Puno hvala mentorici, prof. dr. sc. Željki Mihajlović, na svim korisnim savjetima i pomoći.

Hvala roditeljima i obitelji na konstantnoj podršci i ustrajnom podržavanju moje znatiželje. Oni su učili hodati mene, sada ja učim hodati robote.

Hvala svim prijateljima, a posebno curi i ekipi "Glazkviz" na glazbenim prijedlozima i stalnoj potpori. Isto tako hvala Unitfly timu na ugodnom radnom iskustvu, provodima i razumijevanju.

SADRŽAJ

1. Uvod

Simulacije su je jedan od najznačajnijih alata prilikom izrade skupocjenih fizičkih tehnologija. Cilj simulacije je omogućiti testiranje stvarnih modela unutar računalnog softvera prije nego što se počne raditi na bilo kakvom konkretnom fizičkom proizvodu.

Roboti s inteligentnim sustavima sve su češća ispomoć ljudima u svrhe graditeljstva te transporta opreme i ljudi. Razvoj autonomnih vozila je impresivan te je njihova uporaba sve veći trend automobilskoj industriji.

Osim robota sa umjetnom inteligencijom, sve veća je primjena i inteligentnih softverskih rješenja u realizaciji konkretnih problema u području medicine, strojnog prevođenja teksta, pametnih sustava oglašavanja, itd.

U sklopu ovoga rada, bavit ćemo se simulacijom hoda bipedalnog modela robota u 3D svijetu. Cilj je izraditi simulaciju hoda te korištenjem neuronskih mreža istrenirati bipedalni model u svrhe samostalnog kretanja.

Srodan rad ovome je Wiklendt et al. (2008) u kojemu se trenirao hod bipedalnog modela uz pomoću neuroevolucijskih algoritama. Osim same umjetne inteligencije Wiklendt et al. (2008) se bavi i fiziologijom bipedalnog tijela te mehanikom bipedalnog hoda.

Zhang et al. (2019) se bavi proučavanjem modela učenja bipedalnog hoda konkretno u pomoću programskog paketa za simulaciju robota Webots koji koristi ovaj rad. Robot u tom radu osim hoda može mjenjati stil i parametre hoda kao što su: iskorak, odbojna sila noge od površine, "skakutanje". Sve je to ostvareno uz pomoć naprednih povratnih neuronskih mreža.

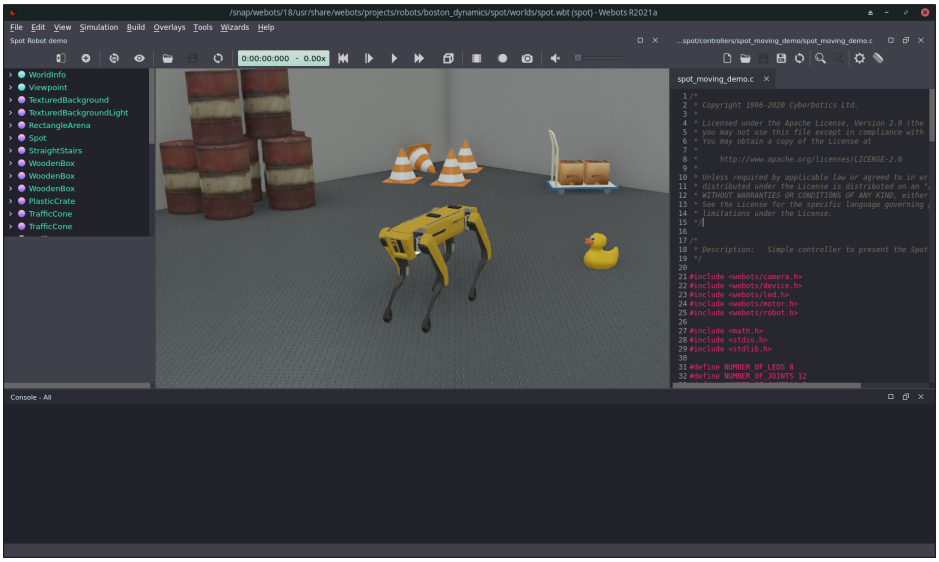
2. Programski paket za simulaciju robota Webots

Prilikom izrade ovoga rada korišten je softverski paket za simulaciju 3D robota Webots (2020).[1]. Webots je paket otvorenog koda (engl. *open source*) namijenjen za izgradnju prototipa 3D modela s fizikalnim svojstvima (masa, zglobovi, koeficijenti trenja) i robotskim komponentama (senzori, servo motori, GPS). Na svojoj stranici, Webots navodi kako je dobro prilagođen za istraživanja i edukacijske projekte. Njegova svrha pronašla se u: prototipima stvarnih robota, robotska lokomotivna svojstva, učenje robotike te istraživanjima u robotskim inteligentnim sustavima.

Implementacija fizike Webotsa temelji se na Open Dynamics Engine programskoj knjižnici (engl. *framework*). Open Dynamics Engine [2] implementira jako dobro dokumentirano C i C++ aplikacijsko programsko sučelje (engl. *application programming interface* - API) što omogućava da se Webots projekti pišu u jezicima C, C++, Java, Python i MATLAB. Osim što je podržan na svim operacijskim sustavima Webots dolazi sa svojim integriranim razvojnim okruženjem (engl. *integrated development environment* - IDE) (Slika 2.1).

[1] Webots - http://www.cyberbotics.com

[2] Open Dynamics Engine - https://www.ode.org/



Slika 2.1: Integrirano razvojno okruženje za WebotsR2021a

2.1. Gradivne jedinice Webots projekta

Webots projekt sadrži se od svjetova, kontrolera (engl. *controller*), robota te opcionalno, korisnički implementirane fizike. Komponente su međusobno u hijerarhijskoj ovisnosti, gdje je roditelj hijerarhije gradivna jedinica svijet. Simulacija se odvija nad jednim svijetom koji može imati više robota te svaki robot može imati više kontrolera, iako samo jedan može biti i aktivno vezan za njega. Prilikom izrade Webots projekta automatski se generira direktorijska struktura sa sljedećim direktorijima: *worlds*, *controllers*, *protos*, *plugins*.

2.1.1. Gradivna jedinica Svijet

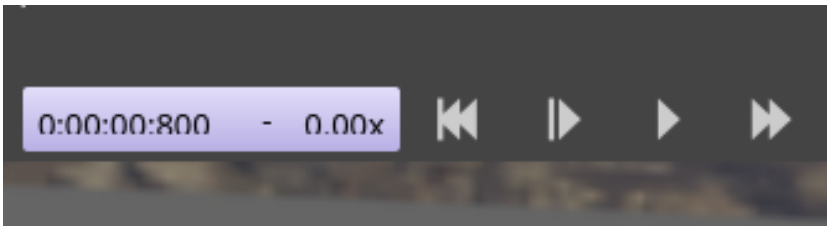
Osnovna gradivna jedinica Webots projekta je svijet (engl. *world*). Svijet sadrži opis svakog objekta, njegovu poziciju u svijetu te svojstva. Kako bi se točno znao odnos između objekata u svijetu svi objekti imaju svoju hijerarhiju. Primjerice ako robot sadrži senzore oni će biti enkapsulirani u objekt robota. U datoteku se svijet sprema s formatom `.wbt`. Datoteka ne sadrži implementaciju logike ponašanja objekata i njihove interakcije. Svi svjetovi korišteni u Webots projektu se nalaze u *worlds* direktoriju.

Instalacija Webots programskog paketa sadrži i razne probne (engl. *demo*) svjetove koji prikazuju osnovne funkcionalnosti robota. Jedan od takvih svjetova je i `highway.wbt` koji prikazuje osnovne funkcionalnosti robota *BmwX5* (Slika 2.2)

Slika 2.2: Prikaz svijeta highway.wbt

2.1.2. Kontrola ponašanja objekta

Program koji kontrolira ponašanje objekta u svijetu naziva se kontroler (engl. *controller*). Kao što je već navedeno svi kontroleri sve mogu pisati u jezicima C, C++, Java i MATLAB. Učitavanjem svijeta svi kontroleri na robotima se pokreću automatski te izvršavaju simulaciju (ako kontroler nema praznu logiku). Jednom pokrenuta simulacija može se resetirati, ubrzati pauzirati ili izvoditi korak po korak. Sva ta kontrola dostupna nam je na alatnoj traci Webots integriranog razvojnog okruženja (Slika 2.3).



Slika 2.3: Alatna traka za kontrolu simulacije Webots svijeta

"Hello, world!" tip programa[3] za Webots kontroler mogao bi izgledati na sljedeći način.

```
#include <webots/Robot.hpp>
#include <iostream>

int main()
{
  webots::Robot *robot = new webots::Robot();
  while (robot->step(32) != -1)
    std::cout << "Ah, ha, ha, ha, stayin' alive" << std::endl;
  delete robot;
  return 0;
}
```

Osim naravno uključivanja svih potrebnih biblioteka u program, najbitnije nam je inicijalizirati objekt klase `Robot` i koristiti njezinu virtualnu funkciju `robot->step(milliseconds)`. Poziv funkcije `robot->step(32)` u našem kodu računa stanje simulacije 32 milisekunde u budućnost simuliranog vremena. Moramo primijetiti kako se simulirano vrijeme razlikuje od stvarnog vremena što nam omogućava točnost simulacije na računalima različitih performansi. Može se dogoditi ako imamo dovoljno kompleksnu scenu da se onda izgenerira tek nakon pet sekundi bez obzira na to što je prošlo 32 milisekunde simulacije.

Povratna vrijednost funkcije `robot->step()` je -1 ako se prekida rad kontrolera što se može dogoditi zbog restartanja simulacije svijeta, gašenjem Webots programa, učitavanjem novog svijeta, itd. U našem primjeru programski isječak unutar `while` petlje će se ponavljati u beskonačnost sve dok ga ručno ne ugasimo.

Kontroler ne može direktno manipulirati tijekom simulacije (pauzirati, zaustavljati, ponovno pokretati). U tu svrhu uvodi se gradivna jedinica *Supervisor*. Ona nije korištena u ovoj implementaciji, ali bi se mogla koristiti primjerice u svrhu implementacije podržanog učenja kretnje bipedalnog modela.

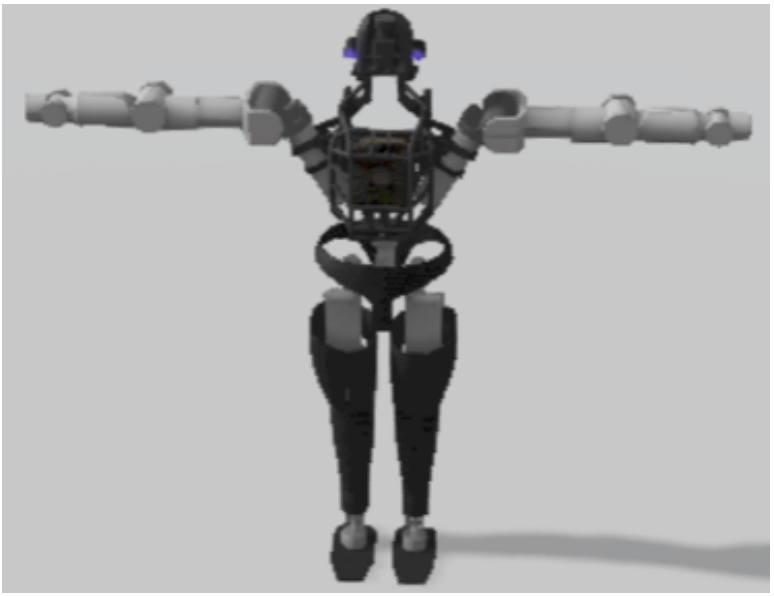
[3] "Hello, world" je najosnovniji tip programa, u određenom programskom jeziku, s kojem se novacima u programiranju pokazuje najjednostavniji mogući izvorni kod čiji izvršni kod ispisuje tekst "Zdravo, svijete!" *"Hello, world!"* na standardni izlaz.

2.1.3. Gradivna jedinica Robot

Modeli čija ponašanja programiramo u Webots svijetu su roboti. Webots programski paket podržava veliku listu modela robota iz stvarnog svijeta te stvaranje vlastitih modela robota. Kako se Webots često koristi u svrhe simuliranja, njegova svrha može biti veoma korisna prije stvaranja modela u stvarnome svijetu. Zbog toga, Webots sadrži razne modele poznatih robota kao što su četveronožni *Spot* i dvonožni *Atlas* (Slika 2.4) tvrtke *Boston Dynamics*.

Osim njih na listi[4] se mogu pronaći robota tvrtki Sony, Sphero, Fujitsu, DJI, itd. Svi navedeni modeli dozvoljeni su za besplatno korištenje pod *Cyberbotics* licencom[5].

Roboti na sebi mogu sadržavati razne senzore čiji se podaci mogu koristiti za implementaciju različitih funkcionalnosti. Senzori koje podržava programski paket Webots su: kamere, senzor udaljenosti, lidar te radar.



Slika 2.4: 3D model robota Atlas u programskog paektu Webots

[4]https://cyberbotics.com/doc/guide/robots

[5]https://cyberbotics.com/webots_assets_license

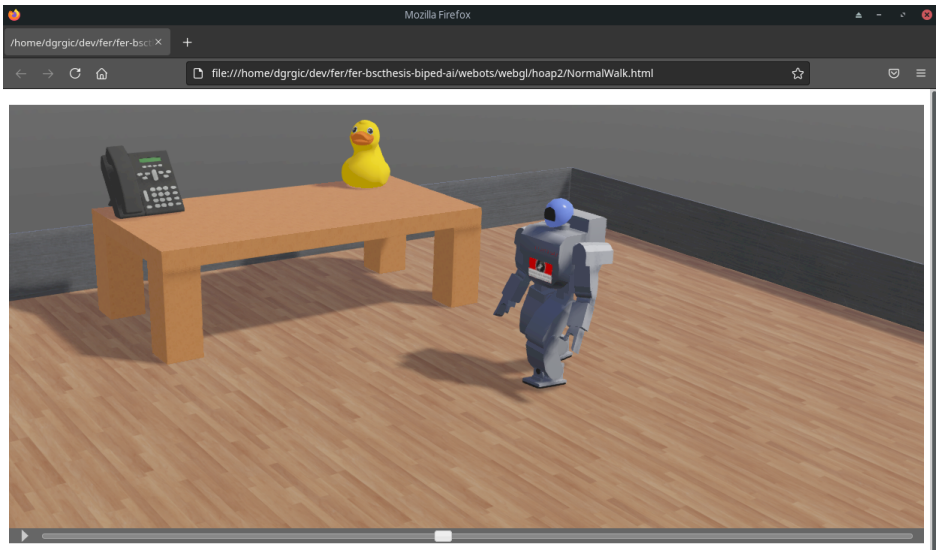
2.2. Simulacija Webots svijeta u web-pregledniku

Vrlo zgodna funkcionalnost Webotsa je što nam omogućava generiranje scene i animacije svijeta za web-preglednik (engl. *web browser*).

Web scena nam omogućava kretanje po svijetu te ručno pomicanje objekata, ako nam je to omogućeno. Scena ne sadrži informacije i simulaciji i kontrolere tako da se one koriste za prezentaciju statičnih objekata.

Web animacija nam omogućava reproduciranje interaktivnog videa Webots svijeta. Animacija se može izraditi snimanjem određenog djela simulacije koje se onda reproducira u web-pregledniku. Prilikom reprodukcije moguće je kretati se kamerom u prostoru svijeta te je pauzirati i premotavati. (Slika 2.5)

Nažalost zbog korištene knjižnice *three.js* temeljene na grafičkoj knjižnici WebGL, reprodukcije animacije u web-pregledniku se ne mogu pokretati u *Google Chrome* web-pregledniku[6].



Slika 2.5: Prikaz simulacije Webots svijeta u Firefox web-pregledniku

[6]https://cyberbotics.com/doc/guide/web-scene#remarks-on-the-used-technologies-and-their-limitations (lipanj, 2021.)

3. Umjetne neuronske mreže

Jedno od najznačajnijih grana računalne znanosti danas je umjetna inteligencija čija popularnost neupitno raste godinama. Dokaz tome je popularizacija servisa u obliku (engl. *cloud service*) čija svrha je dozvoliti pristup istraživačima rad na snažnim jedinicama napravljenim isključivo za obradu problema umjetne inteligencije. Jedan od takvih servisa je *Google Colaboratory*[1] koji omogućava besplatno (na određen broj ciklusa) ili uz vrlo malu nadoplatu korištenje tenzorskih procesorskih jedinica (engl. *tensor procesor unit* - TPU)[2]. Na ovaj način istraživači mogu procesirati kompleksne algoritme čije bi vrijeme izvršavanja bilo znatno veće na njihovim osobnim računalima.

Mogući pristup problemima umjetne inteligencije su umjetne neuronske mreže. Neuronske mreže[3] su sustavi temeljeni na biološkim neurološkim sustavima u tijelu čovjeka koji rješavaju probleme klasifikacije i funkcijske regresije.

Klasifikacija je postupak svrstavanja ulaznih uzoraka u razrede, a funkcijska regresija je postupak aproksimacije funkcije na ulaznim podacima (Čupić (2018)).

Snaga bioloških neurona (živčanih stanica) motivacija je za nastajanje umjetnih neurona kao računalne strukture koja omogućava drugačiji pristup računalno zahtjevnim problemima.

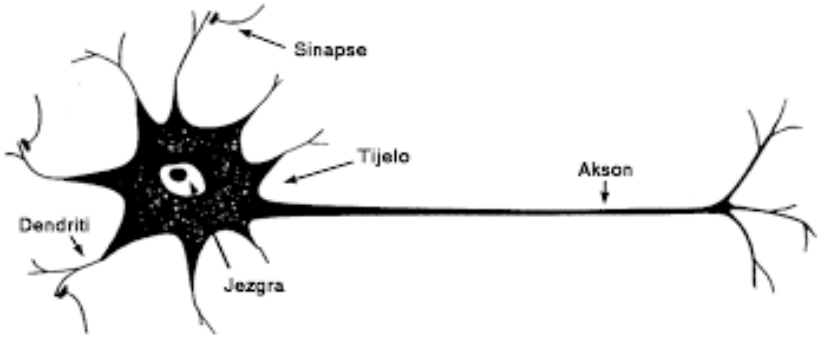
[1] https://research.google.com/colaboratory/

[2] https://cloud.google.com/tpu

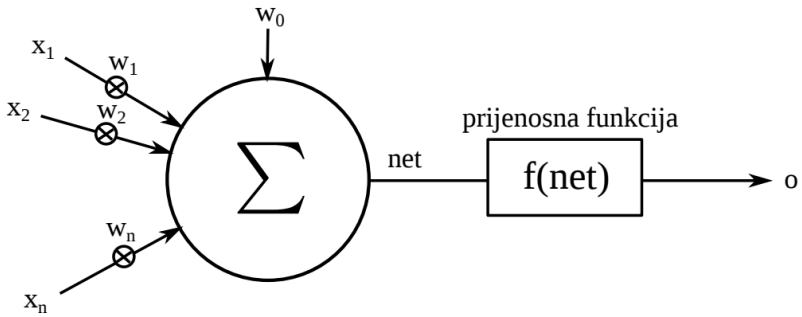
[3] U nastavku rada koristit ćemo naziv neuronske mreže za umjetne neuronske mreže bez gubitka značenja.

3.1. Struktura umjetnog neurona

Gradivna jedinica neuronske mreže je umjetni neuron (Slika 3.2). Njegova građa temelji se na biološkom neuronu (Slika 3.1) te se na takav način veže s ostalim neuronima u mreži.



Slika 3.1: Biološki neuron (izvor: hrcak.srce.hr)



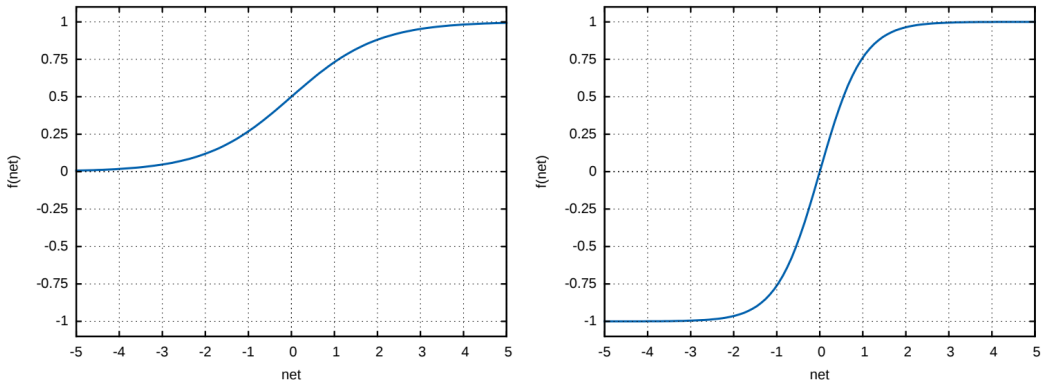
Slika 3.2: Umjetni neuron u strukturi neuronske mreže (izvor: Čupić (2018))

Možemo primijetiti da se ulazi u mrežu $x_1, x_2, \ldots, x_n$ ponašaju poput dendrita biološkog neurona. Ulazi $x_1, x_2, \ldots, x_n$ množe se s težinom na svakom ulazu $w_1, w_2, \ldots, w_n$. Prolaskom kroz čvor sa znakom sume na slici tj. funkciju net dobijemo vrijednost zadanu idućom formulom:

$$net = \sum_{i=1}^{n} x_i * w_i + w_0 \tag{3.1}$$

Nakon izračunavanja net vrijednosti računa se vrijednost prijenosne funkcije $f(net)$. Postoje razne prijenosne funkcije: sigmoidalna, funkcija skoka, funkcija identiteta, tangens hiperbolni, zglobnica (nepropusna i propusna), itd. U svrhu izrade ovoga rada korištene su sigmoidalna funkcija i tangens hiperbolni (Slika 3.3).

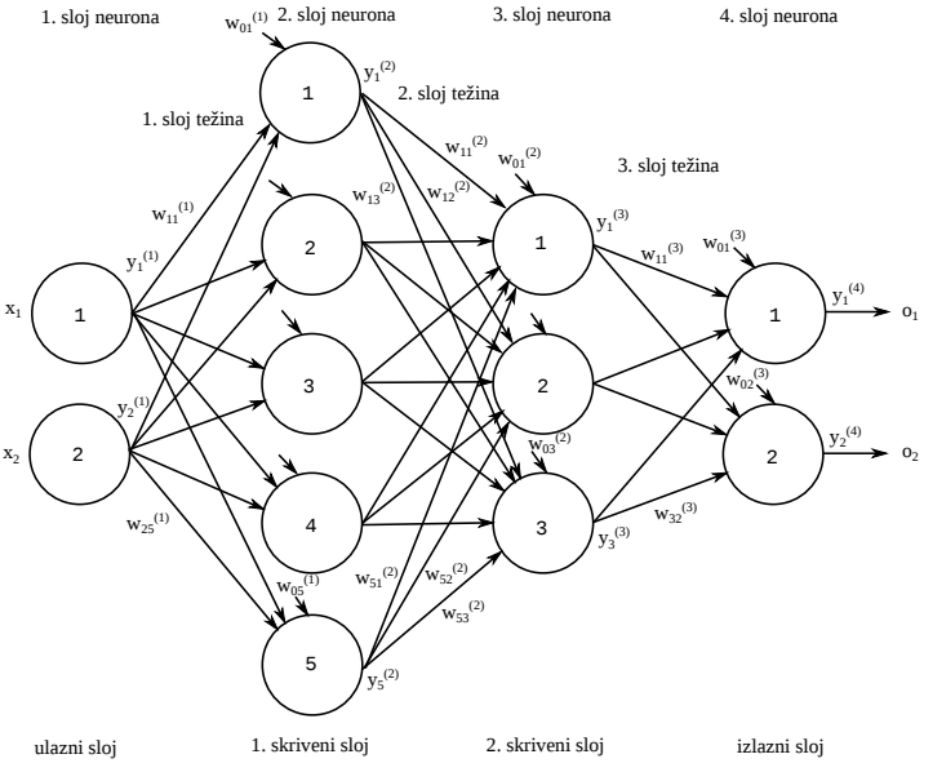
Jedan od poznatijih umjetnih neurona je TLU-perceptron (engl. *Treshold Logic Unit* čija je prijenosna funkcija funkcija skoka (Mcculloch i Pitts (1943))



Slika 3.3: Sigmoidalna funkcije i tangens hiperbolni (izvor: Čupić (2018))

3.2. Struktura neuronske mreže

Neuronske mreže se sastoje od tri djela: ulazni sloj, skriveni slojevi i izlazi sloj. Svaki sloj može imati jedan ili više neurona[4]. Ako neuron pod rednim brojem j u sloju i označimo kao $y_j^{(i)}$ onda je neuron $y_j^{(i)}$ povezan sa svim neuronima u sloju $i+1$ (osim ako je i izlazni sloj). Takvu strukturu grafički možemo prikazati slikom 3.4.



Slika 3.4: Prikaz neuronske mreže sa dva skrivena sloja (izvor: Čupić (2018))

[4]U nastavku rada koristit ćemo naziv neuron za umjetni neuron bez gubitka značenja.

Ako želimo opisati arhitekturu mreže to možemo učiniti tako da za svaki sloj napišemo broj neurona u sloju te takve brojeve povežemo u zapis znakom $\times$. Primjerice arhitektura mreže na slici 3.4 tada bi bila $2 \times 5 \times 3 \times 2$.

3.3. Korištenje neuronskih mreža

Korištenje neuronskih mreža se provodi u dva dijela: treniranje i predviđanje.

3.3.1. Treniranje

U prvom koraku potrebno je vektor vrijednosti značajki iz skupa podataka propagirati kroz neuronsku mrežu unaprijed od ulaznog do izlaznog sloja. Početne vrijednosti težina postavljene su nasumično te se za svaki neuron izračuna njegova net vrijednost korištenjem izraza 3.1. To je samo jedan prolazak kroz mrežu, a učenje se zapravo odvija tako da se korištenjem algoritma unazadnog širenja (engl. *backpropagation*) namještaju težine tako da se smanji greška predviđenih s obzirom na ulazne podatke. Prilikom učenja podatke dijelimo na skup za učenje i na skup za testiranje (u nekim slučajevima dodatno imamo i skup validacije). Bitno je naglasiti i da se skup podataka mora normalizirati na vrijednosti u rasponu [0, 1].

Algoritam unazadnog širenja

Algoritam unazadnog širenja konceptualno popravlja težine mreže tako da se smanji greška s obzirom na vrijednosti izlaznog sloja mreže. Dvije vrijednosti su nam bitne u ovom algoritmu δ i η. Grčko slovo δ nam predstavlja grešku u i-tom neuronu j-tog sloja, a η vrijednost između 0 i 1 koju zadajemo sami. Algoritam po koracima računa:

1. Izračunaj pogrešku i-tog izlaznog neurona sljedećim izrazom $\delta_i^{(j)} = y_i^{(j)} * (1 - y_i^{(j)} * (Y_i^{(j)} - y_i^{(j)})$ gdje $\delta_i^{(j)}$ predstavlja grešku i-tog neurona u j-tom sloju $y_i^{(j)}$ predstavlja predviđenu vrijednost u neuronu, $Y_i^{(j)}$ predstavlja stvarnu vrijednost.

2. Sada se od izlaznog sloja krećemo prema ulaznom sloju mreže te u ovom slučaju grešku računamo izrazom $\delta_i^{(j)} = y_i^{(j)} * (1 - y_i^{(j)}) * \sum_d (\delta_d^{(j+1)} * f'(w))$ gdje $f'(w)$ predstavlja derivaciju prijenosne funkcije, a $\delta_d^{(j+1)}$ greške sloja $j + 1$ koji su povezani s neuronom i.

3. Težinu za svaki neuron korigiramo izrazom $w_i^{(j)} = w_i^{(j)} + \eta * y_i^{(j)} * \delta_i^{(j)}$.

Za svaki ulaz u skupu podataka za treniranje odrađujemo ovaj algoritam dok se greška ne smanji na željenu vrijednost ili sami prekinemo učenje.

3.3.2. Predviđanje

Jednom takvu utreniranu mrežu ubacujemo svoje podatke te vršimo propagaciju unaprijed. Takve izlaze koristimo kao podatke u postupcima funkcijske regresije ili klasifikacije. Bitno je samo da prije korištenja podataka invertiramo normalizaciju tako da nam vrijednosti poprimaju raspone domene kakva je bila u ulaznome skupu.

3.4. Funkcijske knjižnice za implementaciju neuronskih mreža

U svrhu izrade modela u ovome radu korištene su programske knjižnice za izradu neuronskih mreža Sklearn (Pedregosa et al. (2011)) i Keras(Chollet et al. (2015)). Ove dvije biblioteke omogućavaju brzu implementaciju neuronske mreže, njezino korištenje te spremanje i učitavanje već utreniranih modela u programskom jeziku Python.

Stvoriti mrežu sa slike 3.4 putem ovih programskih knjižnica možemo napraviti idućim izvornim kodom:

```
model = Sequential([
    Dense(2),
    Activation('sigmoid'),
    Dense(5),
    Activation('sigmoid'),
    Dense(3),
    Activation('sigmoid'),
    Dense(2)
])
```

U ovome programskom isječku objekt `Dense` predstavlja obični sloj neuronske mreže povezan kao na slici 3.4. U njegov konstruktor navodimo broj neurona u sloju. Između

svakog sloja definiramo prijenosnu funkciju za neurone tog sloja korištenjem objekta `Activation`. Taj objekt u svoj konstruktor prima tip prijenosne funkcije, u našem slučaju to je sigmoidalna funkcija.

Objekt `model` koji smo stvorili možemo koristiti pozivajući funkcije `model.fit()` i `model.predict()`. Osim osnovnih funkcionalnosti funkcijska knjižnica Keras omogućava nam funkcionalnosti poput automatskog zaustavljanja učenja nakon određene minimalne pogreške te pozivanje korisničke funkcije prilikom ostvarivanja događaja (engl. *callback functions*).

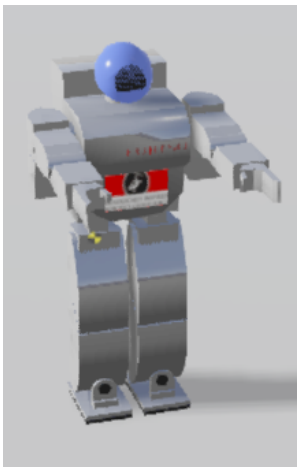
Isto tako možemo definirati maksimalan broj epoha tj. broj ponavljanja po skupu podataka za treniranje prije prestanka učenja. Prema predodređenim postavkama model će podijeliti skup na skup za treniranje i skup za testiranje u postotcima 70% i 30%.

4. Implementacija

U svrhu implementacije kontrole ponašanja robota korišten je programski jezik C++ i programski paket Webots, a za generiranje modela neuronskih mreža, crtanje grafova i generiranje novih podataka korišten je programski jezik Python i već navedene knjižnice Sklearn i Keras.

4.1. Model robota

Robot Fujitsu HOAP2 (engl. *Humanoid for Open Architecture Platform*) je model čovjekolikog robota visine 48 cm s 25 stupnjeva slobode (Slika 4.1). Detaljan popis zglobova nalazi se u Dodatku 7.1. Svaki zglob u 3D prostoru ima mogućih 6 stupnjeva slobode a to su: translacija u svakoj od tri koordinate prostora te rotacija u svakoj od tri koordinate prostora. U Webots modelima svaki stupanj slobode za određeni zglob ima svoju vrijednost u dvostrukoj preciznosti koju možemo postaviti.

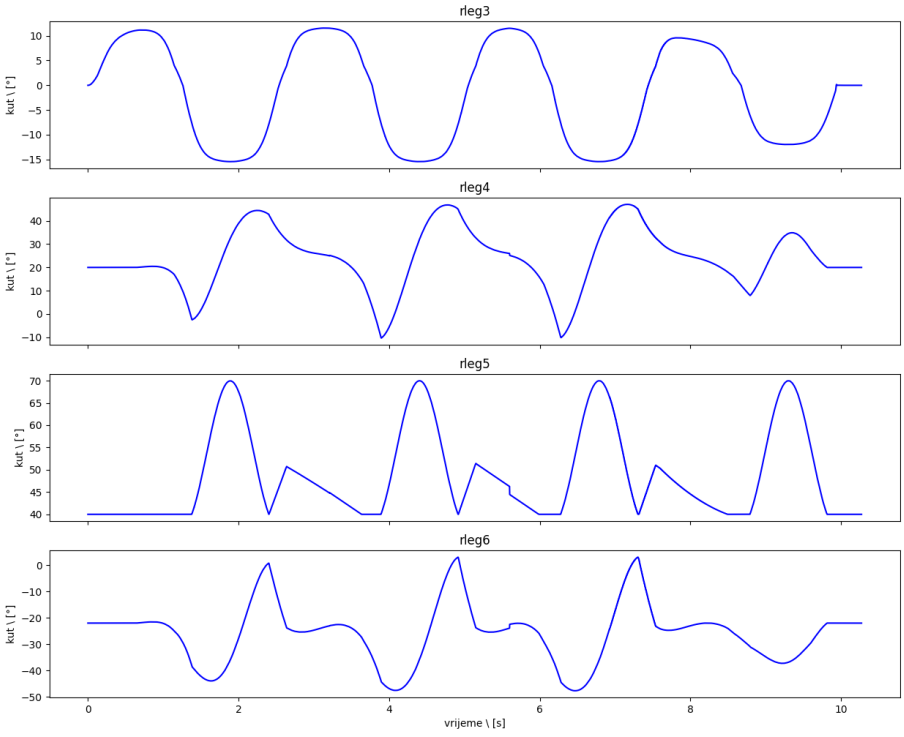


Slika 4.1: Fujitsu HOAP2

4.2. Hod robota

Sve kretnje robota osnivaju se na slijedu točno određenih stupnjeva rotacija zglobova u svakom trenutku simulacije. U našem slučaju od svih dostupnih zglobova (Dodatak 7.1), korišteni su samo: dva stupnja slobode oba ramena, jedan stupanj slobode oba kuka, oba koljena i oba gležnja.

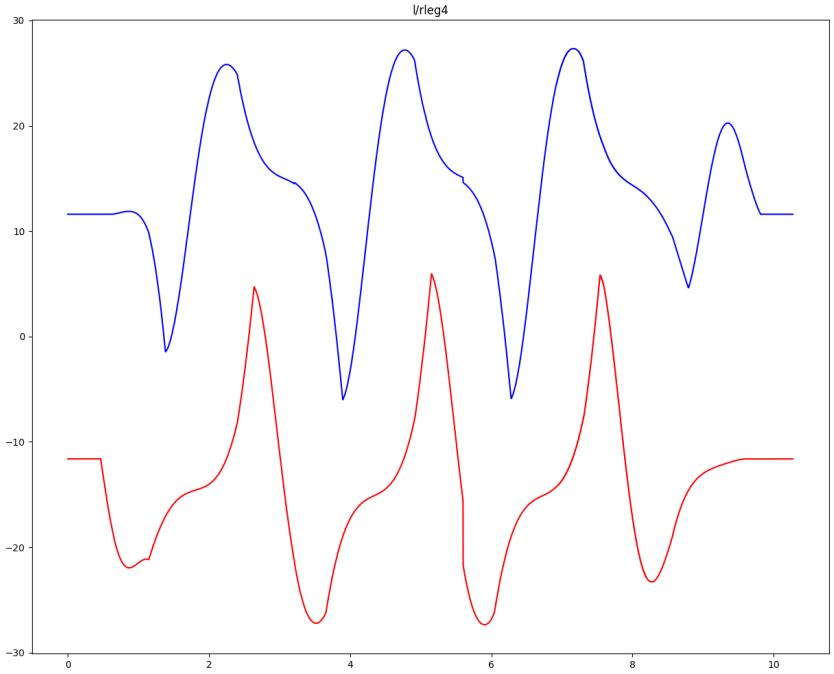
Kretanje je napravljeno periodički. Ako napravimo graf ovisnosti kutu rotacije zgloba i vremenu možemo primijetiti da se vrijednosti rotacije kuka ponavljaju nakon iskoraka lijevom i iskoraka jednom nogom. Na slici 4.2 možemo vidjeti ovisnost kuta zglobova noge kroz vrijeme simulacije. Značenje negativnog kuta na grafu govori da je zglob otklonjen u suprotni smjer od početne pozicije. Isti kut sa suprotnim predznakom daje otklonjenost u suprotnoj orijentaciji.



Slika 4.2: Graf ovisnosti kuta zgloba kroz vrijeme simulacije.

Prvi graf predstavlja jedan kut slobode kuka te vidimo funkciju sličnu sinusoidalnoj funkciji. Na početku i kraju vidimo razliku kako je hod sporiji zbog kretanja i stajanja. Isti takav trend možemo vidjeti na ostalim grafovima. Drugi graf prikazuje kretnje

koljena, a treći i četvrti predstavljaju kretnje gležnja kroz vrijeme. Svi grafovi su jednaki za lijevu nogu samo je faza pomaknuta za duljinu jednoga koraka. To možemo vidjeti vizualno na slici 4.3. Najbolje se trajanje koraka može primijetiti gledajući oštre špiceve crvenog grafa (lijeve noge) i plavog grafa (desne noge).



Slika 4.3: Graf ovisnosti kuta zgloba kroz vrijeme simulacije lijevog i desnog koljena.

4.2.1. Generiranje i učitavanje datoteke s kretnjama

Zbog mogućnosti grafova koje nas navodi na rješavanja problema učenja funkcijskom regresijom, idući prirodni korak je generirati datoteke s podacima o kutovima zglobova kroz vrijeme. Najzgodniji format u te svrhu je CSV datoteka (engl. *comma separated values*). Kreiranu kretnju sada kopiramo u CSV datoteku koja će se lako moći koristiti i u treniranju modela. Prvih 6 stupaca datoteke izgledaju na sljedeći način. Stupci tablice označeni su simboličkim nazivljem zglobova (Dodatak 7.1).

rleg_joint1	**rleg_joint2**	**rleg_joint3**	**rleg_joint4**	**rleg_joint5**	**rleg_joint6**
2	-3	993	-4598	8357	-4598
2	-3	1000	-4598	8357	-4598

Sve varijabilne vrijednosti u tablici pomnožene su s vrijednosti 209 što je vrijednost širine pulsnog signala servo motora Fujitsu HOAP2 robota.

Kontroler robota sadrži učitavanje vrijednosti CSV datoteke te postavljanje vrijednosti kuta zglobova robota svakih 50 milisekundi simulacije.

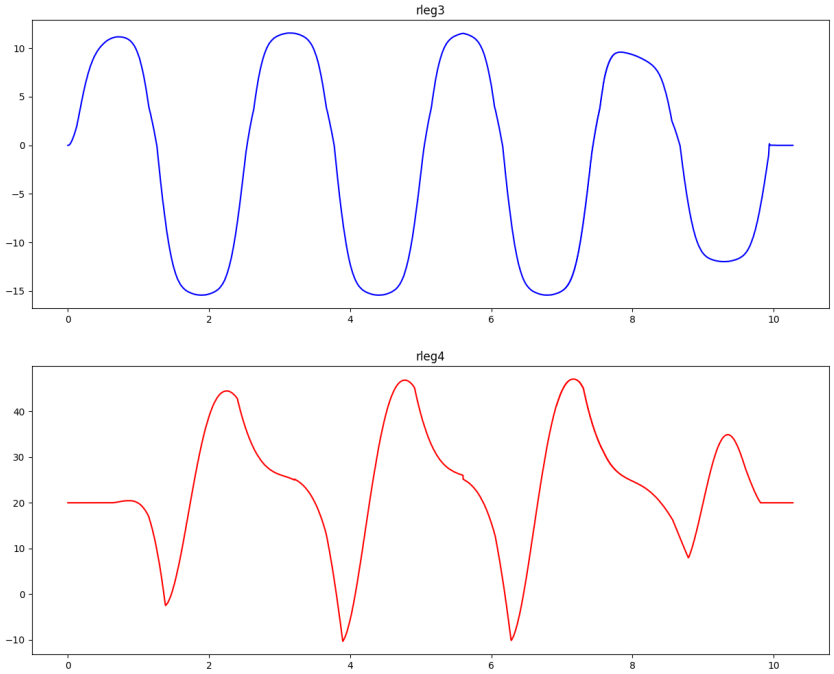
```
// ...
// učitaj podatke idućih pozicija iz CSV datoteke
sscanf(motion,
    "%*d, %d, %d, %d, %d ,%d, %d, %d, %d, %d,"
    " %d, %d, %d, %d, %d, %d, %d, %d, %d, %d,"
    " %d, %d",
    &posFromCsv[(int)Joints::rleg_joint_1],
    // ...
    &posFromCsv[(int)Joints::body_joint_1]);

// pretvori kutove u radijane
for (int i = 0; i < jointsNum; i++)
    nextPosition[i] = posFromCsv[i] * (M_PI / 180.0) / pulse[i];
// postavi kutove zglobova
joint[(int)Joints::body_joint_1]->setPosition(...);
// ...
```

Svi servo motori sadrže svoju enumeraciju kako bi im se moglo lakše pristupiti te se pozicija kuta motora u radijanima postavlja naredbom `join->setPosition(angle)`.

4.3. Neuronske mreže za funkcijsku regresiju

Analizom grafova funkcija, zaključak je da je potrebno napraviti dvije neuronske mreže koje će se trenirati za svaki pojedinačni zglob. Jedna neuronska mreža će trenirati "špicaste" sinusoidne funkcije (iscrtane crveno na slici 4.4), dok će druga trenirati "glatke" sinusoidne funkcije s brjegovima i dolovima (iscrtane plavo na slici 4.4). Takav pristup dao je najbolje ishode učenja i predviđanja vrijednosti.



Slika 4.4: Dva tipa funkcija za koje su namijenjene neuronske mreže.

Prilikom treniranja za vrijednost greške koristi se RMSE (engl. *root mean squared error*) vrijednosti koja se računa računajući standardnu devijaciju pogreške svih neurona u slojevima. Pogreška neurona u sloju računa se tijekom provođenja algoritma širenja unazad (Poglavlje 3.3.1).

4.3.1. Neuronska mreža za "glatke" sinusoidne funkcije

Arhitektura neuronske mreže za "glatke" sinusoidne funkcije je $150 \times 50 \times 25 \times 1$ što znači da sadrži ulazni sloj veličine 150, dva skrivena sloja veličina 50 i 25 te izlazni sloj veličine 1. Prijenosne funkcije u svim neuronima su sigmoidalne funkcije. Model s mješavinom tangens hiperbolne prijenosne funkcije u svim slojevima osim u izlaznom je u slučaju ovoga rada dao je veoma slične.

4.3.2. Neuronska mreža za "špicaste" sinusoidne funkcije

Arhitektura neuronske mreže za "špicaste" sinusoidne funkcije je $150 \times 50 \times 25 \times 25 \times 1$ što znači da sadrži ulazni sloj veličine 150, tri skrivena sloja veličina 50, 25 i 25 te izlazni sloj veličine 1. Prijenosne funkcije u svim neuronima su sigmoidalne funkcije. Model s mješavinom tangens hiperbolne prijenosne funkcije u svim slojevima osim u izlaznom je u slučaju ovoga rada dao je puno lošije rezultate. Isto tako bilo koja promjena broja neurona u skrivenih slojeva bi veličinu RMSE greške povećala s vrijednosti 10^{-4} na 10^{-2}.

4.3.3. Proces treniranja

Obe neuronske mreže trenirane su 500 epoha na svojim odgovarajućim skupovima podataka za pojedinačni zglob. Svakih 50 epoha iskorištena je povratna funkcija za spremanje modela u datoteku. Sljedeći programski isječak korišten je u treniranju oba modela.

```
# normalizacija vrijednosti
x = asarray([i for i in range(0,len(arr))])
y = asarray(arr)

x = x.reshape((len(x), 1))
y = y.reshape((len(y), 1))

scale_x = MinMaxScaler()
x = scale_x.fit_transform(x)
scale_y = MinMaxScaler()
y = scale_y.fit_transform(y)
```

```
12
13 # treniranje modela
14 filepath = "models/" + joint + "/saved-model"
15 checkpoint = ModelCheckpoint(filepath, monitor='loss, mode='min',
16     verbose=1, save_best_only=False, save_weights_only=False)
17 history = model.fit(x, y, epochs=500,
18     batch_size=10, verbose=2, callbacks=[checkpoint])
```

Jednom istrenirani model možemo učitati kada nam je potreban te pozvati funkciju model.predict() kako bi dobili predviđene vrijednosti funkcijske regresije. Taj postupak radimo kada generiramo CSV datoteku koja se učitava prilikom izvršavanja simulacije.

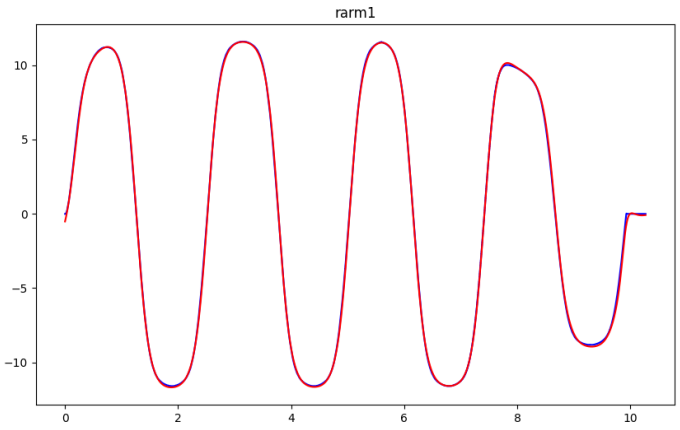
```
1  # normalizacija vrijednosti
2  for joint in joints:
3      path = f'models/{joint}/saved-model'
4      model_load = load_model(path)
5      # ...
6      ywhat = model_load.predict(x)
7      yhat_plot = scale_y.inverse_transform(ywhat)
8      data[joint] = [int(i[0]) for i in yhat_plot.tolist()]
9
10 # spremanje dobivenih vrijednosti u .csv datoteku
11 f = open(f"../data/hoap2/{TITLE}_best.csv", "w")
12     for i in range(0, len(body_angle)):
13         row = []
14         for joint in csv_header:
15             row.append(str(data[joint][i]))
16         f.write(','.join(row))
17         f.write('\n')
18     f.close()
```

Osim spremanja modela nakon određenih broja epoha, moguće je i spremiti najbolji model čija je vrijednost greške najmanja. Tada je samo bitno prilagoditi funkciju model.fit() da sprema model sa najmanjom RMSE vrijednosti.

```
# treniranje modela
filepath = "models/" + joint + "/saved-model"
checkpoint= ModelCheckpoint(filepath, monitor='loss', mode='min',
    verbose=1, save_best_only=True, save_weights_only=False)
earlyStopping = EarlyStopping(monitor='loss',
    patience=100, mode='min')
model.compile(loss='mse', optimizer='adam')
history = model.fit(x, y, epochs=500, batch_size=10,
    verbose=2, callbacks=[checkpoint, earlyStopping])
```

5. Rezultati

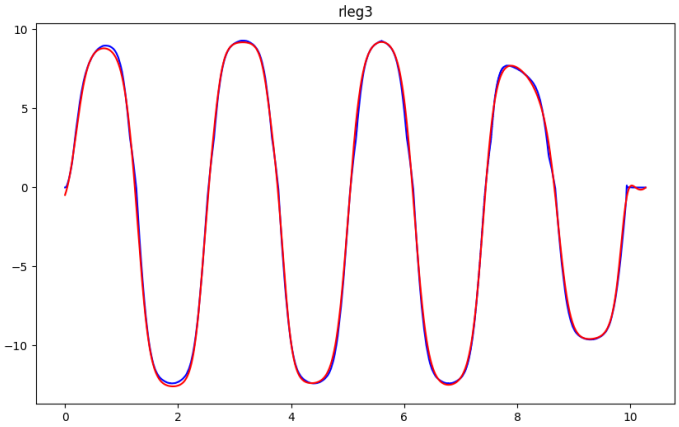
Najbolji rezultat dao nam je hod Fujitsu HOAP2 modela koji napravi pet koraka prije nego što izgubi ravnotežu i padne. Razlog tome je dio funkcije koji obavlja zaustavljanje modela nakon hoda. Kako taj dio prilikom učenja najčešće najviše odstupa od ostalih dijelova funkcije, to se drastično može primijetiti na simulaciji hoda. Modeli neuronskih mreža su vraćale vrijednost greške u rasponu $[10^{-4} - 10^{-5}]$. Iako je to vrlo mala greška ona u kritičnoj fazi hoda kada je model van ravnoteže može rezultirati padom modela na pod. Slike 5.1 - 5.3 prikazuju graf funkcije ovisnosti kuta o vremenu simulacije obojan plavom bojom, a graf dobiven funkcijskom regresijom crvenom bojom.



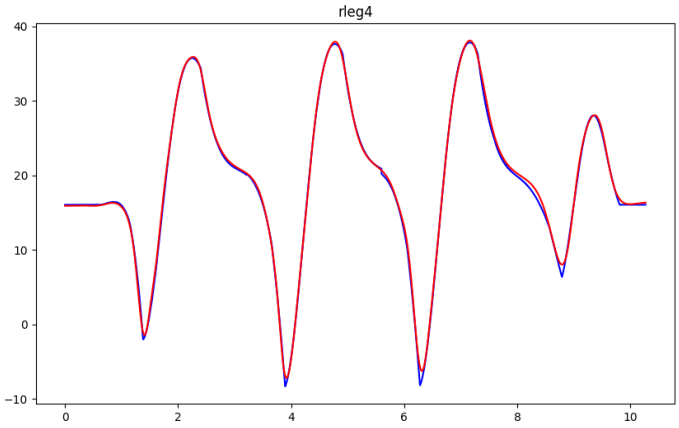
Slika 5.1: Model "glatke" sinusoidne funkcije - desno rame.

Sav izvorni kod, resursi i podaci javno su dostupni na GitLab repozitoriju[1]

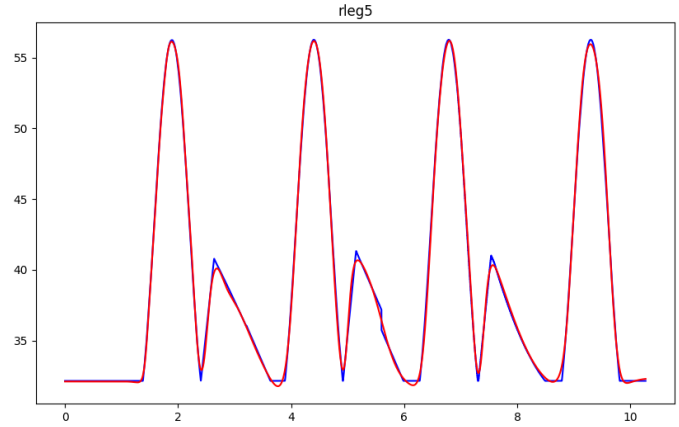
[1] https://gitlab.com/dinogrgic1/fer-bscthesis-biped-ai



Slika 5.2: Model "glatke" sinusoidne funkcije - desni kuk.



Slika 5.3: Model "špicaste" sinusoidne funkcije - desno koljeno.



Slika 5.4: Model "špicaste" sinusoidne funkcije - desni gležanj.

6. Zaključak

Simulacija bipedalnog modela je vrlo složen proces koji sadrži skup raznih interdisciplinarnih znanja od fiziologije čovjeka do robotike i mehanike. Pristup rješavanja učenja hoda bipedalnog modela funkcijskom regresijom daje nam dobre rezultate za ravni teren i kratak hod.

Bilo koja promjena terena utjecat će kobno na naš model. Zbog toga postoji mogućnost poboljšanja modela. Funkcijska regresija s običnim neuronskim mrežama funkcionira dobro na malim domenama. Bolje rezultate bi dao model LSTM koji se temeljni na konvolucijskim neuralnim mrežama. Njegova aproksimacija funkcije bila bi točnija za veće domene te bi bila brža.

Jedno moguće poboljšanje bez uvođenja naprednijih neuronskih mreža je smanjivanje skupa podataka na kraće cikluse tijekom hoda. Ovaj način omogućio bi nam ponavljanje ciklusa tijekom cjelokupnog hoda, a naša obična neuronska mreža bi dala bolje rezultate na još manje domene.

Isto tako moguće je uz pomoć Webots gradivnog objekta *Supervisor* imati konstantnu kontrolu nad simulacijom te obavljati podržano učenje dinamički tijekom izvođenja simulacije što bi omogućilo "spašavanje" robota od sada neizbježnog pada.

7. Dodatak

7.1. Zglobovi Fujitsu HOAP2 robota

- glava (`head_joint[1,2`) - 2 stupnja slobode
- tijelo (`body_joint_1`) - 1 stupanj slobode
- ramena
  - lijevo rame(`larm_joint[1,2,3])` - 3 stupnja slobode
  - desno rame(`rarm_joint[1,2,3])` - 3 stupnja slobode
- laktovi
  - lijevi lakat(`larm_joint_3)` - 1 stupanj slobode
  - desni lakat(`rarm_joint_3)` - 1 stupanj slobode
- zapešća
  - lijevo zapešće(`larm_joint_4)` - 1 stupanj slobode
  - desno zapešće(`rarm_joint_4)` - 1 stupanj slobode
- kuk
  - lijevi kuk(`lleg_joint_[1,2,3])` - 3 stupanja slobode
  - desni kuk(`rleg_joint_[1,2,3])` - 3 stupanja slobode
- koljeno
  - lijevo koljeno(`lleg_joint_4)` - 1 stupanj slobode
  - desno koljeno(`rleg_joint_4)` - 1 stupanj slobode
- gležanj
  - lijevi gležanj(`lleg_joint_[5,6])` - 2 stupnja slobode
  - desni gležanj(`rleg_joint_[5,6])` - 2 stupnja slobode

LITERATURA

François Chollet et al. Keras. https://keras.io, 2015.

Warren Mcculloch i Walter Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:127–147, 1943.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, i E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Webots. http://www.cyberbotics.com, 2020. URL http://www.cyberbotics.com. Open-source Mobile Robot Simulation Software.

Lukasz Wiklendt, Stephan K. Chalup, i María M. Seron. Quadratic leaky integrate-and-fire neural network tuned with an evolution-strategy for a simulated 3d biped walking controller. U Fatos Xhafa, Francisco Herrera, Ajith Abraham, Mario Köppen, i José Manuel Benítez, urednici, *8th International Conference on Hybrid Intelligent Systems (HIS 2008), September 10-12, 2008, Barcelona, Spain*, stranice 144–149. IEEE Computer Society, 2008. doi: 10.1109/HIS.2008.146. URL https://doi.org/10.1109/HIS.2008.146.

Kuangen Zhang, Zhimin Hou, Clarence Silva, Haoyong Yu, i Chenglong Fu. Teach biped robots to walk via gait principles and reinforcement learning with adversarial critics. 10 2019.

Marko Čupić. Umjetne neuronske mreže, 2018. URL http://java.zemris.fer.hr/nastava/ui/ann/ann-20180604.pdf.

Evolucija kretanja dvonožnog modela korištenjem neuronskih mreža

Sažetak

Ovaj završni rad implementira učenje hodanja bipedalnog modela uz pomoć neuronskih mreža. Modeli robota i simulacija hoda odrađena je uz pomoć programskog paketa za simulaciju robota Webots. Učenje hoda izvršeno je aproksimacijom funkcija koje opisuju kut zglobova modela robota kroz vrijeme provodeći funkcijsku regresiju.

Ključne riječi: simulacija kretanja bipedalnog modela, neuronske mreže, fukcijska regresija, Webots, Keras

Movement Evolution of Biped Model using Neural Networks

Abstract

This final work implements learning of walk movement for the bipedal model using neural networks. Robot model and simulation is implemented with Webots programming package for robot simulation. Learning of walk movement is done by approximating functions that describe the angle of model joints throughout time with functional regression.

Keywords: bipedal model movement simulation, neural networks, functional regression, Webots, Keras