České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra počítačů

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Martin Stránský**

Studijní program: Otevřená informatika
Obor: Umělá inteligence

Název tématu: **Neuronové sítě pro řízení letu**

Pokyny pro vypracování:

Cílem práce bude použití neuronové sítě (Artificial Neural Network, ANN) pro řízení letounu v simulátoru FlightGear. Student vyhledá relevantní literaturu. Dále navrhne kritéria úspěšnosti, kterými se bude učená neuronová síť posuzovat a experiment. Vyvine rozhraní, kterým bude možné komunikovat se simulátorem na úrovni potřebné pro experimenty a vizualizaci výsledků. Napíše program schopný paralelizovat proces učení.

Seznam odborné literatury:

- Haykin, Simon: Neural Networks: A Comprehensive Foundation 2nd ed., Prentice Hall PTR, 1998

- Iqbal, J., A. N. Malik, and W. Haider.: Neural network based aircraft control.,Research and Development (SCOReD), 2010 IEEE Student Conference on.IEEE, 2010.

- Perry, Alexander R.: The flightgear flight simulator.,2004 USENIX Annual Technical Conference, Boston, MA.,2004.

- http://wiki.flightgear.org/Portal:Developer

Vedoucí: Ing. Jan Drchal, Ph.D.

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Ing. Filip Zelezný, Ph.D.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 3. 3. 2014

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ

Fakulta elektrotechnická

Katedra počítačů

Neuronové sítě pro řízení letu

Diplomová práce

Martin Stránský

Vedoucí práce: Ing. Jan Drchal, Ph.D.

Leden 2015

Poděkování

Děkuji inženýrovi Janu Drchalovi, PhD. za zadání práce a podporu. Dále bych chtěl poděkovat všem, kteří se podíleli na předmětu Bezpilotní prostředky, který pro mě byl nejcennějším zdrojem ohledně současných autopilotů a všeobecného rozhledu v letectví. Děkuji mé rodině za oporu při psaní této práce.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu. Nemám závažný důvod proti užití tohoto školního díle ve smyslu § 60 zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

V Praze dne .... podpis

Abstrakt

Tato práce se zaměřuje na řízení letounu pomocí neuronových sítí, vyvíjených genetickými algoritmy. Pro tyto účely byla vyvinuta rozhraní komunikující s modelem letové dynamiky (flight dynamics model, FDM) JSBSim a leteckým simulátorem FlightGear. Implementaci modelu jsme upravili, aby se evaluace zrychlila a bylo jí možné paralelizovat. Práce také obsahuje implementaci algoritmu NEAT. Dosažené výsledky lze vizualizovat pomocí FlightGearu.

Abstract

The aim of this diploma theses is to develope neural network able to deal with control of aircraft. For this purposes, interfaces for communication with flight dynamics model JSBSim and with aircraft simulator FlightGear was made. Also, NEAT algorithm was implemented. Few changes in JSBSim appears. Achieved result can be visualized by FlightGear.

Obsah

Seznam algoritmů

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

Řízení letadel pomocí počítačů nemusí být složitou úlohou. Gyroskopické stabilizátory, demonstrované již v roce 1914 Lawrencem Sperrym a jeho slavnou procházkou po křídle za letu, byly prvními přístroji řešící tento problém. Fungovaly na principu negativní zpětné vazby, která jednoduše zvyšovala výchylku řídících ploch, pakliže se letadlo vychylovalo. Nevýhoda těchto autopilotů spočívala hlavně v nespolehlivosti gyroskopu, který se mohl prudším obratem rozladit. Omezující též bylo, že stabilizátor neumožňoval nic jiného, než přímý let. Pilotovi nicméně tento systém ohromě usnadnil provedení letu, protože se mohl více věnovat navigaci. Roku 1933 se Wiley Post podruhé, tentokrát bez kopilota a navigátora, vydal na cestu kolem světa, vybaven gyroskopickým autopilotem a radiozaměřovačem. Úspěchem potvrdil užitečnost použitých systémů a dokonce vylepšil předchozí čas.

V dalších desetiletích se problémy s určením roviny letu pomocí gyroskopu nepodařilo odstranit, proto autopilot chyběl u většiny bojových letadel druhé světové války, i když byl přítomen v řízených střelách V-1. Jednou z možností zneškodnění V-1, údajně využívanou jakožto nejbezpečnější vzhledem k hmotnosti nálože, se stal přímý kontakt (klepnutí křídlem o křídlo) s létající bombou, po kterém se nedokázala vrátit a udržet se v původní poloze a spadla nedaleko místa střetu.

Zlom nastal až nástupem akcelerometrů, které umožnily vznik INS (internal navigation system, původně vyvinuto pro rakety), čímž se stalo přístrojové „sledování horizontu“ přesnějším. Odolnost vůči rychlým změnám by stále nestačila pro leteckou akrobacii, ale systém je považován za dostatečně spolehlivý pro civilní letectví. Poslední dobou, s nástupem levného výpočetního výkonu, je možné detekovat horizont pomocí metod umělé inteligence, konkrétně rozpoznáváním obrazu z kamer[1]. To umožňuje ještě lepší určování orientace letadla a spolu s ostatními systémy pomalu vytlačuje potřebu lidského faktoru.

Na základě výše uvedeného se již lze ptát, zda je možné naučit počítač létat všechny nebo většinu manévrů, které je schopen s letadlem provést člověk. Projekt akrobatického regulátoru a navádění [1] řeší tento problém informovanou cestou skrze modelování letadel a zkoušení dílčích prvků autopilota na dílčích modelech. Naproti tomu náš přístup řeší problém komplexně a jako optimalizační úlohu.

Cíle této práce jsou především:

- prozkoumat možnosti paralelizace výpočtu, aby mohlo být použito zpětnovazební učení,
- implementovat rozhraní komunikující s modelem letové dynamiky,
- implementovat učící algoritmus,
- implementovat algoritmus evaluace,
- implementovat obecnou neuronovou síť.

V dlouhodobém časovém horizontu se pokusíme vyvinout sítě schopné provést vybrané letové manévry. Každá síť bude naučená tak, že z definovaných počátečních podmínek provede manévr. Vývoj sítě se realizuje zpětnovazebním učením pomocí genetického algoritmu Neuroevolution of Augmenting Topologies (NEAT). Simulace letu probíhá v modelu letové dynamiky (FDM) JSBSim, vyhodnocení letu zpracovává vyvinutý program na základě údajů z FDM. Kritérii provedení manévru jsou za prvé dráha, kterou letoun opsal od spuštění evaluace, za druhé další proměnné nastíněné v části popisující experiment.

V první části nastíníme problém a seznámíme se s problematikou řízení letadel. Dále obecně popíšeme neuronové sítě, zpětnovazební učení, genetické algoritmy a NEAT. Druhá část pojednává o implementaci vlastního programu (VirtualPilot, psáno v programovacím jazyku Java) a pro nás důležitých částech programů JSBSim (C++) a FlightGear. Také zde popíšeme vlastní implementaci NEAT. V závěrečné části shrneme dosažené výsledky a navrhneme další rozšíření.

Podobná práce, zabývající se též řízením v 6DOF systému je práce Faustino J. Gomeze a Risto Mikkulainena. Konkrétně se jedná o řízení raket pomocí neuroevoluce. Na tento článek jsem narazil až po implementaci NEAT, jimi zvolený postup Enforced SubPopulations (ESP) by mohl výrazně snížit počet evaluací. Na druhou stranu ESP vyvíjí síť typu feed-forwad s jednou skrytou vrstvou (viz 4), což může být limitující v případě, že by byla potřeba interní paměť sítě. Manévr, provádějící raketa letící na orbit země, je také jednodušší, než námi zamýšlené manévry.

Část I

Teorie

Kapitola 2

Problém

První pokusy o využití neuronových sítí v řízení letounů se objevují už na konci 80. let spolu s rozmachem neuronových sítí [2]. Výhody plynoucí z takového přístupu jsou jednak rychlost, s jakou může být výpočet proveden, zvláště při použití paralelizmu, jednak schopnost učit se a tedy vylepšovat síť jak při simulaci, tak i doslova za letu. Existují přitom dva možné přístupy:

1. integrovat síť do již vyvinutého řídícího systému tak, aby nahradila některou z jeho částí. Získá se tím schopnost učení za letu.

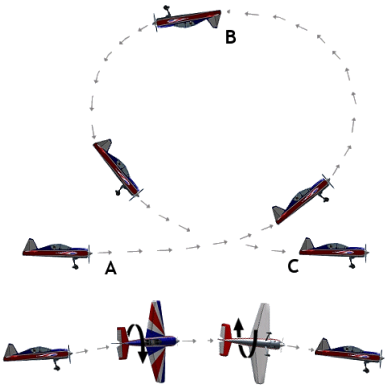
2. nahradit sítí celý systém.

V práci se zaměříme více na druhou možnost, nicméně, jak popíšeme dále, některé prvky řídících systémů musíme ponechat u strojů staticky nestabilních a s proměnnou geometrií křídla.

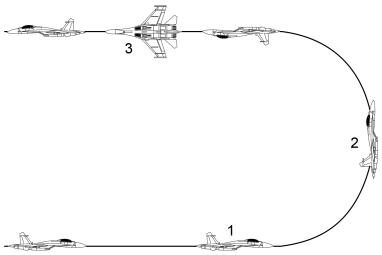
Myšlenka našeho přístupu k řízení letadla je následující. Předpokládejme, že letoun chceme provést po určité trase, která se skládá z přechodů mezi určenými polohami. Těmto přechodům říkáme manévry. Tyto manévry mohou být i akrobatické, protože definice trasy může být libovolná a je otázkou výkonu učícího algoritmu, jestli se dokáže trase přizpůsobit - za předpokladu, že je to z hlediska letové dynamiky možné.

Trasu si dekomponujeme na polohy, do kterých se má letoun dostat, a manévry, které mezi nimi má vykonat. Dále předpokládejme sadu neuronových sítí, u kterých známe druh manévru, který se provede z určité polohy. Z těch trasu složíme tak, že jakmile jedna síť dosáhne definované polohy, předá řízení síti druhé, která provádí vlastní manévr, dokud nedosáhne další polohy atd. Ilustrovat si to můžeme na obrázcích 2.1 a 2.2. Naučili-li jsme dvě sítě na přemet a výkrut, pak immelmann provedeme tak, že nejprve zapneme síť vykonávající přemet a v bodu B přepneme na síť naučenou na výkrut, která letadlo otočí (3). Jednodušším příkladem může být přechod do ostré zatáčky, setrvání v ní, přepnutí sítě a její následné vybrání.

Je otázkou, zda nechat síť zcela neinformovanou o zamýšleném manévru. Jelikož náš přístup nezohledňuje vítr, všechny manévry by jeho vlivem vypadaly jinak. Jednou z možností by bylo dát síti navíc na vstup vektor, který by ukazoval směrem na



Obrázek 2.1: Akrobatické manévry výchozí: přemet, výkrut. Zdroj: rcmodel.cz



Obrázek 2.2: Akrobatické manévry odvozené: immelmann

definovanou cestu. Taková síť, naučená na dlouhé trati s mnoha různými manévry i počasím by mohla být téměř dokonalá i ve větru. Takový přístup si ovšem nemůžeme dovolit, dokud se nám nepodaří ověřit schopnost učícího algoritmu na jednodušších úlohách.

2.1 Zpětnovazební učení

Zpětnovazební učení je metoda, kterou se agent učí orientovat v prostředí prostřednictvím maximalizace odměny. Ta mu je dána až po provedení jedné, nebo, v obtížnějších případech, několika z možných akcí. Od agenta se očekává, že bude v dalším kroku, sekvenci, volit lépe na základě předchozího ocenění.

Kapitola 3

Letouny

V této sekci nastíníme některé z principů letu letounu a jejich konstrukci.

3.1 Letoun

Pro účely práce je třeba vymezit stroje, pro které je níže popsaný algoritmus učení konstruovaný.

> Letoun (aeroplane) je letadlo těžší než vzduch s pohonem, vyvozující vztlak za letu hlavně z aerodynamických sil na plochách, které za daných podmínek letu zůstávají vůči letadlu nepohyblivé.[7]

Aerodynamickými silami se myslí především vztlak, který vzniká v důsledku obtékání křídla vzduchem o různých rychlostech. Vztlak se mění s rychlostí obtékání a úhlem náběhu, což je, zjednodušeně řečeno, úhel, které svítá křídlo s vektorem směru proudícího vzduchu.

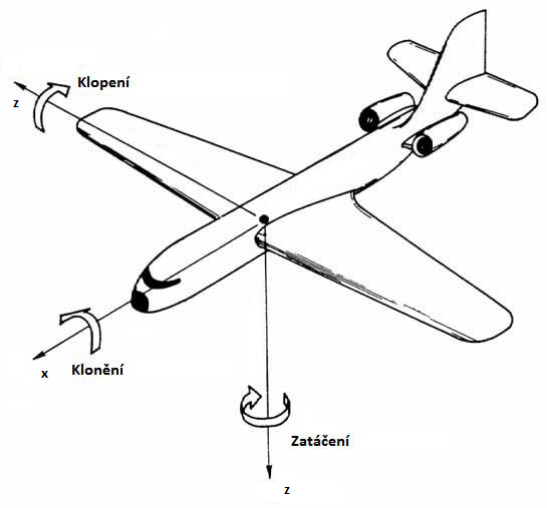
Na nejvyšší úrovni obecnosti ovládá naprostá většina letounů rotaci okolo tří os se středem v těžišti pomocí tří nebo dvou různých typů řídících ploch. Řídící plochou je zpravidla pohyblivá část ocasních ploch nebo nosné soustavy.

Osy, okolo kterých se letadlo otáčí, jsou (viz obrázek)

- osa$y$ (příčná, lateral) - klopení přídě nahoru, dolů, anglicky pitch
- osa z (vertikální) - zatáčení doleva, doprava, aglicky yaw
- osa x (podélná, longtitudal) - klonění doleva, doprava, anglicky roll

Podrobněji jsou osy popsány v části 3.4.

V nejčastějším uspořádání jsou řídícími plochami pro osy y výškovka, z směrové kormidlo a pro x křidélka. Řídící soustava letadla (ať už hydraulická či mechanická) ovládá tyto tři typy ploch zpravidla nezávisle mezi sebou. Křidélka, umístěná vždy v páru, mají obrácený chod (levé dolů, pravé nahoru a naopak), zatímco výškovky nebo směrovky, jsou-li v páru, mají chod souběžný.



Obrázek 3.1: Otáčení letadla kolem os. Zdroj: [5]

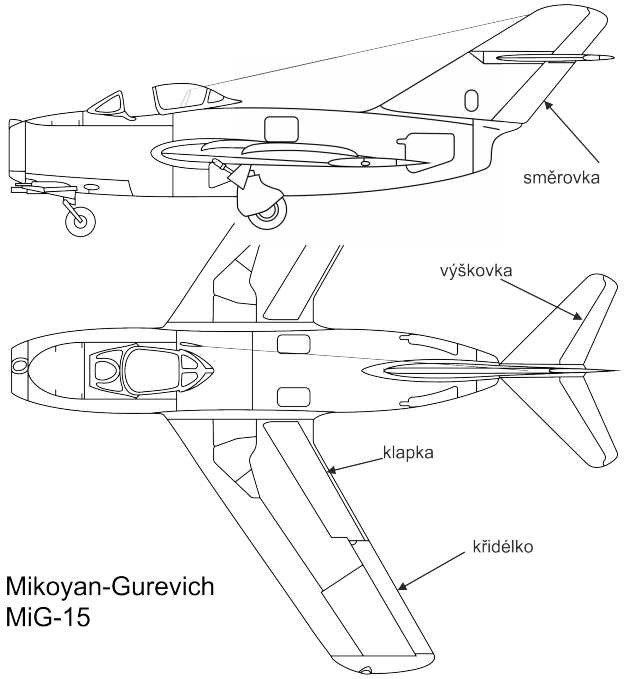
Princip řízení spočívá ve vychylování řídících ploch. Tím se mění obtékání draku letounu vzduchem, což vede k poklesům či nárůstům vztlaku na daných místech. To vytváří momenty setrvačnosti, které mění orientaci letadla vůči souřadnicovým systémům spojeným se zemí.

Popišme nyní letadlo s nejčastějším uspořádáním, poprvé představeném na Blériotu VIII, později modifikované o další prvky. Výškovky jsou umístěny vzadu na vodorovných ocasních plochách, směrovka na svislé ocasní ploše; křidélka se nacházejí na konci křídel (obrázek 3.2).

Výškovka zvyšuje či snižuje vztlak vodorovných ocasních ploch. Snížení vztlaku vede k poklesu zadní části draku letadla (včetně mírného poklesu těžiště), čímž se zvýší úhel náběhu křídel. Vzhledem k tomu, že křídla se na vztlaku podílejí více, než ocasní plochy, celkový vztlak nakonec stoupne a letadlo začne stoupat. Opačně to funguje analogicky.

Křidélka mění vztlak křídel. Křídlo, na kterém se křidélko pohne směrem nahoru, ztratí vztlak a začne klesat. Úbytek celkového vztlaku je kompenzován na opačném křídle křidélkem směrem dolů. Nakloněné letadlo má tendenci „sklouznout po křídle," protože vztlak nepůsobí přímo proti gravitační síle a vytváří boční složku; nedostatek vertikální složky vztlaku zase způsobí klesání. Také zatáčí podél osy z dolů, protože při klouzání po křídle se vertikální ocasní plocha ofukuje ze strany, čímž vytváří točivý moment.

Směrovka mění obtékání vertikální ocasní plochy. Vychýlená směrovka doprava vytvoří vztlak směrem do doleva, kam se také otočí celý zadek letounu. Tím se zároveň vůči vzduchu urychlí levé křídlo a pravé zpomalí, což má také vliv na jejich vztlak. Celkový pohyb je tedy jak zatáčení, tak klonění



Obrázek 3.2: Umístění řídících ploch.

Vztlakové klapky jsou umístěny mezi křidélky a trupem. Podle typu se vysouvají nebo vyklápí, popřípadě kombinují oba pohyby. Slouží k zvýšení vztlaku za cenu výrazného zvýšení odporu. Zpravidla se používají k snížení pádové rychlosti, což je v přímém letu rychlost, kdy se tíha rovná vztlaku a zvýšením úhlu náběhu vztlak již neporoste. Vysunují se plně před přistáním, středně na startu, výjimečně se malá výchylka používala v leteckých soubojích či v kluzácích pro zmenšení poloměru zatáčky (utažení). V jiných případech nemají použití.

Motory generují tah ve směru osy x. V přímém rovnoměrném letu mají za úkol vyvážit odporové síly působící ve směru $-x$. Vrtulové motory můžou vytvářet momenty ve všech osách, vždy ale kloní. Při rychlém přidání plynu nízko nad zemí často dochází k leteckým nehodám, když silný motor začne roztáčet pohyblivé části, zatímco letadlo se přetočí na stranu opačnou, což změní jeho vztlak a následuje pád.

Už pro správné provedení zatáčky je potřeba souhra všech kormidel. Směrovka a při velkém náklonu i výškovka způsobuje zatáčení, křidélky se řídí náklon. Pilot má v letadle možnost kontrolovat průběh zatáčky pomocí příčného relativního sklonoměru (kuličky). To je v podstatě jednoduchá obrácená vodováha, která ukazuje vychýlení výslednice sil působící na letadlo v příčném směru od vertikální osy z. Je-li kulička jinde než uprostřed, zpravidla to znamená, že se letadlo nějakým způsobem sune bokem, což je manévr zvaný skluz a používá se k brzdění (snížení celkové energie).

Lety na minimální rychlosti jsou specifickou úlohou. Ocasní plochy ztrácejí vztlak v poměru pomaleji, než křídla, při zpomalování tedy letadlo klopí dopředu. Pro zachování roviny $x-y$ dráhy letu musíme zvyšovat výchylku horizontálních ocasních řídících ploch, dokud se tah motoru nevyrovná s odporem, nebo dokud nepřekročíme úhel náběhu tak, že se letadlo začne vlivem ztráty tlaku propadat. Při propadání stačí malé otočení okolo osy z k tomu, aby se jedno křídlo urychlilo natolik, že bude mít dostatečný vztlak na nesení letadla. Potom dojde k přepadnutí na jednu stranu a otáčení letadla tak, že křídlo se vztlakem bude nahoře vně zatáčky, druhé bez vztlaku uvnitř ní. Tomuto jevu se říká vývrtka.

Pro úplnost ještě zmíníme let na zádech, který je z fyzikálního hlediska podobný normálnímu letu, jen je potřeba velkých výchylek výškovky, aby se vytvořil dostatečný úhel náběhu křídel. Letadlo musí vytvářet vztlak ofukováním křídla z opačné strany. Některá křídla mají symetrický profil, i tak ale žádné letadlo nemá rovinu tětivy profilu stejnou, jako je rovina letadla, a proto bude let na zádech výrazně odlišný od pilotáže normálního letounu.

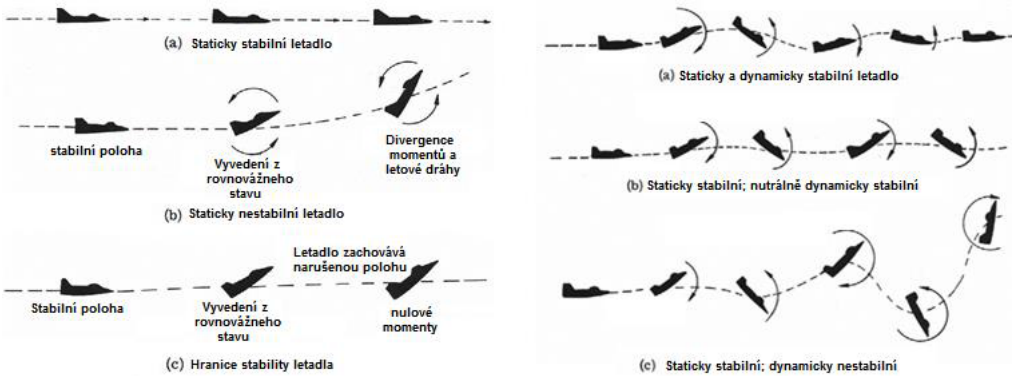
3.2 Vztah řízení a odezvy

Pro vyhodnocení letu je nezbytné zhodnotit všechny komponenty, které se na letu podílejí. Stěžejní je matematický model letadla, tvořený pohybovými rovnicemi. Do toho vstupují tři faktory: počáteční stav, stav konfigurovatelných prvků letadla a vychýlení v důsledku atmosférických jevů. Počáteční stav je v našem algoritmu většinou dán stavem předchozím, do kterého se letoun dostal předchozí interakcí s neuronovou sítí. Některé konfigurovatelné prvky řídí neuronová síť, zbytek zůstává konstantní po celou dobu učení. Atmosférické jevy JSBSim nabízí, budeme je ale při učení zapínat jen tehdy, když by síť nebyla schopna obstát při vizuálním testu s nimi. Hrozí totiž, že by se simulace mohla stát nedeterministickou, což by mělo negativní dopad na výkon učícího algoritmu.

3.3 Fly-By-Wire

Práce neusiluje o stabilizaci letadel staticky nebo dynamicky nestabilních. Statická stabilitu letadla vypovídá o chování letadla při ustáleném režimu letu, a je důležitá jako indikátor schopnosti pilota nastavit a udržet letadlo v požadované poloze. Jedná se především o rychlou a odměřenou reakci na vychýlení z ustáleného režimu. V případě pozdní reakce totiž staticky nestabilní letadlo zvětšuje výchylku, na rozdíl od letadla neutrálního, které výchylku zachová, a stabilního, které se vrátí do ustálené polohy. Reakce nepřiměřená výchylku neopraví, nebo vyvolá výchylku opačnou. Přesto je mnoho moderních letadel staticky nestabilních, a to díky sytému fly-by-wire, který pilotovi řízení usnadní.

Stabilizace nestabilních letadel je problém jiný, než si tato práce klade za cíl, proto se jí nebudeme věnovat, i když by náš přístup mohl vést i zde k aplikovatelným výsledkům. Jedním z důvodů je i potřebná doba odezvy, která, čím je kratší,



Obrázek 3.3: Statická a dynamická stabilita. Zdroj: [5]

tím déle trvá ohodnocení sítě a tím delší je běh celého učícího algoritmu. Pokud bychom přesto chtěli používat modely nestabilních letounů, předpokládáme zmíněný regulátor, který má na vstupu výstup řídící neuronové sítě.

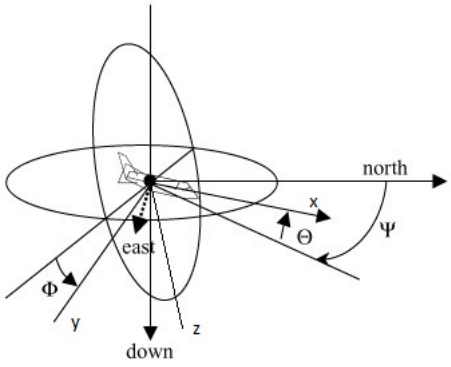
3.4 Souřadnicové systémy

JSBSim používá konvenční značení (podle některých zdrojů shodné s normou ISO 1151–2:1985, tyto normy nejsou zdarma k nahlédnutí) souřadnicových systémů a os, nicméně pro přehled je zde uvedeme, jelikož v manuálu k JSBSim chybí. Stejný systém používá i FlightGear, nicméně v *property tree* (viz část 6) jsou značeny jinak.

První použitý souřadnicový systém má počátek v těžišti letadla a rotuje spolu s letadlem. Podle pilota jsou osy umístěny takto: x dopředu, y doprava po křídle, z dolů. K nim odpovídající rychlosti dopředu, doleva, dolů U, V, W a úhlové rychlosti směrem nahoru, rotace doleva a klopení doprava P, Q, R. Tato soustava se česky nazývá letadlová [5].

Pro řízení použijeme referenční rámec, který má počátek v těžišti, ale s letadlem nerotuje a je svázán se zemí (referenční rámec NED - north, east, down). V něm rozeznáváme úhlové výchylky od severu po horizontu doprava ψ, od horizontu vpředu nahoru $-\theta$ a od horizontu vpravo dolů ϕ. Anglicky se tyto úhly nazývají heading, elevation, bank a patří k Tait-Bryanovým úhlům.

Zásadní nevýhodou tohoto systému je, že v určitých pozicích dochází při malé změně o absolutní úhel k úplnému otočení hodnot jednoho z úhlů. Pro pokusy tento systém ponecháme kvůli jeho jednoduchosti, nicméně v případě neúspěchu bychom mohli uvažovat o jiných popisech, nejlépe pomocí kvaternionů. Ty mají zase nevýhodu ve vyšší dimenzi (čtyři čísla naproti třem).



Obrázek 3.4: Osy x, y, z a eulerovské úhly ψ , θ , ϕ. Zdroj: globalspec.com

3.5 Evaluace

Pro zpětnovazební přístup v učení musíme vyvinout metodu, která síti přiřadí na základě nasimulovaného manévru odměnu. Zvolený přístup umožňuje kombinovat více možností.

1. Porovnat nadefinovanou ideální trajektorii letu od skutečné. To se v praxi děje tak, že v diskrétních časech počítá vzdálenost aktuální polohy letounu k nejbližšímu bodu (úsečce) trasy. Chyba se načítá.

2. Může také vzít jakýkoli proměnnou poskytnutou modelem letové dynamiky a jeho hodnotu se snažit optimalizovat (minimalizovat, absolutní hodnota rozdílu od ideálu).

Druhý bod může být použít hlavně tehdy, nebude li experiment úspěšný přesto, že výsledná fitness bude uspokojivá. Například se může minimalizovat síla působící v ose $\pm y$ značící bočení letadla, způsobující nárůst odporových sil. V přímém letu lze maximalizovat uletěnou vzdálenost.

Opravdovou evaluaci ovšem představuje až náš „lidský“ náhled na vizualizaci simulace, protože

- jednotlivé pokusy nejsou mezi sebou porovnatelné, změníme-li podmínky,
- ne všechny žádané parametry vložíme do evaluační funkce, a i kdyby ano, pak neznáme funkci, pomocí které je kombinovat.

Kapitola 4

Neuronové sítě

Neuronová síť je tvořena výpočetními jednotkami, které se nazývají neurony. Neurony jsou mezi sebou propojeny tak, že celá síť může být popsána prostým orientovaným grafem. Pro každý uzel i s přechodovou funkcí $f(i)$ platí

$$y_i = f_i \left(\sum_{j=1}^{N} w_{ji} x_j - \theta_i \right) \tag{4.1}$$

kde y_i je výstup neuronu i, x_j jeden z jeho N vstupů, w_{ij} váha propojení neuronu j do i a θ_i práh (bias) i. Přechodová funkce je obvykle nelineární funkce jako heavisidovská, sigmoida, gaussovská a podobně.

Rozlišujeme vstupní neurony, které jediné nemají přechodovou funkci a slouží pouze k předání vstupních hodnot ($y_i = x_i$), a výstupní, ze kterých čteme výstup sítě. O množině vstupních neuronů hovoříme jako o vstupní vrstvě, u výstupních o výstupní vrstvě, zbylé nazýváme vrstvou skrytou.

Kvůli snadnější implementaci se často vynechává práh neuronu θ_i a nahrazuje se vstupním neuronem $bias = 0$ s konstantním výstupem $y_{bias} = 1$ a váhami spojení $w_{bias\,j}$ vedoucími do všech neuronů j ze skryté a výstupní vrstvy.

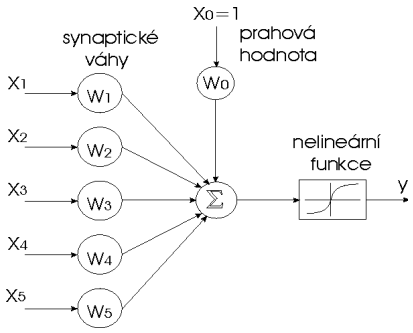
$$\theta_j = -w_{bias\,j} x_{bias} \tag{4.2}$$

Výsledná funkce je tedy

$$y_i = f_i \left(\sum_{j=0}^{N} w_{ji} x_j \right) \tag{4.3}$$

V případě, že graf sítě tvoří dvě a více vrstev, kde hrany vedou jen mezi dvěma sousedními vrstvami ve směru od vstupních do výstupních, hovoříme o *feed-forward* neuronové síti, jinak jí nazýváme rekurzívní. Rekurentní neuronové sítě si narozdíl od *feed-forward* sítí mohou uchovávat hodnoty z minulých vyhodnocení. Mechanizmus závisí na topologii sítě a algoritmu vyhodnocování, my ovšem vzhledem ke zvolenému algoritmu učení topologii dopředu neznáme, uvažujeme tedy o rekurentní síti obecné.

Feed-forward neuronová síť má pouze hrany mezi jednotlivými vrstvami.



Obrázek 4.1: Neuron

4.1 Vstupy a výstupy neuronové sítě

Z hlediska implementace je možné poslat síti na vstup jakoukoli proměnou, kterou nabízí prostředí FDM. Zvolíme ovšem takové parametry, které by si eventuálně mohl reálný letoun sám naměřit či odvodit a zároveň takové, které má smysl vkládat do sítě za předpokladu, že jedním z cílů je možnost síť zapnout či vypnout za letu, tedy aby nebyla závislá na čase, uletěné vzdálenosti a podobně. Zamýšlené parametry jsou z kategorií

- orientace vůči NED,
- orientace v kvaternionech,
- rychlost,
- zrychlení,
- korekce pozice (získaná z GPS),

Na výstupu ovšem budeme vyžadovat ovládání alespoň tří či čtyř základních prvků, což je výškové kormidlo, směrové kormidlo, křidélka a plyn. Tato čtveřice tvoří minimum, které je potřeba k ovládání letadla z hlediska možnosti provedení většiny zamýšlených manévrů. Další prvky jako ovládání podvozku, vztlakových klapek a spojlerů vynecháme.

Kapitola 5

Evoluční algoritmy

Evoluční algoritmy patří mezi optimalizační algoritmy. Cílem je pomocí definovaných pravidel inspirovaných evolučními procesy v přírodě vyvinout pomocí křížení a náhod-ných mutací jedinců nové a lepší jedince, které reprezentují řešení úlohy. Celý proces se děje iterativně, populaci jedinců v jedné iteraci říkáme generace. Každé nově vzniklé řešení se ohodnotí pomocí fitness funkce, která říká, nakolik je jedinec vhodný pro generování další generace. Jedinec s relativně nízkou fitness má malou nebo žádnou pravděpodobnost, že se na další generaci bude podílet. Obecný algoritmus je popsán pseudokódem 1. Zde G_n reprezentuje n-tou generaci. Ukončovacích podmínek je celá řada od dosažení dostatečně dobré fitness, přesáhnutí maximálního počtu generací, vyčerpání časového limitu a jiné sofistikovanější. Stejně tak i výběr nejlepších jedinců se může být jednoduše $r\%$ nejlepších, občas se pro zachování diverzity používají i jiné postupy, jako *Roulette Wheel Selection*.

Problémem těchto nedeterministických optimalizačních algoritmů je, že těžko poznáme, kdy jsme dosáhli lokálního optima, a nemůžeme poznat globální optimum. To implikuje, že neexistuje dobrá zastavovací podmínka. Často se tedy volí buď omezení časové, omezení počtu generací nebo zastavení poklesu průměrné fitness populace.

5.1 Genetické algoritmy

V genetických algoritmech jsou jedinci reprezentováni genomem, což je zpravidla struktura kódující řešení, nad kterou ale lze provádět křížení a mutace. Může to být řetězec znaků, ale i složitější konstrukce. Genom se skládá z genů, které určují vlastnosti organismu.

Instance genomu se nazývá genotypem. Fenotypem se rozumí jedinec vytvořený na základě genotypu v konkrétním prostředí. Definujeme interpretační funkci $f_G : G \to P$ z množiny všech genotypů do fenotypů (vliv prostředí se často v evolučních algoritmech zanedbává, jelikož prostředí považujeme za homogenní). U fenotypu pak definujeme fitness funkci $f_P : P \to \mathbb{R}$. Složením obou funkcí dostáváme fitness funkci $f : G \to \mathbb{R}$, pomocí které ohodnocujeme genotyp jedince. Fitness se může

Vstup: $\emptyset$
Výstup: nejlepší jedinec *nej*
vygeneruj první generaci G_0;
$n = 0$;
while *ukončovací podmínka není splněna* **do**
  ohodnoť jedince v G_n;
  $nej = \max(G_n)$;
  do S vlož vhodné jedince pro další generaci z G_n;
  $S =$ křížení(S);
  $G_{n+1} =$ mutace(S);
  $n = n + 1$;
end
vrať *nej*

Algoritmus 1: Obecný evoluční algoritmus

maximalizovat nebo minimalizovat, je ale vhodné, aby její rozsah neprotínal nulu, což by komplikovalo použití některých přístupů, jako níže zmíněný *fitness-sharing*.

5.2 NEAT

NeuroEvolution of Augmenting Topologies je jedním přístupů k učení neuronových sítí pomocí genetického programování. Jeho hlavní výhodou je, že vyvíjí z nejjednodušších sítí postupně větší s různými topologiemi. V našem problému lze očekávat, že zatímco na některé řízení některých manévrů bude stačit síť jednoduchá, možná i bez skrytých vrstev, u jiných se potřebná topologie odhaduje těžko. Z toho důvodu může být výhodné nemít topologii sítě fixní a nechat jí vyvinout.

Použitá implementace NEAT vychází z původního článku od Kenneth O. Stanley a Risto Miikkulainen z roku 2002 [3].

5.2.1 Genetické kódování

Neuronová síť se v NEAT kóduje pomocí seznamu spojů mezi neurony, představující geny. Součástí spoje jsou

1. váha z reálných čísel,
2. celá čísla odkazující na zdrojový a cílový neuron,
3. globální inovační číslo,
4. příznak, zda je spoj aktivní.

Globální inovační číslo (GIN) je přirozené číslo, které je unikátní pro každou dvojici zdrojového a cílového neuronu. Slouží ke zjednodušení operace křížení. Křížení

různých neuronových sítí obecně vyžaduje složitou analýzu topologie, která navíc bývá výpočetně náročná a snižuje výkon algoritmu. Nové GIN se přiřadí ve chvíli prvního vzniku spoje a každý další spoj mezi stejnými neurony má přiřazen stejné GIN. Tím se křížení výrazně zjednodušuje, protože z těch dvojic spojů, které mají shodné GIN, se při křížení vybere jeden a zbytek spojů, který se neshoduje, je přejat z rodiče s vyšší fitness. Vzniklý jedinec je tedy vždy větší nebo stejně velký, jako lepší z rodičů. Pakliže byl v jedné ze dvojic se shodným GIN příznak aktivace negativní, v novém jedinci bude spoj neaktivní.

Obecně je mutace náhodná změna genu a křížení výměna genů mezi dvěma rodiči, což má jasnou inspiraci v přírodě. V mutaci rozlišujeme tři různé operace:

1. mutace váhy
2. přidání spojení
3. přidání neuronu

Mutace váhy je přičtení náhodného čísla k váze. To se děje s danou pravděpodobností. Propojením dvou neuronů buď vzniká nové propojení s novým GIN, nebo se přiřadí již existující GIN. Stejně tak u přidání neuronu, které vytváří dva nové spoje do a z nového neuronu na místě, kde spoj již byl. Starý spoj se v tom případě ruší, gen starého spoje však nezaniká, místo toho se mu nastavuje příznak na neaktivní spoj. To je důležité pro porovnávání topologií, viz níže.

5.2.2 Druhy a sdílená fitness

Dělení do druhů (speciation) je jedna z komponent algoritmu NEAT. Jde o způsob, jak ochránit nově vzniklé topologie a zároveň zajistit, že se velmi odlišné sítě nebudou křížit. Předpokládáme totiž, že zkřížením dvou příliš rozdílných jedinců pravděpodobně nevznikne smysluplný potomek. Dalším důvodem je možnost chránit nové topologie tak, že je jim přidělen nový druh, čímž dostanou (díky sdílené fitness) několik iterací na vylepšení vah tak, aby mohli konkurovat již ustáleným topologiím.

Rozdílnost jedinců je dána jednak topologií, jednak váhami. Namísto analýzy topologií opět využíváme faktu, že čím větší je poměr společných genů (se stejnými historickými značkami) ku ostatním, tím jsou jedinci ke křížení vhodnější. Mezi geny, které nejsou společné, se ještě rozlišují nové (*excess*) geny, definované jako geny s vyšším číslem, než jaké je nejvyšší v genomu s menším nejvyšším číslem, a zbylé (*disjoint*).

U dvojice genomů určujeme vzdálenost (odlišnost) δ jako lineární kombinaci počtu *dissjoint* (D) a *excess* (E) genů a rozdílů vah $\bar{W}$ u společných genů.

$$\delta = \frac{c_1 E}{N} + \frac{c_2 D}{N} + c_3 \bar{W} \tag{5.1}$$

Koeficienty c_1, c_2, c_3 určují váhu jednotlivých faktorů a N je počet genů ve větším genomu. Tuto vzdálenost porovnáme s zvoleným prahem δ_t a je-li nižší, jsou považovány

```
Vstup: množina druhů S, množina nezařazených jedinců P
Výstup: množina druhů S
for jedinec j := P do
    for druh s := S do
        delta = porovnej reprezentanta druhu s s j;
        if δ < δ_t then
            přidej jedince j do druhu s;
            break;
        end
    end
    if j nebyl přiřazen do žádného druhu then
        vytvoř nový druh ns;
        reprezentant ns je j;
        přidej ns do S;
    end
end
```

Algoritmus 2: Vytváření a zařazování druhů

za příslušníky stejného druhu. Pseudokód 2 popisuje vytváření nových druhů a zařazování jedinců do druhů stávajících.

Novou velikost druhu určuje poměr sdílené fitness ku celkové (u maximalizace), přičemž sdílená fitness je průměrná fitness všech reprezentantů druhu a celková průměrem mezi druhy. Mezi nejlepšími r procenty jedinců z druhu probíhá křížení a mutace, dokud nevznikne dostatečný počet jedinců, který odpovídá nové velikosti druhu. Tito noví jedinci nejsou automaticky zařazeni do stejného druhu jako rodičové, ale jsou znovu porovnáni s reprezentanty a pokud není nalezen vhodný druh, zařadí se do nového.

5.2.3 Selekce

V kódu jsou implementovány dva různé způsoby selekce. Jeden odpovídá již změněnému striktnímu rozřazení pomocí procentního prahu na fitness, to má ovšem nevýhodu jednak ve výpočetním výkonu, druhak se může stát, že všichni dobří jedinci dosáhnou lokálních optim a neponechají žádnou možnost horším nově vzniklým jedincům k vyváznutí z tohoto optima.

Roulette Wheel Selection umožňuje proniknutí i nejhorším jedincům s pravděpodobností

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}$$

kde i je číslo jedince, f_i jeho fitness.

Část II

Implementace

Kapitola 6

JSBSim a FlightGear

JSBSim je open-source softwarová knihovna modelující dynamiku letu letadel. V referenční příručce je uvedeno:

> JSBSim byl koncipován roku 1996 jako kompaktní, daty řízená, nelineární dávková simulace se šesti stupni volnosti (6DoF), zaměřená na modelování letové dynamiky a řízení letadel.[8, (překlad)]

Téměř všechny proměnné simulace jsou umístěny ve struktuře podobné adresářům a lze k nim přistupovat pomocí spojení přes UDP nebo TCP. Program byl mírně upraven, především třídy FGInput, proto doporučujeme používat přiloženou verzi programu.

Nutná je kompilace JSBSim z přiloženého kódu. Přiložený kód se kompiluje stejně jako oficiální distribuce, jejíž kompilace je popsána v příručce [8]. V prostředí NetBeans C++ kompilace neproběhne automaticky, je třeba nejdříve pravým tlačítkem myši kliknout na CMakeList.txt a zvolit vygenerovat Makefile a poté kliknout na build.

6.1 Volba rozhraní

Vývoj rozhraní byl největší výzvou v celém projektu. Nedostatečná dokumentace jak k JSBSim (dokumentace je obsáhlá, ale zaměřená na modelování letadla), tak k FlightGearu neumožňovala jednoduše shromáždit fakta o možnostech komunikace a rozhraních a ty musely být extrahovány z internetových fór a nejčastěji přímo z kódu. Výkon těchto prvků musel být otestován pomocí alespoň základních klientů. Na základě toho pak byl zvolen níže popsaný přístup.

JSBSim nabízí možnost komunikovat přes sockety. Jedná se o obousměrný provoz, kde klient (učící algoritmus) posílá žádosti a příkazy na server (JSBSim) a ten odpovídá. Nabízí ještě další možnosti, a to přímé přesměrování po socketech nebo zápis do souboru a výroba vlastního rozhraní. Obě měly zásadní nevýhody, proto jsem se rozhodl pro obousměrný provoz. Nevýhoda přímého přesměrování spočívala v nemožnosti specifikace výstupního proudu ve skriptovacím souboru JSBSim. Podle

manuálu by se měly totiž příkazy vložit do souboru s letadlem, ačkoli tag s obousměrným provozem akceptoval i ve skriptovacím souboru. Při paralelním spouštění by to znamenalo, že každá vlákno bude mít vlastní letadlo. Možnost úpravy kódu JSBSim se vzhledem k nezaměnitelnému pořadí zpracování souborů nezdála možná, navíc by provoz obou systémů (zvlášť obousměrný na nastavování proměnných a zvlášť odchytávání výstupu) měl dalekosáhlé důsledky pro synchronizaci i evaluaci. Kdyby server posílal data v jiný časový okamžik (myšleno simulace), než kdy se vyhodnocuje síť, mohla by vzniknout mezi přečtením a vyhodnocením proměnných časová mezera téměř jako perioda vyhodnocení, nehledě na potřebu vlastního vlákna pro klienta.

Nové rozhraní pro svou pracnost nepřipadalo v úvahu, zvláště při požadavku na možnost měnit snadno vstupy a výstupy. Existuje rozhraní mezi JSBSim a FlightGearem a ve FlightGearu server, který dokáže komunikovat pomocí UDP i TCP rychleji, ty jsou ovšem natolik rozsáhlá a propojená s dalšími komponentami, že jen jejich separace by byla tématem na semestrální práci. To dokazuje i to, že požadavek na možnost spouštění FlightGearu off-screen - bez grafiky - je již minimálně rok a půl v plánech open-sourcového vývoje a přes vypuštění dvou majoritních updatů 3.0 a 3.2 je stále v nedohlednu. Autor této práce původně počítal právě s touto možností a má vypracovaný klient na komunikaci s FlightGearem.

Ještě podotkněme, že externí program s rozhraními má oproti programu přímo napojenému na JSBSim tu výhodu, že není závislé na platformě. Vyvinuté sítě nemusí být připojeny k tomuto simulátoru, ale může být vyvinuto další rozhraní pro jiné simulátory.

6.2 Výkon rozhraní

Při prvních pokusech se zjistilo, že JSBSim je v dávkovém módu velmi rychlý. Frekvence iterací se pohybovala podle použitého modelu od okolo 600.

V testech se zdálo, že by mělo být možné zaslat až 40 socketů za sekundu, což by vzhledem k cílová frekvenci 10 Hz mělo zrychlit evaluaci minimálně dvakrát oproti simulaci v reálném čase (čtyřikrát v případě, že bychom tolerovali zpoždění odpovědi sítě v dálce jedná desetiny vteřiny; na předchozí odpověď odpovídat zároveň s dotazem). Později se za prvé ukázalo, že 1/10 vteřiny mezi dvěma vyhodnoceními je už pravděpodobně na hraně efektivní reakce při používání velkých výchylek, za druhé že délka dotazu, nejspíše kvůli pomalému zpracování na straně serveru, mírně ovlivňuje odezvu. Důsledek je relativně pomalý běh způsobený právě komunikací; rychlost se blíží reálnému času. To se naštěstí podařilo zmírnit pomocí paralelismu. Čtyřjádrový stroj byl schopen obsloužit 8 vláken, aniž by ty vykazovaly zpomalení. Sice vzrostly oproti předpokladu náklady na obsluhu více vláken, na druhou stranu to není natolik zásadní, aby se vyplatilo kvůli tomu měnit celý přístup ke komunikaci.

6.3 Nastavení JSBSim

JSBSim nabízí celou škálu možností, jak pomocí XML souborů nastavit

- veškeré parametry letadla včetně bodů kontaktu se zemí,
- definovat nové proměnné,
- deklarovat vstup a výstupy,
- události, při kterých se automaticky mění parametry nebo se simulace resetuje
- definovat autopilota.

JSBSim používá tři typy konfiguračních souborů, které nás zajímají. Prvním je soubor reset01.xml, který nastavuje pro dané letadlo výchozí parametry simulace. Vypovídá o pozici v systému ECI (Earth Centered Initial, dvojice zeměpisná délka, šířka), rychlosti ve třech osách x, y, z (značeno *ubody*, *vbody*,*wbody*), orientaci (*attitude*), větru a turbulenci. Dále soubor definující letadlo. Je v něm umístěna informace o výstupu do FlightGearu. Iniciační soubor obsahující událost (*event*) start motoru a informaci o vstupu. Iniciačních souborů je třeba tolik, kolik instancí hodláme spustit, proto jsou na začátku LEARNING.SH tyto soubory generovány pomocí dalšího skriptu.

Kapitola 7

Neuronové sítě

Neuronová síť je tvořena potomky abstraktních tříd AbstractGeneralNeuron a AbstractConnection. Tyto třídy obsahují abstraktní metodu process(), kterou volá třída reprezentující neuronovou síť GeneralANN.

7.1 Algoritmus evaluace

Třída AbstractGeneralNeuron obsahuje metodu sum(), která provede součet popsaný vzorcem 4.3. Implementace této třídy pak obsahuje metodu process() (zpracuj) s přechodovou funkcí. Třída reprezentující síť složenou z neuronů implementuje rozhraní ArtificialNeuralNetwork, a poskytuje metody process(), mutate(), cross(), copy(), reset(). Metoda process(double[] d) provede jednu iteraci sítě s danými vstupním vektorem a vrátí výstupní vektor. Metoda reset() vynuluje paměť sítě.

U rekurentních sítí je více možností, jak spočítat výstup sítě, jelikož výsledek závisí na pořadí výpočtu jednotlivých neuronů. Implementovaný algoritmus popisujeme pseudokódem 3 na následující straně. Tento algoritmus zaručuje, že každý neuron bude zpracován právě jednou (za předpokladu, že při obrácení hran grafu topologie vede cesta z neuronu do některého vstupního neuronu) a v každém vyhodnocení dané sítě se neurony zpracují ve stejném pořadí. Při vyhodnocení feed-forward neuronové sítě se vyhodnotí všechny neurony ve smyslu od vstupních vrstev přes skryté po výstupní vrstvy během jediné iterace tak, aby výsledek odpovídal feed-forward ANN a nebyl ovlivněn předchozí iterací.

Implementace nebrání použití jakýchkoliv neuronů, nicméně používáme výhradně neurony s hyperbolickou tangentou. Důvodem je jejich definiční obor, který odpovídá normalizovanému rozsahu řídících prvků (kromě plynu), takže už se nemusí normalizovat.

```
Vstup: vstupní neurony I, výstupní neurony O, vstupní
       vektor v_in
Výstup: výstupní vektor v_out
vlož do vstupních neuronů I vstupní data z v_in;
přidej všechny vstupní neurony I do nové fronty F;
// while F není prázdná do
   Neuron n = vyjmi první neuron z fronty F;
   if n nebyl zpracován then
      zpracuj n;
      označ n jako zpracovaný;
      přidej všechny neurony, do kterých vede z n propojení,
      do F;
   end
end
kopíruj do vektoru v_out výsledky z výstupních neuronů O;
```

Algoritmus 3: Vyhodnocení neuronové sítě

Kapitola 8

NEAT

Algoritmus NEAT tak, jak je popsán v [3], připouští více rozdílných implementací. Není tam například řešeno, jak se zachází s neurony při křížení a nakládání s prahy neuronů. Taká je zde řada parametrů mutace a křížení, která musí být pečlivě zvoleny, jinak algoritmus nedojde k požadovaným výsledkům. O těchto podrobnostech a celková architektuře implementace NEAT pojednáváme v táto kapitole.

8.1 Struktura programu

Třídy podílející se na neuroevoluci jsou umístěny ve složce EVOLUTIONARYALGORITHM. Kostra algoritmu (1) se nachází v třídě NEAT, mutace a křížení v třídě reprezentující genom sítě ANNGENOME a operujeme s objekty konexe CONNECTIONGENE a neuronu NEURONGENE. Protože se počítá jen s jednou instancí, třída NEAT obsahuje jen statická proměnná a funkce. V souboru NEAT.java dále najdeme třídu druhu NICHE, která je potomkem objektu ARRAYLIST<ANNGENOME> a obsahuje metody na řízení generování potomků a selekci. Parametry NEAT se nachází ve třídě NEATPARAMS, globální proměnné v NEATGLOBALS. Třída NEATPARAMS umožňuje import ze souboru .xml.

V třídě NEAT se také inicializují a spravují instance potomků třídy TESTCASE, které implementují RUNNABLE a představují tedy jednotlivá vlákna řídící běh simulace. TESTCASE také volá metody některého z potomků EVOLUTIONARYALGORITHM.EVALUATOR.

Externí knihovny K parsování argumentů používáme knihovnu commons-cli 1.2. Parsování probíhá v metodě MAIN(ARGS[]) ve VIRTUALPILOT.VIRTUALPILOTEA.

Metoda TOSTRING ANNGENOME vrací graf ve formátu pro program graphviz .dot. V kódu je dále na vizualizaci knihovna pro práci s grafy jung. Po skončení učení se pomocí ní vyvolá okno s grafem s pozicemi vrcholů optimalizovaných pomocí algoritmu Fruchterman-Reingold. Obsluhu knihovny a parametry vizualizace obstarává třída EVOLUTIONARYALGORTITHM.SIMPLEGRAPHDRAW.

Další použitá knihovna je COM.VIVIDSOLUTIONS.JTS.*, která má za úkol počítat odchylky od požadovaná dráhy letu ve třídách FLIGHTMANEUVER.*. U táto knihovny

jsem ovšem narazili na nedostatek. Zdá se, že přesto, že knihovna nabízí možnost pracovat s trojrozměrnými koordinátami, výpočet vzdálenosti od přímky a velmi užitečná funkce výpočtu vzdálenosti od řetězu úseček funguje jen ve dvou rozměrech. Eulerovská vzdálenost (odchylka výšky) se tedy opravuje ve vlastním kódu. Třída NEAT generuje při běhu okna s náhledem aktuální nejlepší sítě.

8.1.1 Mutace

Při mutaci genomu mohou nastat všechny druhy mutace najednou. Metoda mutace je ovlivněna následujícími parametry:

- rozptyl gaussovsky rozložených změn vah (váha se přičítá ke staré)
- rozptyl gaussovsky rozložených nových prahů
- rozptyl gaussovsky rozložených nových spojení
- pravděpodobnost změny jednoho genu spojení (provádí se pro všechny spojení genomu)
- pravděpodobnost přidání nového uzlu
- pravděpodobnost přidání nové hrany

Poznamenejme, že prahy jsou implementovány pomocí hran, které vedou do všech vstupních i výstupních uzlů

8.1.2 Křížení, Speciation

Metoda křížení přímo nevyžaduje žádné parametry, nicméně jí ovlivňuje speciation (rozdělování do druhů) tím, že příslušníci druhu se mohou rekombinovat je mezi sebou. Parametry speciation jsou

- práh δ_t rozhodující příslušnost k druhu,
- c_1, c_2, c_3 ze vzorce 5.1.

Křížení vždy generuje dva potomky, které mají společné geny křížených stejné, zbytek rozdílný.

Kapitola 9

VirtualPilot

virtualpilotea.VirtualPilotEA je hlavní třída, která parsuje parametry. Rozděluje běh programu podle parametrů na testování - vizualizaci - neuronové sítě, nebo na provedení neuroevoluce. virtualpilotea.* obsahuje dále třídu ShowRun, sloužící ke komunikaci s JSBSim při vizualizaci, která je podobná třídě TestCase s tím rozdílem, že se v ní neprovádí evaluace fitness funkce. Třída Properties je určena k předávání proměnných v celém programu, naproti tomu třída Property pouze proměnné popisuje a normuje mezi ± 1.

Při vizualizaci běhu sítě komunikuje ShowRun s JSBSim. JSBSim předává proměnné simulace dále do FlightGearu. Celý proces se zdržuje komunikací mezi programy a tak není simulace v reálném čase. Je ale možné pořídit záznam, ze kterého se pak běh realtimově přehraje. Druhá možnost je síť program konfigurovat na komunikaci s FlightGearem, zde ale nemáme možnost nastavit všechny výchozí i průběžné parametry přesně tak, jak to bylo v JSBSim při učení, takže neuvidíme finální výsledek učení. Konfigurace programů a spouštěcí bashovské skripty jsou uvedeny v příloze.

Při paralelizaci je vhodné myslet na to, že vyhodnocují-li se sítě souběžně, pak je potřeba zvolit populaci tak, aby byla o trochu menší, než násobek vláken. Vlivem zaokrouhlování a při vzniku nového druhu může přibýt několik jedinců nad udávanou populaci, zpravidla v počtu nových druhů. Markantní to je, když se fitness některé sítě přiblíží poměrně blízko nule (při minimalizaci) - v takovém případě zafunguje zastavovací podmínka, která při vzniku dvojnásobné populace učení zastaví.

Všechny skripty jsou psány pro bash shell. Testovány byly na Xubuntu 14.04 a měly by být spustitelné na všech distribucích GNU/Linux. Skripty na prohlížení výsledků vyžadují instalaci FlightGearu ve verzi 3.0. Podle dokumentace by měly být kompatibilní i s verzí 3.2, nicméně tato verze požívá jinak popsané modely v jazyku NASAL (nemá vliv na FDM samotný, ale například na vizualizaci některých systémů), takže při přímém přehrávání bez JSBSim by mohly nastat potíže.

Poznámka k implementaci: kvůli chybě disku jsem týden před odevzdáním této práce ztratil některé z důležitých tříd. Ty jsem rychle naimplementoval znovu s pomocí starších záloh, nicméně, kvůli časově náročným experimentům, neopatřil

jsem je původními komentáři, které v zálohách ještě nebyly. Okomentované jsou tak alespoň třídy, které se budou ještě rozšiřovat o další potomky, nebo ty, které se mi podařilo zachránit. Komentáře časem doplním.

Část III

Závěr

Kapitola 10

Testování NEAT

Nejprve si popišme obrázek 10.1. Červenými body jsou označeny vstupní neurony, žlutými výstupní. Modrý je *bias* neuron, později se setkáme s bílými skrytými neurony. Na obrázku je neuronová síť ve výchozím stavu, což znamená, že v první iteraci je topologie všech sítí takováto.

Implementace algoritmu NEAT se testovala pomocí funkce XOR, kde $a, b, c \in [0, 1]$, $c = XOR(a, b) = (a - b)^2$. Fitness sítě f se počítá z jediného výstupu sítě y_f pro daná a, b, na vstupu sítě denormalizovaná mezi ± 1, jako

$$f = \sum |y_f - XOR(a, b)| \tag{10.1}$$

pro čtyři hodnoty parametrů $(a, b) \in \{(0, 0), (1, 0), (0, 1), (1, 1)\}$.

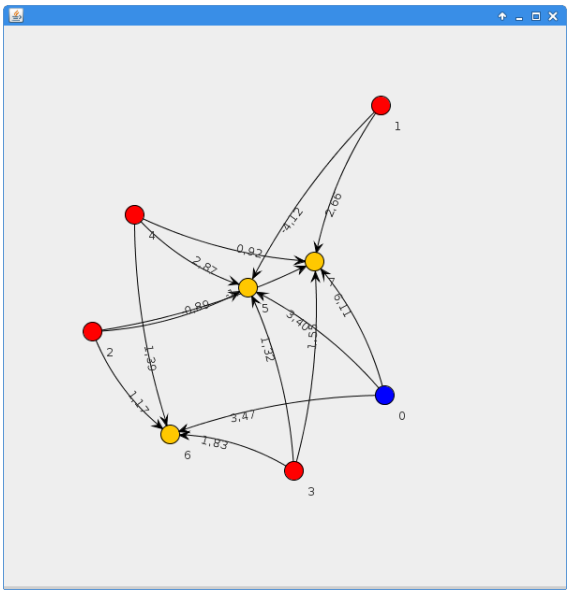
První pokusy s NEAT generovaly sítě, které z principu nemohly separovat funkci XOR, protože neměly dostatek spojů, ale jejich fitness ukazovala správné řešení. Nakonec se ukázalo, že to zapříčinila nikoliv chyba ve výpočtu fitness ani v evaluaci, ale to, že mezi jednotlivými evaluacemi nebyla síť resetována, což vedlo k velmi rychlému nalezení řešení právě pomocí vnitřní paměti.

Nalezené sítě jsou na obrázcích 10.2 a 10.3. Na prvním obrázku vidíme téměř optimální strukturu. Druhý obrázek ukazuje síť, která se vyvinula za příliš vysokých hodnot pravděpodobnosti přidání neuronu a spoje.

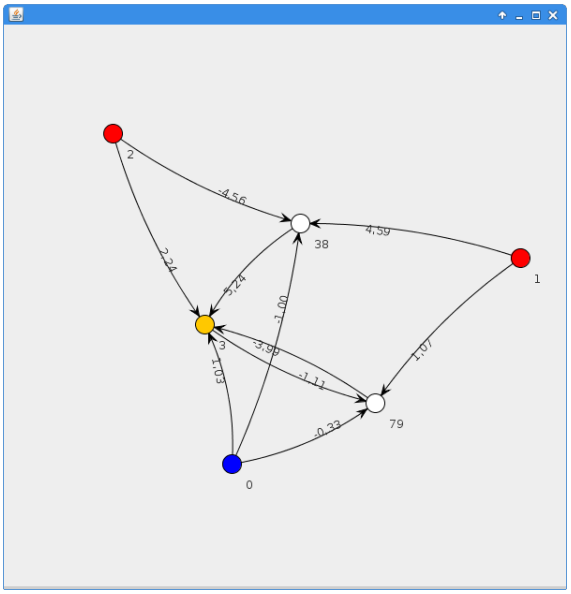
Co se parametrů týče, kromě prahu jsem parametry osvědčené při optimalizování XOR používal i na všechny ostatní pokusy (tabulka 10.1). Práh byl vůbec jedním z nejcitlivějších parametrů, a to ze dvou důvodů. Za prvé, čím větší je výchozí síť, tím větší potřebujeme práh, protože hned na začátku máme více vah. Druhý důvod se týká rychlosti konvergence, protože vyšší práh zapříčiní nepravděpodobné vybřednutí z lokálních optim, protože se do nich dostane celá populace.

10.1 Výkon

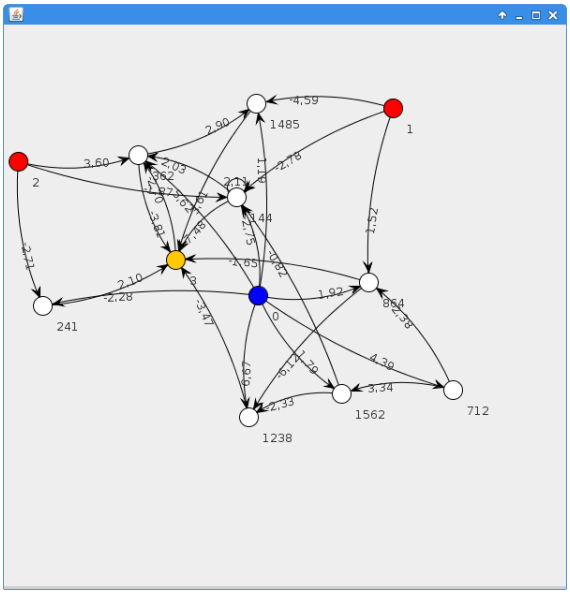
Výkon algoritmu NEAT nás nemusí příliš zajímat, protože oproti vyhodnocování je čas strávený nad jeho operacemi zanedbatelný; i to je důvod, proč se implementace



Obrázek 10.1: Neuronová síť výchozí



Obrázek 10.2: Neuronová síť XOR



Obrázek 10.3: Neuronová síť XOR nepovedená

úmrtnost	60%
práh	10-60
pravděpodobnost mutace váhy	0,2
pravděpodobnost přidání neuronu	0,2
pravděpodobnost přidání spoje	0,2
rozptyl nového prahu	3
rozptyl nového spojení	3
rozptyl změny váhy	0,3

Tabulka 10.1: Nastavení parametrů NEAT

zaměřuje více na přehlednost a stručnost než na výkon. Přesto proveďme alespoň základní pozorování. Nejvíce strojového času NEAT spotřebuje na počítání rozdílu d_t. Tuto informaci máme z profileru NetBeans z běhu pro XOR. Čím více se v algoritmu objeví druhů, tím více je porovnání, což vychází z algoritmu 2; v případě n druhů se provádí nanejvýš n porovnání, abychom si mohli být jisti, že síť nepatří do žádného druhu.

Typický běh algoritmu probíhá následovně. Pokud je k dispozici nová správná topologie, algoritmus téměř každou další iteraci najde lepší nastavení vah. Není-li, pak se několik iterací lepší síť neobjeví, a sítě začnou růst. To způsobovalo potíže při ladění algoritmu, než byl implementován test XOR a fitness nebyla vázána na výstupy, ale byla například náhodná. Vznikaly různé architektury, které, nejsou-li sofistikovaně ořezávány, rozpadnou se do druhů o jednom jedinci, kde si v populaci dva nejsou podobní. Ze stejného důvodu jsou ve výsledcích XOR všechny hrany na obě strany - není totiž důvod, proč by se měly prořezat, když se síť evalvuje jen v jednom kroku. Po dosažení vhodné topologie se série zlepšujících sítí opět obnoví.

Vyhodnocování je zpomalováno síťovou komunikací natolik, že se vyplatí spustit mnoho instancí JSBSimu a v programu vyvolat stejný počet vláken, aby se tato nevýhoda vyrušila. Vyhodnocení 20 sekund letu trvá 38 sekund (pro iterace JSBSim 60 Hz a frekvencí výpočtů 12 Hz), když běží jediné vlákno, a 44, když jich běží 128. Nároky na paměť to přitom téměř nezvedne - JSBSim vyžaduje velikost v paměti okolo 1,3 MB a program s více vlákny se od jedno-vláknového liší o 20 MB. Spotřeba paměti javovského programu se pohybuje okolo 330 MB.

Kapitola 11

Dosažené výsledky

Experimentů byla řada a vzhledem k jejich počtu parametrů, délce a nízké úspěšnosti jsem se nedostal k ničemu složitějšímu, než k přímému letu a zatáčce; na druhou stranu jsem v těchto disciplínách postupně dospěl k uspokojivým výsledkům.

11.1 Parametry

Na všechny pokusy byla frekvence iterací JSBSim nastavena na 60 Hz. První zkoušky probíhaly s frekvencí výpočtů sítě jedna ku deseti ku frekvenci iterací JSBSim, to se ale ukázalo jako nedostatečné, protože síť se naučila střídat krajní výchylky a docházelo ke kymácení letadla. Vyvodil jsem z toho též, že při maximálních výchylkách nedokáže síť letadlo stabilizovat z důvodu potřeby velmi přesně odměřené reakce, které by musela spočítat, aby letadlo v příští iteraci v zůstalo v klidu. Zvýšil jsem tedy frekvenci na dvojnásobek (12 Hz). Tím zároveň téměř dvakrát vzrostla doba potřebná k simulaci.

Při většině pokusů měla síť pouze minimální množství vstupů a výstupů. Při některých jsem přidal zrychlení, nicméně na výsledku to při stejném počtu generací nebylo znát.

Jako jeden z nejdůležitějších parametrů se ukázala délka simulace. Jestliže byla simulace krátká, síť se nemohla mnoho naučit, protože se setkala jen s omezeným množstvím případů. Při vizualizované simulaci se zapnutou turbulencí taková síť reagovala nevhodně.

Problémy nastaly též s inicializací. Přesto, že v ní nastavujeme v podstatě jen orientaci letadla, je potřeba najít relativně rovnovážnou pozici, aby se letadlo na začátku nepropadlo, nevyletělo vzhůru nebo nevybočilo. Všechny tyto jevy způsobili, že se letadlo dostalo hned na začátku mimo zamýšlenou dráhu, čímž dostaly v evoluci přednost sítě, které reagovaly na začátku přehnaně a zbytek už nějak „dohoupaly“. Toto jsem si ovšem uvědomil relativně pozdě, takže mnoho výpočetního času padlo na stabilizaci prvních dvou vteřin a zbytek neměl valnou hodnotu.

Zajímavou zkouškou bylo vypnutí motoru letadla. Letadlo bez motoru v přímém letu ztrácí rychlost, a pokud chce udržovat výšku, musí pilot „přitahovat“ - zvyšovat

výchylku horizontálních ocasních řídících ploch směrem nahoru. Zároveň drží křidélky náklon a směrovkou směr. Místo tohoto manévru síť létala nahoru, dolů a doleva, doprava „esíčka," po kterých měl následovat ztráta vztlaku a pád do vývrtky, který ale překvapivě stabilizovala a takto padala až na zem. Tyto pokusy už jsem prováděl na dvacetisekundové simulaci.

Sítě se vyvíjely do relativně velkých celků. Po sto iteracích NEATu měly, pokud to bylo vynuceno fitness, okolo 40 spojů a 10 skrytých neuronů. u některých pokusů se stalo, že fitness rychle spadla na hodnotu pravděpodobně blízkou optimu a získala si velkou část populace. Taková síť se pak už zpravidla nerozvíjela.

11.2 Zhodnocení a další vývoj

Kdybych tušil, kolik problémů přináší vlastní implementace NEAT, především kvůli nedeterminismu implikující mimořádně špatnou odladitelnost, použil bych některou z dostupných knihoven. Předchozí semestrální práce na toto téma používala knihovnu Encog, která umožňovala jednoduše pracovat s feed-forward neuronovými sítěmi, nicméně pro NEAT volila jinou architekturu programu, což mě odradilo. Mohl bych se tak více věnovat pokusům, na druhou stranu mám teď možnost zcela ovlivňovat výkon algoritmu.

Experimenty ukázaly, že sítě učené algoritmem NEAT mají potenciál k nalezení řešení složitějších úloh v 6DOF prostoru, zvolíme-li správně parametry. Volba parametrů je ovšem vzhledem k délce běhu algoritmu loterií, proto by bylo vhodné zvýšit výkon NEATu, aby nebylo potřeba tolik evaluací. Další možností jsou přístupy, které mění parametry podle aktuálního stavu algoritmu.

Jak jsme předestřeli v kapitole 3, existuje celá řada manévrů, které lze nadefinovat a implementovat. To jsem umožnil, nyní stačí implementovat nové třídy rozšiřující třídu FlightManeuver.

Dále jsem vyvinul funkční rozhraní schopné paralelizovat výpočet, napsal skripty pro snadné spouštění a pomocí algoritmu sestrojil několik sítí pro přímý let a zatáčku.

Konečným cílem tohoto přístupu je, jak již bylo řečeno v úvodu, ukazovat na trať a nechat síť vypořádat se s jakoukoli křivkou, kterou letový model dovoluje

Literatura

[1] Markus Möckli; Aerobatic - Guidance and Control for Aerobatic Maneuvers of an Unmanned Airplane; http://www.uav.ethz.ch/research/projects/Aerobatic; 2006.

[2] Burgin, G.H.; Schnetzler, S.S.; "Artificial neural networks in flight control and flight management systems," Aerospace and Electronics Conference; 1990. NAECON 1990., Proceedings of the IEEE 1990 National , vol., no., pp.567,573 vol.2; 21-25. Březen 1990

[3] Stanley, K., Miikkulainen R.; "Evolving Neural Networks through Augmenting Topologies," Evolutionary Computation , vol.10, no.2, pp.99,127; Červen 2002; ISSN 1063-6560.

[4] Jon S. Berndt; JSBSimFlyer; http://jsbsim.sourceforge.net/JSBSimFlyer.pdf; 2006.

[5] Novák, P.; Návrh řídicího algoritmu pro stabilizaci letadla. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 95 s. Vedoucí diplomové práce doc. Ing. Jiří Krejsa, Ph.D.; 2013.

[6] Xin Yao; "Evolving artificial neural networks," Proceedings of the IEEE , vol.87, no.9, pp.1423,1447; září 1999.

[7] ICAO předpisy, Předpis L2; 15.12.2014; http://lis.rlp.cz/predpisy/predpisy/index.htm.

[8] Jon S. Berndt a kolektiv; JSBSim An open source, platform-independent, flight dynamics model in C++; 6.9.2011.

[9] Andrew G. Barto; Reinforcement Learning: An Introduction, Adaptive computation and machine learning , MIT Press; 1998; ISBN 0262193981, 9780262193986

[10] Faustino J. Gomez, Risto Miikkulainen; Active Guidance for a Finless Rocket using Neuroevolution; 2003

[11] John J. Blakelock; Aircraft & Missile; leden 1991; ISBN: 0471506516

[12] Jan Drchal; Evolution of Recurrent Neural Networks - diplomová práce, 80 s.; 2006.

Část IV

Přílohy

1 Seznam použitých zkratek

FDM flight dynamic model, model letové dynamiky

NED north-east-down, souřadnicový systém

GPS global positioning system, družicový polohový systém

NEAT neuroevolution by augmenting topologies, učící algoritmus

ANN artificial neural network, počítačová neuronová síť

GIN global inovation number, inovační číslo v algoritmu NEAT

ECI earth centered initial, souřadnicový sytém úhlů změpisné délky a šířky

2 Ukázka sítě

Síť zapsaná ve standardu graphwiz .dot; přepsané neaktivní hrany jsou označeny „—“:

```
digraph g { 0 -> 5 [label="0 -1,94"]
--- 0 -> 6 [label="1 -5,69"]
--- 0 -> 7 [label="2 0,03"]
--- 1 -> 5 [label="3 -5,54"]
--- 1 -> 6 [label="4 5,56"]
1 -> 7 [label="5 -1,28"]
2 -> 5 [label="6 -0,27"]
2 -> 6 [label="7 6,74"]
--- 2 -> 7 [label="8 2,49"]
3 -> 5 [label="9 -3,01"]
--- 3 -> 6 [label="10 6,65"]
3 -> 7 [label="11 1,12"]
4 -> 5 [label="12 -3,27"]
4 -> 6 [label="13 3,10"]
4 -> 7 [label="14 -4,25"]
1 -> 121 [label="355 7,51"]
--- 121 -> 6 [label="356 2,86"]
0 -> 121 [label="357 -2,60"]
1 -> 398 [label="1199 1,99"]
398 -> 5 [label="1200 -0,53"]
0 -> 398 [label="1201 -1,26"]
2 -> 932 [label="2861 -1,38"]
932 -> 7 [label="2862 -0,54"]
--- 0 -> 932 [label="2863 0,64"]
--- 7 -> 932 [label="3359 4,61"]
3 -> 398 [label="2312 5,41"]
0 -> 1342 [label="4153 -0,39"]
1342 -> 932 [label="4154 3,56"]
121 -> 1342 [label="4155 5,40"]
0 -> 1375 [label="4257 0,89"]
```

```
1375 -> 6 [label="4258 -2,92"]
0 -> 1641 [label="5090 0,13"]
1641 -> 932 [label="5091 0,20"]
121 -> 1722 [label="5337 -2,62"]
1722 -> 6 [label="5338 -0,38"]
0 -> 1722 [label="5339 3,18"]
3 -> 1815 [label="5620 -1,42"]
1815 -> 6 [label="5621 1,84"]
0 -> 1815 [label="5622 2,34"]
1375 -> 1815 [label="5976 2,37"]
--- 3 -> 1375 [label="5888 -6,11"]
1815 -> 932 [label="6808 -0,06"]
1375 -> 1722 [label="5883 -0,21"]
7 -> 2558 [label="7980 -0,29"]
2558 -> 932 [label="7981 1,97"]
0 -> 2558 [label="7982 1,95"]
3 -> 2669 [label="8332 -2,13"]
2669 -> 1375 [label="8333 0,12"]
0 -> 2669 [label="8334 2,13"]
1641 -> 2669 [label="8335 -0,16"]
3 -> 2766 [label="8666 2,29"]
2766 -> 6 [label="8667 -2,33"]
0 -> 2766 [label="8668 -3,16"]
0 -> 2932 [label="9190 -1,65"]
2932 -> 7 [label="9191 -4,10"]
7 -> 1815 [label="7847 2,88"]
}
```

3 Skripty

Spouštění NEAT:

```
#!/bin/bash
# spouští učení, parametr $1 urcuje pocet paralelnich behu
cd jsbsim/scripts/
defalut=8
# výběr letadla
generateInitsc172.sh ${1-default} 8749
#generateInitsCap10B.sh ${1-default} 8749
#generateInitsp51b.sh ${1-default} 8749
cd ..
defalut=8
for i in $(seq 1 ${1-default}) do
echo -n "Start JSBSim"
echo $i ./src/JSBSim --suspend --script=scripts/cap"$i".xml \
--property="propulsion/engine/set-running=1" \
--property="fcs/throttle-cmd-norm=1" \
--property="fcs/mixture-cmd-norm=1" \
--property="fcs/advance-cmd-norm=1" \
&> JSBsimOutput.txt & # potlaceni vystupu
sleep 0.1 done
cd ..
sleep 0.5 java -jar VirtualPilot/dist/VirtualPilot.jar \
--threads=$1 --simTime=20.0
killall JSBSim &> JSBsimOutput.txt
```

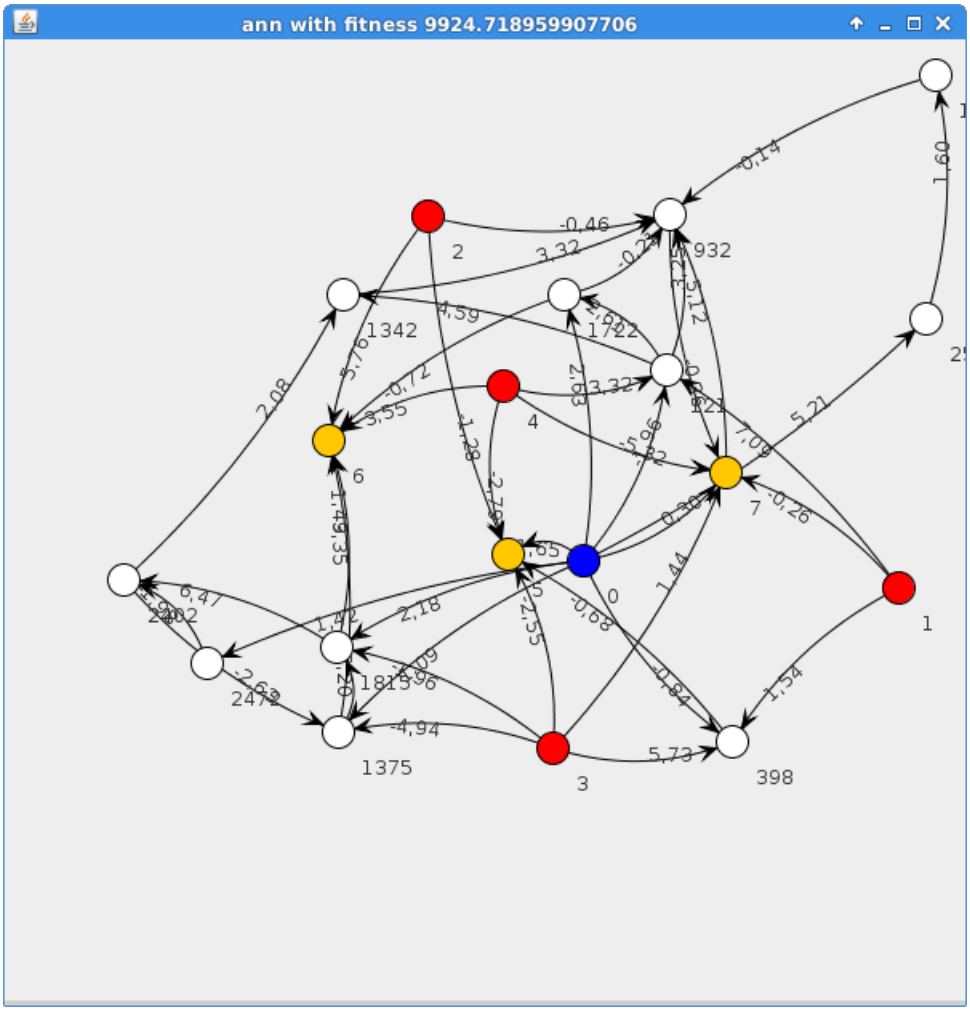
Spouštění XOR NEAT:

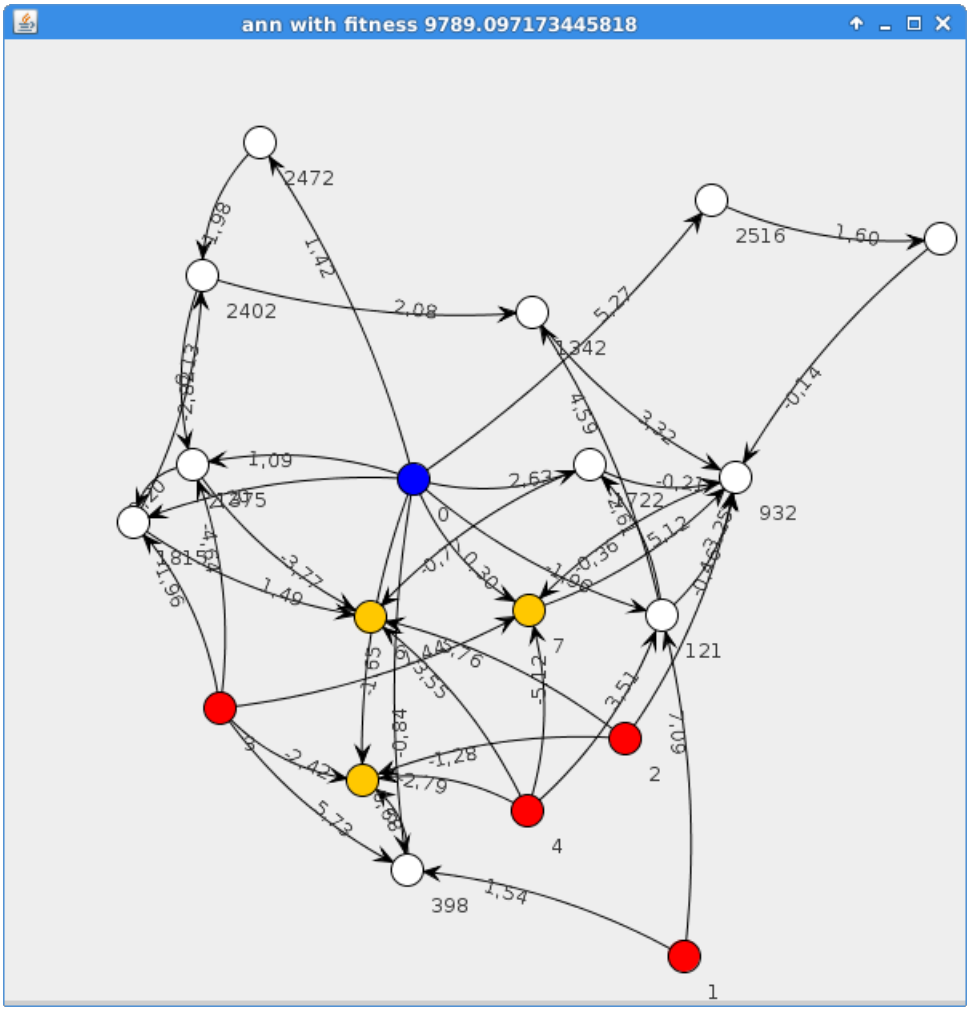
```
#!/bin/bash
#spusti vyvoj XOR; neni potreba prenastavovat parametry,
#nastavi se automaticky
java -jar VirtualPilot/dist/VirtualPilot.jar --threads=0\
--population=200 --maneuver=flightManeuver.XOR
```

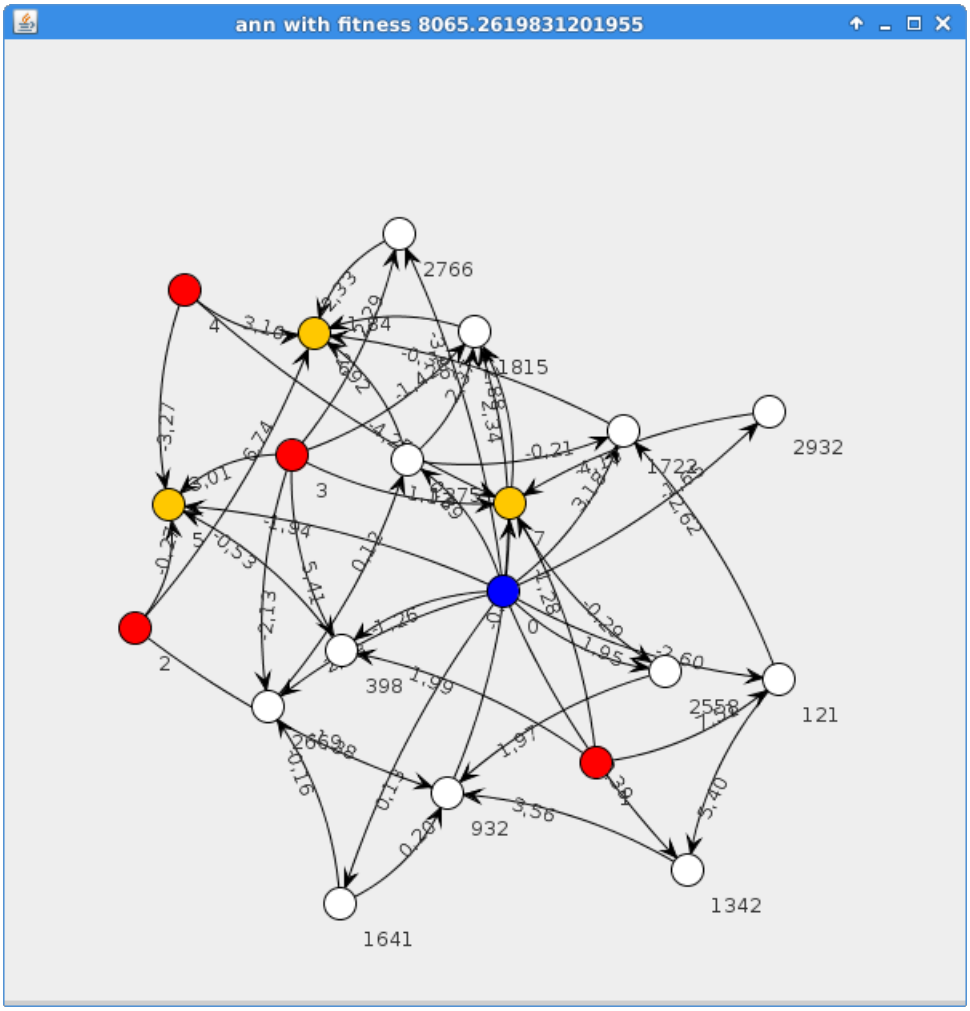
Spouštěné ukázky sítě:

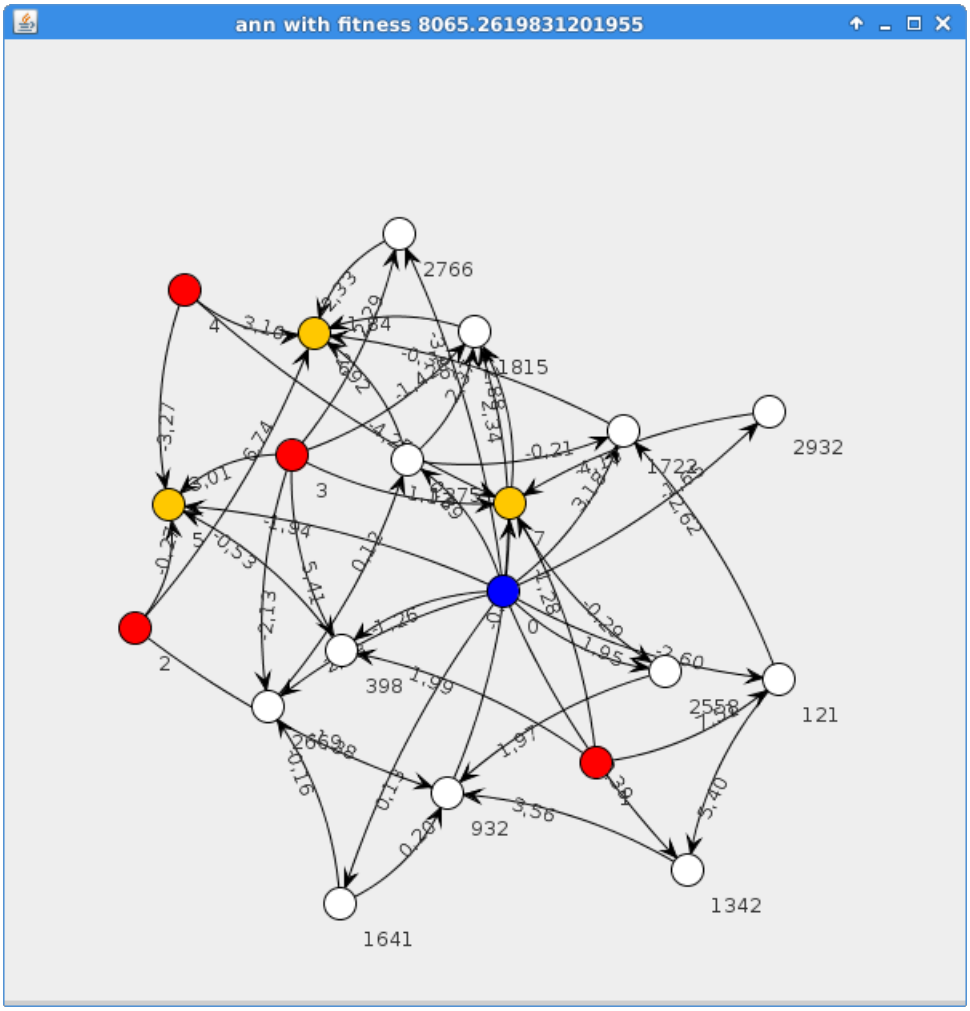
```
#!/bin/bash # spouští ukázku sítě
# --aircraft=Cap10B \
fgfs \
--timeofday=dawn \
--native-fdm=socket,in,60,,5500,tcp \
--fdm=external \
&
sleep 40
cd jsbsim/ ./src/JSBSim --suspend --script=scripts/c172pShow.xml\
&
sleep 0.5
cd .. sleep 1 java -jar VirtualPilot/dist/VirtualPilot.jar --loadANNfrom
killall JSBSim fgfs
```

4 Výstup programu - evoluce přímého letu




ann with fitness 9789.097173445818
2472
2516
2402
1342
932
121
7
2
4
398
1


ann with fitness 8065.2619831201955
2766
1815
2932
1722
3
4
2
398
2558
121
932
1342
1641


ann with fitness 8065.2619831201955
2766
1815
2932
1722
3
4
2
398
2558
121
932
1342
1641