SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 5492

Simulator leta letjelice Drone

Bruno Bijelić

Zagreb, lipanj 2018.

SADRŽAJ

1. Uvod

Bespilotne letjelice (dronovi) su letjelice bez posade koje su jedna od posljedica minijaturizacije tehnologije. Dronovi su se u početku najčešće koristili u vojne svrhe, međutim sve se više koriste u komercijalne i znanstvene svrhe. Dronovi mogu imati veću ili manju razinu autonomnosti, ali većinom dronova, barem do neke mjere, upravljaju ljudi. Unatoč tome, postoje dronovi koji su u potpunosti autonomni, to jest nije potreban čovjek da bi oni sami letjeli. Takvim dronovima moraju upravljati računala s vrlo sofisticiranim algoritmima koje treba jako temeljito testirati. Testiranje tih algoritama bi bilo vrlo nezgodno kada bi se radilo samo na pravim dronovima jer bi to rezultiralo velikim brojem slupanih dronova. Srećom, u inženjerskoj praksi se vrlo često prakticira simuliranje rezultata pomoću grafičkih simulacija.

Središte ovog rada i jest izgradnja "mozga" autonomnog drona pomoću neuronskih mreža, a rezultati neuronskih mreža će biti testirani grafičkim simulacijama. Bit će opisano kako pomoću pogona za izradu video igrica Unity napraviti potrebnu simulaciju drona koja služi za prikupljanje podataka za neuronsku mrežu i procjenu rezultata neuronske mreže. Također će biti opisano kako pomoću biblioteke scikit-learn za programski jezik Python izgraditi neuronsku mrežu koja će upravljati dronom.

2. Unity

Unity je višeplatformski pogon za izradu igara (engl. game engine). Pogon za izradu igara je softverski okvir koji omogućava i olakšava izradu video igrica. Višeplatformski znači da podržava izradu video igrica za više softverskih platformi kao što su: Mac, Linux, Windows, Android, iOs. Prva verzija je izdana 2005. godine, a najnovija verzija je 2018.1.2. koja je izdana 25. svibnja 2018. godine [3].

Unity je vrlo svestran alat i nudi izrade na gotovo svim platformama. U Unity-u je moguće izraditi 2D igrice, 3D igrice, igrice za više igrača, igrice koje su upravljane palicom (engl. joystick), tipkovnicom, ekranom osjetljivim na dodir. Također je moguće raditi igrice koje su u sklopu web stranica, igrice koje se igraju pomoću VR naočala, Kinect naprava i još mnogo toga.

Unity također nudi svoje skladište s dodacima (engl. Asset Store) gdje se mogu prodavati i kupovati dijelovi igrica koje su drugi ljudi napravili. Na skladištu s dodacima ima i besplatnih stvari koje se mogu preuzeti i koristiti u svojim igricama.

Unity je besplatan i jednostavan za instalirati. Nudi veliku podršku početnicima s ogromnim brojem video lekcijama pomoću kojih, u nekoliko dana, osoba koja nikad nije napravila igricu može napraviti svoju igricu, a nakon toga dalje može razvijati svoje znanje o izradi igrica pomoću kompliciranijih lekcija.

3. Neuronske mreže i biblioteka scikit-learn

Umjetne neuronske mreže su računalni sustavi inspirirani ljudskim i životinjskim mozgovima. Neuronske mreže su sustavi koji pokušavaju učiti, međutim oni najčešće nemaju znanje o tome što točno uče. Na primjer, neuronska mreža može pokušati naučiti nalazi li se na rendgenu nekog organa tumor, ali neuronska mreža neće znati što je tumor, što je rendgen, ona čak neće ni znati vizualna obilježja tumora koja doktor zna, nego će neuronska mreža imati velik broj rendgenskih snimki, a svaka od tih snimki je obilježena kao snimka s tumorom, ili snimka bez tumora, te će mreža sama razviti sustav reprezentacije tumora i potencijalno, uz dovoljno veliki broj podataka, moći će vrlo dobro procjenjivati nalazi li se tumor na nekoj rendgenskoj snimci.

Ukratko, umjetne neuronske mreže se sastoje od umjetnih neurona koji su međusobno povezani konekcijama. Neuronske mreže imaju 3 sloja: ulazni sloj, skriveni sloj i izlazni sloj [1]. Ideja je da se na ulazu mreže pojave podaci te oni poput impulsa u pravom živčanom sustavu prolaze kroz konekcije i neurone koji obrađuju te impulse i na kraju mreže se pojave izlazi. Konekcije imaju svoje razine osjetljivosti koje su ništa drugo nego brojevi te svaki put kada impuls prelazi iz neurona u neuron preko konekcije on se množi s razinom osjetljivosti te konekcije. Produkti se zatim sumiraju te im se onda dodaje pomak, a nakon toga dobivena vrijednost prolazi kroz prijenosnu funkciju. U biti, vrlo grubo rečeno, na ulaz neuronske mreže možemo gledati kao na vektor (ulazni sloj) koji se u mreži množi s više matrica (skriveni sloj) i to na kraju rezultira izlaznim vektorom (izlazni sloj). To naravno nije u potpunosti ispravna vizualizacija neuronske mreže zbog postojanja prijenosne funkcije i pomaka, ali daje nekakvu intuitivnu ideju o neuronskim mrežama. Kako bi neuronsku mrežu primijenili na upravljanje dronom, potrebno je postaviti matrice i pomake u skrivenom sloju tako da ulazne podatke o okolini drona pretvore u izlazne podatke pomoću kojih će dron odlučiti kako letjeti u tom trenutku. Matrice i pomaci se postavljaju treniranjem neuronske mreže [1].

Neuronske mreže se mogu trenirati na više načina. U ovom radu mreža će se istrenirati pomoću podataka koji su skupljeni tako da su stvarni ljudi upravljali simulacijom. Uz dovoljno velik broj podataka, neuronska mreža bi trebala moći oponašati stvarne ljude. Nakon prikupljanja podataka konstruiranje neuronske mreže se radi pomoću kompleksnih algoritama, a u ovom radu će se koristiti gotova biblioteka scikit-learn.

Scikit-learn je biblioteka javno dostupnog koda (engl. open source) napisana za Python. Biblioteka sadrži velik broj korisnih alata za područje strojnog učenja. Kako su umjetne neuronske mreže jedno od područja strojnog učenja, u biblioteci scikit-learn ima i korisnih alata za neuronske mreže koji su korišteni u ovom radu.

4. Izrada simulacije

Do autonomnog drona doći ćemo u tri faze. Prva faza je izrada simulacije u obliku igrice pomoću koje ćemo prikupiti podatke o upravljanju dronom stvarnih ljudi. Druga faza je analiza podataka i stvaranje neuronske mreže. Treća i zadnja faza je upravljanje dronom pomoću neuronske mreže u već napravljenoj simulaciji.

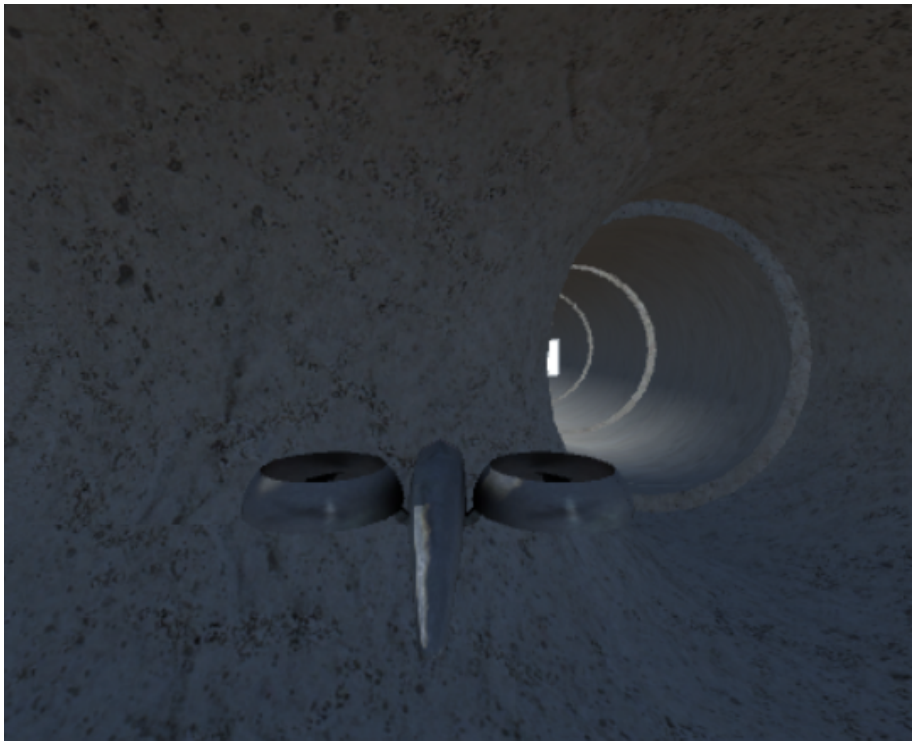
U ovom poglavlju se opisuje prva faza. Ideja je napraviti simulaciju leta drona, ali kao igricu koju će onda igrati stvarni ljudi, a podaci o njihovom upravljanju drona treba spremiti u datoteku.

4.1. Izgled i pravila igrice

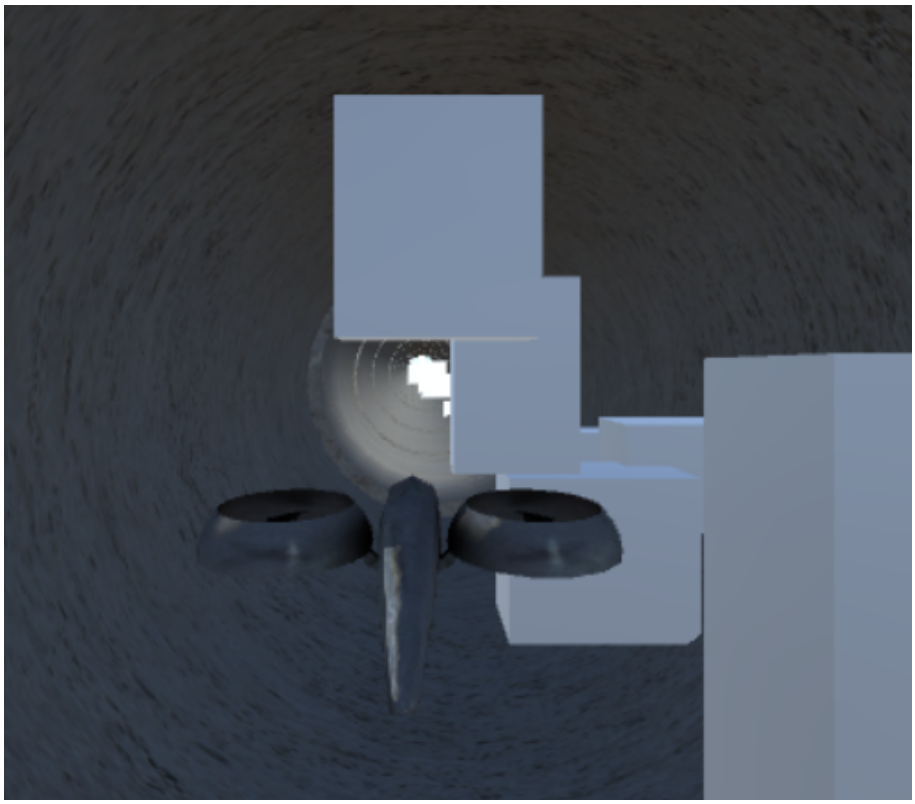
Od sada ćemo na ovaj problem autonomnog drona gledati kao da imamo igricu u kojoj treba voziti dron i želimo dobiti neuronsku mrežu koja je što bolja u toj igrici. Prvo treba napraviti igricu za ljude. Igrica će imati jedan dron kojem igrač upravlja tipkovnicom, a gleda na njega pogledom iz trećeg lica. Korisnik treba ići prema ravno što dalje bez da se sudari. Dron se nalazi u cijevi koja ima svoje zavoje koji se nasumično generiraju te igrač treba uspješno proći te zavoje (Slika 4.1). Također, uz zavoje se nasumično i generiraju prepreke u tunelu koje isto treba izbjegavati (Slika 4.2). Cilj igrača je da što veći dio tunela prođe bez da se sudari. Jednom kada se dron sudari s tunelom ili s preprekom igra je gotova i kreće se ispočetka. Ideja igrice nije previše kompliciranа, a u sljedećim potpoglavljima ćemo ukratko proći kroz temeljne dijelove igrice.

4.2. Izrada drona

Temeljni objekt koji je potreban je model drona. Potrebno je dizajnirati dron, ali još puno važnije što dronu treba su njegova fizikalna obilježja i dodavanje mogućnosti upravljanja dronom preko tipkovnice. Dizajn drona sam preuzeo sa skladišta dodacima koje nudi Unity i koje je bilo spomenuto u poglavlju o Unity-u.



Slika 4.1: Dron u zavoju



Slika 4.2: Dron među preprekama

Da bi dron imao fizikalna svojstva treba mu dodati dvije komponente u Unity-u, a to su sudarač i komponenta krutog tijela.

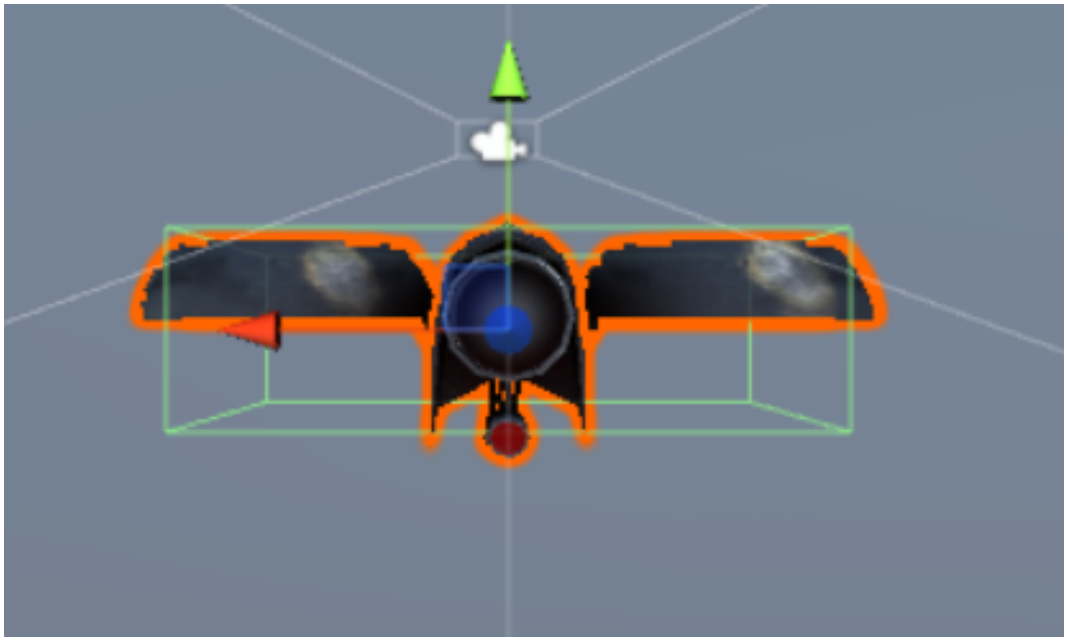
Sudarač je potreban da bi mogli uočiti sudar između drona i nekog drugog objekta. Naime, Unity sve kolizije uočava pomoću sudarača, njemu ništa ne znači dizajn nekog tijela, već je sudarač taj koji predstavlja tijelo u fizikalnom prostoru, to jest kada Unity promatra odnose među objektima u fizikalnom prostoru, on gleda isključivo sudarače tih tijela [3]. Pošto je dron mali u odnosu na okolinu, dovoljno je dronu dodati sudarač u obliku običnog kvadra koji otprilike odgovara njegovim dimenzijama. Inače je moguće napraviti puno kompleksnije sudarače od kvadra, ali u ovom slučaju to nije potrebno.

Komponenta krutog tijela je komponenta bez koje se u Unity-u objekt ne može kretati. Komponenta krutog tijela sadrži važna fizikalna svojstva pomoću kojih Unity računa kako se neko tijelo kreće. Neka od svojstava te komponente su masa tijela, kako na tijelo utječu gravitacija, trenje, otpor zraka i još mnogo toga [3]. Nakon što smo objedinili grafički model drona s ovim dvjema komponentama u jedan objekt imamo dron koji se može kretati te uočavati sudare.

Treba još dodati mogućnost upravljanja dronom preko tipkovnice. To se radi preko skripte. U Unity-u se skripte mogu pisati u jezicima C# i JavaScript [3]. U ovom radu skripte su pisane u C#-u. Skripta za kretanje drona će biti vrlo jednostavna. Naime, u Unity-u skripte možemo prikvačiti na neki objekt i na taj način se zna da se ta skripta odnosi na taj objekt. Znači, napravimo novu skriptu prikvačimo je na objekt drona i nakon toga samo treba mijenjati metode *Start*() i *Update*(). Metoda *Start*() se poziva pri pokretanju igrice i ona služi za inicijalizaciju varijabli. Metoda *Update*() se poziva pri svakom taktu promjene slike (engl. frame). U biti, nakon što smo prikvačili skriptu na drona samo trebamo u metodi *Update*() na svaki pritisak gumba dodati silu na dron. Na primjer, ako korisnik pritisne gumb za ubrzanje drona u smjeru prema gore iz perspektive korisnika, bit će dodana sila prema gore. Nakon što smo napisali skriptu imamo dron kojim možemo upravljati. Na slici 4.3 se može vidjeti model drona, a zeleni kvadar oko drona je njegov sudarač.

4.3. Izrada okoline

Sada kada imamo dron, želimo napraviti tunel i prepreke koje će igrač izbjegavati upravljajući dronom. Treba dodati tunel čiji se zavoji nasumično generiraju te prepreke koje se također nasumično stvaraju u tunelu. Tunel ćemo napraviti tako da ćemo imati objekt cijevi koje ćemo onda nasumično slagati. Naime, preuzeo sam sa skladišta s

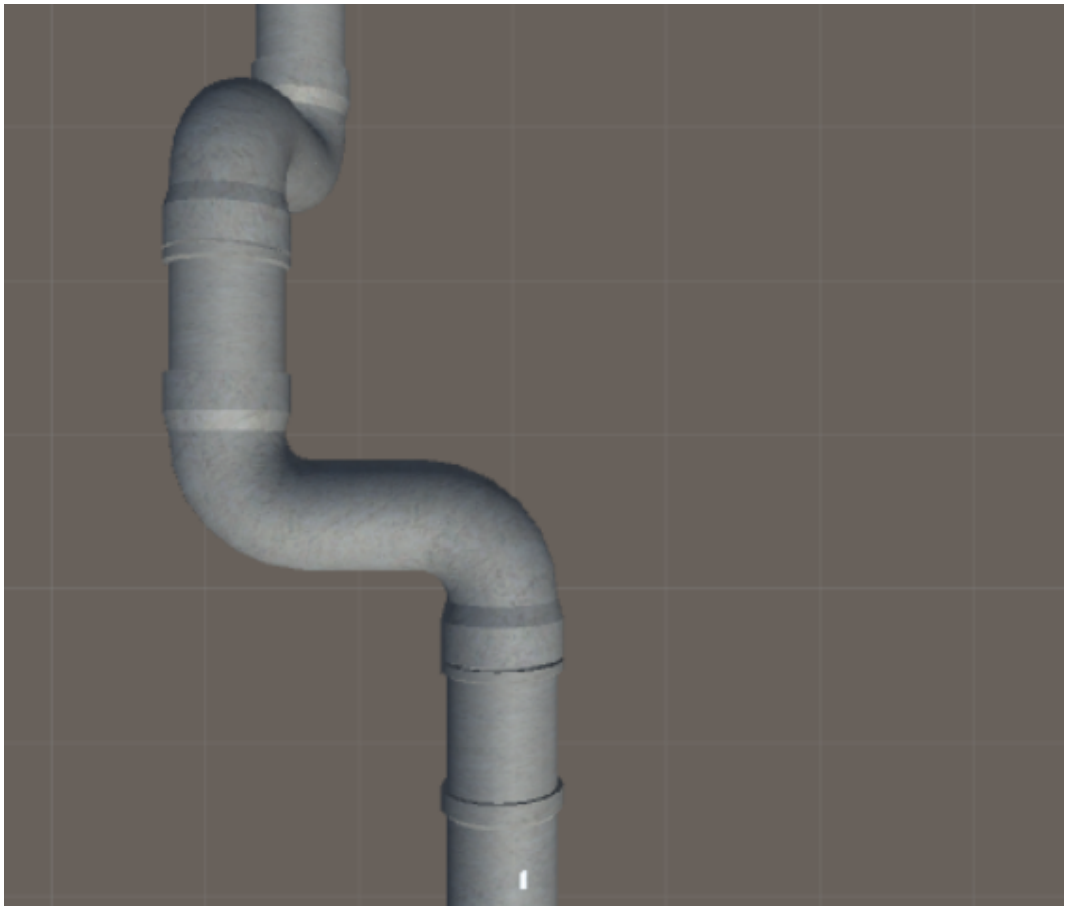


Slika 4.3: Model drona

dodacima dvije vrste cijevi, jedna je ravna cijev, a druga je zakrivljena cijev. Ideja je da slažemo cijevi tako da se međusobno nastavljaju jedna na drugu. Tako kada na prošlu cijev spojimo ravnu cijev dobili smo dio tunela koji nema zavoja, a ako spojimo zakrivljenu cijev, dobili smo zavoj. Rotacijom zaobljenih cijevi možemo dobiti više vrsti zavoja (zavoj ulijevo, udesno, prema gore i prema dolje). Treba napomenuti da cijevi također trebaju imati sudarače tako da mogu uočiti koliziju s dronom. Cijevi su praktički šuplji valjci stoga njihov sudarač ne može biti obični kvadar, nego mora biti kompleksniji. Sudarač jedne cijevi je sustav više poligona. Na sreću, cijevi koje sam preuzeo su već imali pripremljene kompleksne sudarače.

Sada kada imamo cijevi, treba ih nasumično generirati, a to ćemo napraviti pomoću skripte. Napravimo novu skriptu u kojoj u metodi *Update*() provjeravamo je li dron dovoljno blizu kraja već generiranih cijevi. Ako je, onda je potrebno generirati novu cijev i u skripti se nasumično odlučuje koja je sljedeća cijev (ravna cijev, zaobljena cijev zarotirana udesno, zarotirana ulijevo...). To je sve što je potrebno da bi imali tunel s nasumično generiranim zavojima (Slika 4.4).

Dodavanje prepreka je znatno lakše nego generiranje tunela. Naime, prepreke će nam biti obični kvadri, a objekt kvadra je jedan od standardnih objekata koje nam Unity nudi da napravimo jednim klikom. Znači, imamo objekt kvadra kojeg ćemo instancirati također preko skripte na nasumičnim mjestima. Kada smo gotovi s time skoro pa imamo gotovu igricu.



Slika 4.4: Generirani tunel

4.4. Izrada grafičkog korisničkog sučelja

Da bi imali gotovu igricu, treba nam još samo grafičko korisničko sučelje (engl. Graphical User Interface). GUI nam treba tako da igrač može pokrenuti igricu, pauzirati igricu, dobiti poruku da se sudario i slično. U biti, samo treba napraviti ulazni meni, meni kada korisnik pauzira igricu te meni kada se korisnik sudari.

Ulazni meni sadrži tri gumba: gumb za pokretanje igrice bez snimanja, gumb za pokretanje igrice sa snimanjem i izlazak iz igrice. Naime, mi želimo snimati igrače kako voze dron, međutim mi želimo da njima to dobro ide. Kako igrica nije baš prejednostavna te čak i čovjek treba malo trenirati da bi počeo dobro voziti, igrica se može pokretati bez snimanja tako da igrač prvo trenira i postane dobar u igrici, a nakon toga će pokretati igricu sa snimanjem i na taj način prikupljati korisne podatke.

Meni kada korisnik pauzira igricu te meni kada se korisnik sudari su identični i vrlo su jednostavni. Sadrže samo jedan gumb, a to je gumb za povratak na ulazni meni. Jedina razlika između ova dva menija je to što se u meni za pauziranje ulazi

i izlazi tipkom *Escape*, a u meni za sudar se ulazi kada se igrač sudari s dronom i u tom meniju također piše poruka da se igrač sudario. Na slici 4.5 se može vidjeti ulazni meni s još jednim gumbom *Simulate* o kojem ćemo više saznati kasnije.

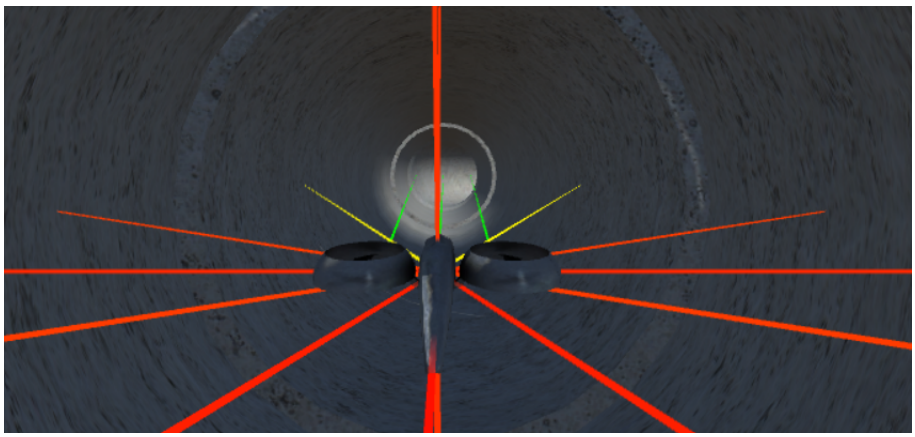
Slika 4.5: Ulazni meni

4.5. Izrada senzora

Sad već imamo gotovu igricu. Međutim, mi želimo da ta igrica služi za prikupljanje podataka. Za to nam trebaju još dvije stvari. Trebaju nam senzori pomoću kojih će autonomni dron gledati svoju okolinu i trebamo u igricu dodati mogućnost ispisa podataka o vožnji u izlaznu datoteku. U ovom potpoglavlju ćemo se baviti senzorima. Kada čovjek igra igricu on svoju okolinu promatra gledajući ono što kamera snima. Kako su računala po mnogočemu različita od ljudi, u praksi autonomna vozila ne promatraju svijet lijepim kamerama visoke rezolucije jer bi to računalu bilo preapstraktno za analizu. Zato u praksi računala okolinu promatraju pomoću senzora i kamera niske rezolucije (vrlo često ne koriste obične video kamere nego termalne kamere i slično). Kako ova igrica nije toliko kompleksna dovoljni će biti senzori, nećemo u ovom radu simulirati nikakvu kameru niske rezolucije. Znači, želimo na dron dodati senzore koji govore koliko su udaljeni od neke prepreke. Svi senzori na ovom dronu će biti kao neki

laseri koji šalju podatak o udaljenosti izvora lasera i mjesta na kojem se laser sudario s nečim (to će uvijek biti ili tunel, ili prepreka). Sada je pitanje koliko senzora dodati i gdje da oni budu usmjereni. Dobra strana većeg broja senzora je što daju precizniju sliku okoline, ali loša strana velikog broja senzora je to što je onda ulazni sloj neuronske mreže jako velik pa je potreban veći broj podataka da bi se neuronska mreža istrenirala, a i ne samo to, jer može se dogoditi da imamo ogroman broj podataka i želimo izgraditi kompleksnu neuronsku mrežu pa nam zbog toga treba jako puno vremena da se neuronska mreža pravilno istrenira (to mogu biti apsurdno veliki vremenski intervali poput nekoliko godina). Treba razmisliti gdje bi bilo dobro imati senzore. Recimo sigurno je dobro imati senzor koji gleda ravno u smjeru u kojem gleda kamera jer na taj način je moguće uočiti da je odmah ravno ispred drona prepreka. Također je bitno nalazi li se prepreka i u drugim smjerovima od drona (desno, lijevo, iznad, ispod, iza). Međutim, logično je da nisu dovoljni senzori koji su okomiti. Očito su potrebni i neki senzori koji gledaju dijagonalno. U ovom radu su korištena 2 različita sustava senzora. Važno je napomenuti kako se senzori dodaju preko C# dronove skripte.

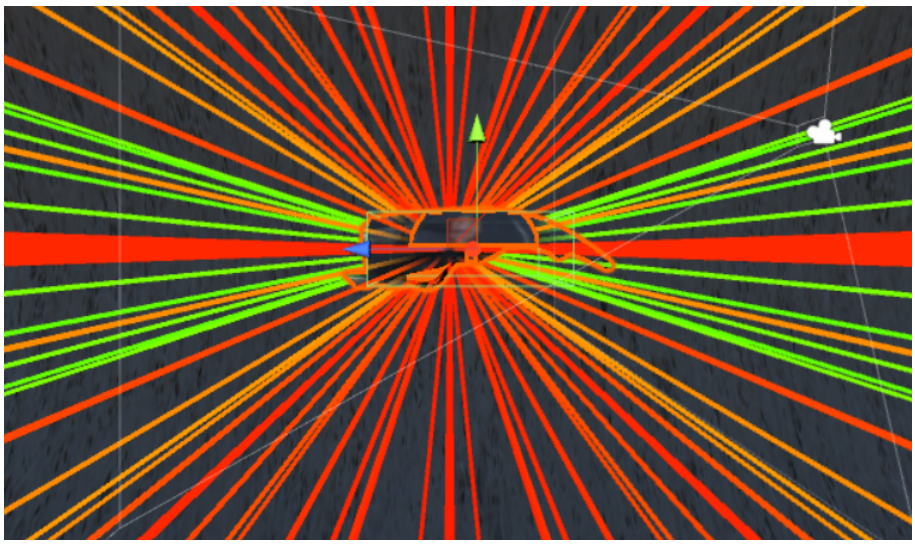
Prvi sustav senzora ima 26 senzora. Nema smisla tekstualno objašnjavati gdje je koji senzor, nego je bolje pogledati na slici 4.6. Prednost ovako malog broja senzora je što će se neuronska mreža brže trenirati, ali mana je što ipak u nekim situacijama ne prikazuje okolinu na dovoljno precizan način. Ovih 26 senzora sam ručno birao.



Slika 4.6: Dron s 26 senzora

Drugi sustav senzora ima 188 senzora. Oni nisu birani ručno, nego su programski generirani. Ideja je da ti senzori budu raspoređeni u svim smjerovima otprilike ravnomjerno. Kako ih nisam ručno birao, to jest nisam uzimao u obzir koji senzori su bitniji za vožnju drona, nego sam ih samo jako puno stvorio u svim smjerovima, postoji opasnost da ovdje postoji dosta skoro beskorisnih senzora, međutim čak i ako se to dogodi, uz dovoljno veliki broj ulaznih podataka, neuronska mreža će sama shvatiti da su ti

senzori nebitni. Na slici 4.7 je dron sa 188 senzora.



Slika 4.7: Dron sa 188 senzora

Još jedna zgodna stvar za dodati je grafički prikaz senzora. Tako igrači mogu vidjeti kako računalo gleda na svijet oko sebe. Senzori su prikazani kao laseri koji mijenjaju boje ovisno o blizini prepreke. Na primjer, ako senzor osjeti da je prepreka jako blizu, bit će crvene boje, a ako je jako daleko bit će zelene boje, a ako je nešto između bit će neka boja između zelene i crvene ovisno o udaljenosti prepreke.

4.6. Ispis podataka o upravljanju dronom u datoteku

Ono što nam je preostalo da završimo prvu fazu je dodati mogućnost snimanja vožnje. Kada igrač igra igricu sa snimanjem treba ispisati što su u nekom trenutku snimili senzori, a koje gumbe je tom trenutku pritisnuo igrač. Na taj način ćemo imati podatke za trening neuronske mreže. Snimanje podataka ćemo napraviti pomoću skripte. Opet ćemo dodavati programski kod u metodu *Update*() dronove skripte. Naime, ako recimo odlučimo da želimo 10 puta u sekundi zapisivati podatke, onda u metodi *Update*() trebamo provjeravati je li prošlo 0.1 sekunda od prošlog zapisivanja. Ako je prošlo dovoljno vremena onda slijedi zapis u binarnu datoteku. Što nam zapravo treba? Trebaju nam podaci senzora i trebaju nam podaci o tome što je korisnik pritisnuo. To je sve vrlo lako prikupiti jer se senzori nalaze u dronovoj skripti i vrlo lako im možemo pristupiti. Prikupimo podatke, zapišemo u binarnu datoteku i igrica je gotova. Ta binarna datoteka se nalazi u istom direktoriju u kojoj se nalazi i igrica. Ako zamolimo nekoga da malo vozi dron i prikuplja podatke za nas, ta osoba nam samo treba poslati

tu datoteku.

Dovršavanjem igrice smo ujedno i dovršili prvu fazu. Sada imamo sustav za prikupljanje korisnih podataka. Nakon toga je potrebno nekoliko sati igrati igricu i već imamo podatke s kojima možemo nešto napraviti.

5. Analiza podataka i stvaranje neuronske mreže

Dovršili smo sustav za prikupljanje podataka o ispravnoj vožnji. Kako će neuronska mreža sada preko podataka naučiti voziti dron? Odgovor na to pitanje je možda najlakše dati preko primjera. Recimo da je desna strana drona jako blizu tunela. U tom slučaju je jako loš potez pritisnuti gumb za ići desno. Međutim, ako mi imamo podatke o kvalitetnoj vožnji, onda u tim podacima gotovo nikad nećemo imati slučaj u kojemu desni senzor pokazuje da je prepreka blizu, a igrač je pritisnuo tipku za ići desno. To je pravilnost koju će naša neuronska mreža primijetiti i zbog toga kada je ulaz u mrežu takav da desni senzor kaže da je prepreka jako blizu, neuronska mreža također gotovo nikada neće pokušati ići desno. Uz dobro postavljene senzore i dovoljno podataka, mreža će uočiti i puno kompleksnije pravilnosti od ove i na taj način će naučiti oponašati ljudske vozače drona.

5.1. Sumiranje i čišćenje podataka

Jedna od stvari koju želimo analizirati je kako će neuronska mreža naučiti voziti koristeći podatke različitih ljudi. Recimo da nam je više ljudi poslalo datoteke o vožnji. Nama će možda biti jednostavnije da imamo to sve u jednoj datoteci. Također, svi se ponekad sudare s dronom. Možda bi bilo dobro da u slučaju da u našim podacima uočimo sudar, izbrišemo podatke od zadnjih sekundu (ideja je da ako se korisnik sudario očito je donio nekakvu lošu odluku u zadnjoj sekundi). Takvo sumiranje i čišćenje podataka je vrlo lako napraviti u jednoj kratkoj Python skripti. Skripta obilazi sve datoteke s podacima u nekoj mapi te prepisuje podatke u jednu novu, veliku datoteku koja objedinjuje sve datoteke. Također, Python skripta neće prepisati podatke koji su snimljeni malo prije, te za vrijeme sudara.

5.2. Treniranje neuronske mreže

Treniranje neuronske mreže ćemo raditi pomoću alata scikit-learn i još par dodatnih funkcija koje trebamo sami napisati. U svakom slučaju moramo napisati funkciju za čitanje podataka iz velike datoteke koju smo generirali u prošlom potpoglavlju. Također treba napisati funkciju za ispis neuronske mreže u trenutku kad je neuronska mreža gotova i spremna za korištenje. Prvu funkciju nije problem napisati, ona je vrlo slična programu iz prošlog potpoglavlja, potrebno je samo obrnuti postupak, to jest umjesto pisanja u binarnu datoteku, potrebno je čitanje iz datoteke. Ispis neuronske mreže isto nije veliki problem ako znamo što trebamo ispisati. U poglavlju o neuronskim mreža smo već ustanovili da je neuronska mreža određena matricama skrivenog sloja te pomacima. Vidjet ćemo kasnije da razredi neuronskih mreža biblioteke scikit-learn omogućuju pregled tih svojstava, tako da ih bez problema možemo predati našoj funkciji koja će ih ispisati u binarnu datoteku.

Ima još jedna funkcija koja bi nam dobro došla. Naime, jednom kada istreniramo neuronsku mrežu, zanima nas je li ta mreža dobra. To možemo provjeriti tako da upalimo simulaciju i pogledamo kako dron vozi. Međutim, malo sporo je to baš svaki put raditi ručno, pogotovo jer ponekad ćemo dobiti mreže koje su toliko loše da ih možemo provjeriti jednostavnije pomoću podataka koje smo već prikupili. Naime, mreži možemo predati ulazne podatke iz kojih će ona predvidjeti izlaze. Potom možemo usporediti predviđene izlaze s izlazima koje smo već prikupili. Stoga bi nam bilo dobro napisati funkciju koja točno to radi i tako detaljno analizira rezultate naše nove neuronske mreže. Funkcija će nam dosta dobro filtrirati vrlo loše neuronske mreže.

Nakon što imamo te tri funkcije, spremni smo koristiti biblioteku scikit-learn. Koristit ćemo Python na način da pišemo kod u Python Shell, umjesto da pišemo gotove skripte, jer na ovaj način ćemo biti puno brži i lakše ćemo raditi promjene. Recimo da se već prije spomenuta funkcija za čitanje podataka zove *readData*(), rad s podacima možemo početi ovako:

```
inputs, outputs = readData()
```

Sada kada smo prikupili podatke o senzorima (ulazima) i o pritisnutim tipkama s obzirom na te senzore (izlazi), trebalo bi te ulaze i izlaze podijeliti u skup podataka namijenjen za treniranje i skup podataka namijenjen za testiranje. To radimo zato što ćemo na podacima namijenjenim za treniranje istrenirati našu mrežu, međutim ne bi bilo dobro na istim podacima i trenirati, i testirati našu mrežu, pa ćemo dio podataka koristiti za treniranje, a dio za testiranje.

```
from sklearn.model_selection import train_test_split
data_splitted = train_test_split(inputs, outputs)
X_train, X_test, y_train, y_test = data_splitted
```

Nakon toga ćemo iskoristiti razred *MLPClassifier* biblioteke scikit-learn [2]. Taj razred predstavlja neuronsku mrežu. Pozovemo konstruktor razreda u kojem možemo definirati veliki broj parametara poput broja neurona u skrivenom sloju.

```
from sklearn.neural_network import MLPClassifier
mlp = MLPClassifier(hidden_layer_sizes=(30, 20))
```

Imamo mrežu i imamo podatke. Jedino što je preostalo je istrenirati mrežu.

```
mlp.fit(X_train, y_train)
```

Nakon nekog vremena mreža je istrenirana, i sada ju možemo testirati pomoću naše funkcije koju smo napisali da nam otprilike analizira kvalitetu nove mreže. Recimo da se naša funkcija zove *evaluatePredictions*().

```
predictions = mlp.predict(X_test)
evaluatePredictions(predictions, y_test)
```

Ako smo zadovoljni našom mrežom, želimo ju spremiti u datoteku. Razred *MLPClassifier* nam omogućuje pristup matricama skrivenog sloja preko varijable *coefs_*, a također omogućuje pristup pomacima pomoću varijable *intercepts_*. Recimo da smo našu funkciju za ispis datoteke nazvali *exportNetwork*().

```
exportNetwork(mlp.coefs_, mlp.intercepts_)
```

Ovime smo završili drugu fazu. Sada imamo spremnu simulaciju i imamo spremnu neuronsku mrežu. Jedino što je preostalo je dodati u simulaciju mogućnost upravljanja dronom pomoću neuronske mreže, a ne samo preko tipkovnice.

6. Upravljanje dronom pomoću neuronske mreže

Imamo neuronsku mrežu koju smo istrenirali pomoću scikit-learn biblioteke. Pomoću te biblioteke se i vrlo lako može pokrenuti neuronska mreža, to jest možemo joj predati ulazne podatke i ona će izračunati izlazne podatke. Međutim, u Unity-u skripte ne pišemo u Pythonu tako da se ne možemo osloniti na scikit-learn. Umjesto toga ćemo u C#-u sami napisati cijeli kod koji uz pomoć matrica skrivenog sloja i pomaka pretvara ulazne podatke u izlazne. Opet ćemo mijenjati dronovu skriptu i to opet metodu *Update*(). No prije toga treba u GUI dodati još jedan gumb na početni meni tako da korisnik uz igranje igrica sa ili bez snimanja može i odabrati simulaciju. Nakon toga, ako je igrica pozvana preko gumba za simuliranje, možemo postaviti neku zastavicu pomoću koje će se u metodi *Update*() znati treba li očitavati što je korisnik pritisnuo, ili dron treba voziti sam sebe.

U slučaju da dron vozi sam sebe, trebamo u C#-u implementirati pokretanje neuronske mreže u realnom vremenu. Na taj način će se u metodi *Update*() izvrtjeti neuronska mreža, to jest u svakom *frame*-u će se očitati što kažu senzori drona te će se uz pomoć tih senzora i neuronske mreže odlučiti kako se dron treba nastaviti kretati.

Već smo pričali kako funkcionira neuronska mreža u poglavlju o neuronskim mrežama. Sada samo trebamo programski ostvariti ono što smo teoretski obradili. Na slici 6.1. je metoda u C#-u koja pomoću senzora i podataka o neuronskoj mreži računa izlaz neuronske mreže (kako će dron voziti).

Time smo dovršili postupak grafičkog simuliranja leta drona pomoću umjetne neuronske mreže. Koliko će dron dobro letjeti ovisi o količini prikupljenih podataka, kvaliteti ulaznih podataka, skrivenom sloju neuronske mreže te o još nekoliko faktora. U sljedećem i zadnjem poglavlju ću prokomentirati rezultate neuronskih mreža nastalim različitim metodama.

```
private int[] CalculateControls()
{
    int[] simulatedControls = new int[6];
    for (int i = 0; i < 6; i++) simulatedControls[i] = 0;
    double[] currentVector = new double[sensors.Length];
    for (int index = 0; index < sensors.Length; index++)
    {
        currentVector[index] = (double)sensors[index];
    }
    for (int i = 0; i < networkMatrices.Count; i++)
    {
        double[] mul = VectorMatrixMul(currentVector, networkMatrices[i]);
        currentVector = new double[mul.Length];
        for (int j = 0; j < mul.Length; j++)
        {
            double x = mul[j] + networkIntercepts[i][j];
            if (i != networkMatrices.Count - 1)
            {
                currentVector[j] = Activation(x);
            }
            else
            {
                currentVector[j] = x;
            }
        }
    }
    for (int i = 0; i < currentVector.Length; i++)
    {
        if (currentVector[i] > 0)
        {
            simulatedControls[i] = 1;
        }
    }
    return simulatedControls;
}
```

Slika 6.1: Metoda za obradu senzora neuronskom mrežom

7. Komentiranje rezultata i eventualna poboljšanja

Rezultati se mogu podijeliti na tri djela: rezultati drona s 26 senzora s podacima jedne osobe, drona s 26 senzora s podacima više ljudi, drona sa 188 senzora s podacima jedne osobe. Dron sa 188 senzora nisam poslao drugima da ga voze. Rezultati sva 3 drona su solidni, to jest svi dronovi su sposobni izdržati dugo na stazi ako im se ne generiraju preteške prepreke, međutim, ako se prepreke (kvadri) zgusnuto generiraju, onda najčešće dron ipak ne uspije. Također, u slučajevima da uđe u zavoj kada je loše poravnat najčešće će se sudariti. Ovaj prvi problem s preprekama se s prva 2 drona događa jer nema dovoljno senzora pa u nekim situacijama ne osjeti dobro okolinu, treći dron to puno bolje rješava, ali njemu ipak nedostaje još podataka da bi postao jako dobar u tome. Što se krivog tiče ulaska u zavoj, to je u ponajviše problem premale količine podataka. Naime, ako dron ulazi u zavoj, a slučajno je poravnat previše udesno, on ne zna što da radi jer u svojim podacima ima premali broj slučajeva kad je igrač baš tako ulazio u zavoj. Razlika između drona koji je treniran na samo mojim podacima i na podacima više ljudi je da ovaj koji je samo po mojim podacima se brže sudari jer ima manje podataka, ali ovaj od više ljudi dosta sporije vozi, to jest puno češće nepotrebno pušta gas jer različiti ljudi voze različitim stilovima pa mu to očito nerijetko stvara nedoumice. U svakom slučaju, definitivno se vidi da dron sa 188 senzora ima puno bolji pregled prostora i dosta bolje vozi, ali kad njemu dodam sat vremena novih podataka, on to puno slabije osjeti od drona s 26 senzora, to jest njega treba hraniti jako velikim brojem podataka. Mislim da bi dron sa 188 senzora uz dovoljno velik broj podataka (oko 30 sati) zaista odlično letio.

Jedno od poboljšanja koje sam probao i daje dobre rezultate je dodati mehanizam sigurnog sudara, to jest u slučaju da dron osjeti da ima ispred sebe prepreku jako blizu, a neuronska mreža smatra da treba ići samo ravno (slijedi siguran sudar), u tom slučaju natjeram neuronsku mrežu da ipak mora odlučiti stisnuti još neki gumb (gumb za koji je najmanje sigurna da ga ne treba stisnuti) jer ako nastavi ići samo ravno sigurno će se

sudariti. Međutim, to sam dodao samo za slučaj smjera ravno, bilo bi možda poželjno to dodati za sve smjerove.

Najveći problem s ovim krivim ulaskom u zavoj je to što ja namjerno kad vozim se uvijek poravnavam jer želim da to i dron radi, stoga ga ne mogu hraniti podacima o krivim ulascima u zavoj, jer kada bih se ja namjerno ne poravnavao dron bi mislio da je poravnavanje nepotrebno. Dron se često i pokuša poravnati, međutim čim se poklopi dronu da se ne poravna vidi se da je puno izgubljeniji nego kada je poravnan. Stoga mislim da bi veliko poboljšanje bilo da dodam mogućnost paljenja i gašenja snimanja tijekom mog leta jer bih na taj način ja mogao ugasiti snimanje kada se odlučim ne poravnati i time će i dalje dron misliti da je dobro poravnavati se, ali bih onda upalio snimanje prije krivog ulaska u zavoj i tako bih prikupio podatke pomoću kojih bi dron znao što treba raditi u slučajevima kada je krivo ušao u zavoj.

Također bi možda bilo dobro da igrač može podešavati postavke staze prije nego što igra sa snimanjem, to jest ako igrač vidi da njegov dron odlično rješava zavoje, a loše izbjegava kvadre, onda igrač može namjestiti da se vozi stazom gdje se generira manje zavoja, a puno više kvadrova i na taj način igrač skuplja podatke koji su puno korisniji za njegovog drona.

Dodatno bi još bilo zanimljivo istražiti još više vrsta neuronskih mreža (neuronske mreže koje mogu pamtiti podatke) i algoritama za treniranje neuronskih mreža poput genetskih algoritama.

8. Zaključak

Autonomne bespilotne letjelice postoje već danas, međutim nisu još nešto što se može vidjeti često i što je dostupno svima. Jedan od mogućih načina da se stvore kvalitetna autonomna vozila je pomoću neuronskih mreža, a jedan od mogućih načina da se simulira uspješnost autonomnih vozila je pomoću grafičke simulacije.

Ovaj rad se baš i bavio grafičkom simulacijom neuronske mreže koja upravlja dronom. Pokazalo se da neuronske mreže itekako imaju potencijala za oponašanje ljudske vožnje. Kada bi se pokušalo napraviti pravi fizički dron, bilo bi potrebno dodati modifikacije u simulaciji i u samoj neuronskoj mreži, a bez sumnje bi trebalo prikupiti jako puno podataka o vožnji drona.

Na kraju bih htio dodati da zahvaljujući napretku tehnologije, autonomna vozila, čak i letjelice, nisu stvar znanstvene fantastike, no vidjet ćemo koliko će autonomna vozila zaživjeti na tržištu te koliko će ih ljudi prihvatiti.

LITERATURA

[1] B. Bašić, M. Čupić, i J. Šnajder. *Predavanja: Umjetne neuronske mreže*, 2012.

[2] scikit-learn developers. *scikit-learn user guide*, 2017.

[3] Unity Technologies. *Unity User Manual*, 2018.

Simulator leta letjelice Drone

Sažetak

Ovaj završni rad implementira grafičku simulaciju umjetne neuronske mreže za vožnju bespilotne letjelice, to jest drona. Detaljno su opisane tri faze potrebne za uspješnu simulaciju. Prva faza je izrada simulacije drona koja je ujedno i sustav za prikupljanje podataka. Druga faza je analiza podataka te izrada neuronske mreže. Treća faza je upravljanje simuliranim dronom pomoću neuronske mreže.

Ključne riječi: dron, neuronska mreža, grafička simulacija, Unity, Python, C#

Drone Simulator

Abstract

This final work implements a graphical simulation of an artificial neural network for controlling a drone. Three phases required for a successful simulation are thoroughly described. The first phase is the creation of a drone simulator, which is also intended for data collection. The second phase is data analysis and the creation of the neural network. The third phase is flying the simulated drone using the neural network.

Keywords: drone, neural network, graphical simulation, Unity, Python, C#