Vizijska aplikacija za otkrivanje kolizije unutar radnog prostora robota

Sever, Lovro

FAKULTET STROJARSTVA I BRODOGRADNJE
SVEUČILIŠTE U ZAGREBU

ZAVRŠNI ZADATAK

Lovro Sever

Zagreb, 2023.

FAKULTET STROJARSTVA I BRODOGRADNJE
SVEUČILIŠTE U ZAGREBU

ZAVRŠNI ZADATAK

Mentor:
Izv. prof. dr. sc. Tomislav Stipančić

Student:
Lovro Sever

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE
Središnje povjerenstvo za završne i diplomske ispite
Povjerenstvo za završne i diplomske ispite studija strojarstva za smjerove:
Procesno-energetski, konstrukcijski, inženjersko modeliranje i računalne simulacije i brodostrojarski

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

ZAVRŠNI ZADATAK

Student: **Lovro Sever** JMBAG: **0035218064**

Naslov rada na hrvatskom jeziku: **Vizijska aplikacija za otkrivanje kolizije unutar radnog prostora robota**

Naslov rada na engleskom jeziku: **A vision application for collision detection within the robot workspace**

Opis zadatka:

U radu je potrebno osmisliti i implementirati sustav aktivne zaštite od kolizije robota i čovjeka koji dijele radni prostor. Sustav se treba temeljiti na strojnom vidu u sklopu kojeg se aktivno mjeri najmanja udaljenost između čovjeka i robota za vrijeme dijeljenja prostora. Vizijska aplikacija za otkrivanje kolizije treba sadržavati nekoliko međusobno povezanih programskih modula, uključujući:

- vizijsku aplikaciju koja se temelji na OpenCV biblioteci otvorenog koda te Python programskom jeziku te mjeri i izračunava najmanju trenutnu udaljenost između čovjeka i robota,
- modul koji upozorava osobu te zaustavlja robota ako izmjerena udaljenost između osobe i robota postane manja od dopuštene,
- analizu opterećenja robotskog sustava u slučaju aktivacije sustava kočenja,
- komunikacijski modul koji ostvaruje aktivnu vezu među komponentama robotskog sustava temeljem TCP komunikacijskog protokola te, ako dođe do prekida u komunikaciji, signalizira da se to dogodilo.

Osim vizijske aplikacije za otkrivanje kolizije potrebno je osmisliti i izraditi konstrukcijsko rješenje za prihvat kamere koja nadzire radni prostor čovjeka i robota.

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

Zadatak zadan: 30. 11. 2022.

Datum predaje rada:
1. rok: 20. 2. 2023.
2. rok (izvanredni): 10. 7. 2023.
3. rok: 18. 9. 2023.

Predviđeni datumi obrane:
1. rok: 27. 2. – 3. 3. 2023.
2. rok (izvanredni): 14. 7. 2023.
3. rok: 25. 9. – 29. 9. 2023.

Zadatak zadao:

Doc. dr. sc. Tomislav Stipančić

Predsjednik Povjerenstva:

Prof. dr. sc. Vladimir Soldo

Izjavljujem kako sam ovaj rad izradio samostalno koristeći znanja stečena tokom studija i koristeći navedenu literaturu.

Zahvaljujem se mentoru izv. prof. dr. sc. Tomislavu Stipančiću na zadavanju zadatka i kontinuiranom praćenju rada. Također se zahvaljujem asistentu Leonu Korenu, mag. ing. mech. na stručnim savjetima koji su mi pomogli prilikom izrade rada.

Posebno se zahvaljujem svojoj obitelji koja me je podržavala tijekom cijelog studija.

Sadržaj

Popis slika

Popis oznaka

M - Moment, Nm
Rob_pos, *rob_posn* - vektori položaja robota
Xaruco, yaruco – koordinate položaja aruco markera
ruke - lista koordinata položaja ruke
uvjet, uvjet1, uvjet2 – uvjeti za pokretanje različitih radnji izvršnog člana sustava.

Popis kratica

Kratica	**Opis**
OpenCV	*Open Source Computer Vision* – biblioteka namijenjena primjeni računalnog vida
TCP	*Transmission Control Protocol* – standardni protokol za izmjenjivanje podataka
RGB	*Red Green Blue* – Model prikaza boja
CMY	*Cyan Magenta Yellow* – model prikaza boja
CMYK	*Cyan Magenta Yellow Black* – model prikaza boja
HSL	*Hue Saturation Lightness* – model prikaza boja
FDM	*Fused deposition modeling* – aditivna metoda 3D ispisivanja
PLA	*Polylactic acid* – polilaktična kiselina
CAD	*Computer Aided Design* – računalom potpomognuto oblikovanje

Popis tehničke dokumentacije

LS_2302_01

SAŽETAK

Sve širom upotrebom robota u industriji sve češće se čovjek i robot mogu nalaziti u zajedničkom radnom prostoru. Tema završnog zadatka je izrada vizijskog sustava za sprječavanje kolizije robota i čovjeka. Vizijska aplikacija izrađena je u programskom jeziku Python uz korištenje biblioteka OpencCV i MediaPipe. Biblioteka MediaPipe omogućuje prepoznavanje i lociranje čovjeka ili određenih dijelova čovjeka (npr. šaka, lice, oči, ruka). Kao senzor korišten za prepoznavanje i lociranje čovjeka odabrana je web kamera. Lociranje robota provedeno je kontinuiranom komunikacijom robota i vizijske aplikacije putem TCP protokola. Za potrebe testiranja vizijske aplikacije korišteni je kolaborativni robot UR5. Uz vizijsku aplikaciju dizajnirani je i izrađeni nosač kamere te je provedena analiza opterećenja robota u normalnom radu i prilikom kočenja. U uvodnom djelu rada ukratko se opisuje tema zadatka dok se kroz ostala poglavlja detaljno opisuje postupak izrade vizijske aplikacije, dinamičke analize i izrade nosača kamere.

Ključne riječi: vizijski sustav, računalni vid, prepoznavanje čovjeka, OpenCV, MediaPipe

SUMMARY

With increasingly common use of robot sin industry, more and more often, man and robot can be located in a common workspace. The topic of the final task is the development of a vision system to prevent the collision between robots and humans. The vision application was created in the Python programming language using the OpencCV and MediaPipe libraries. The MediaPipe library allows you to identify and locate a person or certain parts of a person (e.g., hand, face, eyes, hands). A webcam was chosen as a sensor used to identify and locate a human. Robot locating was carried out through continuous communication of the robot and vision application via the TCP protocol. For the purpose of testing the vision application, the collaborative robot UR5 was used. In addition to the vision application, a camera mount was designed and built and a robot load analysis was carried out in normal operation and during braking. The introductory part of the paper briefly describes the topic of the task, while through other chapters the process of creating a vision application, dynamic analysis and making a camera mount is described in detail.
Keywords: vision system, computer vision, human recognition, OpenCV, MediaPipe

1. Uvod

Razvojem kako proizvodnih tehnologija tako i robota potrebna je sve veća interakcija između robota i čovjeka. Prilikom interakcije, zbog neopreznosti čovjeka ili greške u programu robota, može doći do zalijetanja robota u čovjeka, te prilikom toga do ozljeđivanja čovjeka i/ili oštećivanja robota. Kako bi se spriječila kolizija robota i čovjeka potrebno je ugraditi neki oblik zaštite koji zaustavlja rad robota ukoliko mu se čovjek previše približi.

U današnje vrijeme postoje mnogi oblici senzora pomoću kojih je moguće odrediti udaljenost čovjeka i robota, ali je kao tema rada odabrani isključivo vizijski sustav. Razvojem tehnologija razvijene su i kamere koje mogu snimati u sve većim rezolucijama te računala koja mogu obraditi veliku količinu ulaznih podataka u vrlo kratkom vremenu što ujedno omogućuje njihovu sve širu primjenu i razvoj računalnog vida. Korištenjem vizijskog sustava, umjesto neke vrste daljinomjera, postiže se mnogo veća prilagodljivost radnom okruženju.

Jedan od problema izrade vizijskih sustava je prepoznavanje objekata. Iako je čovjeku jako jednostavno prepoznati čovjeka, auto ili neku životinju, za računalo je prepoznavanje objekata veoma složeni zadatak. U svrhu prepoznavanja objekata koriste se sustavi neuronskih mreža i duboko učenje kako bi se naučilo računalo kako da prepoznaje određene objekte.

Vizijska aplikacija izrađena je u programskom jeziku Python uz korištenje biblioteke OpenCV i MediaPipe. Sve korištene biblioteke će biti dodatno opisane u 2. poglavlju. Aplikacija izmjenjuje podatke sa robotom pomoću TCP protokola te sa razvojnom pločicom Arduino UNO pomoću USB porta.

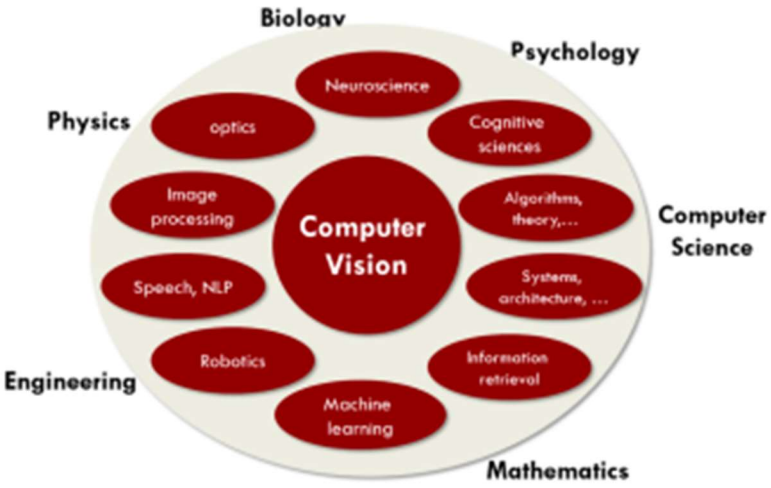
Uz vizijsku aplikaciju konstruirani je i izrađeni nosač kamere te je provedena dinamička analiza opterećenja koji djeluju na robot. Vizijska aplikacija testirana je na robotskoj ruci UR5 te je sukladno tomu provedena dinamička analiza za navedenu robotsku ruku.

2. Vizijska aplikacija

Kao što je u uvodnom djelu navedeno, vizijska aplikacija izrađena je u programskom jeziku Python uz korištenje biblioteka OpenCV i MediaPipe za obradu slika i prepoznavanje čovjeka, biblioteka TensorFlow za duboko učenje i biblioteke urx i pyserial za komunikaciju sa robotskom rukom i razvojnom pločicom Arduino UNO. Aplikacija se u konačnici svodi na to da se uspoređuje trenutna udaljenost između robota i ruke sa minimalnom dopuštenom udaljenosti. Budući kako jedini ulazni podatak koji aplikacija dobiva je slika okruženja robota, koja se sastoji od piksela, potrebno je odrediti minimalni dopušteni broj piksela koji se nalazi između čovjeka i robota. Tokom ovoga poglavlja biti će objašnjena problematika rješavanja sustava, moguća rješenja te detaljno opisano odabrano rješenje problema.

2.1. Računalni vid

Postoji nekoliko definicija što je to računalni vid. Jedna od definicija koje se najčešće navode je sljedeća: Računalni vid [1] je razvojni algoritam koji je sposoban prepoznati kontekst iz slike. Svakako je bitno napomenuti kako je računalni vid složena disciplina koja se sastoji od brojnih znanstvenih područja.



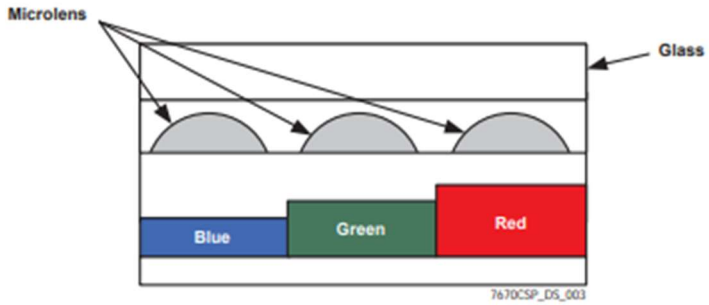
Slika 1: Područja računalnog vida [1]

Računalni vid je složena disciplina te se u sklopu rada neće obraditi cijelo područje računalnog vida. Neke od bitnijih stavka, korištenih u rješavanju zadatka, računalnog vida su svakako uređivanje slika (ucrtavanje oblika i dodavanje teksta na sliku), prepoznavanje objekata te manipulacija bojama na slici.

2.1.1. Boje [2]

Prije objašnjavanja koncepta boja u sklopu računalnog vida potrebno je objasniti što je to boja i kako čovjek raspoznaje boje. Svjetlost koja dolazi iz nekog izvora ili reflektirana svjetlost od površine predmeta sadrži više ili manje energije. Uz varijacije u energetskom spektru, svjetlost se u konačnici sastoji od spektra valnih duljina. Ljudsko oko može prepoznati samo uski spektar valnih duljina svjetlosti i to u rasponu od 400 do 700nm. Područje valnih duljina

svjetlosti koje ljudsko oko može raspoznati naziva se još i vidljiva svjetlost. Receptori u ljudskom oku različito reagiraju na svjetlost različitih valnih duljina a dobivena informacija se pomoću živčanog sustava šalje u ljudski vizijski sustav (mozak). Skup informacija koji mozak obrađuje sastoji se od vrijednost intenziteta za svaku pojedinu valnu duljinu svjetlosti koju receptori u ljudskom oku mogu prepoznati. Objedinjujući te informacije čovjek stvara percepciju o bojama.
S obzirom na nagli raznoj elektronike i općenito tehnike razvijene su i kamere koje mogu prepoznati široki spektar valnih duljina svijetlosti. Uz kamere koje prepoznaju valne duljine svjetlosti u vidljivom spektru, postoje specijalizirane kamere za snimanje npr. infracrvenog spektra valnih duljina koji nije vidljiv ljudskom oku. U pravilu se kamera sastoji od niza receptora gdje svaki receptor može zasebno raspoznati određeni spektar svijetlosti. Kamere koje se koriste u općoj namjeni raspoznaju crvenu, plavu i zelenu svjetlost. Svaki receptor na kameri predstavlja jedan piksel te on sadrži informaciju o intenzitetu svake od tri navedene boje.

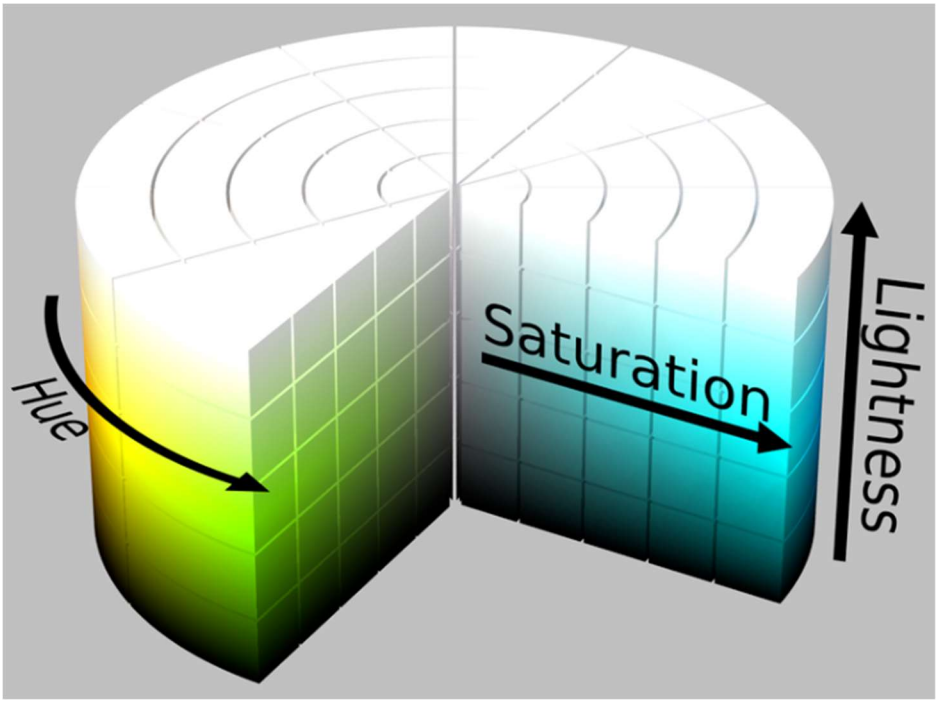


Slika 2: Prepoznavanje boja[3]

Kamere uglavnom koriste RGB model zapisa boja. RGB model zapisa boje sastoji se od liste intenziteta crvene, zelene i plave boje. Ovaj model prikaza boja počiva na modelu dodavanja osnovnih boja različitih intenziteta kako bi se postigla željena boja.

Slika 3: RGB model [4]

Današnji led monitori, kamere i mnogi drugi optički uređaji koriste ovaj model boja. Kao što je sa slike 3 vidljivo, bijela boja postiže se maksimalnom zasićenošću sve tri osnovne boje dok se crna postiže nultim intenzitetom svih osnovnih boja.
Drugi model zapisa boja koji se koristi je HSL. Ovaj model počiva na RGB modelu zapisa boja s razlikom u načinu zapisa. Zapis boja prikazani je u cilindričnom koordinatnom sustavu vrijednost kuta određuje boju dok dimenzija *z* predstavlja svjetloću.



Slika 4: HSL model boja[5]

Uz ova dva modela zapisa boja postoji i CMYK model zapisa boja ali on se u pravilu koristi za prikaz boje pomoću tintnih pisača te se ovdje neće dodatno objašnjavati.

2.1.2. Uređivanje slika

Jedno od područja računalnog vida je uređivanje slika. U područje uređivanja slika spada filtriranje šuma slike, promjena dimenzija slika, promjena načina prikaza boja, upisivanje teksta na sliku i slično. Kako bi računalo moglo prepoznati neki objekt, često je potrebno ukloniti šum slike. Jedan od primjera gdje je potrebna filtracija slike je detekcija bridova objekta prikazanog na slici.

Slika 5: Filtracija šuma[6]

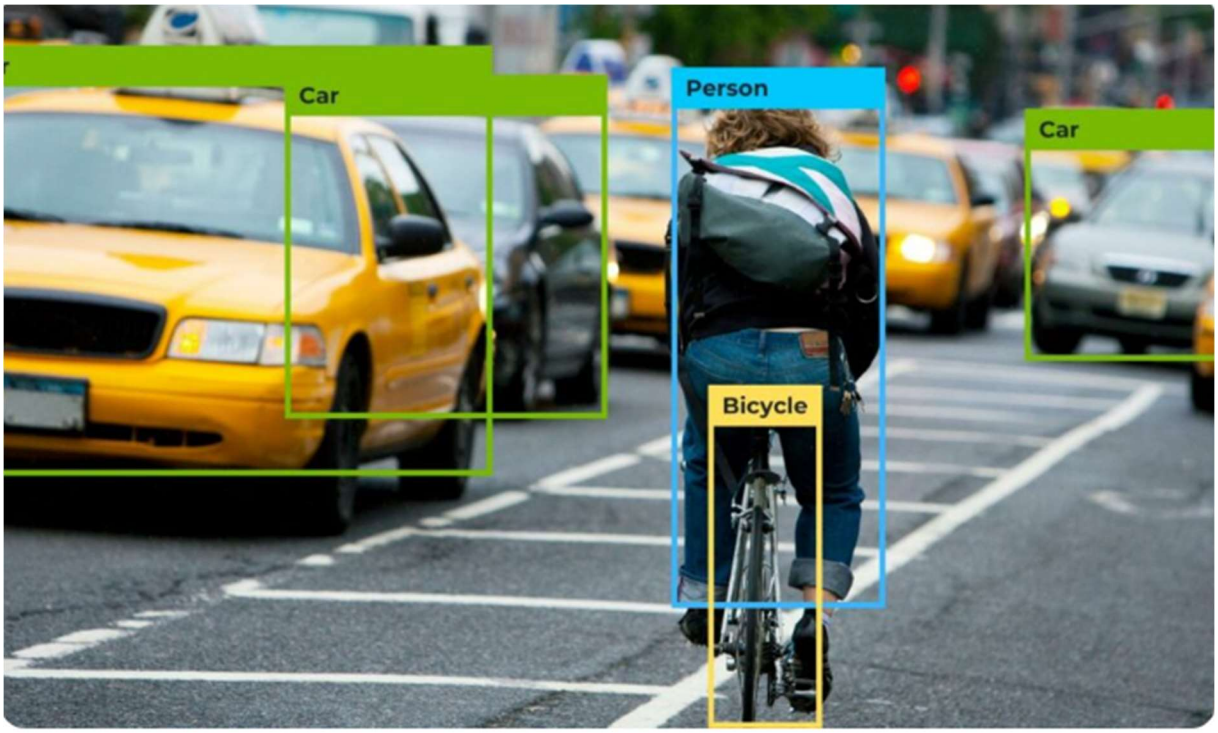
Iako se na slici 5 ne vidi nikakva razlika između lijeve i desne slike, na desnoj slici je provedeni jedan način filtracije slike. Nakon provođenja filtracije uklonjeni je veliki udio šuma na slici te je slika spremna za prepoznavanje bridova prikazanog objekta.
Područje računalnog vida pokriva i sve ostale manipulacije slikom kao što je promjena spektra boja, promjena dimenzija i slično. Primjer potrebe za obrezivanjem slike i smanjivanjem

dimenzija je priprema slika za proces strojnog učenja. Slika nakon manipulacije mora sadržavati samo objekt koji model treba prepoznavati. Osim sadržaja slike, slika mora biti što manja tj. najmanjih mogućih dimenzija a da objekt i dalje bude prepoznatljiv.

2.1.3. Prepoznavanje objekata

U današnje vrijeme je prepoznavanje objekata veoma razvijeno. Razvojem računalnog vida naglo se širi i njegova primjena. Jedan od sustava koji se trenutno razvija je sustav autonomne vožnje. Ovaj sustav mora precizno locirati i prepoznati svaki objekt koji se nalazi na kolniku i u njenoj okolini. Iako razvoj računalnog vida potiče razvoj mnogih vezanih područja, još uvijek postoji mnogo nedostataka ovih sustava. Trenutno postoji mnogo algoritama za prepoznavanje objekata od koji su slijedeći najpopularniji:

- YOLO
- SSD
- HOG
- R-CNN.



Slika 6: YOLO [7]

2.2. Korištene biblioteke

U ovome podpoglavlju biti će ukratko opisane osnovne biblioteke korištene pri rješavanju problema.

2.2.1. OpenCV [9]

OpenCV je biblioteka korištena za obradu i prikaz slika. Slika obrađivana pomoću ove biblioteke može biti statička (ranije fotografirana) i dinamička (video snimljeni ranije ili video dobiveni sa kamere povezane na uređaj). Tokom ovoga rada obrađivati će se video u realnom vremenu dobiveni sa kamere.

Slika 7: OpenCV logo [8]

Biblioteka dostupna je za programske jezike Java, C++, MATLAB i Python. Biblioteka sadrži više od 2500 optimiziranih algoritama koji su namijenjeni strojnom učenju i standardnom „state-of-the-art“ računanom vidu. Među pripadajućim algoritmima nalaze se i oni za detekciju i prepoznavanje lica, prepoznavanje objekata i slično. U današnje vrijeme postoji razvijena podrška za korištenje ove biblioteke koja se sastoji od zajednice korisnika. Ta zajednica broji preko 47000 prijavljenih korisnika. Funkcionalnosti biblioteke moguće je proširiti dodatnim specijaliziranim bibliotekama kao što su TensorFlow, PyTorch, MediaPipe i druge.

2.2.2. MediaPipe

MediaPipe[11] je platforma korištena kao proširenje biblioteke openCV. MediaPipe nudi brojna rješenja u području strojnog učenja te može raditi na stolnim računalima, server računalima, Android pametnim telefonima, iOS-u te na mnogim drugim uređajima kao što je Raspberry Pi. Neka od mnogih rješenja koja nudi MediaPipe su prepoznavanje lica, očiju, ruku, prepoznavanje određene poze itd. U ovome radu primjenjivati će se prepoznavanje šake i prepoznavanje određene poze šake.

Slika 8: MediaPipe logo[10]

2.2.3. TensorFlow

TensorFlow [12] je besplatna biblioteka namijenjena za strojno učenje i umjetnu inteligenciju. Dostupna je za programske jezike kao što su Java, Python, C++ i JavaScript. Pomoću ove biblioteke moguće je koristiti unaprijed trenirane modele za prepoznavanje objekata određenih osobina te je moguće i trenirati vlastite modele.

Slika 9: TensorFlow [12]

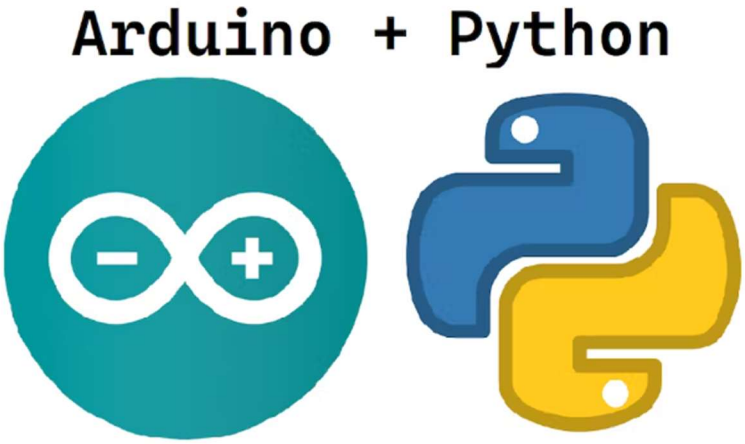
2.2.4. URx

URx je biblioteka izrađena za Python programski jezik te se koristi za komunikaciju robotske ruke serije UR proizvođača Universal Robots. Ova biblioteka je relativno ograničena što se tiče upravljanja robotskom rukom ali je dovoljna za osnovno komuniciranje sa robotskom rukom što podrazumijeva upravljanje kretanja robotske ruke i preuzimanja podataka o lokaciji kako alata priključenog na robotsku ruku tako i lokacije svih zglobova. Sama biblioteka koristi TCP protokol za komunikaciju.

Slika 10: UR5 [13]

2.2.5. PySerial

PySerial je biblioteka namijenjena za Python programski jezik te se koristi za serijsku komunikaciju sa uređajima žičano povezanima sa računalom. U sklopu ovoga rada korištena je za komunikaciju vizijske aplikacije sa razvojnom pločicom Arduino UNO povezanu preko USB porta. Biblioteka sadrži mnoge unaprijed definirane algoritme koji omogućuju stabilnu komunikaciju između uređaja te je prikladna za komunikaciju i sa uređajima koji imaju sigurnosnu ulogu kao što je pokretanje sigurnosne kočnice.



Slika 11: Pyserial [14]

2.3. Razvoj aplikacije

Vizijska aplikacija za sprječavanje kolizije robota i čovjeka je veoma kompleksna te je razvoj sadržavao mnoge ideje o načinu lociranja robota i čovjeka te o mogućnostima komunikacije sa robotskom rukom i pokretanja sigurnosne kočnice. Najvažnija stavka prilikom odabira najbolje opcije je svakako pouzdanost i dostupnost odabranog rješenja.

Prva stavka je odabir načina prepoznavanja čovjeka. U sklopu rada pretpostavljeno je kako u većini slučajeve čovjek prvo prilazi k robotu sa rukom. Bazirajući se na tu pretpostavku odabrano je kako sustav mora prepoznati čovjekovu šaku i odrediti točnu lokaciju iste. Od dostupnih ideja (postavljanja markera na ruku, izrade vlastitog modela za prepoznavanje ruke itd) odabrano je rješenje pomoću biblioteke MediaPipe. MediaPipe nudi gotove algoritme za precizno prepoznavanje ljudske šake i određivanje njene lokacije.
Idući problem koji je potrebno riješiti je određivanje lokacije robotske ruke. Prva dva moguća rješenja se temelje na postavljanju markera na robotsku ruku. Oba rješenja imaju veliku osjetljivost na osvjetljenje i ne postoji mogućnost postići vidljivost markera u svim položajima robotske ruke. Iduće moguće je pomoću dubokog učenja trenirati model koji bi prepoznavao i locirao određeni dio robotske ruke. Iako bi ovo rješenje moglo zadovoljiti sve tražene zahtjeve, odabrano je rješenje s komunikacijom s robotskom rukom pomoću TCP protokola. Odabrano rješenje nudi stabilnost i precizno određivanje lokacije robotske ruke.
Jedan od najbitnijih dijelova sustava je način pokretanja sigurnosne kočnice. Prilikom odabira načina pokretanja kočnice bitno je postići žičanu povezanost računala sa sučeljem za pokretanje kočnice kako bi bila postignuta što veća stabilnost prilikom rada. Sve ideje se temelje na korištenju serijske komunikacije aplikacije sa PLC-om ili nekim mikro računalom. Iako bi odabir PLC-a u industriji bio jedan od najboljih odabira, u ovom slučaju, zbog dostupnosti, odabrana je razvojna pločica Arduino UNO. Arduino UNO je razvojna pločica namijenjena razvoju prototipova. Zbog njene lake dostupnosti i jednostavnog načina programiranja često je korištena za mnoge projekte.

2.4. Prepoznavanje ljudske šake

U čovjekovoj prirodi je da jednostavno prepoznaje ljudsku šaku i precizno odredi njenu poziciju u prostoru. Za računalo je to vrlo kompleksan zadatak te je potrebno izraditi neku vrstu modela za prepoznavanje ljudske šake. Važno je da model bude u potpunosti neovisan o dimenziji šake i boji kože. Postoje razne mogućnosti rješavanja toga problema među kojima je i izrada vlastitog modela prepoznavanja i lociranja ljudske šake. S obzirom da su u današnje vrijeme dostupni gotovi algoritmi prepoznavanja, kako ljudske šake, tako i cijeloga ljudskog tijela može se odabrati mnogo pouzdanije rješenje od izrade vlastitog modela. Jedno od veoma popularnih rješenja je korištenje biblioteke MediaPipe. Biblioteka MediaPipe sadrži algoritam za prepoznavanje i segmentaciju ljudske šake.
U python biblioteci MediaPipe, inicijalizacija algoritma za prepoznavanje i segmentaciju provodi se na sljedeći način:

```
158     # inicijalizacija Mediapipe-a
159     mp_hands = mp.solutions.hands
160     mp_drawing = mp.solutions.drawing_utils
161     mp_drawing_styles = mp.solutions.drawing_styles

185     with mp_hands.Hands(
186             model_complexity=0,
187             max_num_hands=2,
188             min_detection_confidence=0.7,
189             min_tracking_confidence=0.5) as hands:
```

Slika 12: Inicijalizacija MediaPipe

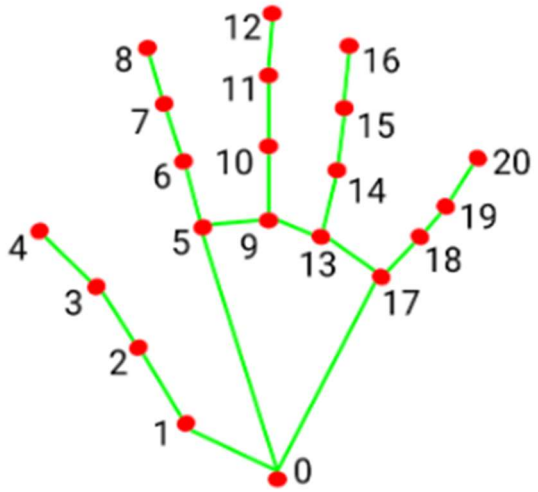
Slika 12 prikazuje način inicijalizacije algoritma za prepoznavanje šake. Nakon inicijalizacije algoritma potrebno je pokrenuti algoritam sa unaprijed definiranim inicijalnim vrijednostima.

```
200                 results = hands.process(image)
```

Slika 13: MediaPipe pokretanje

Slika 13 prikazuje osnovni dio Python aplikacije koja pokreće algoritam za prepoznavanje šake. Kao ulazne informacije za inicijalizaciju algoritma zadaju se željena podudarnost prepoznavanja objekta i maksimalni broj šaka koje algoritam može prepoznati. Povećavajući traženu podudarnost smanjuje se nepotrebno prepoznavanje drugih objekata kao ljudsku šaku ali se ujedno povećava i vjerojatnost da algoritam ne prepozna ljudsku šaku. Kao optimalna vrijednost odabrana je podudarnost iznosa 0,7 koja se pokazala kao najbolji omjer bespotrebnog prepoznavanja „šuma" kao ljudske šake i prepoznavanja prave ljudske šake. Izlazne informacije algoritma pohranjuju se željenu varijablu te ona sadrži slijedeće informacije:

- MULTI_HANDEDNESS – sadrži informaciju dali je prepoznata lijeva ili desna ruka,
- MULTI_HAND_LANDMARKS – sadrži informacije o lokaciji svakog pojedinog zgloba na ruci
- MULTI_HAND_WORLD_LANDMARKS – sadrži koordinate svakog pojedinog zgloba smještenog u realni 3D prostor. Koordinate su izražene u metrima.



Slika 14: Segmentacija šake [15]

Slika prikazuje sve segmente i zglobove čije koordinate sadrži izlazna varijabla algoritma za prepoznavanje. Pretpostavljajući kako će čovjek najbliže alatu priključenom na robotsku ruku prići sa prstima šake, odabrano je da aplikacija provjerava udaljenost alata i vrha kažiprsta. Referirajući se na sliku zaključuje se kako je vrh kažiprsta numeriran za brojem 8 te je u aplikaciji potrebno odrediti koordinatu zgloba broj 8. Dobivene koordinate pohranjene su u obliku liste koja sadrži *x* i *y* koordinatu vrha kažiprsta. Budući da aplikacija u isto vrijeme može prepoznati više od jedne šake (u ovome slučaju zadano je prepoznavanje maksimalno 2 šake) svaka lista sa koordinatama pohranjena u 2D listu koja sadrži koordinate vrha kažiprsta svake ruke.

```
224                 ruke.append([(1-hand_landmarks.landmark[8].x) * debug_image.shape[1],
225                              (hand_landmarks.landmark[8].y) * debug_image.shape[0]])
```

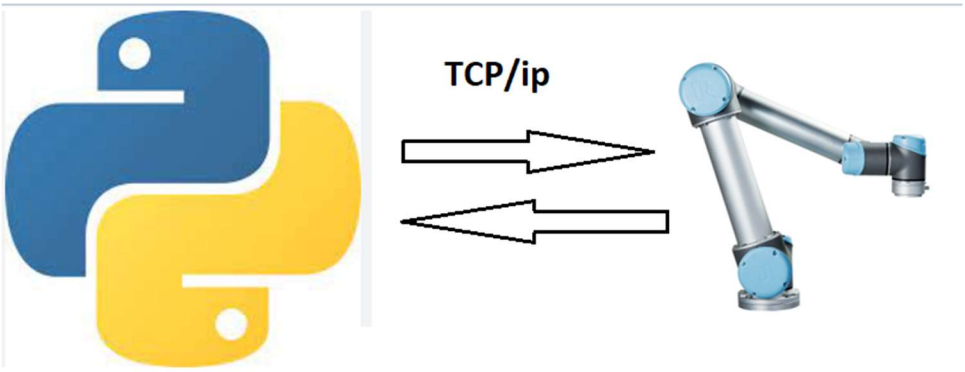
Slika 15: Pohrana koordinata šake

Na slici 15 je prikazani dio izrađene aplikacije koji pohranjuje potrebne informacije u listu nazvanu *ruke*. Pohranjene informacije sadrže *x* i *y* koordinatu vrha kažiprsta za svaku prepoznati šaku izraženu u broju piksela za koji je vrh kažiprsta udaljeni od gornjeg desnog ruba slike preuzete sa kamere. Jedan od najvećih problema je taj što računalo nije u mogućnosti

prepoznavati koordinatu dubine te samim time ne postoji informacija o udaljenosti objekata od kamere. Budući kako sa postojećom opremom ne postoji mogućnost preciznog određivanja udaljenosti objekta od kamere, u sklopu ove aplikacije ta koordinata nije uvrštena u algoritam za određivanje udaljenosti ljudske šake od alata na robotskoj ruci. Postoji nekoliko mogućnosti za rješavanje ovoga problema. Jedno rješenje je postavljanje dodatne kamere koja, kao i prva kamera u aplikaciju šalje sliku koja predstavlja 2D prostor, ali ta kamera mora biti zaokrenuta za točno 90º s obzirom na prvu kameru. Na taj način je moguće definirati 3D koordinate koristeći dvije 2D slike.
Drugo, bolje, rješenje je korištenje binokularne stereo kamere koja snima prostor sa dvije kamere smještene jednu pokraj druge. Pomoću osnovnih trigonometrijskih jednadžbi je iz te dvije slike moguće dobiti informaciju o udaljenosti objekta od kamere.

2.5. Komunikacija sa robotskom rukom UR5

Prilikom izrade vizijskog sustava nailazilo se na mnoge prepreke u rješavanju zadatka. Jedna od velikih prepreka je i određivanje lokacije robota. Kao rješenje problema postojalo je nekoliko mogućih izvedbi od kojih je odabrano rješenje uspostavom komunikacije s robotskom rukom.



Slika 16: TCP komunikacija

Komunikacija sa robotskom rukom UR5 provodi se uz pomoć TCP protokola. Za Python programski jeziku postoji biblioteka naziva URx predviđena upravo za uspostavu komunikacije. Iako biblioteka ima relativno ograničene mogućnosti upravljanja robotskom rukom, u području preuzimanja podataka o poziciji i brzini pojedinih čvorova ima dovoljno mogućnosti za pravilnu provedbu rada.
Prije samog početka preuzimanja podataka sa robota potrebno je odrediti osnovne parametre za uspostavu TCP protokola. Upotrebom navedene biblioteke ovaj dio problema se provodi jednostavno uz zadavanje IP adrese robotske ruke i to na način prikazani na slici.

```
169     robot = urx.Robot("192.168.0.25")
```

Slika 17: UR5 ip adresa

Nakon što je uspostavljena komunikacija može se pristupiti podacima o lokaciji robotske ruke. U zadanom slučaju, jedini potrebni podatak je onaj o poziciji vrha alata priključenog na robotsku ruku. Podatak o poziciji vrha alata moguće je preuzeti koristeći naredbu *getl()* koja vraća vektor položaja koji poprima sljedeći oblik:

$$rob_{pos} = [x, y, z, r_x, r_y, r_z], \quad (2.1)$$

gdje x, y, i z predstavljaju vektor pozicije vrha alata dok r_x, r_y i r_z predstavljaju vektor rotacije vrha alata. Poznajući ove podatke moguće je jednoznačno definirati lokaciju vrha alata robotske ruke.

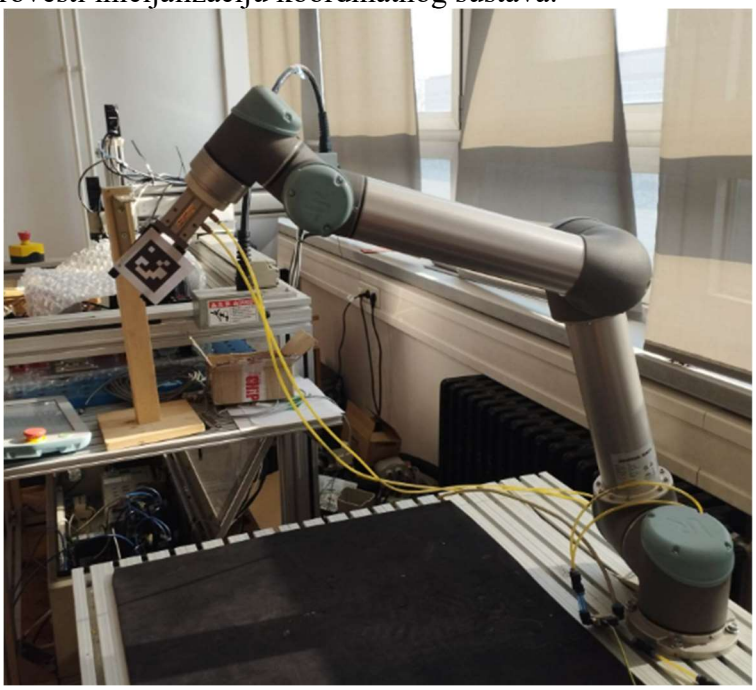
```
178        rob_pos = robot.getl()
```

Slika 18: Preuzimanje lokacije

Slika 18 prikazuje primijenjeni način prizivanja funkcije za preuzimanje koordinata vrha alata. Nakon preuzimanja podataka sa robota potrebno je prikazati dobivene informacije o lokaciji na slici preuzetoj sa kamere. Jedan od najvećih problema prikaza taj što računalo posjeduje jedino podatak o koordinatama slike preuzete sa kamere dok je ishodište koordinatnog sustava u potpunosti nepoznato za računalo. Drugi problem prikaza je određivanje koliko piksela na slici označava pomak robotske za 1mm. Kao što se može i pretpostaviti preslikavanje milimetara u piksele ovisi i o udaljenosti robotske ruke od kamere.
Jedno od najstabilnijih rješenja određivanja koordinatnog sustava je pomoću ArUco markera.

Slika 19: Aruco

Očitavanjem ArUco markera može se odrediti koordinate rubova markera, koordinatu centra markera te njegova orijentacija. Postavljenjem markera na vrh alata priključenog na robotsku ruku moguće je provesti inicijalizaciju koordinatnog sustava.



Slika 20: Postavljanje markera

Slika 20 prikazuje način postavljanja markera na robotsku ruku. Kao što je vidljivo, koordinatni sustav koji povezuje koordinate robotske ruke sa koordinatama na slici postavljeni je na mjestu početne lokacije vrha alata robotske ruke. Nakon očitavanja lokacije robota i lokacije ArUco markera može se ukloniti marker te on nije potreban sve do idućeg pokretanja aplikacije.
S obzirom na utjecaj udaljenosti robotske ruke od kamere potrebno je poznavati početnu udaljenost robotske ruke od kamere. Na idućoj slici prikazano je programsko rješenje usklađivanja podataka dobivenih sa kamere i s robotske ruke.

```
104  def prikazRobotPoz(debug_image):
105      robx = rob_pos[0] * 1000
106      roby = rob_pos[1] * 1000
107      robz = rob_pos[2] * 1000
108
109      robnx = rob_pos_n[0] * 1000
110      robny = rob_pos_n[1] * 1000
111      robnz = rob_pos_n[2] * 1000
112
113      drobx = robnx - robx
114      drobz = robnz - robz
115
116      dy = 1500 + roby
117      hxr = drobx * (1550 / dy)
118      hzr = drobz * (1550 / dy)
119      xr = xaruco - hxr
120      yr = yaruco - hzr
```

Slika 21: sinkronizacija podataka

Varijabla *rob_pos* predstavlja početnu poziciju robota dok varijabla *rob_pos_n* predstavlja trenutnu lokaciju robota. Varijable *xaruco* i *yaruco* su koordinate ArUco markera određene prilikom pokretanja vizijskog sustava. Kao što je vidljivo na slikama jedna od stavki je određivanje odnosa preslikavanja milimetara u piksele (vidljivo u linijama numeriranima sa 117 i 118). Taj problem je moguće riješiti na nekoliko načina dok je jedan od načina izrada fotografije ArUco markera koji je poznatih dimenzija udaljen od kamere za poznati iznos. Pretpostavljajući linearnu ovisnost faktora preslikavanja o udaljenosti predmeta od kamere može se jednostavno odrediti iznos faktora preslikavanja.

2.6. Određivanje udaljenosti robotske ruke i čovjekove šake

Sama srž sustava za sprječavanje kolizije robota i ruke je određivanje udaljenosti robota i ruke te po potrebi pokretanje protokola za zaustavljanje robotske ruke.

Slika 22: udaljenost primjer

Poznajući prethodno određene koordinate ljudske šake i robotske ruke jednostavno je odrediti međusobnu udaljenost. Rješenje algoritma je prikazano na idućoj slici.

```
127  def racunanjeUdaljenosti(ruke, xr, yr):
128      if len(ruke) == 1:
129          xruka, yruka = ruke[0]
130          print(xr, yr, xruka, yruka)
131          udaljenost_min = math.sqrt((xr-xruka)**2 + (yr-yruka)**2)
132      elif len(ruke) == 2:
133          xruka1, yruka1 = ruke[0]
134          xruka2, yruka2 = ruke[1]
135          udaljenost_1 = math.sqrt((xr - xruka1) ** 2 + (yr - yruka1) ** 2)
136          udaljenost_2 = math.sqrt((xr - xruka2) ** 2 + (yr - yruka2) ** 2)
137          if udaljenost_1 < udaljenost_2:
138              udaljenost_min = udaljenost_1
139          else:
140              udaljenost_min = udaljenost_2
141      return udaljenost_min
```

Slika 23: Algoritam računanja udaljenosti

Slika 23 prikazuje kako je udaljenost izračunata pomoću pitagorinog poučka. U slučaju da algoritam za prepoznavanje šake u isto vrijeme prepozna dvije šake potrebno je odrediti iznos udaljenosti robotske ruke i svake šake posebno. U algoritam za pokretanje sigurnosne kočnice kao ulazna informacija ulazi samo iznos najmanje izračunate udaljenosti koja je u algoritmu nazvana *udaljenost_min.* Udaljenost je izražena u broju piksela te je uvjet za pokretanje

algoritma za zaustavljanje robotske šake također izražen u pikselima. Uvjeti za pokretanje algoritma za zaustavljanje, algoritma za upozoravanje čovjeka kako se nalazi blizu robotu i algoritma za ponovno pokretanje prikazani su na sljedećoj slici.

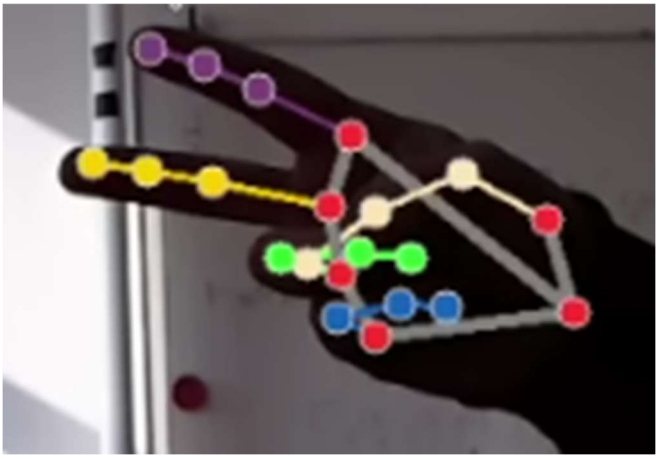
```
239             uvjet = (300 * 915) / dy
240             uvjet2 = (400 * 915) / dy
241             uvjet3 = (500 * 915) / dy
```

Slika 24: Uvjeti

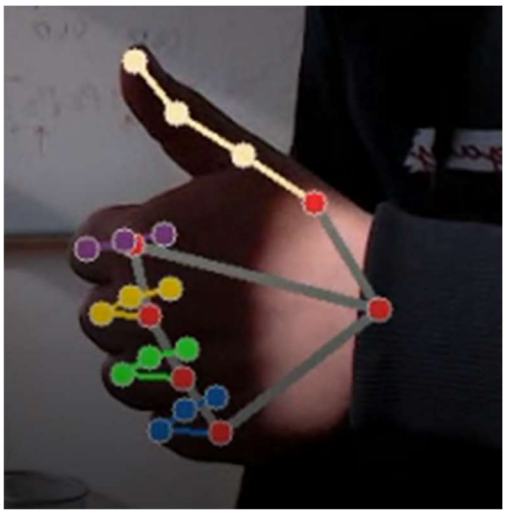
U slučaju kada je ispunjeni prvi uvjet (*udaljenost_min* < *uvjet*) aplikacija šalje poruku sa sadržajem „stani“, kada je ispunjeni drugi uvjet (*udaljenost_min* > *uvjet2*) aplikacija šalje poruku „kreni“. Taj uvjet označava kako se je čovjek udaljio od robotske ruke na sigurnu udaljenost. Posljednji uvjet koji se provjerava (*udaljenost_min* < *uvjet3*) šalje poruku „pazi“ te ona označava kako se je čovjek približio robotskoj ruci te ga zvučnim i svjetlosnim signalom upozorava na opasnost od kolizije sa robotskom rukom.

2.7. Sustav prepoznavanja poze ljudske ruke

Prilikom čovjekovog potpomaganja robotu u provođenju zadataka ponekada postoji potreba za privremeno zaustavljanje robota kako bi se sigurno moglo pristupiti radnom okruženju robota. U tu svrhu je trenirani model prepoznavanja poze ljudske šake. Pomoću biblioteke TensorFlow trenirani je model na temelju koordinata zglobova šake dobivenih kao izlazna informacija biblioteke MediaPipe. Prilikom treniranja modela pohranjeno je oko 1000 uzoraka svake poze ruke i 1000 uzoraka ostalih neodređenih poza. Iako je uzeti relativno veliki uzorak za treniranje modela i dalje postoji mogućnost prepoznavanja pogrešnog znaka. Povećanjem veličine uzorka za treniranje modela postiže se i veća preciznost prepoznavanja uzorka. Na iduće dvije slike prikazane su poze šake koje algoritam mora prepoznati s tim da je na prvoj slici prikazani znak za zaustavljanje robota dok druga slika prikazuje pozu za pokretanje robota.



Slika 25: Znak stani



Slika 26: Znak kreni

Slika 25 i slika 26 prikazuju dvije osnovne poze šake dok je za pravilno treniranje modela potrebno izraditi što veći uzorak u raznim položajima šake naspram kamere. Još bolja preciznost bi bila postignuta kada bi se izradio uzorak na temelju nekoliko različitih osoba

```
153     args = get_args()
154     use_static_image_mode = args.use_static_image_mode
155     min_detection_confidence = args.min_detection_confidence
156     min_tracking_confidence = args.min_tracking_confidence
```

Slika 27: inicijalizacija prepoznavanja poze

Slika 27 prikazuje način inicijalizacije algoritma za prepoznavanje poze. Kao i prilikom inicijalizacije i pozivanja algoritma za prepoznavanje šake, potrebno je zadati traženu podudarnost trenutnih koordinata sa uzorkom.

```
209                 hand_sign_id = keypoint_classifier(pre_processed_landmark_list)
```

Slika 28: Pozivanje algoritma

Slika 28 prikazuje dio aplikacije koji poziva algoritam prepoznavanja i kao izlaznu informaciju vraća numeričku oznaku prepoznate poze šake. Kao ulazna varijabla u funkciju za prepoznavanje poze postavlja se prilagođeni oblik liste koordinata svih zglobova šake. Varijabla *pre_processed_landmark_list* je 1D lista koja sadrži *x* i *y* koordinate svakog zgloba. Izlazna varijabla *hand_sign_id* poprima iznos numeričke vrijednosti te se prema potrebi ta vrijednost može povezati sa tablicom koja sadrži informacije o imenima određenih poza. U sklopu ovoga rada to nije potrebno već se uspoređuju numeričke vrijednosti izlazne varijable.

```
211                 if hand_sign_id == 0:
212                     print("stop_znak")
213                     kontrola = False
214                 elif hand_sign_id == 1:
215                     print("kreni_znak")
216                     kontrola = True
```

Slika 29: Kontrola znak

Na slici 29 je vidljivo kako u slučaju kada je prepoznata numerička vrijednost poze jednaka „0“ pokreće se protokol za zaustavljanje robota te u tom slučaju nije potrebno provjeravati udaljenost između robota i ruke.

2.8. Komunikacija sa Arduino UNO

Prilikom provjere najmanje udaljenosti između čovjeka i robota ili prilikom prepoznavanja određene poze šake, potrebno je provesti komunikaciju sa izvršnim dijelom vizijskog sustava. U ovome slučaju izvršni dio sustava je razvojna pločica Arduino UNO koja upravlja elektromagnetskim sklopkama kojima se pokreće zaustavljanje robotske ruke. Uz osnovnu funkciju zaustavljanja robota, Arduino UNO vrši i zadaću upozoravanja čovjeka o smanjenoj udaljenosti od robota. Upozoravanje je izvedeno pomoću svjetlosnog i zvučnog signala.
Komunikacija je provedena pomoću USB porta. U programskom jeziku Python, za provođenje serijske komunikacije pomoću USB porta koristi se biblioteka PySerial te je način inicijalizacije proveden kao što prikazuje slika 30.

```
165     ser = serial.Serial('/dev/ttyACM0')
166     ser.baudrate = 9600
167     ser.write_timeout = 0
```

Slika 30: Inicijalizacija USB

Osnovna ulazna informacija za inicijalizaciju je odabir serijskog porta. Odabir željenog porta ovisi o operativnom sustavu koji se koristi. Na operativnom sustavu Windows svaki USB port je označen oznakom „COM*x*“ gdje varijabla *x* predstavlja numeričku vrijednost čije se vrijednosti uglavnom kreću između 1 i 9. Budući kako je za provedbu rada odabrani operativni sustav Debian 11, informacije o USB portovima pohranjene su u datoteke smještene u datoteci /dev.
Prilikom komuniciranja vizijske aplikacije preko USB porta potrebno je svaku varijablu transformirati u tekstualni oblik („string“) te se kao takva može koristiti za komunikaciju. Kao odabrane tekstualne poruke koriste se:

- *stani* – označava da se čovjek previše približio robotu te se robot zaustavlja uz dodatno svjetlosno i zvučno upozorenje
- *kreni* – čovjek se udaljio na sigurnu udaljenost te se nastavlja gibanje robota
- *pazi* – označava da je čovjek u blizini robota te se pokreće zvučno i svjetlosno upozoravanje čovjeka ali se robot ne zaustavlja
- *stani_znak* – prepoznati je znak za zaustavljanje robota te se robot zaustavlja.

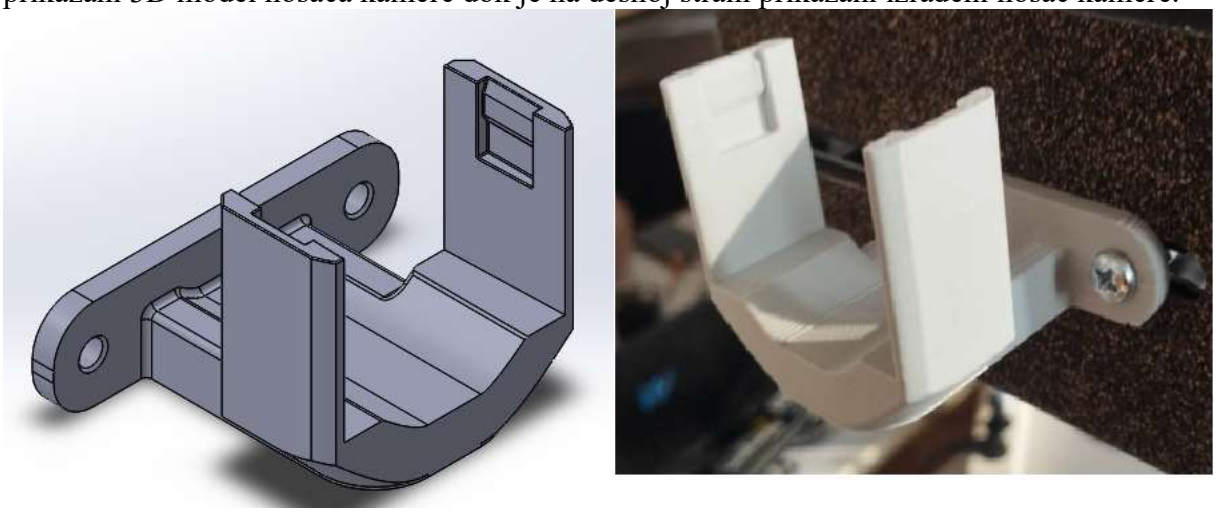
Vizijska aplikacija pokreće algoritam za slanje tekstualne poruke na način prikazan na slici 31.

```
250                 ser.write(str.encode('stani\n'))
```

Slika 31: Slanje poruke

3. Nosač kamere

U sklopu zadatka potrebno je dizajnirati i izraditi nosač kamere. Za provedbu rada odabrana je kamera proizvođača Logitech dok je model kamere „Streamcam“ te je nosač kamere dizajniran prema odabranoj kameri. Uvjet za konstrukciju nosača kamere je da se kamera jednostavno može montirati i demontirati sa nosača te da se nosač može montirati na postojeći stalak za kameru. Prilikom izrade modela nosača kamere postojale su mnoge ideje dok je u konačnici odabrani dizajn prikladan za izradu aditivnim tehnologijama. Na slici je na lijevoj strani prikazani 3D model nosača kamere dok je na desnoj strani prikazani izrađeni nosač kamere.



Slika 32: Nosač kamere

Model nosača je dizajnirani u CAD programu SolidWorks dok je nosač izrađeni pomoću FDM tehnologije na uređaju Ender 3. FDM tehnologija bazirana je na principu taljenja polimerne žice koja se sjedini sa materijalom nanesenim u prethodnom sloju. Nanošenjem velikog broja slojeva (ovisi o dimenziji gotovog proizvoda, dimenziji mlaznice za nanošenje rastaljene žice, željenoj kvaliteti printa) kao rezultat se dobiva funkcionalan predmet. Ova tehnika aditivne proizvodnje u današnje vrijeme je relativno lako dostupna, a omogućuje brzu izradu prototipa. Iako ova tehnologija ima mnogo prednosti, jedan od nedostataka je velika ovisnost čvrstoće o smjeru slojeva.

4. Dinamička analiza opterećenja

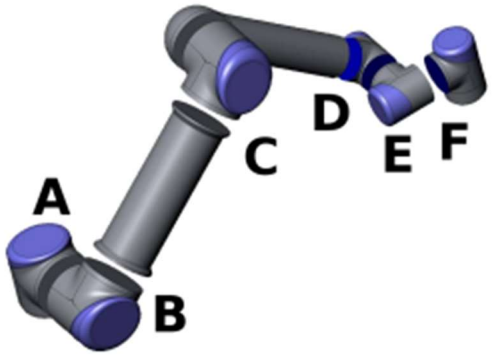
U sklopu ovoga poglavlja biti će obrađena dinamička analiza gibanja robota u normalnom radu, u slučaju aktiviranja sigurnosnog sustava te će na kraju biti provedena usporedba dobivenih rezultata. Analiza je provedena na temelju najvećeg preporučenog dohvata robotske ruke UR5 te za preporučene vrijednosti brzina i ubrzanja alata priključenog na robotsku ruku. Preporučene vrijednosti brzine i ubrzanja preuzete su iz korisničkog priručnika i iz dostupnih informacija dobivenih od strane razvojnih programera firme Universal Robots. Dinamička analiza provedena je pomoću programskog paketa MBDyn. Osnovne informacije o programskom paketu i postupak izrade matematičkog modela te provedbe analize biti će opisani u idućim potpoglavljima.

4.1. MBDyn [16]

MBDyn je prvi besplatan program općenite primjene namijenjen analiziranju dinamike više tijela u sklopu. Program je razvijen u Italiji na sveučilištu „Politecnico di Milano“. MBDyn je aktivno razvijani program i koristi se za simuliranje letjelica, vjetroelektrana, automobila i raznih mehatroničkih sustava. U program su, uz algoritam za analiziranje sustava krutih tijela, integrirani algoritmi za nelinearnu analizu fleksibilnih tijela, električne mreže, hidrauličke mreže i mnogi drugi. Iako program ima širok spektar mogućnosti, njegova primjena je ograničena na uzak krug korisnika. Jedina mana programa je ta što nema grafičko sučelje te je matematički model sustava potrebno tekstualno modelirati. Druga stavka zbog koje je MBDyn ograničeni na mali broj korisnika je taj što je ograničeni na operacijske sustave na bazi linuxa. Sam program razvija se neprekidno od strane zajednice korisnika.

4.2. Modeliranje zadanog problema

Prije samog početka modeliranja problema potrebno je provesti analizu modela koji je potrebno modelirati.



Slika 33: Zglobovi robotske ruke [13]

Slika 33 prikazuje zglobove od kojih se sastoji robot UR5. Uz raspodjelu zglobova na robotskoj ruci, potrebno je odrediti uvjete opterećenja i položaja za koje se provodi dinamička analiza. U ovome radu je provedena dinamička analiza za slučaj kada se zglob *F* nalazi na najvećem preporučenom dohvatu koji iznosi 850 mm. U korisničkom priručniku navedena je maksimalna nosivost robotske ruke koja iznosi 5 kg te se povećavanjem dohvata na 850 mm smanjuje na 3 kg. Za potrebe rada uzeta je u obzir vrijednost tereta koja iznosi 3

kg. Uz odabir položaja i opterećenja potrebno je odrediti profil brzina. Za potrebe rada je potrebno odrediti vrijeme potrebno za ubrzavanje od 0 do maksimalne kutne brzine te iznos maksimalne preporučene kutne brzine. Odabrano vrijeme ubrzavanja iznosi 1,2 s dok iznos preporučene kutne brzine iznosi 1,765 rad/s. S obzirom da se u radu provodi usporedba opterećenja na robotsku ruku u normalnom radu i prilikom aktiviranja sustava kočenja potrebno je i odrediti vrijeme u kojemu se robotska ruka zaustavi prilikom aktivacije sigurnosne kočnice. Prema priručniku vrijeme potrebno za zaustavljanje prilikom aktivacije sigurnosne kočnice iznosi 0,5s.
Modeliranje matematičkog modela započinje zadavanjem imena problemu.

```
begin: data;
   problem: initial value;
end: data;
```

Slika 34: Zadavanje problema

Slika 34 prikazuje početak modeliranja matematičkog modela. Nakon definiranja imena problema koji je potrebno riješiti, potrebno je zadati inicijalne vrijednosti.

```
begin: initial value;
   initial time:   0.;
   final time:     5.;
   time step:      1.e-3;
   max iterations: 10;
   tolerance:      1.e-6;
end: initial value;
```

Slika 35: Inicijalne vrijednosti

Slika 35 prikazuje inicijalne vrijednosti koje je potrebno zadati i njihovi odabrani iznosi. Krajnje vrijeme, kao i korak, odabire se proizvoljno tj. prema potrebama dinamičke analize. Idući korak je postavljanje kontrolnih vrijednosti.

```
begin: control data;
   structural nodes: 9;
   rigid bodies:     8;
   joints:           9;
   gravity;
end: control data;
```

Slika 36: Kontrolne vrijednosti

Na slici 36 je vidljiv broj krutih tijela koji opisuje problem, broj zglobova i postoji li utjecaj gravitacije. Pomoću ovih vrijednosti program vrši provjeru prilikom provedbe dinamičke analize prema matematičkom modelu. Kada su zadane sve kontrolne vrijednosti potrebno je zadati nazive za sve čvorove, kruta tijela i zglobove kako bi model bio pregledniji.

```
set: integer JoAxRot_Link0_Link1  = 1;
set: integer JoAxRot_Link1_Link2  = 2;
set: integer JoClamp_Ground       = 3;
set: integer JoAxRot_Link2_Link3  = 4;
set: integer JoRevh_Mab           = 5;
set: integer JoRevh_Mc            = 6;
set: integer JoRevh_Md            = 7;
set: integer JoRevh_Met           = 8;
set: integer JoAxRot_Link0_Ground = 9;
```

Slika 37: Imena zglobova

Slika 37 prikazuje način zadavanja imena pojedinim zglobovima. Iako sam način zadavanja nije bitan, bitna je brojčana vrijednost navedena pokraj, postoje preporuke za zadavanje imena varijablama kako bi se postigla što veća preglednost. Idući korak modeliranja problema je zadavanje početnog stanja svakog čvora.

```
structural: Node_Link1, dynamic,
   reference, Ref_Link1, 1./2.*L1, 0., 0., # absolute position
   reference, Ref_Link1, eye,              # absolute orientation
   reference, Ref_Link1, null,             # absolute velocity
   reference, Ref_Link1, null;             # absolute angular velocity
```

Slika 38: Zadavanje čvorova

Slika 38 prikazuje način zadavanja početnih parametara čvorova. Vidljivo je kako je radi jednostavnosti modeliranja moguće modelirati i pomoćne koordinatne sustave te se prilikom zadavanja čvorova referirati na pomoćni koordinatni sustav. Referiranje sa provodi pomoću naredbe *reference* iza koje se navodi ime pomoćnog sustava. Vrlo je bitno napomenuti kako se koordinate pomaka zadaju u metrima, zakreta u radijanima, brzine u metrima po sekundi dok se kutna brzina zadaje u radijanima u sekundi. Naredba *null* označava kako su vrijednosti u smjeru sve tri koordinatne osi jednake 0.
Kada su zadani početni parametri svih čvorova potrebno je zadati parametre krutih tijela.

```
body: Body_Link2, Node_Link2,
   M2,                              # mass
   null,                            # relative center of mass
   diag, 0., M2*L2^2./12., M2*L2^2./12.; # inertia matrix
```

Slika 39: Zadavanje krutih tijela

Slika 39 prikazuje potrebne parametre i ormu zadavanja krutog tijela. Vidljivo je kako se svaki čvor povezuje sa pripadnim čvorom koji predstavlja centar mase elementa te se matrica inercije zadaje prema pripadnom čvoru. Nakon zadavanja svih krutih tijela potrebno je zadati zglobove.

```
joint: JoClamp_Ground,
   clamp,
      Node_Ground,
         null,
         eye;
joint: JoAxRot_Link2_Link3,
   axial rotation,
      Node_Link2,
         reference, Ref_Link2, null,                                  # relative offset
         hinge, reference, Ref_Link3, 1, 0., 0., 1., 3, 1., 0., 0., # relative axis orientation
      Node_Link3,
         reference, Ref_Link2, null,                                  # relative offset
         hinge, reference, Ref_Link3, 1, 0., 0., 1., 3, 1., 0., 0., # relative axis orientation
      const, 0.0;
```

Slika 40: Spojevi

MBDyn nudi širok izbor postojećih zglobova te je potrebno odabrati zglob koji može najtočnije predstavljati stvarni zglob. Na svakom zglobu robotske ruke nalazi se motor koji pokreće ruku te je primjereno korištenje zgloba pod nazivom *axsial rotation*. Ovaj zglob onemogućuje sve tri translacije i dvije rotacije. Preostali stupanj slobode koristi se kao motor te se za njega zadaje željeni broj okretaja. Postoji mnogo načina zadavanja aktuatora dok se u ovome radu koriste *const,* koji predstavlja konstantnu brzinu u cijelom vremenu simulacije i *skalar function* u koja može predstavljati mnoge linearne promjene brzine. Uz modeliranje zglobova potrebno je postaviti i nepomičnu podlogu na koju se pomoću zgloba veže željeni dio. Postavljanje nepomične podloge provodi se pomoću naredbe *clamp* pomoću kojeg se ograničava svih 6 stupnjeva slobode tijelu (u ovome slučaju podlozi).

```
scalar function: "vozi",
   multilinear,
      0.0, 0.0,
      1.2, 1.765,
      3.2, 1.765,
      4.4, 0.0,
      5.0, 0.0;
```

Slika 41: Zadavanje raspodjele kutne brzine

Slika 41 prikazuje način zadavanja kutne brzine zgloba numeriranog sa brojem 9. Opcija *multilinear* omogućuje linearnu raspodjelu vrijednosti kutne brzine između dvaju susjednih zadanih vrijednosti.

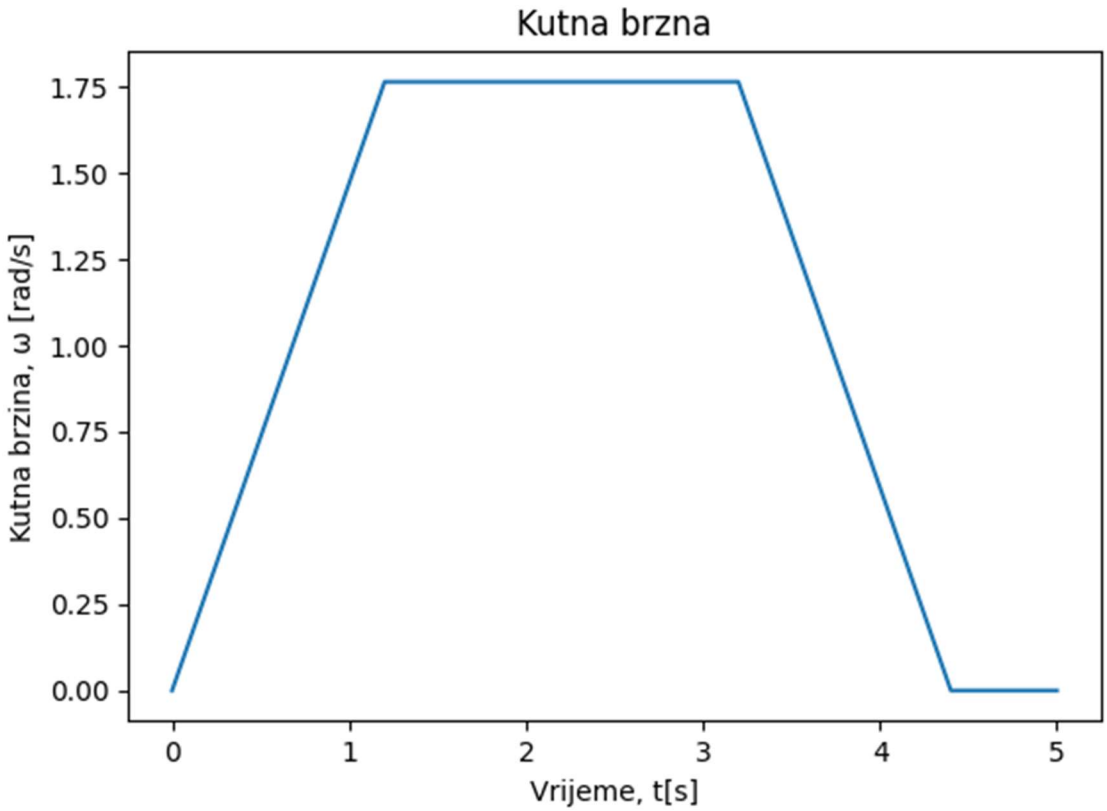
```
gravity: 0., 0., -1., const, 9.81;
```

Slika 42: Gravitacija

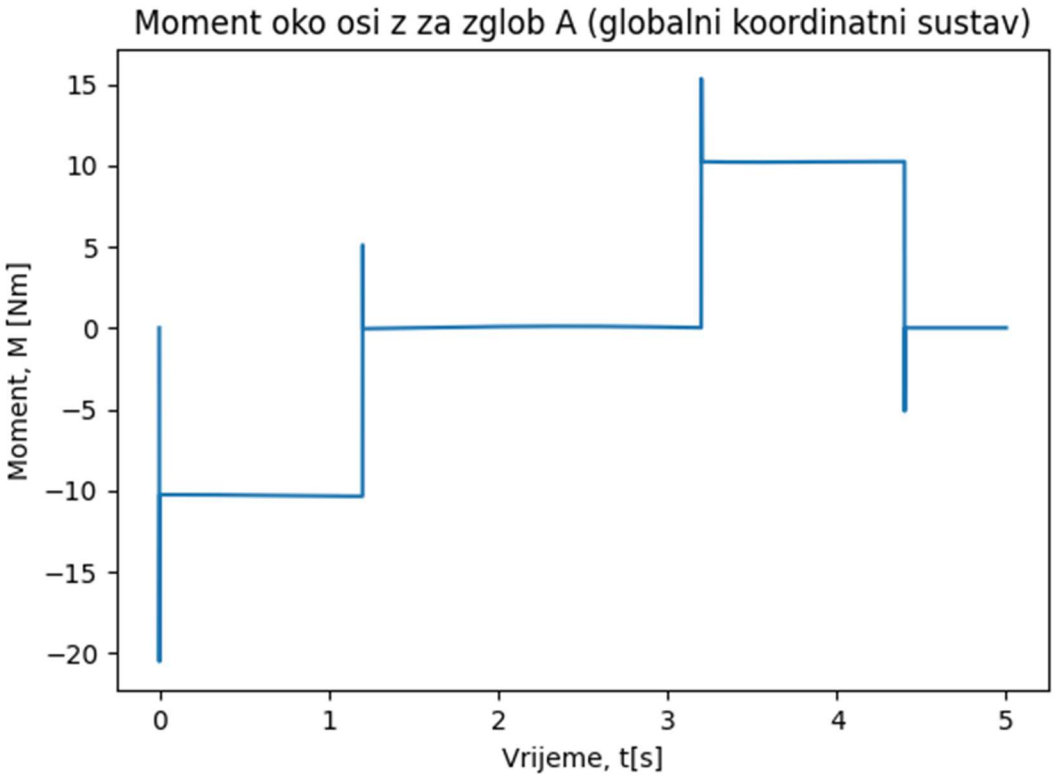
Slika 42 prikazuje način zadavanja utjecaja gravitacije. Vidljivo je kako gravitacija djeluje u smjeru suprotnom od osi *z* sa iznosom od 9,81 m/s^2. Zadavanjem gravitacije završava se matematičko modeliranje problema. Kako bi se mogla provesti usporedba dobivenih vrijednosti u normalnom radu i prilikom aktiviranja sustava kočenja izrađena su dva modela koja se razlikuju jedino u raspodjeli brzina i ubrzanja. Prvi slučaj je provedeni sa preporučenim vrijednostima brzina i ubrzanja dok je za drugi slučaj simulirani proces normalnog ubrzavanja te nakon toga prisilnog kočenja. Prema priručniku je zadano da se ruka mora zaustaviti za najviše 500 ms prilikom čega se brzina smanjuje linearno. Za modeliranje linearne promjene brzina može se koristiti funkcija *multilinear* koja provodi linearnu. interpolaciju između zadanih točaka.

4.3. Prikaz rezultata

Programski paket MBDyn kao rezultate stvara nekoliko datoteka od kojih su najvažnije one sa ekstenzijama .mov i .jnt. Navedene datoteke sadrže podatke o koordinatama, brzinama, kutovima zakreta i kutnim brzinama svih čvorova te sadrže sile i momente koji opterećuju sve zglobove. Dobiveni podaci su neupotrebljivi u trenutnom načinu zapisa te ih je potrebno procesuirati kako bi se podaci prilagodili za prikaz. Datoteka sa prikazom opterećenja zglobova pohranjena je sa ekstenzijom .jnt te sadrži podatke o silama i momentima koji opterećuju pojedini čvor. U prvome stupcu .jnt datodeke nalazi se podatak o tome na koji čvor se vrijednosti opterećenja odnose, u idućih 6 stupaca upisane su vrijednosti opterećenja prikazane u lokalnim koordinatnim sustavima svakog zgloba. Nakon vrijednosti opterećenja izraženih u lokalnim koordinatnim sustavima zapisani su i podaci o opterećenju s obzirom na globalni koordinatni sustav. U radu je provedena analiza za 4 čvora (čvorovi *E* i *F* na slici 33 nisu uzeti u obzir). Za svaki čvor izračunate su vrijednosti iznosa sila u smjeru sve tri koordinatne osi i vrijednosti momenata oko sve tri koordinate osi. Kako bi se izračunata opterećenja mogla prikazati što preglednije, prikazani će biti prvo svi dijagrami za slučaj normalnog rada a nakon toga dijagrami dobiveni za slučaj pokretanja kočnice. Provođenjem analize određeno je kako je najopterećeniji čvor *A* te su za njega svi dijagrami prikazani niže u tekstu dok su dijagrami za ostale čvorove stavljeni u prilog.

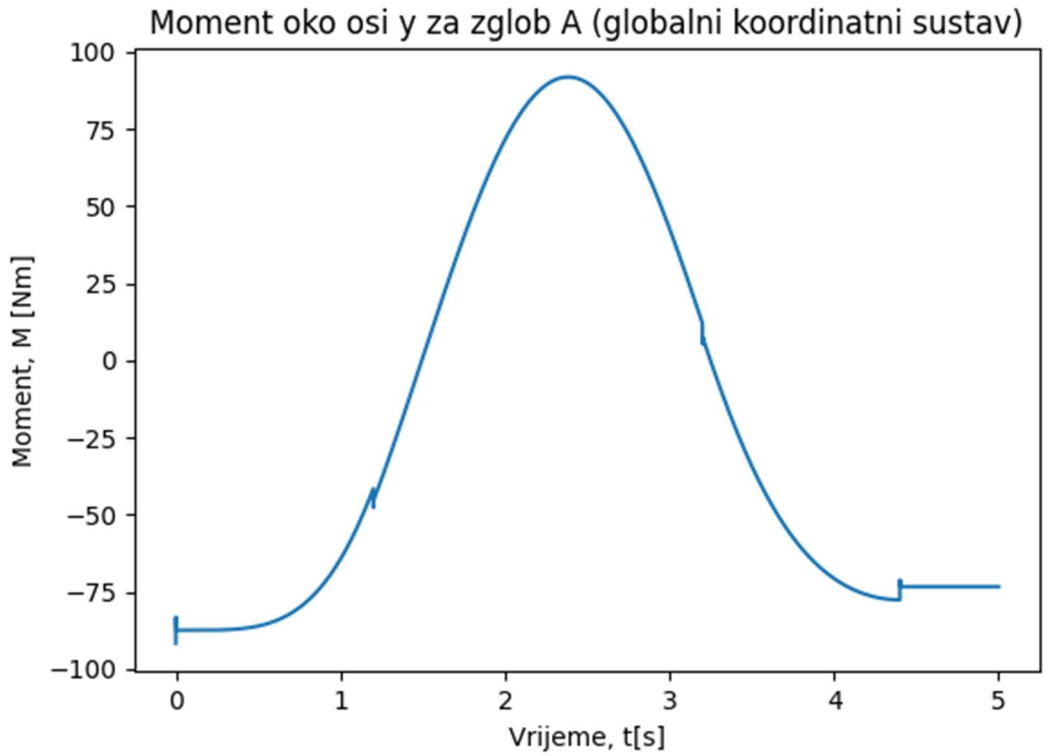


Slika 43: Raspodjela brzine u normalnom radu

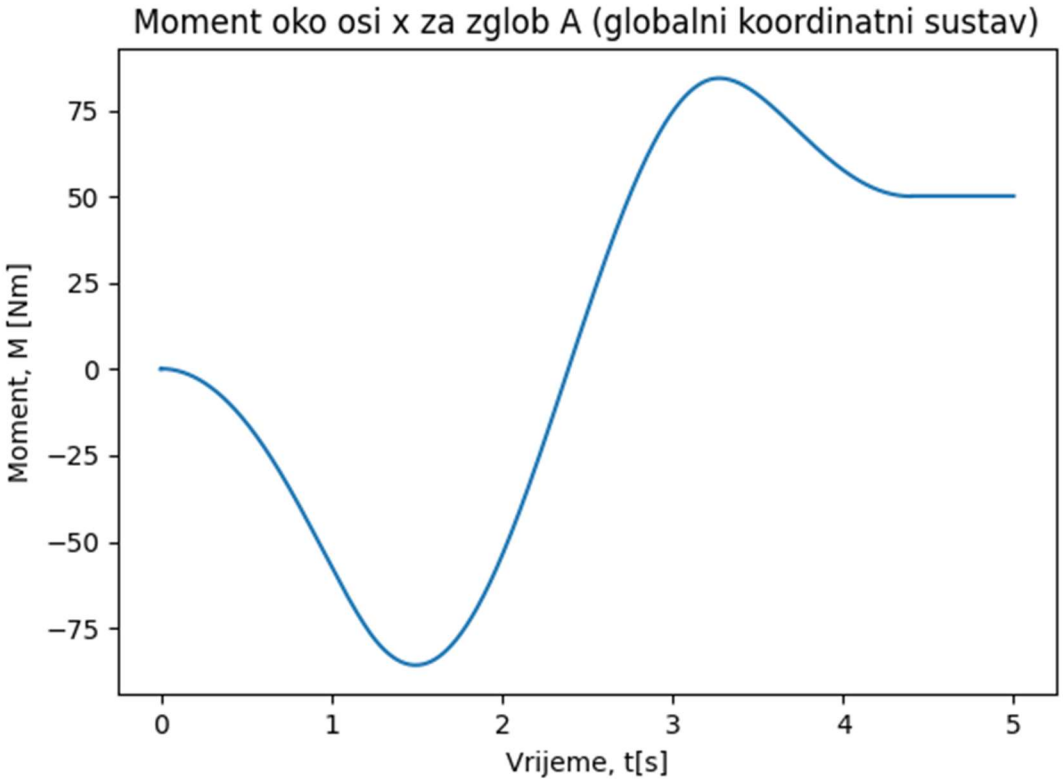


Slika 44: Raspodjela momenata u normalnom radu oko globalne osi Z

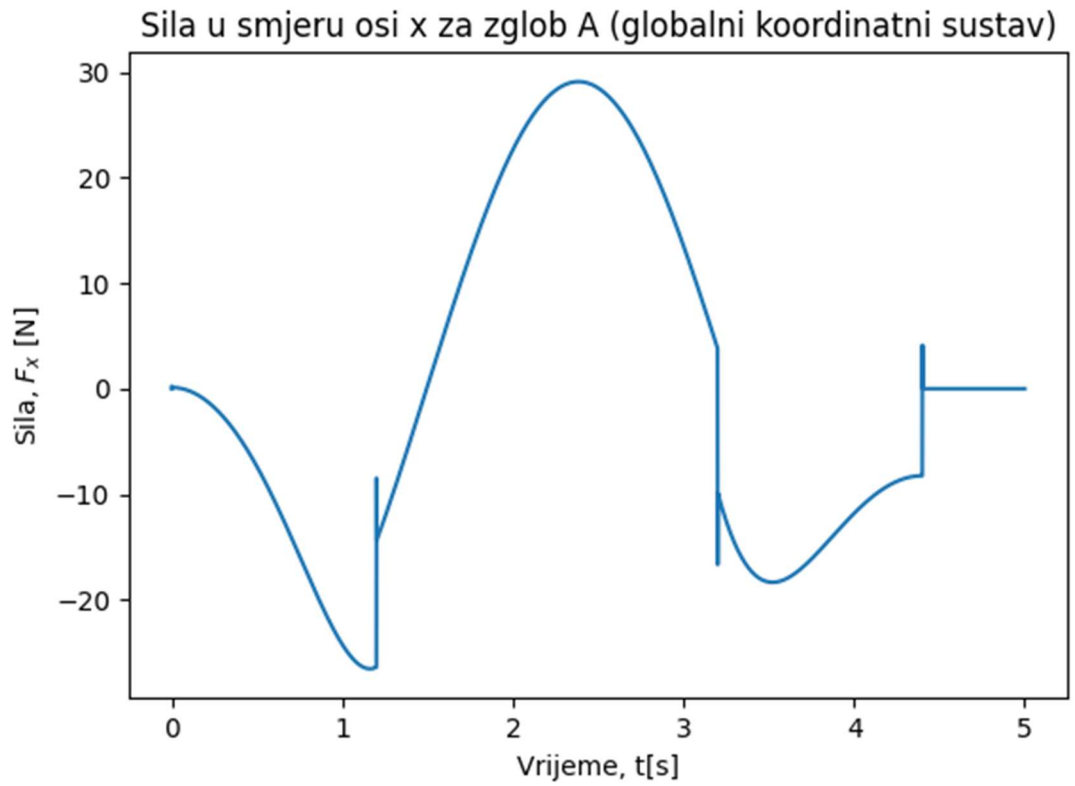
Slika 43 prikazuje raspodjelu kutne brzine u normalnom radu. Budući kako se radi o pojednostavljenoj analizi, samo jedan čvor sadrži relativnu brzinu naspram prethodnog. Na slici 44 prikazana je raspodjela momenata na mjestu motora koji se nalazi u bazi robota (zglob *A*). Najveća apsolutna očitana vrijednost momenta iznosi 20 Nm.



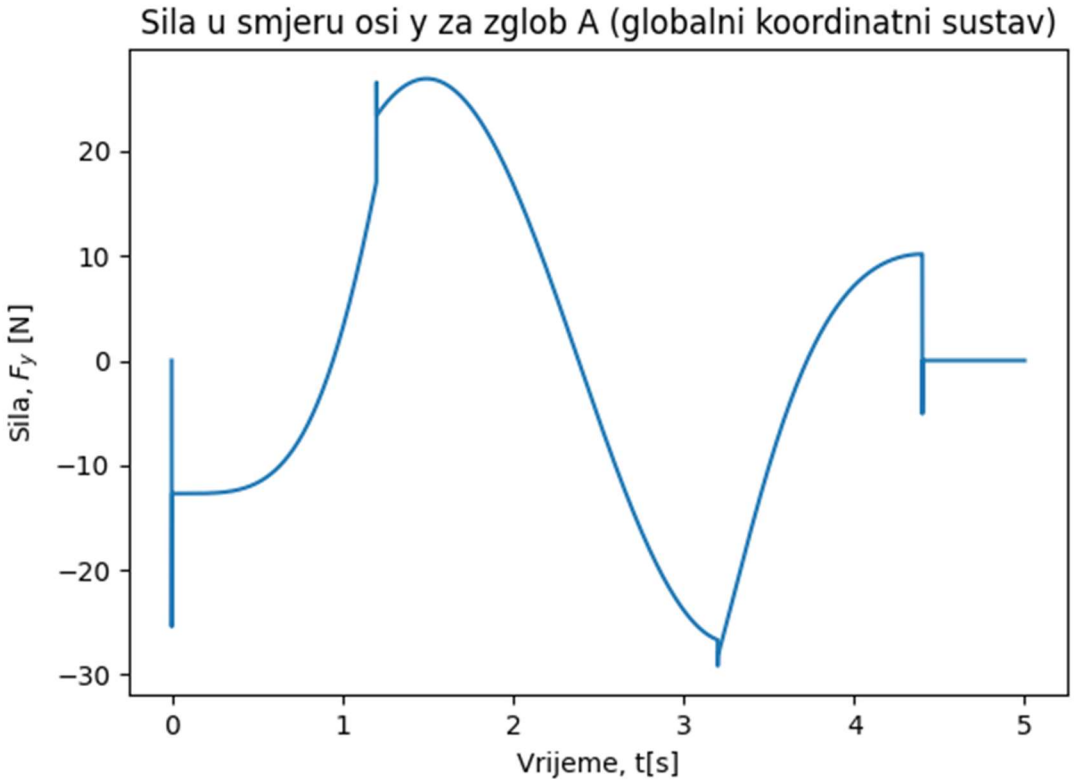
Slika 45: Moment oko osi y u normalnom radu



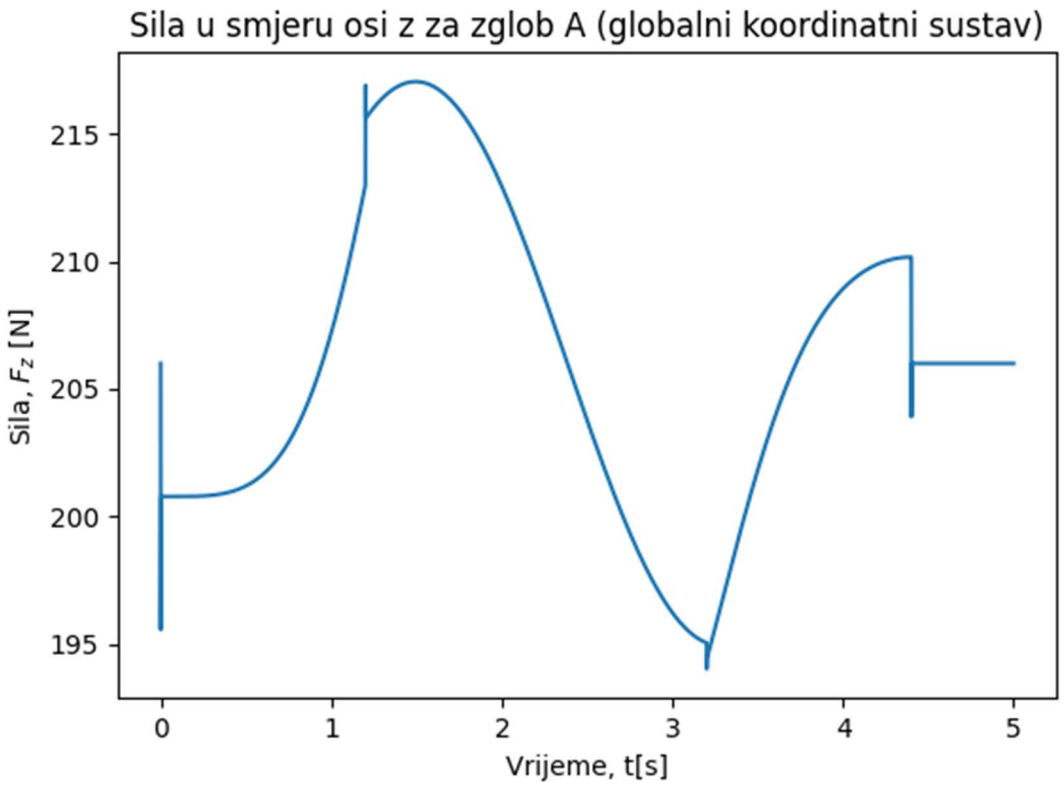
Slika 46: Moment oko osi x u normalnom radu



Slika 47: Sila u smjeru osi x u normalnom radu

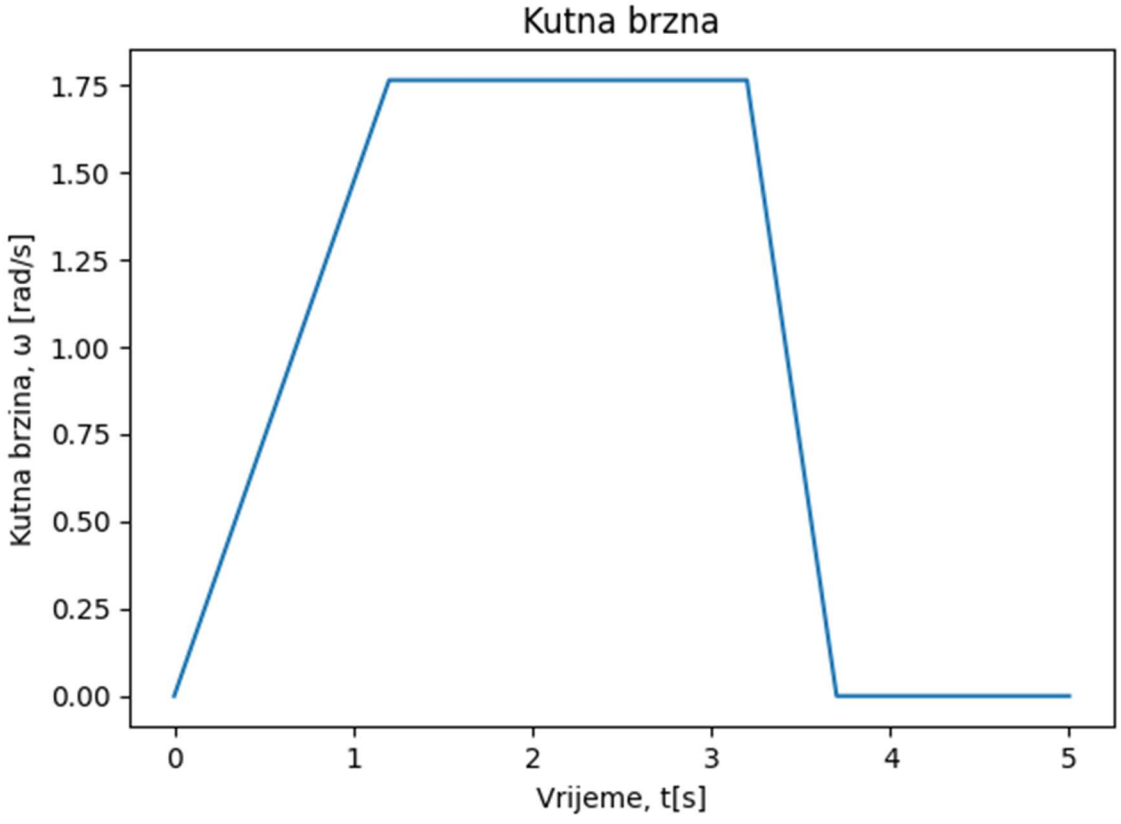


Slika 48: Sila u smjeru osi y u normalnom radu

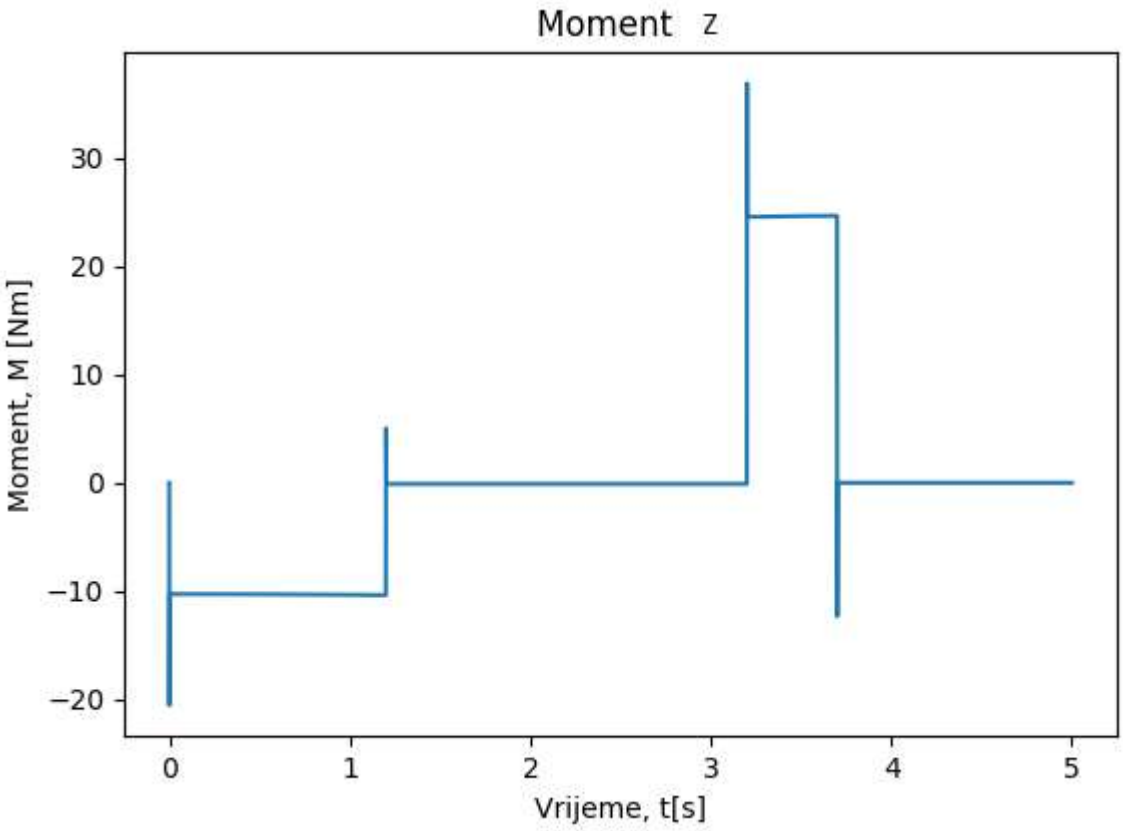


Slika 49: Sila u smjeru osi z u normalnom radu

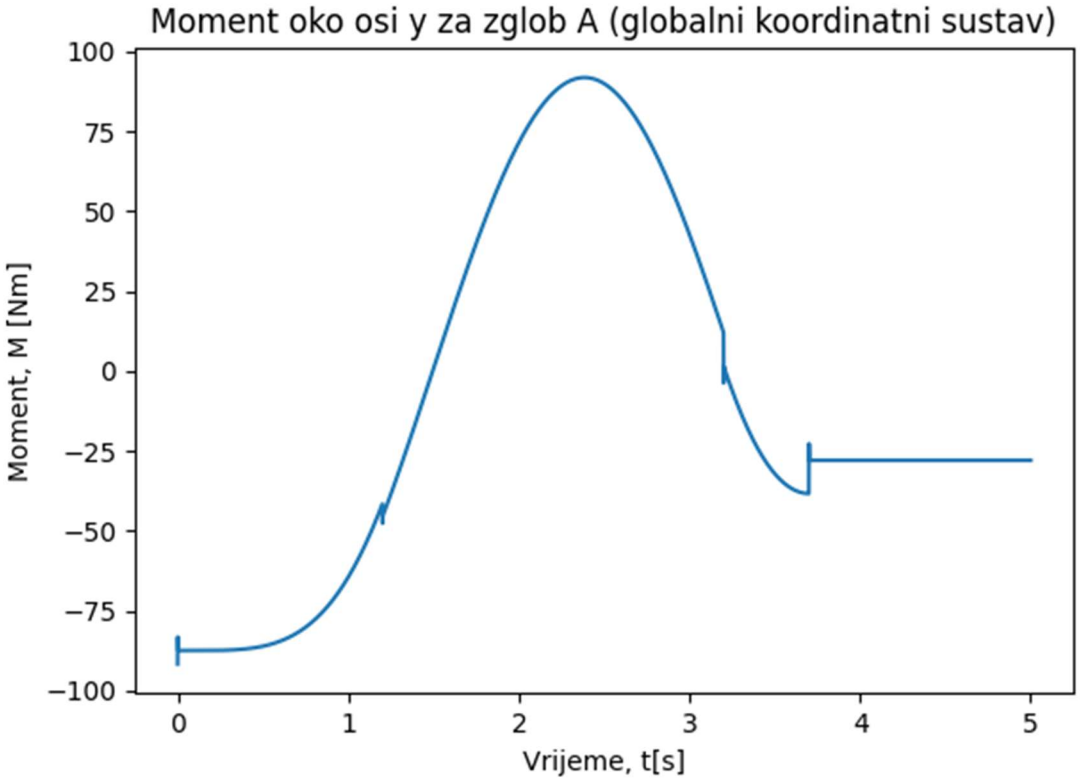
Nakon prikaza rezultata dobivenih za normalni rad potrebno je prikazati i rezultate dobivene za slučaj aktiviranja sigurnosne kočnice.



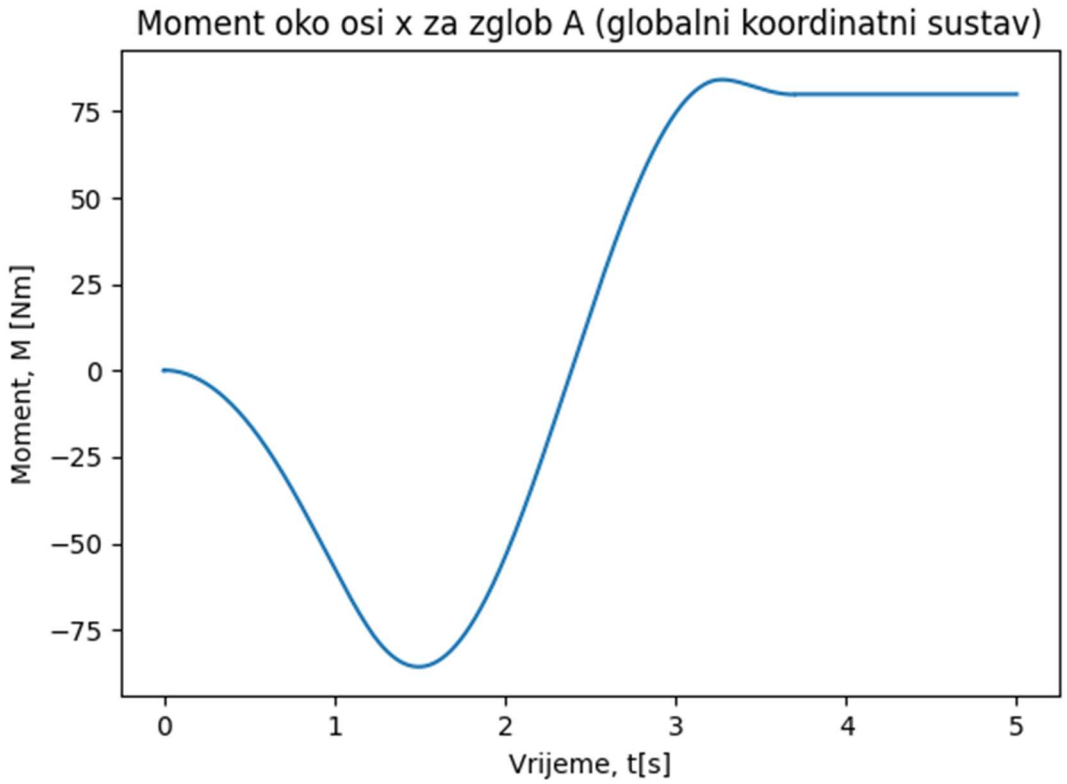
Slika 50: Raspodjela brzina u slučaju aktiviranja kočnice



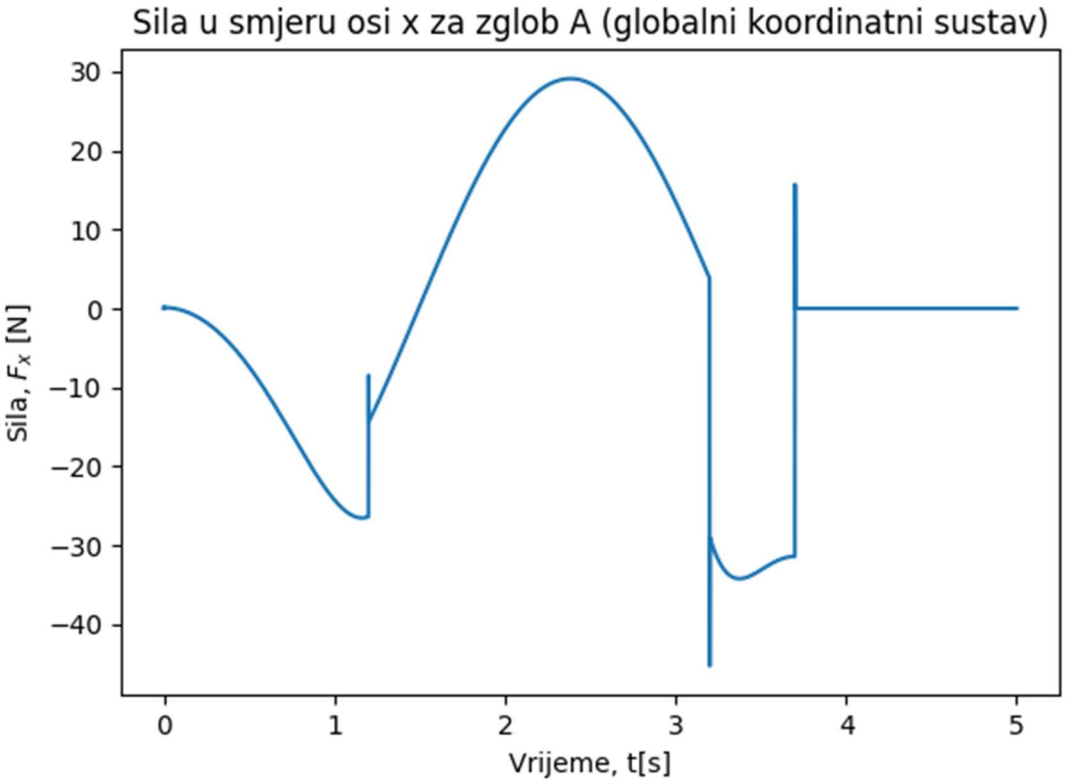
Slika 51: Raspodjela momenata oko osi z za slučaj kočenja



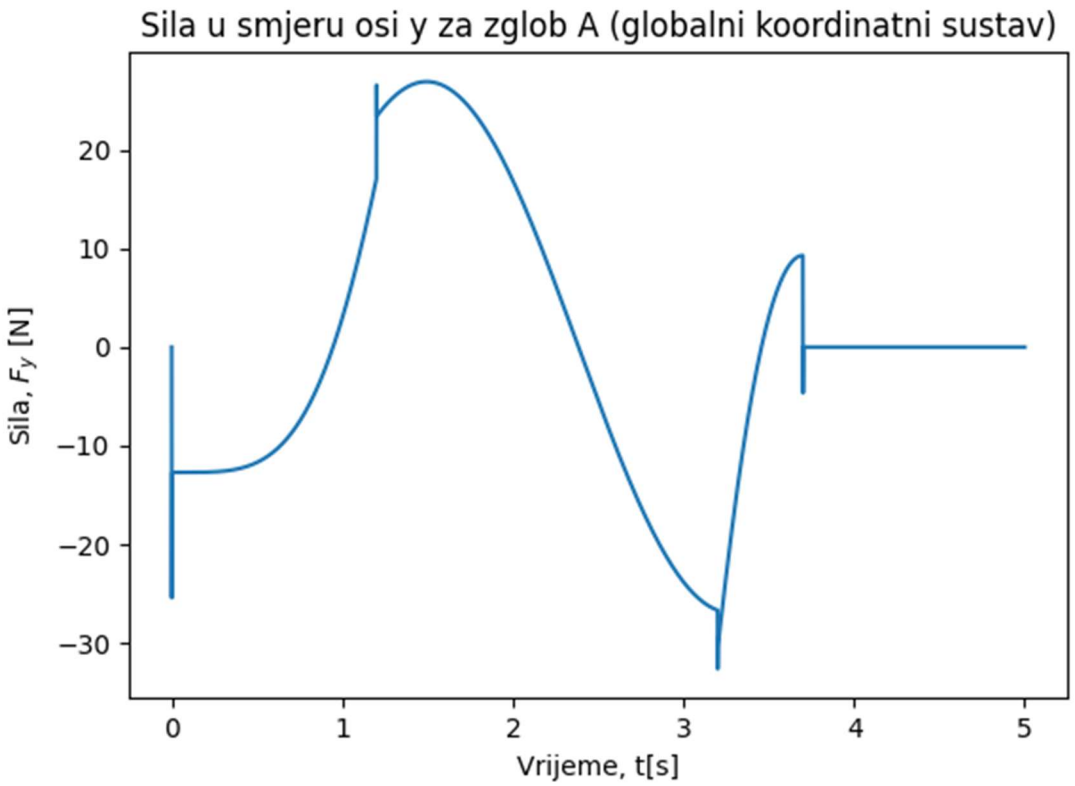
Slika 52: Moment oko osi y za slučaj kočenja



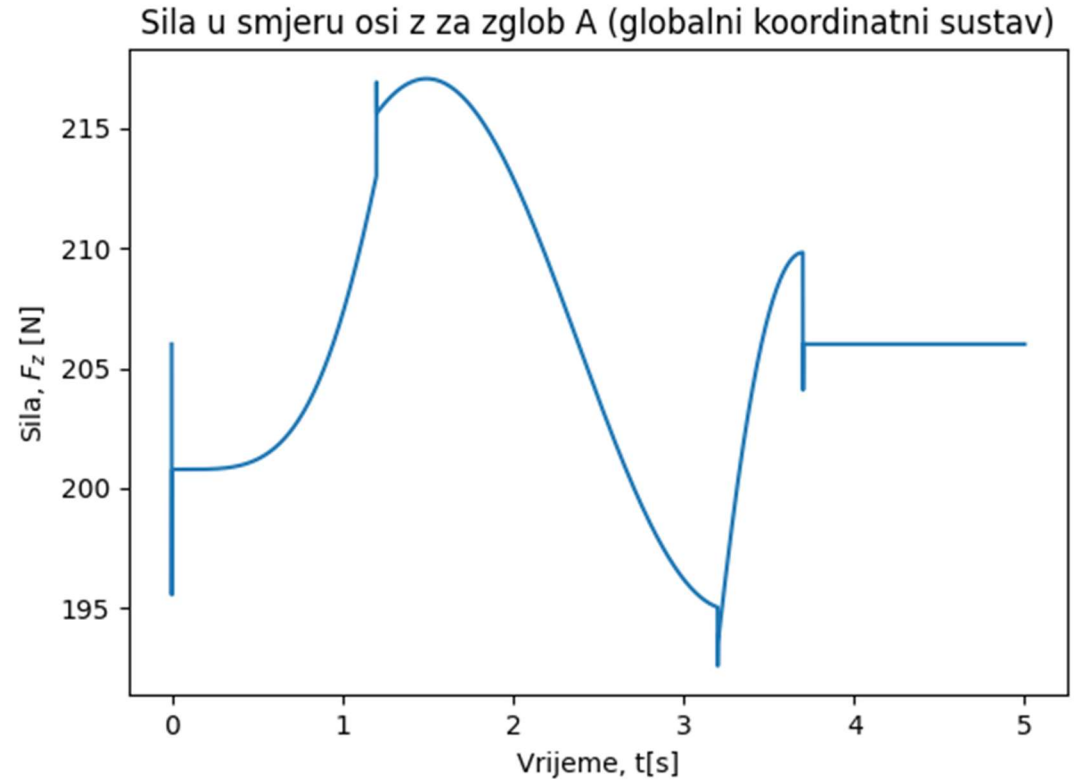
Slika 53: Moment oko osi x za slučaj kočenja



Slika 54: Sila u smjeru osi x za slučaj kočenja



Slika 55: Sila u smjeru osi y za slučaj kočenja



Slika 56: Sila u smjeru osi z za slučaj kočenja

Prvi dio simulacije je jednaki kao i za prethodni slučaj tj. robotska ruka ubrza na preporučenu brzinu i nastavi se gibati tom brzinom. U vremenu od 3,2 sekunde od početka simulacije pokrenuti je proces kočenja. Proces kočenja traje najviše 0,5 sekundi. Na slici 51 očitani je maksimalni iznos momenta na motoru od 38 Nm.
Iz očitanih vrijednosti moguće je izraziti relativnu postotnu razliku u očitanim maksimalnim momentima za prvi i drugi slučaj:

$$\%M_z = \frac{M_k - M_n}{M_n} = \frac{38 - 20}{20} = 0{,}9. \quad (1)$$

Ovime je dokazano kako prilikom aktivacije sigurnosne kočnice robotsku ruku opterećuju dodatne dinamičke sile. Tokom provođenja analize zaključeno je kako najveće opterećenje na zglob *A* gdje najveći utjecaj aktivacije kočnice je na moment oko osi z. Moment u zglobu A oko osi z je 90 $^{0}/_{0}$. Kako bi se izbjeglo nepotrebno oštećivanje robota potrebno je pridržavati se preporučenih vrijednosti brzina i ubrzanja alata te pridržavati se ovisnosti maksimalne nosivosti robota i radijusa dohvata.

5. Zaključak

U rad je opisani postupak izrade vizijske aplikacije za sprječavanje kolizije robota i čovjeka. Korištenjem računalnog vida provedeno je prepoznavanje ljudske šake i određenih poza šake. Radni prostor robota snimati je sa jednom kamerom te nedostaje podatak o udaljenosti od kamere ali je jednostavnijom izvedbom omogućena mnogo kvalitetnija obrada teme sa mnogo više razumijevanja. Na početku rada zadane su teorijske osnove potrebne za rješavanje problema dok je tokom rada opisan postupak izrade vizijske aplikacije. Robotska ruka na kojoj je sustav testirani je UR5 te postoji mogućnost komunikacije robotske ruke i vizijske aplikacije putem TCP protokola. Uz izradu vizijske aplikacije konstruirani je nosač kamere koji je izrađen aditivnom tehnologijom FDM. Nosač je konstruirani u CAD programu SolidWorks. Na samome kraju provedena je i dinamička analiza opterećenja robotske ruke u normalnom radu i prilikom kočenja. Najgori slučaj je kada je vrh alata postavljen na najvećem preporučenom dohvatu i kada je robotska ruka opterećena najvećim dopuštenim teretom. Vidljiva je velika razlika u opterećenju te se dokazuje potreba za pridržavanjem korištenja preporučenih vrijednosti brzina i ubrzanja kako bi se produljio radni vijek robotske ruke.
Računalni vid je disciplina koja se neprekidno razvija. Razvojem računala i kamera ubrzo će biti mogući autonomni sustav upravljanja vozilima koji je u potpunosti siguran za sve sudionike u prometu. Kao što se razvija sustav autonomne vožnje razvijaju se još mnogi sustavi bazirani na računalnom vidu te je svakim danom sve izglednija njihova integracija u svakodnevni život.

Literatura:

[1.] „Computer vision: foundaions and applications“ , Ranjay Krišna, Stanford Universiti

[2.] „Vomputer vision: a modern approach“, David. A. Forsyth, Jean Ponce

[3.] https://datasheetspdf.com/pdf-file/555220/OmniVisionTechnologies/OV7670/1, pristupljeno 10.02.2023.

[4.] „Computer Vision Algorithms and Applications“, Richard Szeliski, Springer

[5.] https://en.wikipedia.org/wiki/HSL_and_HSV#/media/File:HSL_color_solid_cylinder_saturation_gray.png, pristupljeno 10.02.2023.

[6.] https://learnopencv.com/edge-detection-using-opencv/, pristupljeno 10.01.2023.

[7.] https://deci.ai/blog/yolov6-vs-yolov5-vs-yolox/, pristupljeno 11.01.2023.

[8.] https://commons.wikimedia.org/wiki/File:OpenCV_Logo_with_text.png, pristupljeno 11.02.2023.

[9.] https://opencv.org/about/, pristupljeno 11.01.2023.

[10.] https://www.fiverr.com/hilbertone/solve-your-mediapipe-related-problems, pristupljeno 12.02.2023.

[11.] https://mediapipe.dev/, pristupljeno 08.01.2023.

[12.] https://en.wikipedia.org/wiki/TensorFlow, pristupljeno 8.02.2023.

[13.] https://www.usna.edu/Users/weaprcon/kutzer/_files/documents/User%20Manual,%20UR5.pdf, pristupljeno 20.01.2023.

[14.] https://www.hackster.io/ansh2919/serial-communication-between-python-and-arduino-e7cce0, pristupljeno 12.02.2023.

[15.] https://google.github.io/mediapipe/solutions/hands, pristupljeno 11.01.2023.

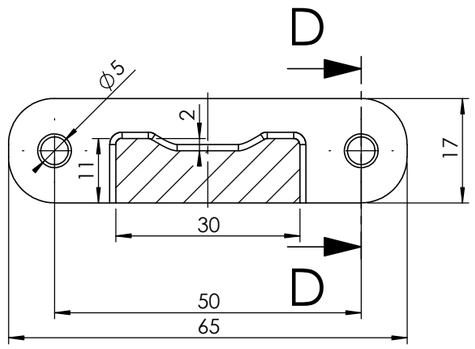
[16.] https://www.mbdyn.org/, pristupljeno 15.02.2023.

[17.] https://www.sky-engin.jp/en/MBDynTutorial/index.html, pritupljeno 15.02.2023.

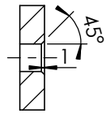
[18.] https://www.mbdyn.org/userfiles/documents/mbdyn-input-1.7.2.pdf

Prilozi

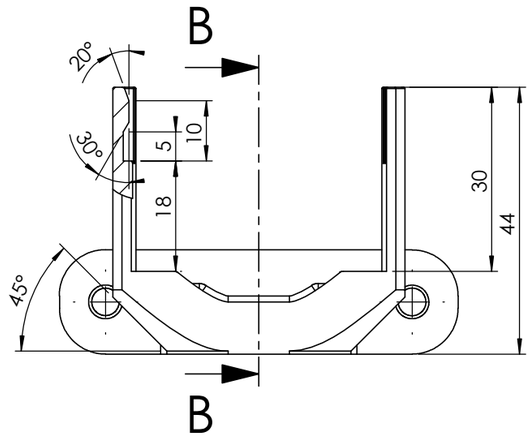
I. Tehnička dokumentacija
II. Dijagrami opterećenja kolaborativnog robota UR5
III. Program vizijske aplikacije
IV. MBDyn kod

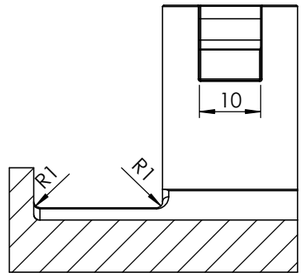


C-C

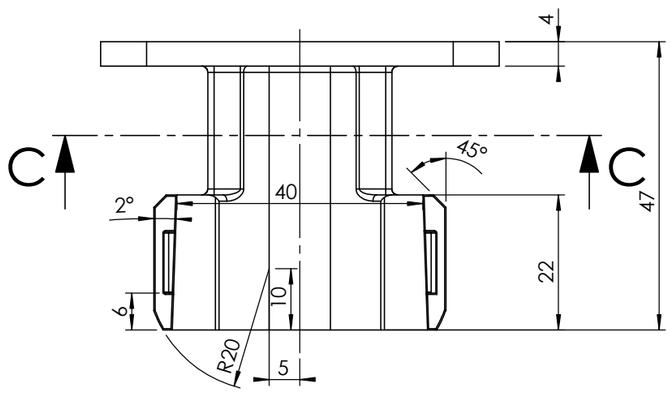


D-D





B-B



NAPOMENE: - svi nekotirani radijusi izvedeni su kao R 0,2 mm
- izraditi aditivnom tehnologijom (FDM metoda)

Poz.	Naziv dijela		Kom.	Crtež broj / Norma	Materijal	Sirove dimenzije / Proizvođač	Masa
Broj naziva - code		Datum	Ime i prezime		Potpis	FSB Zagreb	
	Projektirao	15.2.2023	Lovro Sever				
	Razradio	15.2.2023	Lovro Sever				
	Crtao	18.2.2023	Lovro Sever				
	Pregledao						
	Mentor		Tomislav Stipančić				

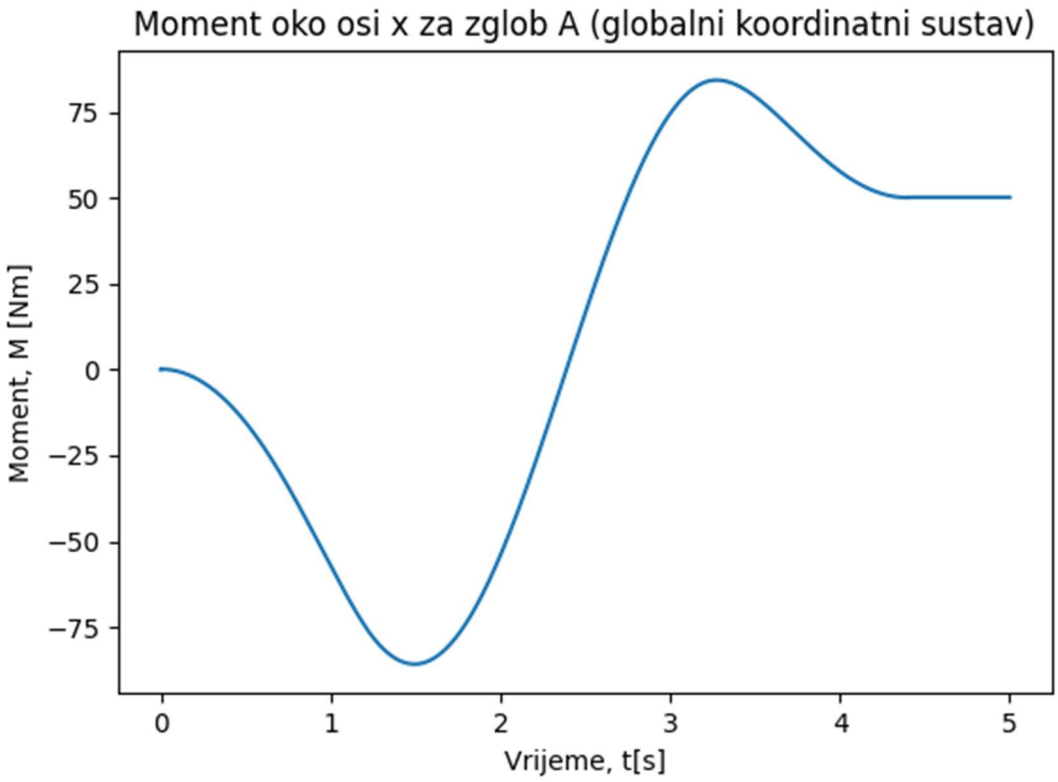
ISO - tolerancije	Objekt:	Objekt broj:
		R. N. broj:
	Napomena:	Kopija
	Materijal: PLA — Masa: 29 g	
	Mjerilo originala: M 2:1	Naziv: Nosač kamere — Pozicija:
		Format: A3
		Listova: 1
	Crtež broj: LS_2302_01	List: 1

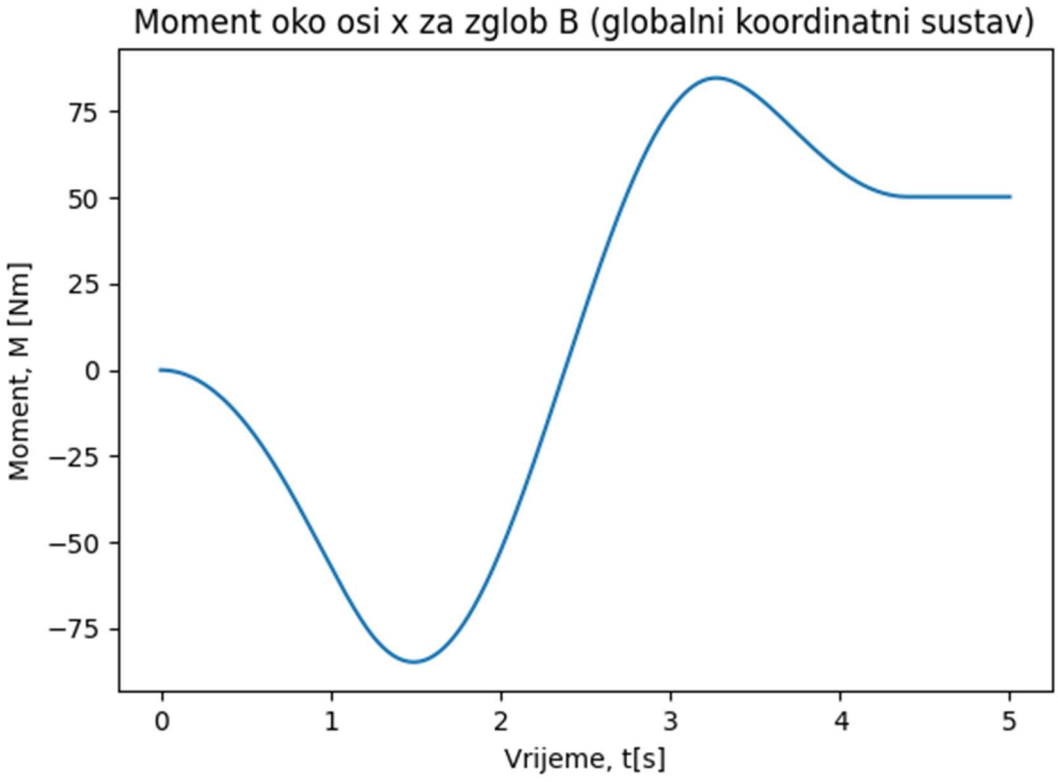
Design by CADLab

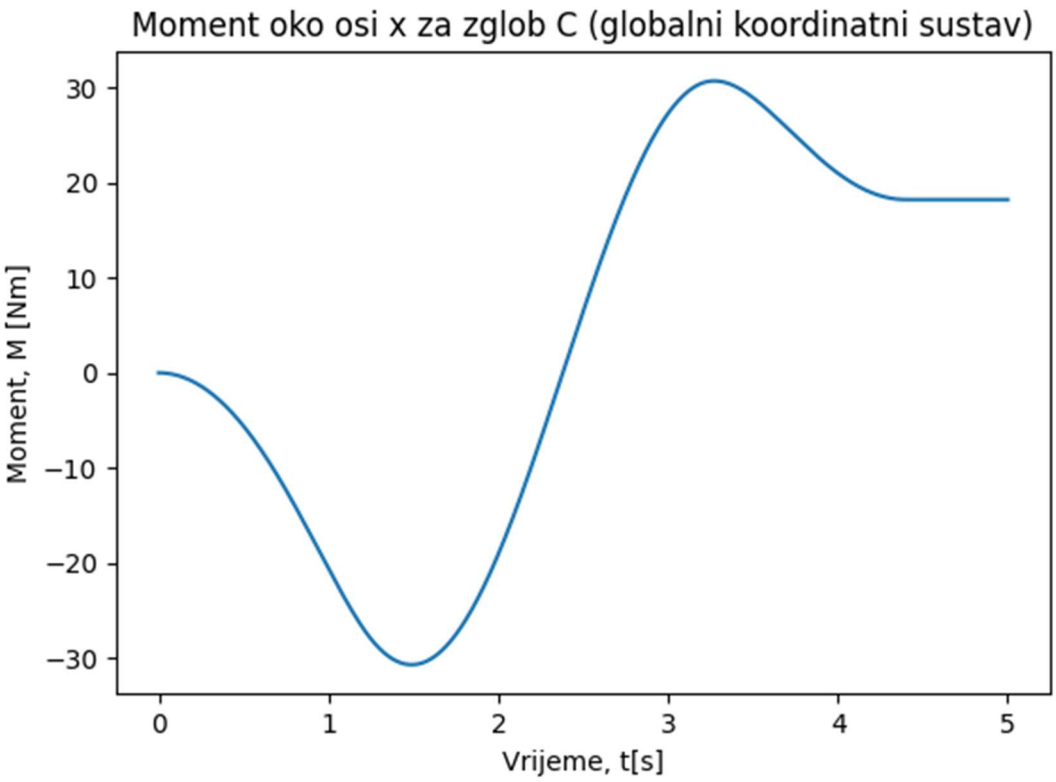
0 10 20 30 40 50 60 70 80 90 100

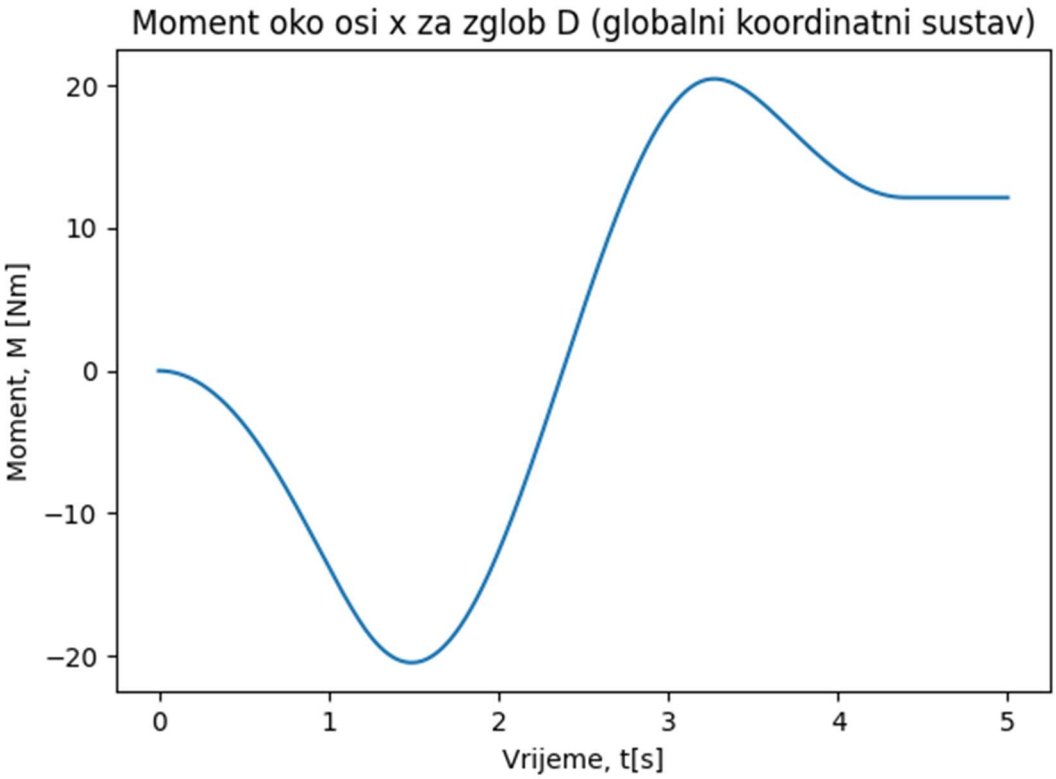
DIJAGRAMI OPTEREĆENJA KOLABORATIVNOG ROBOTA UR5

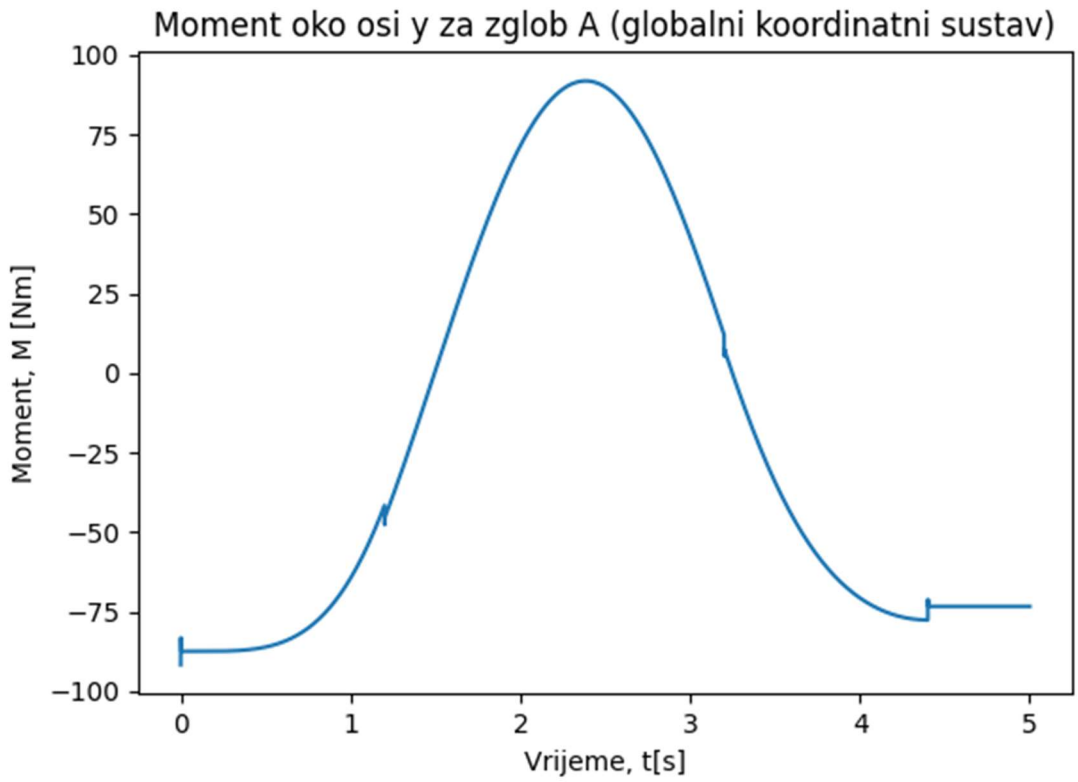
Opterećenje robota u normalnom radu

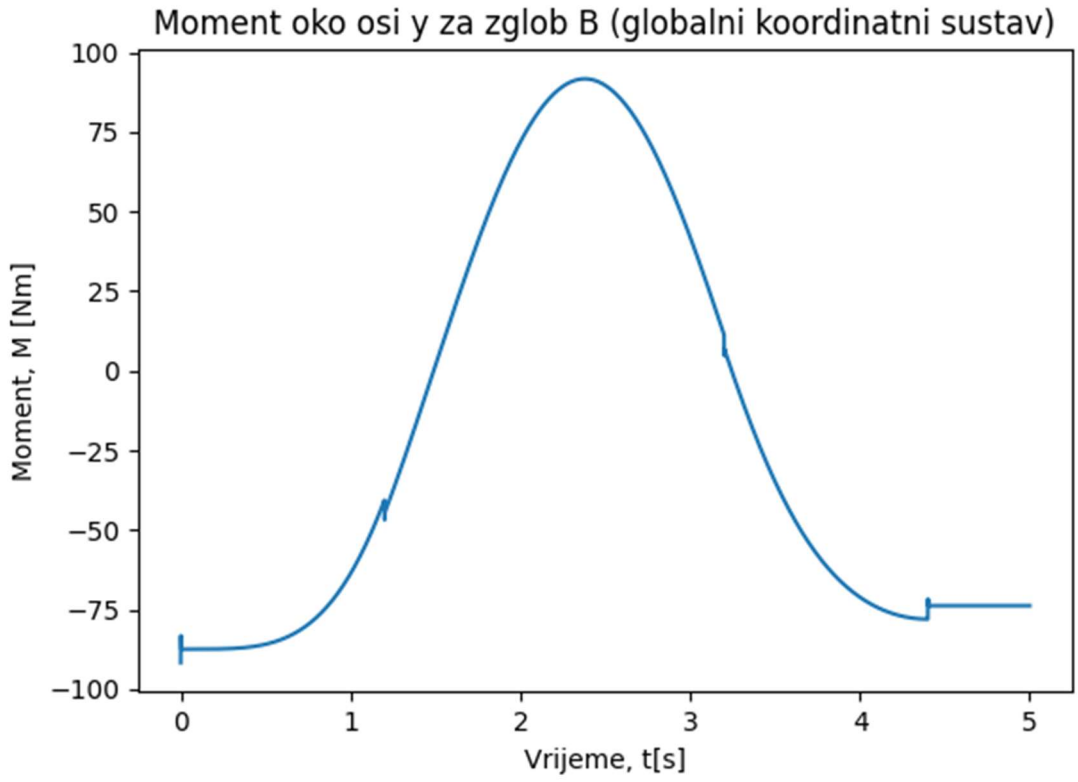


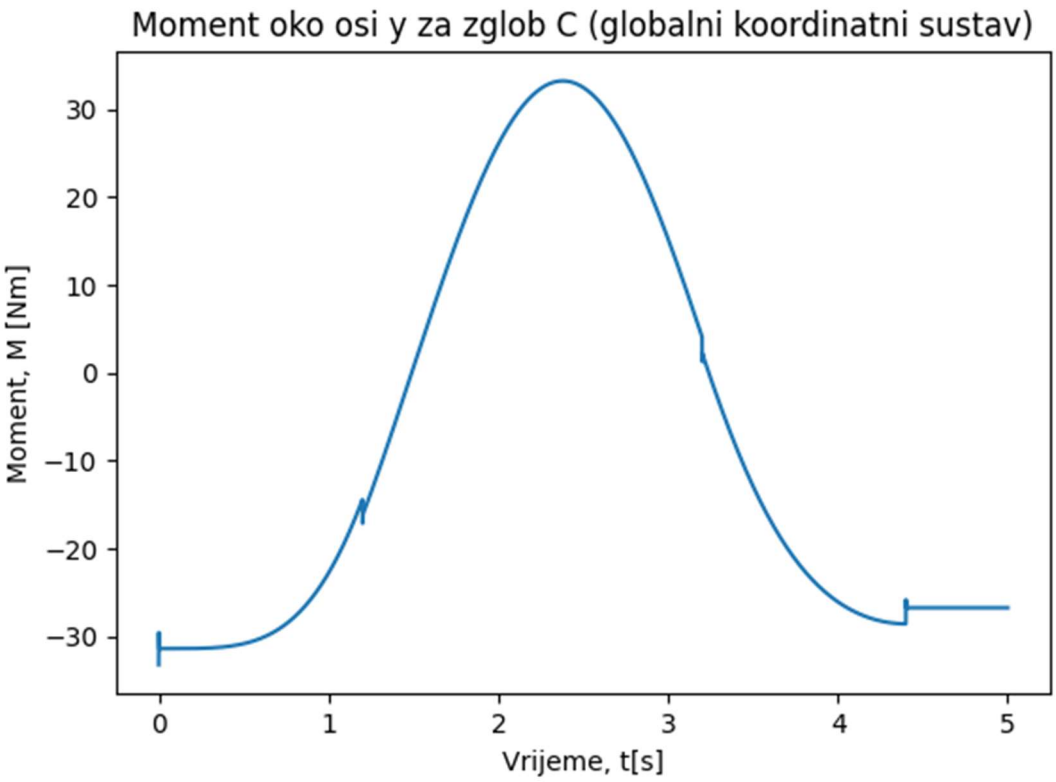


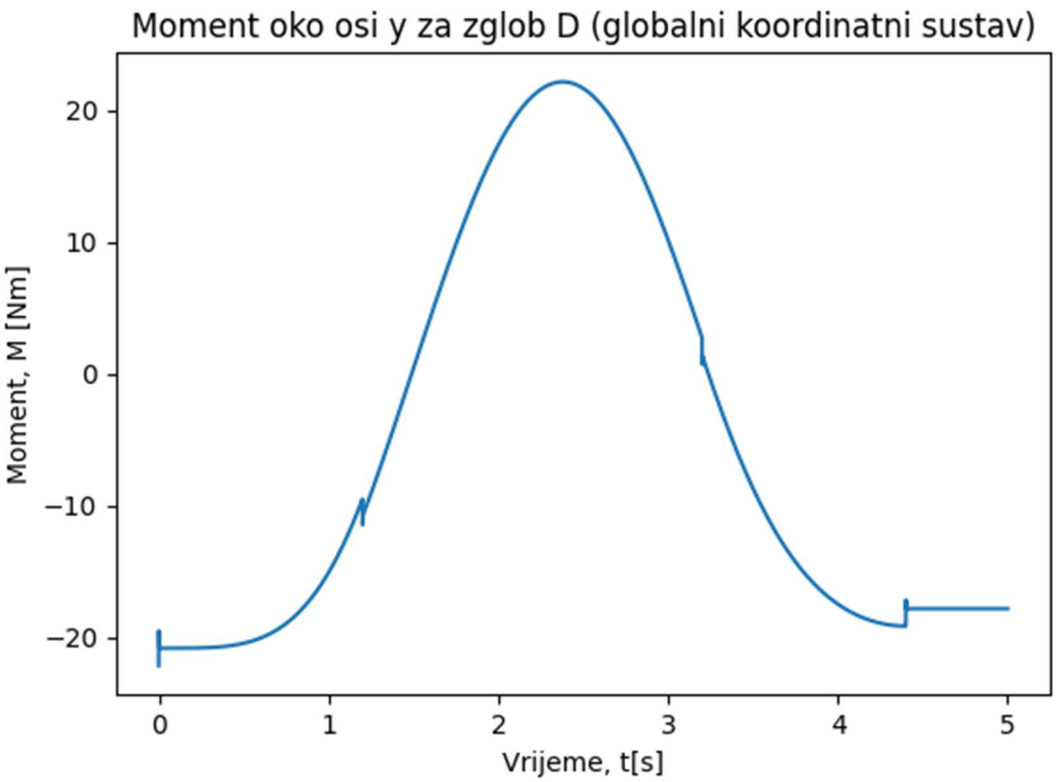

Moment oko osi x za zglob C (globalni koordinatni sustav)
Moment, M [Nm]
30
20
10
0
−10
−20
−30
0
1
2
3
4
5
Vrijeme, t[s]


Moment oko osi x za zglob D (globalni koordinatni sustav)
Moment, M [Nm]
20
10
0
−10
−20
0
1
2
3
4
5
Vrijeme, t[s]

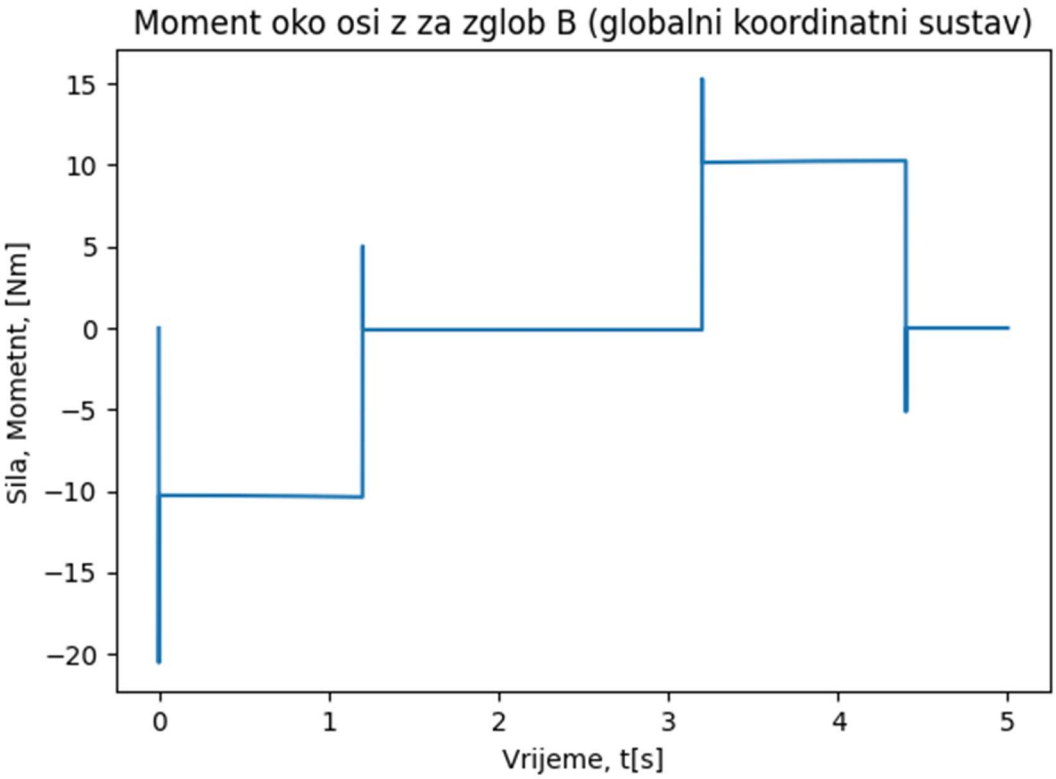

Moment oko osi y za zglob A (globalni koordinatni sustav)
Moment, M [Nm]
100
75
50
25
0
−25
−50
−75
−100
0
1
2
3
4
5
Vrijeme, t[s]

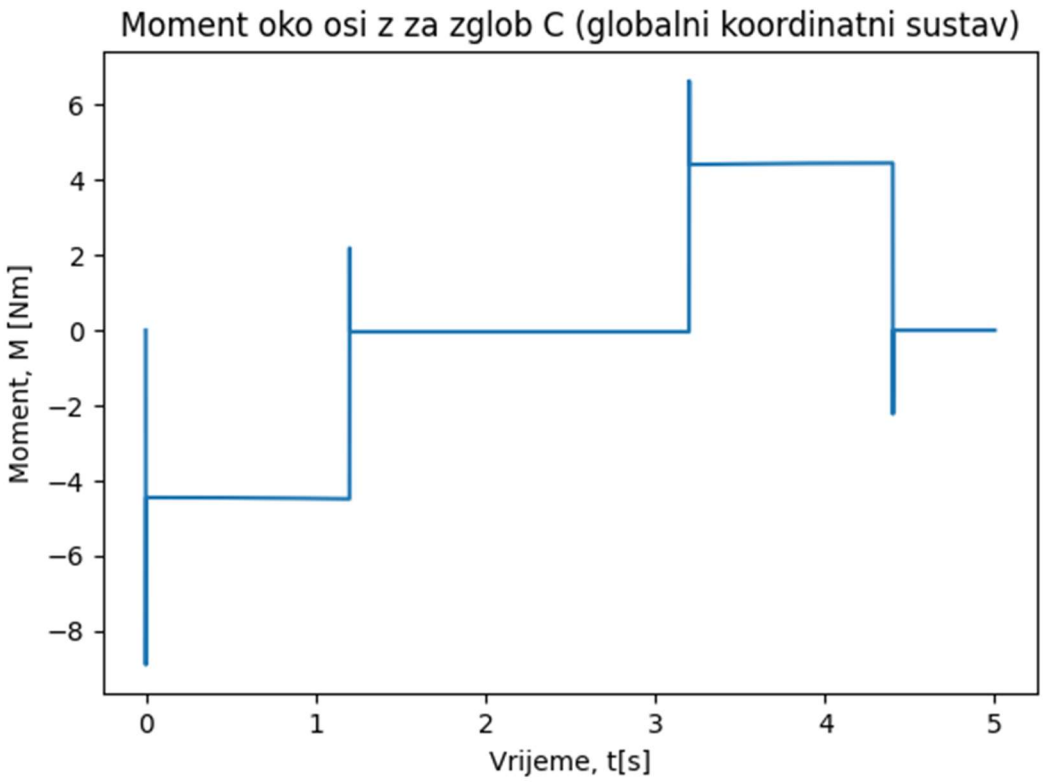

Moment oko osi y za zglob B (globalni koordinatni sustav)
Moment, M [Nm]
100
75
50
25
0
−25
−50
−75
−100
0
1
2
3
4
5
Vrijeme, t[s]


Moment oko osi y za zglob C (globalni koordinatni sustav)
Moment, M [Nm]
30
20
10
0
−10
−20
−30
0
1
2
3
4
5
Vrijeme, t[s]

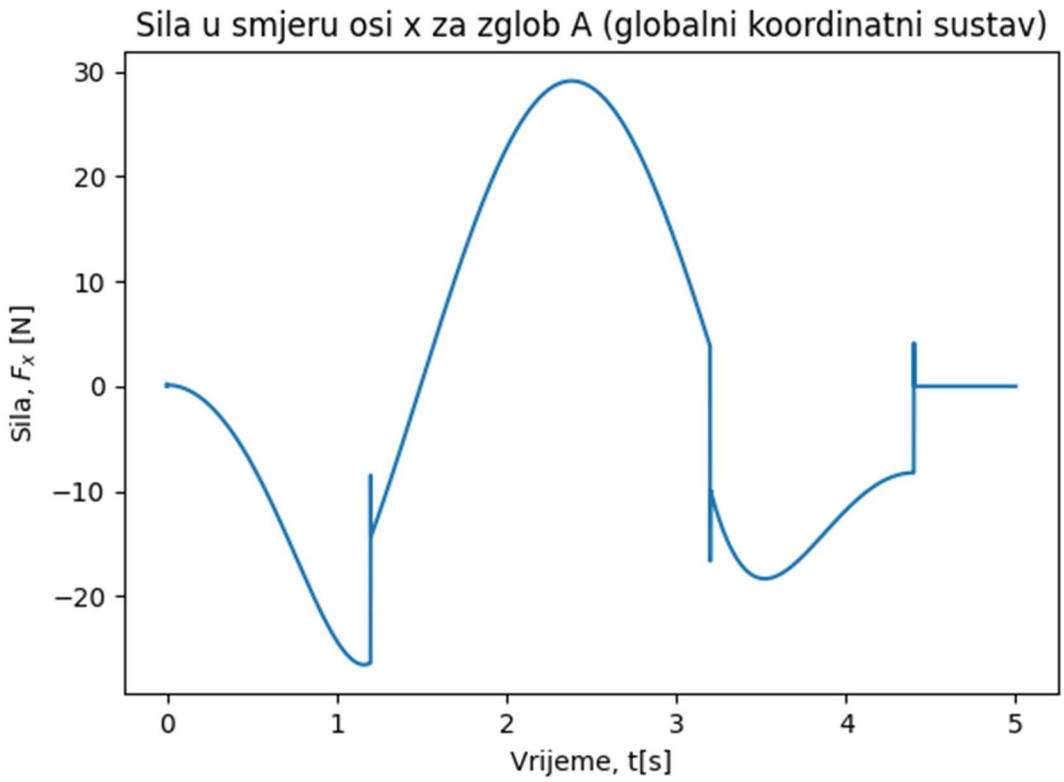

Moment oko osi y za zglob D (globalni koordinatni sustav)
Moment, M [Nm]
20
10
0
−10
−20
0
1
2
3
4
5
Vrijeme, t[s]

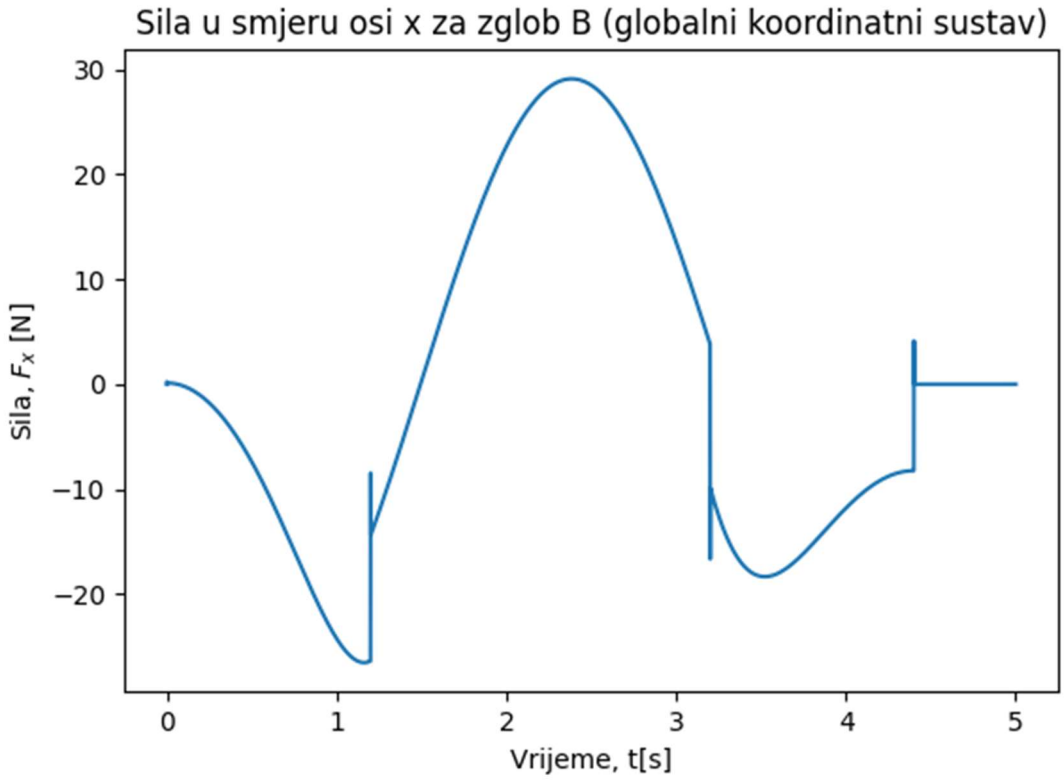

Moment oko osi z za zglob A (globalni koordinatni sustav)
Moment, M [Nm]
15
10
5
0
−5
−10
−15
−20
0
1
2
3
4
5
Vrijeme, t[s]

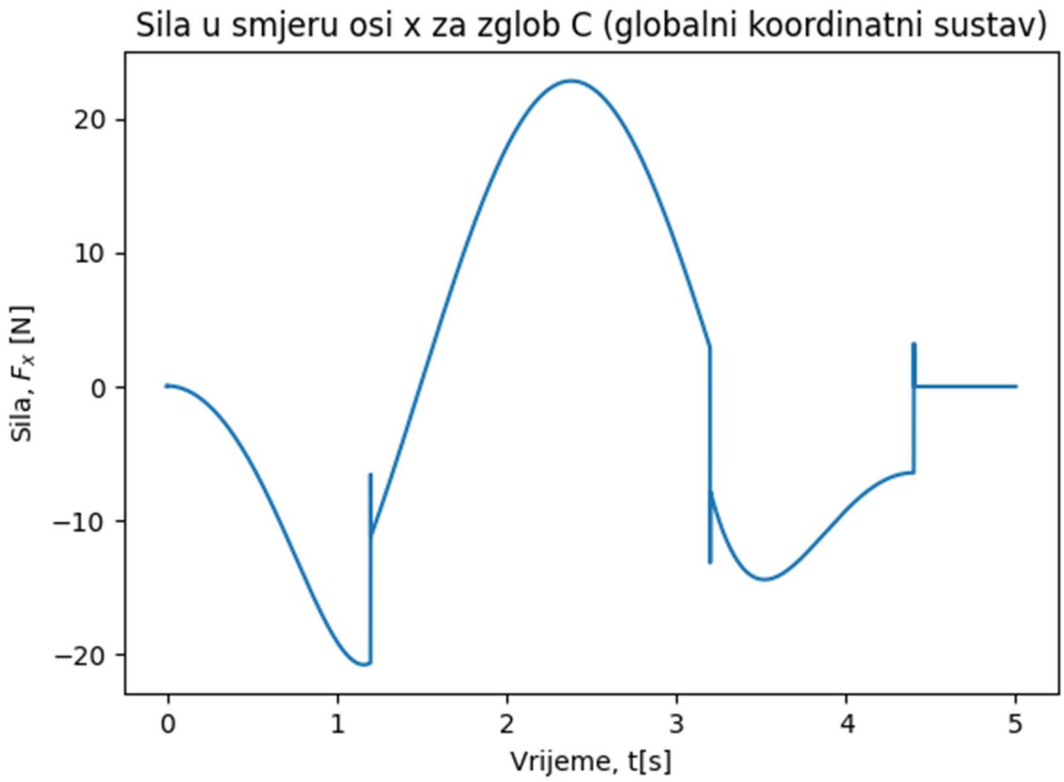

Moment oko osi z za zglob B (globalni koordinatni sustav)
Sila, Mometnt, [Nm]
15
10
5
0
−5
−10
−15
−20
0
1
2
3
4
5
Vrijeme, t[s]

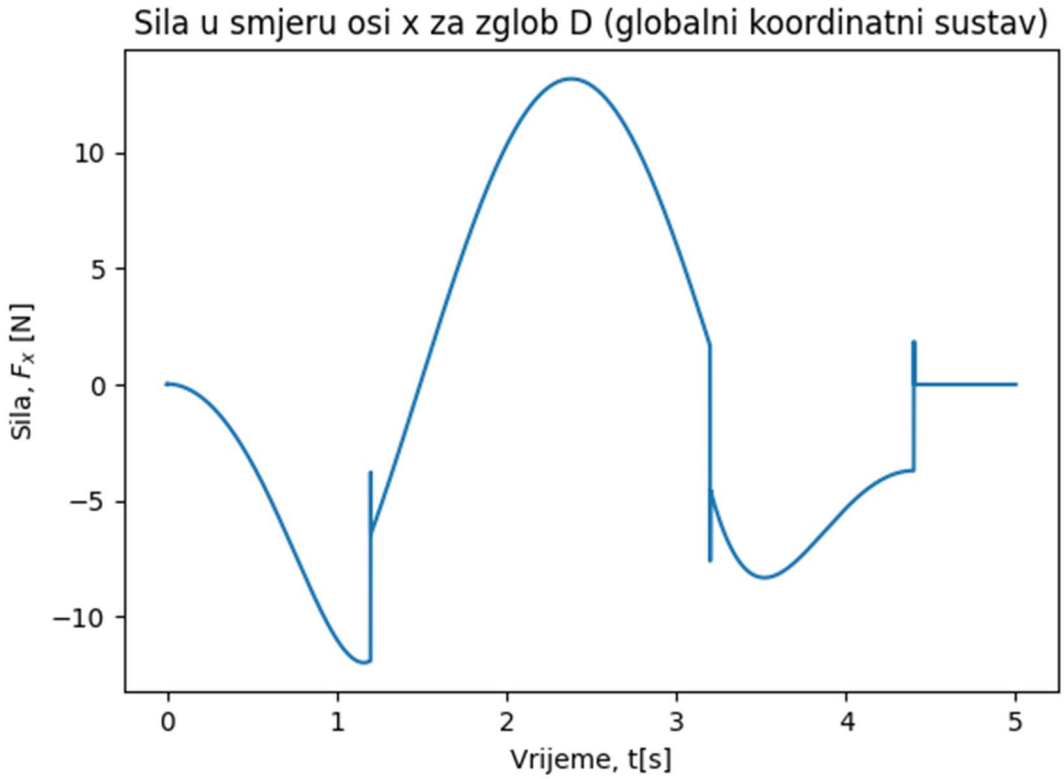

Moment oko osi z za zglob C (globalni koordinatni sustav)
Moment, M [Nm]
6
4
2
0
−2
−4
−6
−8
0
1
2
3
4
5
Vrijeme, t[s]

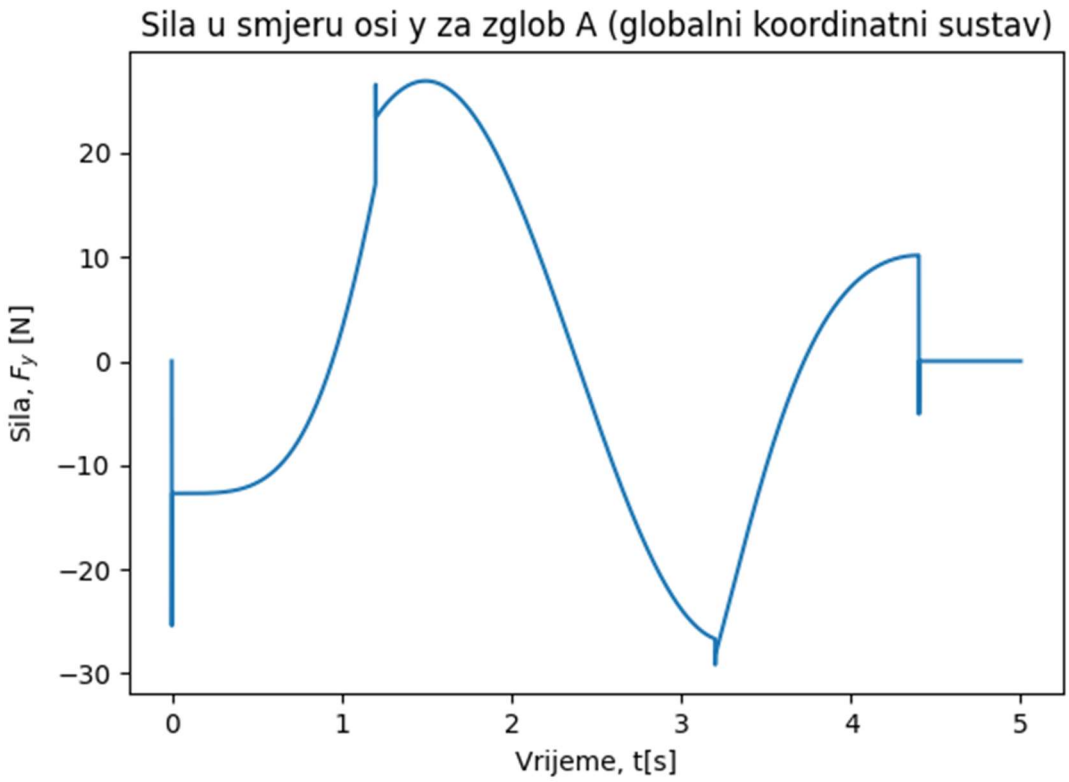

Moment oko osi z za zglob D (globalni koordinatni sustav)
Moment, M [Nm]
4
2
0
−2
−4
−6
0
1
2
3
4
5
Vrijeme, t[s]

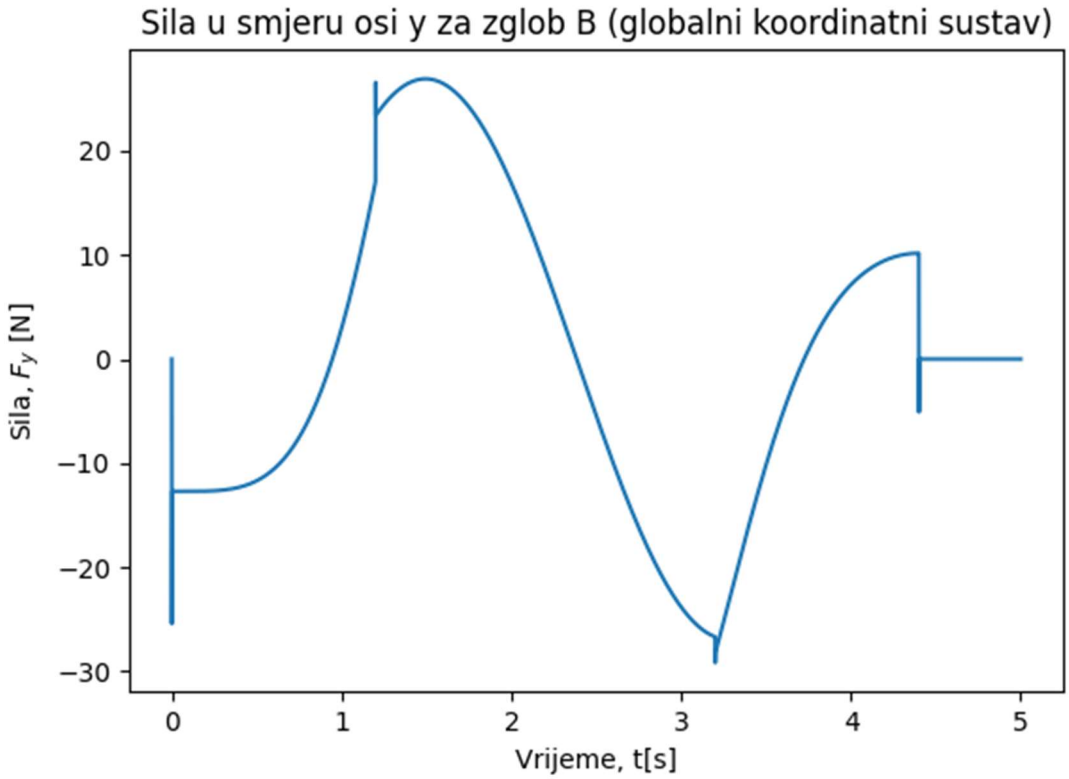

Sila u smjeru osi x za zglob A (globalni koordinatni sustav)
Sila, F_x [N]
30
20
10
0
−10
−20
0
1
2
3
4
5
Vrijeme, t[s]

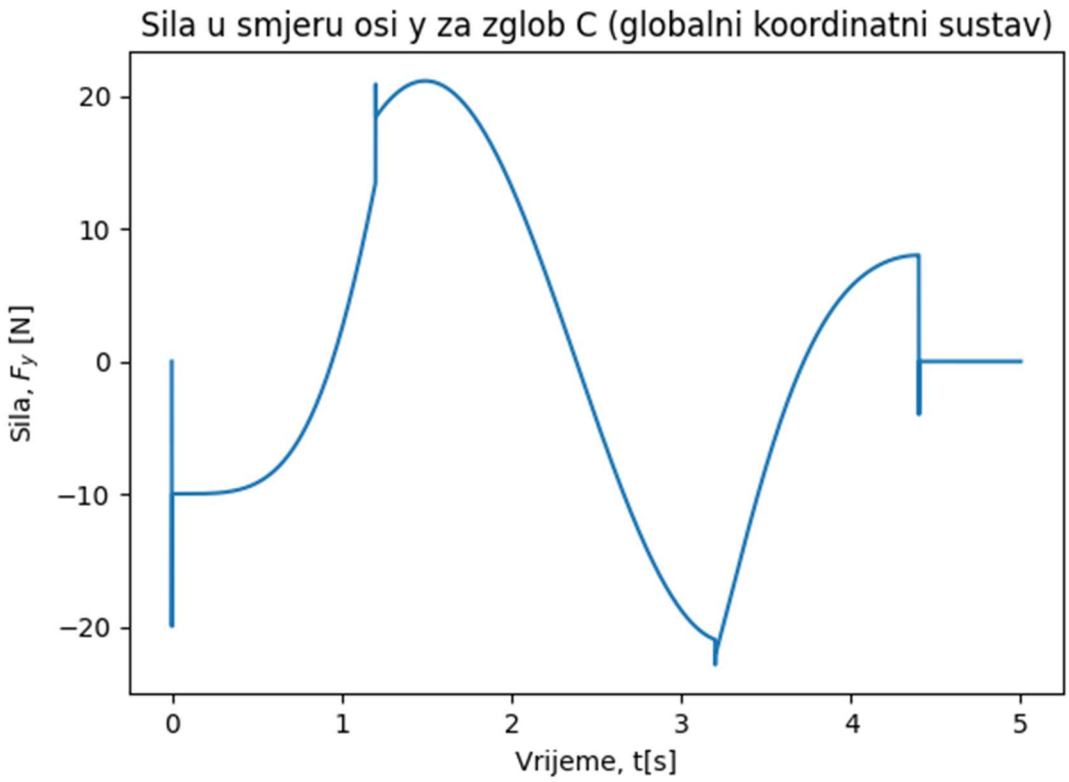

Sila u smjeru osi x za zglob B (globalni koordinatni sustav)
Sila, F_x [N]
30
20
10
0
−10
−20
0
1
2
3
4
5
Vrijeme, t[s]

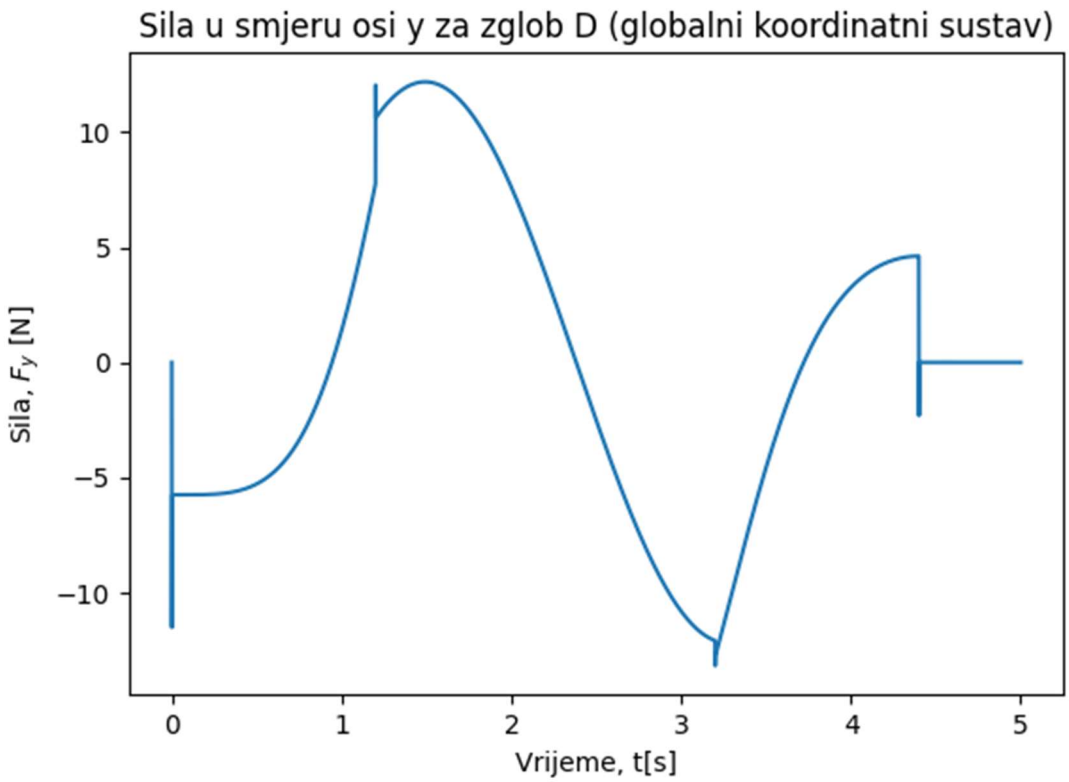

Sila u smjeru osi x za zglob C (globalni koordinatni sustav)
Sila, F_x [N]
20
10
0
−10
−20
0
1
2
3
4
5
Vrijeme, t[s]

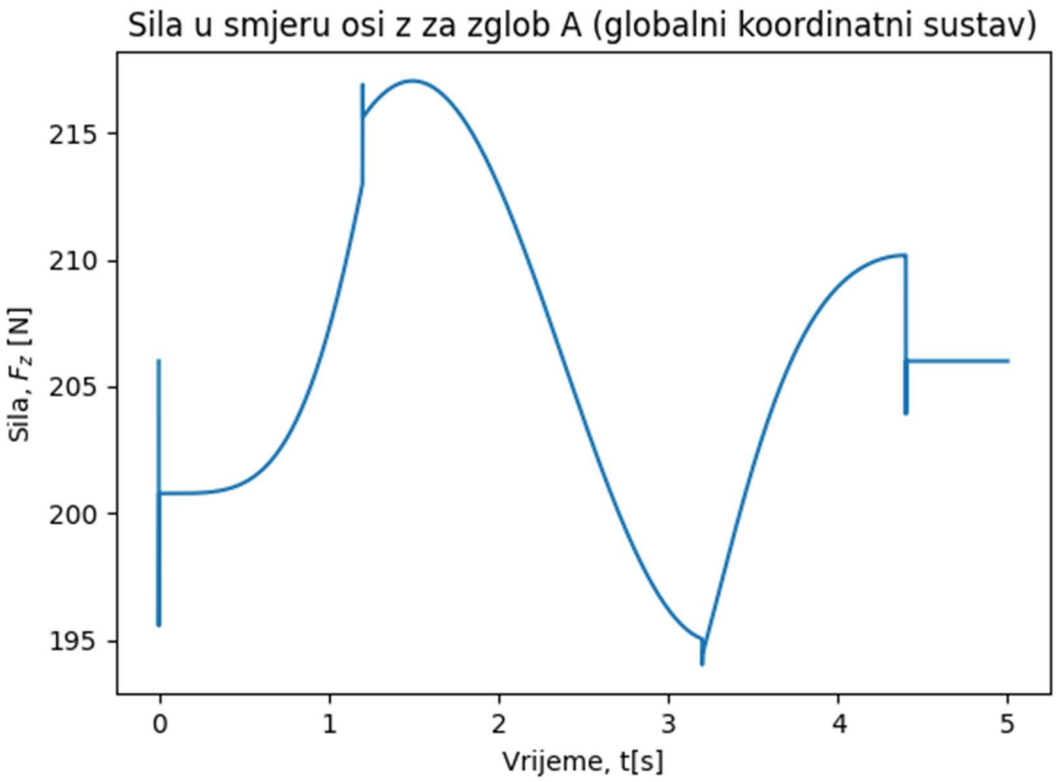

Sila u smjeru osi x za zglob D (globalni koordinatni sustav)
Sila, F_x [N]
10
5
0
−5
−10
0
1
2
3
4
5
Vrijeme, t[s]

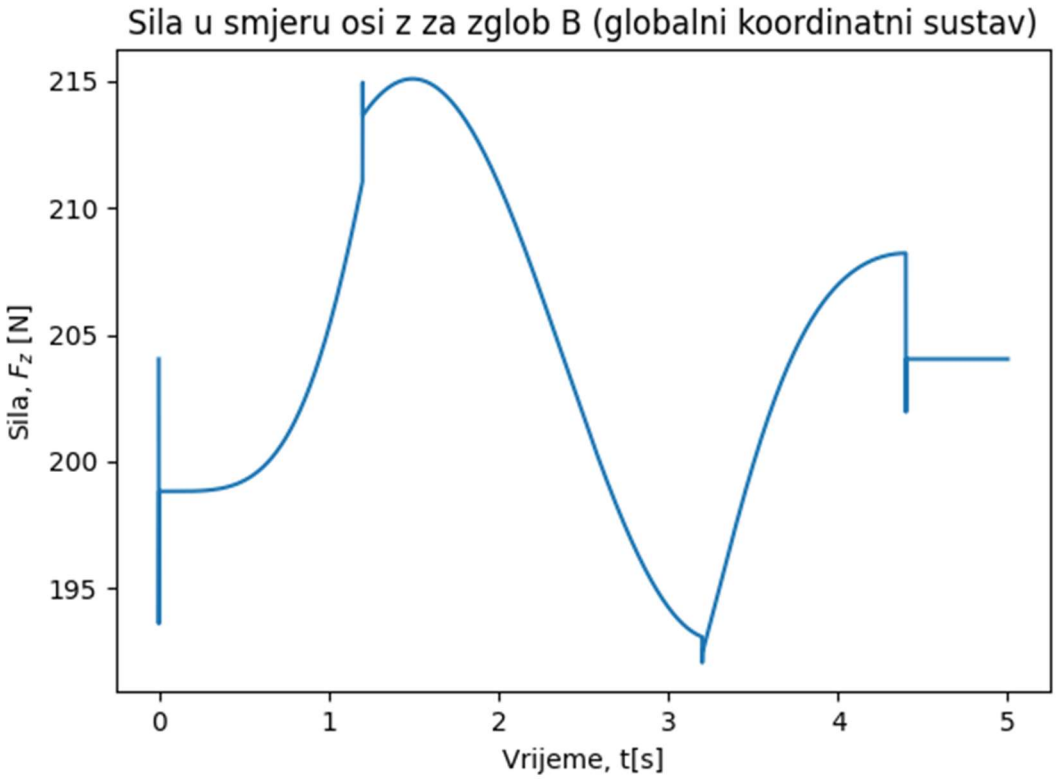

Sila u smjeru osi y za zglob A (globalni koordinatni sustav)
Sila, F_y [N]
20
10
0
−10
−20
−30
0
1
2
3
4
5
Vrijeme, t[s]

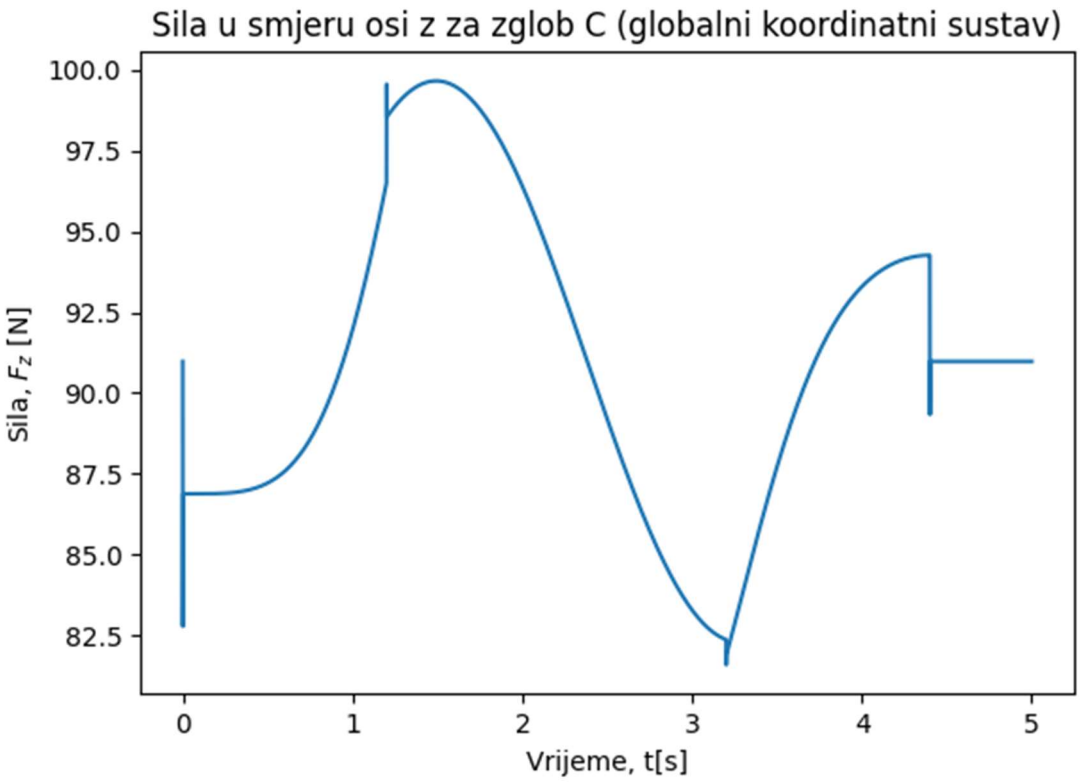

Sila u smjeru osi y za zglob B (globalni koordinatni sustav)
Sila, F_y [N]
20
10
0
−10
−20
−30
0
1
2
3
4
5
Vrijeme, t[s]

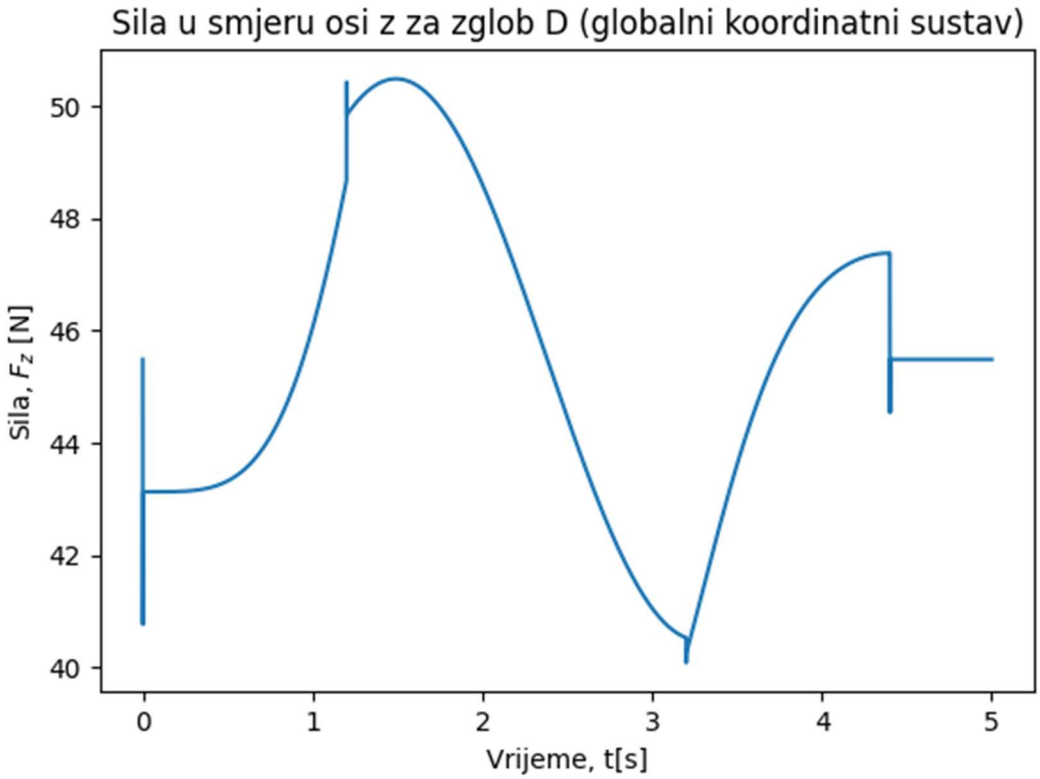

Sila u smjeru osi y za zglob C (globalni koordinatni sustav)
Sila, F_y [N]
20
10
0
−10
−20
0
1
2
3
4
5
Vrijeme, t[s]


Sila u smjeru osi y za zglob D (globalni koordinatni sustav)
Sila, F_y [N]
10
5
0
−5
−10
0
1
2
3
4
5
Vrijeme, t[s]

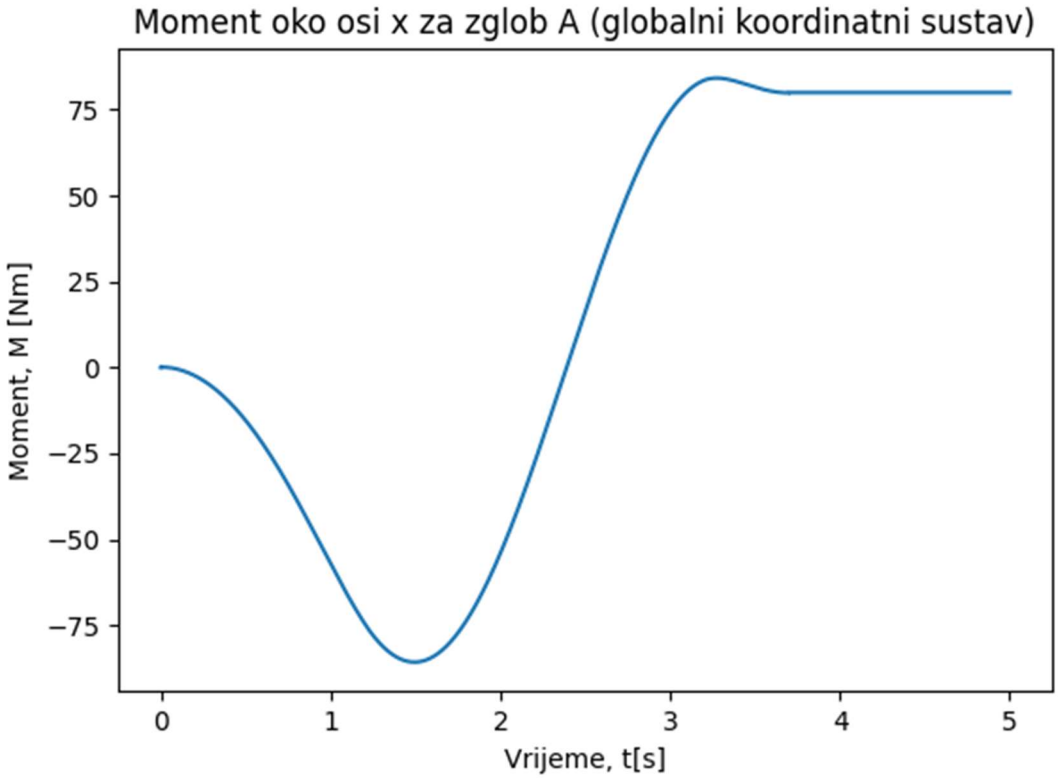

Sila u smjeru osi z za zglob A (globalni koordinatni sustav)
Sila, F_z [N]
215
210
205
200
195
0
1
2
3
4
5
Vrijeme, t[s]

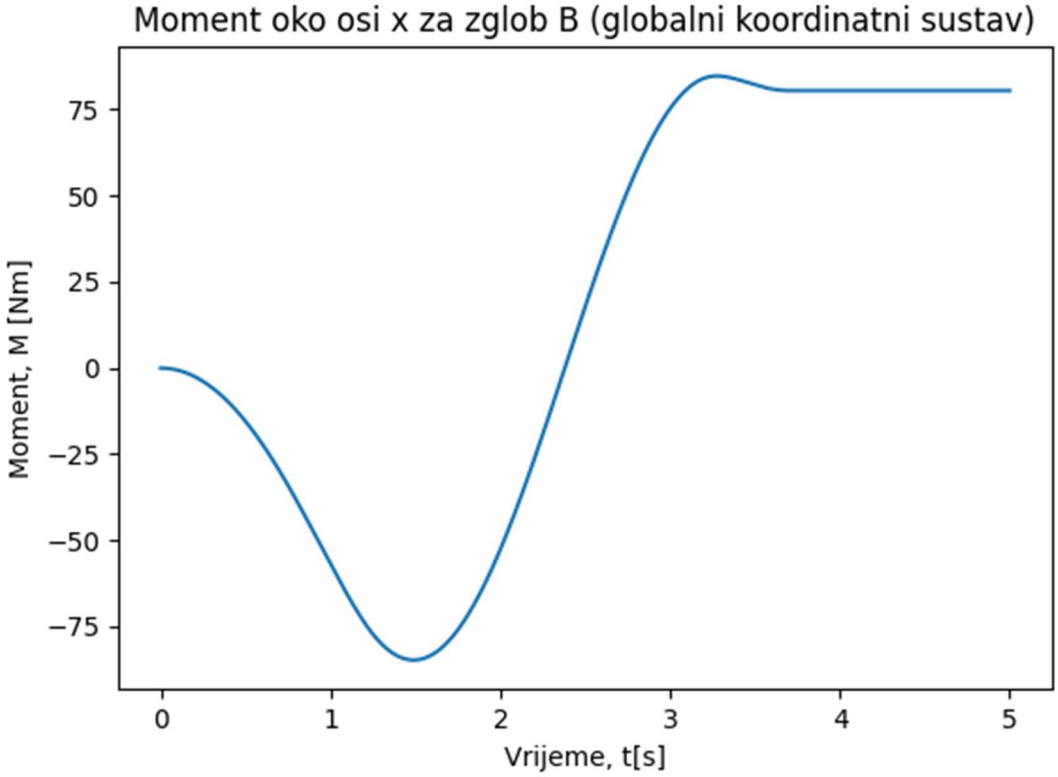

Sila u smjeru osi z za zglob B (globalni koordinatni sustav)
Sila, F_z [N]
215
210
205
200
195
0
1
2
3
4
5
Vrijeme, t[s]

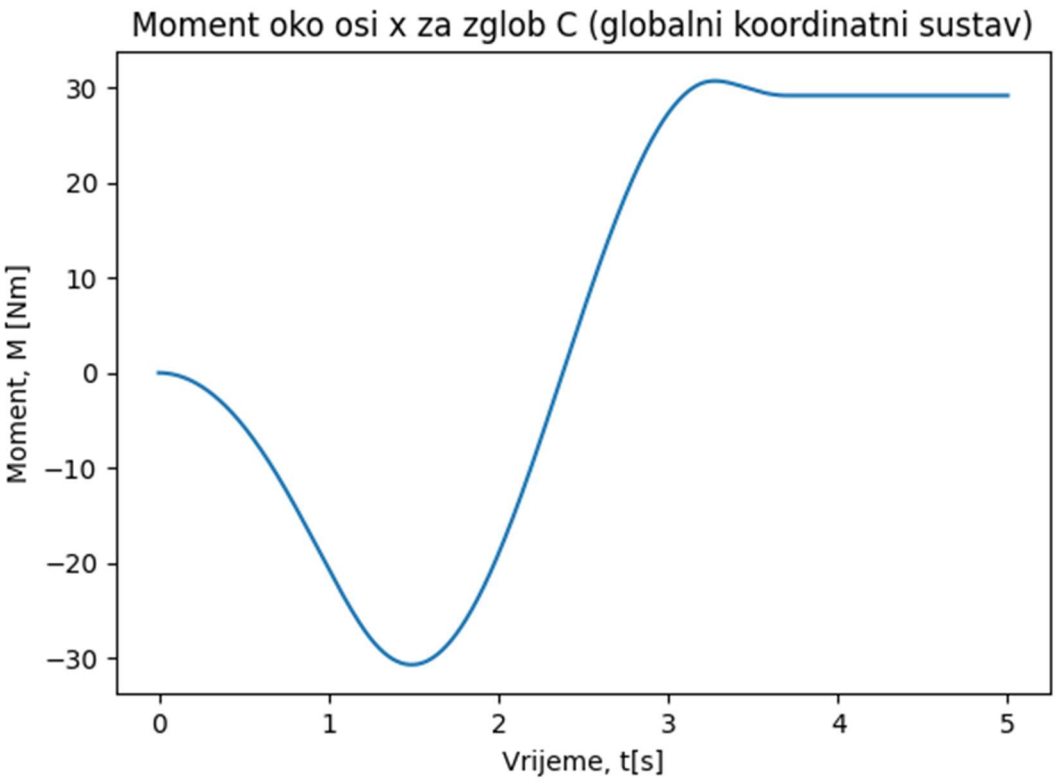

Sila u smjeru osi z za zglob C (globalni koordinatni sustav)
Sila, F_z [N]
100.0
97.5
95.0
92.5
90.0
87.5
85.0
82.5
0
1
2
3
4
5
Vrijeme, t[s]

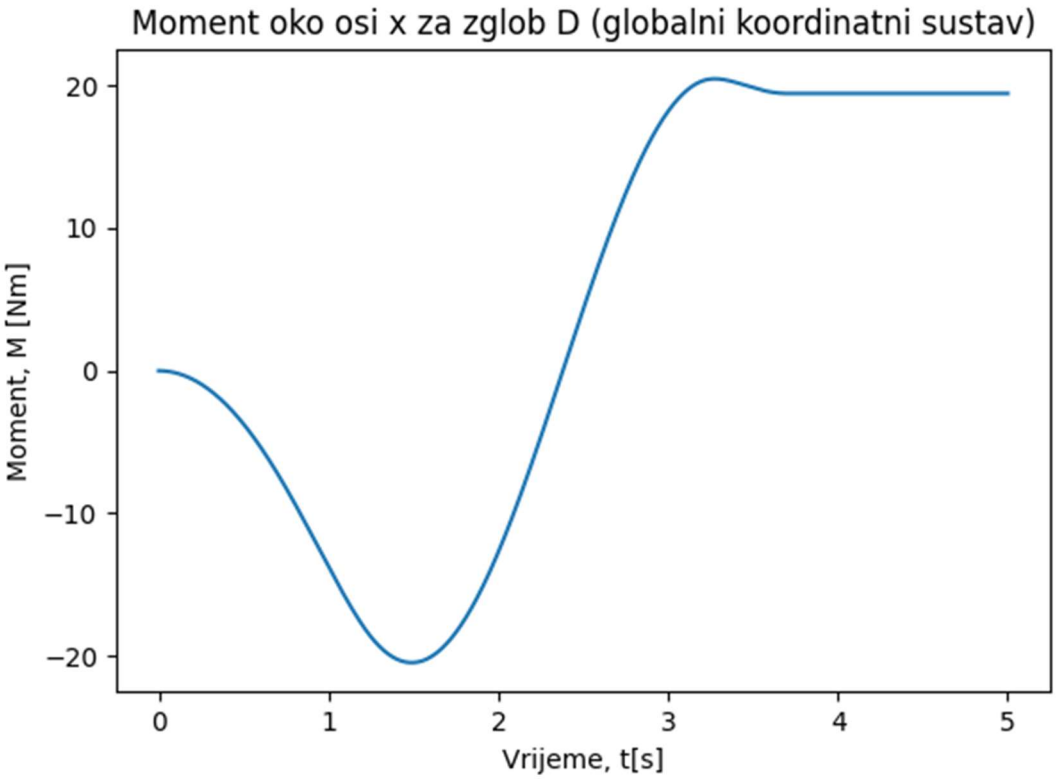

Sila u smjeru osi z za zglob D (globalni koordinatni sustav)
Sila, F_z [N]
50
48
46
44
42
40
0
1
2
3
4
5
Vrijeme, t[s]

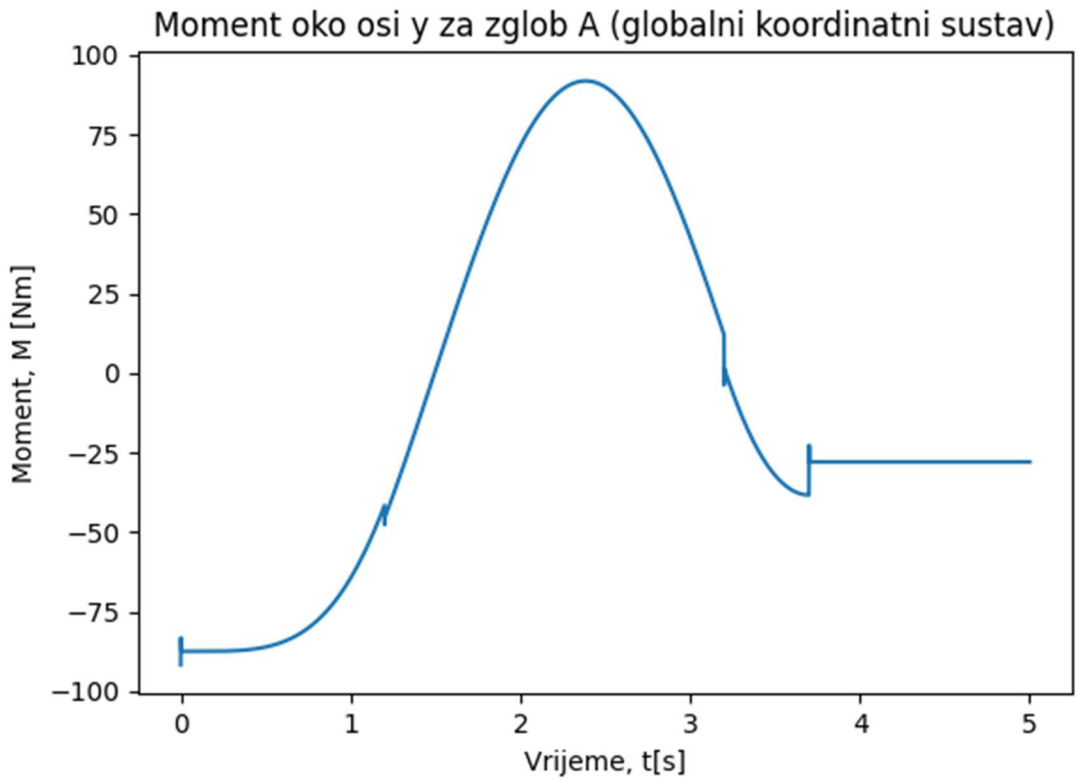
Opterećenja prilikom kočenja

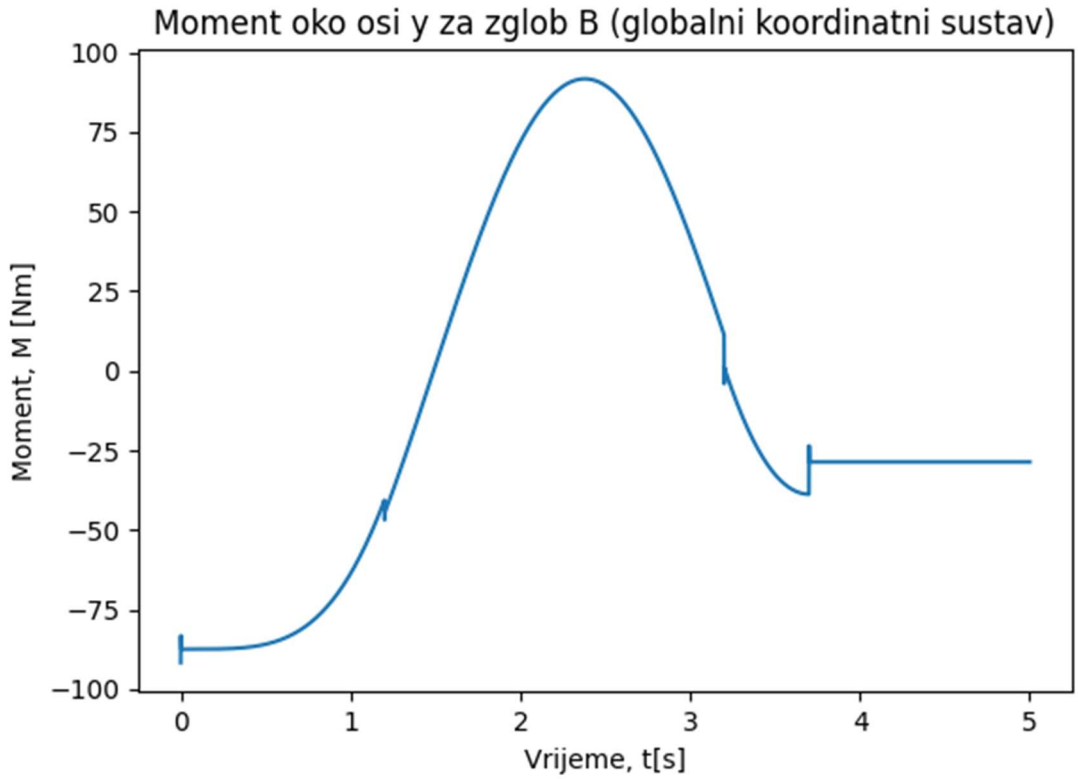


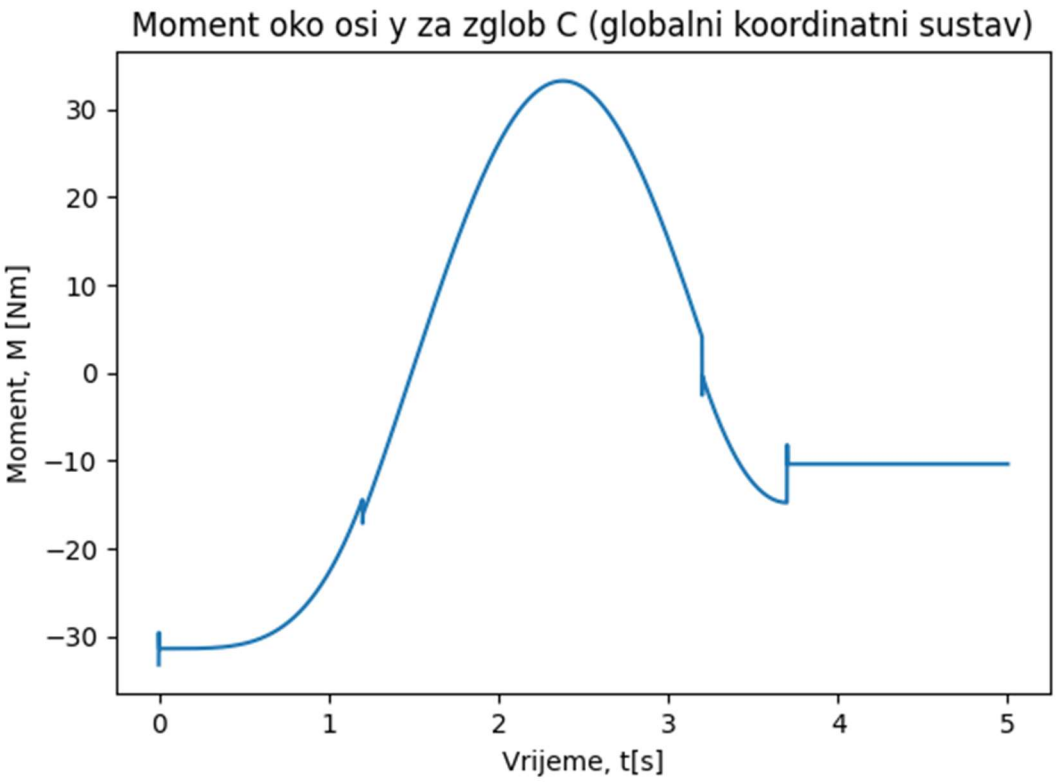


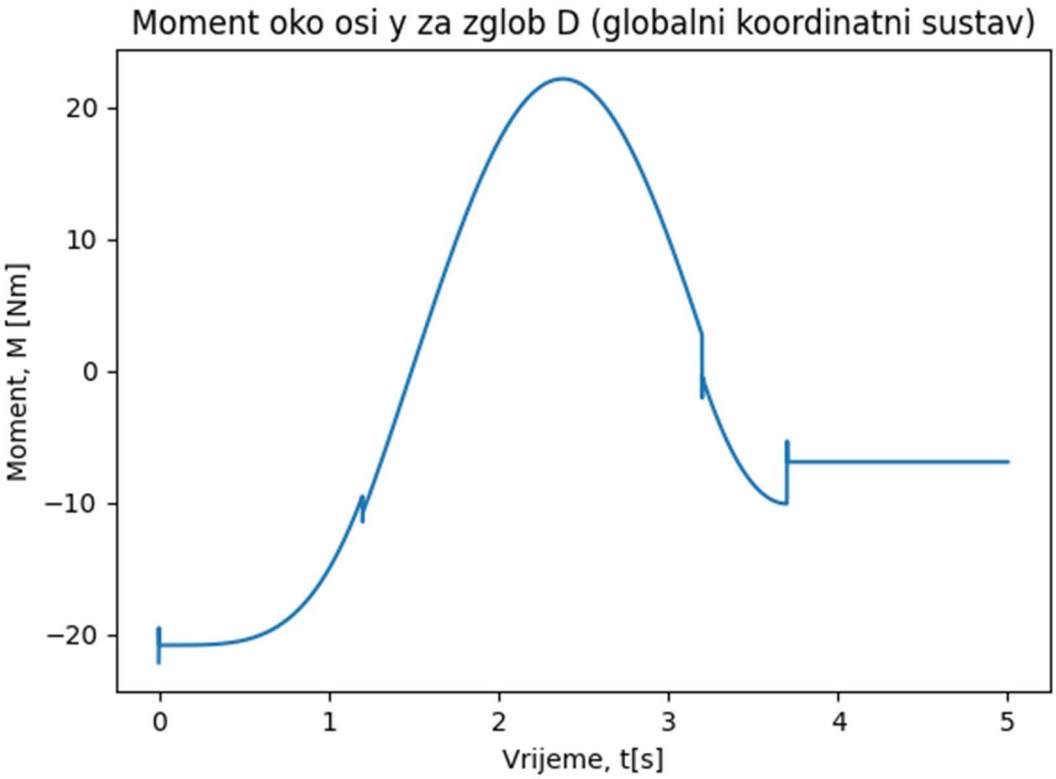

Moment oko osi x za zglob C (globalni koordinatni sustav)
Moment, M [Nm]
30
20
10
0
−10
−20
−30
0
1
2
3
4
5
Vrijeme, t[s]

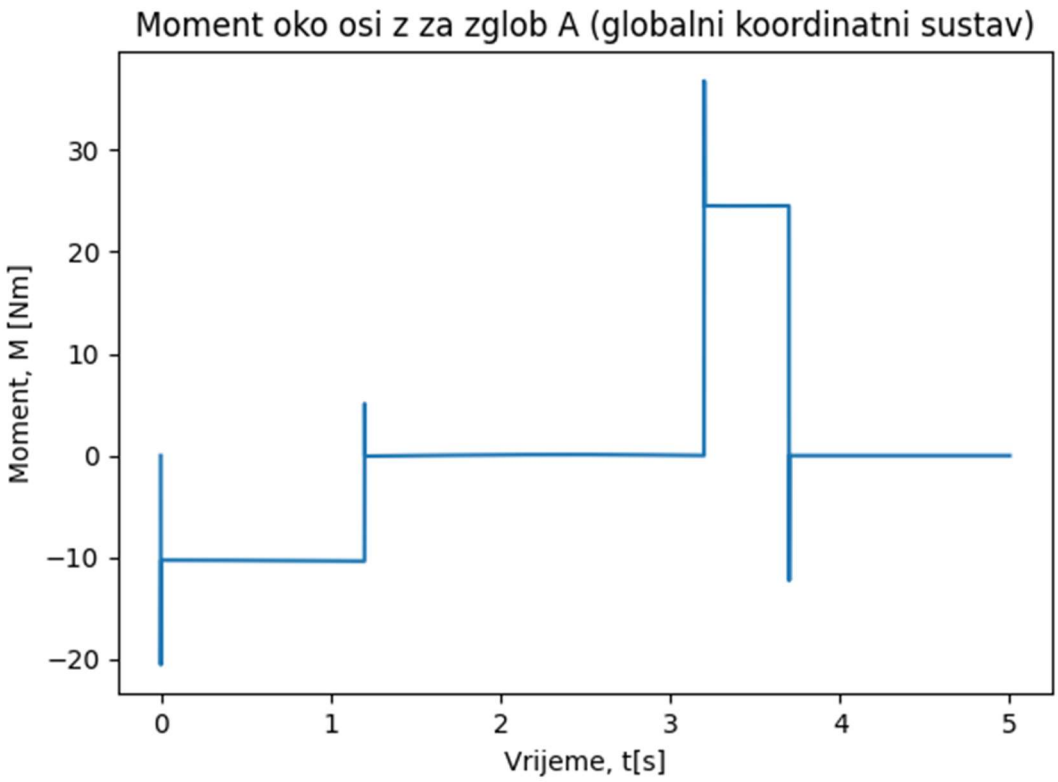

Moment oko osi x za zglob D (globalni koordinatni sustav)
Moment, M [Nm]
20
10
0
−10
−20
0
1
2
3
4
5
Vrijeme, t[s]

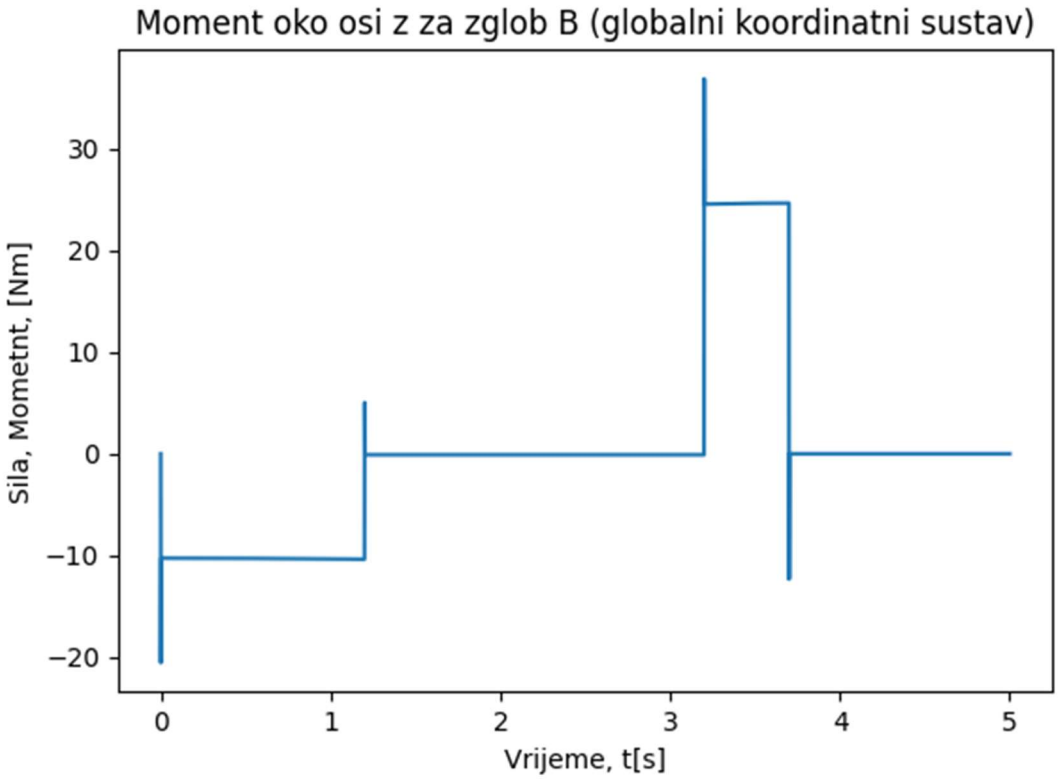

Moment oko osi y za zglob A (globalni koordinatni sustav)
Moment, M [Nm]
100
75
50
25
0
−25
−50
−75
−100
0
1
2
3
4
5
Vrijeme, t[s]

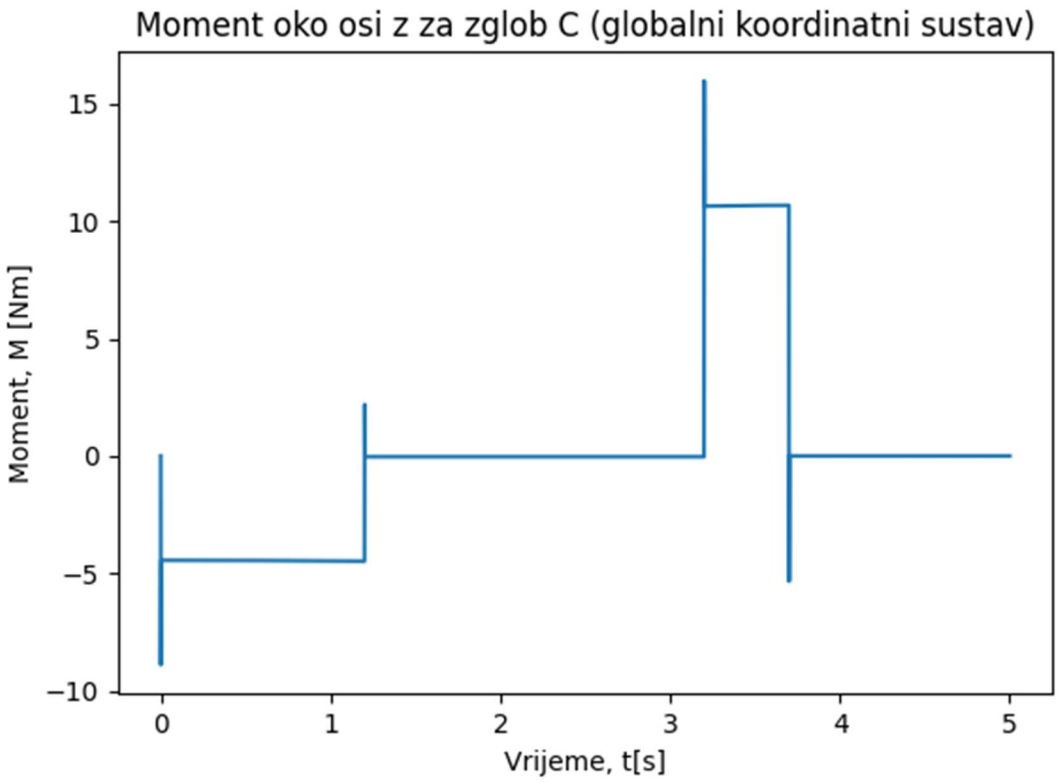

Moment oko osi y za zglob B (globalni koordinatni sustav)
Moment, M [Nm]
100
75
50
25
0
−25
−50
−75
−100
0
1
2
3
4
5
Vrijeme, t[s]

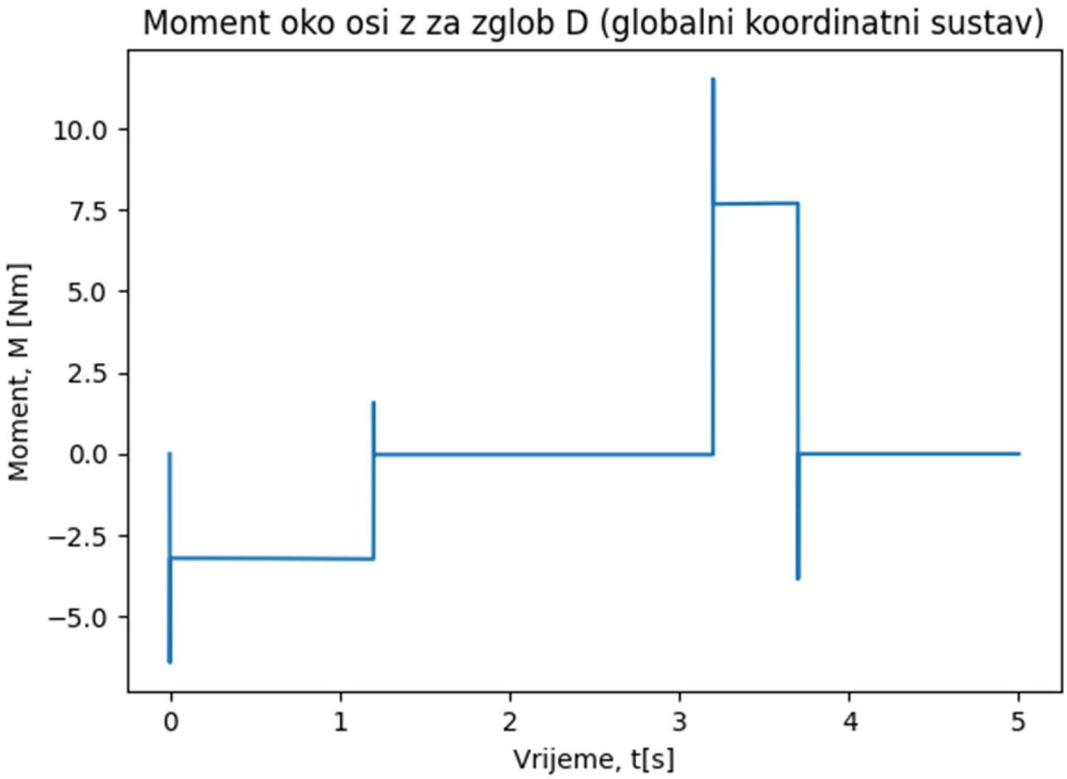

Moment oko osi y za zglob C (globalni koordinatni sustav)
Moment, M [Nm]
30
20
10
0
−10
−20
−30
0
1
2
3
4
5
Vrijeme, t[s]

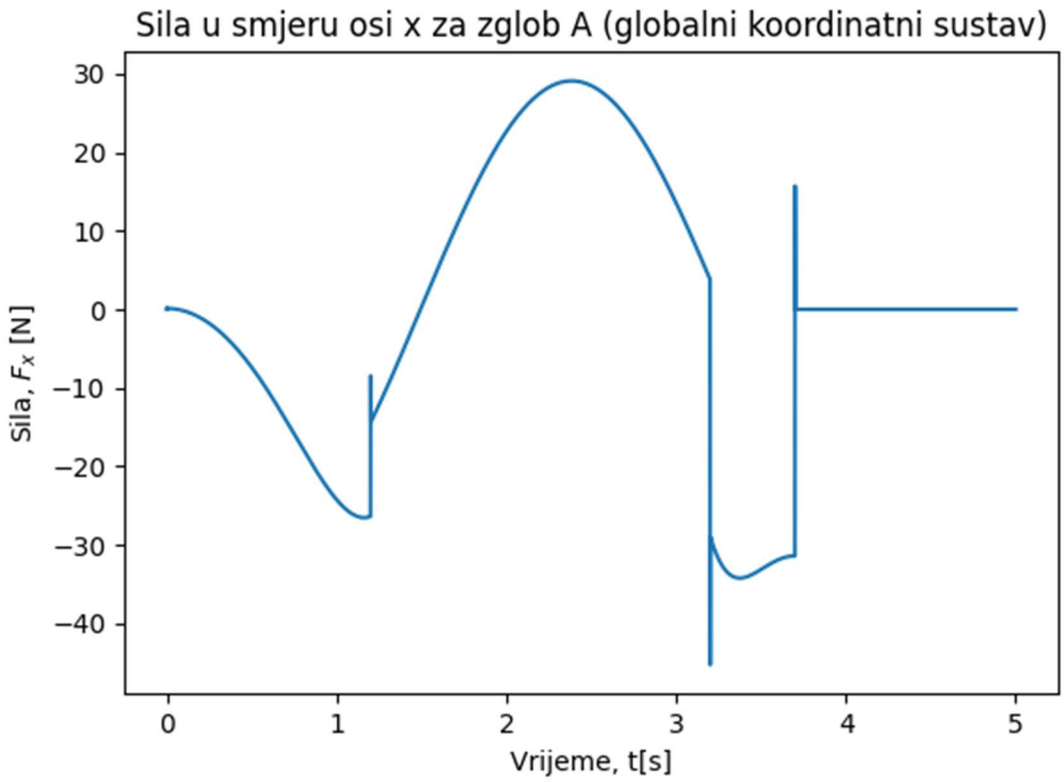

Moment oko osi y za zglob D (globalni koordinatni sustav)
Moment, M [Nm]
20
10
0
−10
−20
0
1
2
3
4
5
Vrijeme, t[s]

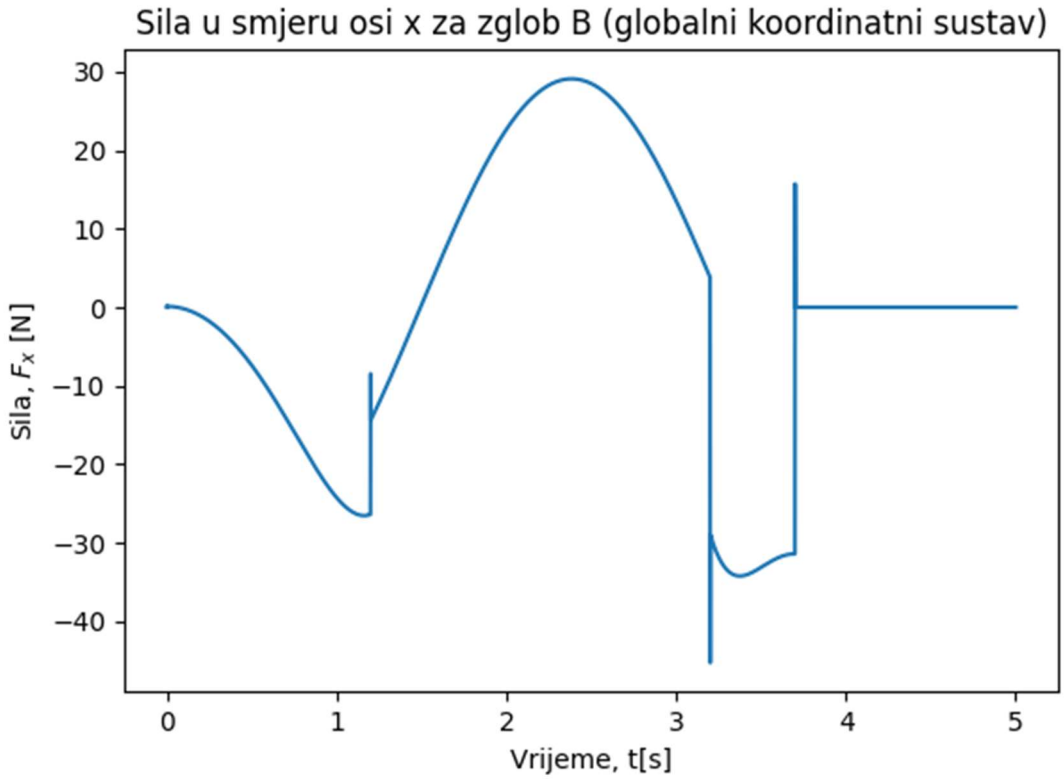

Moment oko osi z za zglob A (globalni koordinatni sustav)
Moment, M [Nm]
30
20
10
0
−10
−20
0
1
2
3
4
5
Vrijeme, t[s]

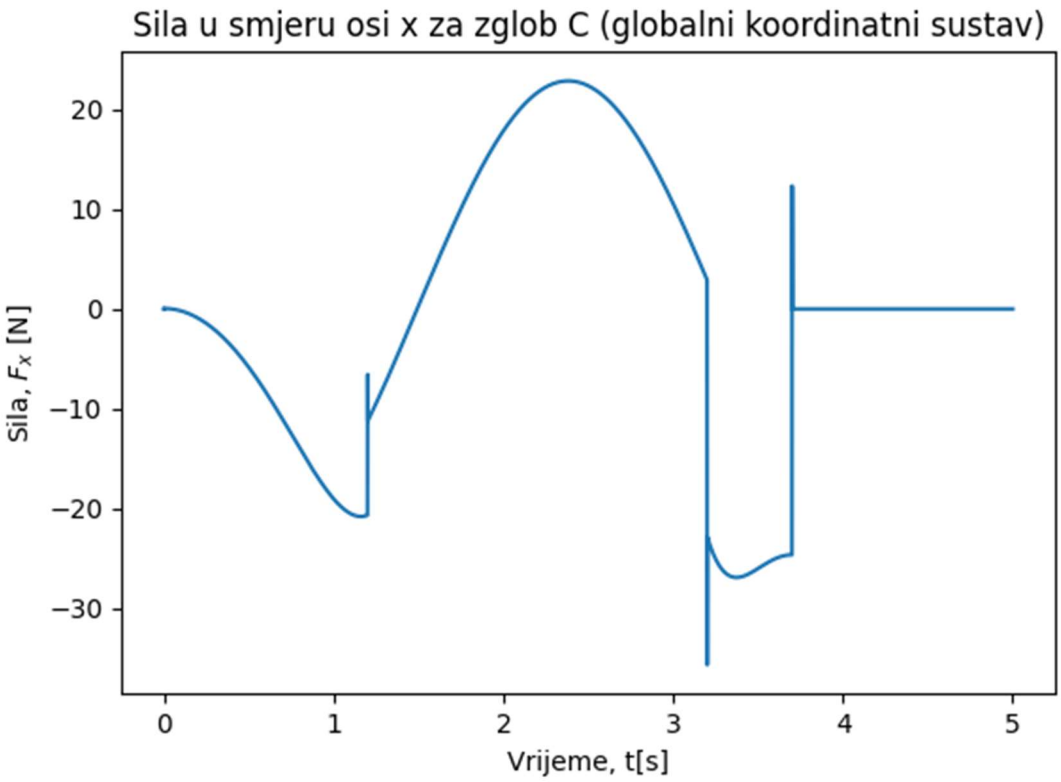

Moment oko osi z za zglob B (globalni koordinatni sustav)
Sila, Mometnt, [Nm]
30
20
10
0
−10
−20
0
1
2
3
4
5
Vrijeme, t[s]

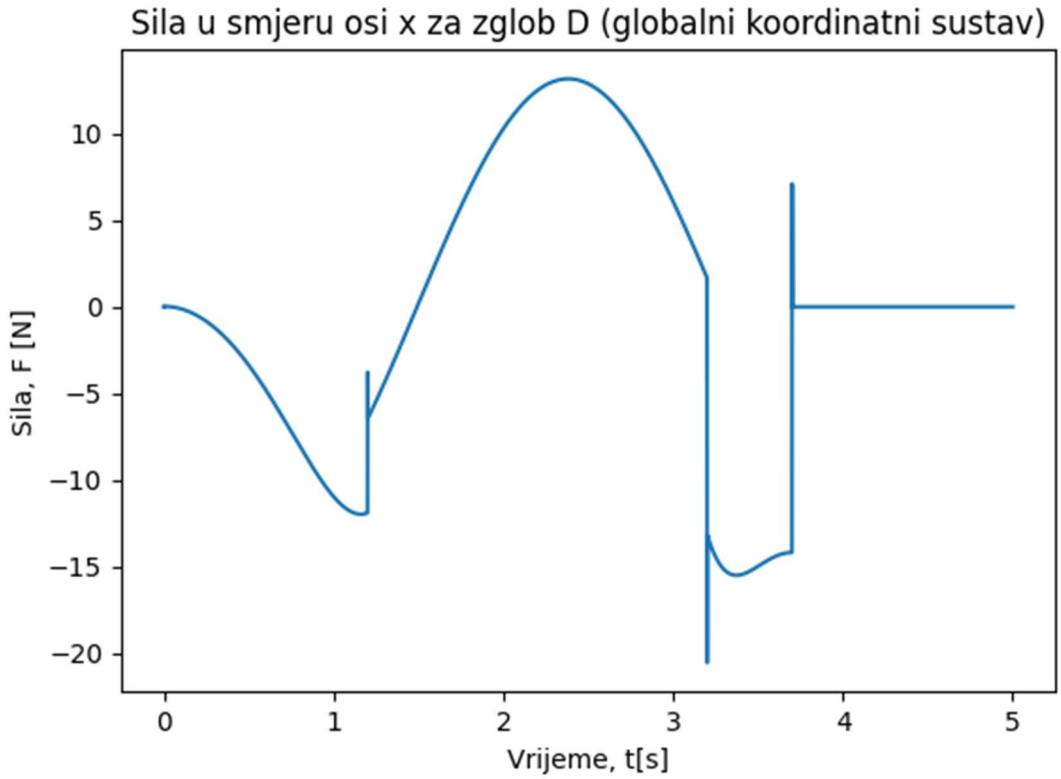

Moment oko osi z za zglob C (globalni koordinatni sustav)
Moment, M [Nm]
15
10
5
0
−5
−10
0
1
2
3
4
5
Vrijeme, t[s]

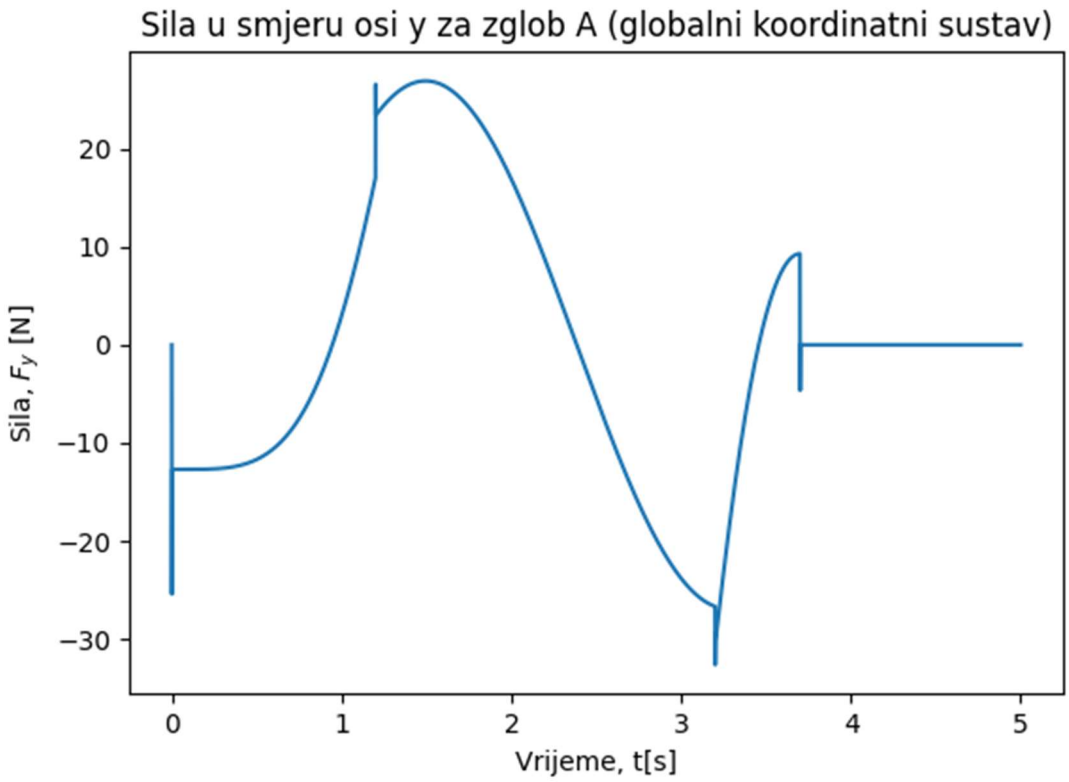

Moment oko osi z za zglob D (globalni koordinatni sustav)
Moment, M [Nm]
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
0
1
2
3
4
5
Vrijeme, t[s]

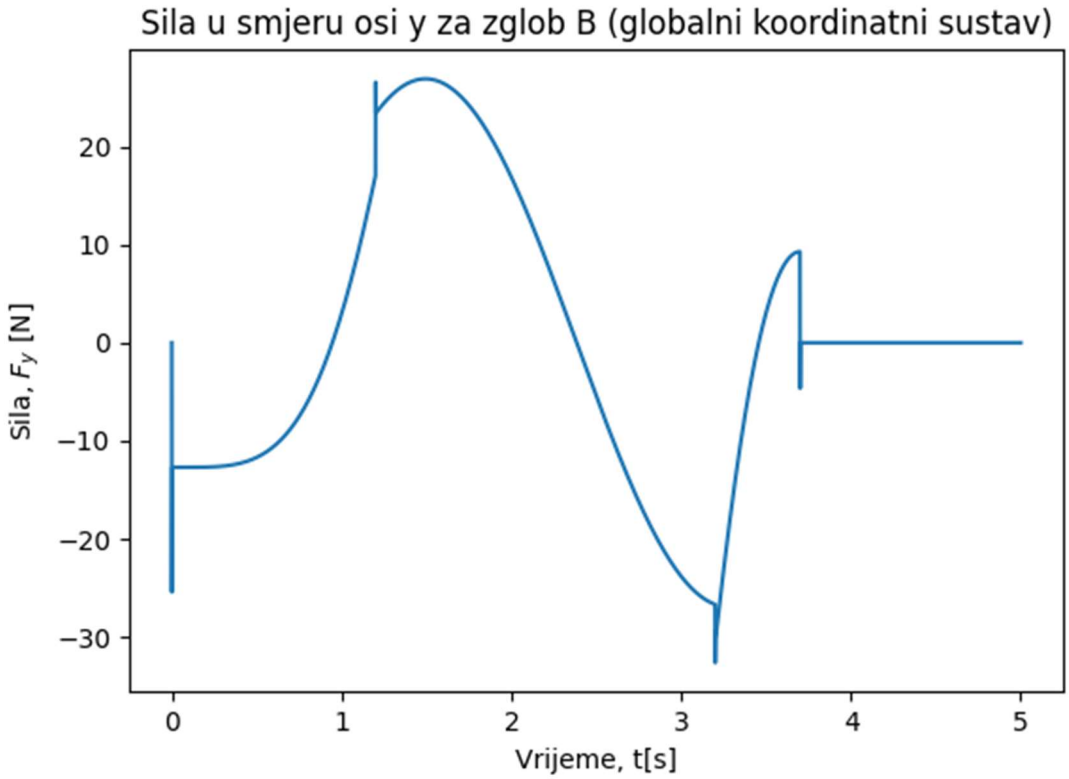

Sila u smjeru osi x za zglob A (globalni koordinatni sustav)
Sila, F_x [N]
30
20
10
0
−10
−20
−30
−40
0
1
2
3
4
5
Vrijeme, t[s]

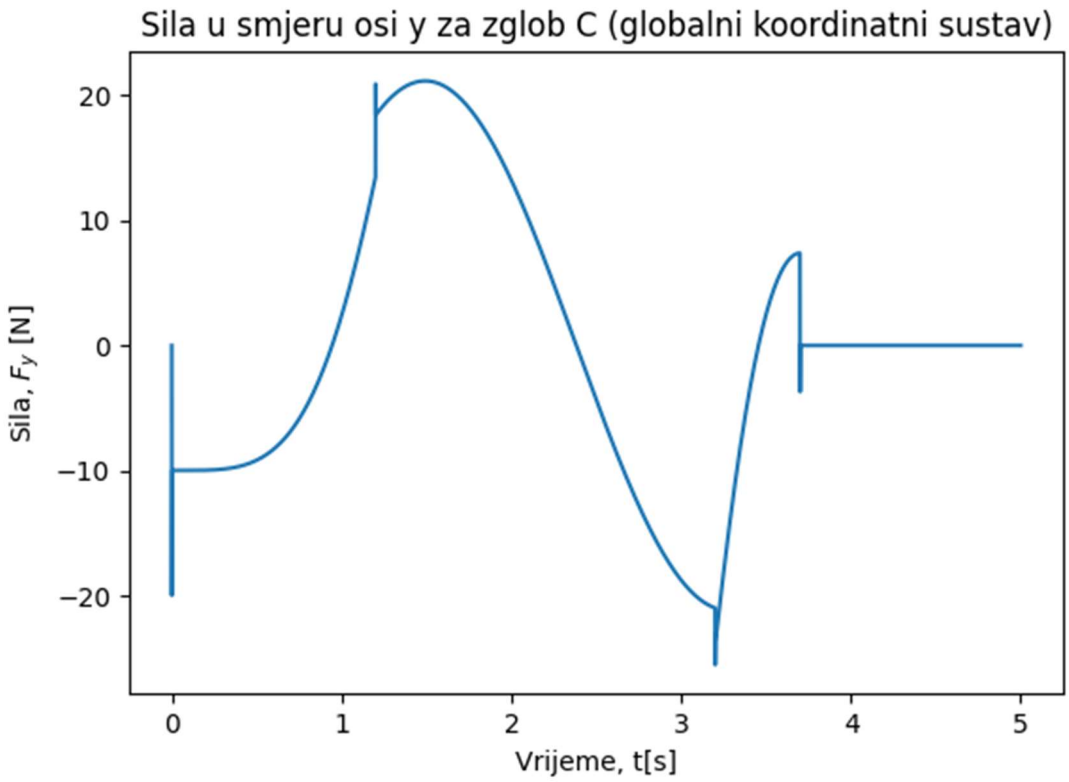

Sila u smjeru osi x za zglob B (globalni koordinatni sustav)
Sila, F_x [N]
30
20
10
0
−10
−20
−30
−40
0
1
2
3
4
5
Vrijeme, t[s]

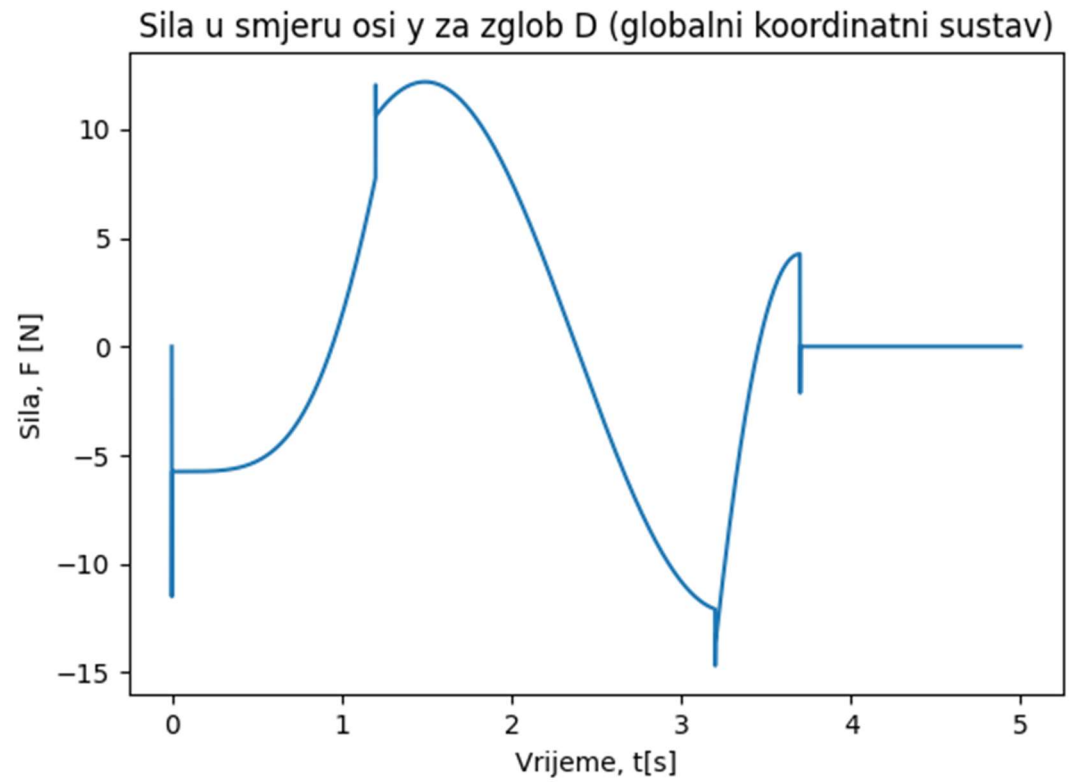

Sila u smjeru osi x za zglob C (globalni koordinatni sustav)
Sila, F_x [N]
20
10
0
−10
−20
−30
0
1
2
3
4
5
Vrijeme, t[s]

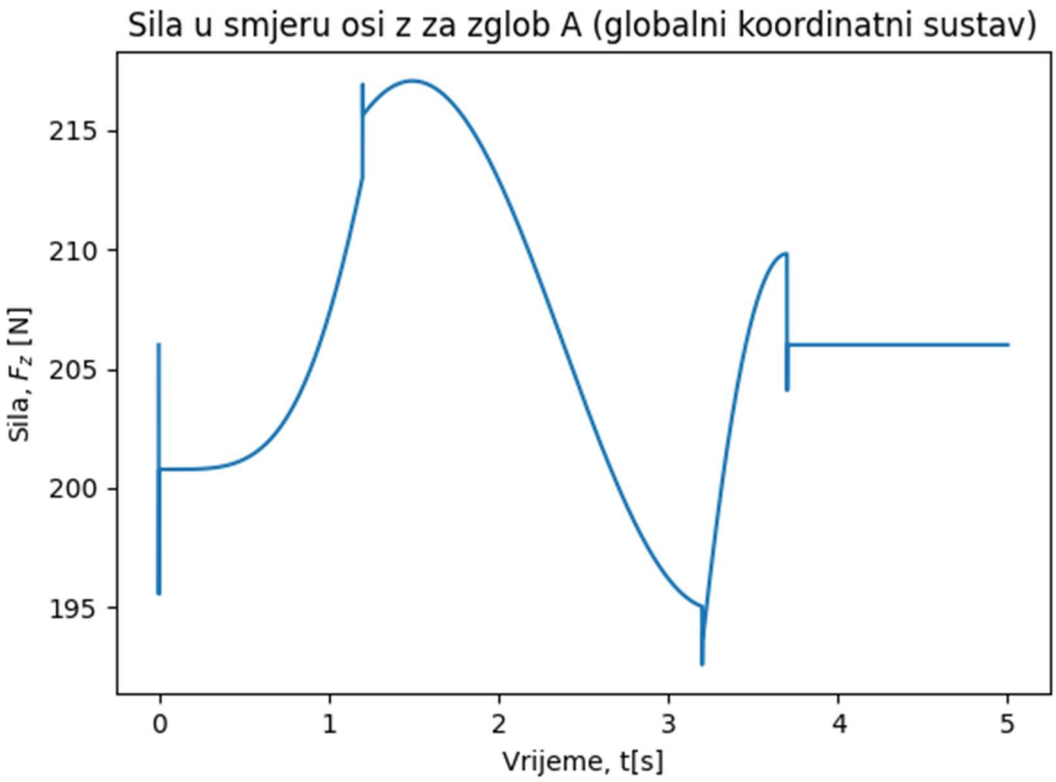

Sila u smjeru osi x za zglob D (globalni koordinatni sustav)
Sila, F [N]
10
5
0
−5
−10
−15
−20
0
1
2
3
4
5
Vrijeme, t[s]

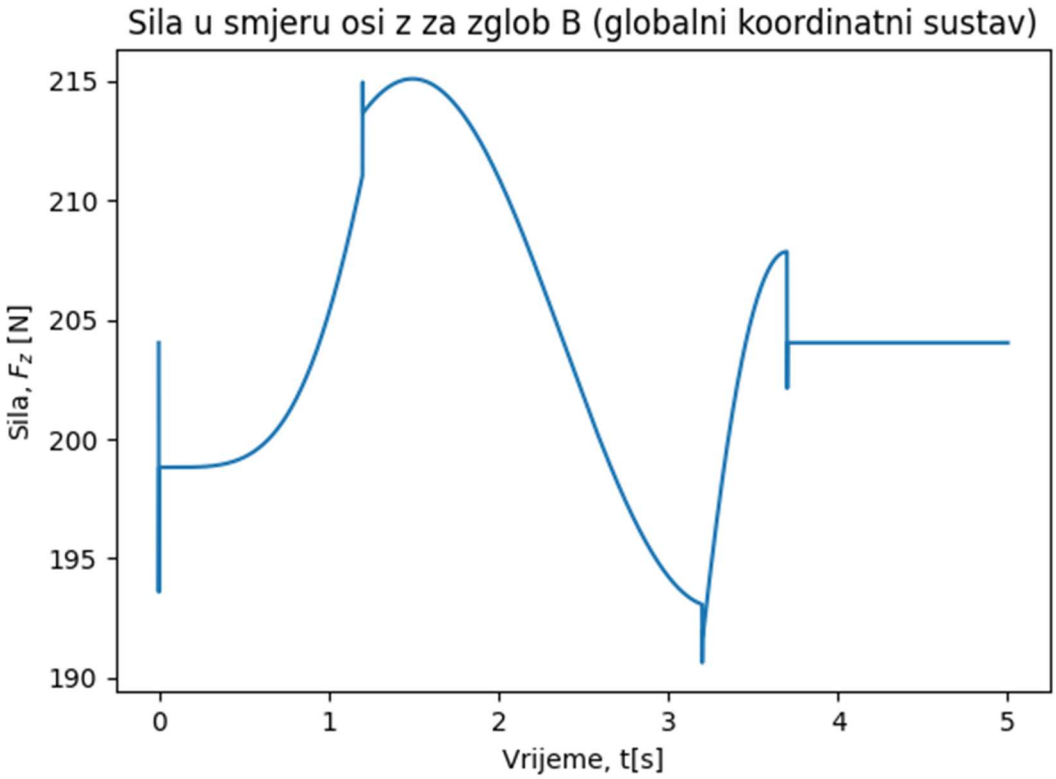

Sila u smjeru osi y za zglob A (globalni koordinatni sustav)
Sila, F_y [N]
20
10
0
−10
−20
−30
0
1
2
3
4
5
Vrijeme, t[s]

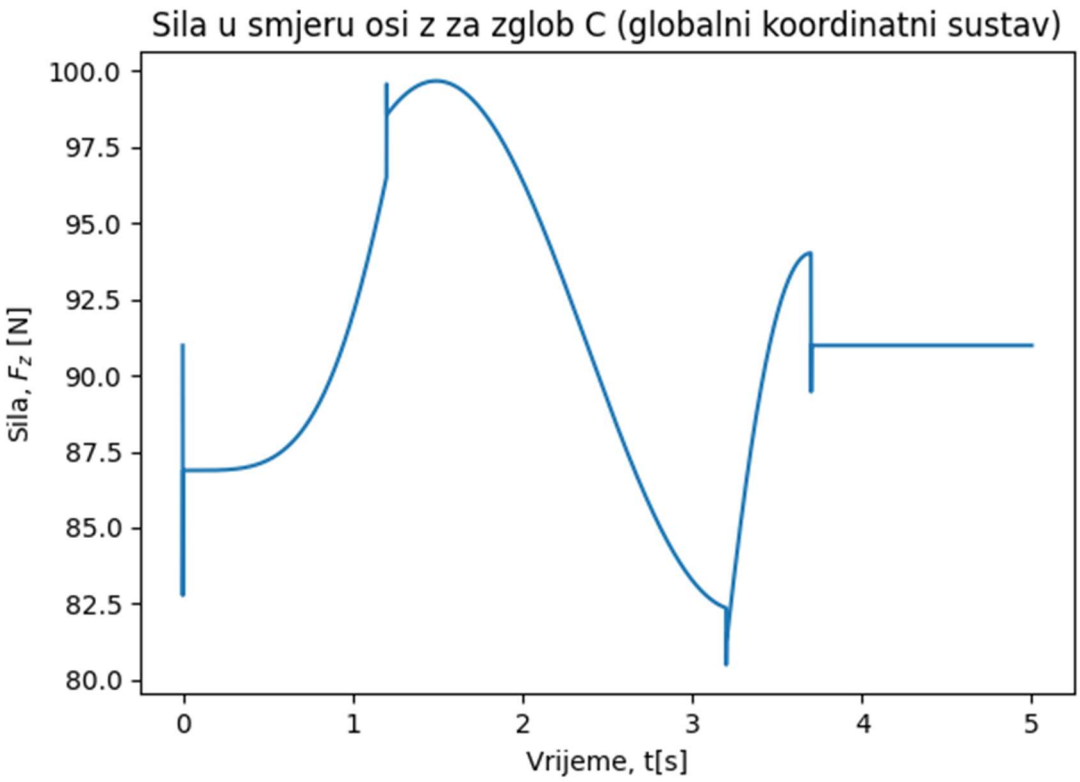

Sila u smjeru osi y za zglob B (globalni koordinatni sustav)
Sila, F_y [N]
20
10
0
−10
−20
−30
0
1
2
3
4
5
Vrijeme, t[s]

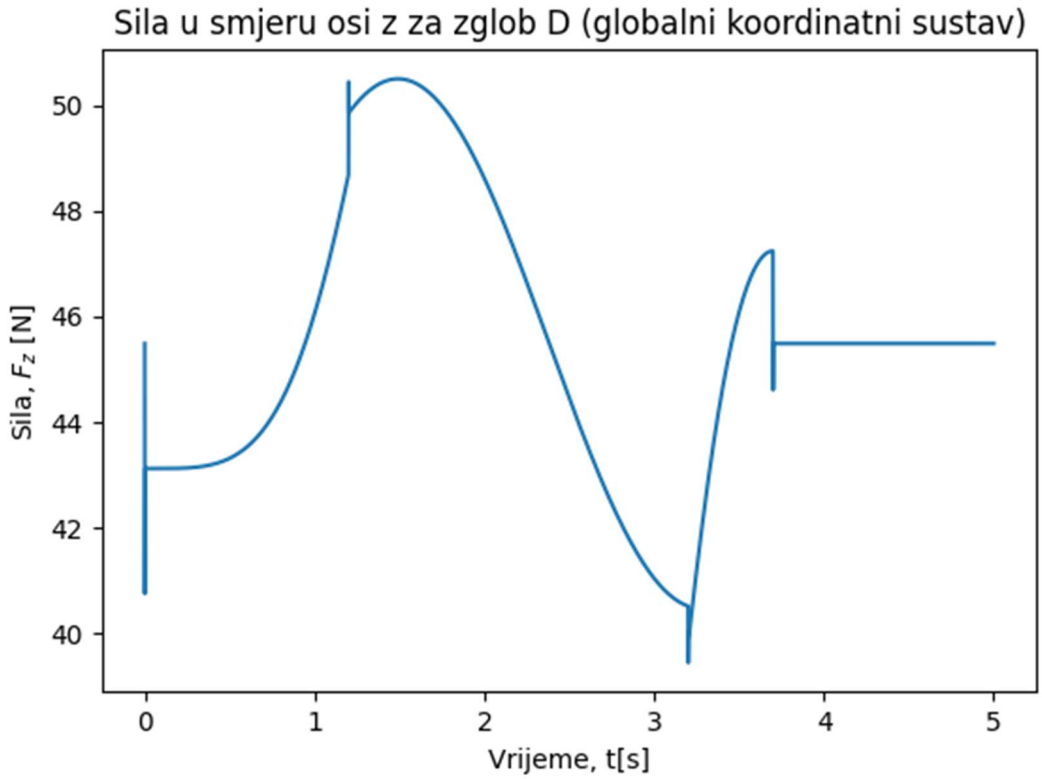

Sila u smjeru osi y za zglob C (globalni koordinatni sustav)
Sila, F_y [N]
20
10
0
−10
−20
0
1
2
3
4
5
Vrijeme, t[s]


Sila u smjeru osi y za zglob D (globalni koordinatni sustav)
Sila, F [N]
10
5
0
−5
−10
−15
0
1
2
3
4
5
Vrijeme, t[s]


Sila u smjeru osi z za zglob A (globalni koordinatni sustav)
Sila, F_z [N]
215
210
205
200
195
0
1
2
3
4
5
Vrijeme, t[s]


Sila u smjeru osi z za zglob B (globalni koordinatni sustav)
Sila, F_z [N]
215
210
205
200
195
190
0
1
2
3
4
5
Vrijeme, t[s]


Sila u smjeru osi z za zglob C (globalni koordinatni sustav)
Sila, F_z [N]
100.0
97.5
95.0
92.5
90.0
87.5
85.0
82.5
80.0
0
1
2
3
4
5
Vrijeme, t[s]


Sila u smjeru osi z za zglob D (globalni koordinatni sustav)
Sila, F_z [N]
50
48
46
44
42
40
0
1
2
3
4
5
Vrijeme, t[s]

Program vizijske aplikacije

```python
import numpy
import time
import math
import copy
import csv
import argparse
import itertools
from model import KeyPointClassifier

import urx
import serial

import cv2
from cv2 import aruco
import mediapipe as mp
import tensorflow


def get_args():
    parser = argparse.ArgumentParser()
    parser.add_argument('--use_static_image_mode', action='store_true')
    parser.add_argument("--min_detection_confidence",
                        help='min_detection_confidence',
                        type=float,
                        default=0.7)
    parser.add_argument("--min_tracking_confidence",
                        help='min_tracking_confidence',
                        type=int,
                        default=0.5)

    args = parser.parse_args()
    return args

def findAruco(frame, marker_size=6, total_markers=250):
    gray=cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    key=getattr(aruco,f'DICT_{marker_size}X{marker_size}_{total_markers}')
    arucoDict=aruco.Dictionary_get(key)
    arucoParam=aruco.DetectorParameters_create()
    bbox, ids, _ = aruco.detectMarkers(gray, arucoDict, parameters=arucoParam)
    corners, ids, _ = aruco.detectMarkers(gray, arucoDict, parameters=arucoParam)
    global xaruco
    global yaruco
    xaruco = 0
    yaruco = 0
    pronadeno = False
    if corners:
        x1 = 1280-int(corners[0][0][0][0])
        xaruco = int(x1)
        y1 = int(corners[0][0][0][1])
        yaruco = int(y1)
        print(xaruco, yaruco)
        pronadeno = True
    frame = cv2.flip(frame, 1)
    cv2.circle(frame, (int(xaruco), int(yaruco)), 5, (0, 0, 255), -1)

    scale_percent = 80
    width = int(frame.shape[1] * scale_percent / 100)
    height = int(frame.shape[0] * scale_percent / 100)
    dim = (width, height)
    resized = cv2.resize(frame, dim, interpolation=cv2.INTER_AREA)
    cv2.imshow("resized", resized)
    return ids, bbox, frame, pronadeno

def calc_landmark_list(image, landmarks):
    image_width, image_height = image.shape[1], image.shape[0]

    landmark_point = []

    # Keypoint
    for _, landmark in enumerate(landmarks.landmark):
        landmark_x = min(int(landmark.x * image_width), image_width - 1)
        landmark_y = min(int(landmark.y * image_height), image_height - 1)

        landmark_point.append([landmark_x, landmark_y])

    return landmark_point

def pre_process_landmark(landmark_list):
    temp_landmark_list = copy.deepcopy(landmark_list)

    # Convert to relative coordinates
```

```
    base_x, base_y = 0, 0
    for index, landmark_point in enumerate(temp_landmark_list):
        if index == 0:
            base_x, base_y = landmark_point[0], landmark_point[1]

        temp_landmark_list[index][0] = temp_landmark_list[index][0] - base_x
        temp_landmark_list[index][1] = temp_landmark_list[index][1] - base_y

    # Convert to a one-dimensional list
    temp_landmark_list = list(
        itertools.chain.from_iterable(temp_landmark_list))

    # Normalization
    max_value = max(list(map(abs, temp_landmark_list)))

    def normalize_(n):
        return n / max_value

    temp_landmark_list = list(map(normalize_, temp_landmark_list))

    return temp_landmark_list

def prikazRobotPoz(debug_image):
    robx = rob_pos[0] * 1000
    roby = rob_pos[1] * 1000
    robz = rob_pos[2] * 1000

    robnx = rob_pos_n[0] * 1000
    robny = rob_pos_n[1] * 1000
    robnz = rob_pos_n[2] * 1000

    drobx = robnx - robx
    drobz = robnz - robz

    dy = 1500 + roby
    hxr = drobx * (1550 / dy)
    hzr = drobz * (1550 / dy)
    xr = xaruco - hxr
    yr = yaruco - hzr

    cv2.circle(debug_image, (int(xr), int(yr)), 5, (0, 0, 255), -1)
    cv2.circle(debug_image, (int(xaruco), int(yaruco)), 5, (0, 0, 255), -1)

    return debug_image, xr, yr, dy

def racunanjeUdaljenosti(ruke, xr, yr):
    if len(ruke) == 1:
        xruka, yruka = ruke[0]
        print(xr, yr, xruka, yruka)
        udaljenost_min = math.sqrt((xr-xruka)**2 + (yr-yruka)**2)
    elif len(ruke) == 2:
        xruka1, yruka1 = ruke[0]
        xruka2, yruka2 = ruke[1]
        udaljenost_1 = math.sqrt((xr - xruka1) ** 2 + (yr - yruka1) ** 2)
        udaljenost_2 = math.sqrt((xr - xruka2) ** 2 + (yr - yruka2) ** 2)
        if udaljenost_1 < udaljenost_2:
            udaljenost_min = udaljenost_1
        else:
            udaljenost_min = udaljenost_2
    return udaljenost_min




def main():
    cap = cv2.VideoCapture(2)
    cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1280)
    cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 720)
    cap.set(cv2.CAP_PROP_FPS, 60)

    args = get_args()
    use_static_image_mode = args.use_static_image_mode
    min_detection_confidence = args.min_detection_confidence
    min_tracking_confidence = args.min_tracking_confidence

    # inicijalizacija Mediapipe-a
    mp_hands = mp.solutions.hands
    mp_drawing = mp.solutions.drawing_utils
    mp_drawing_styles = mp.solutions.drawing_styles


    #inicijalizacija robota i serial porta
    ser = serial.Serial('/dev/ttyACM0')  # open serial port
```

```
    ser.baudrate = 9600
    ser.write_timeout = 0

    robot = urx.Robot("192.168.0.25")
    input("postavi marker na vrh alata i pritisni ENTER")
    while True:
        _,frame=cap.read()
        ids, bbox, frame, pronadeno =findAruco(frame)
        if cv2.waitKey(5) & pronadeno:
            break
    global rob_pos
    #rob_pos = [-0.2855348234598093, 0.3870873627436223, 0.1098566235193552]
    rob_pos = robot.getl()
    print(xaruco, yaruco)
    cv2.destroyAllWindows()
    input("ukloni marker i pritisni ENTER")
    keypoint_classifier = KeyPointClassifier()
    kontrola = True

    with mp_hands.Hands(
            model_complexity=0,
            max_num_hands=2,
            min_detection_confidence=0.7,
            min_tracking_confidence=0.5) as hands:
        while cap.isOpened():
            success, image = cap.read()
            if not success:
                print("Ignoring empty camera frame.")
                # If loading a video, use 'break' instead of 'continue'.
                continue
            cv2.flip(image, 1)
            debug_image = copy.deepcopy(image)
            image.flags.writeable = False
            image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
            results = hands.process(image)
            if results.multi_hand_landmarks:
                ruke = []
                for hand_landmarks, handedness in zip(results.multi_hand_landmarks, results.multi_handedness):

                    landmark_list = calc_landmark_list(debug_image, hand_landmarks)

                    pre_processed_landmark_list = pre_process_landmark(landmark_list)

                    hand_sign_id = keypoint_classifier(pre_processed_landmark_list)
                    print(hand_sign_id)
                    if hand_sign_id == 0:
                        print("stop_znak")
                        kontrola = False
                    elif hand_sign_id == 1:
                        print("kreni_znak")
                        kontrola = True

                    mp_drawing.draw_landmarks(
                        debug_image,
                        hand_landmarks,
                        mp_hands.HAND_CONNECTIONS,
                        mp_drawing_styles.get_default_hand_landmarks_style(),
                        mp_drawing_styles.get_default_hand_connections_style())
                    ruke.append([(1-hand_landmarks.landmark[8].x) * debug_image.shape[1],
                                 (hand_landmarks.landmark[8].y) * debug_image.shape[0]])
            rob_pos_n1 = robot.getl()
            type_rob = type(rob_pos_n1)
            global rob_pos_n
            if isinstance(rob_pos_n1, list):
                rob_pos_n = rob_pos_n1
            else:
                rob_pos_n = rob_pos
            #rob_pos_n = [-0.3855348234598093, 0.3870873627436223, 0.2098566235193552]

            debug_image = cv2.flip(debug_image, 1)
            debug_image, xr, yr, dy = prikazRobotPoz(debug_image)
            print(xr, yr)

            uvjet = (300 * 915) / dy
            uvjet2 = (400 * 915) / dy
            uvjet3 = (500 * 915) / dy

            if kontrola:
                print("Kontrola udaljenosti: ")
                if results.multi_hand_landmarks:
                    udaljenost_min = racunanjeUdaljenosti(ruke, xr, yr)
                    print(udaljenost_min)
                    if udaljenost_min < uvjet:
                        print("stop")
```

```
                        ser.write(str.encode('stani\n'))
                        #time.sleep(0.002)
                    elif udaljenost_min > uvjet2:
                        print("kreni")
                        ser.write(str.encode('kreni\n'))
                        #time.sleep(0.002)
                    elif udaljenost_min < uvjet3:
                        print("pazi")
                        ser.write(str.encode('pazi\n'))
                        #time.sleep(0.002)
            else:
                print("stani1")
                ser.write(str.encode('stani_znak\n'))
                time.sleep(0.002)


            scale_percent = 85
            width = int(debug_image.shape[1] * scale_percent / 100)
            height = int(debug_image.shape[0] * scale_percent / 100)
            dim = (width, height)
            resized = cv2.resize(debug_image, dim, interpolation=cv2.INTER_AREA)
            cv2.imshow("resized", resized)
            if cv2.waitKey(5) & 0xFF == 27:
                break
    ser.close()
    time.sleep(2)
    robot.close()
    cv2.destroyAllWindows()
    cap.release()


if __name__ == '__main__':
    main()
    # z = 1085 mm
```

MBDyn kod

```
#MBDyn matematički model kolaborativnog robota UR5

begin: data;
   problem: initial value;
end: data;

begin: initial value;
   initial time:   0.;
   final time:     5.;
   time step:      1.e-3;
   max iterations: 10;
   tolerance:      1.e-6;
end: initial value;

begin: control data;
   structural nodes: 9;
   rigid bodies:     8;
   joints:           9;
   gravity;
end: control data;

# Design Variables
set: real M0  = 0.2
set: real M1  = 4.9;
set: real M2  = 3.5;
set: real M3  = 0.5;
set: real Mab = 4.35;
set: real Mc  = 2.275;
set: real Md  = 1.137;
set: real Met = 4.137;

set: real L1 = 0.425;
set: real L2 = 0.392;
set: real L3 = 0.1;
set: real L0 = 0.01;

set: real theta1 = 0.390;
set: real theta2 = 2.325;

# Reference Labels
set: integer Ref_Link1 = 1;
set: integer Ref_Link2 = 2;
set: integer Ref_Link3 = 3;
set: integer Ref_Link0 = 4;

# Node Labels
set: integer Node_Link1  = 1;
set: integer Node_Link2  = 2;
set: integer Node_Ground = 3;
set: integer Node_Link3  = 4;
set: integer Node_Mab    = 5;
set: integer Node_Mc     = 6;
set: integer Node_Md     = 7;
set: integer Node_Met    = 8;
```

```
set: integer Node_Link0  = 9;

# Body Labels
set: integer Body_Link1 = 1;
set: integer Body_Link2 = 2;
set: integer Body_Link3 = 3;
set: integer Body_Mab   = 4;
set: integer Body_Mc    = 5;
set: integer Body_Md    = 6;
set: integer Body_Met   = 7;
set: integer Body_Link0 = 8;

# Joint Labels
set: integer JoAxRot_Link0_Link1  = 1;
set: integer JoAxRot_Link1_Link2  = 2;
set: integer JoClamp_Ground       = 3;
set: integer JoAxRot_Link2_Link3  = 4;
set: integer JoRevh_Mab           = 5;
set: integer JoRevh_Mc            = 6;
set: integer JoRevh_Md            = 7;
set: integer JoRevh_Met           = 8;
set: integer JoAxRot_Link0_Ground = 9;

scalar function: "vozi",
   multilinear,
      0.0, 0.0,
      1.2, 1.765,
      3.2, 1.765,
      4.4, 0.0,
      5.0, 0.0;


# Reference
reference: Ref_Link0,
   null,
   euler, -pi/2., 0., 0.,
   null,
   null;
reference: Ref_Link1,
   reference, Ref_Link0, L0., 0., 0,                        # absolute position
   euler, -pi + theta1 , 0., 0., # absolute orientation
   null,                         # absolute velocity
   null;                         # absolute angular velocity

reference: Ref_Link2,
   reference, Ref_Link1, L1, 0., 0.,              # absolute position
   reference, Ref_Link1, euler, -theta2, 0., 0., # absolute orientation
   reference, Ref_Link1, null,                   # absolute velocity
   reference, Ref_Link1, null;                   # absolute angular velocity

reference: Ref_Link3,
   reference, Ref_Link2, L2, 0., 0.,
   euler, -pi, 0., 0.,
   reference, Ref_Link2, null,
```

```
   reference, Ref_Link2, null;

begin: nodes;
   structural: Node_Link0, dynamic,
      reference, Ref_Link0, 1./2.*L0, 0., 0., # absolute position
      reference, Ref_Link0, eye,              # absolute orientation
      reference, Ref_Link0, null,             # absolute velocity
      reference, Ref_Link0, null;
   structural: Node_Link1, dynamic,
      reference, Ref_Link1, 1./2.*L1, 0., 0., # absolute position
      reference, Ref_Link1, eye,              # absolute orientation
      reference, Ref_Link1, null,             # absolute velocity
      reference, Ref_Link1, null;             # absolute angular velocity
   structural: Node_Link2, dynamic,
      reference, Ref_Link2, 1./2.*L2, 0., 0., # absolute position
      reference, Ref_Link2, eye,              # absolute orientation
      reference, Ref_Link2, null,             # absolute velocity
      reference, Ref_Link2, null;             # absolute angular velocity
   structural: Node_Ground, static,
      null, eye, null, null;
   structural: Node_Link3, dynamic,
      reference, Ref_Link3, 1./2.*L3, 0., 0.,
      reference, Ref_Link3, eye,
      reference, Ref_Link3, null,
      reference, Ref_Link3, null;
   structural: Node_Mab, dynamic,
      reference, Ref_Link1, null,
      reference, Ref_Link1, eye,
      reference, Ref_Link1, null,
      reference, Ref_Link1, null;
   structural: Node_Mc, dynamic,
      reference, Ref_Link2, null,
      reference, Ref_Link2, eye,
      reference, Ref_Link2, null,
      reference, Ref_Link2, null;
   structural: Node_Md, dynamic,
      reference, Ref_Link3, null,
      reference, Ref_Link3, eye,
      reference, Ref_Link3, null,
      reference, Ref_Link3, null;
   structural: Node_Met, dynamic,
      reference, Ref_Link3, L3, 0., 0.,
      reference, Ref_Link3, eye,
      reference, Ref_Link3, null,
      reference, Ref_Link3, null;

end: nodes;

begin: elements;
   body: Body_Link0, Node_Link0,
      M0,
      null,
      diag, 0., M0*L0^2./12., M0*L0^2./12.;
   body: Body_Link1, Node_Link1,
```

```
      M1,                                # mass
      null,                             # relative center of mass
      diag, 0., M1*L1^2./12., M1*L1^2./12.; # inertia matrix

   body: Body_Link2, Node_Link2,
      M2,                                # mass
      null,                             # relative center of mass
      diag, 0., M2*L2^2./12., M2*L2^2./12.; # inertia matrix

   body: Body_Link3, Node_Link3,
      M3,
      null,
      diag, 0., M3*L3^2./12., M3*L3^2./12.;
   body: Body_Mab, Node_Mab,
      Mab,
      null,
      diag, 0.3, 0.3, 0.3;
   body: Body_Mc, Node_Mc,
      Mc,
      null,
      diag, 0.15, 0.15, 0.15;
   body: Body_Md, Node_Md,
      Md,
      null,
      diag, 0.076, 0.076, 0.076;
   body: Body_Met, Node_Met,
      Met,
      null,
      diag, 0.3, 0.3, 0.3;

   joint: JoAxRot_Link0_Ground,
      axial rotation,
                Node_Ground,
                        reference, Ref_Link0, null,
  # relative offset
                        hinge, reference, Ref_Link1, 1, 0., 0., 1., 3, 1., 0.,
0., # relative axis orientation
                Node_Link0,
                        reference, Ref_Link0, null,
  # absolute pin position
                        hinge, reference, Ref_Link1, 1, 0., 0., 1., 3, 1., 0.,
0., # absolute pin orientation
                scalar function, "vozi";
    joint: JoAxRot_Link0_Link1,
      axial rotation,
         Node_Link0,
            reference, Ref_Link1, null,                                 #
relative offset
            hinge, reference, Ref_Link2, 1, 0., 0., 1., 3, 1., 0., 0., #
relative axis orientation
         Node_Link1,
            reference, Ref_Link1, null,                                 #
relative offset
            hinge, reference, Ref_Link2, 1, 0., 0., 1., 3, 1., 0., 0., #
```

```
relative axis orientation
         const, 0.0;
   joint: JoAxRot_Link1_Link2,
      axial rotation,
         Node_Link1,
            reference, Ref_Link2, null,                                  #
relative offset
            hinge, reference, Ref_Link2, 1, 0., 0., 1., 3, 1., 0., 0., #
relative axis orientation
         Node_Link2,
            reference, Ref_Link2, null,                                  #
relative offset
            hinge, reference, Ref_Link2, 1, 0., 0., 1., 3, 1., 0., 0., #
relative axis orientation
         const, 0.0;
   joint: JoClamp_Ground,
      clamp,
         Node_Ground,
            null,
            eye;
   joint: JoAxRot_Link2_Link3,
      axial rotation,
         Node_Link2,
            reference, Ref_Link2, null,                                  #
relative offset
            hinge, reference, Ref_Link3, 1, 0., 0., 1., 3, 1., 0., 0., #
relative axis orientation
         Node_Link3,
            reference, Ref_Link2, null,                                  #
relative offset
            hinge, reference, Ref_Link3, 1, 0., 0., 1., 3, 1., 0., 0., #
relative axis orientation
         const, 0.0;

   joint: JoRevh_Mab,
      revolute hinge,
         Node_Link1,
            reference, Ref_Link1, null,
            hinge, reference, Ref_Link1, 1, 0., 0., 1., 3, 1., 0., 0.,
         Node_Mab,
            reference, Ref_Link1, null,
            hinge, reference, Ref_Link1, 1, 0., 0., 1., 3, 1., 0., 0.;
   joint: JoRevh_Mc,
      revolute hinge,
         Node_Link1,
            reference, Ref_Link2, null,
            hinge, reference, Ref_Link2, 1, 0., 0., 1., 3, 1., 0., 0.,
         Node_Mc,
            reference, Ref_Link2, null,
            hinge, reference, Ref_Link2, 1, 0., 0., 1., 3, 1., 0., 0.;
   joint: JoRevh_Md,
      revolute hinge,
         Node_Link2,
            reference, Ref_Link3, null,
```

```
            hinge, reference, Ref_Link3, 1, 0., 0., 1., 3, 1., 0., 0.,
         Node_Md,
            reference, Ref_Link3, null,
            hinge, reference, Ref_Link3, 1, 0., 0., 1., 3, 1., 0., 0.;
   joint: JoRevh_Met,
      revolute hinge,
         Node_Link3,
            reference, Ref_Link3, L3, 0., 0.,
            hinge, reference, Ref_Link3, 1, 0., 0., 1., 3, 1., 0., 0.,
         Node_Met,
            reference, Ref_Link3, L3, 0., 0.,
            hinge, reference, Ref_Link3, 1, 0., 0., 1., 3, 1., 0., 0.;

   gravity: 0., 0., -1., const, 9.81;

end: elements;
```