Strojno učenje u području robota pokretanih nogama

Tadić, Tomislav

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Tomislav Tadić

Zagreb, 2023.

SVEUČILIŠTE U ZAGREBU
FAKULTET STROJARSTVA I BRODOGRADNJE

DIPLOMSKI RAD

Mentor:

Doc. dr. sc. Petar Ćurković, dipl. ing.

Student:

Tomislav Tadić

Zagreb, 2023.

Izjavljujem da sam ovaj rad izradio samostalno koristeći znanja stečena tijekom studija i navedenu literaturu. Zahvaljujem mentoru doc. dr. sc. Petru Ćurkoviću, dipl. ing. na pruženom znanju i pomoći. Također zahvaljujem svojim roditeljima na pruženoj potpori, savjetima i osloncu, kao i svim kolegama i prijateljima koji su bili uz mene tokom studija i koji su mi na bilo koji način pomogli da uspješno položim diplomski studij fakulteta strojarstva i brodogradnje.

Tomislav Tadić

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

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

DIPLOMSKI ZADATAK

Student: **TOMISLAV TADIĆ** Mat. br.: 0035212200

Naslov rada na hrvatskom jeziku: **Strojno učenje u području robota pokretanih nogama**

Naslov rada na engleskom jeziku: **Machine learning in legged robotics**

Opis zadatka:

Strojno učenje omogućuje rješavanje složenih tehničkih problema. Jedan od takvih je pronalaženje upravljačkog zakona za mobilne robote, a posebno za robote pokretane nogama. Pri tome se zadaci mogu strukturirati na zadatke niske razine, npr. hodanje bez padanja, izbjegavanje prepreka, te zadatke visoke razine, npr. pronalaženje određene osobe ili predmeta u radnom prostoru robota.

U okviru ovog rada potrebno se upoznati s metodama strojnog učenja pogodnim za rješavanje problema niske i visoke razine. Potrebno je istražiti odgovarajuća simulacijska okružja prikladna za implementaciju strojnog učenja te simulaciju i evaluaciju ponašanja robota.

Dodatno je u okviru rada potrebno:

1. Osmisliti skup kriterija za formuliranje nagradnog signala
2. Implementirati metodu dubokog učenja za robota pokretanog nogama u fizičkom simulacijskom okružju
3. Identificirati probleme pri implementaciji dobivenog upravljačkog zakona na realnom robotskom sustavu

U radu je potrebno navesti korištenu literaturu i dobivenu pomoć.

Zadatak zadan: 17. studenog 2022.

Rok predaje rada: 19. siječnja 2023.

Predviđeni datum obrane: 23. siječnja do 27. siječnja 2023.

Zadatak zadao:
izv. prof. dr. sc. Petar Ćurković

Predsjednica Povjerenstva:
prof. dr. sc. Biserka Runje

SADRŽAJ

POPIS SLIKA

SAŽETAK

U ovom radu je na početku dan kratak opis strojnog učenja i nekih metoda strojnog učenja kao što su: nadgledano strojno učenje, nenadgledano strojno učenje, polunadgledano strojno učenje i učenje s pojačanjem. Osim toga, ukratko su opisane i umjetne neuronske mreže te genetski algoritmi koji su dijelovi umjetne inteligencije. Nakon toga fokus je stavljen na metodu strojnog učenja pod nazivom "učenje s pojačanjem". Glavni cilj ovog rada bio je implementirati tu metodu na primjeru modela dvonožnog robota, odnosno pomoću učenja s pojačanjem doći do zakona upravljanja koji će u simulacijskom okruženju pokretati robota, odnosno upravljati njegovim hodom. U radu je opisan cjelokupni postupak od izrade modela robota u SolidWorksu, prebacivanja tog modela u Simulink okruženje (dio Matlaba), prilagodbe tog modela za implementaciju učenja s pojačanjem, treniranja RL (eng. reinforcement learning) Agenta, sve do prikaza hoda modela robota upravljanog istreniranim RL Agentom unutar simulacijskog okruženja kreiranog u Simulink-u.

Ključne riječi: učenje s pojačanjem, Matlab, Simulink, SolidWorks, neuronske mreže, Agent, Okolina, Reinforcement Learning Designer, Deep Network Designer

SUMMARY

In this paper, a brief description of machine learning and some machine learning methods such as: supervised machine learning, unsupervised machine learning, semi-supervised machine learning and reinforcement learning is given at the beginning. In addition, artificial neural networks and genetic algorithms that are parts of artificial intelligence are briefly described. After that, the focus was placed on the machine learning method called "reinforcement learning". The main goal of this work was to implement that method on the example of a bipedal robot model, that is, by means of reinforcement learning, find the control laws that will move the robot in a simulation environment, that is, manage its gait. The paper describes the entire process of creating a robot model in SolidWorks, transferring that model to the Simulink environment (part of Matlab), adapting that model to implement reinforcement learning, training the RL (reinforcement learning) Agent, up to the presentation of the gait of the robot model controlled by the trained RL Agent within the simulation environment created in Simulink.

Key words: reinforcement learning, Matlab, Simulink, SolidWorks, neural networks, Agent, Environment, Reinforcement Learning Designer, Deep Network Designer

1. UVOD

Matlab je računalna platforma za programiranje i numeriku koju koriste milijuni inženjera iznanstvenika za analiziranje podataka, razvijanje algoritama i stvaranje modela. Matlab ima veliki broj dodatnih programskih paketa (toolbox-a) koji mu proširuju mogućnosti. Reinforcement Learning Toolbox i Deep Learning Toolbox su bili potrebni u ovom radu za implementaciju učenja s pojačanjem. Oni omogućuju kreiranje sustava za učenje s pojačanjem. Pružaju aplikaciju za stvaranje dubokih neuronskih mreža (Deep Network Designer aplikacija) te aplikaciju za kreiranje RL Agenata (Reinforcement Learning Designer aplikacija). Osim toga korisniku stavljaju na raspolaganje cijeli niz funkcija za potrebe razvoja ovakvih sustava strojnog učenja. U radu je demonstrirano korištenje navedenih aplikacija i nekih funkcija koje pružaju navedeni toolbox-i.

2. STROJNO UČENJE

2.1. Što je strojno učenje?

Strojno učenje (eng. ML) je disciplina umjetne inteligencije (eng. AI) koja omogućuje strojevima da automatski uče iz podataka i prošlih iskustava dok identificiraju obrasce za izradu predviđanja uz minimalnu ljudsku intervenciju.

Metode strojnog učenja omogućuju računalima da rade autonomno bez eksplicitnog programiranja. ML aplikacije se hrane novim podacima i mogu samostalno učiti, rasti, razvijati se i prilagođavati.

Strojno učenje izvlači pronicljive informacije iz velikih količina podataka korištenjem algoritama za prepoznavanje uzoraka i učenje u iterativnom procesu. ML algoritmi koriste računalne metode za izravno učenje iz podataka umjesto da se oslanjaju na bilo koju unaprijed određenu jednadžbu koja može poslužiti kao model.

Performanse ML algoritama adaptivno se poboljšavaju s povećanjem broja dostupnih uzoraka tijekom procesa 'učenja'. Na primjer, duboko učenje je poddomena strojnog učenja koja obučava računala da oponašaju prirodne ljudske osobine poput učenja iz primjera. Ono nudi bolje parametre izvedbe od konvencionalnih ML algoritama.

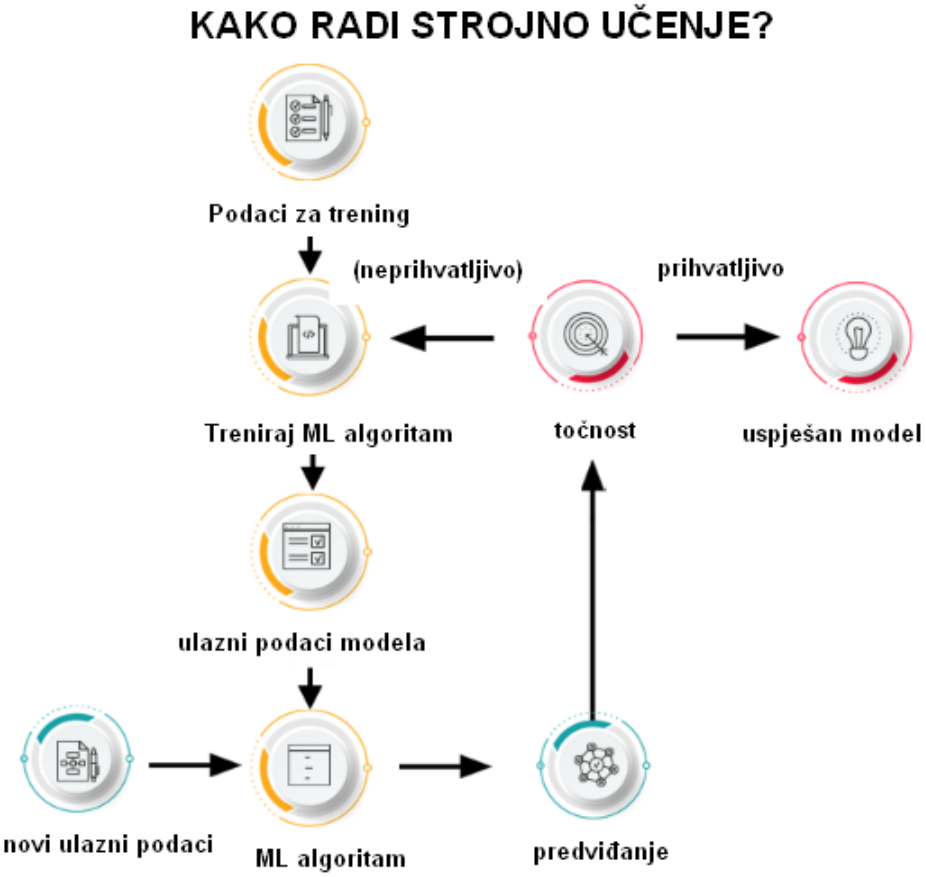
Iako strojno učenje nije novi koncept (potječe iz Drugog svjetskog rata kada je korišten “Enigma Machine“), mogućnost automatske primjene složenih matematičkih izračuna na sve veće količine i raznolikost dostupnih podataka relativno je novi razvoj.

Danas, s porastom velikih podataka (eng. big data), IoT-a (internet stvari) i sveprisutnog računalstva (eng. ubiquitous computing), strojno učenje postalo je ključno za rješavanje problema u brojnim područjima, kao što su:

- Računalne financije (bodovanje kredita, algoritamsko trgovanje)
- Računalni vid (prepoznavanje lica, praćenje pokreta, detekcija objekata)
- Računalna biologija (DNA sekvenciranje, otkrivanje tumora na mozgu, otkrivanje lijekova)
- Automobilska industrija, zrakoplovstvo i proizvodnja (prediktivno održavanje)
- Obrada prirodnog jezika (prepoznavanje glasa) [1]

2.2. Kako funkcionira strojno učenje?

Algoritmi strojnog učenja oblikuju se na skupu podataka za obuku kako bi se stvorio model. Kako se uvode novi ulazni podaci u obučeni ML algoritam, on koristi razvijeni model za izradu predviđanja.



Slika 1 Princip rada strojnog učenja [1]

Napomena: slika 1 prikazuje scenarij slučaja korištenja na visokoj razini. Međutim, tipični primjeri strojnog učenja mogu uključivati mnoge druge čimbenike, varijable i korake.

Nadalje, provjerava se točnost predviđanja. Na temelju svoje točnosti, ML algoritam se ili implementira ili trenira više puta s proširenim skupom podataka za obuku dok se ne postigne željena točnost. [1]

2.3. Vrste strojnog učenja

Algoritmi strojnog učenja mogu se trenirati na mnogo načina, a svaka metoda ima svoje prednosti i mane. Na temelju ovih metoda i načina učenja, strojno učenje se općenito kategorizira u četiri glavne vrste, kao što je prikazano na **slici 2**.

Slika 2 Vrste strojnog učenja [1]

2.3.1. Nadzirano strojno učenje

Ova vrsta strojnog učenja uključuje nadzor, pri čemu se strojevi obučavaju na označenim skupovima podataka i omogućava im se predviđanje rezultata na temelju pružene obuke. Označeni skup podataka podrazumijeva da su neki ulazni i izlazni parametri već mapirani. Dakle, stroj se trenira s ulazom i odgovarajućim izlazom. Uređaj se osposobljava za predviđanje ishoda pomoću skupa testnih podataka u uzastopnim fazama.

Na primjer, neka su ulazni skup podataka slike papiga i vrana. U početku se stroj obučava da razumije slike, uključujući boju, oči, oblik i veličinu papige i vrane. Nakon treninga daje se ulazna slika papige, a od stroja se očekuje da identificira objekt i predvidi rezultat. Istrenirani stroj provjerava različite značajke objekta (kao što su boja, oči, oblik itd., na ulaznoj slici), kako bi napravio konačno predviđanje. Ovo je proces identifikacije objekata u nadziranom strojnom učenju.

Primarni cilj tehnike nadziranog učenja je mapiranje ulazne varijable (a) s izlaznom varijablom (b). [1]

Nadzirano strojno učenje dalje se klasificira u dvije široke kategorije:

Klasifikacija: odnosi se na algoritme koji se bave problemima klasifikacije gdje je izlazna varijabla kategorička; na primjer, da ili ne, točno ili netočno, muško ili žensko, itd. Stvarne aplikacije ove kategorije vidljive su u otkrivanju neželjene pošte i filtriranju e-pošte.

Regresija: regresijski algoritmi rješavaju probleme regresije gdje ulazne i izlazne varijable imaju linearni odnos. Poznato je da ovakvi algoritmi predviđaju kontinuirane izlazne varijable. Primjeri uključuju vremensku prognozu, analizu tržišnih trendova itd. [1]

2.3.2. Nenadzirano strojno učenje

Učenje bez nadzora se odnosi na tehniku učenja koja je lišena nadzora. Ovdje se stroj obučava korištenjem neoznačenog skupa podataka i omogućeno mu je predviđanje rezultata bez ikakvog nadzora. Algoritam za učenje bez nadzora ima za cilj grupirati nerazvrstani skup podataka na temelju sličnosti, razlika i uzoraka ulaza.

Na primjer, neka je ulazni skup podataka slika posude pune voća. Ovdje slike nisu poznate modelu strojnog učenja. Kada unosimo skup podataka u ML model, zadatak modela je identificirati uzorke objekata, kao što su boja, oblik ili razlike koje se vide na ulaznim slikama i kategorizirati ih. Nakon izvršene kategorizacije ulaza stroj predviđa izlaz tokom testiranja sa testnim skupom podataka.

Nenadzirano strojno učenje dalje se klasificira u dvije vrste:

Grupiranje (eng. clustering): tehnika grupiranja se odnosi na grupiranje objekata u klastere na temelju parametara kao što su sličnosti ili razlike između objekata. Na primjer, grupiranje kupaca prema proizvodima koje kupuju.

Udruživanje (eng. assocoation): učenje udruživanja se odnosi na identificiranje tipičnih odnosa između varijabli velikog skupa podataka. Ono određuje ovisnost različitih podatkovnih stavki i mapira povezane varijable. Tipične primjene uključuju istraživanje korištenja weba i analizu tržišnih podataka. [1]

2.3.3. Polunadzirano učenje

Polu-nadzirano učenje uključuje karakteristike i nadziranog i nenadziranog strojnog učenja. Koristi kombinaciju označenih i neoznačenih skupova podataka za obuku svojih algoritama. Koristeći obje vrste skupova podataka, polu-nadzirano učenje prevladava nedostatke gore navedenih opcija.

Prethodno navedene tri vrste strojnog učenja mogu se jednostavno usporediti na primjeru studenta. Student koji uči koncept pod nadzorom nastavnika na fakultetu naziva se učenje pod nadzorom. U učenju bez nadzora, učenik samostalno uči isti koncept kod kuće bez vodstva učitelja. U međuvremenu, student koji revidira koncept nakon učenja pod vodstvom nastavnika na fakultetu je polunadzirani oblik učenja. [1]

2.3.4. Učenje s pojačanjem (eng. reinforcement learning)

Učenje s pojačanjem je proces temeljen na povratnim informacijama. Ovdje AI (eng. Artificial Intelligence) komponenta (Agent) automatski provjerava svoje okruženje iterativnom metodom pokušaja i promašaja, poduzima radnje, uči iz iskustava i poboljšava performanse. Agent se nagrađuje za svaku dobru akciju i kažnjava za svaki pogrešan potez. Dakle, Agent kao komponenta učenja s pojačanjem ima za cilj maksimizirati nagrade izvođenjem dobrih radnji.

Za razliku od nadziranog učenja, učenju s pojačanjem nedostaju označeni podaci, a agenti uče samo kroz iskustva. Razmotrite video igre. Ovdje igra specificira okruženje/okolinu, a svaki potez agenta pojačanja definira njegovo stanje. Agent ima pravo primati povratne informacije putem kazni i nagrada, čime utječe na ukupni rezultat igre. Krajnji cilj agenta je postizanje visokog rezultata.

Učenje s pojačanjem primjenjuje se u različitim područjima kao što su teorija igara, teorija informacija i sustavi s više agenata. Učenje s pojačanjem dalje se dijeli na dvije vrste metoda ili algoritama:

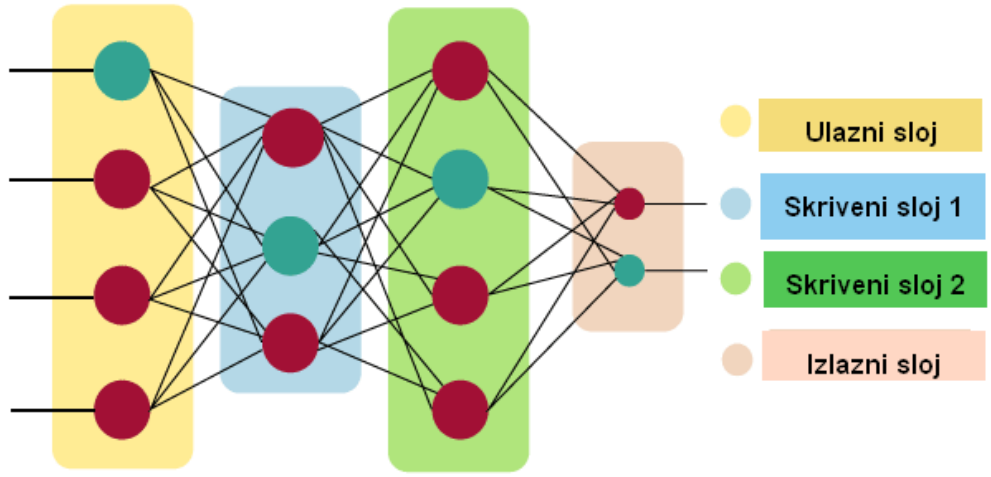
Pozitivno učenje s pojačanjem: ovo se odnosi na dodavanje pojačavajućeg podražaja nakon specifičnog ponašanja agenta, što čini vjerojatnijim da se ponašanje može ponoviti u budućnosti, npr. dodavanje nagrade nakon ponašanja.

Negativno učenje s pojačanjem: negativno učenje s pojačanjem se odnosi na jačanje specifičnog ponašanja koje izbjegava negativan ishod (kaznu). [1]

2.4. Arhitektura umjetne neuronske mreže

Da bismo razumjeli koncept arhitekture umjetne neuronske mreže, moramo razumjeti od čega se neuronska mreža sastoji. Kako bi se definirala neuronska mreža koja se sastoji od velikog broja umjetnih neurona, koji se nazivaju jedinicama raspoređenim u nizu slojeva. Pogledajmo različite vrste slojeva dostupnih u umjetnoj neuronskoj mreži prikazane na **slici 3**.

Umjetna neuronska mreža prvenstveno se sastoji od tri sloja:



Slika 3 Arhitektura umjetne neuronske mreže [1]

Ulazni sloj:

Kao što ime sugerira, prihvaća unose u nekoliko različitih formata koje nudi programer.

Skriveni sloj:

Skriveni sloj nalazi se između ulaznog i izlaznog sloja. Izvodi sve izračune kako bi pronašao skrivene značajke i uzorke.

Izlazni sloj:

Ulaz prolazi kroz niz transformacija pomoću skrivenog sloja, što konačno rezultira izlazom koji se prenosi pomoću ovog sloja.

Umjetna neuronska mreža uzima ulazne podatke i izračunava ponderirani zbroj ulaznih podataka te uključuje pristranost (eng. bias). Ovaj izračun je predstavljen u obliku prijenosne funkcije.

$$\sum_{i=1}^{n} \boldsymbol{Wi} * \boldsymbol{Xi} + \boldsymbol{b}$$

Određuje ponderirani ukupni iznos koji se prosljeđuje kao ulaz u aktivacijsku funkciju za proizvodnju izlaza. Aktivacijske funkcije biraju hoće li se čvor aktivirati ili ne. Samo oni koji su otpušteni stignu do izlaznog sloja. Dostupne su različite funkcije aktivacije koje se mogu primijeniti na vrstu zadatka koji obavljamo.

Prednosti umjetne neuronske mreže (UNM)

Mogućnost paralelne obrade:

Umjetne neuronske mreže imaju brojčanu vrijednost koja može obavljati više od jednog zadatka istovremeno.

Pohranjivanje podataka na cijeloj mreži:

Podaci koji se koriste u tradicionalnom programiranju pohranjuju se na cijeloj mreži, a ne u bazi podataka. Nestanak nekoliko podataka na jednom mjestu ne sprječava rad mreže.

Sposobnost za rad s nepotpunim znanjem:

Nakon obuke UNM, informacije mogu proizvesti izlaz čak i s neadekvatnim podacima. Gubitak performansi ovdje ovisi o važnosti podataka koji nedostaju.

Postojanje distribucije memorije:

S obzirom da UNM moraju biti prilagodljive, važno je odrediti primjere i potaknuti mrežu prema željenom izlazu demonstracijom tih primjera mreži. Uspješnost mreže izravno je proporcionalna odabranim instancama, a ako se događaj ne može prikazati mreži u svim svojim aspektima, može proizvesti lažne rezultate mreže.

Imati toleranciju na greške:

Nedostatak jedne ili više ćelija UNM joj ne zabranjuje generiranje izlaza, a ova značajka čini mrežu otpornom na greške.

Nedostaci umjetne neuronske mreže:

Osiguranje ispravne strukture mreže:

Ne postoji posebna smjernica za određivanje strukture umjetnih neuronskih mreža. Odgovarajuća struktura mreže postiže se iskustvom, pokušajima i pogreškama.

Neprepoznato ponašanje mreže:

To je najznačajniji problem UNM. Kada UNM daje rješenje prilikom testiranja, ona ne daje uvid zašto i kako je došla do takvog rješenja. To smanjuje povjerenje u mrežu.

Ovisnost o hardveru:

Umjetne neuronske mreže trebaju procesore s paralelnom procesorskom snagom, prema njihovoj strukturi. Stoga je realizacija opreme ovisna.

Poteškoće u prikazivanju problema mreži:

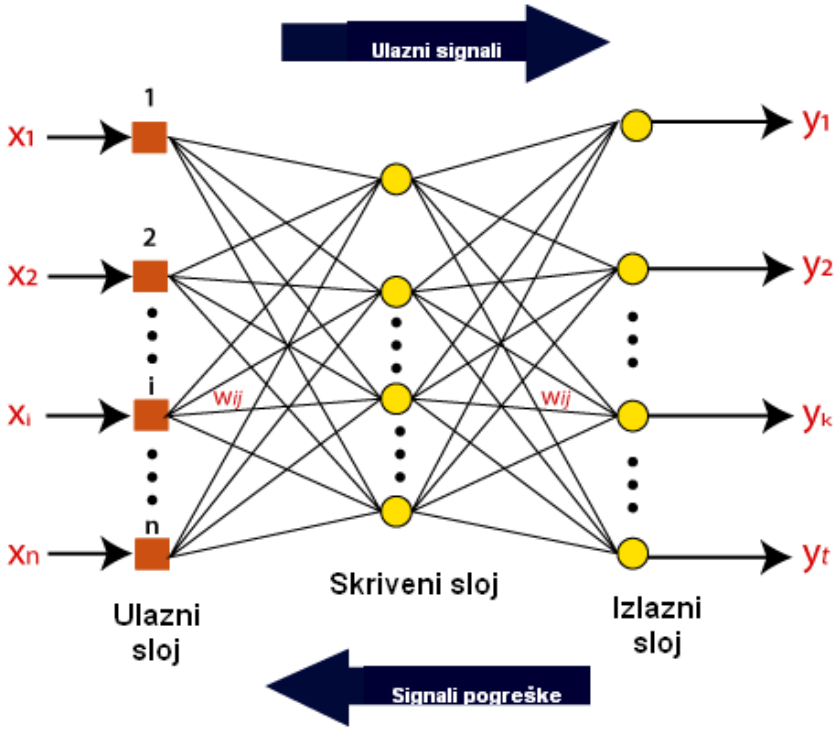
UNM mogu raditi s numeričkim podacima. Problemi se moraju pretvoriti u numeričke vrijednosti prije uvođenja u UNM. Mehanizam prezentacije koji ovdje treba osmisliti izravno će utjecati na performanse mreže. Dakle, UNM se oslanja na sposobnosti korisnika.

Trajanje mreže nije poznato:

Mreža je svedena na određenu vrijednost pogreške, a ta nam vrijednost ne daje optimalne rezultate. [1]

2.4.1. Kako rade umjetne neuronske mreže?

Umjetna neuronska mreža može se najbolje prikazati kao težinski usmjereni graf, gdje umjetni neuroni tvore čvorove, kako prikazuje **slika 4**. Povezanost između neuronskih izlaza i neuronskih ulaza može se promatrati kao usmjereni rubovi s težinama. Umjetna neuronska mreža prima ulazni signal iz vanjskog izvora u obliku uzorka i slike u obliku vektora. Ovi ulazi se zatim matematički dodjeljuju oznakama x(n) za svaki n broj ulaza.



Slika 4 Princip rada umjetnih neuronskih mreža [2]

Nakon toga, svaki ulaz se množi sa svojim odgovarajućim težinama (ove težine su detalji koje koriste umjetne neuronske mreže za rješavanje određenog problema). Općenito govoreći, ove

težine obično predstavljaju snagu međupovezanosti između neurona unutar umjetne neuronske mreže. Svi ponderirani ulazi sumiraju se unutar računske jedinice.

Ako je ponderirani zbroj jednak nuli, tada se dodaje pristranost (eng. bias) kako bi izlaz postao različit od nule ili nešto drugo kako bi se skalirao na odgovor sustava. Bias ima isti ulaz, a težina je jednaka 1. Ovdje zbroj ponderiranih ulaza može biti u rasponu od 0 do pozitivne beskonačnosti. Ovdje se, kako bi se odgovor zadržao u granicama željene vrijednosti, određuje određena maksimalna vrijednost, a ukupni rezultat ponderiranih ulaza prolazi kroz aktivacijsku funkciju.

Aktivacijska funkcija odnosi se na skup prijenosnih funkcija koje se koriste za postizanje željenog učinka. Postoje različite vrste aktivacijskih funkcija, ali primarno su to linearni ili nelinearni skupovi funkcija. Neki od najčešće korištenih skupova aktivacijskih funkcija su binarne, linearne i Tan-hiperbolične sigmoidalne aktivacijske funkcije.

Binarne aktivacijske funkcije:

U binarnoj aktivacijskoj funkciji izlaz je ili **1** ili **0**. Postavljena je vrijednost praga da bi se to postiglo. Ako je neto ponderirani ulaz neurona veći od 1, tada se konačni izlaz aktivacijske funkcije vraća kao jedan, a u suprotnom se izlaz vraća kao 0.

Sigmoidna hiperbolika:

Funkcija sigmoidalne hiperbole općenito se vidi kao krivulja u obliku slova "S".

Ovdje se tan-hiperbolička funkcija koristi za aproksimaciju izlaza iz stvarnog neto ulaza. Funkcija je definirana kao:

F(x) = (1/1 + exp(-????x))

Gdje se “????“ smatra parametrom strmosti.

Vrste umjetne neuronske mreže

Postoje različite vrste umjetnih neuronskih mreža (UNM) ovisno o funkcijama neurona i mreže ljudskog mozga. Umjetna neuronska mreža na sličan način obavlja zadatke. Većina umjetnih neuronskih mreža ima neke sličnosti sa složenijim biološkim partnerom i vrlo su učinkovite u svojim očekivanim zadacima. Na primjer, segmentacija ili klasifikacija.

UNM s povratnom vezom (eng. Feedback ANN):

U ovoj vrsti UNM, izlaz se vraća u mrežu kako bi se interno postigli najbolje evoluirani rezultati. Mreže s povratnom vezom vraćaju informacije u sebe i dobro su prikladne za rješavanje optimizacijskih problema. Ispravci internih grešaka sustava koriste povratne UNM.

UNM s unaprijednom vezom (eng. Feed-Forward ANN):

Ovo je osnovna neuronska mreža koja se sastoji od ulaznog sloja, izlaznog sloja i najmanje jednog sloja neurona. Kroz procjenu izlaza pregledom ulaza, intenzitet mreže može se uočiti na temelju grupnog ponašanja povezanih neurona i odlučuje se o izlazu. Primarna prednost ove mreže je to što može shvatiti kako procijeniti i prepoznati uzorke danih ulaza. [2]

2.5. Genetski algoritami

Genetski algoritmi (GA) su klasa algoritama pretraživanja osmišljenih na temelju prirodnog evolucijskog procesa. Genetski algoritmi temelje se na principu preživljavanja najjačih. Genetski algoritmi se koriste za rješavanje problema optimizacije kopiranjem evolucijskog ponašanja vrsta. Od početne nasumične populacije rješenja, ova populacija napreduje kroz selekciju, mutaciju i operatore križanja, inspiriranih prirodnom evolucijom. Implementacijom zadanog skupa operacija populacija prolazi kroz iterativnu proceduru tokom koje dolazi do različitih stanja, a svako stanje se naziva generacija. Kao rezultat ovog postupka, očekuje se da populacija dosegne generaciju u kojoj se nalazi pristojno rješenje problema. U genetskom algoritmu, rješenje problema je kodirano kao niz bitova ili realnih brojeva .

U praksi su se pokazali kao vrlo učinkoviti u optimiziranju funkcija. Koriste se u pretraživanju velikih i sofisticiranih prostora. Genetski algoritmi (GA) su algoritmi koji se koriste za optimizaciju i strojno učenje na temelju različitih značajki biološke evolucije.

Potrebne su im slijedeće komponente:

- Proces kodiranja rješenja za rješavanje problema kromosoma.

- Funkcija evaluacije koja svakom kromosomu dodjeljuje ocjenu dobrote

- Operatori koji se mogu primijeniti na roditelje kada se oni razmnožavaju kako bi im se promijenio genetski sastav. Standardni operatori su mutacije i križanja.

Razvoj UNM-a s evolucijskim računanjem:

Unapređenje UNM-a je tema kojom se naširoko bavilo vrlo različitim tehnikama. Svijet evolucijskih algoritama nije iznimka, a dokaz tome je nevjerojatna količina radova koji su objavljeni o različitim tehnikama u ovom području. Općenito gledano, polje generiranja UNM-

a korištenjem evolucijskih algoritama podijeljeno je u tri glavna područja: Evolucija težine, Arhitekture i Pravila učenja .

Inicijalno, evolucija težine počinje od UNM s prethodno određenom topologijom. Problem koji treba riješiti je obuka asocijacijskih težina, čime se nastoji ograničiti pogreška mreže. Uz korištenje evolucijskog algoritma, težine se mogu prikazati kao veza binarnih ili realnih vrijednosti.

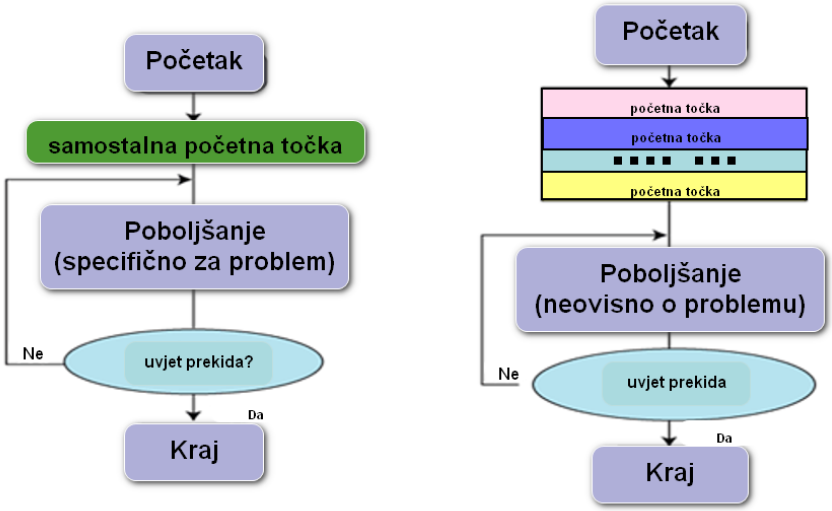
Nakon toga, evolucija arhitekture uključuje stvaranje topološke strukture. Kako bi se koristili evolucijski algoritmi za stvaranje arhitektura UNM-a, potrebno je odabrati način šifriranja genotipa dane mreže da bi ju genetski operateri mogli koristiti.

Kod prve opcije (izravno kodiranje), postoji uravnotežena analogija između svih gena i njihovih rezultirajućih fenotipova. Najuobičajenija tehnika kodiranja sastoji se od matrice koja predstavlja arhitekturu gdje svaka komponenta otkriva prisutnost ili odsutnost povezanosti između dva čvora.

U shemama kodiranja, GA je korišten za istovremeno stvaranje i arhitekture i asocijacijskih težina, bilo za unaprijedne ili rekurentne UNM, bez ograničenja u njihovoj arhitekturi. Ova nova shema kodiranja dopušta i stjecanje osnovnih mreža s minimalnim brojem neurona i asocijacija, a objavljeni rezultati su povoljni. Osim izravnog kodiranja, postoje i neke indirektne tehnike kodiranja. U ovim tehnikama je samo nekoliko karakteristika arhitekture kodirano u kromosomu. Ove tehnike imaju različite vrste reprezentacija. Prvo, parametarske reprezentacije prikazuju mrežu kao skupinu parametara. Na primjer, broj čvorova za svaki sloj, broj asocijacija između dva sloja, broj skrivenih slojeva itd. Drugi tip indirektnog kodiranja ovisi o gramatičkim pravilima. U ovom sustavu mreža je predstavljena skupinom propisa, izgrađenih kao proizvodna pravila koja čine matricu koja predstavlja mrežu. S obzirom na evoluciju pravila učenja, postoje različiti pristupi, međutim, većina njih se temelji samo na tome kako učenje može promijeniti ili upravljati evolucijom, kao i na odnosu između arhitekture i asocijacijskih težina. [3]

2.5.1. Princip rada UNM-a

Princip rada standardnog genetskog algoritma ilustriran je na danoj **slici**. Značajni koraci su generiranje populacije rješenja, definiranje funkcije cilja i fitnes funkcije te primjena genetskih operatora. Ovi aspekti su opisani uz pomoć dolje prikazanog temeljnog genetskog algoritma, **slika 5**.



Slika 5 Shema rada genetskog algoritma [3]

Početak:

Generiranje slučajne populacije od n kromosoma.

Fitnes:

Izračunavanje fitnesa (sposobnosti/kvalitete) f(x) svakog kromosoma x u populaciji.

Nova populacija:

Generiranje nove populacije ponavljanjem sljedećih koraka dok se Nova populacija ne završi.

Selekcija:

Odabir dva roditeljska kromosoma iz populacije prema njihovom fitnesu. Što je veći fitnes kromosoma, to je veća vjerojatnost da budu odabrani.

Križanje:

U skladu s vjerojatnosti križanja, križanje roditelja kako bi se formirali novi potomci (djeca). Ako nije izvršeno križanje, potomak je točna kopija roditelja.

Mutacija:

U skladu s vjerojatnosti mutacije, mutiraju se novi potomci na svakom lokusu.

Prihvaćanje:

Smještanje novih potomaka u novu populaciju.

Zamjena:

Korištenje novogenerirane populacije za daljnji rad algoritma.

Test:

Ako je krajnji uvjet zadovoljen, tada se GA zaustavlja i vraća najbolje rješenje u trenutnoj populaciji.

Petlja:

U ovom koraku se prelazi na drugi korak za procjenu fitnesa.

Osnovno načelo iza genetskih algoritama je da oni stvaraju i održavaju populaciju jedinki predstavljenu kromosomima. Kromosomi su znakovni niz (eng. character string) praktički ekvivalentan kromosomima koji se pojavljuju u DNK-u. Ti su kromosomi obično kodirana rješenja problema. Odvija se proces evolucije prema pravilima selekcije, reprodukcije i mutacije. Svaki pojedinac u okolini (predstavljen kromosomom) dobiva mjeru svog fitnesa (prikladnosti) za okolinu. Reprodukcija odabire jedinke s visokim vrijednostima fitnessa u populaciji. Kroz križanje i mutaciju takvih jedinki, određuje se nova populacija u kojoj bi jedinke mogle još bolje odgovarati svojoj okolini. Proces križanja uključuje dva kromosoma koji zamjenjuju dijelove podataka i analogan je procesu reprodukcije. [3]

2.5.2. Razlika između tradicionalnog i genetskog pristupa

Algoritam je niz koraka za rješavanje problema. Genetski algoritam je tehnika rješavanja problema koja koristi genetiku kao svoj model rješavanja problema. To je metoda pretraživanja za pronalaženje približnih rješenja problema optimizacije i pretraživanja. Lako se može razlikovati tradicionalni i genetski algoritam.

Prednosti genetskog algoritma:

- Koncept genetskog algoritma je lako razumjeti.
- GA podržava optimizaciju s više ciljeva.
- GA je prikladan za bučna okruženja.
- GA je robustan s obzirom na lokalne minimume/maksimume.
- GA koristi probabilistička pravila prijelaza.
- GA koristi informacije o ciljnoj funkciji, a ne izvedenice.
- GA dobro radi na mješovitim diskretnim funkcijama.

Ograničenja genetskog algoritma:

Iako su se genetski algoritmi pokazali kao brz i moćan pristup rješavanju problema, određena ograničenja su ugrađena u njih. Neka od tih ograničenja navedena su u nastavku:

Prvo, i najvažnije, u izradi genetskog algoritma važna je karakterizacija prikaza problema. Jezik koji se koristi za određivanje mogućih rješenja mora biti robustan. Mora biti u stanju podnijeti nasumične promjene kako ne bi došlo do fatalnih grešaka.

Jedna značajna prepreka genetskim algoritmima je kodiranje fitnes funkcije (evaluacije) tako da se može postići veći fitnes i mogu proizvesti bolja rješenja za problem. Pogrešna odluka fitnes funkcije može dovesti do značajnih posljedica. Na primjer, nemogućnost pronalaska rješenja za problem i vraćanje krivih rješenja.

Uz pristojan izbor fitnes funkcije, različiti parametri genetskog algoritma poput veličine populacije, mutacije i stope križanja također moraju biti učinkovito odabrani. Mala veličina populacije neće biti u stanju genetskom algoritmu dati dovoljno potencijalnih rješenja da GA proizvede precizne rezultate. Učestalost genetske promjene ili loša selekcijska shema rezultirat će remećenjem korisne sheme.

Za analitičke probleme se ne preporučuje korištenje genetskih algoritama. Iako genetski algoritmi mogu pronaći točna rješenja za ove vrste problema, tradicionalne analitičke tehnike mogu pronaći ista rješenja u kratkom vremenu s malo računskih podataka. [3]

2.5.3. Genetski algoritam u robotici

Robotika je jedno od područja o kojima se najviše raspravlja u današnjoj računalnoj industriji. Koristi se u raznim industrijama kako bi se povećala profitabilnost, učinkovitost i točnost. Kako se okruženja u kojima roboti rade s vremenom mijenjaju, programerima postaje vrlo teško uzeti u obzir svako moguće ponašanje robota kako bi se roboti mogli nositi s promjenama. Ovo je mjesto gdje genetski algoritam ima vitalnu ulogu. Stoga je potrebna odgovarajuća metoda koja će dovesti robota do njegovog cilja i učiniti ga prilagodljivim novim situacijama kada se s njima susretne. Genetski algoritmi su prilagodljive tehnike pretraživanja koje se koriste za učenje struktura znanja visokih performansi. [3]

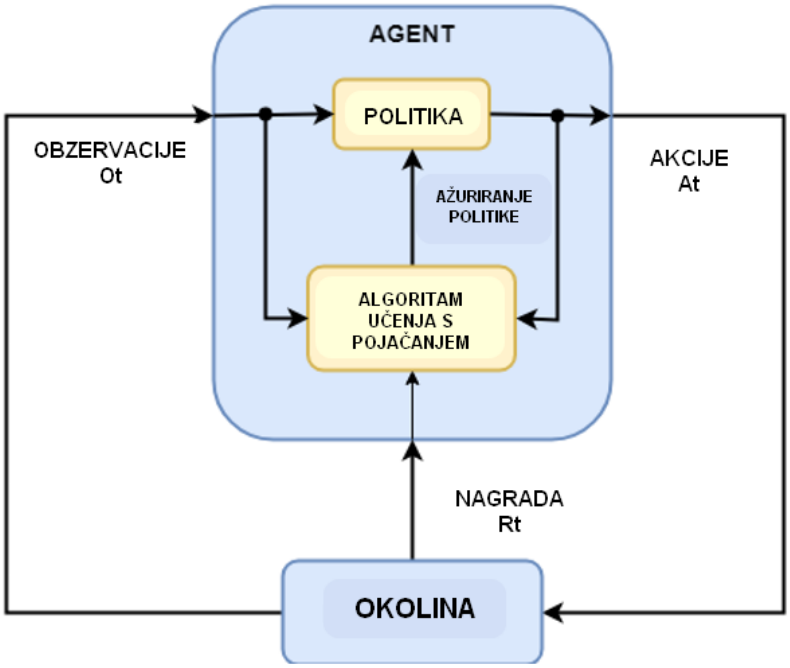
3. OPIS REINFORCEMENT LEARNING TOOLBOX-A

U sklopu ovog diplomskog rada, od navedenih metoda strojnog učenja korištena je metoda **učenje s pojačanjem (eng. reinforcement learning),** na primjeru učenja hodanja jednostavnog modela dvonožnog robota.

Reinforcement Learning Toolbox nudi aplikaciju, funkcije i Simulink blok za politike obuke pomoću algoritama za **učenje s pojačanjem**, uključujući DQN (Deep Q-Networks), PPO (Proximal Policy Optimization), SAC (Soft Actor-Critic) i DDPG (Deep Deterministic Policy Gradient) algoritme. Ove politike (skupovi pravila ponašanja) mogu se koristiti za implementaciju sustava upravljanja i algoritama donošenja odluka za složene aplikacije kao što su dodjela resursa, robotika i autonomni sustavi. Ovaj toolbox omogućuje predstavljanje politika i vrijednosnih funkcija korištenjem dubokih neuronskih mreža ili preglednih (eng. look-up) tablica i njihovo uvježbavanje kroz interakcije s okruženjima modeliranima u MATLAB-u ili Simulink-u. Mogu se procjenjivati algoritmi za **učenje s pojačanjem** s jednim ili više agenata koji se nalaze unutar toolbox-a ili razvijati vlastiti algoritmi. Moguće je eksperimentirati s postavkama hiperparametara, pratiti napredak obuke i simulirati obučene agente interaktivno putem aplikacije ili programski. Za poboljšanje performansi treninga, simulacije se mogu izvoditi paralelno na više CPU-a, GPU-a, računalnih klastera i oblaka (s Parallel Computing Toolbox-om i MATLAB Parallel Server-om). [4]

3.1. Što je učenje s pojačanjem?

Učenje s pojačanjem je računalni pristup usmjeren cilju, gdje računalo uči izvršavati neki zadatak interakcijom s nepoznatim dinamičkim okruženjem. Ovaj pristup učenju omogućuje računalu da donese niz odluka kako bi maksimizirao kumulativnu nagradu za zadatak bez ljudske intervencije i bez eksplicitnog programiranja za postizanje tog zadatka. Dijagram koji slijedi (**slika 6**) prikazuje općenitu reprezentaciju scenarija učenja s pojačanjem. Putem formata modela ONNX (Open Neural Network Exchange), postojeće politike (eng. policies) se mogu uvesti iz radnih okvira dubokog učenja kao što su TensorFlow Keras i PyTorch (s Deep Learning Toolbox-om). Mogu se generirati optimizirani C, C++ i CUDA kôdovi za implementaciju uvježbanih politika na mikrokontrolerima i GPU-ovima. Ovaj toolbox uključuje referentne primjere koji pomažu korisnicima da započnu s radom na implementaciji učenja s pojačanjem.



Slika 6 Općenitu reprezentacija scenarija učenja s pojačanjem [5]

Cilj učenja s pojačanjem je istrenirati agenta da izvrši zadatak unutar nepoznatog okruženja. Agent prima obzervacije i nagradu od okoline te šalje akcije okolini. Nagrada je mjera za to koliko je akcija uspješna u pogledu ispunjavanja cilja zadatka.

Agent sadrži dvije komponente: politiku (skup pravila ponašanja) i algoritam učenja.

Politika predstavlja mapiranje koje odabire akcije na temelju obzervacija iz okoline. Uobičajeno, politika je aproksimator funkcije s podesivim parametrima, kao što je na primjer

duboka neuronska mreža. Algoritam učenja kontinuirano ažurira parametre politike na temelju akcija, obzervacija i nagrade. Cilj algoritma učenja je pronaći optimalnu politiku koja maksimizira kumulativnu nagradu primljenu tijekom izvođenja zadatka. Drugim riječima, **učenje s pojačanjem** uključuje agenta koji uči optimalno ponašanje kroz ponovljene interakcije pokušaja i pogrešaka s okolinom bez ljudskog angažmana. Kao primjer, razmotrite zadatak parkiranja vozila pomoću sustava automatizirane vožnje. Cilj ovog zadatka je da računalo vozila (agent) parkira vozilo u ispravnom položaju i orijentaciji. Da bi to učinio, kontroler koristi očitanja s kamera, akcelerometara, žiroskopa, lidar-a (eng. Light Detection and Ranging) i GPS prijemnika (obzervacije) za generiranje naredbi (akcije) upravljanja, kočenja i ubrzanja. Naredbe za djelovanje (akcije) šalju se aktuatorima koji upravljaju vozilom. Rezultirajuće obzervacije ovise o aktuatorima, senzorima, dinamici vozila, površini ceste, vjetru i mnogim drugim manje važnim čimbenicima. Svi ti čimbenici, odnosno sve što nije agent, čine okruženje (eng. environment) u **učenju s pojačanjem**. Kako bi naučilo generirati ispravne akcije iz obzervacija, računalo iterativno pokušava parkirati vozilo koristeći postupak pokušaja i pogreške. Za usmjeravanje procesa učenja potrebno je davati signal koji je 1 kada automobil uspješno dosegne željenu poziciju i orijentaciju i 0 u suprotnom (nagrada). Tijekom svake probe, računalo odabire akcije pomoću mapiranja (politike) inicijaliziranog nekim zadanim vrijednostima. Nakon svake probe, računalo ažurira mapiranje kako bi se maksimizirala nagrada (algoritam učenja). Ovaj se proces nastavlja sve dok računalo ne nauči optimalno mapiranje koje uspješno parkira automobil. [5]

3.2. Tijek rada učenja s pojačanjem

Opći tijek rada za treniranje agenta pomoću učenja s pojačanjem uključuje sljedeće korake.

Formuliranje problema: definiranje zadataka koji agent treba naučiti, uključujući način interakcije agenta s okolinom i sve primarne i sekundarne ciljeve koje agent mora postići.

Stvaranje okruženja: definiranje okruženja unutar kojeg agent radi, uključujući sučelje između agenta i okruženja i dinamički model okruženja.

Definiranje nagrade: određivanje nagradnog signala kojeg agent koristi za mjerenje svoje izvedbe u odnosu na ciljeve zadatka i načina kako izračunati taj signal iz okruženja.

Stvaranje agenta: stvaranje agenta, što uključuje definiranje reprezentacije politike (pravila) i konfiguriranje algoritma učenja agenta.

Treniranje agenta: treniranje reprezentacije politike agenta korištenjem definiranog okruženja, nagrade i algoritma učenja agenta.

Validacija agenta: procjena izvedbe istreniranog agenta zajedničkim simuliranjem agenta i okruženja.

Primjena politike: implementacija istrenirane reprezentacije politike pomoću, na primjer, generiranog GPU koda.

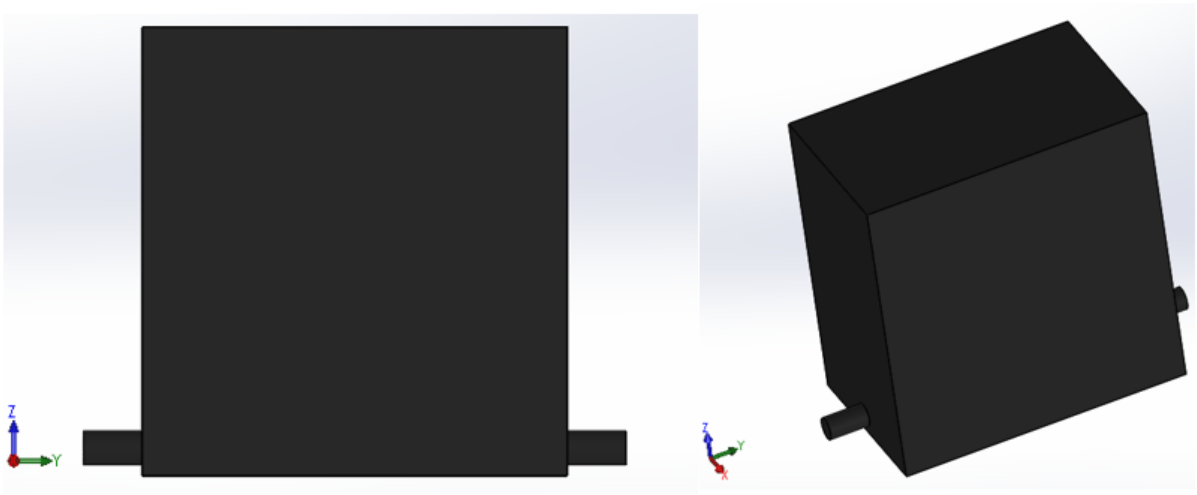
Treniranje agenta pomoću učenja s pojačanjem je iterativni proces. Odluke i rezultati u kasnijim fazama mogu zahtijevati povratak na raniju fazu u tijeku rada učenja. Na primjer, ako proces treniranja ne konvergira prema optimalnoj politici unutar razumnog vremena, možda će biti potrebno ažurirati nešto od sljedećeg prije ponovne obuke agenta:

- Postavke treninga
- Konfiguracija algoritma učenja
- Reprezentacija politike
- Definicija nagradnog signala
- Signali akcija i obzervacija
- Dinamika okoline [5]

4. OPIS PRAKTIČNOG DIJELA RADA

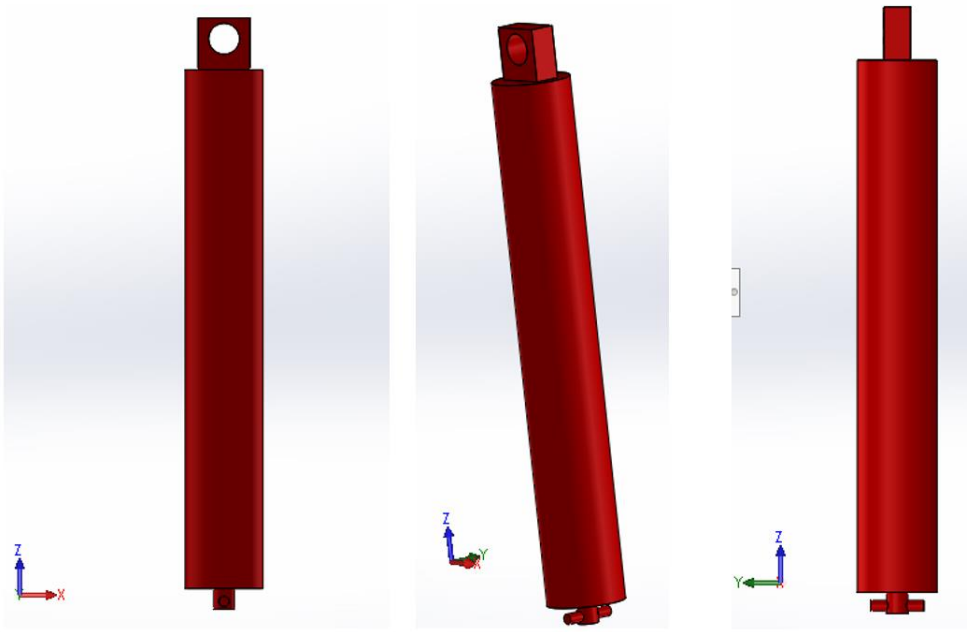
4.1. Izrada modela robota

Dijelovi modela robota izrađeni su u programu **SolidWorks**, te su u istom programu spojeni u sklop (eng. assembly**)**, u kojem su definirani međusobni položaji dijelova te rotacijski zglobovi. Ukupno je definirano 6 zglobova (kuk, koljeno i gležanj za obje noge modela robota!). Na slikama su prikazani dijelovi robota u SolidWorks-u, prvo pojedinačno, a zatim i u sklopu.



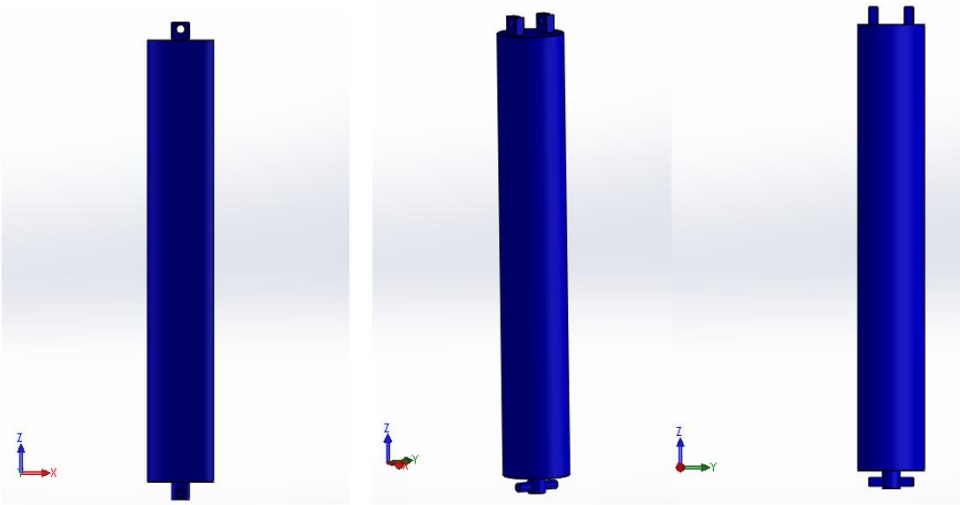
Slika 7 Torzo modela robota u SolidWorksu (SW)

Slika 7 prikazuje torzo modela robota na kojega se spajaju robotove noge.



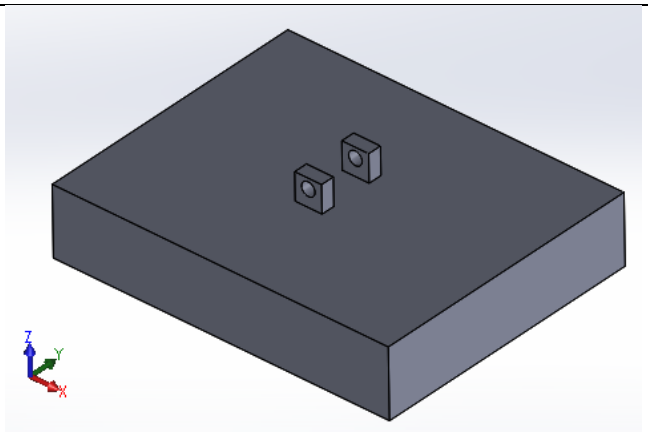
Slika 8 Nadkoljenica modela robota u SW

Slika 8 prikazuje nadkoljenicu modela robota koja se najprije spaja s torzom, a zatim se na nju spaja potkoljenica modela. Lijeva i desna nadkoljenica su identičnog oblika i svojstava.



Slika 9 Potkoljenica modela robota u SW

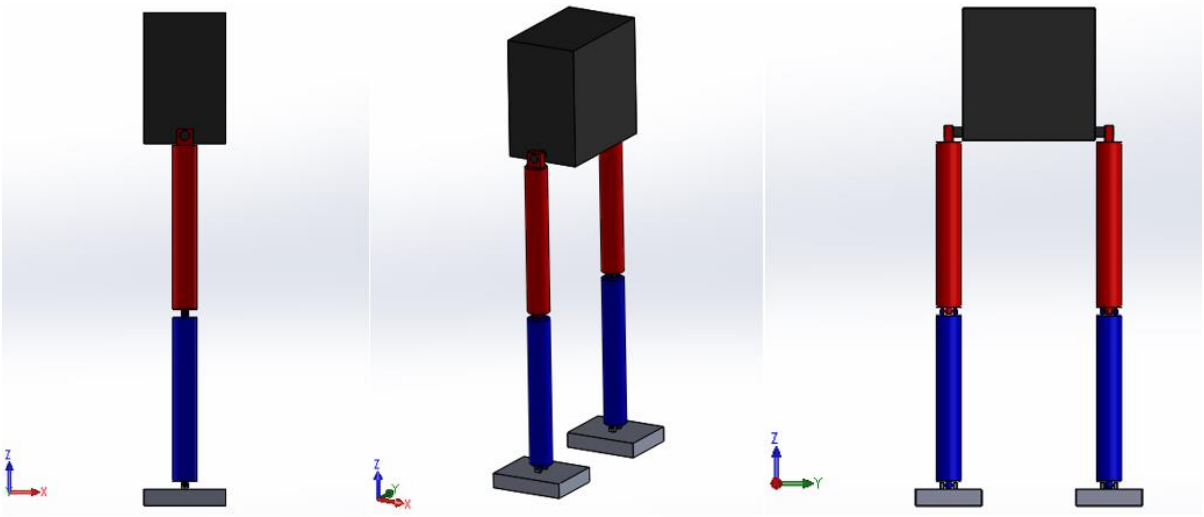
Slika 9 prikazuje potkoljenicu modela robota koja se najprije spaja s nadkoljenicom, a zatim se na nju spaja stopalo modela. Lijeva i desna potkoljenica su identičnog oblika i svojstava.



Slika 10 Stopalo modela robota u SW

Slika 10 prikazuje stopalo modela robota koje se spaja na potkoljenicu. Lijevo i desno stopalo su identičnog oblika i svojstava.

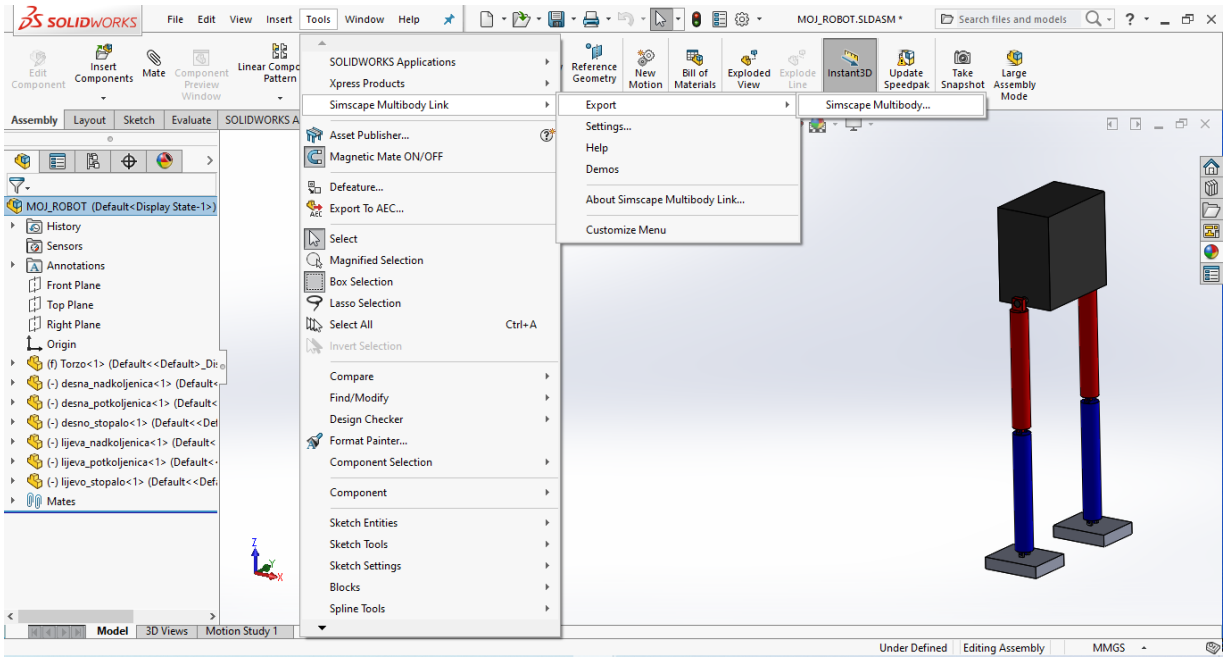
4.2. Prebacivanje modela u Simulink okruženje



Slika 11 Gotovi sklop (eng. assembly) modela dvonožnog robota

Slika 11 prikazuje gotovi sklop (eng. assembly) modela dvonožnog robota. Pri izradi sklopa bilo je potrebno definirati međusobne odnose pojedinih dijelova sklopa te 6 rotacijskih zglobova. Ovaj model je za svrhu primjene učenja s pojačanjem (eng. Reinforcement Learning) bilo potrebno prebaciti u Matlab, odnosno u Simulink okruženje. Zbog toga je model prebačen u **.xml** datoteku pomoću programskog paketa "**Simscape Multibody Link**", koji je potrebno instalirati kao dodatak **Matlabu**.

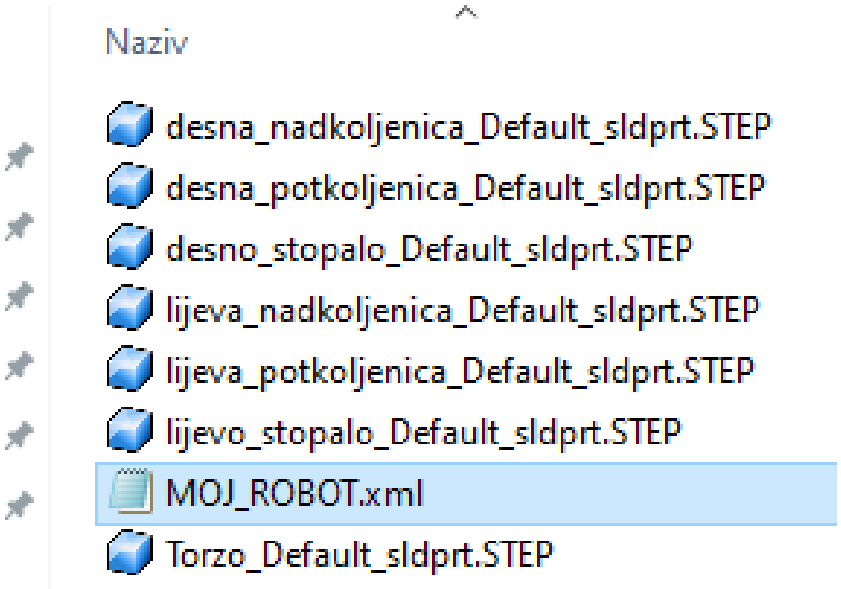
Nakon instalacije, potrebno ga je povezati sa **SolidWorks**-om, a to se postiže upisom naredbe "smlink_linksw " u Matlabu. Na **slici 12** je prikazan postupak prebacivanja sklopa u .xml datoteku.



Slika 12 postupak prebacivanja sklopa u .xml datoteku

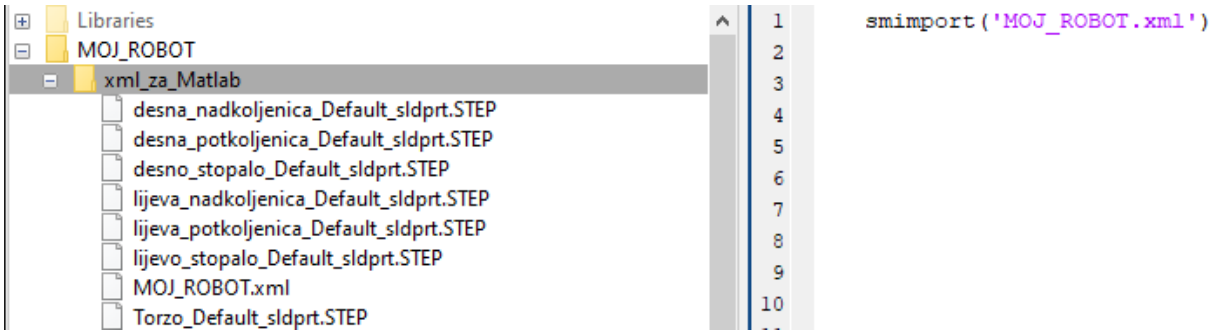
Nakon niza izbora iz padajućih izbornika u SolidWorksu (kako je prikazano na gornjoj slici), započinje automatsko generiranje **.xml** datoteke i pripadnih potrebnih **.STEP** datoteka dijelova modela robota.

Na **slici 13** su prikazane dobivene datoteke (.STEP i .xml datoteke)



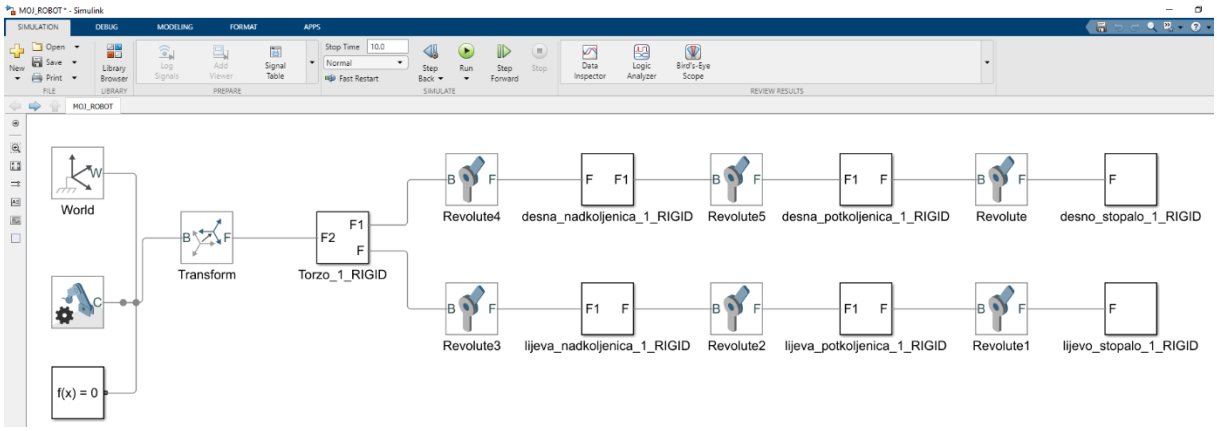
Slika 13 dobivene datoteke (.STEP i .xml datoteke)

Zatim je **.xml** datoteka otvorena u Matlabu pomoću funkcije **smimport()**, kao što je prikazano na **slici 14**. (Napomena: da bi to bilo moguće, potrebno je u Matlabu instalirati **Simscape Multibody** toolbox!).

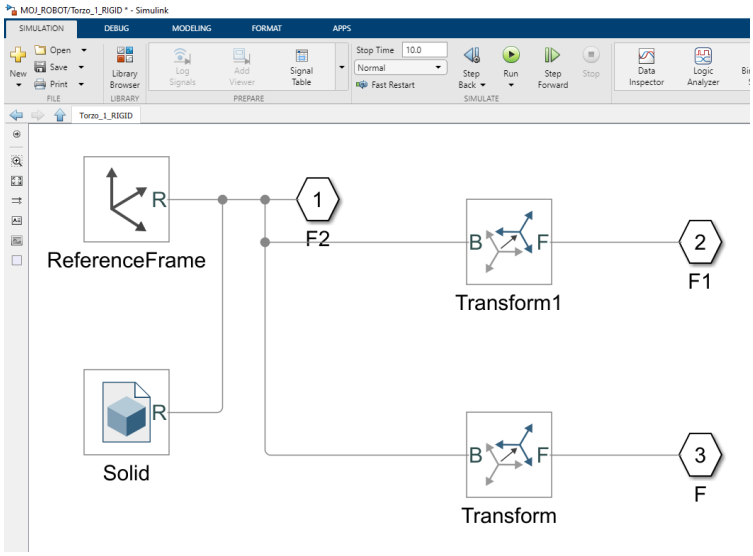


Slika 14 Otvaranje .xml datoteke u Matlabu

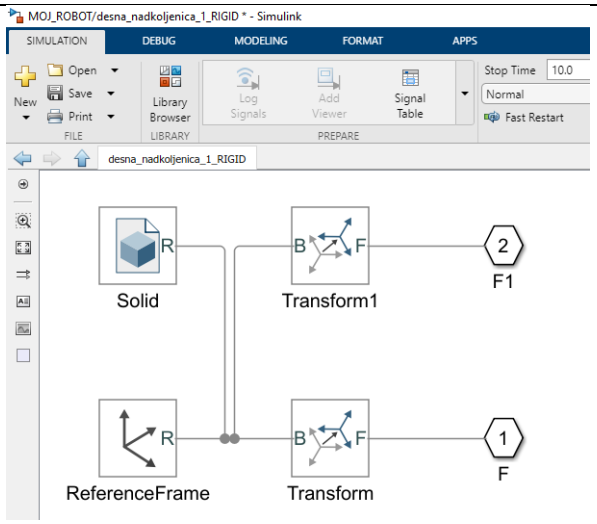
Slike 15, 16, 17, 18 i 19 prikazuju dijelove modela robota u **Simulink** okruženju.



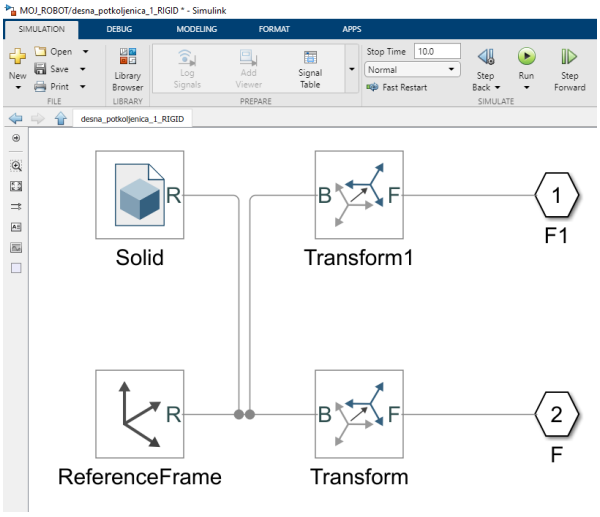
Slika 15 Blokovski prikaz osnovnog modela robota u simulinku



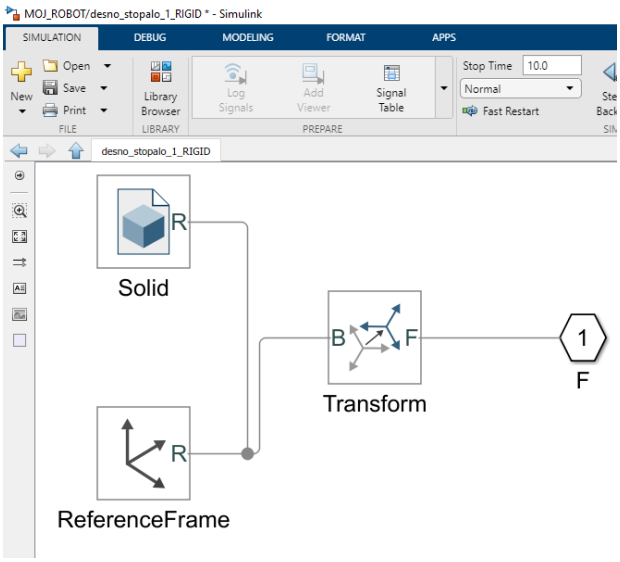
Slika 16 Podsustav "Torzo_1_RIGID"



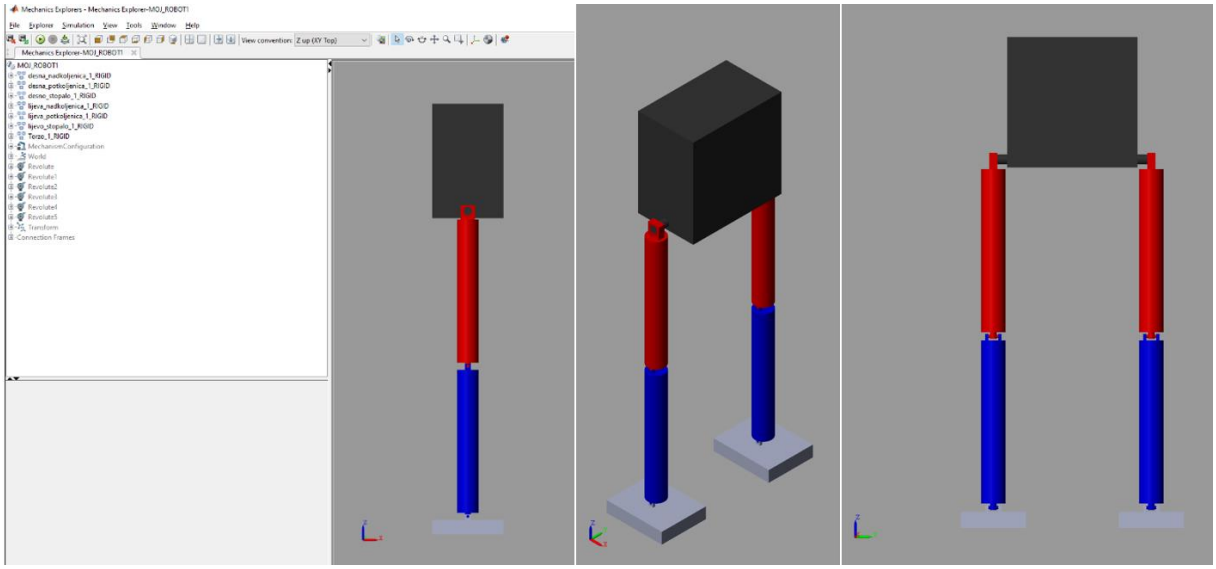
Slika 17 Podsustav "desna_nadkoljenica_1_RIGID"



Slika 18 Podsustav "desna_potkoljenica_1_RIGID"



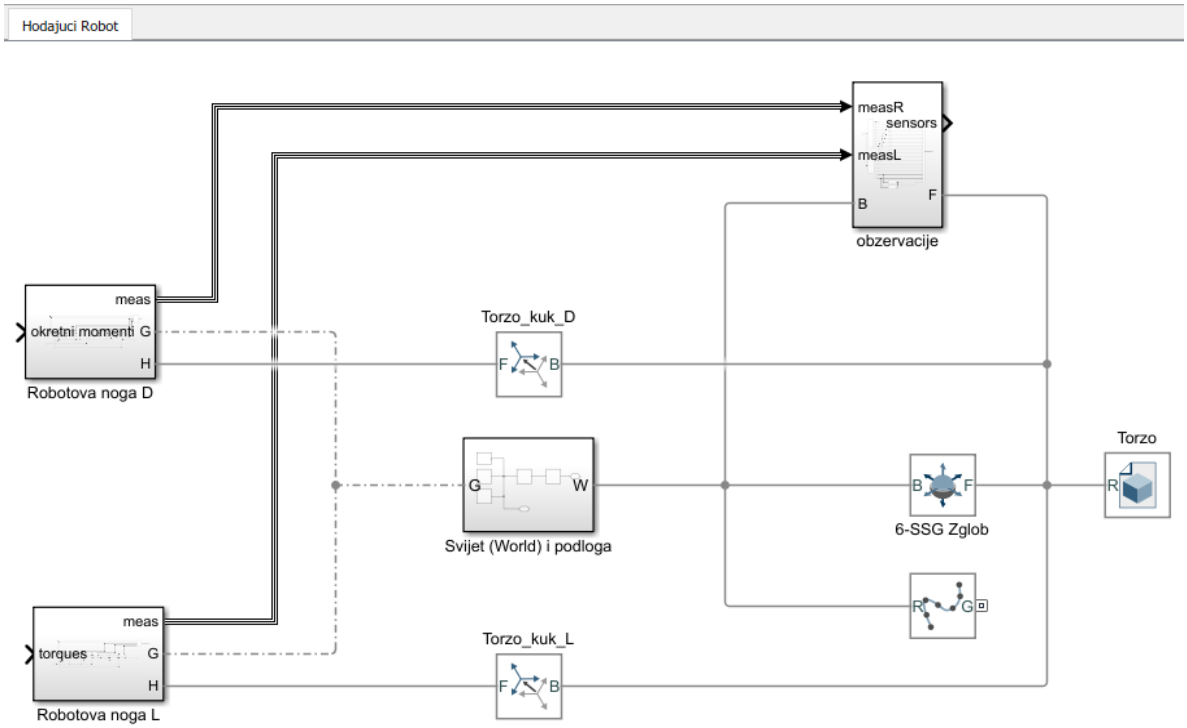
Slika 19 Podsustav "desno_stopalo _1_rigid"



Slika 20 osnovni model robota unutar Simulink okruženja

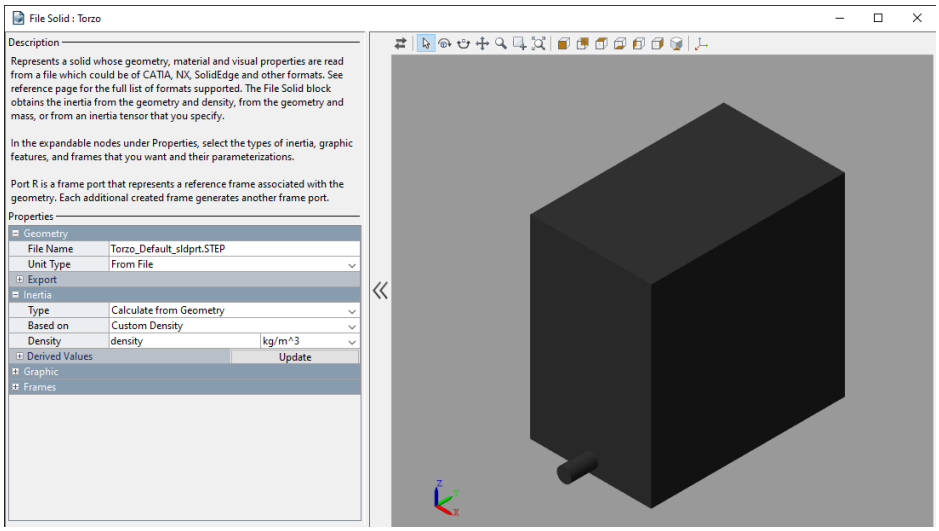
Time je dobiven osnovni model robota (modeliranog u SolidWorksu) unutar Simulink okruženja (dio Matlaba!). **Slika 20** prikazuje taj model. Međutim, taj model je potrebno prilagoditi za implementaciju **učenja pojačanjem**.

4.3. Prilagodba modela za implementaciju učenja s pojačanjem



Slika 21 Blokovski prikaz izmjenjenog modela robota

Slika 21 prikazuje izgled modela robota koji je izmijenjen na način da bude prikladniji za primjenu učenja s pojačanjem. Vidljivo je da je izdvojen blok Torzo (prikazan na **slici 22**), te da su lijeva i desna noga robota odvojene u zasebne podsustave.

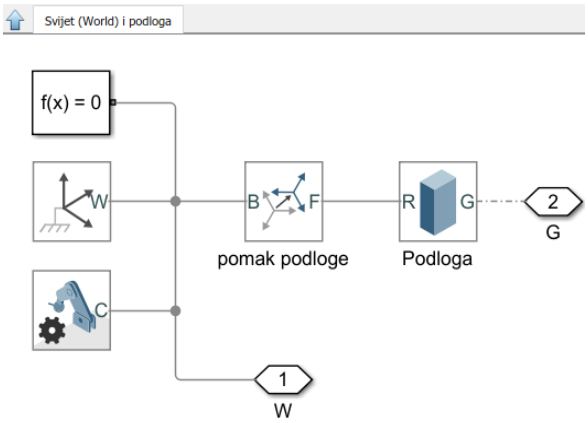


Slika 22 Torzo modela robota u Simulinku

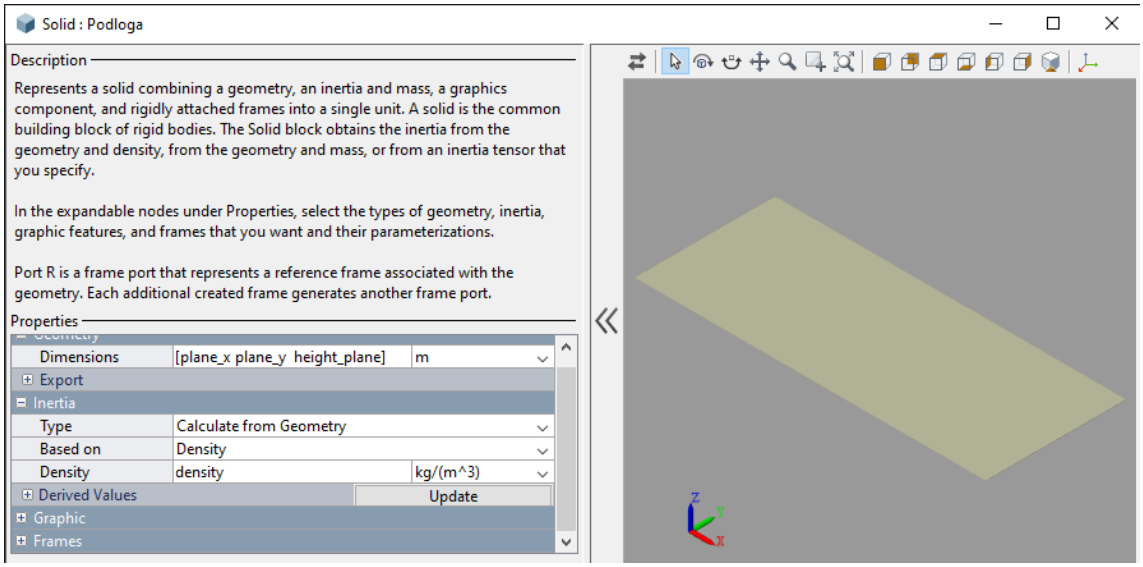
Osnovnom modelu je dodan podsustav pod nazivom "Svijet(World) i podloga". Taj je podsustav detaljno prikazan na **slici 23**. Važno je napomenuti da je blok "World Frame" (blok u kojem piše slovo "W") dodan iz Simulink knjižnice, dakle to nije kopirani blok koji se vidi na **slici** koja prikazuje osnovni model robota iz **.xml** datoteke (taj blok je izbrisan!).

Razlog je taj što je "World Frame" blok osnovni okvir na koji se vežu svi ostali dijelovi Simulink modela. Ako bi se koristio blok iz osnovnog modela, došlo bi do problema tokom simulacije odnosno učenja hodanja!

U bloku "Podloga" su definirani geometrija, izgled i položaj podloge po kojoj će model robota učiti hodati.



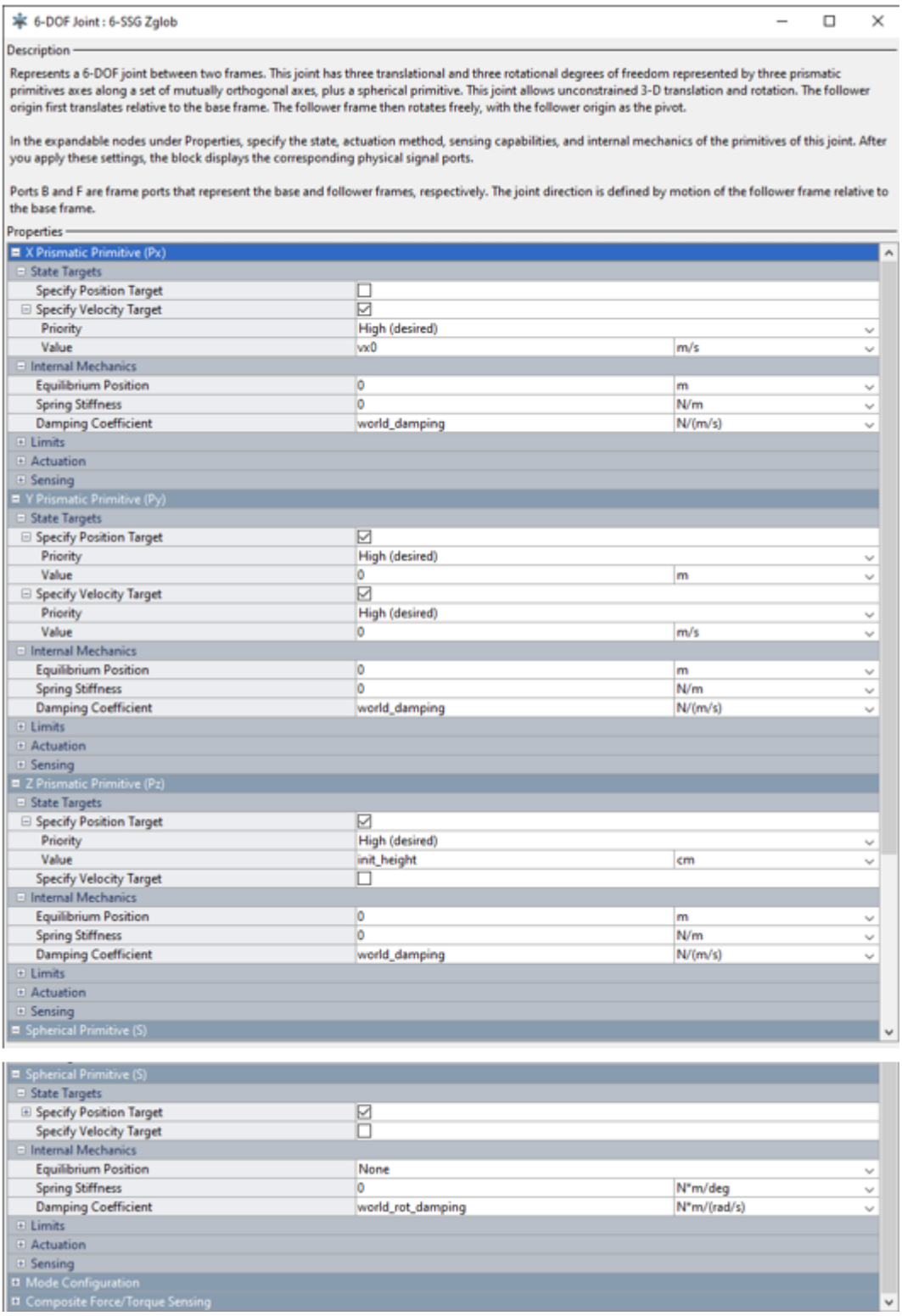
Slika 23 Podsustav "Svijet (World) i podloga"



Slika 24 Blok "Podloga" u Simulinku

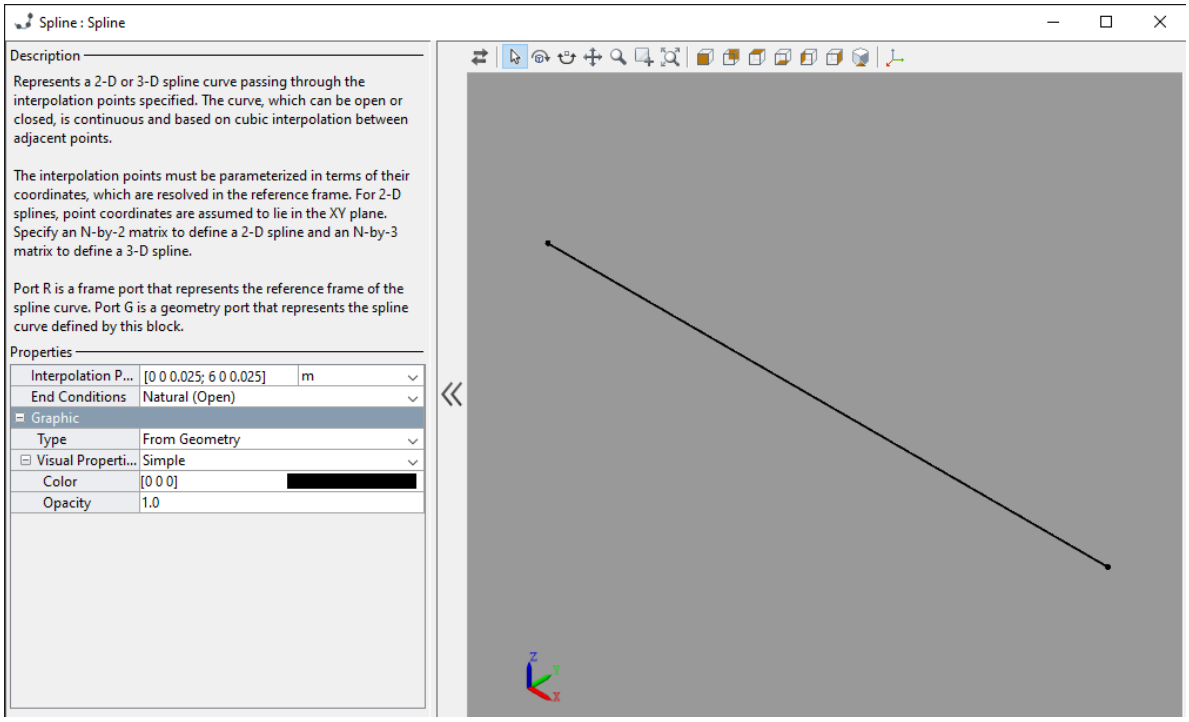
Slika 24 prikazuje blok "podloga", gdje se vide podaci za geometriju, inerciju i izgled podloge.

Dodan je i blok "6SSG-Zglob", koji predstavlja kuglični zglob sa 6 stupnjeva slobode gibanja, također iz **Simulink** knjižnice. Taj zglob je povezan sa torzom modela robota, jer je torzu potrebno omogućiti Torzu slobodno kretanje tokom hodanja, dakle torzo ne smije biti fiksiran! **Slika 25** prikazuje parametre bloka "6SSG-Zglob".



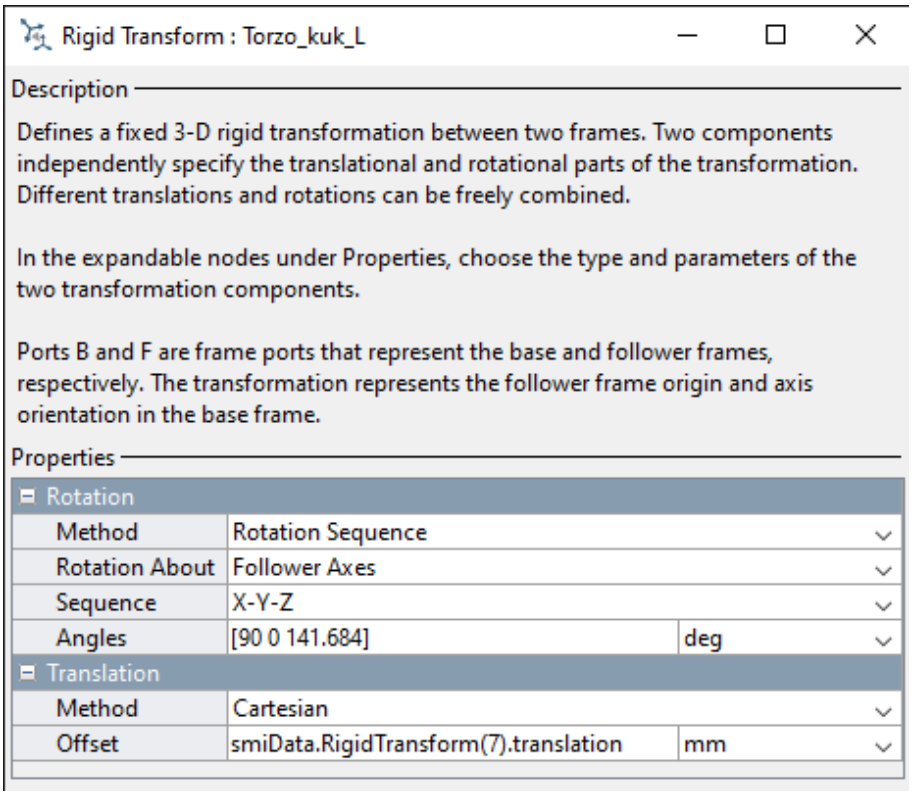
Slika 25 Blok "6 SSG-Zglob"

Radi ljepšeg prikaza pri simulaciji hoda, uz podlogu je dodana i krivulja u pozitivnom smjeru osi X. To je blok koji se nalazi ispod bloka "6SSG-Zglob" u modelu, a podaci za njega prikazani su na **slici 26.**



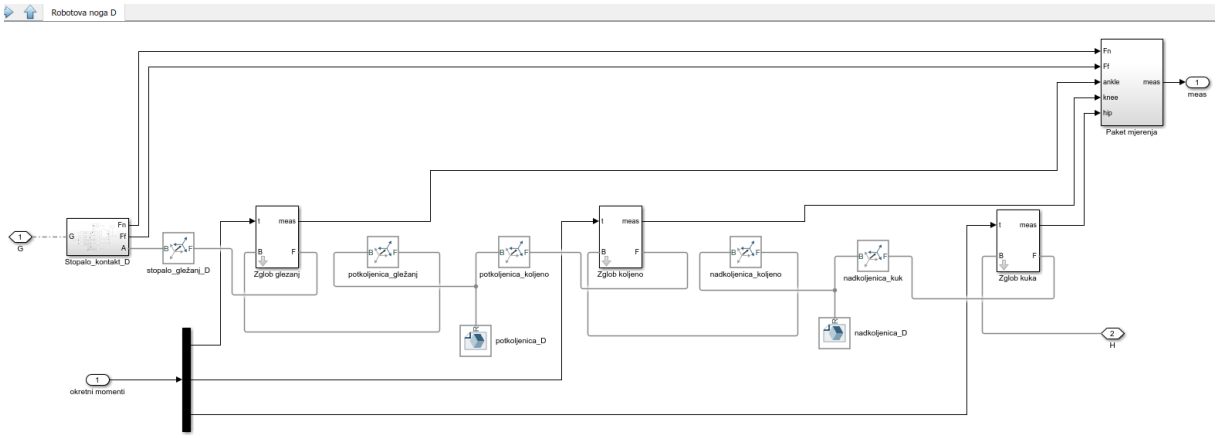
Slika 26 Blok "Spline"

Na novom modelu se nalaze izvučeni blokovi za krutu 3-D transformaciju između koordinatnih sustava. Vidljiva su dva bloka, između torza i desnog kuka, te između torza i lijevog kuka modela robota. Oni su kopirani iz osnovnog modela, pa zadržavaju odnose definirane u SolidWorksu. Za desni kuk nije ništa mijenjano, a za lijevi su napravljene male izmjene u rotaciji, kao što prikazuje **slika 27.** Izmjene su napravljene radi namještanja robota u željeni početni položaj pri učenju hoda, vrijednosti rotacija dobivene su iteracijski namještanjem!

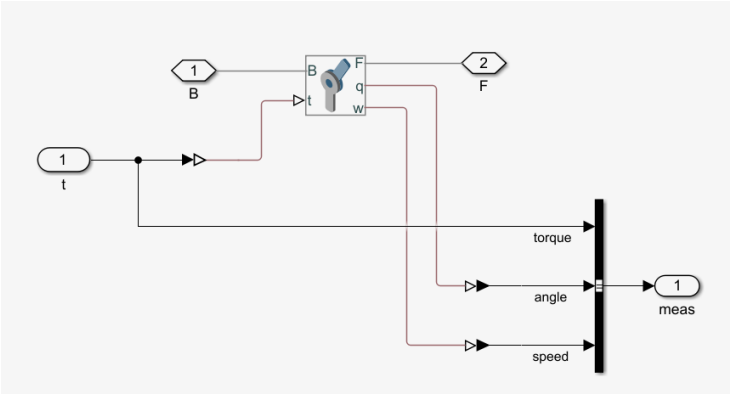


Slika 27 Blok "Torzo_kuk_L"

Slika 28 prikazuje podsustav "Robotova noga D", koji predstavlja izmijenjeni model robotove desne noge. Za zglobove su iskorišteni blokovi pruženi od strane Matlabovog tima koji je izradio određene blokove za primjenu učenja s pojačanjem na dvonožne modele robota. Izgled tih blokova prikazuje **slika 29**. Dakle izbrisani su zglobovi iz osnovnog modela iz **.xml** datoteke. Ovakvi blokovi za zglobove su upravljivi preko okretnog momenta (port "t"), koje će model primati od DDPG agenta u procesu učenja s pojačanjem (no to će se detaljnije objasniti u nastavku). Osim toga, oni imaju izlazni port ("meas") preko kojeg šalju podatke o trenutnom kutu, kutnoj brzini i primjenjenom okretnom momentu na svakom pojedinom zglobu na blok "Paket mjerenja"!

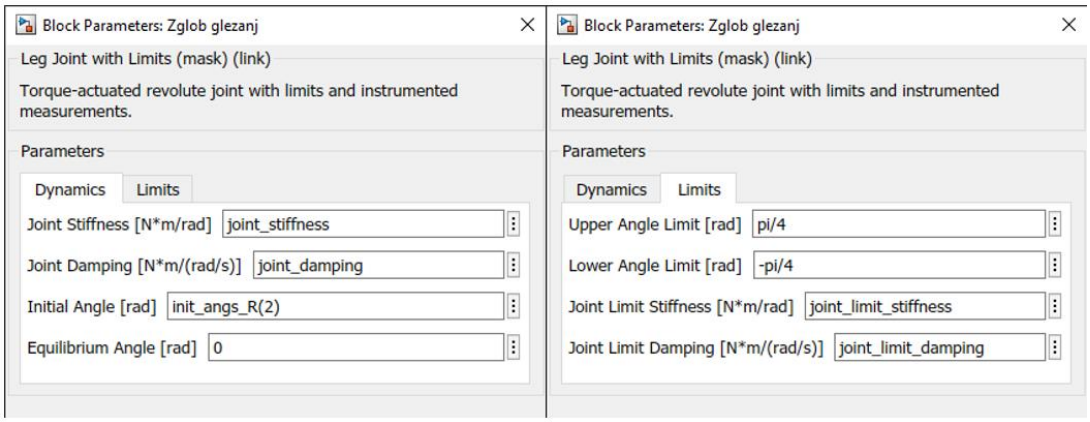


Slika 28 Podsustav "Robotova noga D"

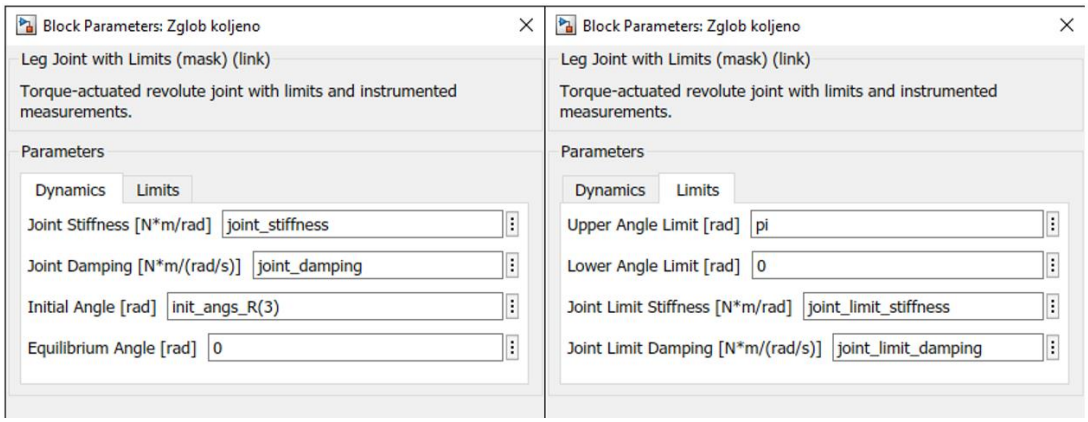


Slika 29 Izgled prilagođenog bloka za zglobove modela robota

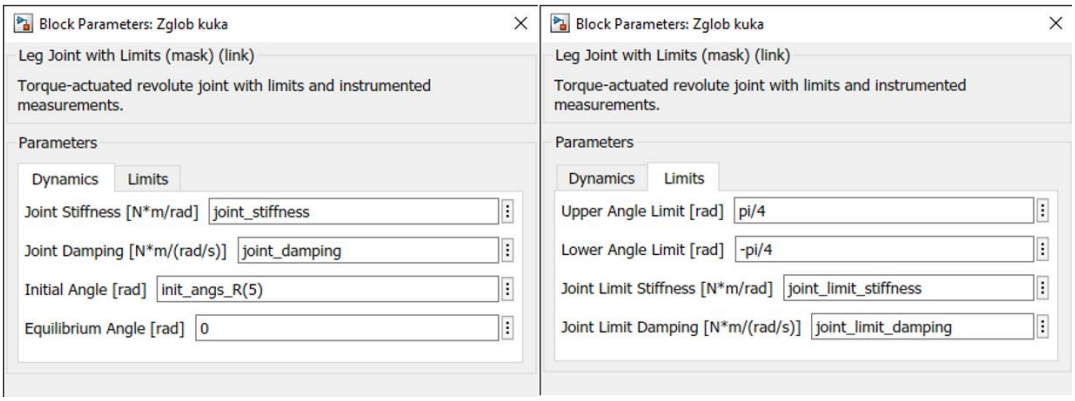
Na **slikama 30, 31** i **32** prikazani su blokovski podaci za pojedine zglobove (gležanj, koljeno i kuk) desne noge modela robota.



Slika 30 Podaci za blok zgloba gležnja

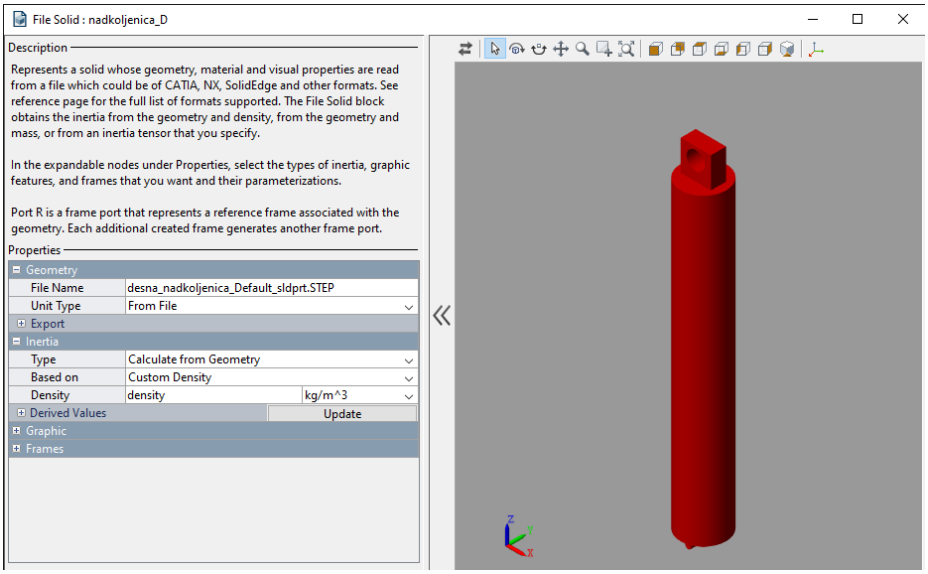


Slika 31 Podaci za blok zgloba koljena



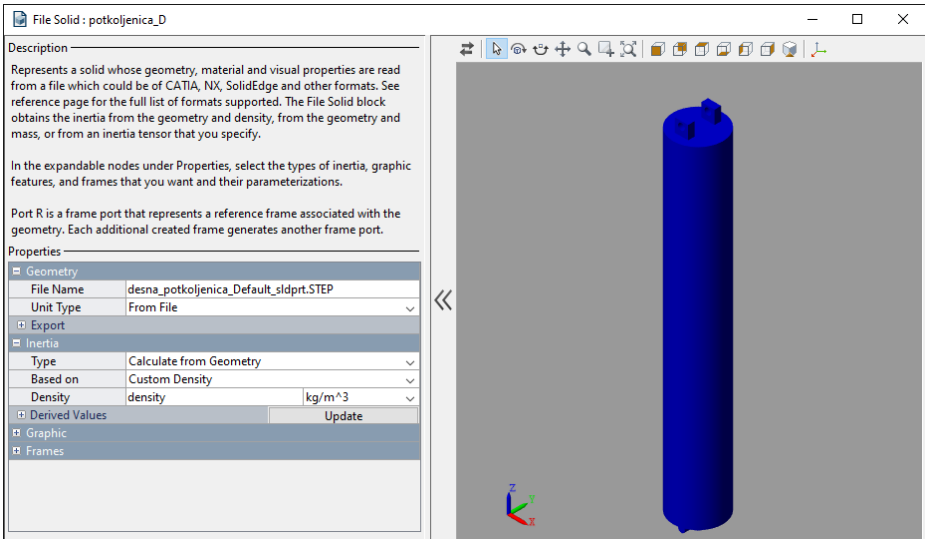
Slika 32 Podaci za blok zgloba kuka

Slika 33 prikazuje izgled bloka “nadkoljenica_D“. To je blok iz osnovnog modela s malim izmjenama u podacima za inerciju koji se vide na donjoj slici.



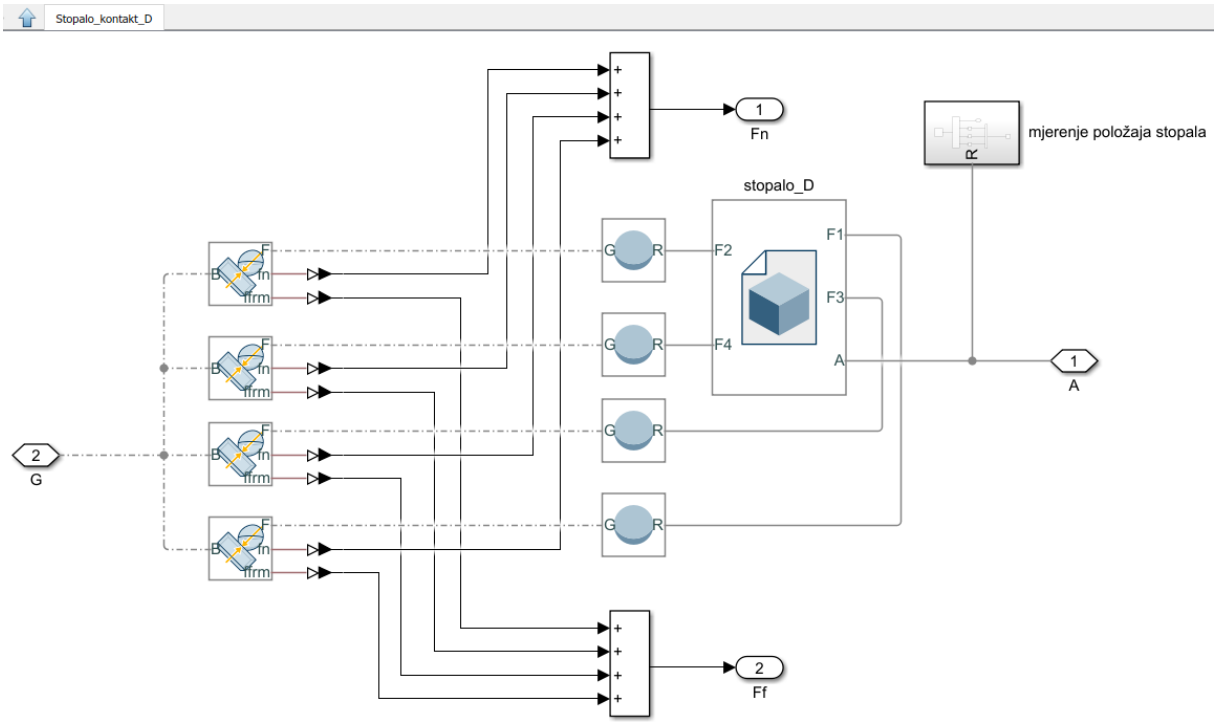
Slika 33 blok “nadkoljenica_D“ u Simulinku

Slika 34 prikazuje izgled bloka “potkoljenica_D“. To je blok iz osnovnog modela s malim izmjenama u podacima za inerciju koji se vide na donjoj slici.



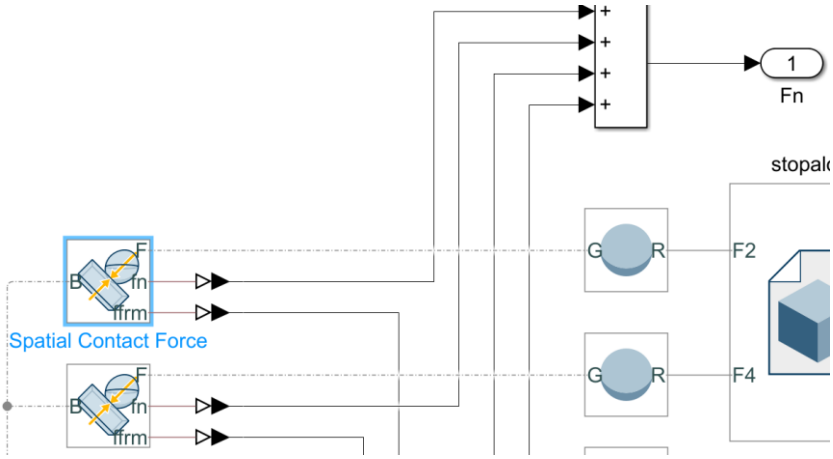
Slika 34 blok “potkoljenica_D“ u Simulinku

Slika 35 prikazuje podsustav "Stopalo_kontakt_D". Ovaj podsustav služi za realizaciju hodanja, odnosno uspostavu kontakta između robota i podloge. Prikazuje implementaciju "Spatial Contact Force" bloka iz **Simulink** knjižnice, pomoću kojeg je moguće modelirati kontakt između dvije površine te definirati parametre tog kontakta. U primjeru hodajućeg robota kontakt koji se modelira je na dodiru stopala robota i podloge po kojoj robot hoda. U ovom podsustavu vidljivo je da su četiri "Spatial Contact Force" bloka povezana sa četiri kuglična bloka. Četiri kuglice su postavljene u četiri kuta modela stopala robota, čime je realiziran kontakt u četiri točke dodira između stopala robota i podloge po kojoj robot hoda.



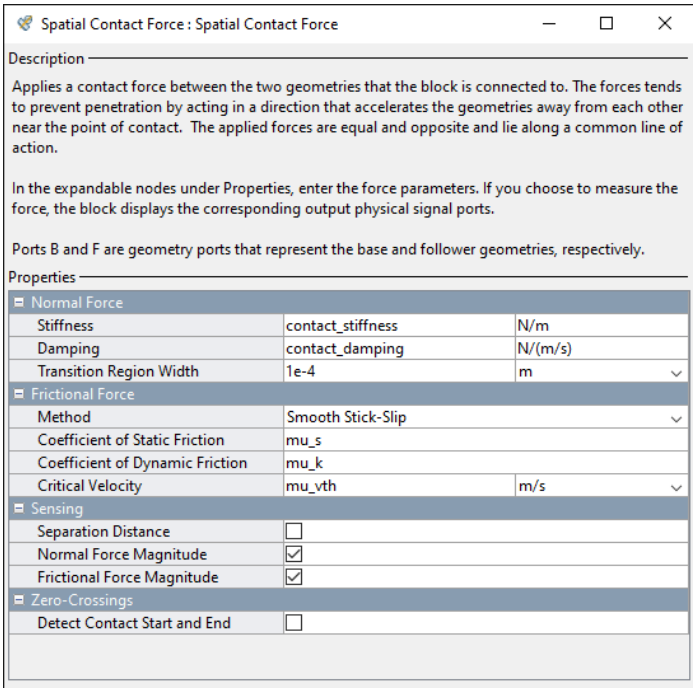
Slika 35 Podsustav "Stopalo_kontakt_D"

Slika 36 daje bolji prikaz spomenutog "Spatial Contact Force" bloka za modeliranje kontakta. Ovdje je vidljivo da se koordinatni sustav svakog pojedinog bloka spaja sa koordinatnim sustavom kugličnog bloka te da se šalju podaci o normalnoj sili (fn) i sili trenja (ffrm) s mjesta dodira površine kugle i podloge na sumator. Sumator zbraja te sile sa 4 kugle na stopalu i daje izlazni podatak o ukupnoj sili dodira stopala s podlogom.



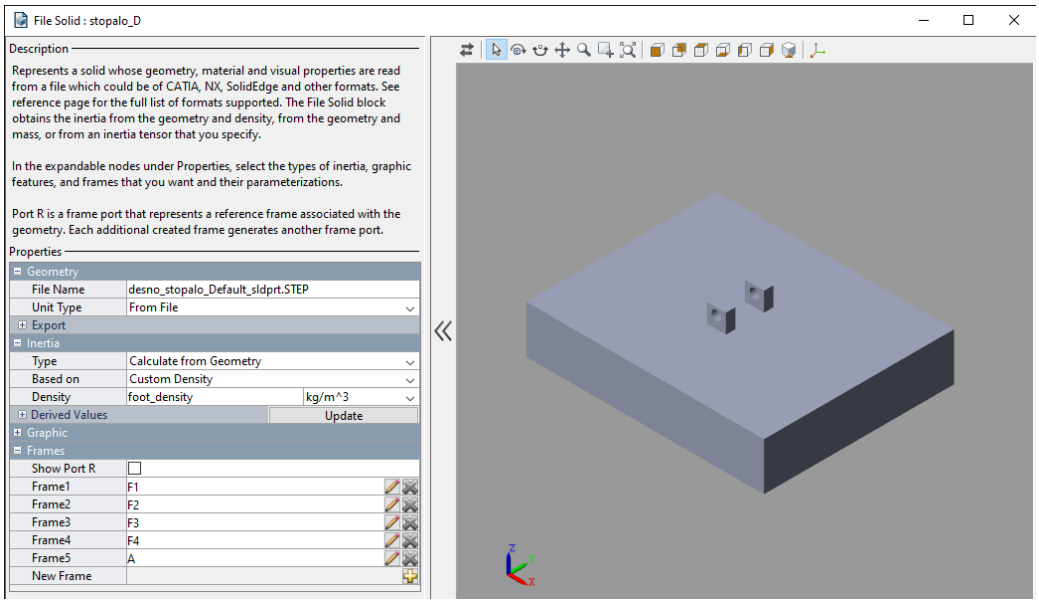
Slika 36 Bolji prikaz "Spatial Contact Force" bloka za modeliranje kontakta

Slika 37 pokazuje podatke za definiranje kontakta unutar "Spatial Contact Force" bloka. Podaci su isti za sva četiri "Spatial Contact Force" bloka.



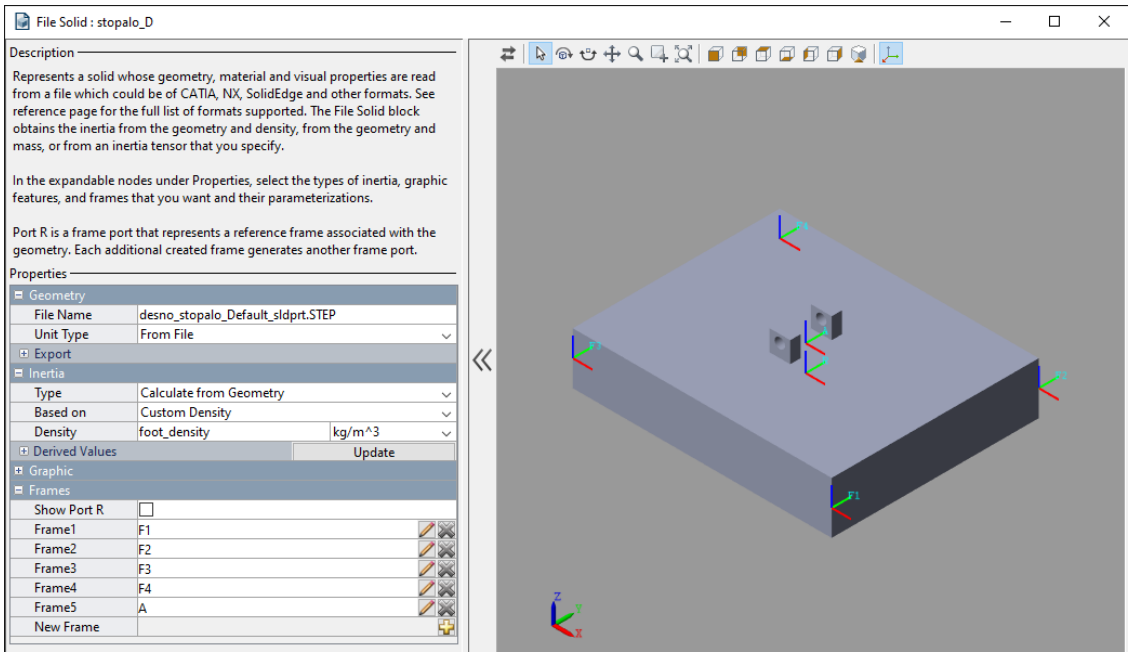
Slika 37 podaci za definiranje kontakta unutar "Spatial Contact Force" bloka

Slika 38 prikazuje blok "stopalo_D", u kojem se vidi geometrija stopala te podaci za inerciju. Osim toga, vidi se i popis definiranih koordinatnih sustava koji predstavljaju pozicije kuglica koje detektiraju dodir s podlogom.



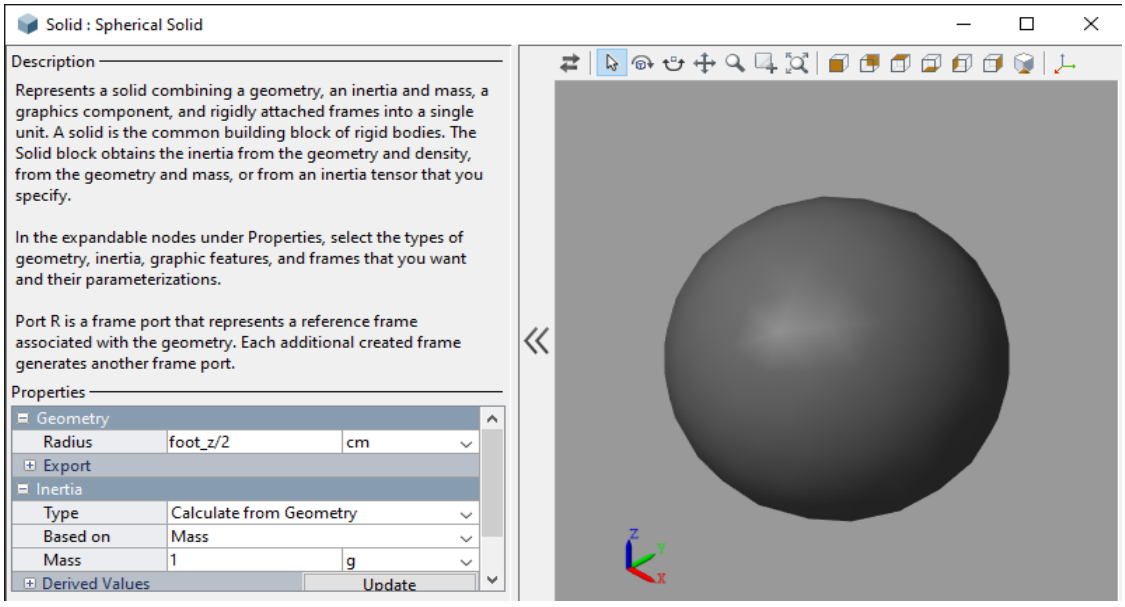
Slika 38 blok "stopalo_D" u Simulinku

Slika 39 daje vizualizaciju definiranih koordinatnih sustava za kuglice ("F1", "F2", "F3" i "F4"), kao i koordinatnog sustava "A" pomoću kojeg se definira veza između stopala i zgloba gležnja (spoj stopala i potkoljenice).



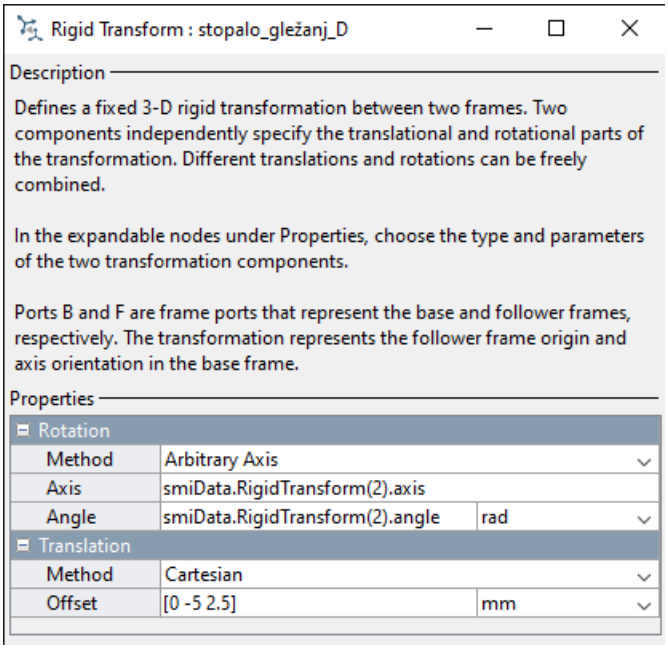
Slika 39 Vizualizacija definiranih koordinatnih sustava za kuglice

Slika 40 prikazuje blok "Spherical Solid", odnosno kuglični blok, gdje se vide podaci za geometriju i inerciju kao i izgled same kuglice.



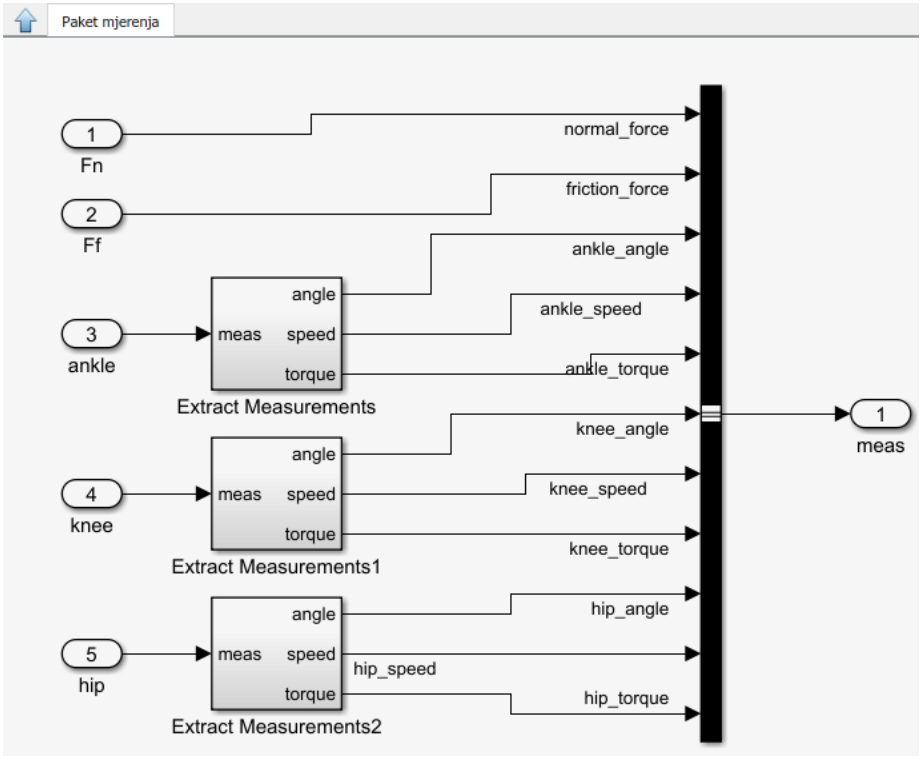
Slika 40 blok "Spherical Solid"

Na **slici 41** je prikazan blok "stopalo_glezanj_D" u kojem se vide podaci za fiksnu 3-D transformaciju između koordinatnih sustava desnog stopala i gležnja. Rotacija je ostala nepromijenjena (preuzeta iz osnovnog modela robota, definirana na sklopu u SolidWorksu), a translacija je prilagođena iterativno kako bi se robot postavio u željeni početni položaj za učenje hodanja.



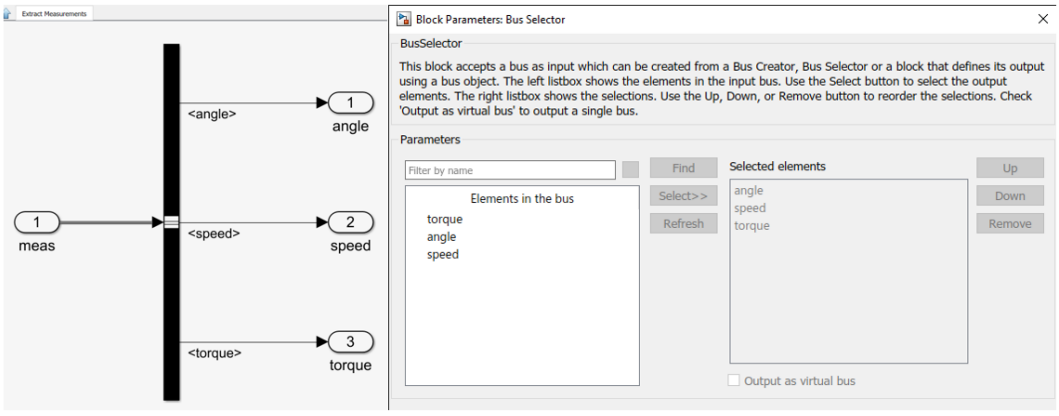
Slika 41 Blok "stopalo_glezanj_D"

Slika 42 prikazuje blok "Paket mjerenja" na koji se šalju podaci o kontaktu (ukupna normalna sila i sila trenja između stopala i podloge), te podaci iz zglobova (gležanj, koljeno i kuk). Podaci od zglobova dolaze kao paketi od tri podatka sa svakog zgloba (kut, kutna brzina i okretni moment), pa ih je potrebno raspakirati u pojedinačne signale. Za raspakiravanje se koriste blokovi "Extract Measurements".



Slika 42 Blok "Paket mjerenja"

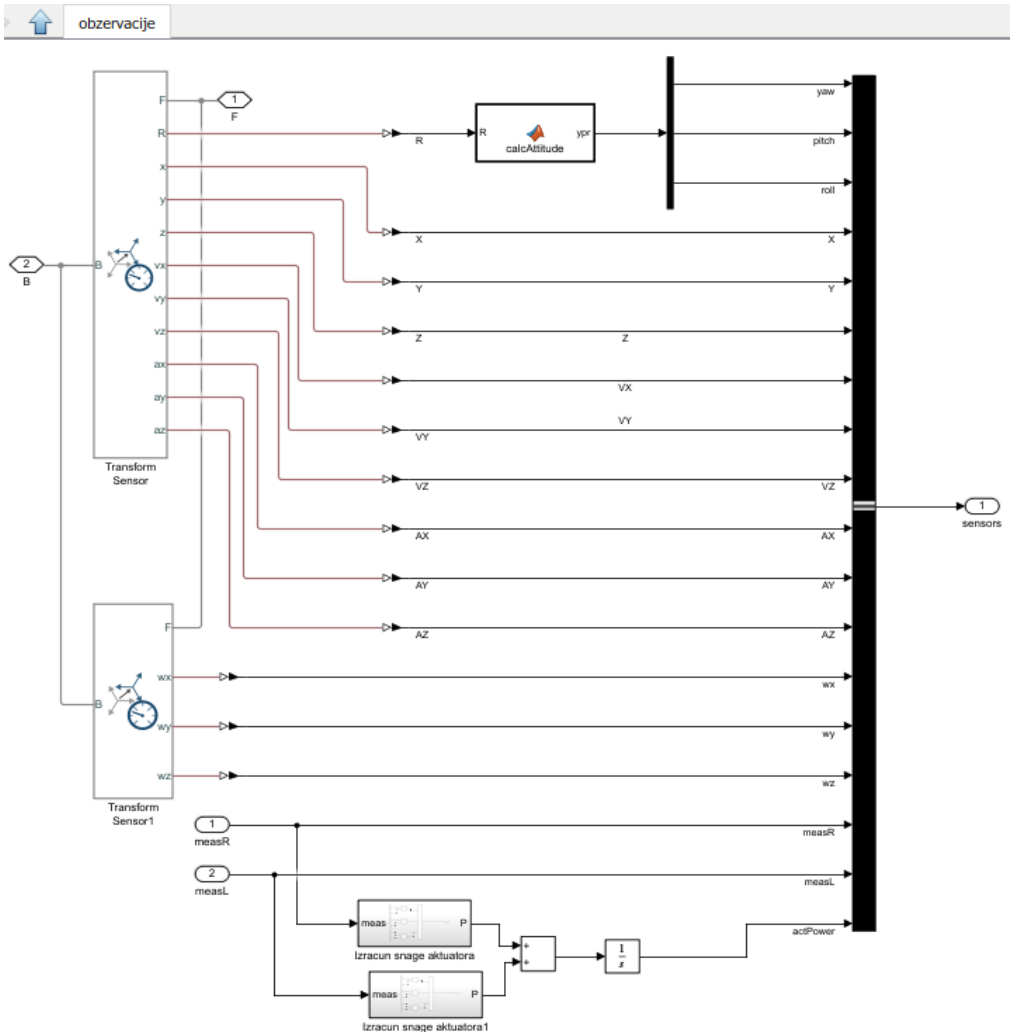
Slika 43 prikazuje sadržaj bloka "Extract Measurements", gdje se vide podatci koji dolaze od zgloba (lijeva strana ("torque", "angle", "speed")), te se s desne strane biraju željeni podaci u željenom redosljedu.



Slika 43 Sadržaj bloka "Extract Measurements"

Ovime su opisani svi dijelovi podsustava "Robotova noga D" prilagođenog modela robota. Podsusutav "Robotova noga L" je u potpunosti analogan pa ga nema potrebe prikazivati i opisivati.

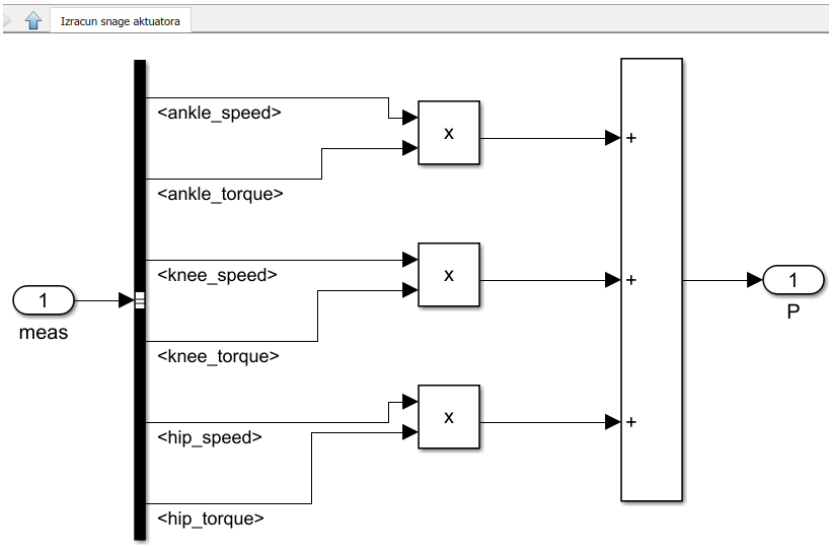
Slika 44 prikazuje blok "obzervacije" u kojem se objedinjuju sva mjerenja modela prilagođenog za učenje s pojačanjem. To su mjerenja iz lijeve i desne noge robota (ukupno **22** podatka, 11 po nozi, v. **slika 42**), te podatci o stanju torza modela robota koji je povezan sa kugličnim zglobom (6 stupnjeva slobode gibanja) koji se izvlače pomoću **Simulink** bloka "Transform sensor". To je još dodatnih **12** podataka (položaj torza po x, y i z-osi, brzina torza u smjerovima x, y i z-osi, ubrzanje torza u smjeru x, y i z-osi, kutna brzina torza oko x, y i z-osi). Osim toga, uzimaju se u obzir još **3** podatka koja se dobivaju pomoću Matlab funkcije "calcAltitude" i **1** podatak o ukupnoj uloženoj snazi za aktuiranje (pokretanje) zglobova obje noge modela robota (v. **slika 45**). Dakle obzervacije se sastoje od ukupno **38** (22+12+3+1=38) podataka o modelu robota.



Slika 44 Blok "obzervacije"

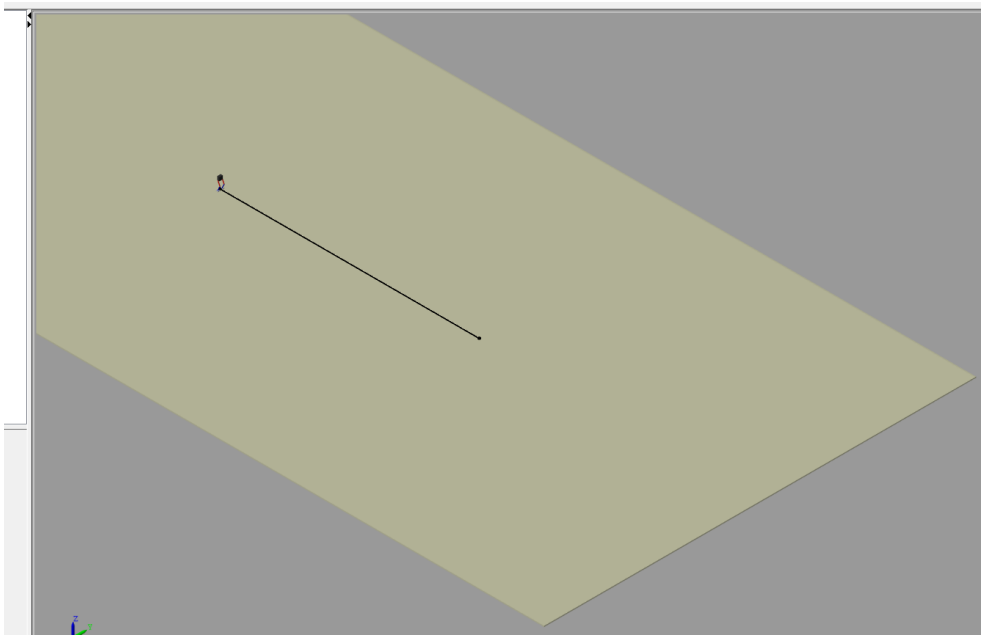
Na **slici 44** se nalazi "MATLAB Function" blok pod nazivom "calcAltitude" koji povezuje simulink model sa slijedećom matlabovom funkcijom:

```
function ypr = calcAttitude(R)
ypr = [atan2(R(2,1),R(1,1)); ...
      atan2(-R(3,1),sqrt(R(3,2)^2 + R(3,3)^2)); ...
      atan2(R(3,2),R(3,3))];
```



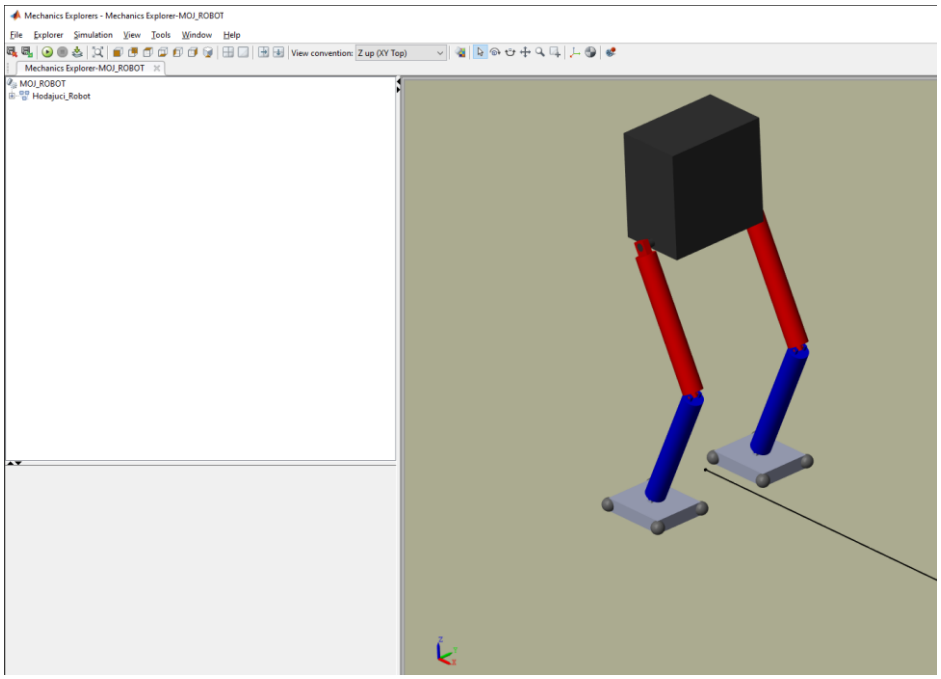
Slika 45 Podsustav "Izracun snage aktuatora"

Slika 46 prikazuje izgled modela robota u početnom položaju na podlozi i na početku krivulje u smijeru x-osi.



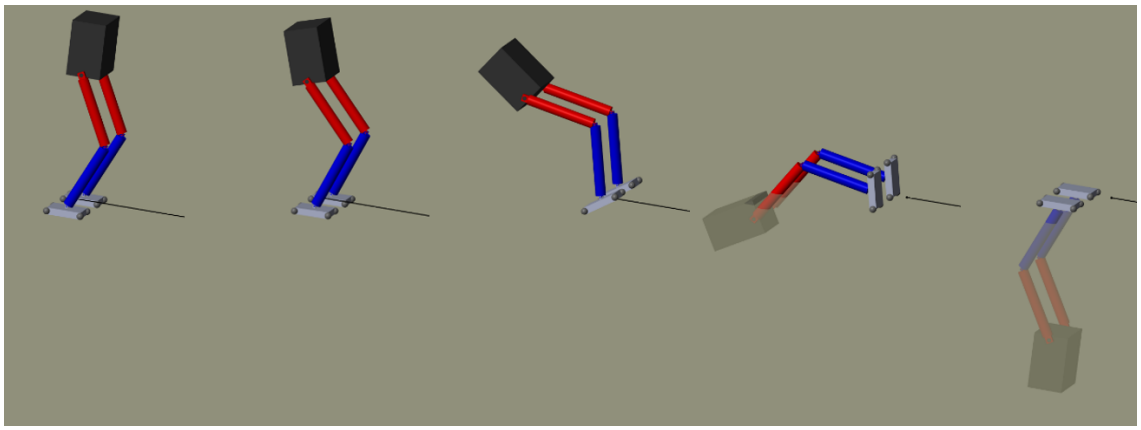
Slika 46 Početni položaj za učenje modela robota

Na **slici 47** je dan uvećani prikaz modela robota u početnom položaju. Vidljivi su položaji kuglica na stopalima robota koje su povezane sa "Spatial Contact Force" blokovima i preko kojih je ostvarena detekcija dodira stopala sa podlogom čime je omogućena simulacija hodanja.



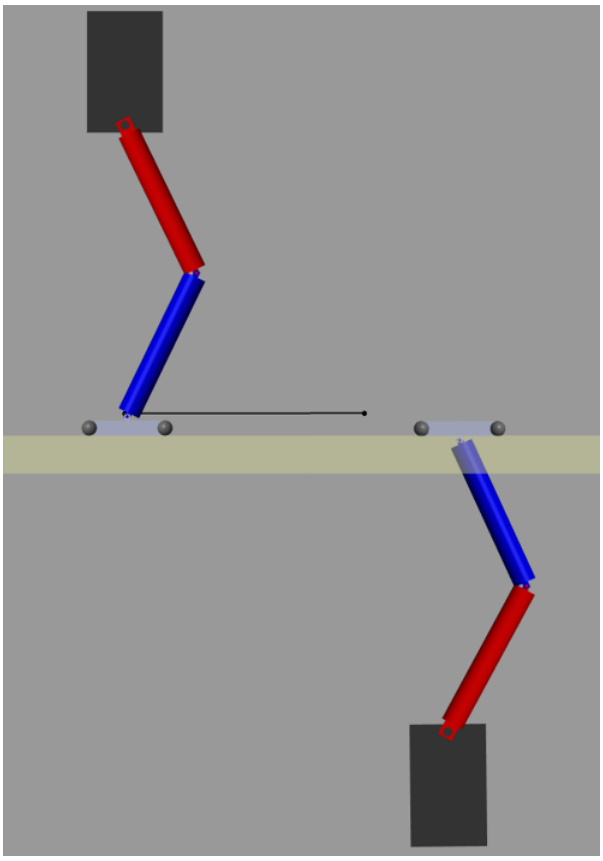
Slika 47 uvećani prikaz modela robota u početnom položaju

Slika 48 prikazuje ponašanje modela robota pri pokretanju simulacije u **Simulink** okruženju. Na robota djeluje simulirana gravitacijska sila, te s obzirom da zglobovi robota nisu ničime upravljani, robot s vremenom padne. S obzirum da je definiran kontakt stopala s podlogom, robot ostane u "visećem položaju". Kontakt nije definiran na drugim dijelovima modela robota, pa torzo i noge propadnu kroz podlogu.



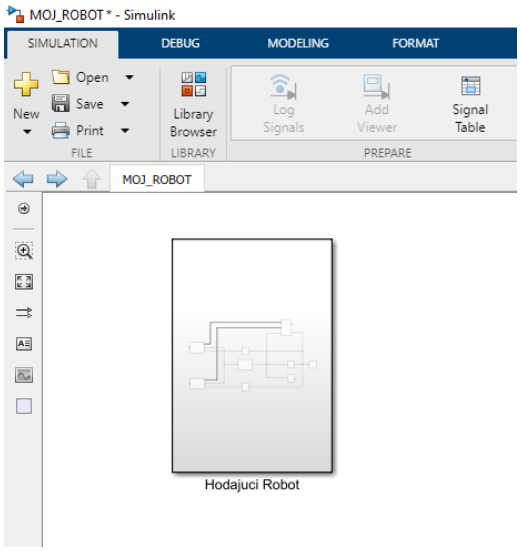
Slika 48 Ponašanje modela robota bez upravljanja pri pokretanju simulacije

Slika 49 pokazuje početni i krajnji položaj modela robota pri simulaciji bez upravljanja.



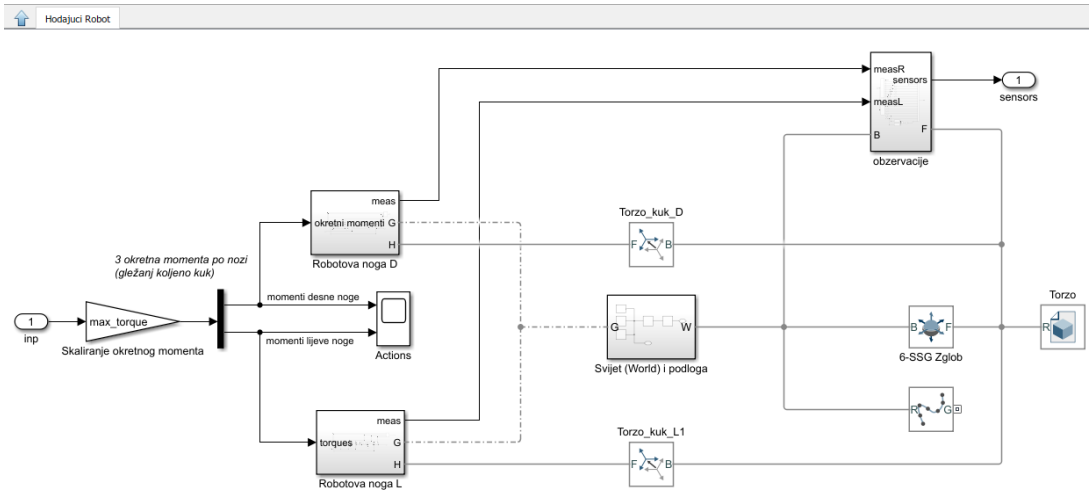
Slika 49 Početni i krajnji položaj modela robota pri simulaciji bez upravljanja

Sada je potrebno pronaći zakon upravljanja koji će robotu omogućiti hodanje po podlozi bez padova i sa što manjim odstupanjem od prikazane crne linije. Taj problem hodanja mobilnih robota je vrlo složen za rješavanje klasičnim analitičkim metodama upravljanja, a u ovom radu je riješen pomoću metode strojnog učenja pod nazivom "učenje s pojačanjem".



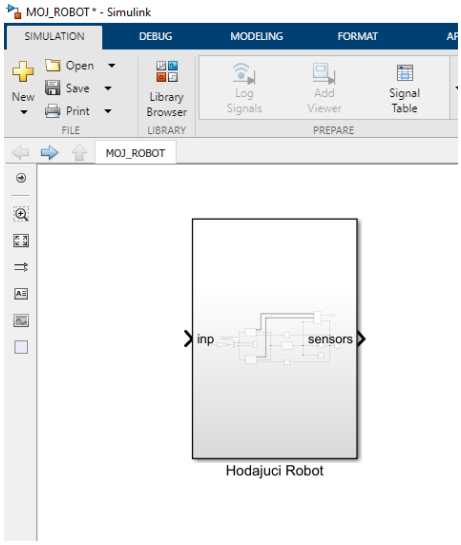
Slika 50 Trenutni izgled modela hodajućeg robota kompresiranog u jedan blok

Slika 50 prikazuje trenutni izgled modela hodajućeg robota kompresiranog u jedan blok. Vidljivo je da blok nema ulazne niti izlazne portove za komunikaciju s upravljačkim sustavom. Dakle, za implementaciju učenja s pojačanjem, potrebne su još neke prilagodbe. **Slika 51** prikazuje nadogradnju na prethodni model. Dodani su ulazni i izlazni portovi za komunikaciju s upravljačkim sustavom, te je ulazni port skaliran (pošto će akcije poslane od strane Agenta kod učenja s pojačanjem biti ograničene na vrijednosti između -1 i 1!). Osim skaliranja, ulaz će biti vektor od 6 elemenata (okretni momenti za pokretanje 6 zglobova modela robota), pa ih je potrebno razdvojiti u dva vektora po tri elementa za lijevu i desnu nogu robota!



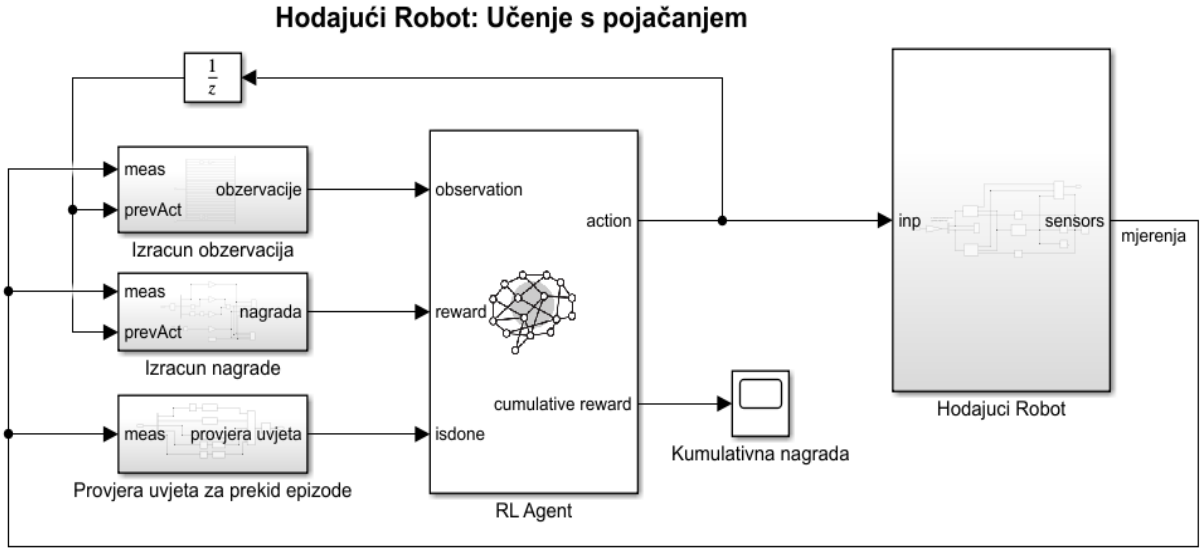
Slika 51 Dodavanje komunikacijskih portova prethodnom modelu robota

Slika 52 prikazuje prikazuje izgled prilagođenog modela hodajućeg robota kompresiranog u jedan blok. Sada je vidljivo da model robota može komunicirati s vanjskim upravljačkim sustavom.



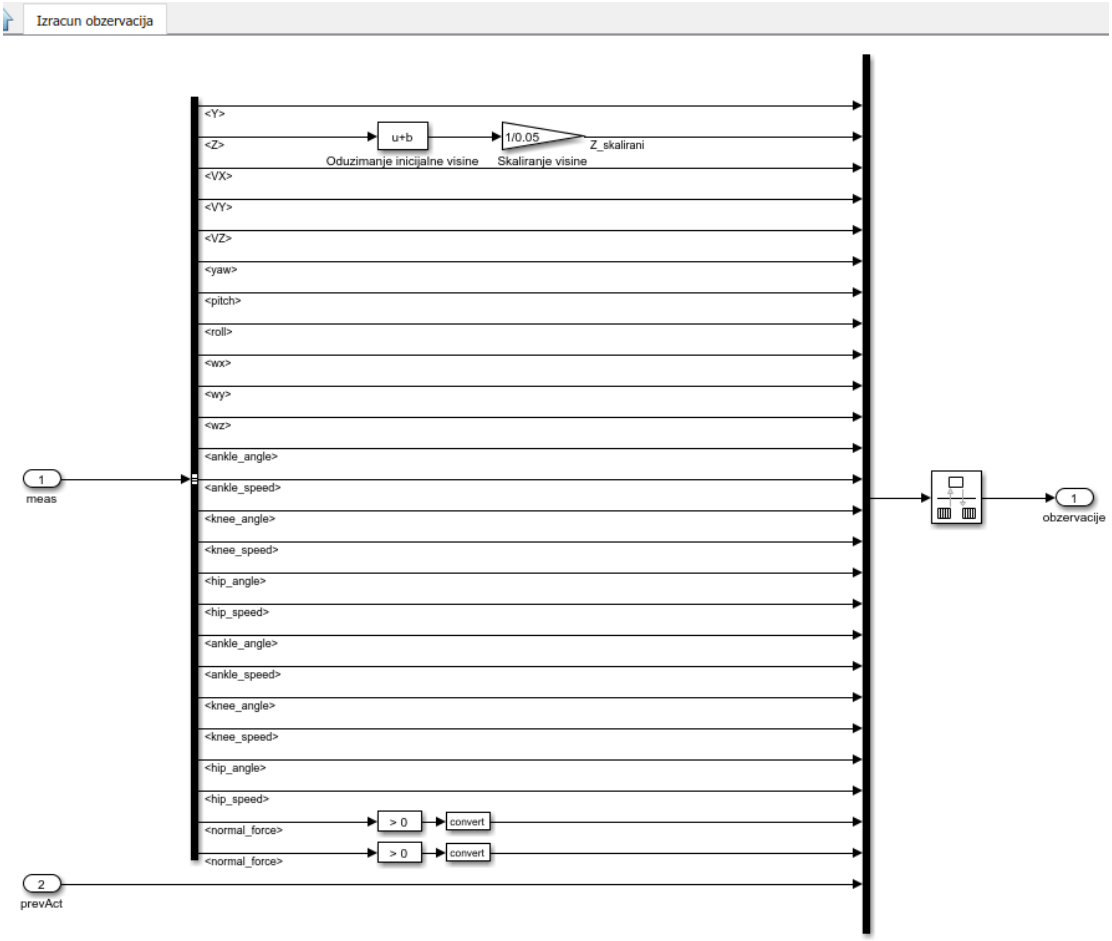
Slika 52 Prilagođeni modela hodajućeg robota kompresiranog u jedan blok

Slika 53 prikazuje izgled upravljačkog sustava svojstvenog učenju s pojačanjem. Ovakvi se sustavi općenito sastoje od RL (eng. Reinforcement Learning) Agenta koji se sastoji od umjetnih neuronskih mreža čiji broj ovisi o vrsti agenta. U ovom radu korišten je DDPG (eng. Deep Deterministic Policy Gradient) Agent, koji se sastoji od dvije umjetne neuronske mreže ("Critic" i "Actor"). Strukture mreža odabire korisnik. Osim Agenta, ovakvi sustavi moraju imati i okolinu (eng. Environment) u kojoj Agent djeluje i nastoji naučiti izvršavati određeni zadatak. Na ovom primjeru okolinu predstavlja model hodajućeg robota. Također se moraju definirati načini komunikacije između Agenta i okoline, a za to služe obzervacije (iz okoline) i akcije (od Agenta). Donja slika prikazuje izgled implementiranog sustava za učenje s pojačanjem, koji služi za upravljanje hodom modela robota (spremljen pod nazivom "MOJ_ROBOT.slx")



Slika 53 Model za implementaciju učenja s pojačanjem

Slika 54 prikazuje podsustav "Izracun obzervacija", gdje se od ukupno 38 podataka iz modela robota izabire **25** podataka za konačne obzervacije iz kojih će se unutar RL Agenta tražiti upravljački zakon za hod robota. Neke od izabranih obzervacija se dodatno obrađuju. Od trenutnog položaja torza modela po z-osi se oduzima početni položaj torza po z-osi, jer je za učenje hoda bitno odstupanje od početnog položaja (cilj je što manje odstupanje po z-osi), te se ta razlika skalira kako bi imala veći utjecaj na ponašanje robota. Osim toga, u obzervacije se dodaju i akcije koje je RL Agent generirao u prethodnom koraku učenja, to je još **6** podataka. Dakle, ukupno se RL Agentu kontinuirano šalje **31** (25+6) obzervacija tokom procesa učenja s pojačanjem.

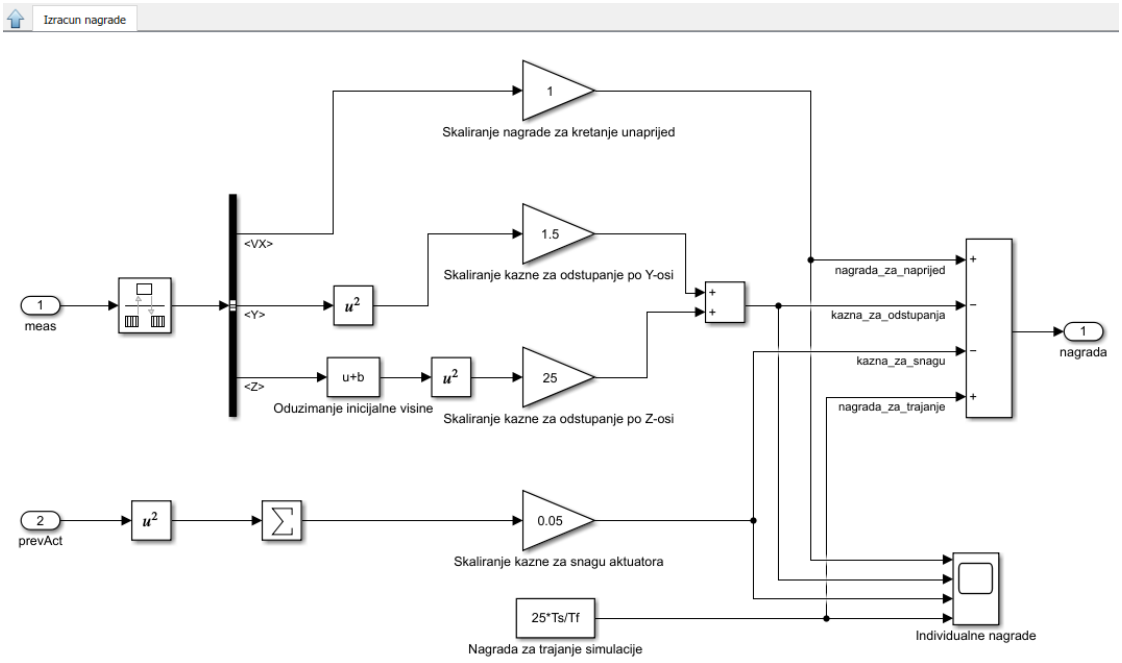


Slika 54 Podsustav "Izracun obzervacija"

Slika 55 prikazuje podsustav "Izracun nagrade", gdje se formulira funkcija nagrade, odnosno računa nagradni signal. Za ovaj primjer hodajućeg robota funkcija nagrade u matematičkom obliku glasi: $r_t = v_x + 0.0625 - 25 * \hat{z}^2 - 0.05 * \sum_i (u_{t-1}^i)^2 - 3 * y^2$, gdje je $\hat{z} = z - z_0$

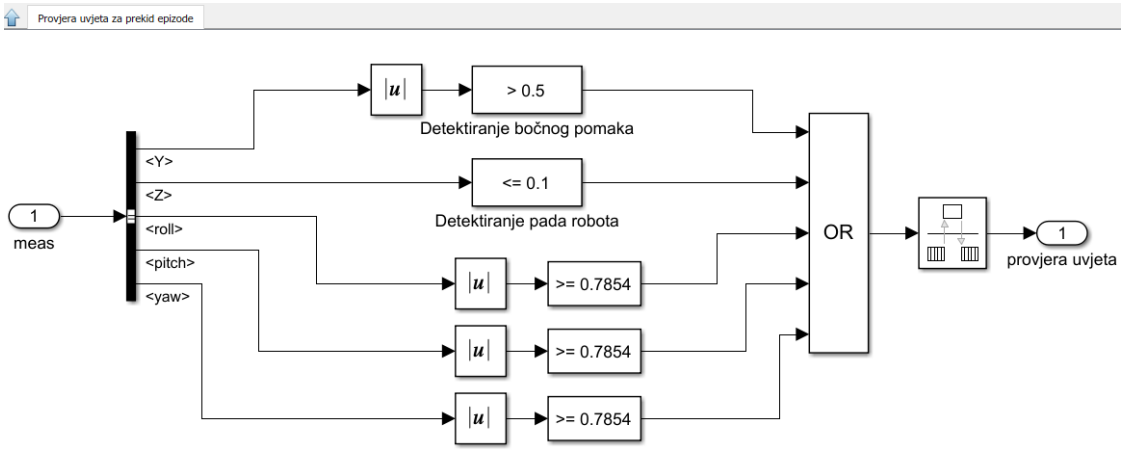
z_0 – početni položaj torza po z-osi

Dakle, iz funkcije nagrade može se zaključiti željeno ponašanje robota pri hodu. Prvi član jednadžbe nagrađuje brzinu kretanja po x-osi, drugi član nagrađuje trajanje hoda (za svaki vremenski korak simulacije "Ts" Agent dobije nagradu od 0.0625, sve dok se ne prekine simulacija, npr. zbog pada robota), treći član poprilično kažnjava odstupanje od početnog položaja torza po z-osi, četvrti član kažnjava ukupnu snagu potrebnu za pokretanje robota, dakle ne želimo prevelike iznose okretnih momenata u zglobovima (time se spriječava zabacivanje nogu robota i neujednačeno pokretanje lijeve i desne noge robota), i peti član kažnjava biočno odstupanje robota od x-osi (po y-osi u oba smjera). Na donjoj slici vidi se implementacija funkcije nagrade u **Simulink** modelu.



Slika 55 Podsustav "Izracun nagrade"

Slika 56 prikazuje podsustav "Provjera uvjeta za prekid epizode". Ovdje se definiraju uvjeti za prekid epizode prilikom treniranja RL Agenta. Zadano je maksimalno dozvoljeno bočno odstupanje od osi-x u smjeru osi-y. Također postavljen je minimalni iznos položaja torza robota po z-osi za detektiranje pada robota. Osim toga, postavljeni su maksimalni dozvoljeni kutevi rotacije torza robota oko x, y i z-osi, jer oni ukazuju na gubitak ravnoteže i siguran pad robota, odnosno na neprirodnu kretnju robota koju želimo izbjeći.



Slika 56 Podsustav "Provjera uvjeta za prekid epizode"

4.4. Matlab skripte potrebne za implementaciju metode učenja s pojačanjem na primjeru dvonožnog modela robota

Matlab skripta za učitavanje parametara modela robota za učenje s pojačanjem (kôd spremljen pod nazivom "robotParametersRL.m")

U ovoj prikazanoj Matlab skripti definirani su potrebni parametri modela robota koji se koriste u Simulink blokovima za definiranje dinamičkih i mehaničkih svojstava dijelova modela i tokom treniranja RL Agenta.

```
%% Parametri modela
% Mehanički parametri
density = 500;
foot_density = 1000;
if ~exist('actuatorType','var')
    actuatorType = 1;
end
world_damping = 1e-3;
world_rot_damping = 5e-2;

% Parametri kontakta/trenja
contact_stiffness = 500;
contact_damping = 50;
mu_k = 0.7;
mu_s = 0.9;
mu_vth = 0.01;
height_plane = 0.025;
plane_x = 25;
plane_y = 10;
contact_point_radius = 1e-4;

% Dimenzije stopala
foot_x = 5;
foot_y = 4;
foot_z = 1;
foot_offset = [-1 0 0];

% Dimenzije nogu
leg_radius = 0.75;
lower_leg_length = 10;
upper_leg_length = 10;

% Dimenzije Torza
torso_y = 8;
torso_x = 5;
torso_z = 8;
torso_offset_z = -2;
torso_offset_x = -1;
mass = (0.01^3)*torso_y*torso_x*torso_z*density;
g = 9.80665;

% Parametri zglobova
joint_damping = 1;
joint_stiffness = 0;
```

```
motion_time_constant = 0.01;
joint_limit_stiffness = 1e4;
joint_limit_damping = 10;

%% Parametri učenja s pojačanjem
Ts = 0.025; % Vrijeme uzorkovanja za agenta
Tf = 10;    % Završno vrijeme simulacije

% Faktor skaliranja za RL akciju [-1 1]
max_torque = 3;

% Početni uvjeti
h = 18;     % Visina kuka [cm]
init_height = foot_z + h + ...
              torso_z/2 + torso_offset_z + height_plane/2;
vx0 = 0;    % Početna linearna brzina po x-osi [m/s]
vy0 = 0;    % Početna linearna brzina po y-osi [m/s]
wx0 = 0;    % Početna kutna brzina po x-osi [rad/s]
wy0 = 0;    % Početna kutna brzina po x-osi [rad/s]

% Početni položaj stopala [m]
leftinit =  [0;0;-h/100];
rightinit = [0;0;-h/100];

% Izračun početnih kutova zglobova
init_angs_L = walkerInvKin(leftinit, upper_leg_length/100,
lower_leg_length/100,'3D');
init_angs_R = walkerInvKin(rightinit, upper_leg_length/100,
lower_leg_length/100,'3D');
```

Matlab skripta za kreiranje funkcije "walkerInvKin" koja računa početne kuteve zglobova modela robota (kôd spremljen pod nazivom "walkerInvKin.m")

U ovoj Matlab skripti definirana je funkcija koja koristi funkciju za izračun inverzne kinematike noge robota i pomoću nje računa 5 rotacija (rotacije gležnja oko osi x i osi y, rotacija koljena oko osi y te rotacije kuka oko osi x i osi y). Vektor tih iznosa rotacija vraća kao rezultat.

```
function angs = walkerInvKin(pos, upper_leg_length, lower_leg_length,dim)

    % Raspakiravanje ulaznih argumenata pri pozivu funkcije "walkerInvKin"
    x = pos(1);
    if isequal(dim,'3D')
        y = pos(2);
    else
        y = 0;
    end
    z = pos(3);

    % Izračun kuta kuka
    hipRoll = -atan2(y,-z);

    % Poziv simbolički generirane funkcije za 2D krak
    zMod = z/cos(hipRoll);

    maxMag = (upper_leg_length + lower_leg_length)*0.99; % Zabrana punog
ispružanja noge
    mag = sqrt(x^2 + zMod^2);
    if sqrt(x^2 + zMod^2) > maxMag
        x = x*maxMag/mag;
        zMod = zMod*maxMag/mag;
    end

    % Poziv funkcije za 2D Inverznu Kinematiku
    theta = legInvKin(upper_leg_length,lower_leg_length,-x,zMod);

    % Adresiranje višestrukih izlaza
    if size(theta,1) == 2
       if theta(1,2) < 0
          hipPitch = theta(2,1);
          kneePitch = theta(2,2);
       else
          hipPitch = theta(1,1);
          kneePitch = theta(1,2);
       end
    else
        hipPitch = theta(1);
        kneePitch = theta(2);
    end

    % Postavljanje kutova gležnja
    anklePitch = -kneePitch - hipPitch;
    ankleRoll  = -hipRoll;

    % Pakiranje
    angs = [ankleRoll; anklePitch; kneePitch; hipRoll; hipPitch];
   end
```

Matlab skripta za kreiranje funkcije "legInvKin" koja računa 2D inverznu kinematiku noge robota (kôd spremljen pod nazivom "legInvKin.m")

Ova Matlab skripta sadrži kôd za funkciju koja računa inverznu kinematiku noge modela robota.

```
function out1 = legInvKin(L1,L2,x,y)

t2 = L1.^2;
t3 = L2.^2;
t4 = x.^2;
t5 = y.^2;
t6 = L1.*L2.*2.0;
t7 = -t2-t3+t4+t5+t6;
t8 = t2+t3-t4-t5+t6;
t9 = t7.*t8;
t10 = sqrt(t9);
t11 = 1.0./t7;
t12 = t2-t3+t4+t5-L1.*y.*2.0;
t13 = 1.0./t12;
t14 = L1.*x.*2.0;
t15 = t4.*t10.*t11;
t16 = t5.*t10.*t11;
t17 = L1.*L2.*t10.*t11.*2.0;
t18 = t10.*t11;
t19 = atan(t18);
out1 = reshape([atan(t13.*(t14+t15+t16+t17-t2.*t10.*t11-
t3.*t10.*t11)).*2.0,atan(t13.*(t14-t15-t16-
t17+t2.*t10.*t11+t3.*t10.*t11)).*2.0,t19.*-2.0,t19.*2.0],[2,2]);
```

Matlab skripta za treniranje DDPG (eng. Deep Deterministic Policie Gradient) Agenta (kôd spremljen pod nazivom "createWalkingAgent2D.m")

Ova Matlab skripta sadrži kôd za treniranje RL DDPG Agenta. Kreiraju se informacije o obzervacijama i akcijama. Kao što je ranije spomenuto, obzervacija za trening Agenta ima ukupno 31, a akcija ima 6 te su ograničene na vrijednosti između -1 i 1 (radi kvalitetnijeg/bržeg/lakšeg treniranja umjetnih neuronskih mreža!). Kreira se okolina, koja se povezuje sa Simulink modelom robota (v. slika 53). Za potrebe treninga, definirana je funkcija za resetiranje, odnosno vraćanje robota u početni položaj nakon svake epizode treninga. Zatim se poziva Matlab skripta "moje_neuronske_mreze" čime se kreiraju umjetne neuronske mreže za RL Agenta u željenoj strukturi. Nakon toga se poziva Matlab skripta "createDDPGOptions" čime se kreiraju željene opcije RL Agenta. Potom se kreira Agent i započinje njegovo treniranje po definiranim opcijama treninga iz skripte "createDDPGOptions". Tokom treniranja, u mapu "savedAgents" spremaju se oni Agenti koji zadovolje uvjet za spremanje (koji je definiran u skripti "createDDPGOptions"). Spremljeni Agenti se nakon treninga mogu simulirati unutar Simulink modela robota (v. slika 53) ili aplikacije za učenje s pojačanjem.

```
%% POSTAVLJANJE RADNE OKOLINE
% Opcije ubrzanog učenja
useFastRestart = true;
useGPU = true;
%useParallel = true;
useParallel = false;

% Kreiranje obzervacijskih informacija
numObs = 31;
observationInfo = rlNumericSpec([numObs 1]);
observationInfo.Name = 'observations';

% Kreiranje akcijskih informacija
numAct = 6;
actionInfo = rlNumericSpec([numAct 1],'LowerLimit',-1,'UpperLimit', 1);
actionInfo.Name = 'foot_torque';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Radna okolina za treniranje agenta
mdl = 'MOJ_ROBOT';
load_system(mdl);
blk = [mdl,'/RL Agent'];
env = rlSimulinkEnv(mdl,blk,observationInfo,actionInfo);
env.ResetFcn =
@(in)walkerResetFcn(in,upper_leg_length/100,lower_leg_length/100,h/100,'2D'
);
if ~useFastRestart
   env.UseFastRestart = 'off';
end
%% KREIRANJE UMJETNIH NEURONSKIH MREŽA ZA AGENTA ("actor" i "critic" mreže)
moje_neuronske_mreze
```

```
%% KREIRANJE I TRENIRANJE AGENTA
createDDPGOptions;
agent = rlDDPGAgent(actor,critic,agentOptions);
trainingResults = train(agent,env,trainingOptions)

%% SPREMANJE DOBRIH AGENATA TOKOM TRENINGA
reset(agent); % Čišćenje međuspremnika iskustava (eng. experience buffer-a)
curDir = pwd;
saveDir = 'savedAgents';
cd(saveDir)
save(['trainedAgent_2D_' datestr(now,'mm_DD_YYYY_HHMM')],'agent');
save(['trainingResults_2D_'
datestr(now,'mm_DD_YYYY_HHMM')],'trainingResults');
cd(curDir)
```

Matlab skripta za kreiranje funkcije "walkerResetFcn" koja služi za resetiranje položaja modela robota prilikom postupka treiranja DDPG Agenta (kôd spremljen pod nazivom " walkerResetFcn.m")

U ovoj Matlab skripti nalazi se kôd za kreiranje funkcije "walkerResetFcn", to je funkcija koja služi za resetiranje, odnosno vraćanje robota u početni položaj nakon svake epizode treninga. Unutar ove funkcije definirane su šanse za vraćanja u nulti početni položaj nakon epizode treninga, kao i šanse za postavljanje robota u neki nasumično generirani početni položaj, jer je pri učenju s pojačanjem dobro ponekad unositi neke promjene radi pretraživanja prostora potencijalno dobrih zakona upravljanja.

```
%Pomoćna funkcija za resetiranje simulacije Hodajućeg robota s drugačijim
%početnim uvjetima

function in =
walkerResetFcn(in,upper_leg_length,lower_leg_length,init_height,dim)

    % Parametri randomizacije
    max_displacement_x = 0.05;
    max_speed_x = 0.05;
    max_displacement_y = 0.025;
    max_speed_y = 0.025;

    % Vjerojatnost za randomizaciju početnih uvjeta
    if rand < 0.5
        if rand < 0.5
            in = in.setVariable('vx0',2*max_speed_x *(rand-0.5));
            in = in.setVariable('vy0',2*max_speed_y *(rand-0.5));
        else
            in = in.setVariable('vx0',0);
            in = in.setVariable('vy0',0);
        end
        leftinit  = [(rand(2,1)-[0.5;0]).*[2*max_displacement_x;
max_displacement_y]; -init_height];
        rightinit = [-leftinit(1); -rand*max_displacement_y ; -
init_height]; % Osiguravanje da su stopala simetrično pozicionirana radi
stabilnosti

    % Vjerojatnost za početak iz nultih početnih uvjeta
    else
        in = in.setVariable('vx0',0);
        in = in.setVariable('vy0',0);
        leftinit  = [0;0;-init_height];
        rightinit = [0;0;-init_height];
    end

    % Rješavanje inverzne kinematike za položaje lijevog i desnog stopala

    init_angs_L = walkerInvKin(leftinit, upper_leg_length,
lower_leg_length, dim);
    in = in.setVariable('leftinit',leftinit);
    in = in.setVariable('init_angs_L',init_angs_L);
```

```
    init_angs_R = walkerInvKin(rightinit, upper_leg_length,
lower_leg_length, dim);
    in = in.setVariable('rightinit',rightinit);
    in = in.setVariable('init_angs_R',init_angs_R);

end
```

Matlab skripta za kreiranje neuronskih mreža Agenta za učenje s pojačanjem (kôd spremljen pod nazivom "moje_neuronske_mreze.m")

U ovoj Matlab skripti nalazi se kôd za kreiranje dviju umjetnih neuronskih mreža za DDPG RL Agenta. To su "Critic" i "Actor" neuronske mreže.

"Actor" mreža ima jedan ulazni i jedan izlazni sloj. U ulaznom sloju prima 31 obzervaciju iz modela robota, zatim se ti podaci izmjenjuju kroz slojeve mreže pomoću množenja sa težinama (eng. weights) i na zadnjem sloju mreže dobije se izlaz koji šalje vektor od 6 skalara, vrijednosti između -1 i 1. Ti brojevi se u modelu robota skaliraju i šalju na zglobove koji pokreću robota. "Fully Connected" sloj množi ulaz s matricom težina i potom dodaje "bias" vektor. To je jedinični vektor koji dimenzijama odgovara broju težina u sloju. "ReLu" sloj izvodi operaciju praga (eng. threshold) na svakom ulaznom elementu u sloj, pri čemu se svaka vrijednost manja od nule postavlja na nulu. Aktivacijski sloj hiperboličnog tangenta "tanh" primjenjuje funkciju tanh na sve ulaze sloja. Time se ograničavaju izlazi "Actor" neuronske mreže na vrijednosti između -1 i 1!

"Critic" mreža ima dvije ulazne grane koje se spajaju prije izlaznog sloja. Lijeva grana započinje ulaznim slojem gdje dolaze podaci obzervacija s modela robota, a desna strana započinje ulaznim slojem s podacima o akcijama koje generira RL Agent pomoću "Actor" neuronske mreže. Na lijevoj grani podaci prolaze kroz dva "Fully Connected" sloja, jer ulaz sadrži 31 podatak, pa je poželjno te podatke obrađivati kroz više slojeva i primjenjivati na njih veći broj težina. U ovoj mreži primjenjen je i "Addition" sloj, koji zbraja ulaze iz više slojeva neuronske mreže, zbrajanje se vrši po elementima. "Critic" mreža dakle ima dva ulazna sloja i jedan izlazni sloj. Ovdje taj sloj daje samo jedan skalar, koji nema ograničen iznos (za razliku od izlaznog sloja "Actor-a"), a taj broj predstavlja Q-vrijednost (eng. Q-value), koja predstavlja procjenu "Critic" neuronske mreže o kvaliteti ponašanja Agenta, odnosno to je predviđanje ove mreže o ukupnoj nagradi koja će se postići na kraju trenutno aktivne epizode. Ta vrijednost se postavlja na novi iznos (na temelju prethodnih iskustava iz prethodnih epizoda) prije svake pojedine epizode treninga Agenta. "Q-value" je naziv koji potječe od engleske riječi za kvalitetu ("Quality"), što ima smisla jer se koristi za oznaku dobrote/kvalitete ponašanja RL Agenta.

```
%% "CRITIC" NEURONSKA MREŽA
% Kreiranje slojeva "critic" neuronske mreže
criticLayerSizes = [400 300];
statePath = [
    imageInputLayer([numObs 1 1],'Normalization','none','Name', 'input_1')
    fullyConnectedLayer(criticLayerSizes(1), 'Name', 'CriticStateFC1', ...
            'Weights',2/sqrt(numObs)*(rand(criticLayerSizes(1),numObs)-
0.5), ...
            'Bias',2/sqrt(numObs)*(rand(criticLayerSizes(1),1)-0.5))
    reluLayer('Name','CriticStateRelu1')
    fullyConnectedLayer(criticLayerSizes(2), 'Name', 'CriticStateFC2', ...

'Weights',2/sqrt(criticLayerSizes(1))*(rand(criticLayerSizes(2),criticLayer
Sizes(1))-0.5), ...

'Bias',2/sqrt(criticLayerSizes(1))*(rand(criticLayerSizes(2),1)-0.5))
    ];
actionPath = [
    imageInputLayer([numAct 1 1],'Normalization','none', 'Name', 'input_2')
    fullyConnectedLayer(criticLayerSizes(2), 'Name', 'CriticActionFC1', ...
            'Weights',2/sqrt(numAct)*(rand(criticLayerSizes(2),numAct)-
0.5), ...
            'Bias',2/sqrt(numAct)*(rand(criticLayerSizes(2),1)-0.5))
    ];
commonPath = [
    additionLayer(2,'Name','add')
    reluLayer('Name','CriticCommonRelu1')
    fullyConnectedLayer(1, 'Name', 'CriticOutput',...
            'Weights',2*5e-3*(rand(1,criticLayerSizes(2))-0.5), ...
            'Bias',2*5e-3*(rand(1,1)-0.5))
    ];

% Spajanje grafa slojeva (eng. Layer Graph-a)
CriticNetwork = layerGraph(statePath);
CriticNetwork = addLayers(CriticNetwork, actionPath);
CriticNetwork = addLayers(CriticNetwork, commonPath);
CriticNetwork = connectLayers(CriticNetwork,'CriticStateFC2','add/in1');
CriticNetwork = connectLayers(CriticNetwork,'CriticActionFC1','add/in2');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Kreiranje reprezentacije "critic" neuronske mreže (za implementaciju
učenja s pojačanjem!)
criticOptions = rlRepresentationOptions('Optimizer','adam','LearnRate',1e-
3, ...

'GradientThreshold',1,'L2RegularizationFactor',2e-4);

if useGPU
   criticOptions.UseDevice = 'gpu';
end

o=env.getObservationInfo;
a=env.getActionInfo;
critic =
rlQValueRepresentation(CriticNetwork,o,a,'Observation',{'input_1'},'Action'
,{'input_2'});
critic.Options=criticOptions
```

```
%plot(CriticNetwork)

%% "ACTOR" NEURONSKA MREŽA
% Kreiranje slojeva "actor" neuronske mreže
actorLayerSizes = [400 300];
actorNetwork = [
    imageInputLayer([numObs 1 1],'Normalization','none','Name','input_1')
    fullyConnectedLayer(actorLayerSizes(1), 'Name', 'ActorFC1', ...
            'Weights',2/sqrt(numObs)*(rand(actorLayerSizes(1),numObs)-0.5),
...
            'Bias',2/sqrt(numObs)*(rand(actorLayerSizes(1),1)-0.5))
    reluLayer('Name', 'ActorRelu1')
    fullyConnectedLayer(actorLayerSizes(2), 'Name', 'ActorFC2', ...

'Weights',2/sqrt(actorLayerSizes(1))*(rand(actorLayerSizes(2),actorLayerSiz
es(1))-0.5), ...
            'Bias',2/sqrt(actorLayerSizes(1))*(rand(actorLayerSizes(2),1)-
0.5))
    reluLayer('Name', 'ActorRelu2')
    fullyConnectedLayer(numAct, 'Name', 'ActorFC3', ...
            'Weights',2*5e-3*(rand(numAct,actorLayerSizes(2))-0.5), ...
            'Bias',2*5e-3*(rand(numAct,1)-0.5))
    tanhLayer('Name',"tanh")
    ];

% Spajanje grafa slojeva (eng. Layer Graph-a)
ActorNetwork = layerGraph(actorNetwork);

% Kreiranje reprezentacije "actor" neuronske mreže (za implementaciju
učenja s pojačanjem!)
actorOptions = rlRepresentationOptions('Optimizer','adam','LearnRate',1e-4,
...

'GradientThreshold',1,'L2RegularizationFactor',1e-5);
if useGPU
   actorOptions.UseDevice = 'gpu';
end

actor =
rlDeterministicActorRepresentation(ActorNetwork,o,a,'Observation',{'input_1
'},'Action',{'tanh'});
actor.Options=actorOptions


%figure(2);
%plot(ActorNetwork)
```

Matlab skripta za kreiranje opcija DDPG agenta i opcija treniranja za hodajućeg robota (kôd spremljen pod nazivom "createDDPGOptions.m")

U ovoj Matlab skripti definirane su opcije za DDPG Agenta, kao i opcije za trening tog Agenta u okolini (kreiranoj u Simulink-u).

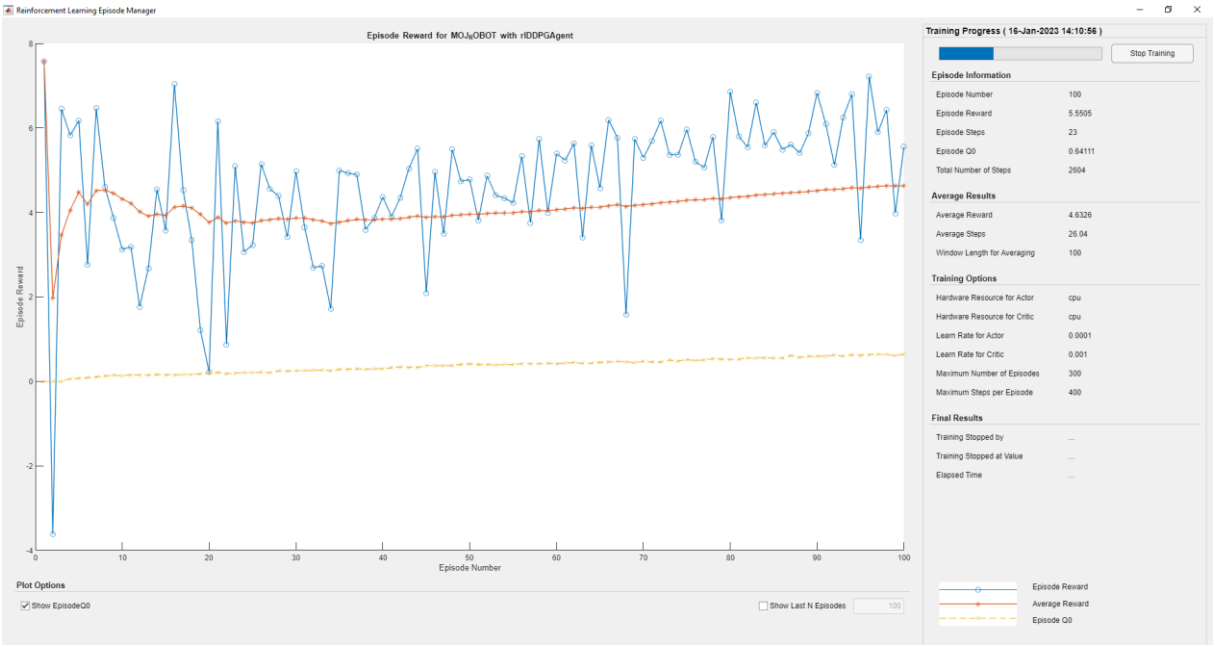
```
% Kreiranje opcija za DDPG agenta i opcija treniranja za hodajućeg robota

%% Opcije DDPG Agenta
agentOptions = rlDDPGAgentOptions;
agentOptions.SampleTime = Ts;
agentOptions.DiscountFactor = 0.99;
agentOptions.MiniBatchSize = 128;
agentOptions.ExperienceBufferLength = 5e5;
agentOptions.TargetSmoothFactor = 1e-3;
agentOptions.NoiseOptions.MeanAttractionConstant = 5;
agentOptions.NoiseOptions.Variance = 0.5;
agentOptions.NoiseOptions.VarianceDecayRate = 1e-5;

%% Opcije treniranja
trainingOptions = rlTrainingOptions;
trainingOptions.MaxEpisodes = 3000;
trainingOptions.MaxStepsPerEpisode = Tf/Ts;
trainingOptions.ScoreAveragingWindowLength = 100;
trainingOptions.StopTrainingCriteria = 'AverageReward';
trainingOptions.StopTrainingValue = 110;
trainingOptions.SaveAgentCriteria = 'EpisodeReward';
trainingOptions.SaveAgentValue = 120
trainingOptions.Plots = 'training-progress';
trainingOptions.Verbose = true;
if useParallel
    trainingOptions.Parallelization = 'async';
    trainingOptions.ParallelizationOptions.StepsUntilDataIsSent = 32;
end
```

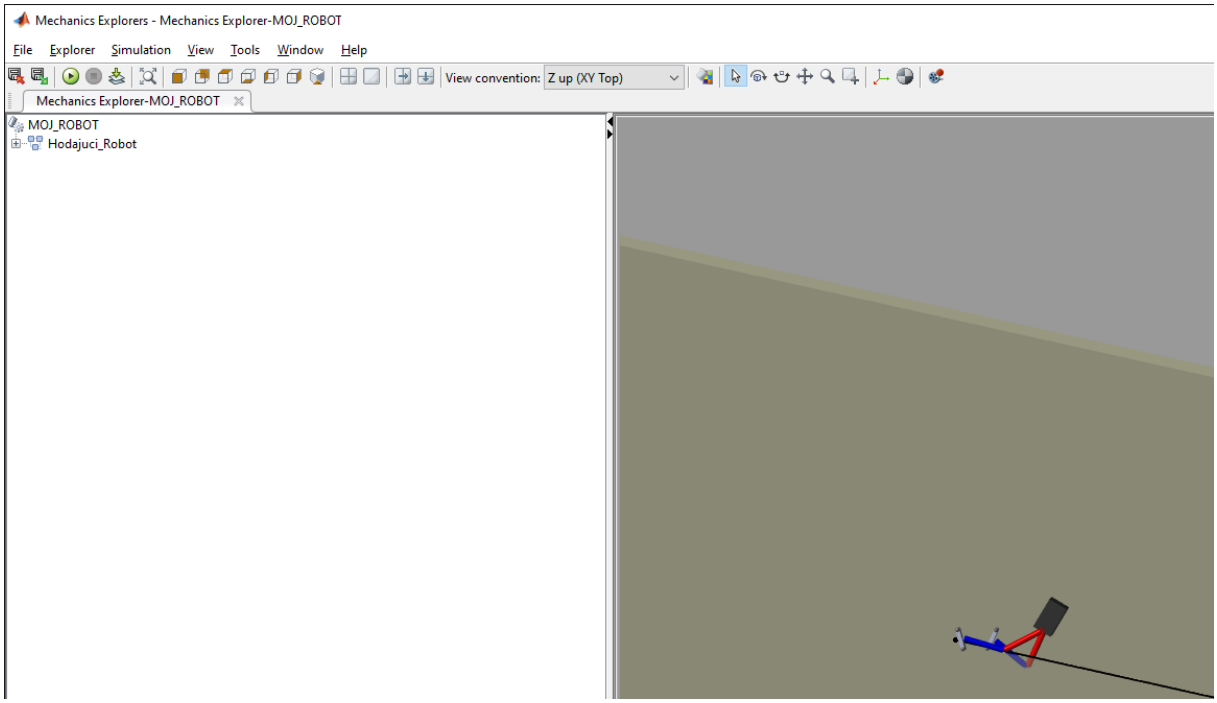
4.5. Treniranje RL Agenta koji upravlja hodom modela robota

Pokretanjem skripte za učitavanje parametara modela robota, a potom i skripte za treniranje DDPG Agenta, otvara se prozor prikazan na **slici 57** te započinje treniranje DDPG Agenta koji djeluje unutar okoline Simulink modela hodajućeg robota pod nazivom "MOJ_ROBOT.slx". Cilj treninga je maksimizirati ukupnu nagradu, a u Matlab skripti "createDDPGOptions" definirani su uvjeti o maksimalnom broju epizoda, te uvjeti za spremanje dobrih Agenata (sve epizode koje postignu nagradu 120 ili više), kao i uvjet prekida treniranja Agenta (ako prosječna nagrada (eng. AverageReward) dosegne vrijednost od 110).



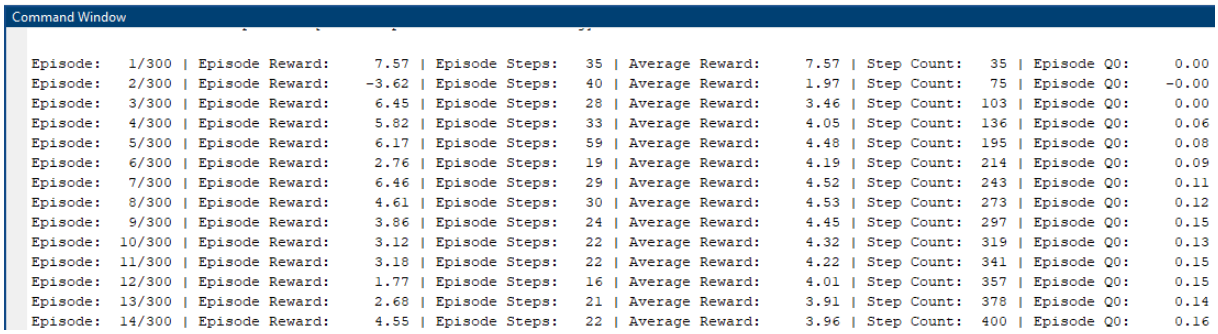
Slika 57 Treniranje DDPG Agenta

Slika 58 prikazuje kretnju modela robota na početku učenja. Očito je da robot ne hoda ispravno, ali to je normalno s obzirom da RL DDPG Agent još nema iskustva i nema razvijena pravila ponašanja (odnosno težinske faktore u neuronskim mrežama) kojima bi postizao velike iznose funkcije nagrade.



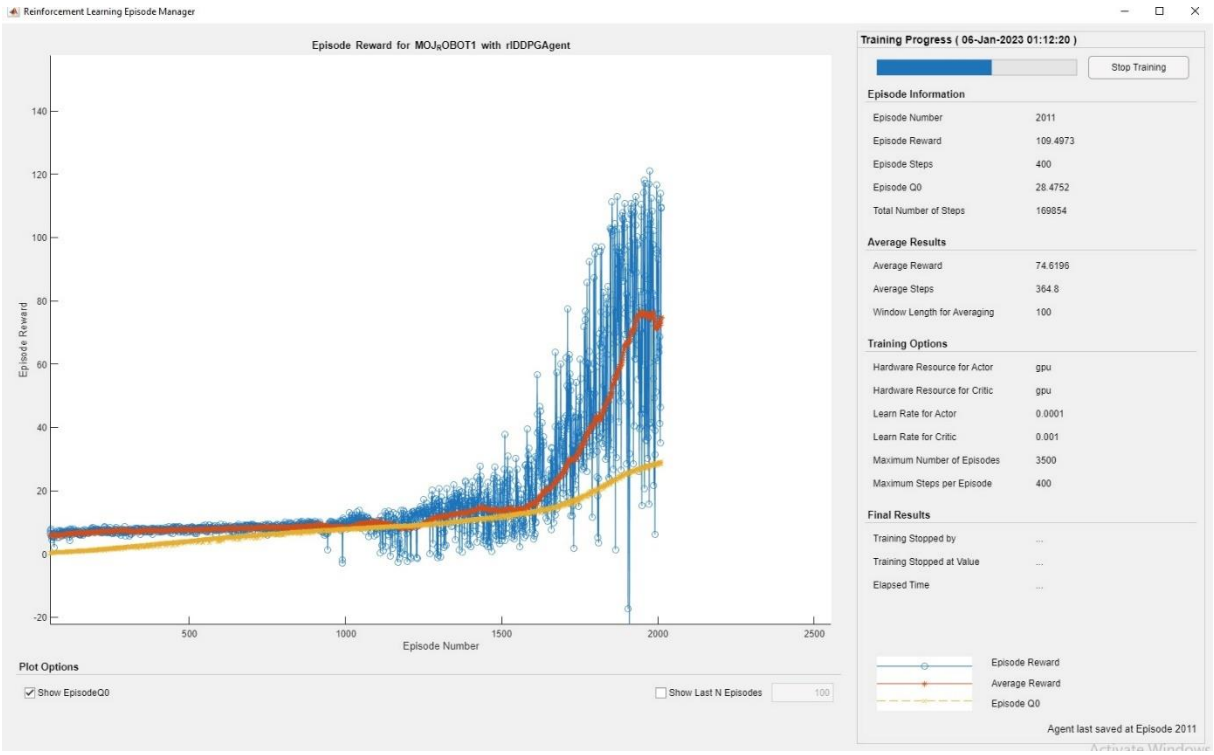
Slika 58 Kretnja modela robota na početku učenja

Slika 59 prikazuje podatke o treningu koji se ispisuju u naredbenom prozoru (eng. Command Window) u Matlabu tokom treniranja Agenta.



Slika 59 Podaci o treningu

Slika 60 sadrži graf koji prikazuje izgled vrijednosti nagradne funkcije za pojedine epizode tokom treniranja Agenta. Plave linije i kružići označavaju nagrade pojedinačnih epizoda, narančasta krivulja prikazuje prosječnu nagradu (tu se vidi pozitivan trend tokom učenja Agenta), a žuta krivulja predstavlja procjenu "Critic" neuronske mreže o kvaliteti ponašanja, odnosno predviđenu nagradu prije početka svake pojedine epizode učenja.

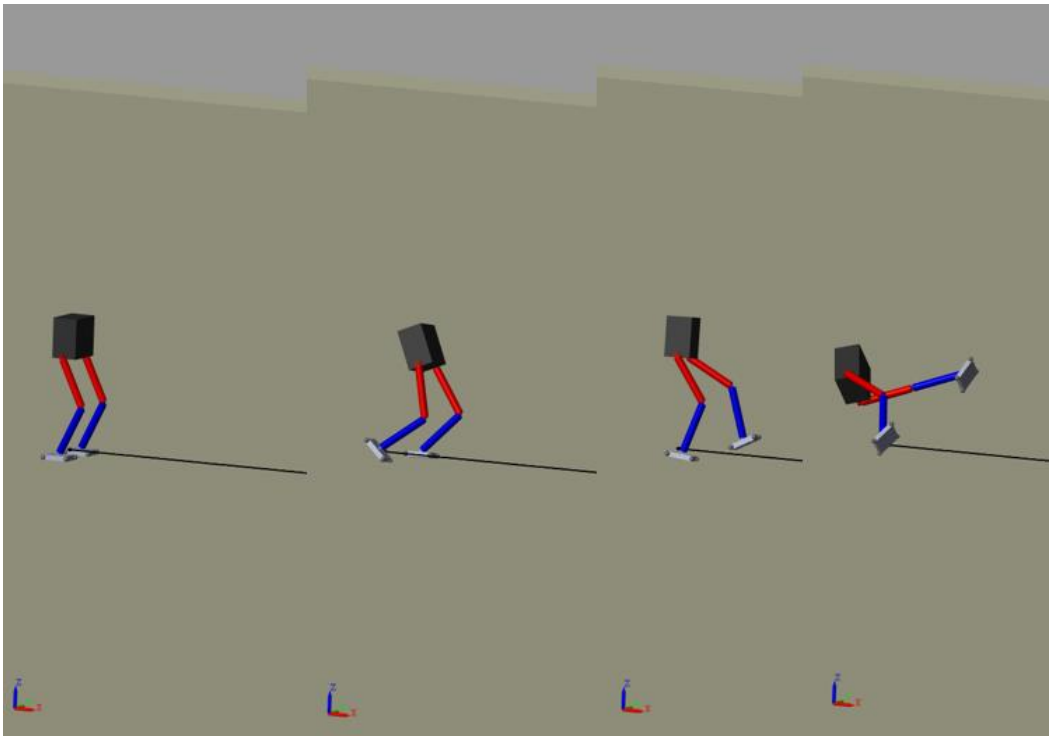


Slika 60 Graf s vrijednostima nagradne funkcije za pojedine epizode tokom treniranja Agenta

Ovaj trening je bio predviđen za 3500 epizoda, međutim, trajanje učenja je bilo više od 12h do trenutka prekida, te je učenje prekinuto iz razloga što je računalo postalo preopterećeno i proces se znatno usporio. S druge strane, učenje se dobro odvijalo, te su desetci agenata spremljeni zbog zadovoljenog uvjeta spremanja definiranog u "createDDPGOptions" skripti.

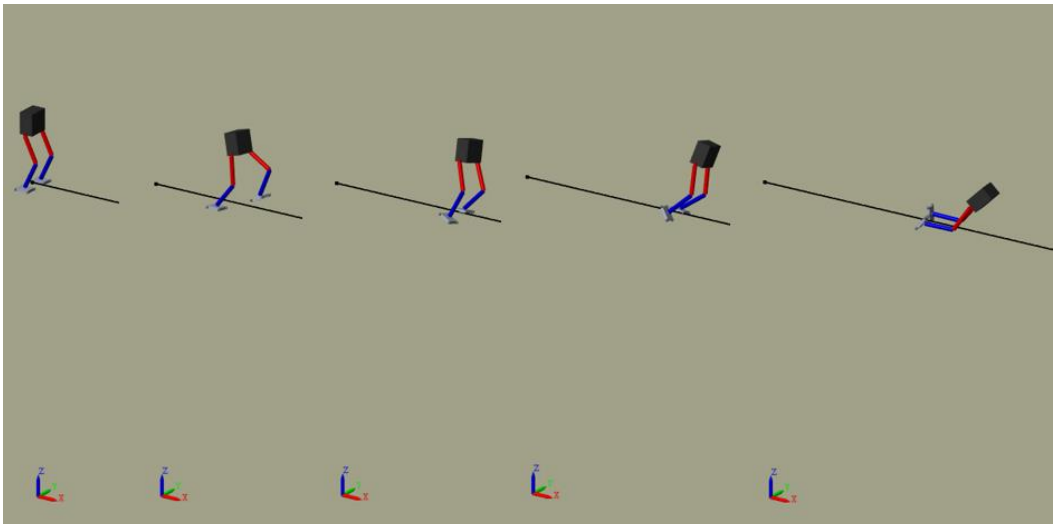
4.6. Prikazivanje simulacijskih rezultata odabranih primjera istreniranih RL Agenata u Simulink okruženju

Na **slici 61** je primjer jednog od spremljenih Agenata. S obzirom da je za ovaj trening bio zadan vrlo nizak prag za spremanje agenata (nagrada od 45 ili više), spremljeni su i dobri i loši Agenti. Ova slika prikazuje primjer lošeg Agenta koji pada na samom startu, bez ijednog koraka prema naprijed.



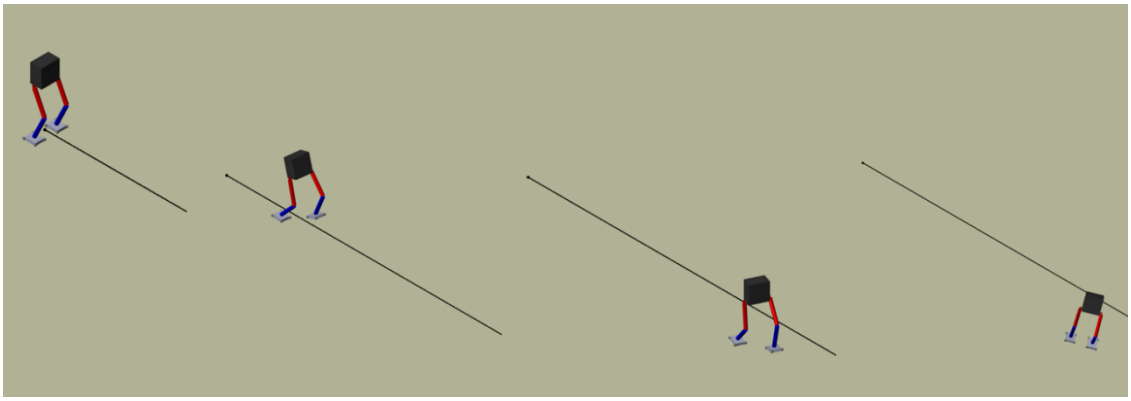
Slika 61 Primjer jednog od spremljenih Agenata

Slika 62 prikazuje Agenta koji ima malo bolje naučeno ponašanje. Robot je u simulaciji napravio nekoliko koraka unaprijed, međutim i ovaj Agent je loš jer robot gubi ravnotežu i završava padom.



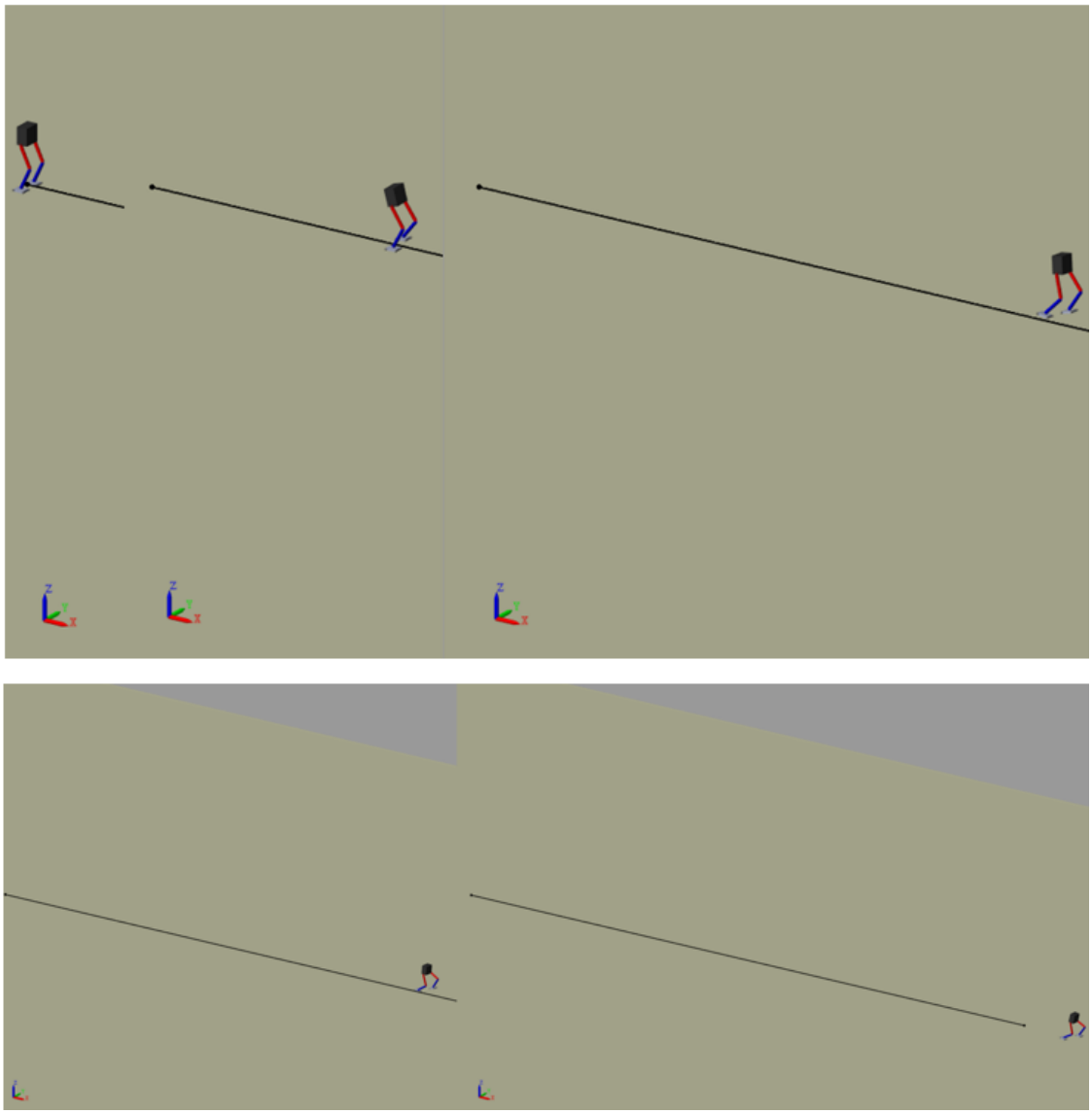
Slika 62 Agent s boljim ponašanjem

Slika 63 pokazuje Agenta koji u početku hoda dobro, međutim, nakon nekog vremena skreće naglo desno i odvaja se previše od zacrtane linije, odnosno ima preveliki bočni pomak u negativnom smjeru osi y. Time je zadovoljen uvjet za prekid epizode, prekida se simulacija te se i ovaj Agent smatra lošim.



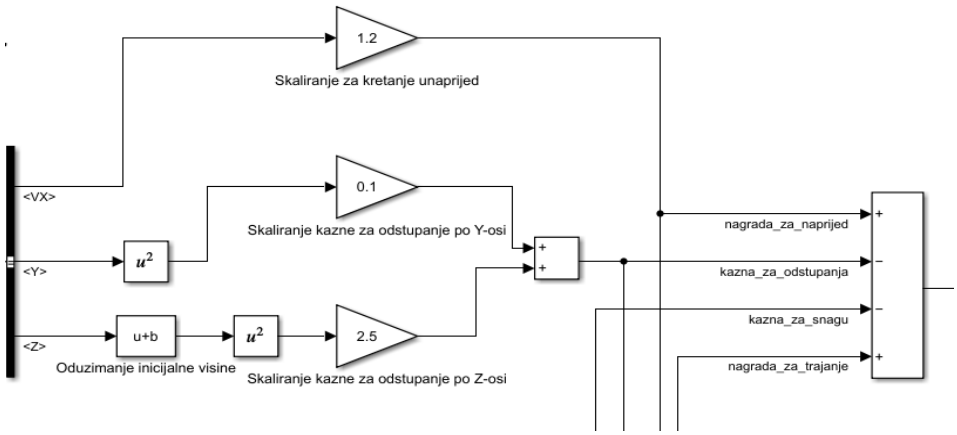
Slika 63 Još jedan primjer neuspješnog Agenta

Slika 64 prikazuje primjer dobrog Agenta. Vidljivo je da je robot u simulaciji prešao put označen crnom krivuljom bez pada. Osim toga, robot nije tokom hoda previše odstupio ni u smjeru osi y niti u smjeru osi z, niti je zadovoljio ijedan od uvjeta za prekid epizode. Prema tome, ovaj RL DDPG Agent se smatra dobrim, odnosno pronađena je zadovoljavajuća kombinacija težinskih faktora u umjetnim neuronskim mrežama Agenta koja rezultira dobrim ponašanjem Agenta u okolini, odnosno dobrim zakonom upravljanja za hodanje modela dvonožnog robota.



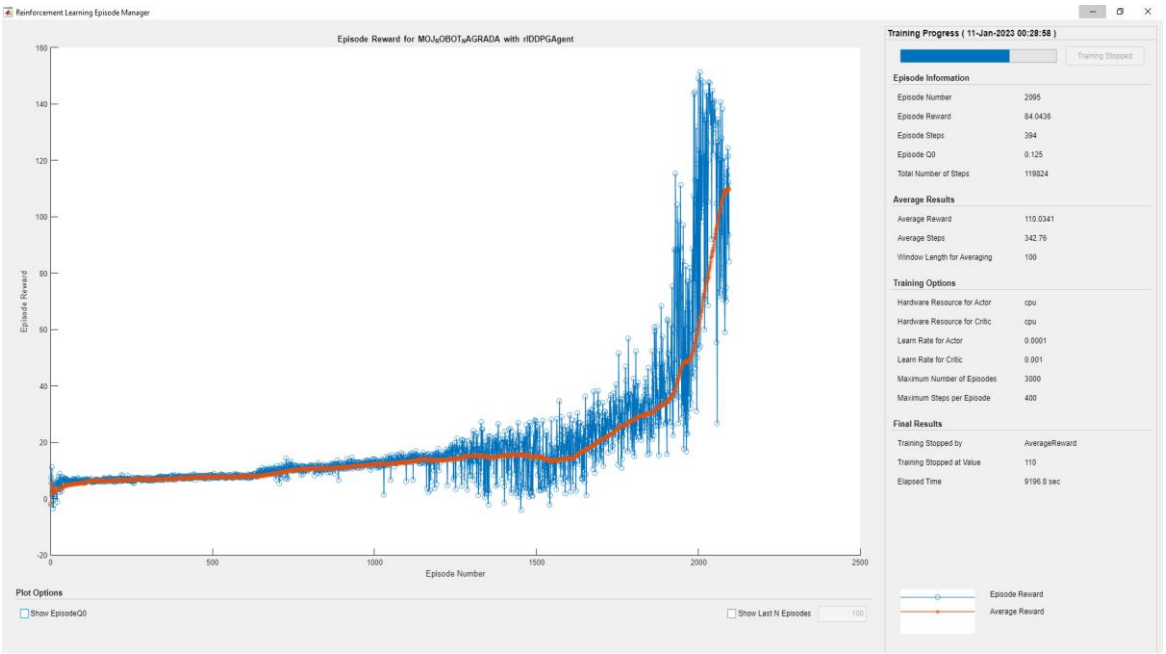
Slika 64 Primjer dobrog Agenta

Slika 65 pokazuje izmjene u podsustavu "Izracun nagrade", u kojem se definira funkcija nagrade, odnosno funkcija cilja kojom se modelira željeno ponašanje Agenta. Ako se usporedi ova slika sa **slikom 55**, vidljivo je da su promijenjene vrijednosti u pojedinim blokovima za skaliranje signala. U odnosu na prethodni model, sada se više nagrađuje brzina u smjeru x-osi (20% veća nagrada), a smanjuje se kazna za odstupanje u smjeru osi y (15 puta manja kazna) i osi z (10 puta manja kazna). To će utjecati na proces treniranja Agenta.



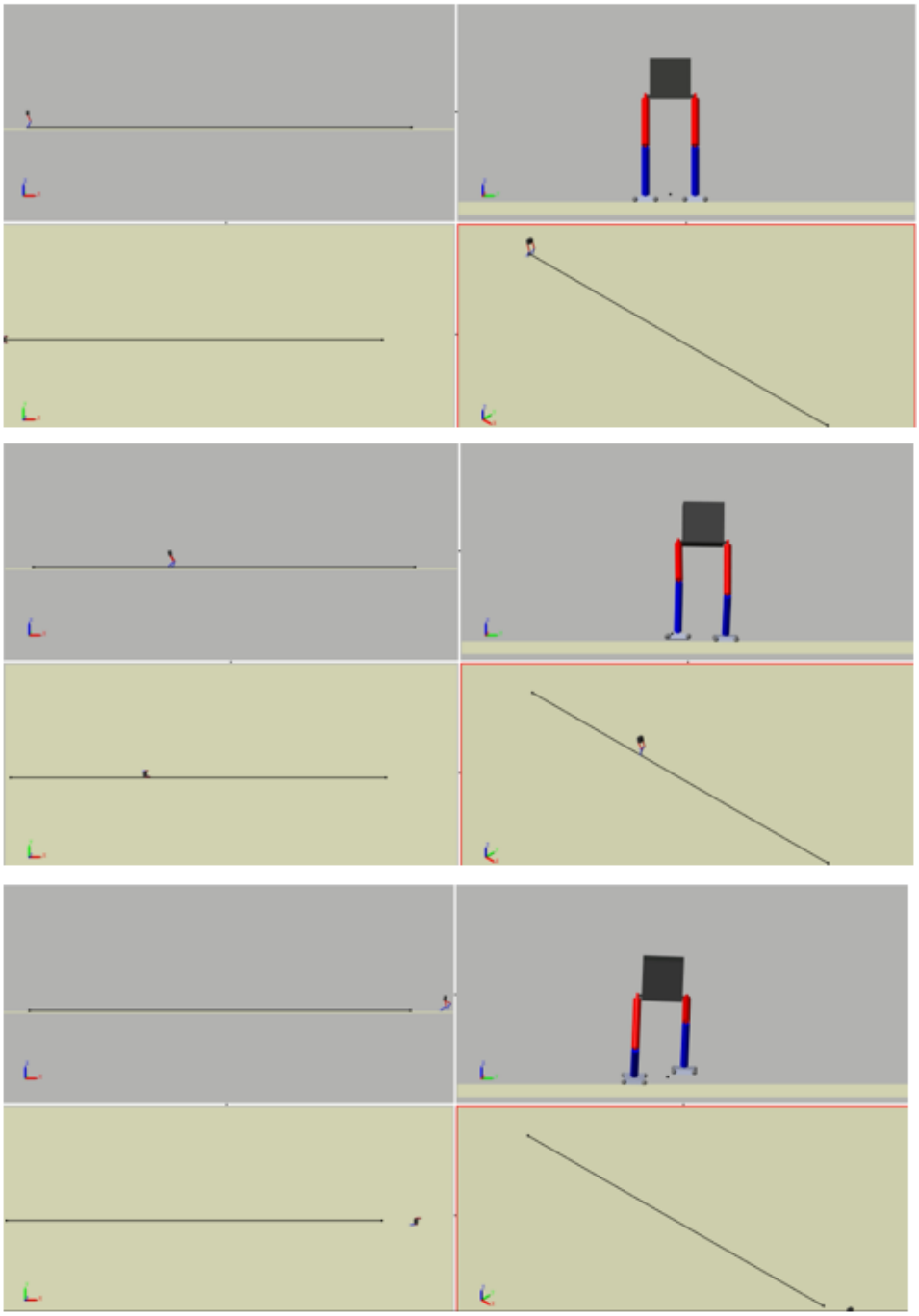
Slika 65 Izmjene u podsustavu "Izracun nagrade"

Slika 66 prikazuje graf s nagradama tokom procesa treniranja RL DDPG Agenta s izmjenama u nagradnoj funkciji. Ovdje je trening prekinut jer je zadovoljen uvjet za prekid (prosječna nagrada po epizodi je prešla vrijednost od 110).



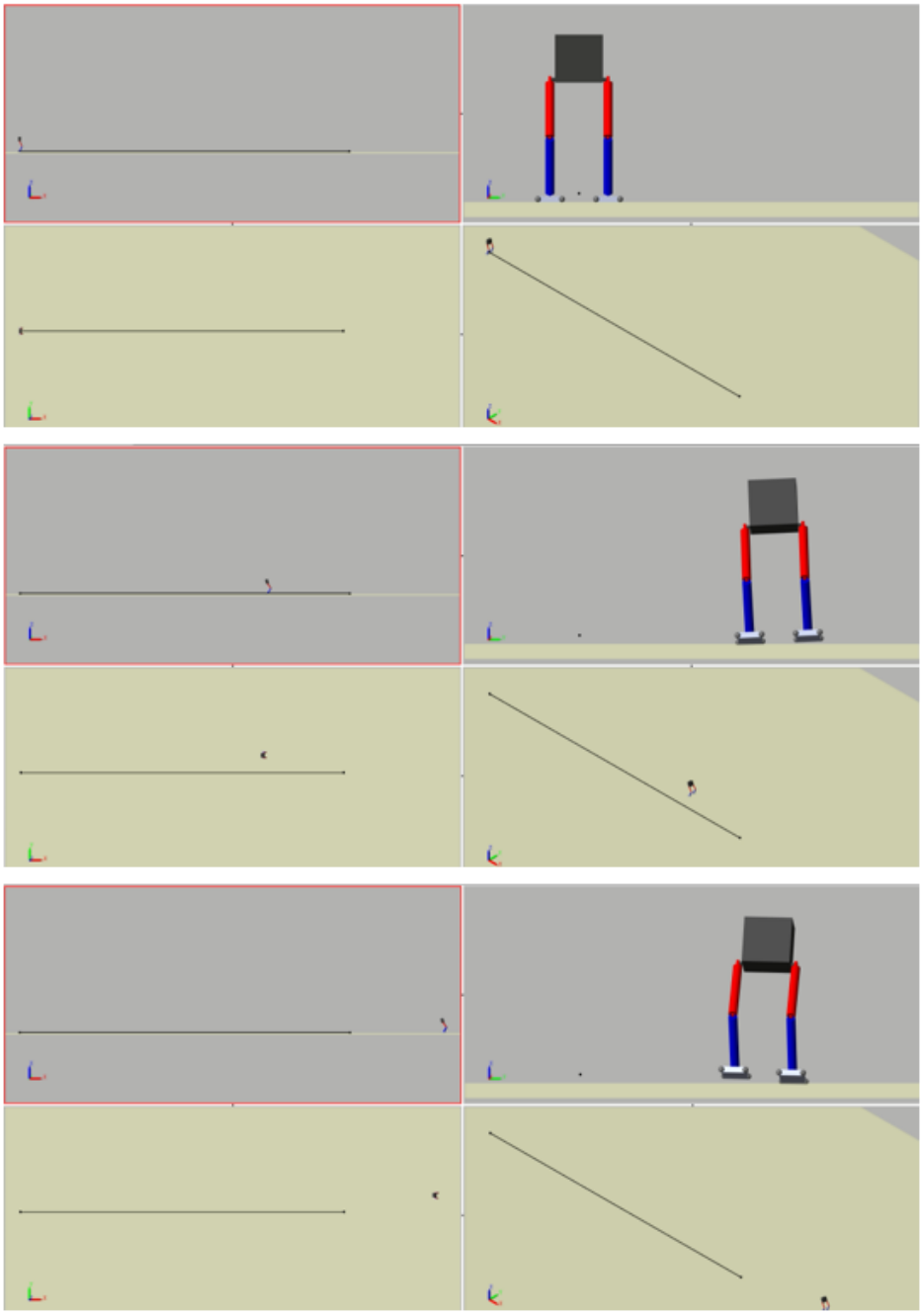
Slika 66 graf s nagradama tokom treniranja Agenta s izmjenama u nagradnoj funkciji

Slika 67 prikazuje tijek simulacije hoda modela dvonožnog robota upravljanog Agentom koji je istreniran tokom prvog treninga prikazanog na **slici 60**. Vidljivo je da robot prelazi označeni put, te nastavlja i dalje hodati bez pada (da se pusti dulje vrijeme simulacije, robot bi prešao i veći put). To ukazuje na dobro pronađen zakon upravljanja. Također je uočljivo da robot tokom hoda ne odstupa puno od osi x i tokom hoda se ne njiše previše lijevo-desno, niti mu se torzo naginje unazad, odnosno ne odstupa puno od početnog položaja torza po z-osi. To je u skladu sa modelom nagradne funkcije za taj slučaj treniranja.



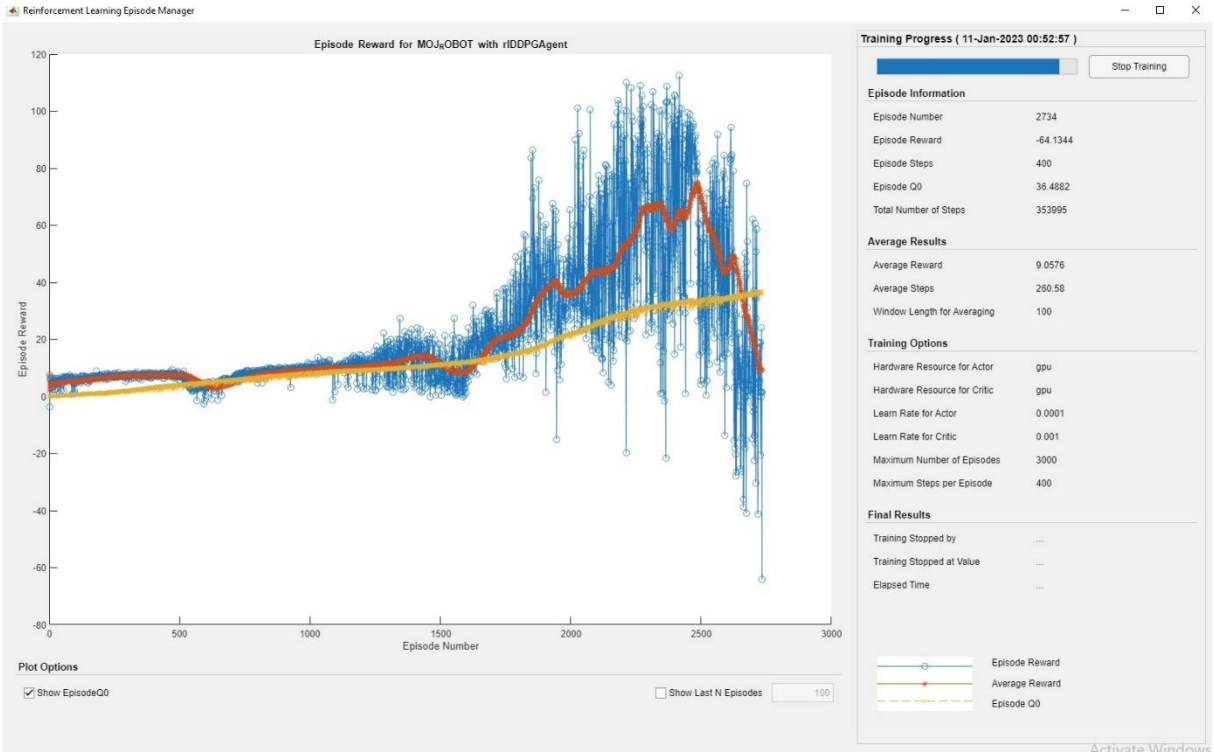
Slika 67 Tijek simulacije hoda modela dvonožnog robota

Slika 68 pokazuje tijek simulacije hoda modela dvonožnog robota upravljanog Agentom koji je istreniran tokom treninga na **slici 66**. Kod ovog treniranja uvedene su spomenute promjene u nagradnoj funkciji, pa se očekuje drugačije ponašanje Agenta. To je potvrđeno donjom slikom, gdje je vidljivo da je za isto vrijeme trajanja simulacije (od 20 sekundi), robot prešao veći put u smjeru osi x (20% veće forsiranje v_x). Također, uočljivo je veće bočno odstupanje u smjeru y-osi (jer je smanjena kazna za odstupanje po y-osi) te se može primjetiti malo veće naginjanje lijevo-desno pri hodu te naginjanje torza unazad, čime dolazi i do većih pomaka torza po z-osi (posljedica manje kazne za odstupanje po z-osi).



Slika 68 Tijek simulacije hoda modela dvonožnog robota s drugačijom nagradnom funkcijom

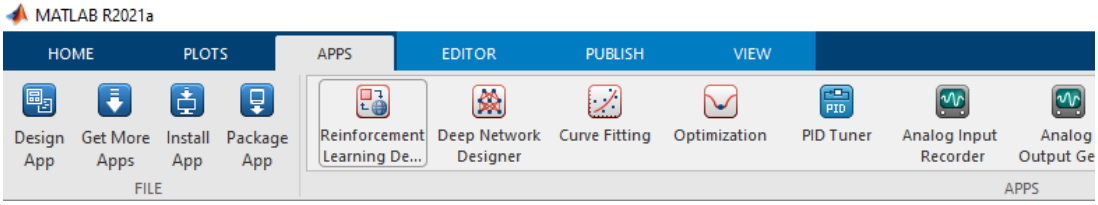
Slika 69 prikazuje graf treniranja DDPG Agenta za slučaj kada učenje nije bilo baš uspješno. S obzirom da je DDPG kao RL algoritam poprilično stohastičan, ne može se garantirati kvalitetno učenje Agenta svaki put. Ovaj graf je prikazan radi ilustracije opisane prirode DDPG algoritma.



Slika 69 graf treniranja DDPG Agenta za slučaj neuspješnog učenja

4.7. Opis funkcionalnosti "Reinforcement Learning Designer" (RLD)aplikacije

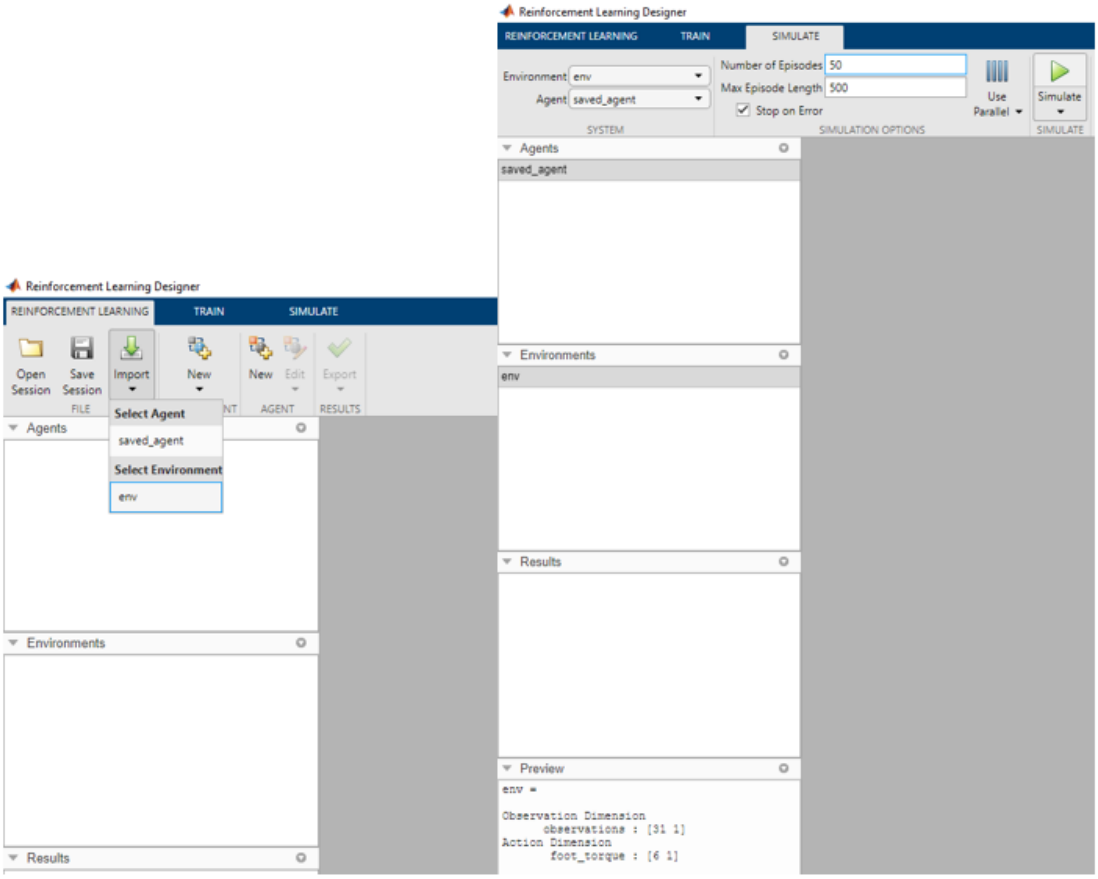
Slika 70 prikazuje način otvaranja "Reinforcement Learning Designer" aplikacije za učenje s pojačanjem iz izbornika s aplikacijama u Matlabu.



Slika 70 Pokretanje "Reinforcement Learning Designer" aplikacije

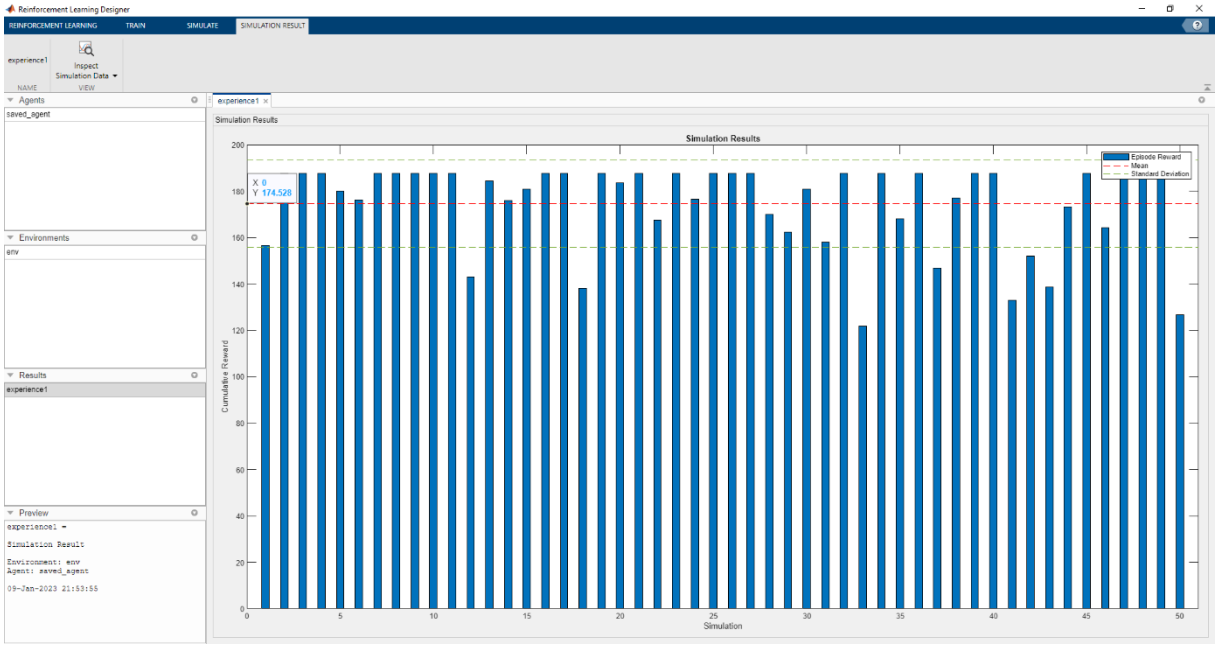
Postoji i drugi način pokretanja iste aplikacije, a to je upisom naredbe: `reinforcementLearningDesigner` u Matlabu.

Slika 71 prikazuje izgled početne stranice po otvaranju RLD aplikacije. U startu nema definiranih Agenata (eng. Agents) niti okolina (eng. Environments). Ova aplikacija služi za treniranje i simuliranje RL Agenata u definiranim okolinama. Kako bi to bilo moguće, u radnom prostoru Matlaba (eng. Workspace) moraju se nalaziti RL Agent (prethodno istreniran!) i okolina za treniranje Agenta. Ako su ti uvjeti ispunjeni, klikom na izbornik "Import" izabiru se Agent i okolina, kao što je prikazano na donjoj slici.



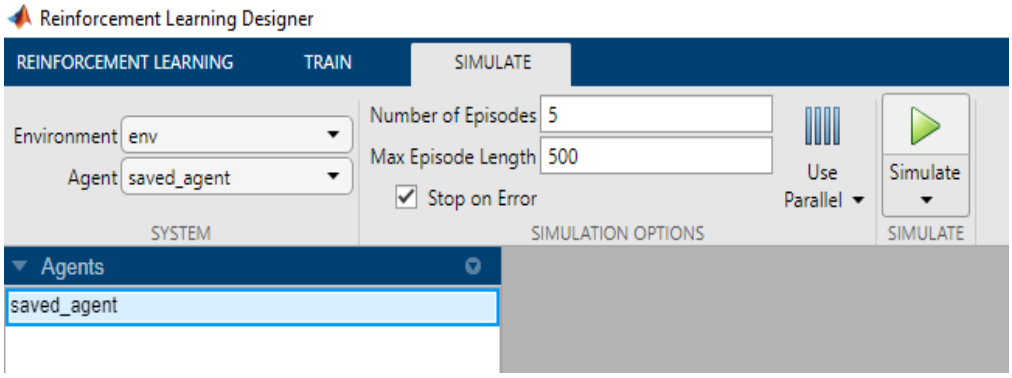
Slika 71 Izgled početne stranice RLD aplikacije

Slika 72 prikazuje izgled gotove simulacije RL Agenta (prikazanog na **slici 67**) u okolini definiranoj u **Simulink**-u (v. **slika 53**). Agent je simuliran 50 puta, te se na grafu vide nagrade pojedinačnih epizoda simulacije (plavi stupci), a vidi se i prosječna nagrada Agenta kroz 50 simulacija u okolini koja iznosi 174.528. Vidljive su oscilacije u nagradama kroz različite epizode, međutim, prosječna nagrada je dosta visoka i u usporedbi s ostalim Agentima spremljenim tokom procesa treniranja (bilo ih je više od 50), ovaj Agent se pokazao kao najbolji.



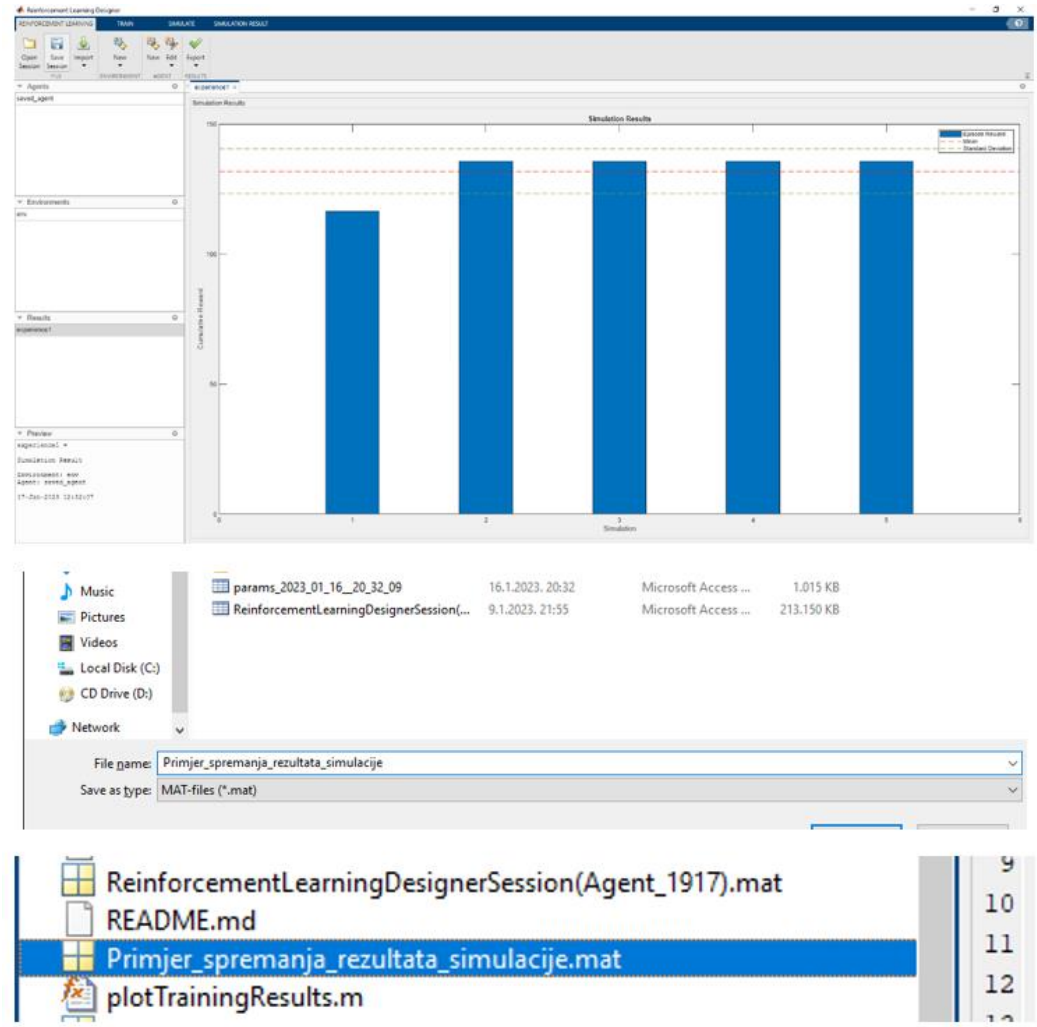
Slika 72 Rezultati gotove simulacije RL Agenta u definiranoj okolini

Slika 73 pokazuje način pokretanja simulacije Agenta u okolini. Određuje se broj epizoda simulacija kao i trajanje pojedine epizode. Osim toga, moguće je aktivirati i paralelno procesiranje koje je korisno kod zahtjevnijih simulacija radi bržeg izvođenja.



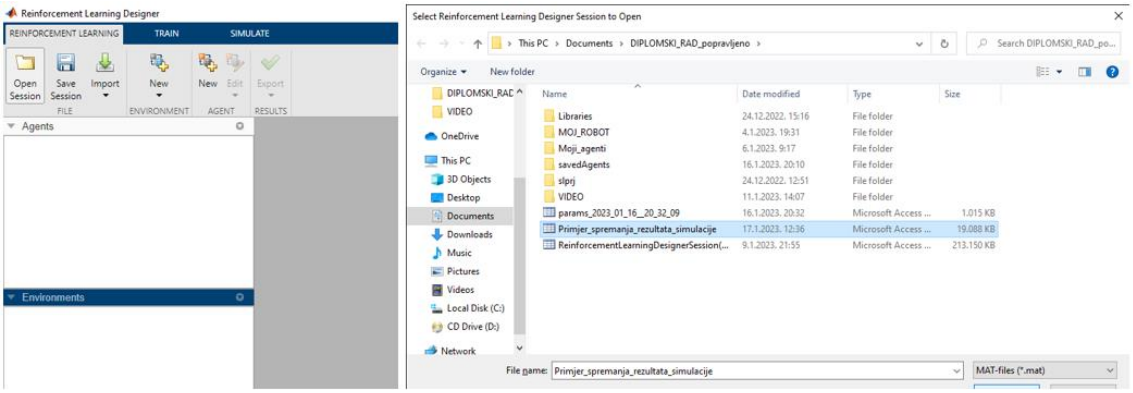
Slika 73 Pokretanje simulacije Agenta u okolini

Slika 74 prikazuje graf na kraju simulacije čije je pokretanje pokazano na **slici 73**. Na donjoj slici prikazan je i postupak spremanja simulacije. To se može napraviti klikom na izbornik "Save Session" unutar RLD aplikacije, nakon čega se odabire mjesto spremanja datoteke i njezin naziv. Na donjoj slici se također vidi da je Simulacija spremljena u mapi koja je dodana na Matlabov put (eng. Path), pa se nalazi u Matlabovom radnom prostoru.



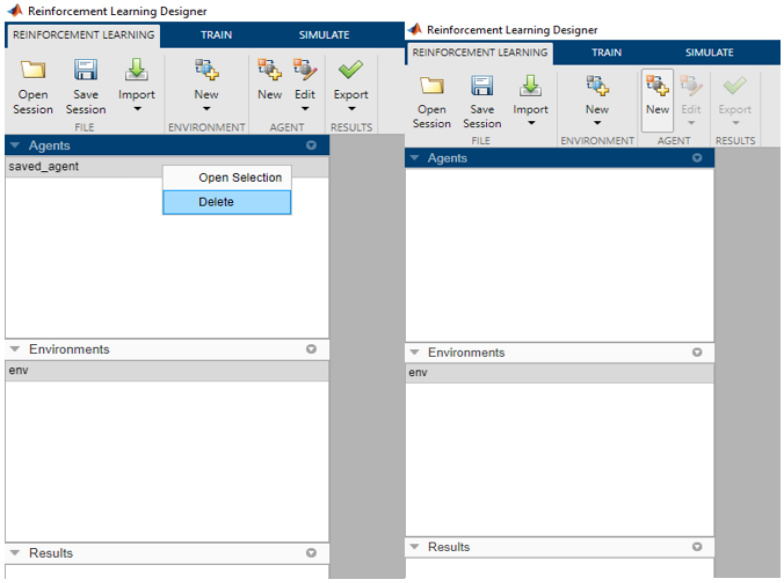
Slika 74 Graf na kraju prethodno pokrenute simulacije

Slika 75 prikazuje način otvaranja spremljenih simulacija. Klikom na izbornik "Open Session" unutar RLD aplikacije možemo odabrati prethodno spremljene simulacije, te ih po želji pregledavati i uspoređivati.



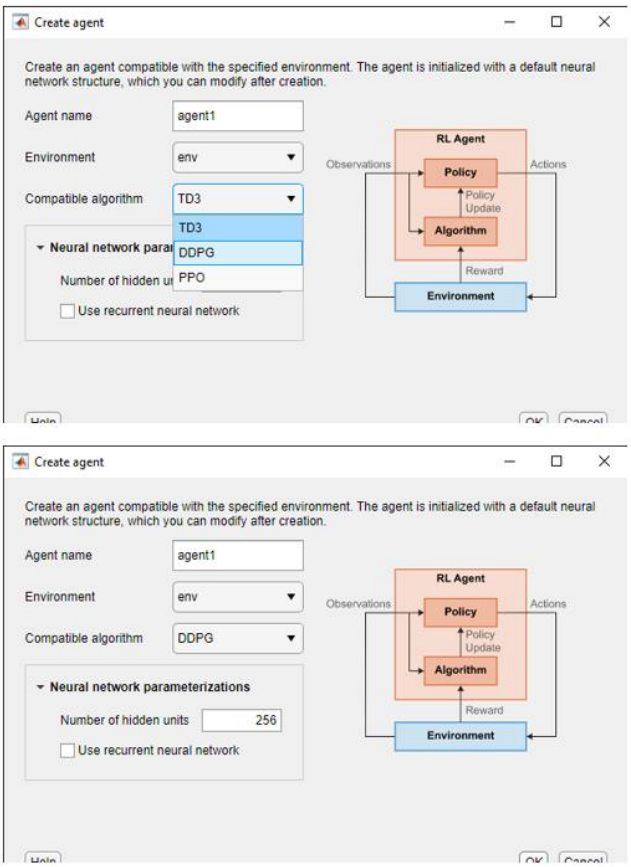
Slika 75 način otvaranja spremljenih simulacija

Ukoliko želimo koristiti druge agente i okoline, prethodno uvedeni agenti i okoline se mogu i brisati unutar aplikacije (radi preglednosti), taj postupak prikazuje **slika 76**. Ako imamo samo nekoliko Agenata i okolina koje želimo trenirati/simulirati unutar RLD aplikacije, možemo ih istovremeno držati u radnom prostoru bez brisanja, te izabirati željene Agente i okoline iz padajućih izbornika za potrebe simulacija/treninga unutar aplikacije. Donja slika također prikazuje način kreiranja novih RL Agenata. Kreiranje novog agenta vrši se klikom na ikonu "New" u izborniku "AGENT". Da bi se kreirao novi Agent, potrebno je prethodno učitati okolinu u kojoj će Agent trenirati!



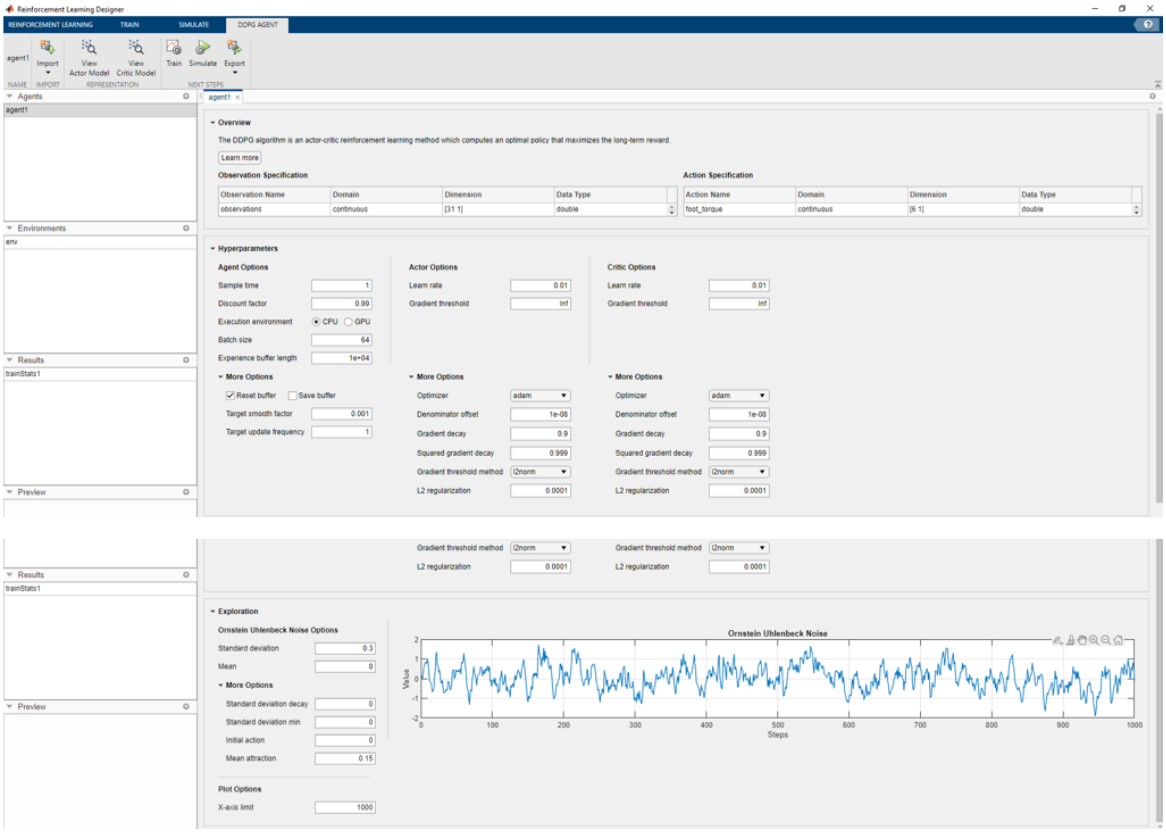
Slika 76 Brisanje Agenata i kreiranje novog Agenta

Slika 77 prikazuje prozor za kreiranje novog Agenta. Ime je dodjeljeno automatski, ali može se mijenjati po volji. Također se može izabrati RL algoritam po želji među ponuđenima (automatski se ponude RL algoritmi prikladni za učitanu okolinu!). Na primjeru sa donje slike izabran je DDPG RL algoritam, koji je i korišten za sva treniranja Agenata u ovom radu. Klikom na "OK" otvara se novokreirani Agent.



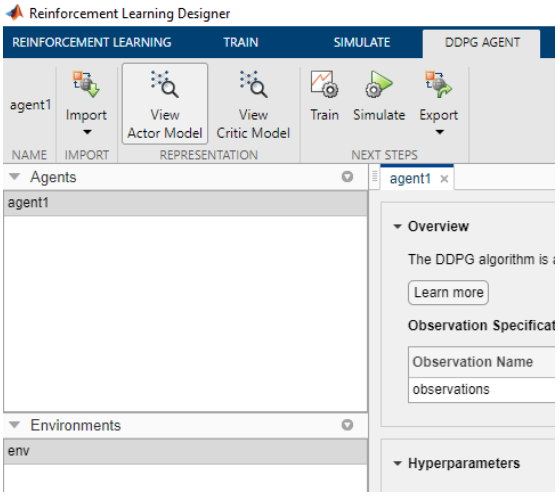
Slika 77 prozor za kreiranje novog Agenta

Slika 78 prikazuje izgled novog DDPG RL Agenta s automatski postavljenim opcijama Agenta.



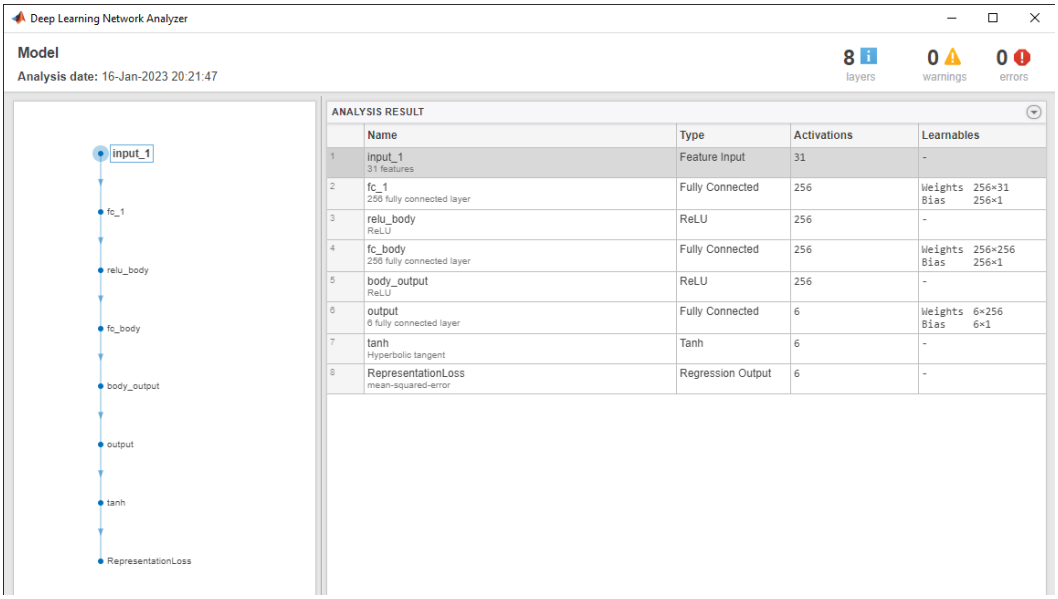
Slika 78 izgled novog DDPG RL Agenta s automatski postavljenim opcijama Agenta

Slika 79 prikazuje naziv novog Agenta koji se nakon stvaranja nalazi u radnom prostoru RLD aplikacije. Na slici se također vide ikone pomoću kojih je moguće pregledati izgled automatski generiranih umjetnih neuronskih mreža “Actor“ i “Critic“.



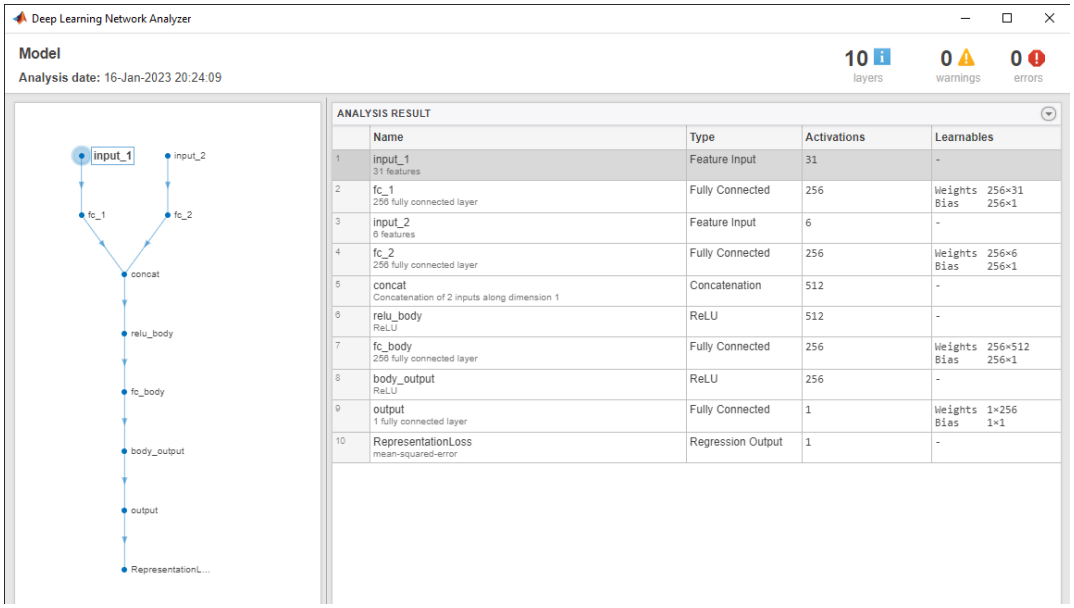
Slika 79 Način pregleda izgleda trenutnih neuronskih mreža Agenta

Slika 80 prikazuje izgled i svojstva automatski generirane "Actor" neuronske mreže RL DDPG Agenta.



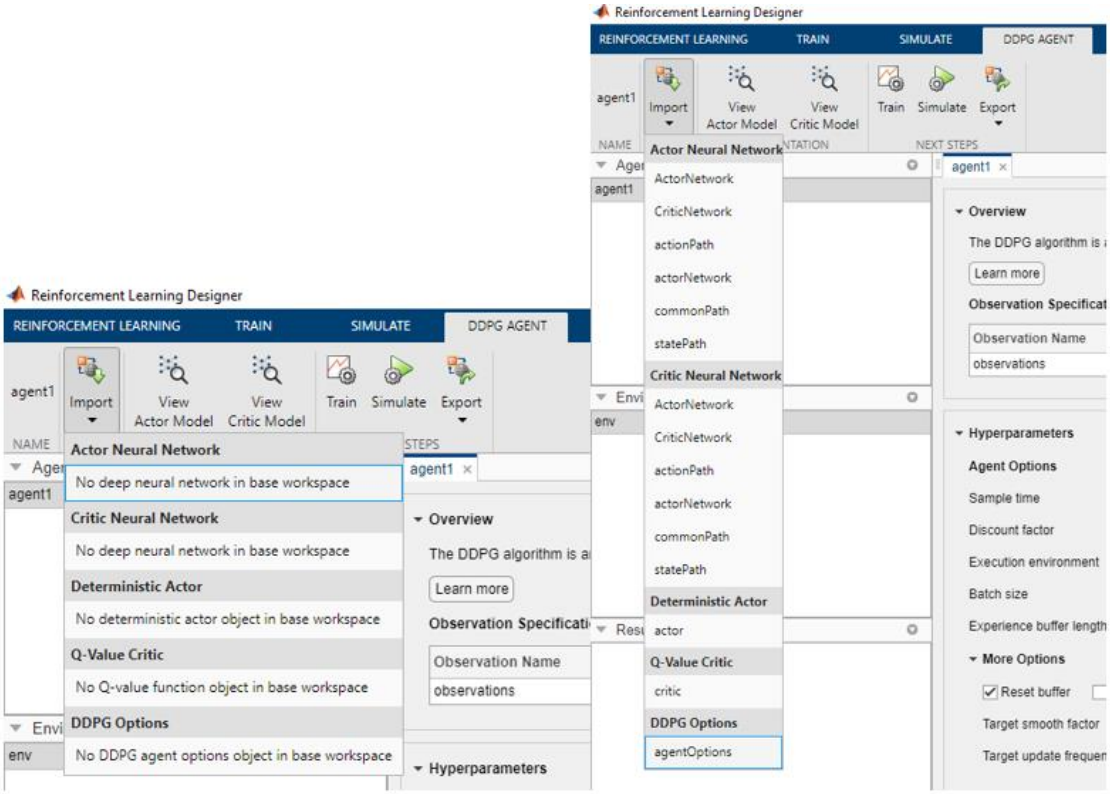
Slika 80 Izgled i svojstva automatski generirane "Actor" neuronske mreže

Slika 81 prikazuje izgled i svojstva automatski generirane "Critic" neuronske mreže RL DDPG Agenta.



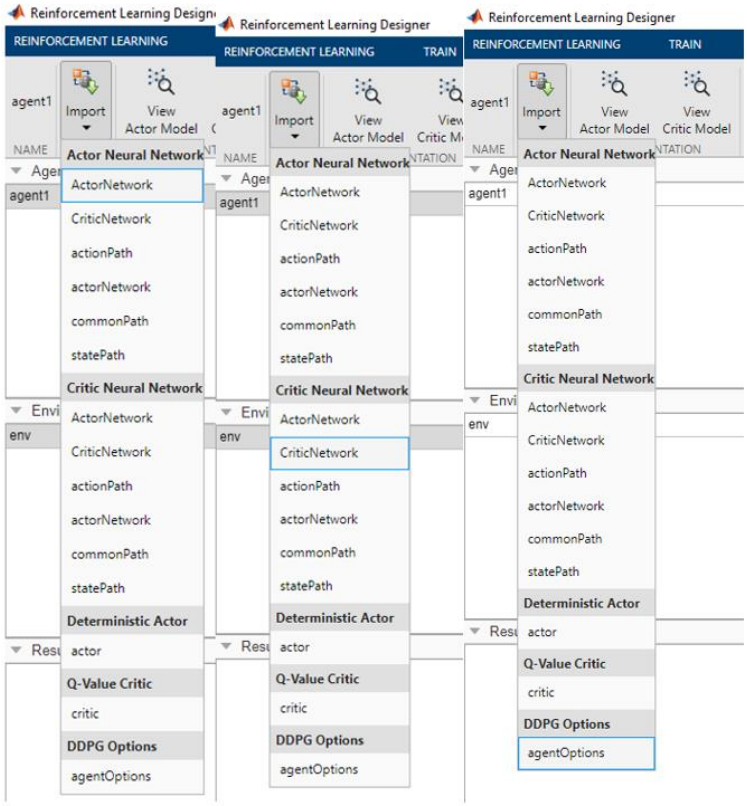
Slika 81 Izgled i svojstva automatski generirane "Critic" neuronske mreže

Rad unutar RLD aplikacije nije ograničen na Agente s automatski generiranim neuronskim mrežama. Ovisno o zadatku kojeg Agent treba naučiti izvršavati, različite strukture neuronskih mreža će rezultirati boljim ili lošijim učenjem Agenta. Zato je važno imati mogućnost kreiranja vlastitih neuronskih mreža po volji, te učitavanja istih pomoću RLD aplikacije. **Slika 82** prikazuje način učitavanja vlastitih neuronskih mreža koje će zamijeniti automatski generirane mreže. To se radi klikom na ikonu "Import", nakon čega se u padajućem izborniku prikazuju sve komponente dostupne za učitavanje koje se nalaze unutar radnog prostora Matlaba. Na donjoj slici je vidljivo da neće biti moguće učitati nove neuronske mreže bez prethodnog učitavanja mreža u radni prostor Matlaba.



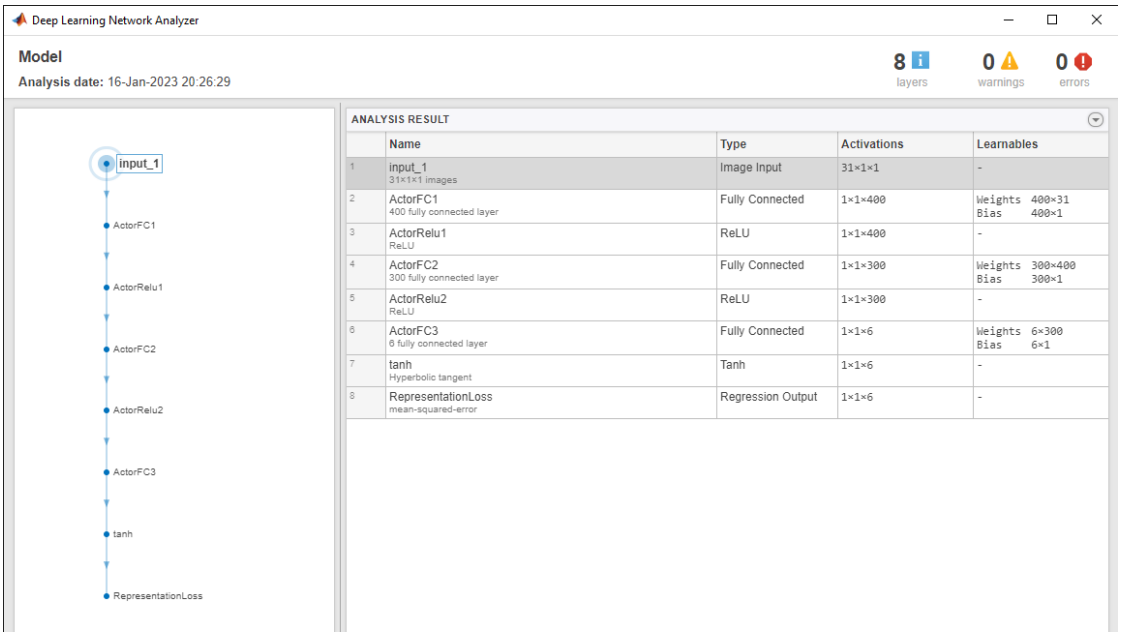
Slika 82 Način učitavanja vlastitih neuronskih mreža

Slika 83 prikazuje padajući izbornik s ponuđenim elementima iz radnog prostora Matlaba. Na slici su također prikazani izbori za "Actor Neural Network" i "Critic Neural Network" (koje su definirane u Matlab skripti "moje_neuronske_mreze.m"). Osim toga, mogu se učitati i elementi: "Deterministic Actor" i "Q-Value Critic" (također definirani u Matlab skripti "moje_neuronske_mreze"), te "DDPG Options" (definirane u Matlab skripti "createDDPGOptions.m"). Osim neuronskih mreža, donja slika prikazuje i učitavanje opcija za DDPG agenta.



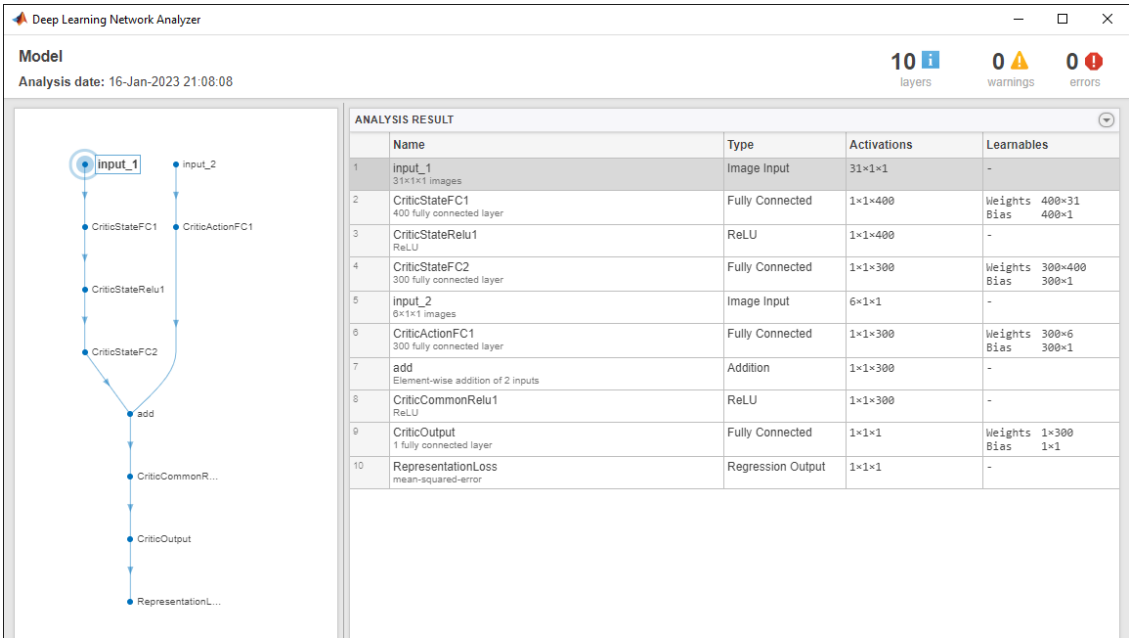
Slika 83 Padajući izbornik s ponuđenim elementima iz radnog prostora Matlaba

Slika 84 prikazuje izgled i svojstva "Actor" neuronske mreže (kreirane u Matlab skripti "moje_neuronske_mreze.m") za potrebe učenja s pojačanjem na primjeru dvonožnog hodajućeg robota.



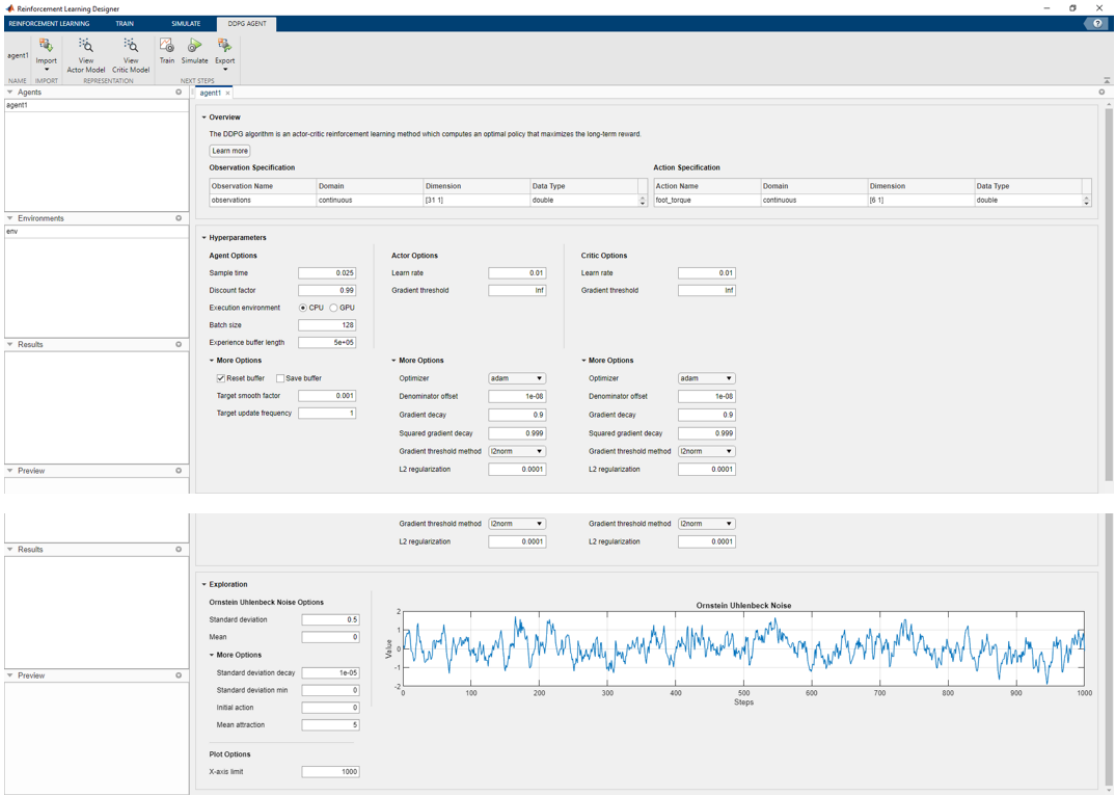
Slika 84 Izgled i svojstva kreirane "Actor" neuronske mreže

Slika 85 prikazuje izgled i svojstva "Critic" neuronske mreže (kreirane u Matlab skripti "moje_neuronske_mreze.m") za potrebe učenja s pojačanjem na primjeru dvonožnog hodajućeg robota.



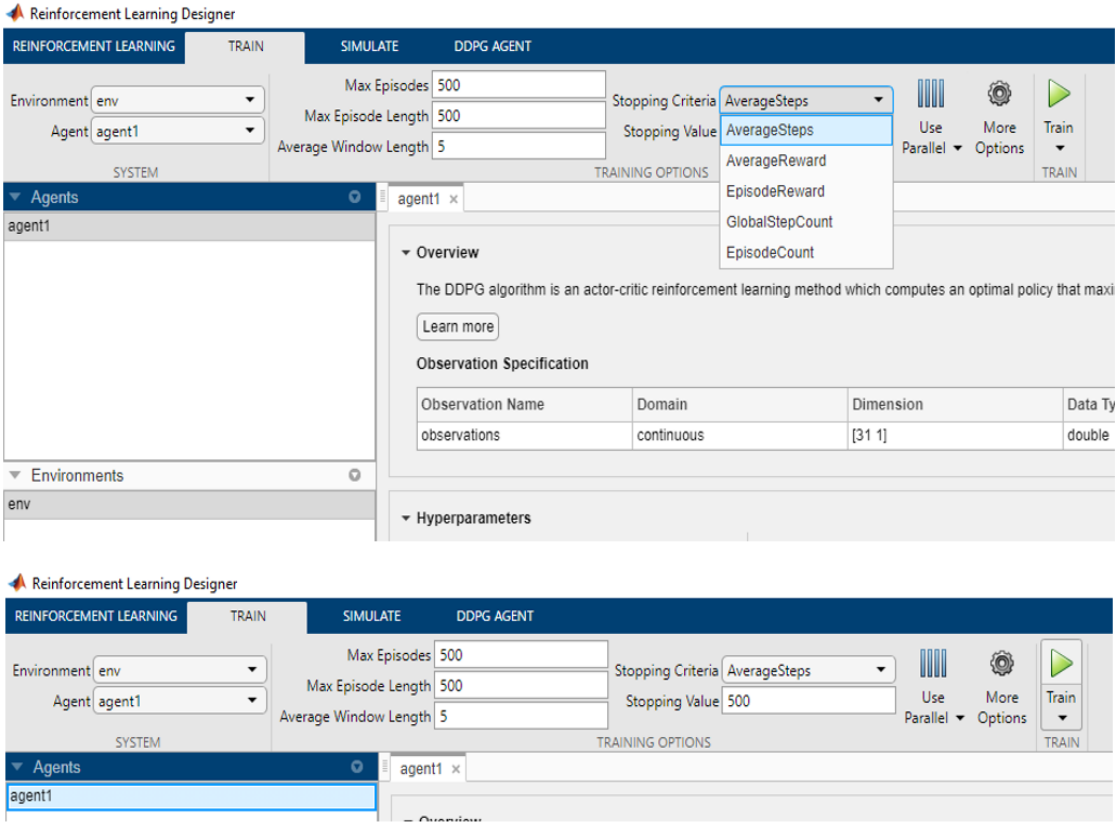
Slika 85 Izgled i svojstva kreirane "Critic" neuronske mreže

Slika 86 prikazuje nove učitane opcije DDPG Agenta (kreirane u Matlab skripti "createDDPGOptions.m") za potrebe učenja s pojačanjem na primjeru dvonožnog hodajućeg robota.



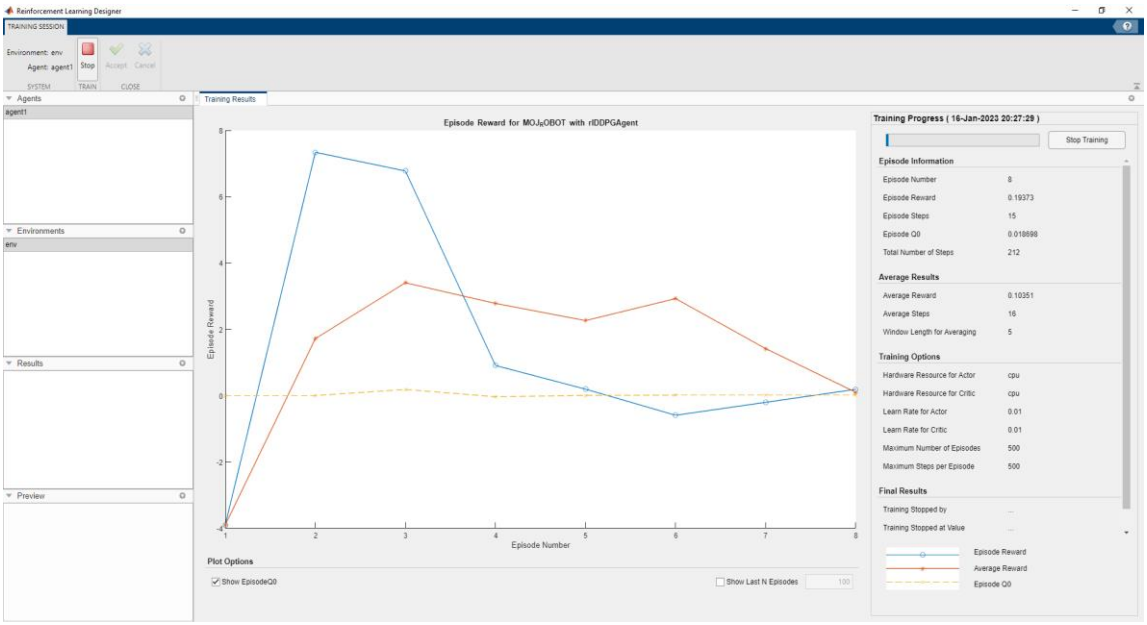
Slika 86 Nove učitane opcije DDPG Agenta

Slika 87 prikazuje postupak za pokretanje treniranja RL Agenta sa učitanim neuronskim mrežama i opcijama iz Matlabovog radnog prostora unutar učitane **Simulink** okoline (pod nazivom "MOJ_ROBOT.slx")



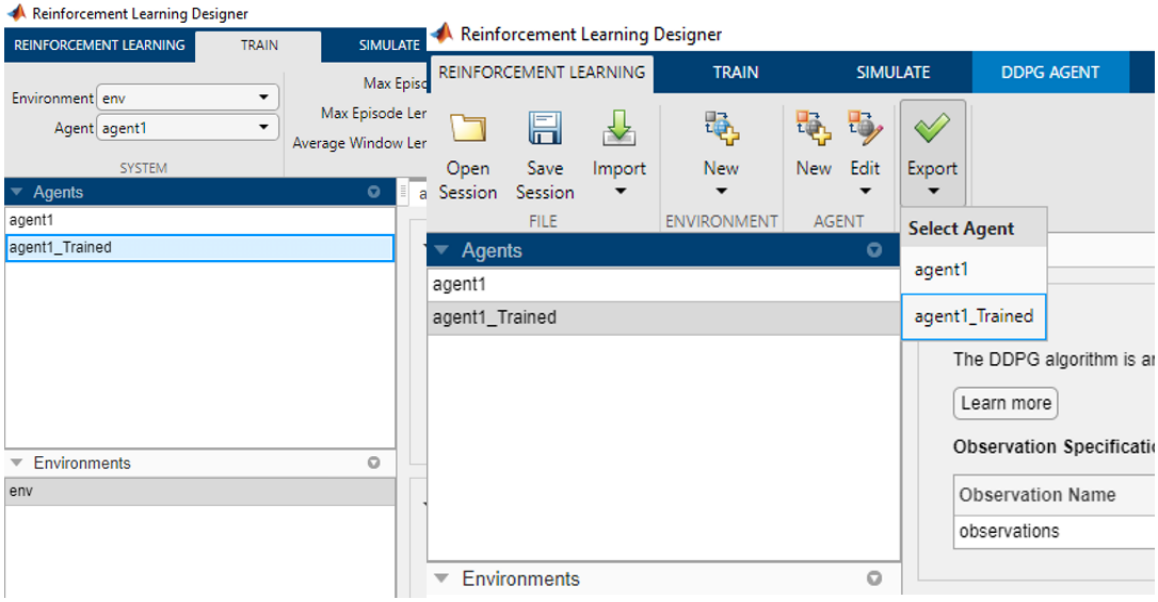
Slika 87 Postupak za pokretanje treniranja RL Agenta unutar RLD aplikacije

Slika 88 prikazuje graf nagrade nakon pokretanja učenja Agenta unutar RLD aplikacije (čiji je postupak prikazan na gornjoj slici).



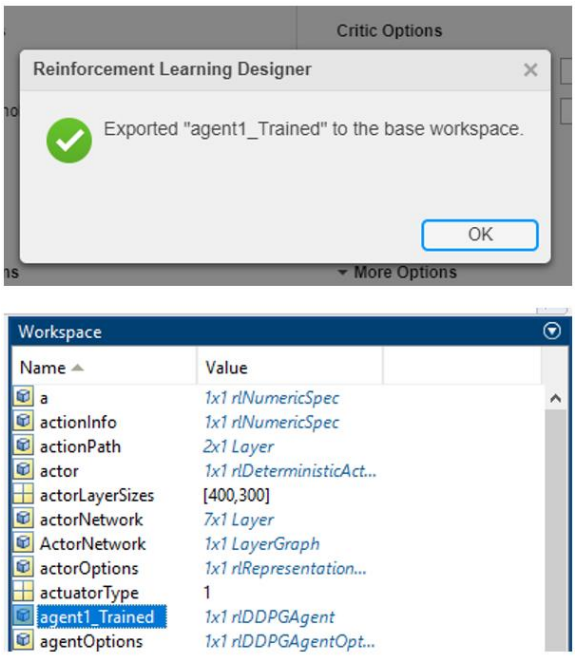
Slika 88 Graf nagrade nakon pokretanja učenja Agenta unutar RLD aplikacije

Slika 89 prikazuje istreniranog agenta koji je spremljen u radni prostor RLD aplikacije po završetku procesa treniranja sa **slike 88**. Donja slika također prikazuje postupak za "izvoz" (eng. Export) istreniranog agenta u Matlabov radni prostor.



Slika 89 Prikaz načina izvoza istreniranog Agenta

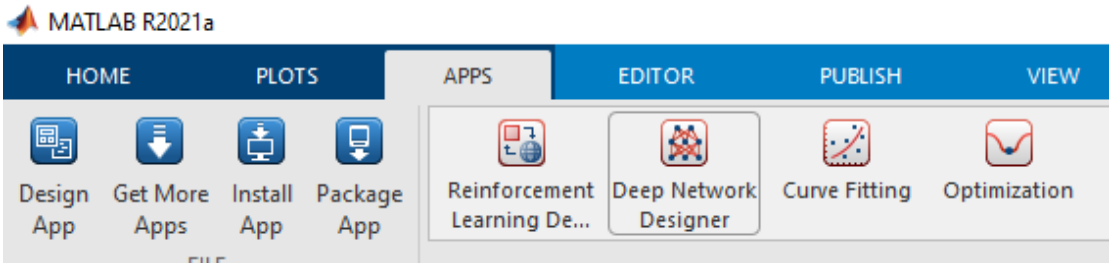
Slika 90 prikazuje poruku potvrde nakon izvoza istreniranog Agenta, te izvezenog Agenta koji se sada nalazi u radnom prostoru Matlaba. Nakon izvoza istrenirani Agent se može preimenovati i spremiti u mapu po želji, te se naknadno može simulirati unutar **Simulink** okoline ("MOJ_ROBOT.slx") ili unutar RLD aplikacije (unutar koje se može i dodatno utreniravati).



Slika 90 Prikaz izvezenog Agenta u radnom prostoru Matlaba

4.8. Opis funkcionalnosti "Deep Network Designer" (DND) aplikacije

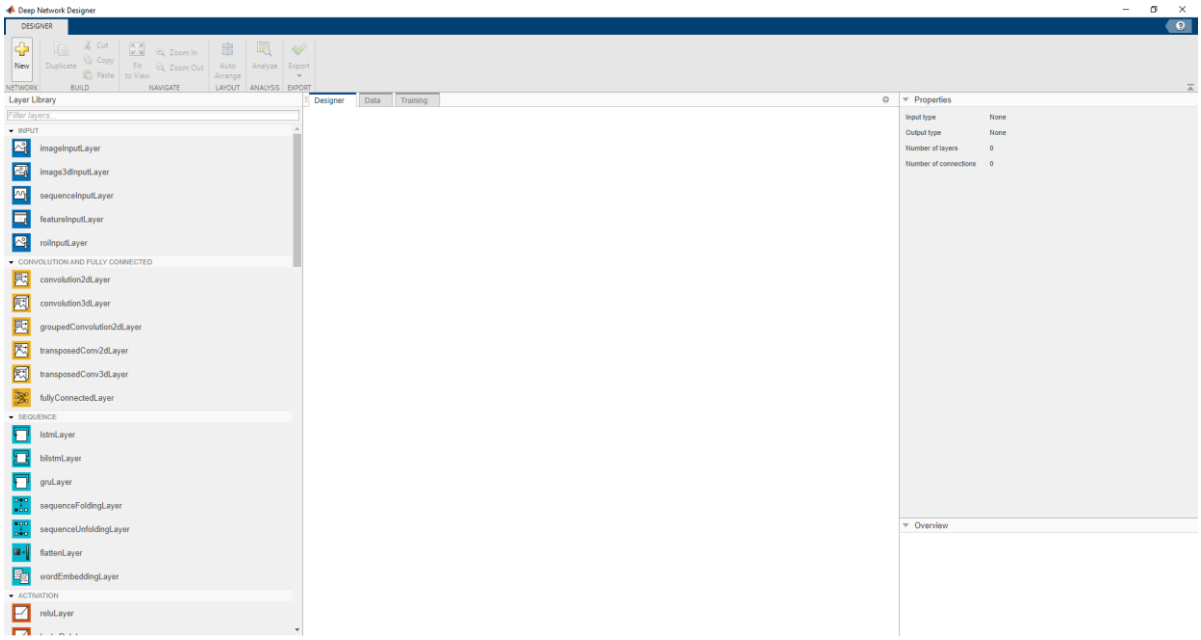
Slika 91 prikazuje način otvaranja "Deep Network Designer" (DND) aplikacije, odnosno aplikacije za kreiranje dubokih neuronskih mreža (NM sa više skrivenih slojeva).



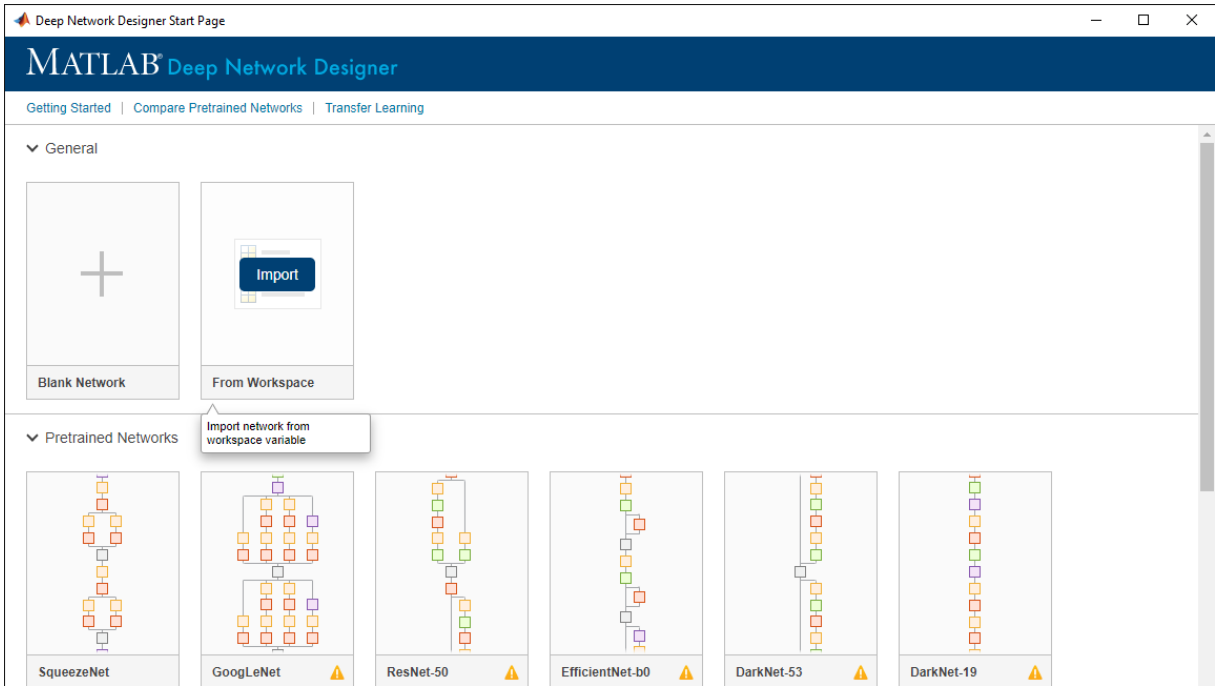
Slika 91 Način otvaranja "Deep Network Designer" (DND) aplikacije

Postoji i drugi način pokretanja iste aplikacije, a to je upisom naredbe: `deepNetworkDesigner` u Matlabu.

Slika 92 prikazuje početnu stranicu DND aplikacije nakon njenog pokretanja. Klikom na ikonu "New" mogu se učitati istrenirane neuronske mreže, kao i neuronske mreže kreirane u Matlab funkcijama od strane korisnika (prvo se moraju učitati u radni prostor Matlaba!). Taj postupak je prikazan na **slici 93**.

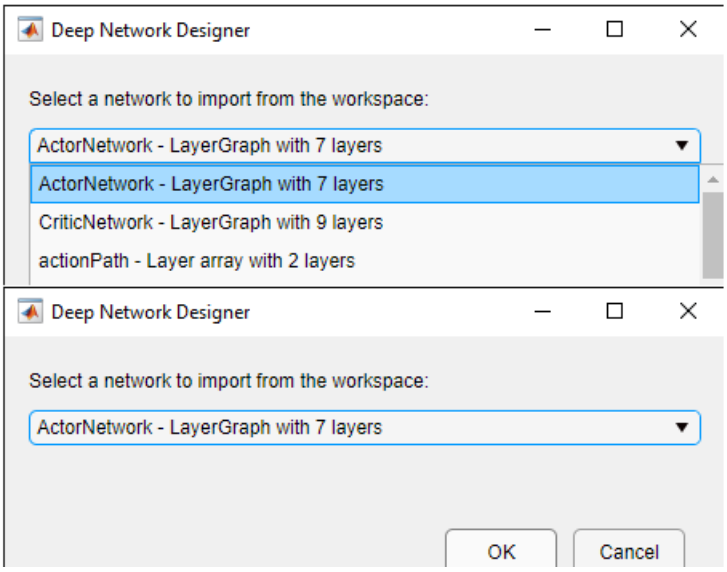


Slika 92 Početna stranica DND aplikacije



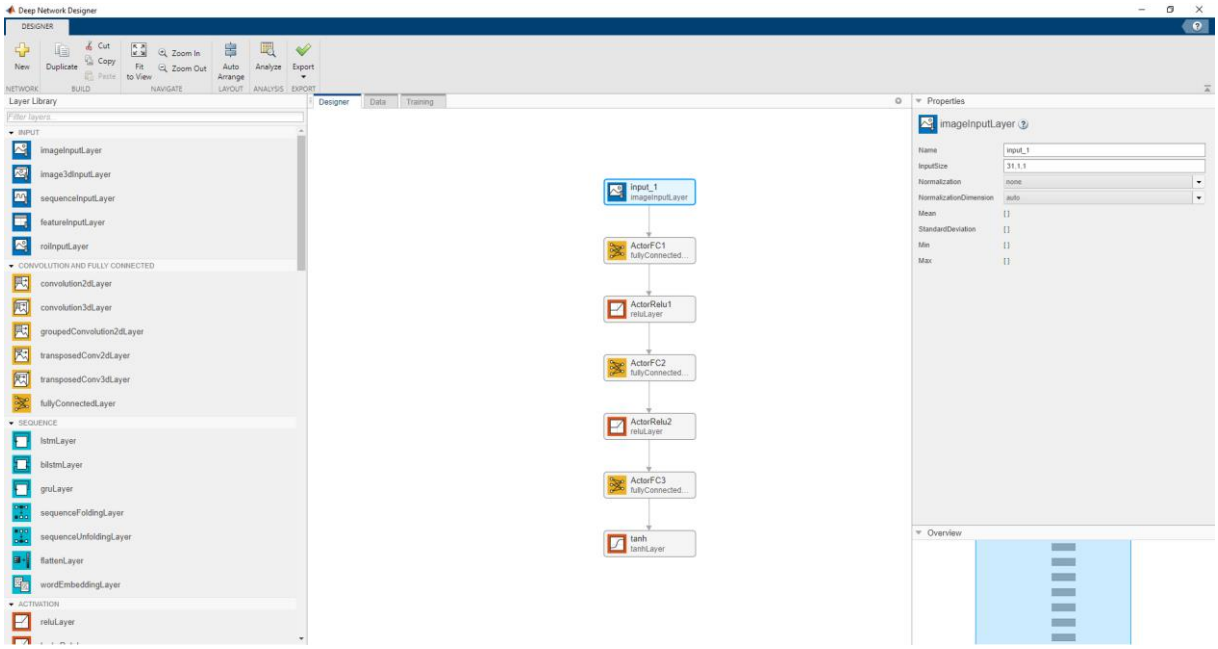
Slika 93 Učitavanje neuronskih mreža iz radnog prostora Matlaba

Slika 94 prikazuje postupak učitavanja "Actor" neuronske mreže iz radnog prostora Matlaba u DND aplikaciju.



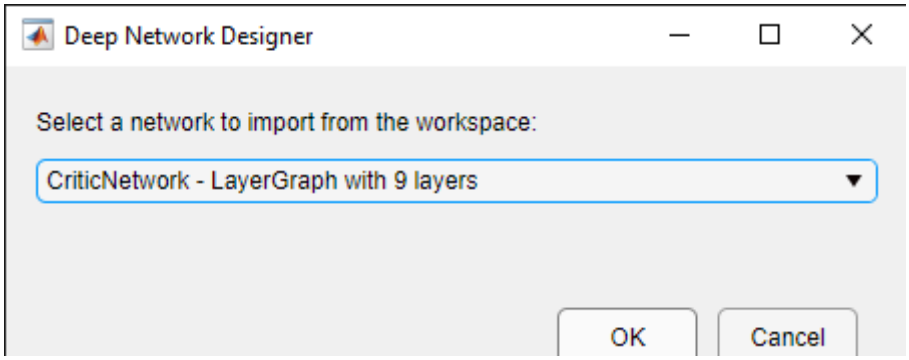
Slika 94 Učitavanje "Actor" neuronske mreže iz radnog prostora Matlaba u DND aplikaciju.

Klikom na "OK" iz prozora prikazanog na prethodnoj slici u DND aplikaciju se učitava izabrana neuronska mreža. Ta mreža je prikazana na **slici 95**. Nakon učitavanja mreže, mogu se vizualno pregledavati njezini slojevi i svojstva tih slojeva, te veze među slojevima neuronske mreže. Osim toga, mogu se brisati ili dodavati slojevi, odnosno uređivati neuronska mreža.



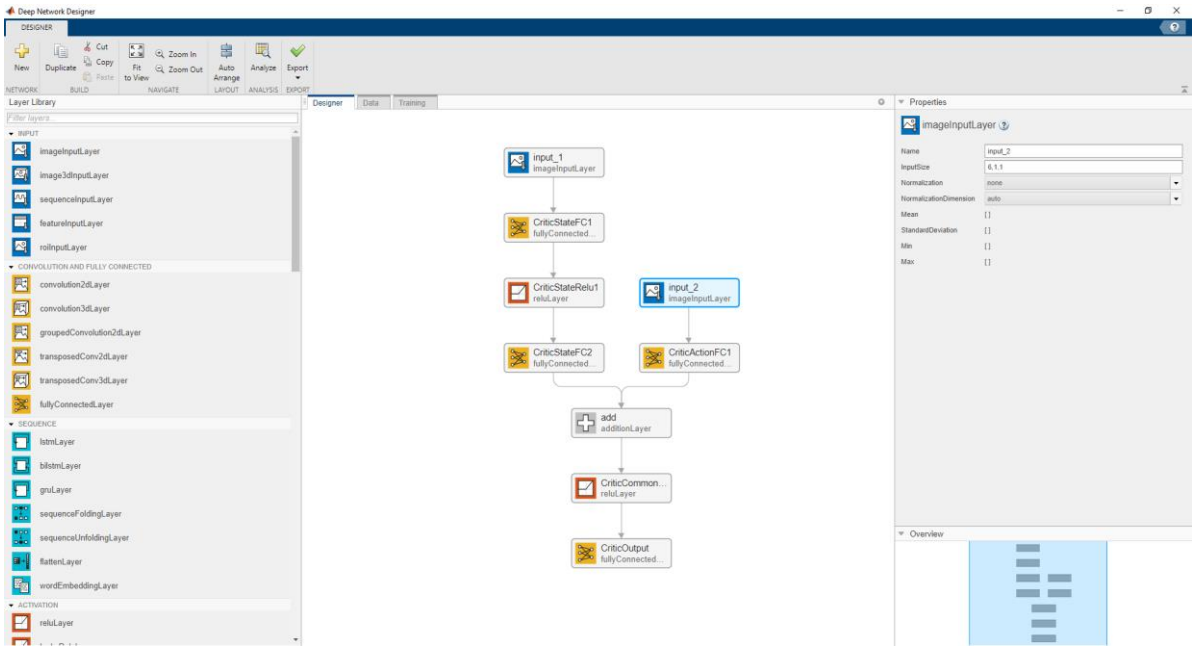
Slika 95 Prikaz "Actor" neuronske mreže učitane iz radnog prostora Matlaba

Slika 96 prikazuje postupak učitavanja "Critic" neuronske mreže iz radnog prostora Matlaba u DND aplikaciju.



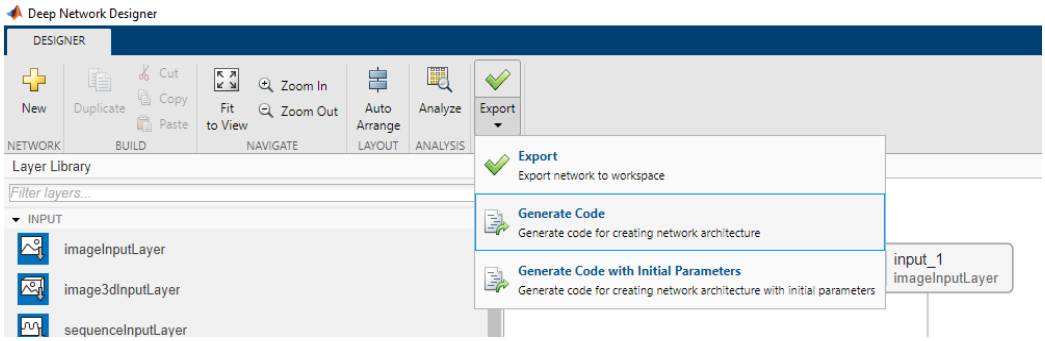
Slika 96 Učitavanje "Critic" neuronske mreže iz radnog prostora Matlaba u DND aplikaciju

Klikom na "OK" iz prozora prikazanog na prethodnoj slici u DND aplikaciju se učitava izabrana neuronska mreža. Ta mreža je prikazana na **slici 97** i za nju vrijedi isto kao za "Actor" mrežu.



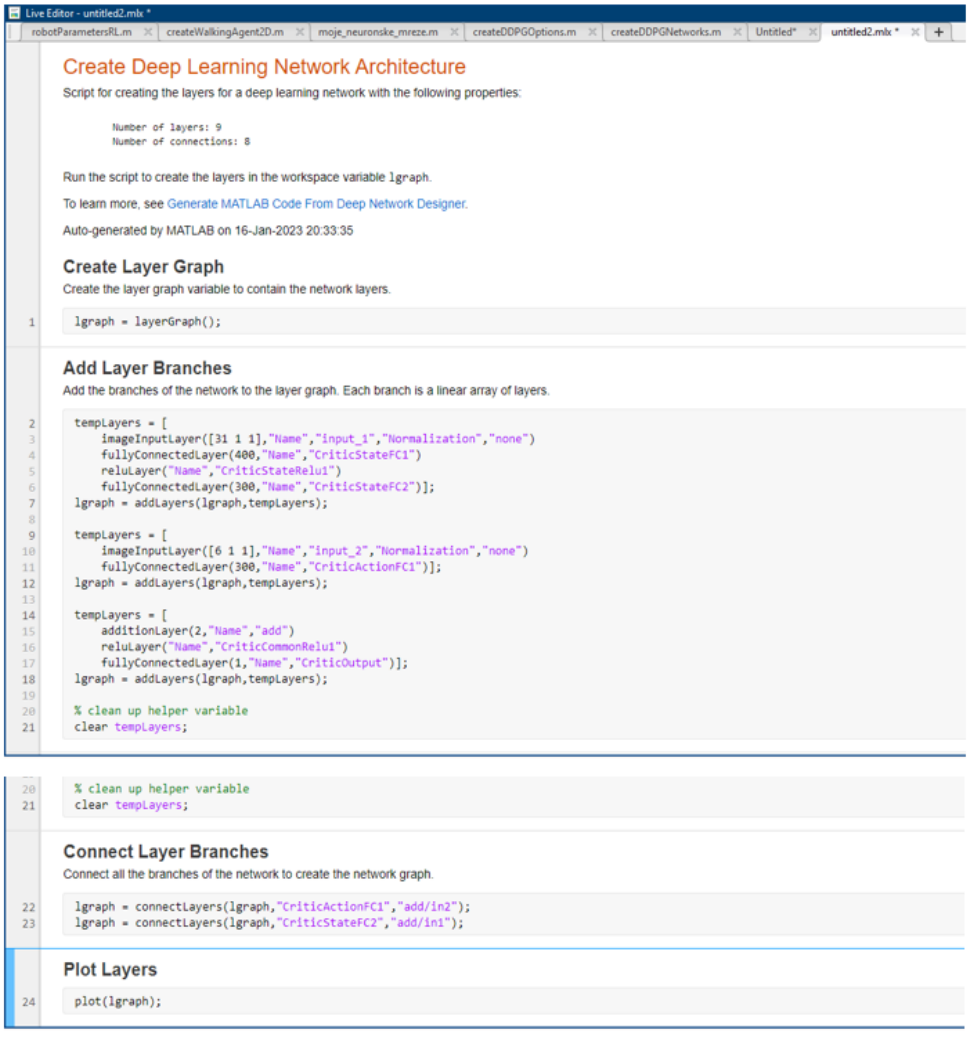
Slika 97 Prikaz "Critic" neuronske mreže učitane iz radnog prostora Matlaba

Nakon učitavanja mreže i eventualnih izmjena na njoj, može se generirati kôd te neuronske mreže, koji će se otvoriti u "Live Script" u Matlabu. **Slika 98** prikazuje postupak generiranja kôda neuronske mreže unutar DND aplikacije.



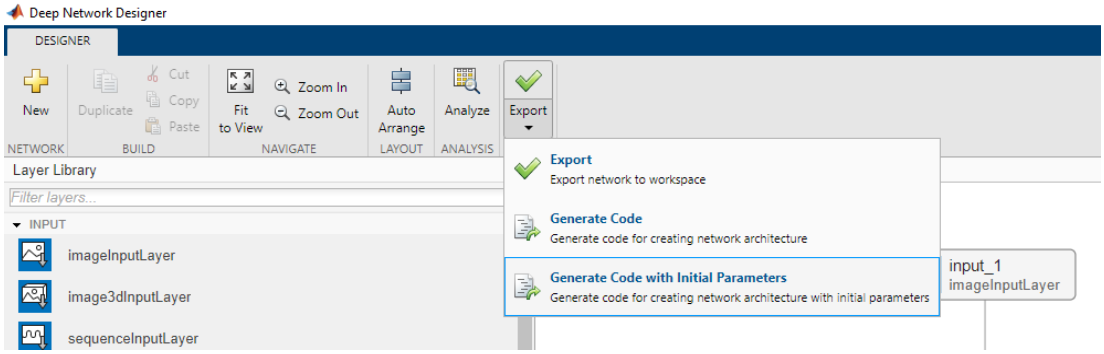
Slika 98 Postupak generiranja kôda neuronske mreže unutar DND aplikacije

Na **slici 99** je prikazan generirani kôd neuronske mreže (postupak generiranja je prikazan na gornjoj slici) otvoren u Matlab "Live Script".



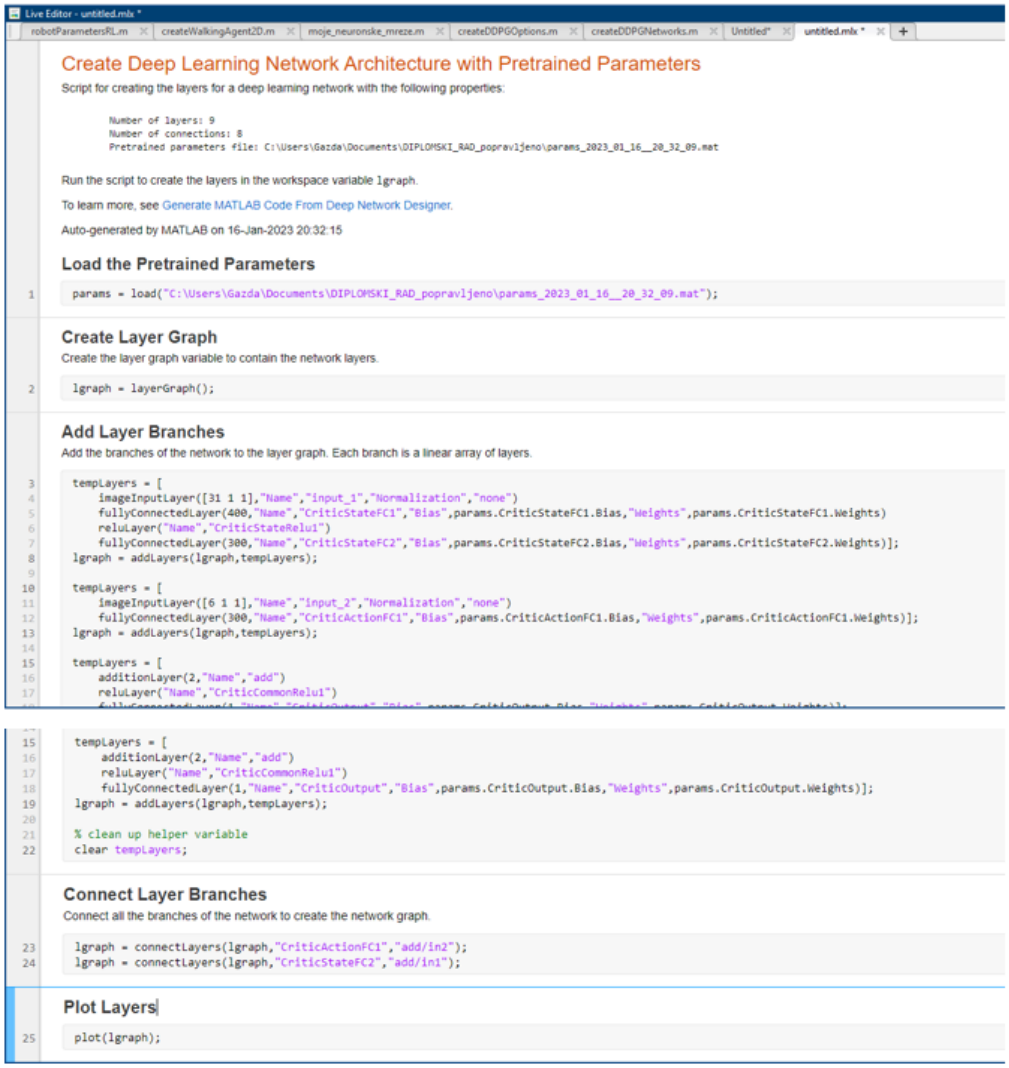
Slika 99 Generirani kôd neuronske mreže

Također se može generirati i kôd neuronske mreže s početnim parametrima. Taj postupak prikazuje **slika 100**.



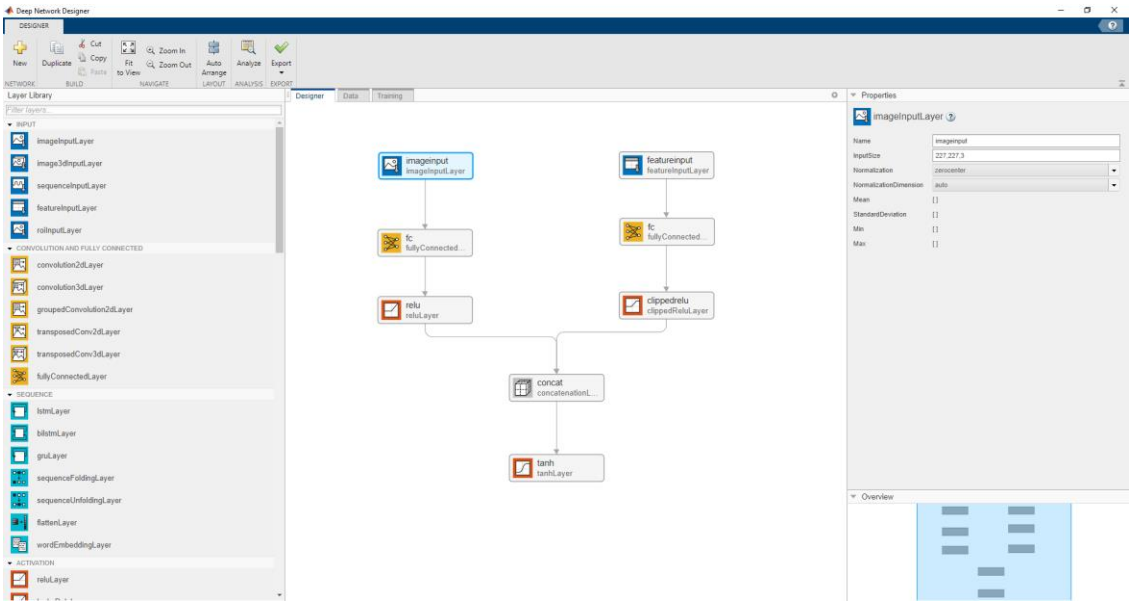
Slika 100 Postupak generiranja kôda neuronske mreže s početnim parametrima

I ovakav kôd se otvara u Matlabu u "Live Script", kao što prikazuje **slika 101**.



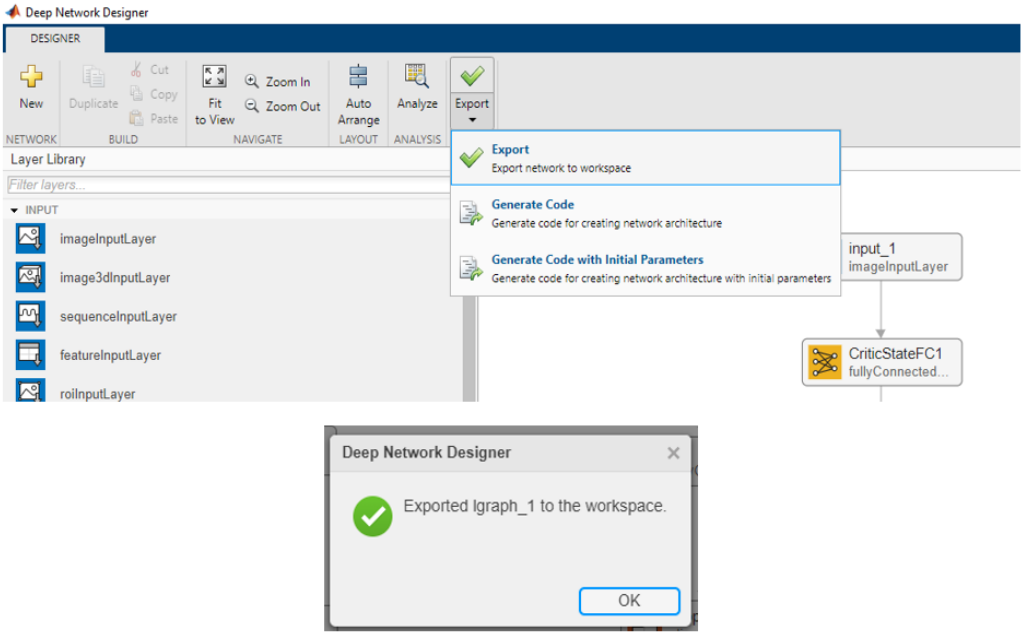
Slika 101 generirani kôd neuronske mreže s početnim parametrima

Kao što je već spomenuto, osim učitavanja neuronskih mreža iz Matlabovog radnog prostora, unutar DND aplikacije mogu se kreirati potpuno nove neuronske mreže. Aplikacija nudi niz vrsta slojeva za kreiranje neuronskih mreža. **Slika 102** prikazuje primjer neuronske mreže kreirane u DND Aplikaciji.



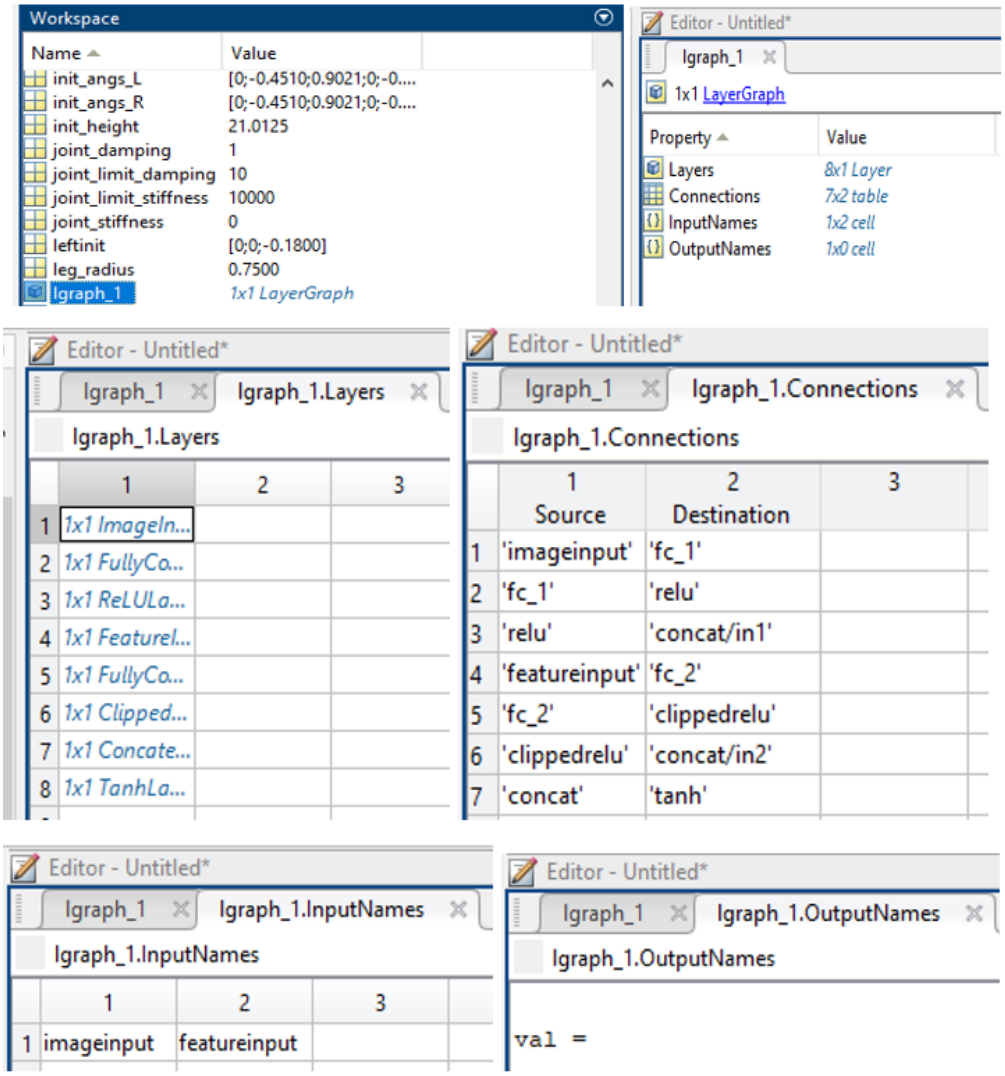
Slika 102 Primjer neuronske mreže kreirane u DND Aplikaciji

Nakon kreiranja mreže, ona se može (osim generiranja kôda) i izvesti direktno u radni prostor Matlaba kao "LayerGraph" objekt. Taj postupak prikazuje **slika 103**.



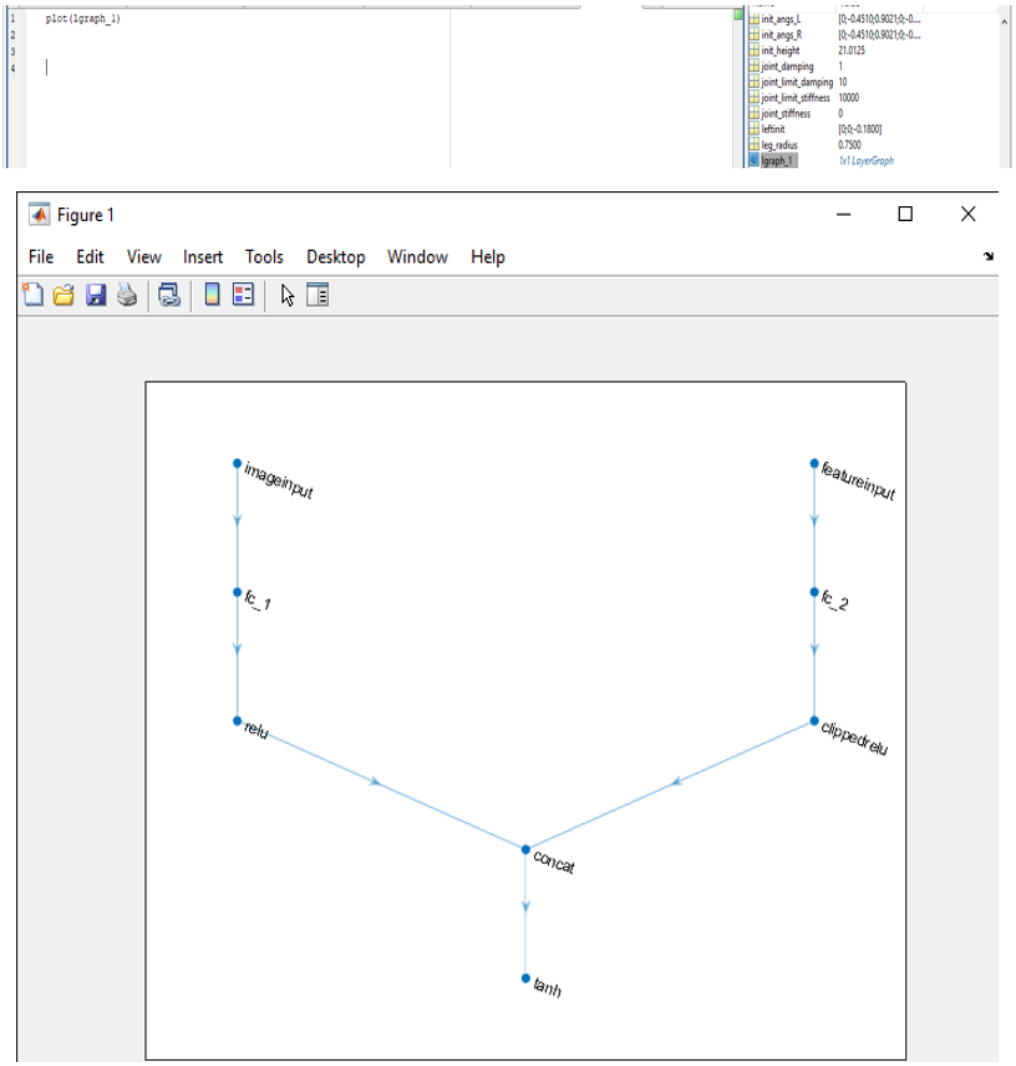
Slika 103 Postupak izvoza neuronske mreže direktno u radni prostor Matlaba

Slika 104 prikazuje "LayerGraph" objekt izvezene neuronske mreže unutar Matlabovog radnog prostora, te svojstva tog objekta



Slika 104 "LayerGraph" objekt izvezene neuronske mreže unutar Matlabovog radnog prostora

Za vizualizaciju izvezene neuronske mreže u Matlabu može se koristiti naredba: `plot(lgraph_1)`, gdje je "lgraph_1" automatski dodijeljeno ime neuronskoj mreži od strane DND aplikacije. To ime se može proizvoljno promijeniti. **Slika 105** prikazuje izgled izvezene neuronske mreže u Matlabu.



Slika 105 prikazuje izgled izvezene neuronske mreže u Matlabu

5. ZAKLJUČAK

Tokom izrade diplomskog rada upoznao sam se s metodom strojnog učenja s kojom do sada nisam imao nikakvog kontakta. Ova metoda učenja s pojačanjem pokazala se kao vrlo moćan alat za rješavanje kompleksnog problema upravljanja hodom dvonožnog modela robota. Taj problem je još i moguće riješiti tradicionalnim analitičkim metodama iz teorije upravljanja, međutim u robotici postoje i daleko kompliciraniji problemi upravljanja za koje je jako teško, a možda čak i nemoguće doći do rješenja analitičkim metodama (barem trenutno poznatim). Upravo se zbog takvih problema i razvijaju metode strojnog učenja, čija je osnovna ideja računalima stvoriti/omogućiti uvjete da uče i iterativno dolaze do nekih zakona upravljanja do kojih bi čovjek teško došao opremljen logičkim razmišljanjem i dostupnim analitičkim metodama. Iako su očite vrijednosti metoda strojnog učenja, postoje i neki problemi. Jedan od glavnih je to što nije jasno na koji je točno način umjetna neuronska mreža došla do zakona upravljanja hodom robota. U idealnom svijetu to možda i ne bi bio veliki problem, jer nam ne bi bilo toliko sporno što ne znamo kako se dođe do zakona upravljanja, bitno je samo da (u ovom slučaju) robot hoda. Međutim, jasno je da svijet u simulaciji nije identičan realnom svijetu. Ne možemo nikada s apsolutnom točnošću odrediti sve parametre i utjecaje u nekom realnom sustavu i prenijeti ih savršeno u simulaciju. Zbog toga zakon upravljanja koji savršeno pokreće model robota u simulaciji možda neće biti zadovoljavajući nakon implementacije na realnom hardveru. U nekim slučajevima se takvi problemi mogu anulirati naknadnim podešavanjem parametara regulacijskog sustava (u klasičnoj primjeni analitičkih metoda upravljanja). Međutim, ako je sustav kompleksan i ima veliki broj parametara koje je vrlo teško naštimati tako da se dobije optimalno ponašanje sustava, imamo problem. Kod učenja s pojačanjem cijela ta priča se dodatno komplicira činjenicom da su parametri upravljanja nekakvi težinski faktori smješteni u vezama između slojeva umjetnih neuronskih mreža i nisu dostupni za razmatranje niti naštimavanje. A čak i kada bi bili dostupni, to su neke brojčane vrijednosti za koje ne možemo utvrditi što točno fizikalno predstavljaju, pa opet ne bi bilo jednostavno njima manipulirati za postizanje optimalnog ili kvazioptimalnog ponašanja sustava. Iz tih razloga inženjeri i znanstvenici koji se bave razvojem raznih metoda strojnog učenja nastoje objediniti analitičke metode s metodama strojnog učenja, kako bi se iskoristile pogodnosti iz oba pristupa. Time bi se kompliciranim dijelovi u pronalaženju zakona upravljanja prepustili računalima, ali bi se implementirali i dijelovi klasičnih sustava upravljanja (neke regulacijske petlje s povratnom vezom) koji bi služili za naštimavanje upravljačkog sustava prilikom implementacije na hardveru.

LITERATURA

[1] https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-ml/(datum posjete 9. siječnja 2023.)

[2] https://www.javatpoint.com/artificial-neural-network (datum posjete 10. siječnja 2023.)

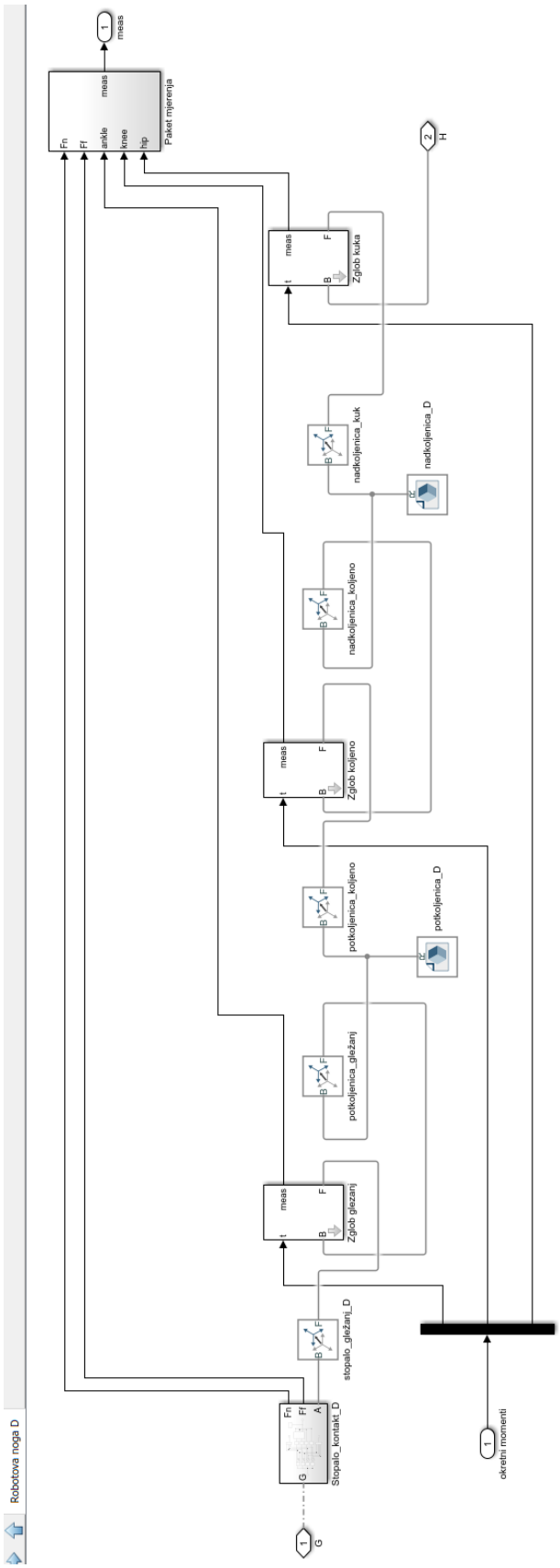
[3] https://www.javatpoint.com/artificial-neural-network-genetic-algorithm(datum posjete 10. siječnja 2023.)

[4] https://www.mathworks.com/help/reinforcement-learning (datum posjete 12. siječnja 2023.)

[5] https://www.mathworks.com/help/reinforcement-learning/ug/what-is-reinforcement-learning.html(datum posjete 12. siječnja 2023.)

PRILOZI

I. Uvećani prikaz modela desne noge robota



II. Uvećani prikaz modela robota prilagođenog za komunikaciju sa upravljačkim sustavom