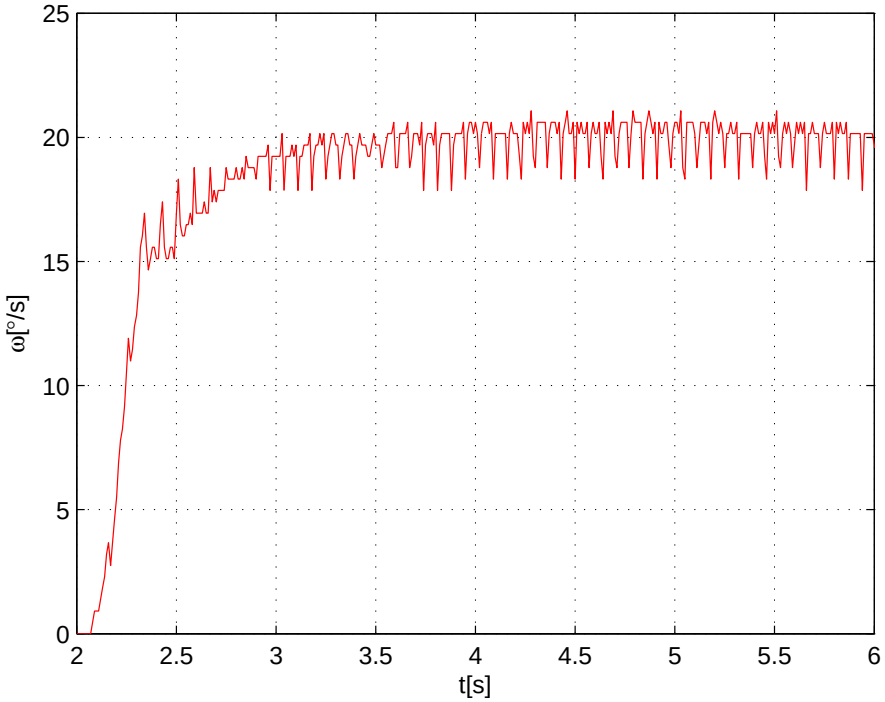
Slika 6.7: Ispitni model pri parametriranju regulatora

Upravljačka veličina podređene petlje upravljanja po brzini vrtnje, kao i upravljačka veličina nadređene petlje upravljanja po poziciji zgloba ograničena je blokom *saturation*, PI regulator ostvaren je korištenjem bloka PID uz uključenu metodu efekta zaleta, te odabranu metodu uvjetnog integriranja (engl. *clamping*).

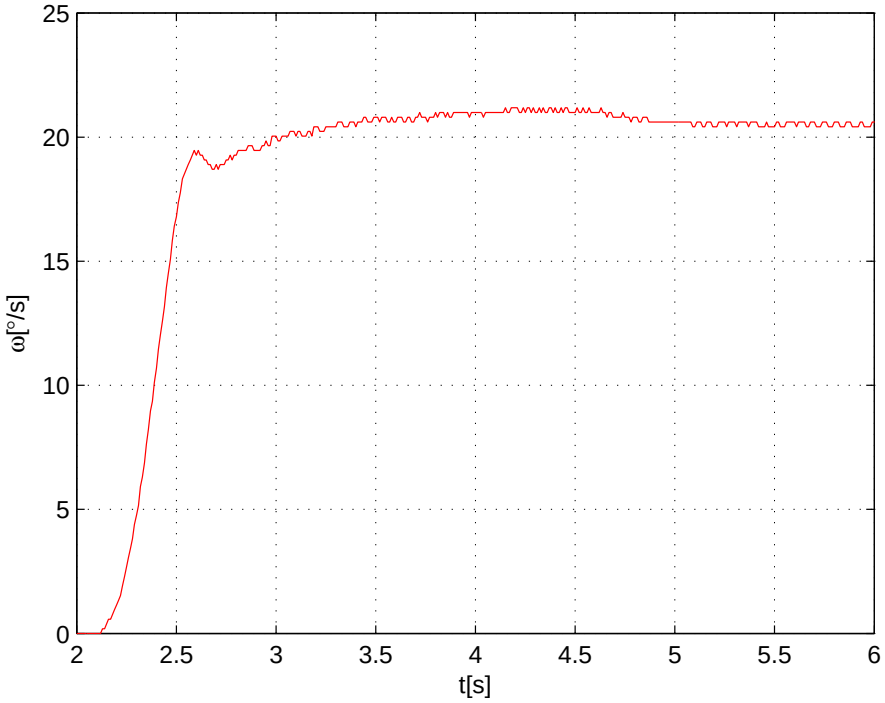
Parametri regulatora

Parametri regulatora zadanog oblika određeni su eksperimentalno, uz neke prethodno definirane parametre. Zasićenjem upravljačke veličine u podređenoj upravljačkoj petlji brzine vrtnje upravljačka veličina ograničena je na $[-0.95U_{max}, 0.95U_{max}]$. Za vrijeme diskretizacije regulatora odabrano je $T_s = 0.005s$ jer je za manja vremena uzorkovanja uočeno neispravno primanje poruka s *CAN* sabirnice. Postupak parametriranja regulatora započet je određivanjem parametara PI regulatora u podređenoj upravljačkoj petlji brzine vrtnje. Za svaki regulator određena je maksimalna brzina vrtnje pri kojoj odziv zadržava željena svojstva, odnosno, brzina vrtnje nakon koje regulator ne jamči željenu kvalitetu upravljanja.

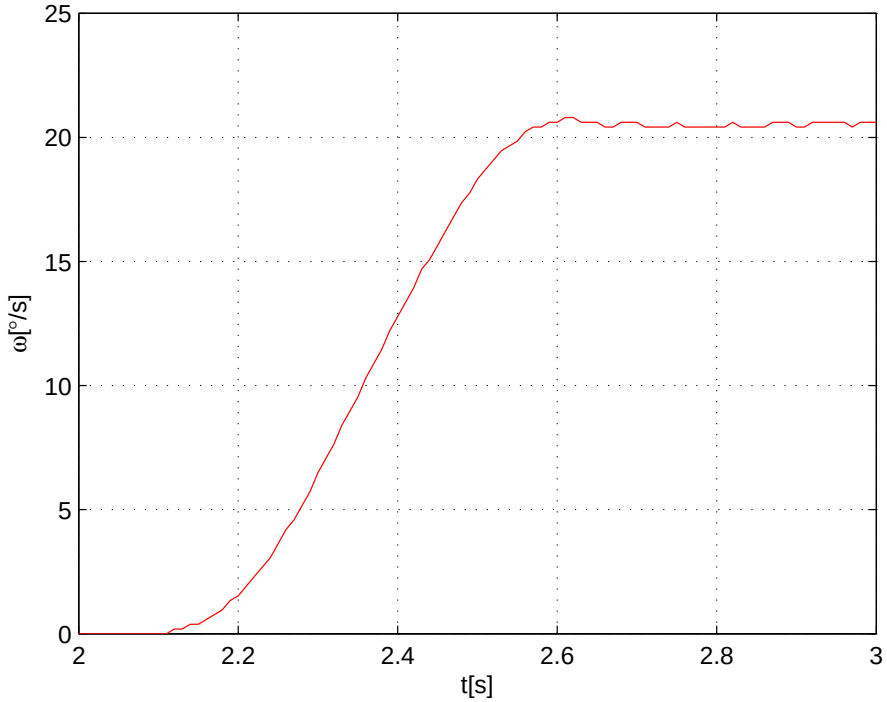
Odzivi brzine na skokovitu pobudu iznosa $20°/s$ uz odabrane parametre za svaki zglob prikazani su slikama 6.8, 6.9, 6.10 i 6.11.



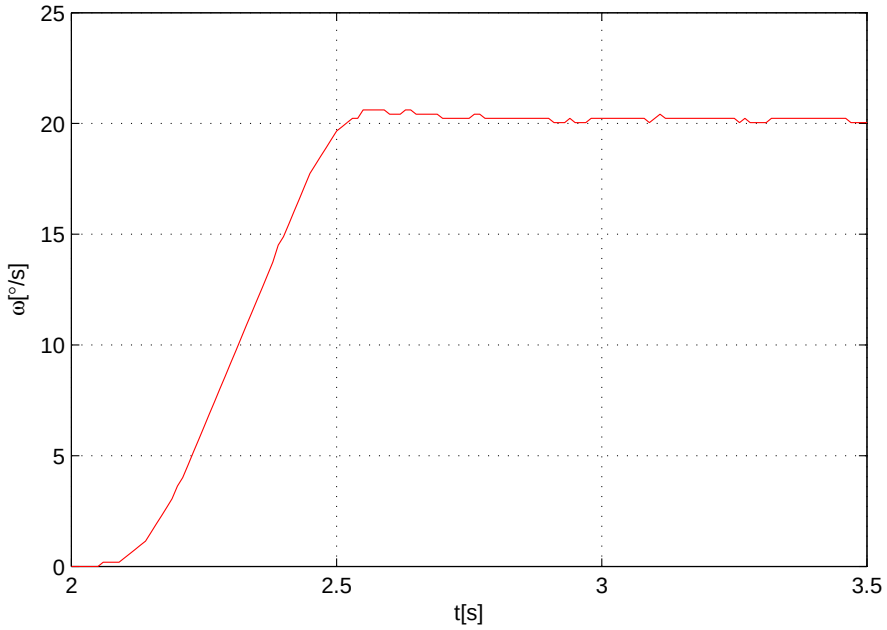
Slika 6.8: Odziv brzine zgloba struka na skokovitu pobudu



Slika 6.9: Odziv brzine zgloba ramena na skokovitu pobudu



Slika 6.10: Odziv brzine zgloba lakta na skokovitu pobudu

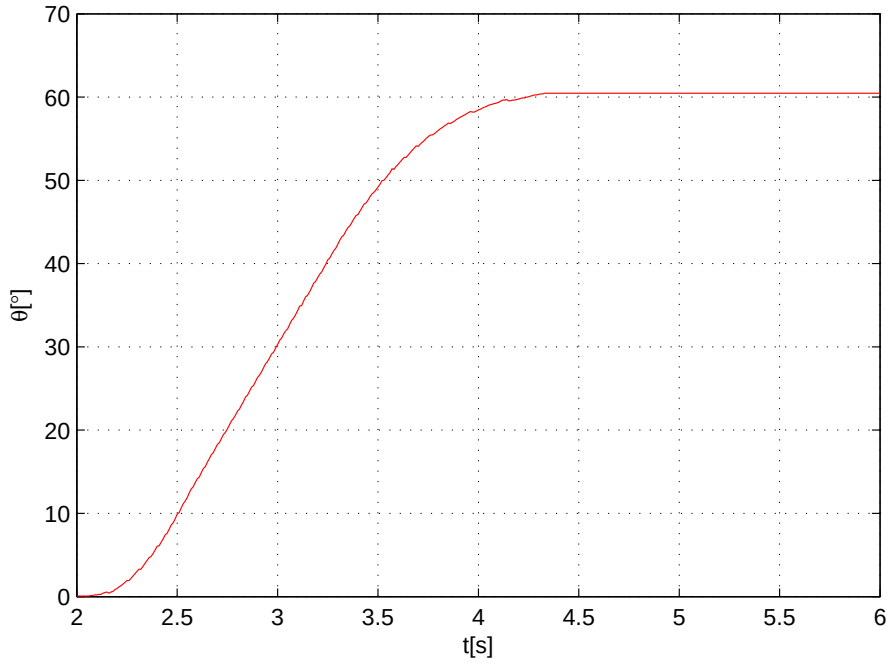


Slika 6.11: Odziv brzine zgloba zapešća na skokovitu pobudu

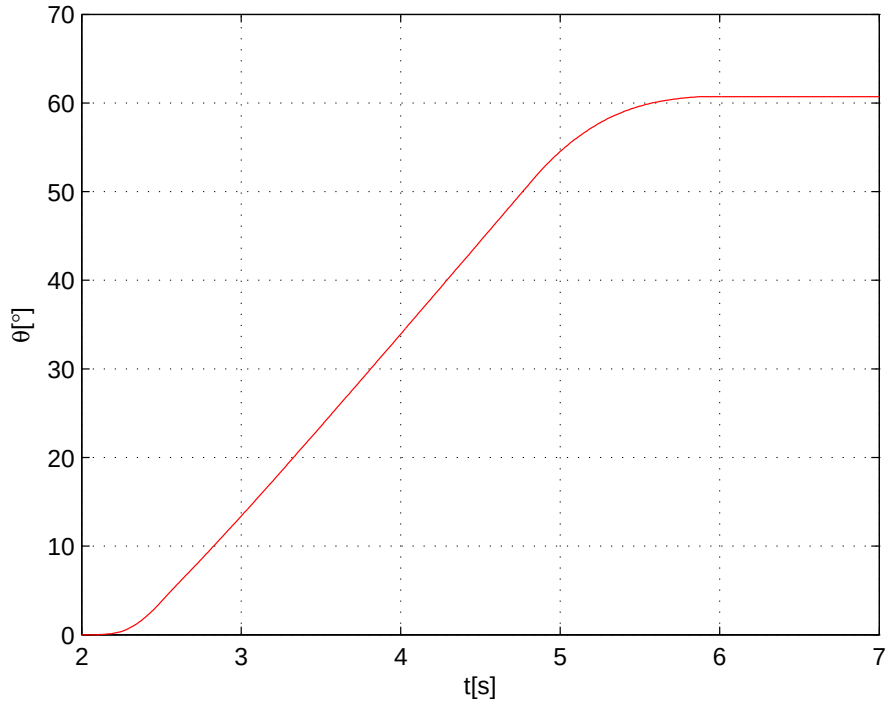
Dobiveni odzivi zadovoljavaju ranije postavljene uvjete kvalitete. Odstupanje brzine u stacionarnom stanju, kao i nadvišenje je zanemarivo, dok je vrijeme porasta prihvatljivo.

Po ugađanju parametara PI regulatora u podređenoj upravljačkoj petlji upravljanja po brzini vrtnje određeni su parametri P regulatora u vanjskoj petlji upravljanja po poziciji zgloba. Zasićenje upravljačke veličine u nadređenoj upravljačkoj petlji po

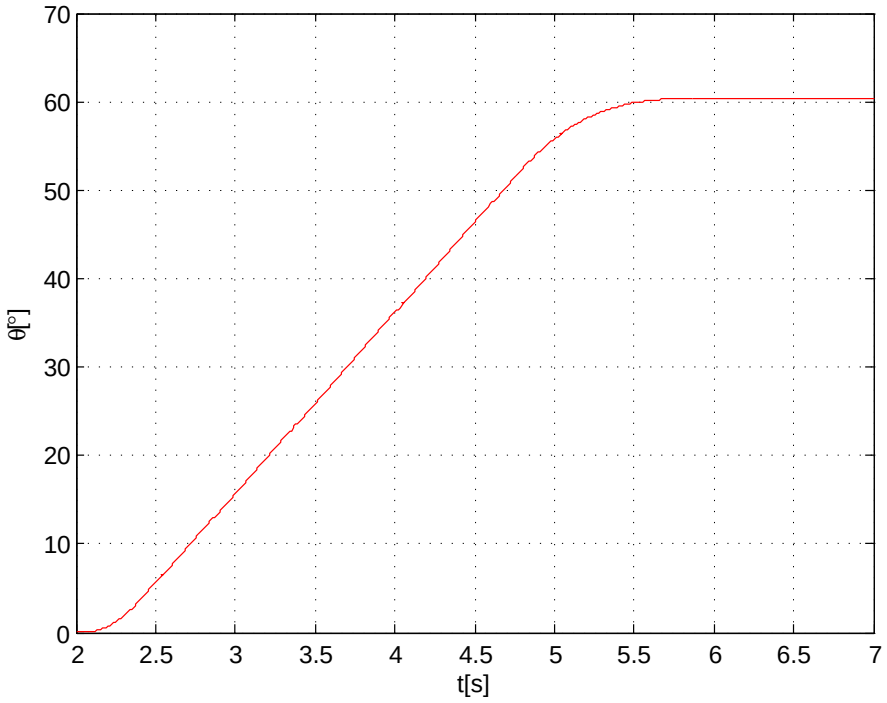
poziciji vrtnje ograničeno je na $[-v_max, v_max]$ kako se podređene petlje upravljanja po brzini ne bi dovelo u područje nezajamčene kvalitete odziva. Odzivi pozicije zglobova uz odabrane parametre za svaki zglob prikazani su slikama 6.12, 6.13,6.14 i 6.15.



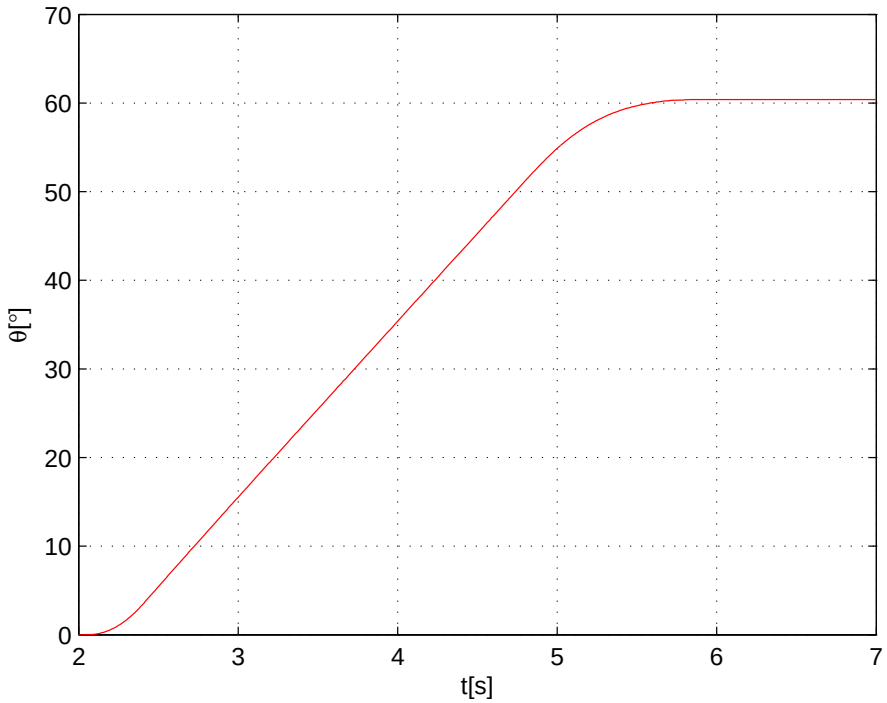
Slika 6.12: Odziv pozicije zgloba struka na skokovitu pobudu



Slika 6.13: Odziv pozicije zgloba ramena na skokovitu pobudu



Slika 6.14: Odziv pozicije zgloba lakta na skokovitu pobudu



Slika 6.15: Odziv pozicije zgloba zapešća na skokovitu pobudu

Odzivi pozicije upravljanog sustava u skladu su sa zahtjevima kvalitete postavljenim na upravljački sustav. Odstupanje pozicije u stacionarnom stanju neznatno je, a mala odstupanja vidljiva pri odzivima rezultat su činjenice da konačni izmjereni kut odgovara položaju potpuno ispružene ruke. U tom položaju, utjecaj momenta gravitacijske sile

najjače je izražen (najdulji krak uz okomitu silu).

Konačni parametri svakog od regulatora, dobiveni eksperimentalnim ugađanjem, prikazani su tablicom

Tablica 6.1: Eksperimentalno određeni parametri regulatora)

zglob	$\mathbf{K_R}$	$\mathbf{u_{\theta l}}$	$\mathbf{K_P}$	$\mathbf{T_I}$	$\mathbf{u_{\omega l}}$	$\delta_{\mathbf{q}}$	$\delta_{\mathbf{w}}$
F	1.95	40	9	0.0143	11.4	0.5	0.5
E	2.2	20	21	0.0086	11.4	0.5	0.5
D	2.1	20	10	0.0083	11.4	0.5	0.5
C	2.1	20	7	0.008	11.4	0.5	0.5
B	8		8	0	12	1	0.5
A	3	900	1	0	12	5	**0**

gdje je: K_R - koeficijent pojačanja regulatora u petlji upravljanja po poziciji.

$u_{\theta l}$ - gornja granica zasićenja upravljačke veličine u petlji upravljanja po poziciji (granice su simetrične), V

K_P-koeficijent pojačanja regulatora u petlji upravljanja po brzini

T_I - vremenska konstanta integratora u petlji upravljanja po brzini, s

$u_{\theta l}$ - gornja granica zasićenja upravljačke veličine u petlji upravljanja po poziciji (granice su simetrične), V

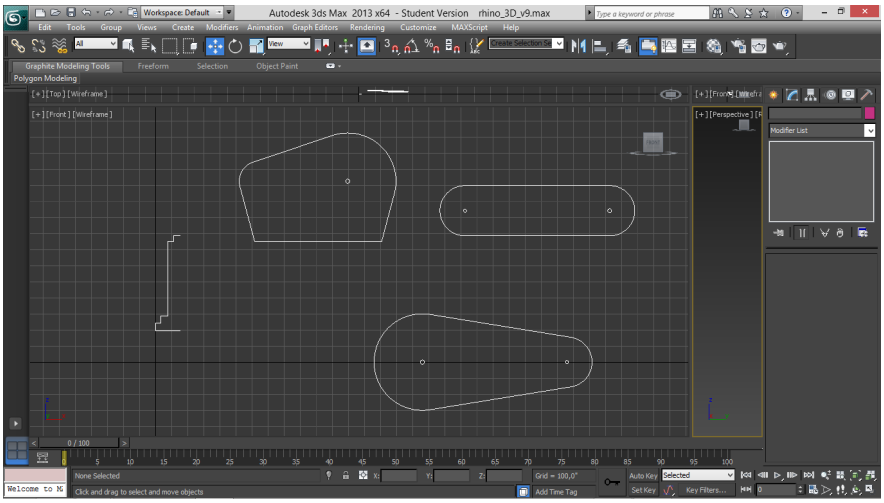
δ_θ - kraj mrtve zone pogreške u petlji upravljanja po poziciji (početak i kraj su simetrični), V

δ_ω - kraj mrtve zone pogreške u petlji upravljanja po brzini (početak i kraj su simetrični), V

7. Virtualni 3D model

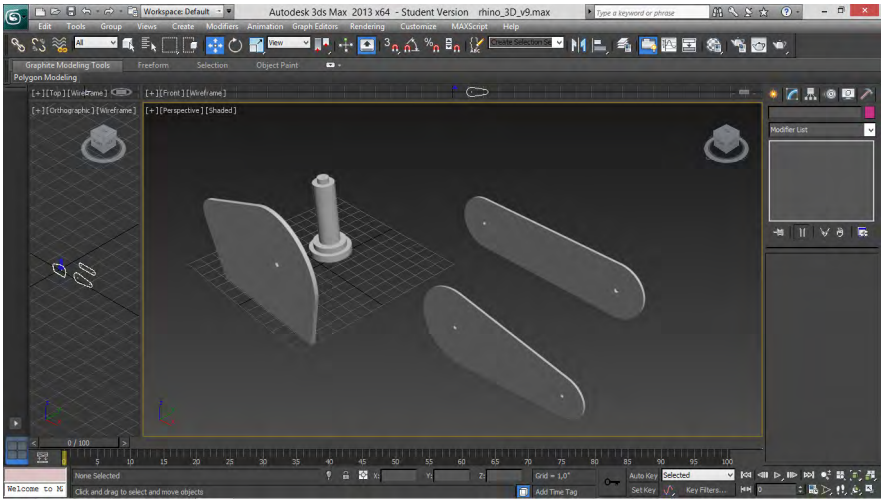
7.0.3. Izrada 3D modela

3D model *Rhino XR-4* robota izrađen je na temelju dimenzijskog crteža(2.2). Iako su označene sve veličine potrebne za izračun kinematike, određeni podaci, neophodni za izradu vjerne replike nisu navedeni, te su nepoznati parametri izmjereni korištenjem alata *Length* unutar *Autodesk Design Review* (duljine) ili izračunati na temelju izmjerenih duljina (kutevi kružnih lukova). Sam model izrađen je u *3ds Max* računalnom grafičkom alatu. Na temelju potpunog dimenzijskog crteža izrađeni su 2D objekti (koristeći primitive unutar *3ds Max*a) prikazani slikom 7.1



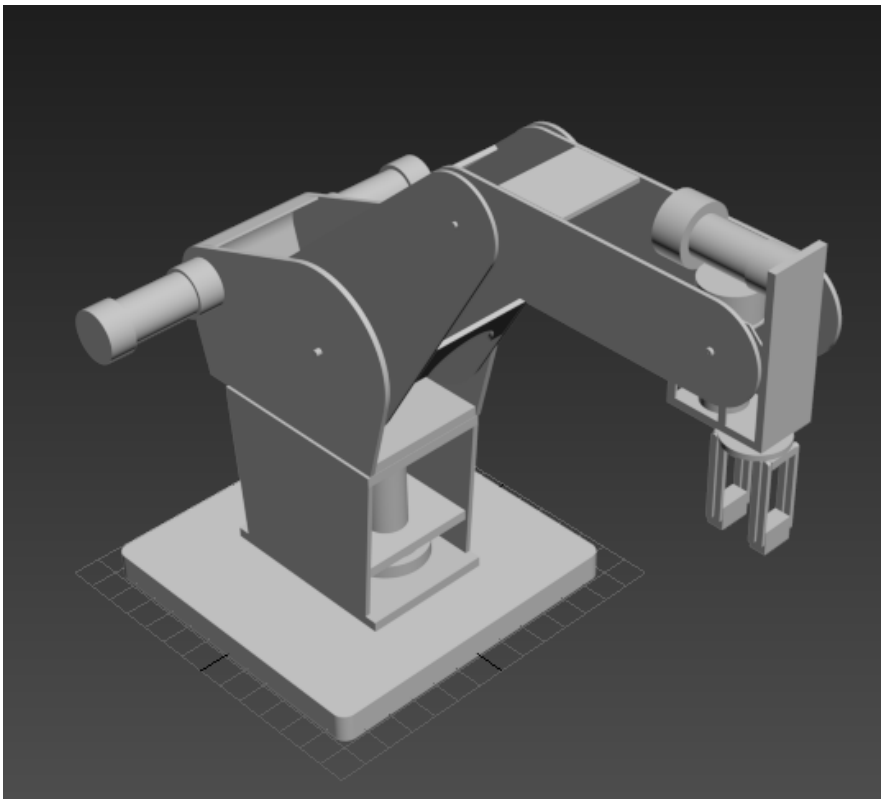
Slika 7.1: Izrada 2D objekata

Gotovi 2D objekti pretvoreni su u 3D objekte korištenjem modifikatora *Extrude* i *Lathe* ovisno o simetriji samog 3D objekta. Nekoliko 3D objekata prikazano je slikom 7.2



Slika 7.2: Izrada 3D objekata

Izrađeni jednostavni 3D objekti spojeni su u složenije na temelju njihove konstrukcijske pripadnosti, a zatim složeni u gotov 3D model robota (Slika7.5).

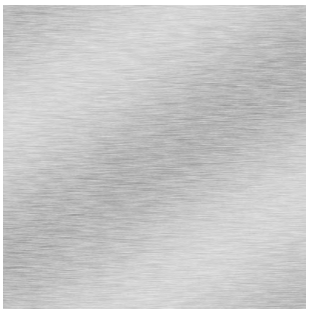


Slika 7.3: 3D model Rhino XR-4 robota

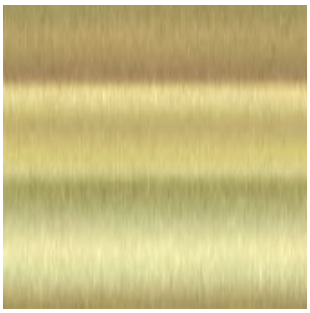
Kako bi virtualni 3D model što bolje predstavljao stvarni robot, 3D objektima dodijeljeni su materijali. Teksture korištene pri izradi materijala su:

- Grubo brušeni aluminij (Slika **??**)
- Fino brušeni mesing (Slika **??**)

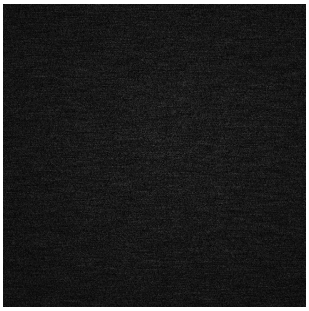
- Plastika (crna) (Slika **??**)
- Mat aluminij (Slika **??**)
- Fino brušeni aluminij (Slika **??**)
- Oštećeni mesing (Slika **??**)



(a) Grubo brušeni aluminij

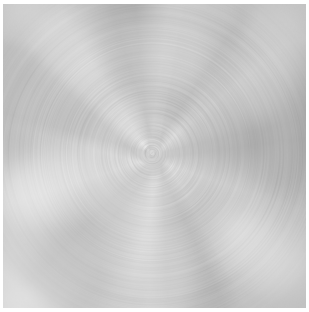


(b) Fino brušeni mesing

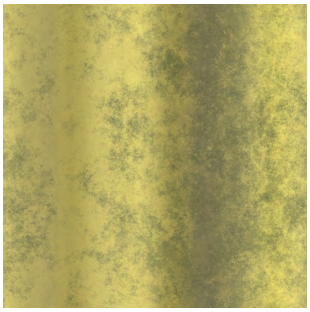


(c) Plastika (crna)

(d) Mat aluminij

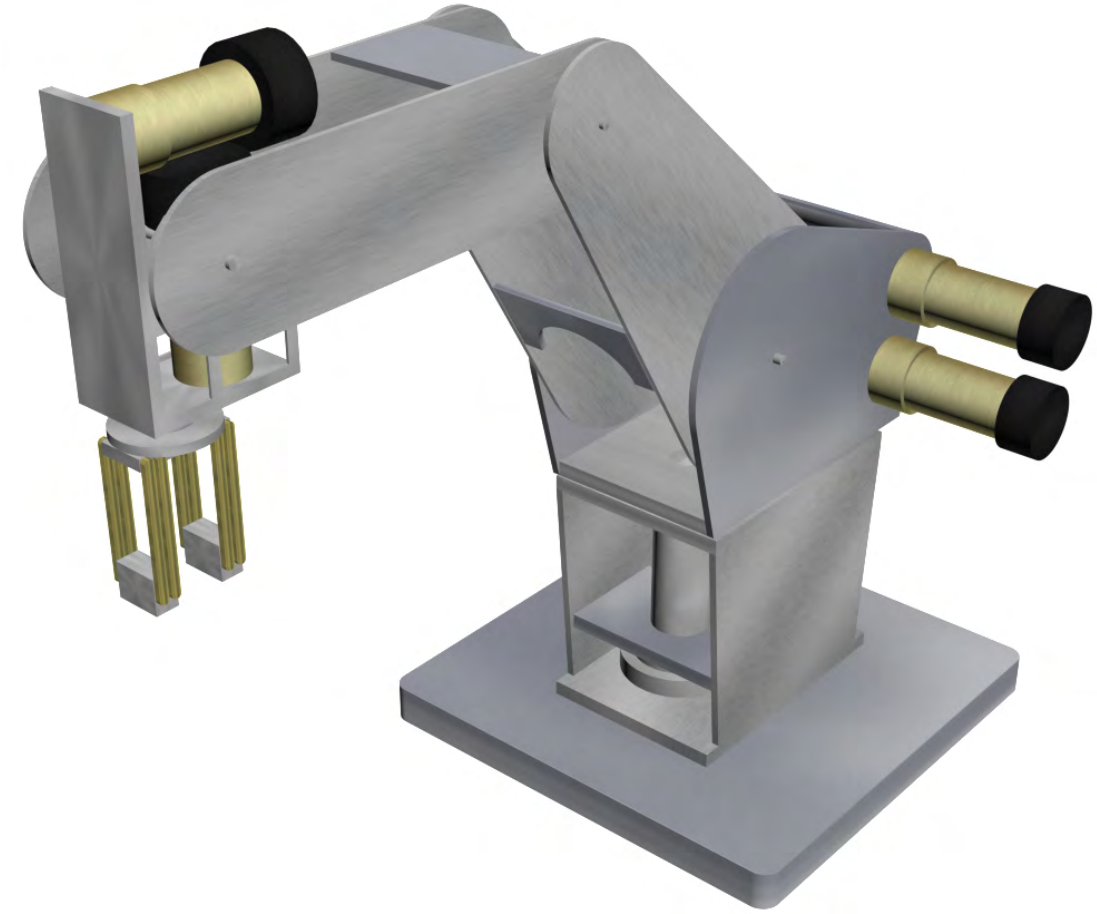


(e) Fino brušeni aluminij



(f) Oštećeni mesing

Konačan 3D model Rhino XR-4 robota uz pridijeljene materijale prikazan je slikom 7.5



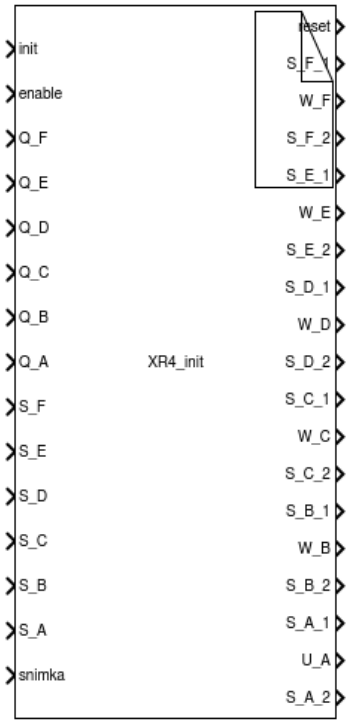
Slika 7.5: Konačan 3D model Rhino XR-4 robota

Objekti su grupirani u funkcionalne cjeline (baza, tijelo, nadlaktica, podlaktica, alat) kako bi se omogućila animacija i praćenje kretnji stvarnog robota.

7.1. Upravljanje više razine

7.1.1. Inicijalizacija

Za određivanje pozicije u motorima *Rhino XR-3* i *Rhino XR-4* robota korišteni su optički inkrementalni enkoderi. Prilikom isključenja *Roboteq* kontrolera s napajanja interni registar kontrolera se prazni te nije moguće odrediti početnu poziciju zglobova. Kako bi upravljanje u povratnoj vezi bilo moguće, potrebno je poznavati poziciju robota u odnosu na referentnu točku (ishodište sustava). Inicijalizacijom se svakom zglobu robota određuje referentna točka. Procesom inicijalizacije upravlja napisana S-funkcija `rhino_init`. [dodatak D] Blok S-funkcije i signali kojima funkcija upravlja tijekom inicijalizacije prikazani su slikom **??**

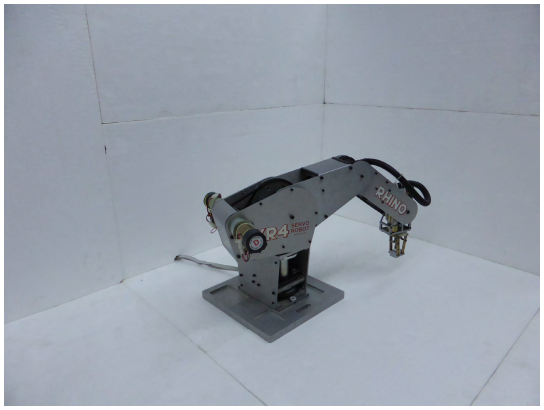
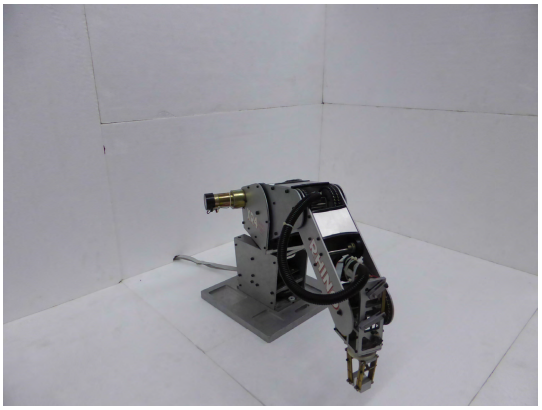
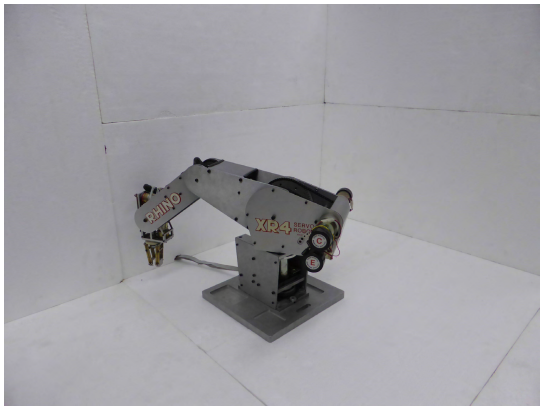
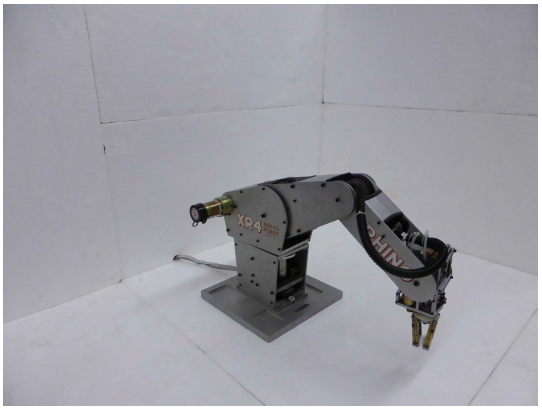
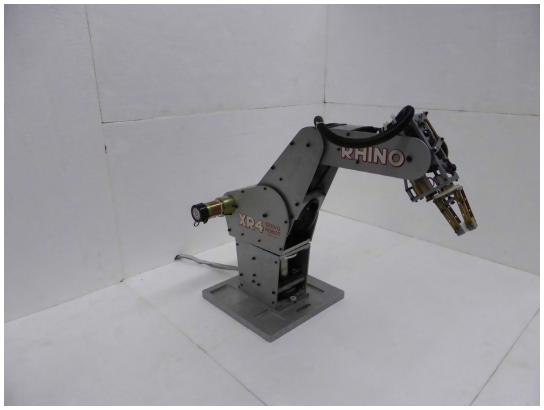


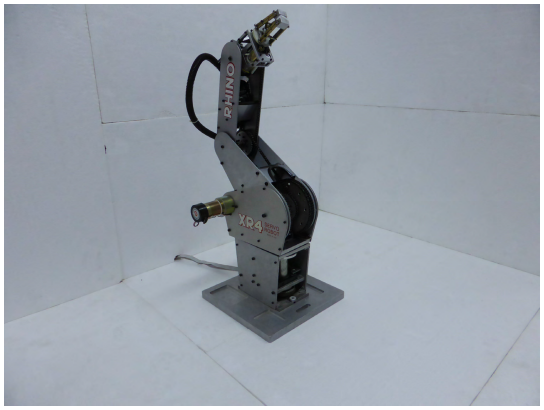
7.1.2. Tijek inicijalizacije

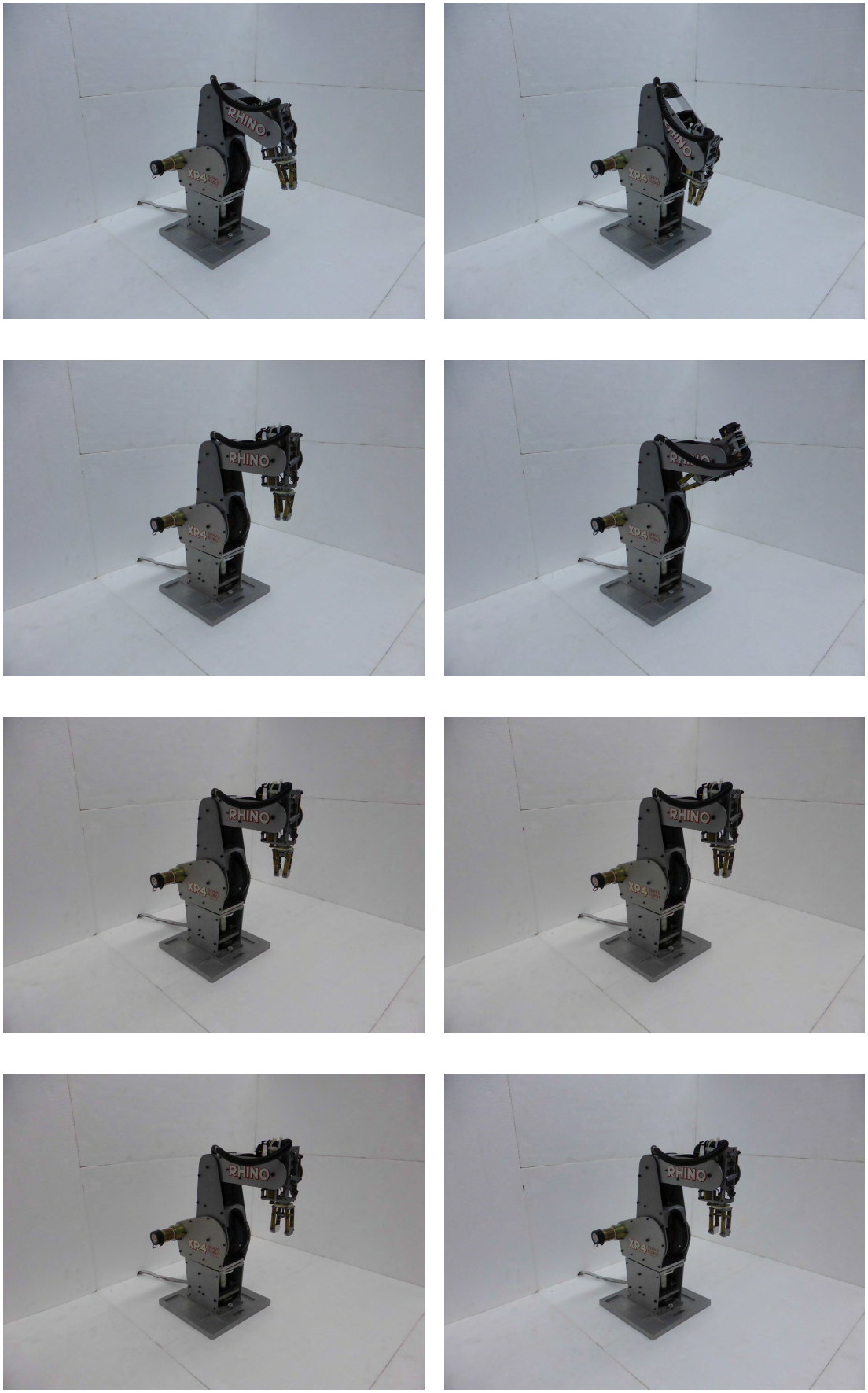
Radni prostor robota određen je graničnim sklopkama. Prilikom pokretanja simulacije, blok će provjeriti stanje simulacije te će, ukoliko sustav nije inicijaliziran, pokrenuti prvu fazu inicijalizacije.

Svakom zglobu dodijeljena su dva koraka inicijalizacije. U prvom koraku robot se stalnom brzinom zgloba približava rubu radnog prostora što je ostvareno preusmjeravanjem reference brzine sa regulatora upravljanja po poziciji na izlazni signal funkcije inicijalizacije. Po okidanju sklopke sustav prelazi u drugu fazu i zglob se vraća u određeno ishodište što je ostvareno preusmjeravanjem reference pozicije sa vanjske reference na izlazni signal funkcije inicijalizacije. U trenutku postizanja ishdišta kreće proces inicijalizacije sljedećeg zgloba i tako sve do posljednjeg neinicijaliziranog zgloba.

Faze inicijalizacije prikazane su slikom 7.7.



Slika 7.7: Koraci inicijalizacije

7.1.3. Kinematika

Direktna kinematika

Problem direktne kinematike odnosi se na to da ako je zadan vektor varijabli zglobova robotskog manipulatora, tada treba odrediti položaj i orijentaciju alata u odnosu prema koordinatnom sustavu pridruženom bazi robota. Stoga je potrebno odrediti matričnu jednadžbu peteroosnog rotacijskog robota *Rhino XR-4*. Kod robota je motor baze pričvršćen okomito prema podlozi. Motori ramena, lakta i poniranja alata pričvršćeni su vodoravno na tijelo robota te rotiraju oko osi baze. Motori za valjanje alata te otvaranje i zatvaranje prstiju hvataljke nalaze se u šaci robota.[**?**]

Taj manipulator ima sljedeće kinematičke parametre:

$$
\begin{array}{llll}
1.os : \theta_1 = q_1, & d_1 = d_1, & \alpha_1 = -\frac{\pi}{2}, & a_1 = 0 \\
2.os : \theta_2 = q_2, & d_2 = 0, & \alpha_2 = 0, & a_2 = a_2 \\
3.os : \theta_3 = q_3, & d_3 = 0, & \alpha_3 = 0, & a_3 = a_3 \\
4.os : \theta_4 = q_4, & d_4 = 0, & \alpha_4 = -\frac{\pi}{2}, & a_4 = a_4 \\
5.os : \theta_5 = q_5, & d_5 = d_5, & \alpha_5 = 0, & a_5 = 0
\end{array}
$$

Za dobivanje matrične jednadžbe takvog robota potrebne su sljedeće matrice homogene transformacije, dobivene uz poznate kinematičke parametre (način zapisa skraćen je pa vrijedi : $sin(q_k) = S_k$ i $cos(q_k) = C_k$):

$$T_0^1 = \begin{bmatrix} C_1 & 0 & -S_1 & 0 \\ S_1 & 0 & C_1 & 0 \\ 0 & -1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_1^2 = \begin{bmatrix} C_2 & -S_2 & 0 & a_2C_2 \\ S_2 & C_2 & 0 & a_2S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_2^3 = \begin{bmatrix} C_3 & -S_3 & 0 & a_3C_3 \\ S_3 & C_3 & 0 & a_3S_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_3^4 = \begin{bmatrix} C_4 & 0 & -S_4 & a_4C_4 \\ S_4 & 0 & C_4 & a_4S_4 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_4^5 = \begin{bmatrix} C_5 & -S_5 & 0 & 0 \\ S_5 & C_5 & 0 & 0 \\ 0 & 0 & 1 & d_5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Matrična jednadžba može se dobiti množenjem matrica homogene transformacije na sljedeći način:

$$T_{baza}^{alat} = T_0^5(q) = T_0^1(q_1) \cdot T_1^2(q_2) \cdot T_2^3(q_3) \cdot T_3^4(q_4) \cdot T_4^5(q_5) \tag{7.1}$$

Ako se koristi skraćeni način zapisa: $sin(q_i + q_j + q_k) = S_{ijk}$ i $cos(q_i + q_j + q_k) = C_{ijk}$, dobiva se sljedeća matrična jednadžba:

$$T_0^5 = \begin{bmatrix} C_1C_{234}C_5 + S_1S_5 & -C_1C_{234}S_5 + S_1C_5 & -C_1S_{234} & C_1(a_2C_2 + a_3C_{23} + a_4C_{234} - d_5S_{234}) \\ S_1C_{234}C_5 - C_1S_5 & -S_1C_{234}S_5 - C_1C_5 & -S_1S_{234} & S_1(a_2C_2 + a_3C_{23} + a_4C_{234} - d_5S_{234}) \\ -S_{234}C_5 & S_{234}S_5 & -C_{234} & d_1 - a_2S_2 - a_3S_{23} - a_4S_{234} - d_5C_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

[14]

Inverzna kinematika

Da bi se definirao zadatak koji robot mora obaviti, potrebno je zadati točke u prostoru kroz koje alat mora proći, a to znači da valja zadati točne vrijednosti zglobova koje će dovesti alat u željenu točku.

Iz matrične jednadžbe manipulatora dobije se sljedeći vektor konfiguracije alata:

$$w(q) = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \\ w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} C_1(a_2C_2 + a_3C_{23} + a_4C_{234} - d_5S_{234}) \\ S_1(a_2C_2 + a_3C_{23} + a_4C_{234} - d_5S_{234}) \\ d_1 - a_2S_2 - a_3S_{23} - a_4S_{234} - d_5C_{234} \\ -e^{\frac{q_5}{\pi}}C_1S_{234} \\ -e^{\frac{q_5}{\pi}}S_1S_{234} \\ -e^{\frac{q_5}{\pi}}C_{234} \end{bmatrix} \tag{7.2}$$

Vektor konfiguracije alata sastoji se od tri kartezijske koordinate položaja ($w_1 = y$, $w_2 = x$ i $w_3 = z$) i tri komponente orijentacije alata ($w_4 - w_6$) koji predstavljaju kuteve skretanja, poniranja i valjanja (Eulerovi kutovi). [**?** 2, 12]

Baza

Sada se iz vektora konfiguracije alata (7.2) može vidjeti da se najjednostavnije, tj. dijeljenjem komponenti w_2 i w_1 te primjenom funkcije atan2(y,x), može dobiti izraz za zakret baze q_1:

$$q_1 = atan2(w_2, w_1) \tag{7.3}$$

Lakat

Zakret lakta q_3 najteže se odrediti, jer je povezan sa zakretom ramena q_2 i kutom poniranja alata q_4. Zbroj tih triju kutova $q_{234} = q_2 + q_3 + q_4$ naziva se ukupnim kutom

poniranja alata te predstavlja kut poniranja alata mjeren prema radnoj površini. Na takav način zakret lakta q_3 može se izračunati pomoću ukupnog kuta poniranja alata q_{234}.

Iz izraza (7.2) može se dobiti sljedeća veza između posljednjih triju komponenata w_4, w_5 i w_6: $S_{234}/C_{234} = -(C_1w_4 + S_1w_5)/ - w_6$. Primjenom funkcije atan2(y,x), uz poznat zakret baze q_1, slijedi izraz za ukupni kut poniranja alata q_{234}:

$$q_{234} = atan2[-(C_1w_4 + S_1w_5), -w_6] \tag{7.4}$$

Sada je potrebno, za određivanje zakreta lakta q_3, uvesti ove dvije varijable:

$$p_1 = C_1w_1 + S_1w_2 - a_4C_{234} + d_5S_{234} \tag{7.5}$$

$$p_2 = d_1 - a_4S_{234} - d_5C_{234} - w_3 \tag{7.6}$$

Uvrštavanjem poznatih komponenti w_1 i w_2 iz (7.2) u prethodne izraze dobije se :

$$p_1 = a_2C_2 + a_3C_{23} \tag{7.7}$$

$$p_2 = a_2S_2 + a_3S_{23} \tag{7.8}$$

iz čega slijedi:

$$p_1^2 + p_2^2 = a_2^2 + 2a_2a_3C_3 + a_3^2 \tag{7.9}$$

Iz te se jednakosti može dobiti izraz za zakret lakta:

$$q_3 = \pm arccos\frac{p_1^2 + p_2^2 - a_2^2 - a_3^2}{2a_2a_3} \tag{7.10}$$

Na takav način dobivaju se dva rješenja za kut zakreta lakta q_3, pri čemu se pozitivno rješenje naziva gornjim položajem lakta (veća udaljenost lakta od radne površine i mogućnost izbjegavanja prepreka unutar radne okolice), a negativno donjim položajem lakta. [**?**]

Rame

Za dobivanje zakreta ramena q_2 komponente p_1 i p_2 zapišu se na sljedeći način, uz upotrebu trigonometrijskih funkcija zbroja dvaju kutova :

$$p_1 = (a_2 + a_3C_3)C_2 - a_3S_2S_3 \tag{7.11}$$

$$p_2 = (a_2 + a_3C_3)S_2 + a_3C_2S_3 \tag{7.12}$$

Izraze li se sada nepoznanice S_2 i C_2 iz gornjih jednadžbi te se zatim podjele dobije se izraz za q_2:

$$q_2 = atan2[p_2(a_2 + a_3C_3) - p_1a_3S_3, p_1(a_2 + a_3C_3) + p_2a_3S_3] \tag{7.13}$$

Poniranje alata

Ako se od ukupnog kuta poniranja alata q_{234} oduzmu zakret ramena q_2 i zakret lakta q_3, rezultat je kut poniranja alata q_4:

$$q_4 = q_{234} - q_2 - q_3 \tag{7.14}$$

Valjanje alata

Na kraju je potrebno odrediti kut valjanja alata q_5. Taj se kut određuje pomoću posljednjih triju komponenata w_4, w_5 i w_6 u izrazu (7.2) na sljedeći način:

$$q_5 = \pi \ln \sqrt{w_4^2 + w_5^2 + w_6^2} \tag{7.15}$$

Ukupno rješenje

Na temelju izraza (7.2)-(7.15) dobiva se ovo rješenje inverznog kinematičkog problema peteroosnog rotacijskog robota RHINO XR-4:

$$\begin{aligned}
q_1 &= atan2(w_2, w_1) \\
q_{234} &= atan2[-(C_1w_4 + S_1w_5), -w_6] \\
p_1 &= C_1w_1 + S_1w_2 - a_4C_{234} + d_5S_{234} \\
p_2 &= d_1 - a_4S_{234} - d_5C_{234} - w_3 \\
q_3 &= \pm arccos\frac{p_1^2 + p_2^2 - a_2^2 - a_3^2}{2a_2a_3} \\
q_2 &= atan2[p_2(a_2 + a_3C_3) - p_1a_3S_3, p_1(a_2 + a_3C_3) + p_2a_3S_3] \\
q_4 &= q_{234} - q_2 - q_3 \\
q_5 &= \pi \ln \sqrt{w_4^2 + w_5^2 + w_6^2}
\end{aligned}$$

Inverzna kinematika ostvarena je *Matlab* funkcijom `inverzna_kinematika` [14]

7.1.4. Planiranje trajektorije

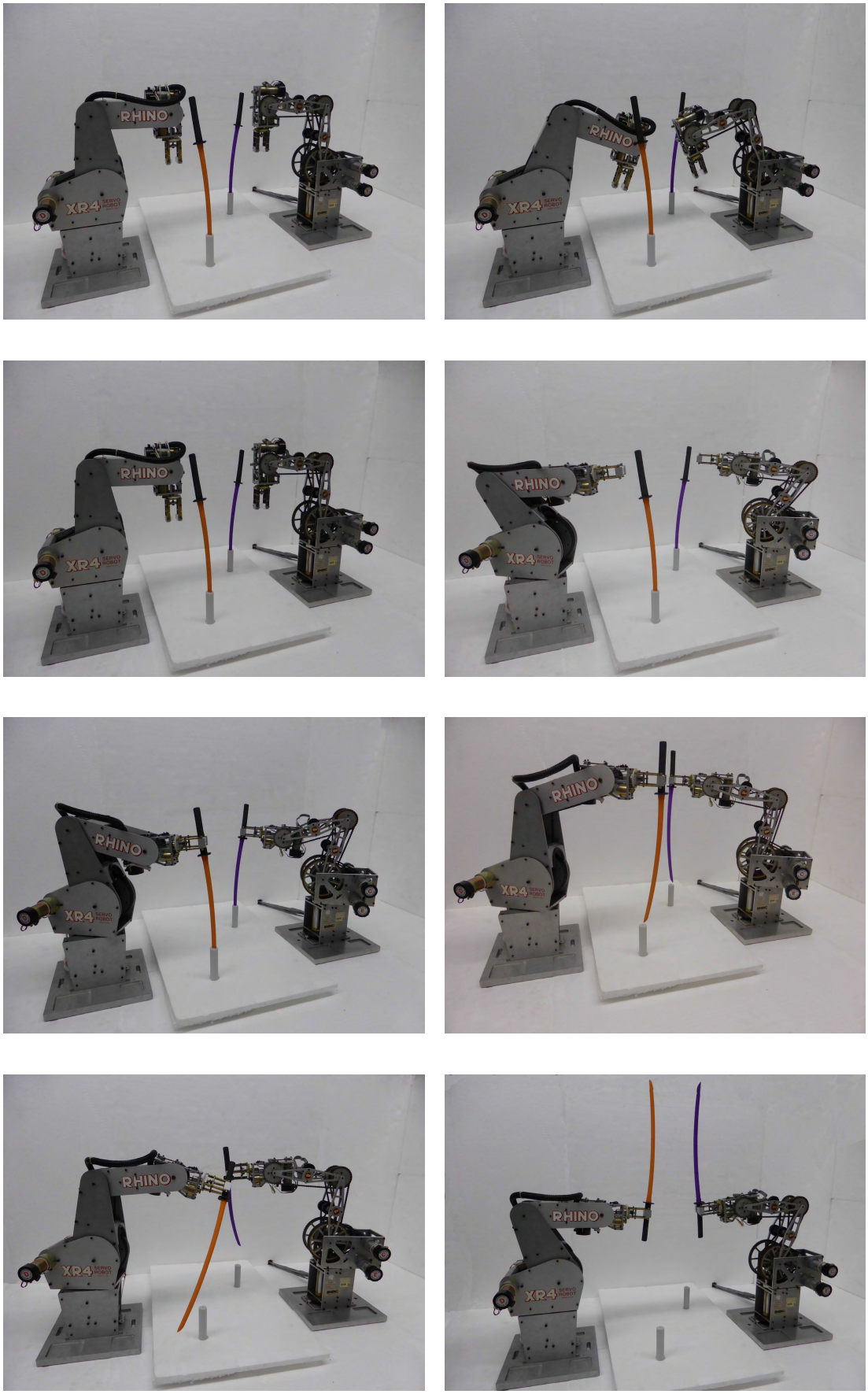
Planiranje trajektorije ostvareno je funkcijama `taylor_path` koja za zadane dvije točke u prostoru vektora konfiguracije alata w , toleranciju i funkcije inverzne i direktne kinematike ubacuje točke u prostoru koordinata zglobova uz koje će odstupanje u prostoru vektora konfiguracije alata odstupanje biti u okviru tolerancije te funkcijom `trajectory` koja za zadane točke u prostoru koordinata zglobova i parametrijsko vrijeme vraća točke u prostoru koordinata zglobova i brzine u tim točkama, te vremenske trenutke u kojima se pozicije i brzine postižu.

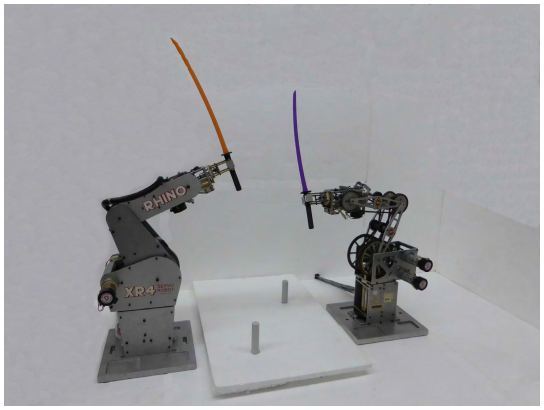
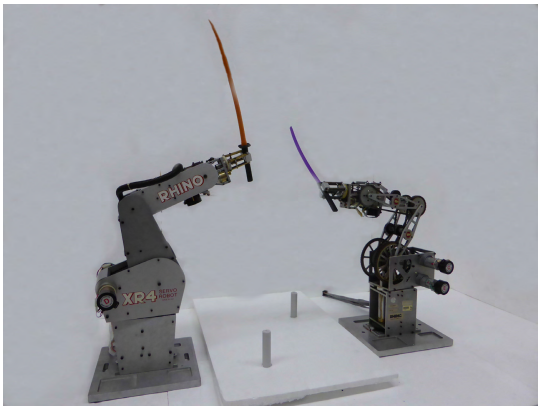
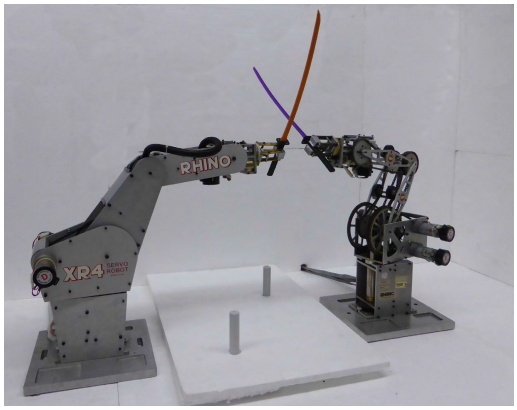
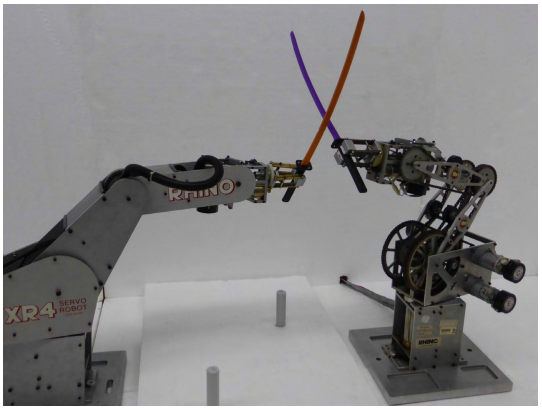
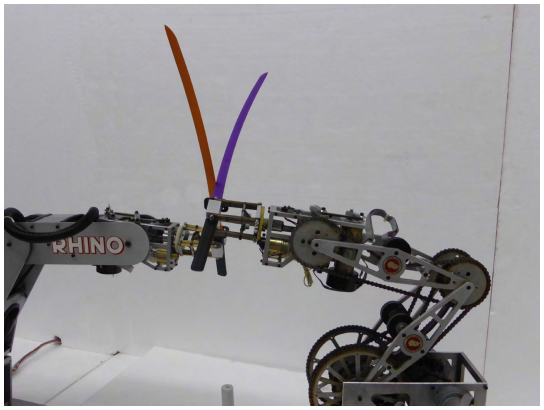
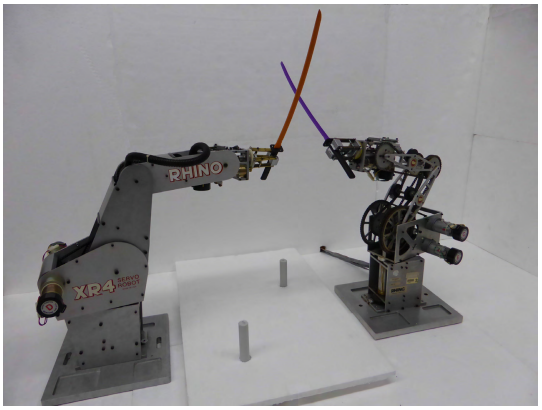
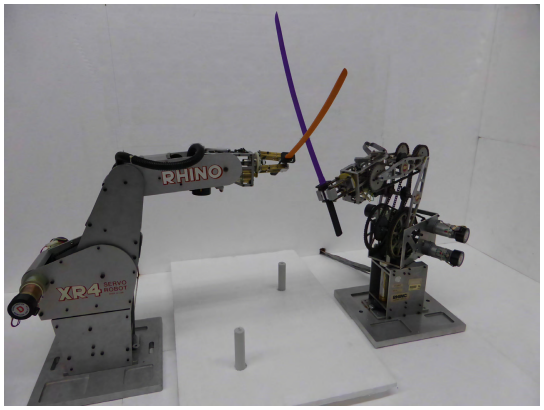
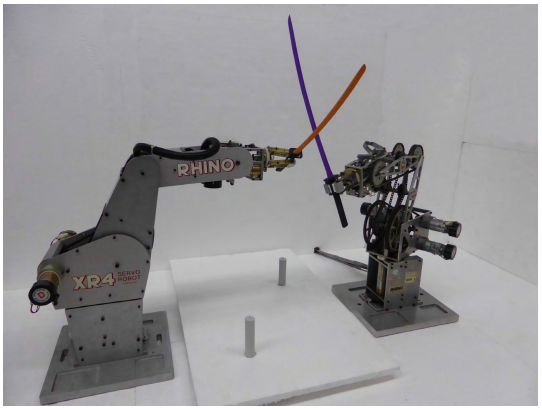
Praćenje trajektorije

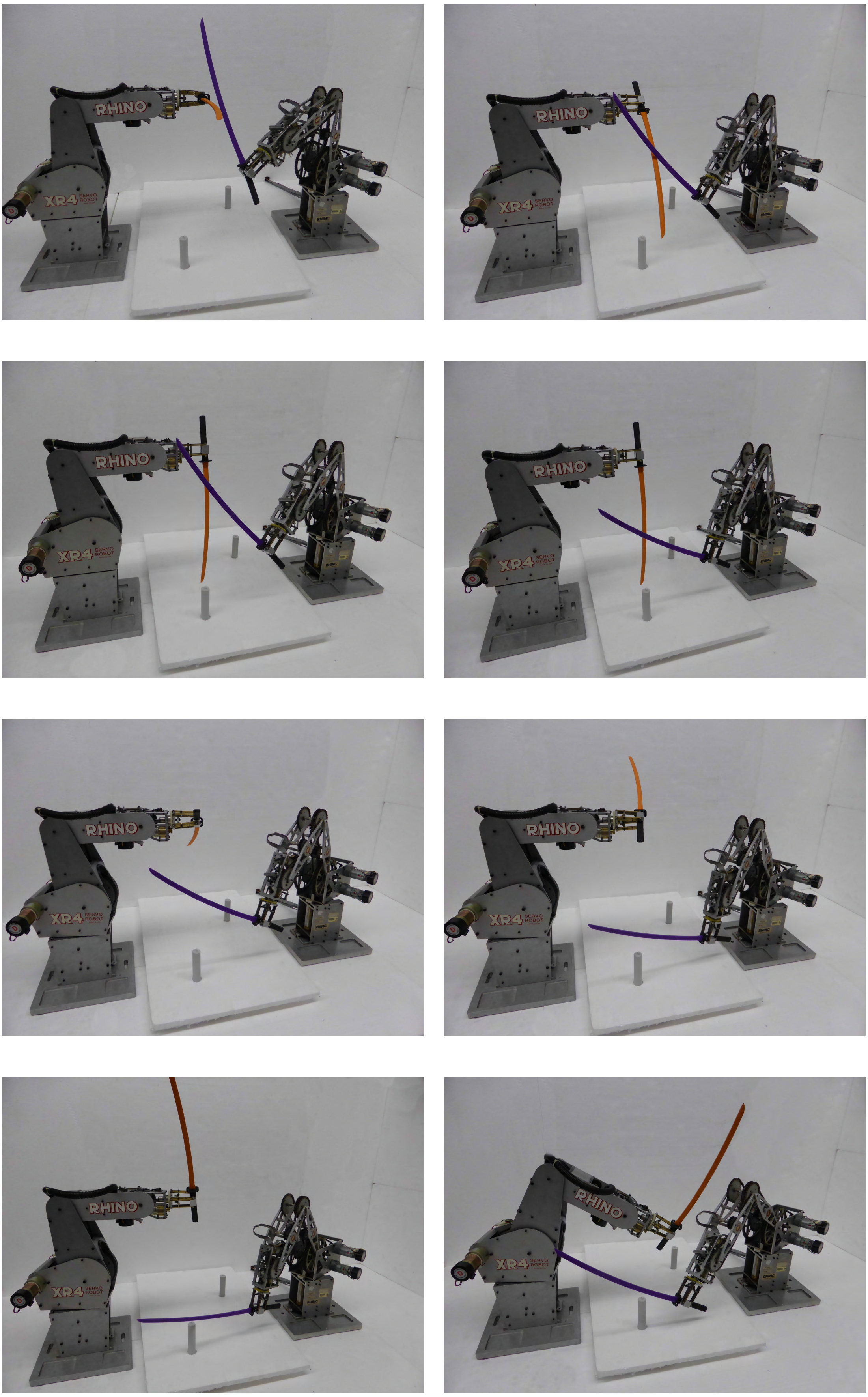
Praćenje trajektorije ostvareno je S-funkcijom koja kao argument prima vektore u kojima su sadržane pozicije, brzine i vremena isplanirane trajektorije te potom u svakom koraku kao referencu zadaje poziciju definiranu trajektorijom, a brzina definirana trajektorijom propuštena je na referencu podređene petlje upravljanja po brzini vrtnje svakog od regulatora.

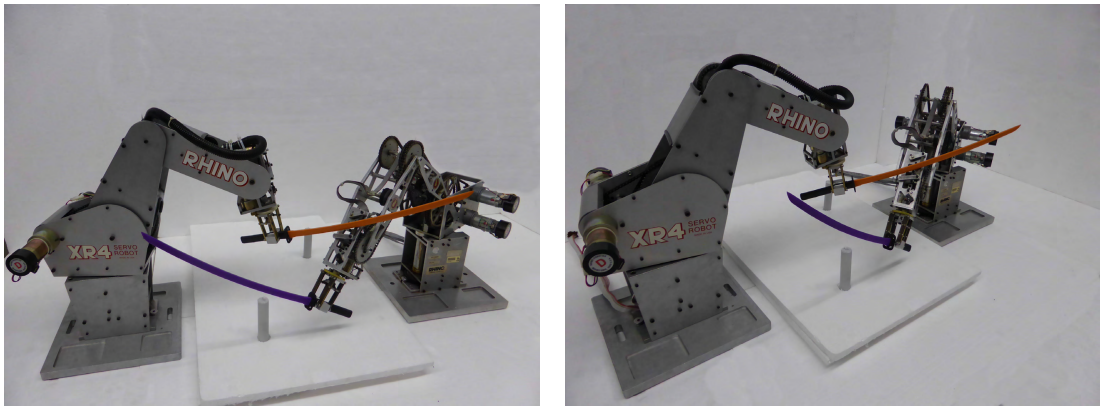
7.1.5. Sinkronizirano izvođenje trajektorije

Sinkronizirano praćenje trajektorije dva *Rhino* robota ostvareno je zadavanjem segmenata trajektorije s jednakim parametrijskim vremenima uz jednaka vremena uzorkovanja. Navedenim kreiranjem trajektorije će roboti u krajnju točku segmenta doći u isto vrijeme, čime je ostvarena sinkronizacija robota i na svakom od uzorkovanih trenutaka za vrijeme praćenja segmenta trajektorije.


RHINO
XR4


RHINO
XR4
SERVO ROBOT



Slika 7.11: Koordinirano praćenje složene trajektorije

8. Zaključak

U okviru rada osuvremenjen je postojeći zastarjeli robotski sustav namijenjen izvođenju laboratorijskih vježbi kolegija robotike na Fakultetu elektrotehnike i računarstva. Uočen nedostatak kvalitetne i pouzdane razvojne okoline, integrirane s programskim paketima *Matlab* i *Simulink* potakao je implementaciju vlastitog rješenja. Vlastita razvojna platforma temeljena je na suvremenim servokontrolerima tvrtke *Roboteq* i programskim rješenjima otvorenog koda. Navedeni pristup osigurava visoku kvalitetu i stabilnost implementiranog sustava upravljanja u stvarnom vremenu. Pri izradi rada praćene su preporuke vodećih autoriteta robotike i automatizacije. Algoritmi upravljanja više i niže razine, karakteristični za laboratorijske vježbe kolegija robotike na Fakultetu elektrotehnike i računarstva, implementirani su na razvijenom eksperimentalnom sustavu. Mogućnosti upravljačkih struktura demonstrirane su koordiniranom složenom operacijom mačevanja ostvarenom sinkroniziranim praćenjem složenih trajektorija.

9. Zahvala

Zahvaljujem se mentoru prof. dr. sc. Zdenku Kovačiću na ukazanom povjerenju i poticaju za rad, te dr. sc. Damjanu Mikliću i dr. sc. Matku Orsagu na pomoći pruženoj pri izradi rada.

DODATAK A

```
1 Dim switch_1 as Integer
2 Dim switch_2 as Integer
3 Dim offset_1 as Integer
4 Dim offset_2 as Integer
5 switch_1 = 0
6 switch_2 = 0
7 offset_1 = 0
8 offset_2 = 0
9 setCommand(_VAR, 3, 1)
10 setCommand(_VAR, 4, 1)
11
12
13 std_loop:
14 'debouncing filter
15 If getValue(_DIN, 3) = 0 Then switch_1++ Else switch_1 = 0
16 If getValue(_DIN, 4) = 0 Then switch_2++ Else switch_2 = 0
17 If (switch_1>=5) And (getValue(_VAR, 3) == 1) Then
18   setCommand(_VAR, 3, 0)
19   offset_1 = getValue(_ABCNTR, 1)-2155
20 End If
21
22 If (switch_2>=5) And (getValue(_VAR, 4) == 1) Then
23   setCommand(_VAR, 4, 0)
24   offset_2 = getValue(_ABCNTR, 2) + 1520
25 End If
26
27
28 setCommand(_VAR, 1, getValue(_ABCNTR, 1)-offset_1)
29 setCommand(_VAR, 2, getValue(_ABCNTR, 2)-offset_2)
30
31 setCommand(_GO, 1, getValue(_VAR, 9))
32 setCommand(_GO, 2, getValue(_VAR, 10))
33
34 wait(1)
35 goto std_loop
```

kontroler1.mbs

DODATAK B

```
1  #define S_FUNCTION_NAME  moja
2  #define S_FUNCTION_LEVEL 2
3
4
5  #include "simstruc.h"
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <string.h>
9  #include <math.h>
10 #include "bci.h"
11 #include "rhino.h"
12
13
14
15
16 BCI_BRD_HDL CANBoard;
17 int ret;
18 int flag = 0;
19
20 #define CAN0_ID 0x10
21 #define CAN1_ID 0x11
22 #define MESSAGES_NUMBER 10
23
24 int BCI_Status2Str (UINT16 Status, char *StatusStr);
25 int BCI_ShowCANMsg (BCI_ts_CanMsg * CANMsg);
26 int BCI_CreateCANMsg (BCI_ts_CanMsg * CANMsg, UINT32 ID, UINT8 * Data,
27                       UINT8 DLC,
28                       UINT8 MFF);
29 void BCI_MDelay (unsigned int msec);
30
31
32 static void mdlInitializeSizes(SimStruct *S)
33 {
34     ssSetNumSFcnParams(S, 0);
35     if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
36         return;
37     }
38
39     ssSetNumContStates(S, 0);
40     ssSetNumDiscStates(S, OUTVAR_NUM+1);
41
42     if (!ssSetNumInputPorts(S, INVAR_NUM+2)) return;
43     ssSetInputPortWidth(S,  0, 1);
44     ssSetInputPortWidth(S,  1, 1);
45     ssSetInputPortWidth(S,  2, 1);
46     ssSetInputPortWidth(S,  3, 1);
47     ssSetInputPortWidth(S,  4, 1);
48     ssSetInputPortWidth(S,  5, 1);
49     ssSetInputPortWidth(S,  6, 1);
50     ssSetInputPortWidth(S,  7, 1);
51     ssSetInputPortWidth(S,  8, 1);
52     ssSetInputPortWidth(S,  9, 1);
```

```
53      ssSetInputPortWidth(S, 10, 1);
54      ssSetInputPortWidth(S, 11, 1);
55      ssSetInputPortWidth(S, 12, 1);
56      ssSetInputPortWidth(S, 13, 1);
57
58      ssSetInputPortRequiredContiguous(S,  0, true);
59      ssSetInputPortRequiredContiguous(S,  1, true);
60      ssSetInputPortRequiredContiguous(S,  2, true);
61      ssSetInputPortRequiredContiguous(S,  3, true);
62      ssSetInputPortRequiredContiguous(S,  4, true);
63      ssSetInputPortRequiredContiguous(S,  5, true);
64      ssSetInputPortRequiredContiguous(S,  6, true);
65      ssSetInputPortRequiredContiguous(S,  7, true);
66      ssSetInputPortRequiredContiguous(S,  8, true);
67      ssSetInputPortRequiredContiguous(S,  9, true);
68      ssSetInputPortRequiredContiguous(S, 10, true);
69      ssSetInputPortRequiredContiguous(S, 11, true);
70      ssSetInputPortRequiredContiguous(S, 12, true);
71      ssSetInputPortRequiredContiguous(S, 13, true);
72
73      ssSetInputPortDirectFeedThrough(S,  0, 1);
74      ssSetInputPortDirectFeedThrough(S,  1, 1);
75      ssSetInputPortDirectFeedThrough(S,  2, 1);
76      ssSetInputPortDirectFeedThrough(S,  3, 1);
77      ssSetInputPortDirectFeedThrough(S,  4, 1);
78      ssSetInputPortDirectFeedThrough(S,  5, 1);
79      ssSetInputPortDirectFeedThrough(S,  6, 1);
80      ssSetInputPortDirectFeedThrough(S,  7, 1);
81      ssSetInputPortDirectFeedThrough(S,  8, 1);
82      ssSetInputPortDirectFeedThrough(S,  9, 1);
83      ssSetInputPortDirectFeedThrough(S, 10, 1);
84      ssSetInputPortDirectFeedThrough(S, 11, 1);
85      ssSetInputPortDirectFeedThrough(S, 12, 1);
86      ssSetInputPortDirectFeedThrough(S, 13, 1);
87
88
89      if (!ssSetNumOutputPorts(S, OUTVAR_NUM)) return;
90      ssSetOutputPortWidth(S,  0, 1);
91      ssSetOutputPortWidth(S,  1, 1);
92      ssSetOutputPortWidth(S,  2, 1);
93      ssSetOutputPortWidth(S,  3, 1);
94      ssSetOutputPortWidth(S,  4, 1);
95      ssSetOutputPortWidth(S,  5, 1);
96      ssSetOutputPortWidth(S,  6, 1);
97      ssSetOutputPortWidth(S,  7, 1);
98      ssSetOutputPortWidth(S,  8, 1);
99      ssSetOutputPortWidth(S,  9, 1);
100     ssSetOutputPortWidth(S, 10, 1);
101     ssSetOutputPortWidth(S, 11, 1);
102     ssSetOutputPortWidth(S, 12, 1);
103     ssSetOutputPortWidth(S, 13, 1);
104     ssSetOutputPortWidth(S, 14, 1);
105     ssSetOutputPortWidth(S, 15, 1);
106     ssSetOutputPortWidth(S, 16, 1);
```

```
107     ssSetOutputPortWidth(S, 17, 1);
108     ssSetOutputPortWidth(S, 18, 1);
109     ssSetOutputPortWidth(S, 19, 1);
110     ssSetOutputPortWidth(S, 20, 1);
111     ssSetOutputPortWidth(S, 21, 1);
112     ssSetOutputPortWidth(S, 22, 1);
113     ssSetOutputPortWidth(S, 23, 1);
114     ssSetOutputPortWidth(S, 24, 1);
115
116     ssSetNumSampleTimes(S, 1);
117     ssSetNumRWork(S, 0);
118     ssSetNumIWork(S, 0);
119     ssSetNumPWork(S, 0);
120     ssSetNumModes(S, 0);
121     ssSetNumNonsampledZCs(S, 0);
122
123
124     ssSetSimStateCompliance(S, USE_DEFAULT_SIM_STATE);
125
126     ssSetOptions(S, 0);
127 }
128
129
130
131
132 static void mdlInitializeSampleTimes(SimStruct *S)
133 {
134     ssSetSampleTime(S, 0, 0.005);
135     ssSetOffsetTime(S, 0, 0.0);
136
137 }
138
139
140
141 #define MDL_INITIALIZE_CONDITIONS
142 #if defined(MDL_INITIALIZE_CONDITIONS)
143
144   static void mdlInitializeConditions(SimStruct *S)
145   {
146     int i = 0;
147     real_T *Rhino = ssGetRealDiscStates(S);
148     for (i = 0; i < OUTVAR_NUM+1; i++)
149       Rhino[i] = 0;
150   }
151
152
153 #endif
154
155
156
157 #define MDL_START
158 #if defined(MDL_START)
159
160   static void mdlStart(SimStruct *S)
```

```
161   {
162     char DevFileName[64] = "/dev/can0";
163     extern BCI_BRD_HDL CANBoard;
164     extern int ret;
165     ret = BCI_OpenBoard (&CANBoard, DevFileName);
166
167   }
168 #endif
169
170
171
172
173 static void mdlOutputs(SimStruct *S, int_T tid)
174 {
175     extern int ret, flag;
176     BCI_ts_CanMsg msg_r, msg_s;
177     int value1, value2;
178     // XR4
179     const real_T *XR4VOLF = (const real_T*) ssGetInputPortSignal(S,XR4_VOL_F);
180     const real_T *XR4VOLE = (const real_T*) ssGetInputPortSignal(S,XR4_VOL_E);
181     const real_T *XR4VOLD = (const real_T*) ssGetInputPortSignal(S,XR4_VOL_D);
182     const real_T *XR4VOLC = (const real_T*) ssGetInputPortSignal(S,XR4_VOL_C);
183     const real_T *XR4VOLB = (const real_T*) ssGetInputPortSignal(S,XR4_VOL_B);
184     const real_T *XR4VOLA = (const real_T*) ssGetInputPortSignal(S,XR4_VOL_A);
185     const real_T *XR4enable = (const real_T*) ssGetInputPortSignal(S,12);
186     const real_T *XR3enable = (const real_T*) ssGetInputPortSignal(S,13);
187
188     // XR3
189     const real_T *XR3VOLF = (const real_T*) ssGetInputPortSignal(S,XR3_VOL_F);
190     const real_T *XR3VOLE = (const real_T*) ssGetInputPortSignal(S,XR3_VOL_E);
191     const real_T *XR3VOLD = (const real_T*) ssGetInputPortSignal(S,XR3_VOL_D);
192     const real_T *XR3VOLC = (const real_T*) ssGetInputPortSignal(S,XR3_VOL_C);
193     const real_T *XR3VOLB = (const real_T*) ssGetInputPortSignal(S,XR3_VOL_B);
194     const real_T *XR3VOLA = (const real_T*) ssGetInputPortSignal(S,XR3_VOL_A);
195
196
197
198     // XR4
199     real_T        *XR4POSF = ssGetOutputPortSignal(S,XR4_POS_F);
200     real_T        *XR4POSE = ssGetOutputPortSignal(S,XR4_POS_E);
201     real_T        *XR4LSWF = ssGetOutputPortSignal(S,XR4_LSW_F);
202     real_T        *XR4LSWE = ssGetOutputPortSignal(S,XR4_LSW_E);
203     real_T        *XR4POSD = ssGetOutputPortSignal(S,XR4_POS_D);
204     real_T        *XR4POSC = ssGetOutputPortSignal(S,XR4_POS_C);
205     real_T        *XR4LSWD = ssGetOutputPortSignal(S,XR4_LSW_D);
206     real_T        *XR4LSWC = ssGetOutputPortSignal(S,XR4_LSW_C);
207     real_T        *XR4POSB = ssGetOutputPortSignal(S,XR4_POS_B);
208     real_T        *XR4POSA = ssGetOutputPortSignal(S,XR4_POS_A);
209     real_T        *XR4LSWB = ssGetOutputPortSignal(S,XR4_LSW_B);
210     real_T        *XR4LSWA = ssGetOutputPortSignal(S,XR4_LSW_A);
211
212     // XR3
213     real_T        *XR3POSF = ssGetOutputPortSignal(S,XR3_POS_F);
214     real_T        *XR3POSE = ssGetOutputPortSignal(S,XR3_POS_E);
```

```
215     real_T        *XR3LSWF = ssGetOutputPortSignal(S,XR3_LSW_F);
216     real_T        *XR3LSWE = ssGetOutputPortSignal(S,XR3_LSW_E);
217     real_T        *XR3POSD = ssGetOutputPortSignal(S,XR3_POS_D);
218     real_T        *XR3POSC = ssGetOutputPortSignal(S,XR3_POS_C);
219     real_T        *XR3LSWD = ssGetOutputPortSignal(S,XR3_LSW_D);
220     real_T        *XR3LSWC = ssGetOutputPortSignal(S,XR3_LSW_C);
221     real_T        *XR3POSB = ssGetOutputPortSignal(S,XR3_POS_B);
222     real_T        *XR3POSA = ssGetOutputPortSignal(S,XR3_POS_A);
223     real_T        *XR3LSWB = ssGetOutputPortSignal(S,XR3_LSW_B);
224     real_T        *XR3LSWA = ssGetOutputPortSignal(S,XR3_LSW_A);
225
226     real_T        *initialized = ssGetOutputPortSignal(S,init);
227
228
229
230
231     UINT8 Controller = 0, PacketData[8];
232     if (flag == 0) {
233       BCI_InitCan (CANBoard, Controller, BCI_1000KB, 0);
234       BCI_ConfigRxQueue (CANBoard, Controller, BCI_POLL_MODE);
235
236       BCI_SetAccMask (CANBoard, Controller, BCI_11B_MASK, 0, BCI_ACC_ALL);
237
238       BCI_StartCan (CANBoard, Controller);
239     }
240     flag = 1;
241
242     real_T *Rhino = ssGetRealDiscStates(S);
243
244     // XR4
245     XR4POSF[0] = -Rhino[XR4_POS_F] / (XR4_CPR_F / 360.0);
246     XR4POSE[0] = Rhino[XR4_POS_E] / (XR4_CPR_E / 360.0);
247     XR4LSWF[0] = !(Rhino[XR4_LSW_F]);
248     XR4LSWE[0] = !(Rhino[XR4_LSW_E]);
249     XR4POSD[0] = -Rhino[XR4_POS_D] / (XR4_CPR_D / 360.0)-Rhino[XR4_POS_E] /
            (XR4_CPR_E / 360.0);
250     XR4POSC[0] = -Rhino[XR4_POS_C] / (XR4_CPR_C / 360.0)+Rhino[XR4_POS_D] /
            (XR4_CPR_D / 360.0);
251     XR4LSWD[0] = !((int)Rhino[XR4_LSW_D]%2);
252     XR4LSWC[0] = !((int)Rhino[XR4_LSW_C]%2);
253     XR4POSB[0] = Rhino[XR4_POS_B] / (XR4_CPR_B / 360.0);
254     XR4POSA[0] = Rhino[XR4_POS_A];
255     XR4LSWB[0] = !((int)Rhino[XR4_LSW_B]%2);
256     XR4LSWA[0] = !((int)Rhino[XR4_LSW_A]%2);
257
258
259     // XR3
260     XR3POSF[0] = Rhino[XR3_POS_F] / (XR3_CPR_F / 360.0);
261     XR3POSE[0] = Rhino[XR3_POS_E] / (XR3_CPR_E / 360.0);
262     XR3LSWF[0] = !((int)Rhino[XR3_LSW_F]%2);
263     XR3LSWE[0] = !((int)Rhino[XR3_LSW_E]%2);
264     XR3POSD[0] = -Rhino[XR3_POS_D] / (XR3_CPR_D / 360.0)-Rhino[XR3_POS_E] /
            (XR3_CPR_E / 360.0);
```

```
265     XR3POSC[0] = -Rhino[XR3_POS_C] / (XR3_CPR_C / 360.0)+Rhino[XR3_POS_D] /
            (XR3_CPR_D / 360.0);
266     XR3LSWD[0] = !((int)Rhino[XR3_LSW_D]%2);
267     XR3LSWC[0] = !((int)Rhino[XR3_LSW_C]%2);
268     XR3POSB[0] = -Rhino[XR3_POS_B] / (XR3_CPR_B / 360.0);
269     XR3POSA[0] = Rhino[XR3_POS_A];
270     XR3LSWB[0] = !((int)Rhino[XR3_LSW_B]%2);
271     XR3LSWA[0] = !((int)Rhino[XR3_LSW_A]%2);
272
273
274     initialized[0] =  0;
275
276     if (abs(XR4VOLF[0])<=12 && abs(XR4VOLE[0]) <= 12 && XR4enable[0])
277       packMessage(&msg_s, XR4_VOL_F, XR4_VOL_E, (int)(-XR4VOLF[0] / (float)12 *
              1000), (int)( XR4VOLE[0] / (float)12 * 1000));
278     else
279       packMessage(&msg_s, XR4_VOL_F, XR4_VOL_E, 0, 0);
280
281     BCI_TransmitCanMsg (CANBoard, Controller, &msg_s);
282
283
284     if (abs(XR4VOLD[0])<=12 && abs(XR4VOLC[0]) <= 12 && XR4enable[0])
285       packMessage(&msg_s, XR4_VOL_D, XR4_VOL_C, (int)(-XR4VOLD[0] / (float)12 *
              1000), (int)( -XR4VOLC[0] / (float)12 * 1000));
286     else
287       packMessage(&msg_s, XR4_VOL_D, XR4_VOL_C, 0, 0);
288
289     BCI_TransmitCanMsg (CANBoard, Controller, &msg_s);
290
291     if (abs(XR4VOLB[0])<=12 && abs(XR4VOLA[0]) <= 12 && XR4enable[0])
292       packMessage(&msg_s, XR4_VOL_B, XR4_VOL_A, (int)(XR4VOLB[0] / (float)12 *
              1000), (int)( XR4VOLA[0] / (float)12 * 1000));
293     else
294       packMessage(&msg_s, XR4_VOL_B, XR4_VOL_A, 0, 0);
295
296     BCI_TransmitCanMsg (CANBoard, Controller, &msg_s);
297
298
299     if (abs(XR3VOLF[0])<=12 && abs(XR3VOLE[0]) <= 12 && XR3enable[0])
300       packMessage(&msg_s, XR3_VOL_F, XR3_VOL_E, (int)(-XR3VOLF[0] / (float)12 *
              1000), (int)( -XR3VOLE[0] / (float)12 * 1000));
301     else
302       packMessage(&msg_s, XR3_VOL_F, XR3_VOL_E, 0, 0);
303
304     BCI_TransmitCanMsg (CANBoard, Controller, &msg_s);
305
306
307     if (abs(XR3VOLD[0])<=12 && abs(XR3VOLC[0]) <= 12 && XR3enable[0])
308       packMessage(&msg_s, XR3_VOL_D, XR3_VOL_C, (int)(XR3VOLD[0] / (float)12 *
              1000), (int)( XR3VOLC[0] / (float)12 * 1000));
309     else
310       packMessage(&msg_s, XR3_VOL_D, XR3_VOL_C, 0, 0);
311     BCI_TransmitCanMsg (CANBoard, Controller, &msg_s);
312
```

```
313       if (abs(XR3VOLB[0])<=12 && abs(XR3VOLA[0]) <= 12 && XR3enable[0])
314         packMessage(&msg_s, XR3_VOL_B, XR3_VOL_A, (int)(XR3VOLB[0] / (float)12 *
                1000), (int)( XR3VOLA[0] / (float)12 * 1000));
315       else
316         packMessage(&msg_s, XR3_VOL_B, XR3_VOL_A, 0, 0);
317
318       BCI_TransmitCanMsg (CANBoard, Controller, &msg_s);
319
320     }
321
322
323
324 #define MDL_UPDATE
325 #if defined(MDL_UPDATE)
326
327   static void mdlUpdate(SimStruct *S, int_T tid)
328   {
329
330     real_T *Rhino = ssGetRealDiscStates(S);
331
332     UINT8 Controller = 0;
333     int valueID1, valueID2, value1, value2;
334     BCI_ts_CanMsg msg_r;
335
336
337     while (BCI_ReceiveCanMsg(CANBoard, Controller, &msg_r, BCI_NO_WAIT) == BCI_OK)
338     {
339       unpackMessage(msg_r, &valueID1, &valueID2, &value1, &value2);
340       Rhino[valueID1] = value1;
341       Rhino[valueID2] = value2;
342     }
343
344
345   }
346 #endif
347
348
349
350 #define MDL_DERIVATIVES
351 #if defined(MDL_DERIVATIVES)
352
353   static void mdlDerivatives(SimStruct *S)
354   {
355   }
356 #endif
357
358 static void mdlTerminate(SimStruct *S)
359 {
360 }
361
362
363 #ifdef  MATLAB_MEX_FILE
364 #include "simulink.c"
365 #else
```

```
366 #include "cg_sfun.h"
367 #endif
```

CAN.c

DODATAK C

```
1 #include "rhino.h"
2 #include "bci.h"
3
4
5 int BCI_CreateCANMsg (BCI_ts_CanMsg * CANMsg, UINT32 ID, UINT8 * Data,
6                       UINT8 DLC,
7                       UINT8 MFF);
8
9
10 int unpackMessage(BCI_ts_CanMsg msg, int *valueID1, int *valueID2, int *value1, int
       *value2)
11 {
12   int ret = BAD_MESSAGE;
13   int nodeID = msg.id % 16;
14   int VAR_ID1 = (msg.id / 256) * 2 - 1;
15   int VAR_ID2 = (msg.id / 256) * 2;
16   *value1 = ((msg.a_data[3] * 256 + msg.a_data[2]) * 256 + msg.a_data[1]) * 256 +
         msg.a_data[0];
17   *value2 = ((msg.a_data[7] * 256 + msg.a_data[6]) * 256 + msg.a_data[5]) * 256 +
         msg.a_data[4];
18   *valueID1 = (nodeID - 1)*4 + VAR_ID1 - 1;
19   *valueID2 = (nodeID - 1)*4 + VAR_ID2 - 1;
20   ret = UNPACK_OK;
21   return ret;
22
23 }
24
25
26 int packMessage(BCI_ts_CanMsg *msg, int valueID1, int valueID2, int value1, int
       value2)
27 {
28   UINT8 PacketData[8];
29   int nodeID = valueID1 / 2 + 1;
30   int VAR_ID = valueID1 % 2 + 2;
31   int msg_id = VAR_ID * 0x100 + nodeID;
32
33
34   PacketData[0] = (value1 & 0x000000FF) >>  0;
35       PacketData[1] = (value1 & 0x0000FF00) >>  8;
36       PacketData[2] = (value1 & 0x00FF0000) >> 16;
37       PacketData[3] = (value1 & 0xFF000000) >> 24;
38
39   PacketData[4] = (value2 & 0x000000FF) >>  0;
40       PacketData[5] = (value2 & 0x0000FF00) >>  8;
41       PacketData[6] = (value2 & 0x00FF0000) >> 16;
42       PacketData[7] = (value2 & 0xFF000000) >> 24;
43
44       BCI_CreateCANMsg(msg, msg_id, PacketData, 8, BCI_MFF_11_DAT);
45 }
```

rhino.c

DODATAK D

```
1 function ert_linux_make_rtw_hook(hookMethod,modelName,~,~,~,~)
2 switch hookMethod
3     case 'after_make'
4         linuxAfterMakeHook(modelName);
5 end
```

ert_linux_make_rtw_hook.m

```
1 function [ ] = linuxAfterMakeHook( modelName )
2
3 %  Sending TERM signal to previously started process. Model is terminated
4 %  by PID listed in /Simulink/started file. TERM signal is sent without
5 %  checking if the listed process has previously been terminated.
6
7 disp('Killing started models...');
8 system('ssh root@161.53.68.185 ''kill $(cat /Simulink/started)''');
9
10 %  Every previously loaded model is deleted (every file in /Simulink/models/ is
       removed).
11
12 disp('Removing previously loaded models...');
13 unix('ssh root@161.53.68.185 ''rm /Simulink/models/$(ls /Simulink/models/)''');
14
15 %  New model is downloaded via 'scp' to /Simulink/models/
16 %  Access mode is changed to allow model execution
17
18 disp('Downloading model to target...');
19 unix(['scp -q ' pwd '/../' modelName ' root@161.53.68.185:/Simulink/models']);
20 unix(['ssh root@161.53.68.185 ''chmod +x /Simulink/models/' modelName '''']);
21
22 %  Model is executed with '-w' option waiting to be started via Simulink command
       interface
23 %  Model is executed as backgorund process and all I/O streams are redirected to
       allow exiting
24 %  ssh session (shell hanging on logout otherwise)
25
26 %  Process PID is redirected to /Simulink/started file
27
28 disp('Model waiting for start...');
29 unix(['ssh root@161.53.68.185 ''nohup /Simulink/models/' modelName ' -w > foo.out
       2>foo.err </dev/null & echo $! > /Simulink/started''']);
30
31 end
```

linuxAfterMakeHook.m

LITERATURA

[1] R. Schwebel J. Singh, L. Fu. *Real-time Linux, često postavljana pitanja*. URL www.rt.wiki.kernel.org/index.php/Frequently_Asked_Questions.

[2] R N Jazar. *Theory of Applied Robotics: Kinematics, Dynamics, and Control (2nd Edition)*. Springer, 2010. ISBN 9781441917492. URL http://books.google.com/books?id=hHS40c6sYucC.

[3] Rhino Robotics ltd. *Rhino*, . URL www.rhinorobotics.com/.

[4] Rhino Robotics ltd. *Dimenzijski crtež Rhino XR-4*, . URL www.rhinorobotics.com/XR4%20Dims.dwf.

[5] Mathworks. *Matlab*. URL www.mathworks.com/products/matlab.

[6] *Simulink Real-Time*. Mathworks, 1 izdanju, 1 2014.

[7] Matworks. *Simulink*. URL www.mathworks.com/products/simulink/.

[8] OSADL. *Latest stable*. URL http://www.osadl.org/Latest-Stable-Realtime.latest-stable-realtime-linux.0.html.

[9] *Rhino XR3 Manual*. Rhino robotics ltd., 1 izdanju, 1 2002.

[10] Roboteq. *Roboteq SDC21xx datasheet*, . URL www.roboteq.com/index.php/docman/motor-controllers-documents-and-files/documentation/datasheets/sdc21xx-datasheet/63-sdc21xx-datasheet/file.

[11] Roboteq. *NextGen Controllers User Manual*, . URL www.roboteq.com/index.php/docman/motor-controllers-documents-and-files/documentation/user-manual/7-nextgen-controllers-user-manual/file.

[12] R J Schilling. *Fundamentals of robotics: analysis and control*. Prentice Hall, 1990. URL http://books.google.hr/books?id=LcxSAAAAMAAJ.

[13] Julijan Šribar, Boris Motik, i Bruno Motik. *Demistificirani C++*. Element, 2010.

[14] V. Krajči Z. Kovačić, S. Bogdan. *Osnove robotike*. Graphis, 2002.

[15] Lj. Kuljača Z. Vukić. *Automatsko upravljanje*. Kigen, 2005.

Razvoj i implementacija višerobotskog sustava za novi robotički laboratorij

Sažetak

U okviru rada osuvremenjen je postojeći zastarjeli robotski sustav temeljen na dva *Rhino* robotska manipulatora. Korištenjem servokontrolera tvrtke *Roboteq* ostvareno je upravljanje robotskim sustavom u stvarnom vremenu. Komunikacija između ciljnog računala i servokontrolera odvija se preko *CAN* sabirnice, uz *CANopen* protokol. Razvijena programska platforma, temeljena na *Real-time Linux* operacijskom sustavu, u potpunosti je integrirana u napredne programske pakete *Matlab* i *Simulink*. Na razvijenom eksperimentalnom sustavu implementirane su strukture upravljanja više i niže razine karakteristične za laboratorijske vježbe kolegija robotike na Fakultetu elektrotehnike i računarstva. Mogućnosti upravljačkih struktura demonstrirane su koordiniranom složenom operacijom mačevanja ostvarenom sinkroniziranim praćenjem složenih trajektorija.

Ključne riječi: Rhino, Roboteq, Linux, CAN, Matlab, Simulink, upravljanje u stvarnom vremenu

Development and implementation of a multi-robot system for the new robotic laboratory

Abstract

In this paper, the existing outdated robotic system, based on two *Rhino* robotic manipulators, has been modernized. Robotic system real-time control is achieved using *Roboteq* servo controllers. Communication between target computer and *Roboteq* servo controllers is implemented via *CAN* bus using *CANopen* protocol. Developed software environment, based on *Real-time Linux* operating system, has been integrated in *Matlab* and *Simulink* advanced software packages. High and low level control algorithms, typical for robotics laboratory exercises have been implemented on the developed experimental system. Control algorithm features are demonstrated through coordinated complex sword fight sequence.

Keywords: Rhino, Roboteq, Linux, Can, Matlab, Simulink, real-time control

POPIS SLIKA

POPIS TABLICA