UNIVERZITA KOMENSKÉHO V BRATISLAVE FAKULTA MATEMATICKY, FYZIKY A INFORMATIKY

Kód práce: 80c4d0b6-2a7c-4afd-9903-b6d5b12b9183

Robotour

Diplomová práca

Študijný program : Aplikovaná informatika

Študijný odbor: 9.2.9 APLIKOVANÁ INFORMATIKA

Školiace pracovisko: KATEDRA APLIKOVANEJ INFORMATIKY

Školiteľ: Mgr. Pavel Petrovič, PhD.

Bratislava, 2011

Bc. Miroslav Nadhajský

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Miroslav Nadhajský
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov: Robotour

Cieľ: Vypracovať návrh outdoorového mobilného robota pre riešenie úlohy súťaže RoboTour. Zostrojenie robota, návrh a implementácia algoritmov, overenie v praxi. Riadiaci algoritmus robota by mal byť využívať algoritmy s umelou inteligenciou.

Vedúci: Mgr. Pavel Petrovič, PhD.

Dátum zadania: 13.11.2009

Dátum schválenia: 03.05.2011

za
prof. RNDr. Ivan Kalaš, CSc.
garant študijného programu

student

vedúci

Čestné prehlásenie

Čestne prehlasujem, že som diplomovú prácu vypracoval samostatne s použitím uvedenej literatúry.

_____

Poďakovanie

Ďakujem vedúcemu diplomovej práce Mgr. Pavel Petrovičovi, PhD. za cenné rady, konzultácie a spoluprácu. Ďakujem tiež svojej rodine a priateľom za psychickú podporu. Táto práca vznikla s podporou grantu Nadácie Tatra banky „Podpora kvality vzdelávania na vysokých školách“, a v spoločnom laboratóriu mobilnej robotiky FEI STU, FMFI UK a Microstep-MIS (http://www.robotika.sk/events/05December/opening.php).

Abstrakt

V tejto práci sa zaoberáme návrhom a realizáciou autonómneho robota pre robotickú súťaž Robotour. Dôraz kladieme na rozpoznávanie ciest z kamery. Rozhodli sme sa na tento účel použiť neurónové siete. Použili sme nový spôsob riešenia, ktorý v tejto súťaži ešte nebol použitý. Zistili sme že tento spôsob je efektívne použiteľný. Výsledkom práce je dokončený robot pripravený na ďalší ročník súťaže.

Kľúčové slová

autonómny robot, rozpoznávanie cesty z obrazu, neurónová sieť

Abstract

In this thesis we cover design and realization of autonomous robot for the robotic contest Robotour. We are focusing on a vision based recognition. We decided to use neural networks for this purpose. This is a solution that has not been used in Robotour contest previously. In conclusion, proposed solution can be effectively used. The result is complete robot, prepared for next year contest.

Keywords

autonomous robot, vision-based recognition, neural network

Obsah

1 Časť

1.1 Úvod

V súčasnosti sa častejšie objavujú pokusy o autonómne riadené vozidlá. Najznámejšou súťažou v tejto oblasti DARPA Grand Challenge. Túto inšpiráciu prebralo aj české združenie, ktoré poriada podobnú súťaž Robotour (viac v kapitole 1.6), avšak vo zmenšenej podobe. Ide o autonómne riadené mobilné roboty, ktoré majú za úlohu dopraviť náklad na určité miesto v parku. Pohyb je možný len po cestách, čo vieme zabezpečiť rozpoznávaním ciest z kamery. Tento problém sa dá riešiť rôznymi spôsobmi, avšak použitie rozpoznávania z kamery je najdostupnejšie riešenie. Ďalej budeme riešiť navigáciu obchádzanie / rozpoznávanie prekážok a koordináciu jednotlivých pod-problémov.

Najskôr v prvej časti rozoberieme súvisiace oblasti s danou tematikou, tak ako aj pravidlá súťaže Robotour.

V druhej časti navrhneme celý systém robota. Najviac sa budeme zaoberať rozpoznávaním cesty z obrazu na ktoré použijeme vlastný postup ktorý pozostáva z neurónových sietí. Ostatné pod-problémy rozoberieme hneď za tým.

Následne budeme systém realizovať, postavíme robota, naimplementujeme softvér. Na záver zhrnieme čo sme dosiahli, zistili a čo by sa dalo v budúcnosti vylepšiť.

1.2 Robotika

Na konci druhej svetovej vojny začali prvé pokusy s nukleárnymi zbraňami. Pri ich výrobe však radiácia neumožňovala vedcom a inžinierom priamo manipulovať s vysoko rádioaktívnym materiálom, kvôli zdravotnému riziku. Pre tento účel boli vytvorené telemanipulátory. Mechanické zariadenie ktoré umožňovalo na diaľku uchytiť a manipulovať s objektom.

Neskôr sa z týchto telemanipulátorov vyvinuli priemyselné manipulátory - **robotické ramená**. Tieto zariadenia sú väčšinou programovateľné a multifunkčné, určené na premiestňovanie objektov, častí, nástrojov alebo špeciálnych zariadení (napríklad na zváranie, farbenie, …). Sú navrhnuté hlavne na opakujúce sa činnosti, napríklad pre špecifickú pozíciu v montážnej linke, kde vedia vykonávať tieto činnosti s vysokou presnosťou a rýchlosťou.

Ďalšia kategória robotov, ktorá bola vyvinutá sú mobilné roboty. Patria sem autonómne navigované vozidlá (AGV, autonomous guided vehicle), ktoré plnia úlohy spojené s prepravou. Ďalej sa mobilné roboty používali napríklad pre NASA na skúmanie Marsu. Neskôr prišli aj bezpilotné lietadlá (UAV, unmanned aerial vehicle), autonómne ponorky (AUV, autonomous underwater vehicle) a iné typy robotov.

Robotické systémy môžu mať rôzny stupeň autonómnosti:

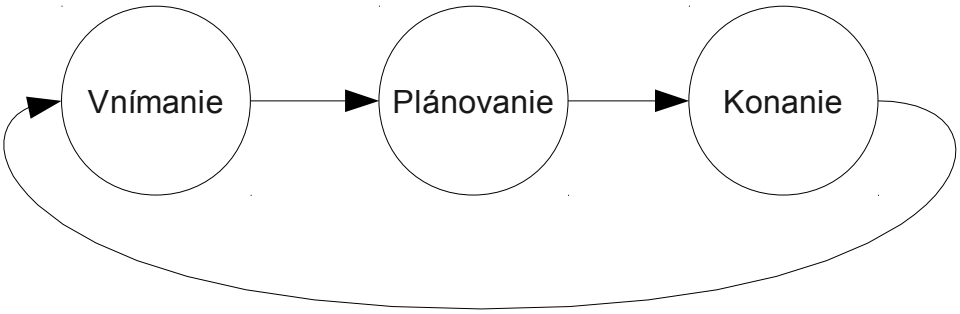
1. Operátor ovláda robota na diaľku (teleoperation)
2. Polo-autonómne, určitú časť úloh riadi operátor a iná časť úloh je vykonávaná autonómne. Tu sa dajú rozlišovať dva spôsoby, prvý keď operátor prevezme kontrolu v kritických situáciách. Druhá možnosť je keď operátor iniciuje úlohu, ktorú robot sám dokončí
3. Robot vykonáva úlohy bez zásahu operátora

„Inteligentný robot je mechanický tvor, ktorý dokáž fungovať autonómne.“ Budeme sa ďalej orientovať hlavne na autonómne/inteligentné roboty. Čo je to umelá inteligencia podrobne rozoberieme v ďalšej kapitole.

Existujú stri hlavné spôsoby/paradigmy ako organizovať inteligenciu v robotoch.

Hierarchická paradigma

Je najstaršou metódou organizovania inteligencie, ktorú priniesol prvý autonómny mobilný robot s umelou inteligenciou, Shakey v roku 1967.



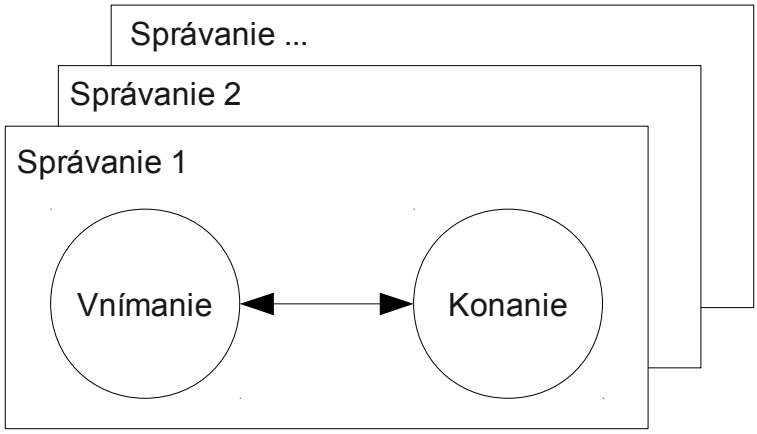
Obrázok 1: Diagram hierarchickej paradigmy

Najskôr robot vníma okolie a vytvorí si mapu prostredia. Potom na základe mapy prostredia plánuje akcie na dosiahnutie cieľa. Nakoniec robot vykoná naplánovanú akciu.

Najväčší problém je náročnosť plánovania v každom cykle. Ale je tiež náročné vytváranie globálneho modelu prostredia.

Reaktívna paradigma

Pôvodne bolo toto správanie pozorované na zvieratách, keď napríklad zviera pri zhliadnutí predátora okamžite zmenilo správanie. Správanie, ktoré je naučené a zároveň sa vykonáva bez potreby vedomého rozmýšľania sa nazýva reaktívne.



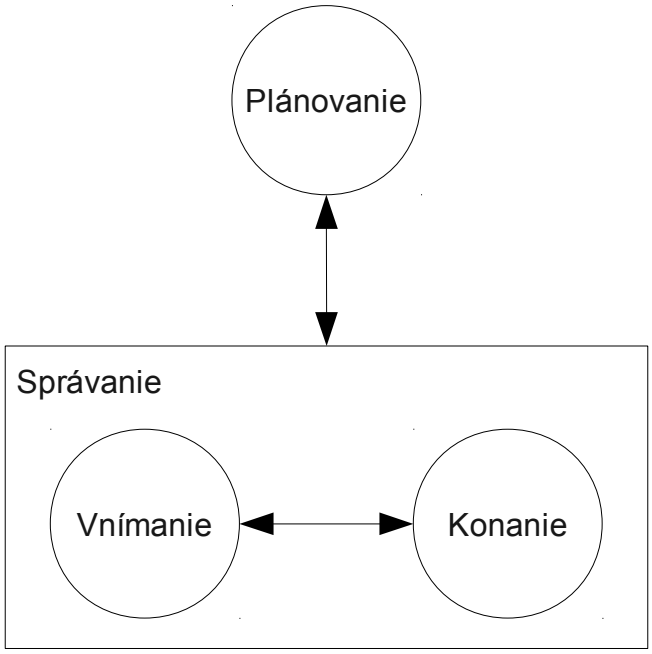
Obrázok 2: Diagram reaktívnej paradigmy

Pri reaktívnej paradigme systém pozostáva z jednotlivých správaní, čo sú funkcie, ktoré transformujú vstupy senzorov na ovládanie aktuátorov.

Hlavné výhody sú: rýchlosť a dekompozícia na jednotlivé správania, čo zlepšuje prehľadnosť systému. Pri porovnaní s hierarchickou paradigmou vidíme, že chýba plánovanie.

Hybridná plánovacia(deliberative) / reaktívna paradigma

V reaktívnej paradigme robot nemal žiadnu globálnu informáciu o prostredí. Bez tejto informácie sa však nedá v mnohých prípadoch efektívne plniť cieľ. Hybridná paradigma spája výhody reaktívnej, ale obsahuje aj plánovanie nad modelom prostredia. Najskôr sa vykoná plánovanie a na základe neho sa vygeneruje množina správaní, ktoré sa budú vykonávať, až kým sa nesplní plán. A potom sa znova opakuje plánovanie.



Obrázok 3: Diagram Hybridnej plánovacej(deliberative) / reaktívnej paradigmy

V hybridnej paradigme sa môže vyskytovať reflexívne, vrodené a naučené správanie na rozdiel od reaktívnej paradigmy, kde sú len reflexívne správania.

V tejto kapitole sme sa dozvedeli niečo z histórie robotov a tiež čo je autonómny robot a akými spôsobmi sa dá organizovať inteligencia v robotickom systéme. Informácie som čerpal hlavne z knihy Introduction to AI Robotics [Murphy, 2000].

1.3 Umelá inteligencia

Čo je to umelá inteligencia? Na celkom rozsiahle vysvetlenie som narazil v knihe Artificial Inteligence A Modern Approach [Russel, 1995], kde sa spomínajú tieto 4 vlastnosti:

Správanie podobné človeku

V roku 1950 predstavil Alan Turing, test na definíciu inteligencie (Turingov test). Počítač v tomto teste komunikuje s človekom textovými správami cez klávesnicu a obrazovku. Rozhodca, ktorý sleduje komunikáciu má rozhodnúť, kto je v konverzácii počítač a kto človek. Ak to rozhodca nevie spoľahlivo rozhodnúť, počítač prejde týmto testom.

Na úspešné zvládnutie toho testu počítač potrebuje:

- spracovanie prirodzeného jazyka
- reprezentáciu znalostí
- automatizované odvodzovanie
- strojové učenie

Rozmýšľanie podobné človeku

Najskôr je potrebné zistiť ako ľudia rozmýšľajú a aké procesy prebiehajú v ľudskej mysli, ktoré sa následne dajú napodobňovať. Tomuto prístupu sa venuje kognitívna veda, čo je interdisciplinárna veda, ktorá zahŕňa: psychológiu, filozofiu, umelú inteligenciu, lingvistiku, neurovedu a antropológiu.

Do kognitívnej vedy patrí aj oblasť konekcionizmu, ktorá sa zaoberá najmä neurónovými sieťami(viac v kapitole o neurónových sieťach).

Racionálne rozmýšľanie

Gréckeho filozofa Aristotela, považujeme za prvého, kto skúmal formálnu logiku. Zaviedol syglogizmy, ktorými sa dalo argumentovať „správne“. Zaviedol dedukciu, ktorá mala predpoklady a dôsledky. Pomocou sylogizmov vedel argumentovať, tak že to viedlo vždy ku správnym dôsledkom. Z tohto sa neskôr vyvinula logika.

Prináša to však so sebou aj problémy. Je ťažké formulovať neformálnu znalosť pomocou formálnych logických výrazov, a najmä ak nemáme 100%-nú istotu tejto znalosti. V praxi môžu byť zjavne jednoduché problémy veľmi výpočtovo náročné.

Racionálne správanie

Od počítačových agentov očakávame aby sa správali autonómne, prispôsobovali zmenám, vnímali prostredie a hlavne aby plnili nejaké ciele. Agent sa správa racionálne ak dosahuje najlepšie výsledky alebo aspoň očakávané.

Racionálne rozmýšľanie môže byť časťou racionálneho správania, aj keď sú situácie keď požadujeme od agenta vykonanie akcie, aj keď na to nie je logický dôvod. Taktiež všetky schopnosti, ktoré treba na zvládnutie Turingovho testu znamená pre agenta správať sa racionálne. Teda tento prístup zahŕňa všetky prechádzajúce, s tým že je viac všeobecný.

1.3.1 Oblasti

Umelá inteligencia sa dá rozdeliť na oblasti:

- Riešenie problémov pomocou prehľadávania stavového priestoru
- Usudzovanie na základe bázy znalostí
- Plánovanie
- Usudzovanie s neurčitosťou
- Učenie
- Komunikácia

1.3.2 Učenie

Na pôsobenie agenta v prostredí, nie je vždy najlepšie naprogramovať konkrétne akcie. Ak nie je úplne známe prostredie, učenie je jediný spôsob, ako agent zistí to čo potrebuje.

Podľa typu spätnej väzby rozlišujeme 3 druhy učenia:

1. učenie s učiteľom
2. učenie bez učiteľa
3. učenie s odmenou a trestom

Učenie s učiteľom

Pri tomto type je cieľom naučiť sa mapovanie zo vstupu konkrétny výsledok. Čiže vstupný vektor má korešpondujúci cieľový vektor. Patrí sem klasifikácia, kde je výsledkom konečný počet

diskrétnych kategórií. Ak sa výstup skladá aspoň z jednej alebo viac spojitých premenných je to regresia.

Učenie bez učiteľa

V tomto prípade máme len vstupné dáta a učenie je zamerané na hľadanie vzťahov v priestore vstupov. Cieľom je nájsť podobnosti vo vstupe. Niektoré vzorky sa vyskytujú častejšie ako iné. V štatistike sa tomu hovorí odhadovanie hustoty(density estimation). Jeden s prístupov pre odhadovanie hustoty je zhlukovanie (clustering), kde je cieľom hľadať zhluky alebo zoskupovať vstupy.

Učenie s odmenou a trestom

Niekedy je v systéme podstatná sekvencia akcií, nie len jedna samotná. Vieme povedať len či daná sekvencia akcií (taktika) je správna. V tomto prípade by sa mal systém učiť z predchádzajúcich taktík a vedieť generovať nové.

Na rozdiel od učenia s učiteľom, učiaci algoritmus nedostáva optimálny výstup, ale ho musí objaviť sám, pomocou metódy pokus-omyl.

1.4 Neurónové siete

Pod pojmom neurónové siete v celej tejto práci máme na mysli umelé neurónové siete(artificial neural networks). Sú matematickým modelom ktorý sa snaží dosiahnuť vlastnosti biologických neurónových sietí.

Neurónové siete sú zložené z malých výpočtových jednotiek ktoré sú medzi sebou poprepájané. Sú schopné sa učiť a riešiť rôzne zložité úlohy ako aproximácia matematickej funkcie, klasifikácia dát, zhlukovanie dát (clustering) a predpovedanie časových radov.

História

Prvé umelé neurónové siete predstavili McCulloch a Pitts v roku 1943. Sieť sa skladala z binárnych neurónov, ktoré boli spájané orientovanými hranami buď excitačného alebo inhibičného typu. Ak aktivácia neurónu bola väčšia ako prah, potom neurón bol excitovaný.

„Ak je táto sieť nekonečne veľká, tak je výpočtovo ekvivalentná univerzálnemu

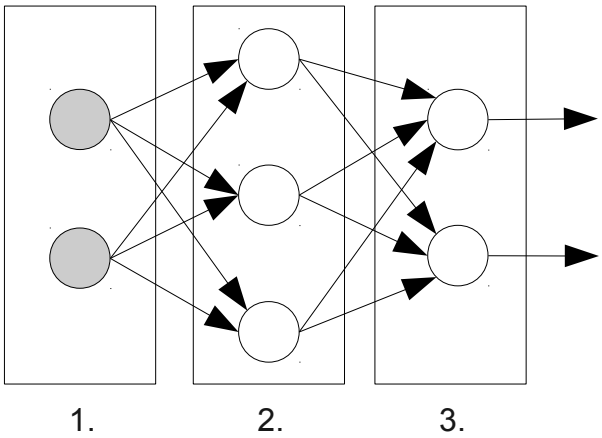
Turingovmu stroju “ [Kvasnička, 1997].

V roku 1958 Rosenblatt predstavil perceptrón, model ktorý mal modifikovateľné váhy. Pomocou nich sa dali rozpoznávať a klasifikovať vzory.

Tento model bol študovaný Minskym a Papertom, ktorý skúmali problémy týchto sietí, že nedokážu riešiť lineárne neseparovateľné problémy.

Tento problém bol vyriešený v roku 1986 (Rumelhart, Hinton a Williams) a bola navrhnutá efektívna metóda na učenie týchto sietí spätným šírením chýb (backpropagation).

Neuróny obsahujú nelineárnu aktivačnú funkciu, najčastejšie sigmoidálneho tvaru. Sieť je dopredná, číže orientovaný acyklický graf. A je rozdelená na vrstvy neurónov, pričom medzi dvoma vrstvami sú všetky neuróny spojené každý s každým, ale len jednosmerne (smerom zo vstupnej vrstvy na výstupnú). Takáto sieť sa nazýva viac-vrstvový perceptrón (MLP, multi-layer perceptron).



Obrázok 4: Príklad doprednej neurónovej siete s jednou skrytou vrstvou,1. vstupná vrstva, 2.skrytá vrstva, 3. výstupná vrstva

1.4.1 MLP a Backpropagation

Backpropagation je algoritmus na trénovanie dopredných sietí pre učenie s učiteľom. Je to prvorádový optimalizačný algoritmus (neberie do úvahy derivácie vyššieho rádu). Využíva metódu zostupovania gradientom, kde sa podľa smeru gradientu snažíme minimalizovať funkciu chyby siete[Haykin, 1999].

Chybu pre neurón vo výstupnej vrstve s indexom *j* pre trénovací príklad *n* budeme počítať ako:

$$e_j(n) = d_j(n) - y_j(n)$$

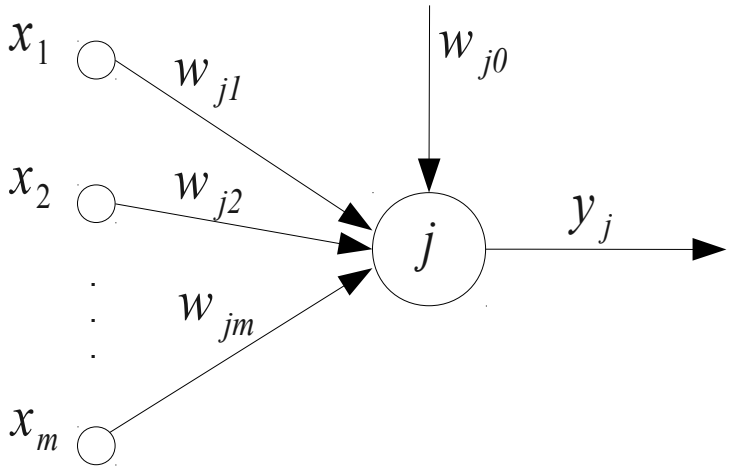
Ďalej definujeme kvadratickú chybu pre celú sieť, čiže suma chýb všetkých neurónov vo výstupnej vrstve:

$$E(n)=\frac{1}{2}\sum_j e_j^2(n)$$

Nakoniec strednú kvadratickú chybu ako priemer pre všetky trénovacie príklady:

$$MSE=\frac{1}{N}\sum_{n=1}^{N} E(n)$$

Funkcia *E* aj *MSE* sú funkciami všetkých voľných parametrov siete. Teraz je potrebné minimalizovať *MSE* pre danú množinu trénovacích príkladov.



Obrázok 5: Diagram jedného neurónu

Výpočet aktivácií neurónov

Vstup *v* do aktivačnej funkcie ϕ je suma váhovaných vstupov neurónu. Vstup $x_0 = -1$ a váha w_{j0} je prah.

$$v_j = \sum_{i=0}^{m} w_{ji}(n)\, x_i(n)$$

Aktivačná funkcia je najčastejšie sigmoid, táto funkcia prináša do neurónu nelineárnosť a musí byť spojitá a aj jej prvá derivácia:

$$\phi_j(v_j(n)) = \frac{1}{1+e^{-v_j(n)}}$$

Teda aktiváciu *y* neurónu s indexom *j* definujeme ako:

$$y_j(n) = \phi_j(v_j(n))$$

Korekcia chyby

Na korekciu chyby neurónu použijeme pravidlo, čím sa pohneme smerom bližšie ku minimálnej chybe. Čiže zmena váhy je:

$$\Delta w_{ji}(n) = -\mu \frac{\partial E(n)}{\partial w_{ji}(n)}$$

kde μ je rýchlosť učenia($0 < \mu \leq 1$). Po úprave dostaneme:

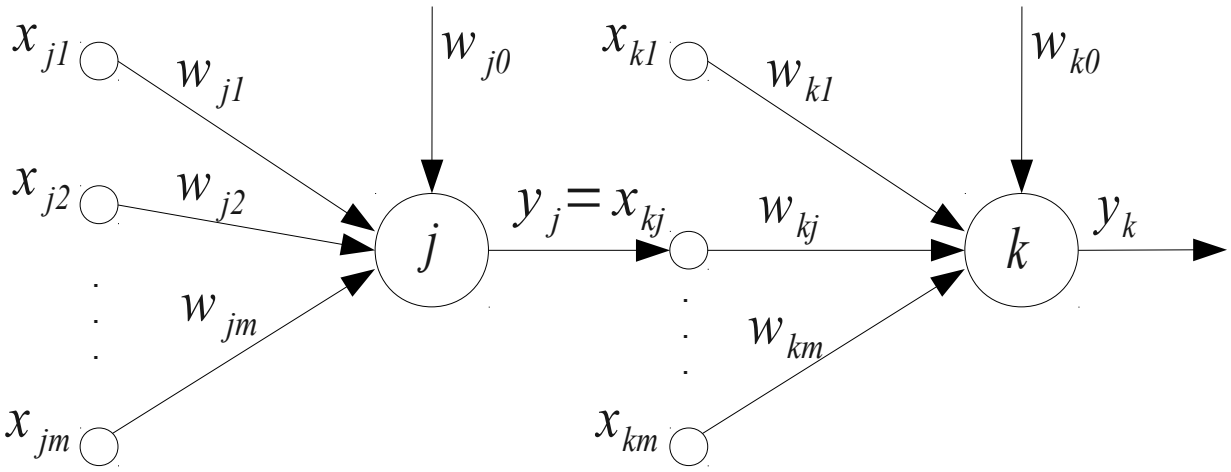
$$\Delta w_{ji}(n) = \mu\, \delta_j(n)\, y_i(n)$$

a lokálny gradient δ :

$$\delta_j(n) = e_j(n)\phi_j'(v_j(n))$$

kde ϕ' je derivácia aktivačnej funkcie. Chyba *e* je dostupná len pre výstupnú vrstvu.

Spätné šírenie chýb



Obrázok 6: Dva neuróny za sebou, výstup neurónu j je vstupom pre neurón k, ktorý sa nachádza v ďalšej vrstve

Ak sa neurón nachádza v skrytej vrstve musíme lokálny gradient vypočítať z predchádzajúcej vrstvy neurónov.

$$\delta_j(n) = \phi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$

Teda pre lokálny gradient pre neurón v skrytej vrstve s indexom *j* vstupuje váhovaná suma všetkých δ z predchádzajúcej vrstvy neurónov s ktorými je spojený.

Algoritmus

V prvej *doprednej* fáze vypočítame postupne po vrstvách aktivácie neurónov (smerom od vstupnej vrstvy). Nasledujúca vrstva dostáva ako vstup aktivácie z predchádzajúcej vrstvy.

V *spätnej* fáze začíname od výstupnej vrstvy a postupne po vrstvách počítame lokálne gradienty a tiež zmeny váh.

- Sekvenčný režim algoritmu: upravujeme váhy po každom trénovacom príklade.
- Dávkový režim algoritmu: upravujeme váhy až po vyhodnotení všetkých trénovacích príkladov, namiesto chyby $e_j(n)$ použijeme priemernú cez všetky trénovacie príklady.

1.4.2 Rprop

Rprop alebo resilient backpropagation vytvorili Riedmiller a Braun v roku 1993 [Riedmiller, 1993].

Je to rovnako ako Backpropagation optimalizačný algoritmus prvého rádu, ktorý slúži na trénovanie dopredných neurónových sietí pomocou metódy zostupovania gradientom.

V Rprop algoritme smer zmeny váhy závisí na znamienku parciálnej derivácie chyby podľa váhy $\partial E / \partial w_{ij}$. Veľkosť kroku však nezávisí od absolútnej hodnoty tejto derivácie. Tiež nie je treba nastavovať rýchlosť učenia, ako pri backpropagation, pretože algoritmus je adaptívny.

Každá váha w_{ij} má vlastnú veľkosť kroku Δ_{ij} , ktorý sa upravuje podľa nasledujúceho pravidla:

$$\Delta_{ij}(t)=\begin{cases}\eta^{+}\cdot\Delta_{ij}(t-1), & ak \ \frac{\partial E}{\partial w_{ij}}(t-1)\cdot\frac{\partial E}{\partial w_{ij}}(t)>0\\ \eta^{-}\cdot\Delta_{ij}(t-1), & ak \ \frac{\partial E}{\partial w_{ij}}(t-1)\cdot\frac{\partial E}{\partial w_{ij}}(t)<0\\ \Delta_{ij}(t-1), & inak\end{cases}$$

Kde η^{-} je parameter ktorým sa znižuje veľkosť kroku a η^{+} , ktorým sa zvyšuje a platí $0<\eta^{-}<1<\eta^{+}$. Ďalej veľkosť krokov je ohraničená dvomi ďalšími parametrami Δ_{min} a Δ_{max} . Boli vyvinuté 4 rôzne variácie tohto algoritmu, ktoré v svojej práci zhrnuli a porovnali Igel a Hüsken v roku 2000 [Igel, 2000].

Rprop+

Pôvodný algoritmus s „weight-backtracking“. Každú váhu v sieti upravíme nasledovným spôsobom.

Pri každej iterácii sa zväčšuje veľkosť kroku. Kontroluje sa či sa zmenilo znamienko parciálnej derivácie. Ak sa nezmenilo, váha sa zmení podľa pravidla:

$$\begin{aligned}&Ak \ \frac{\partial E}{\partial w_{ij}}(t-1)\cdot\frac{\partial E}{\partial w_{ij}}(t)\geq 0 \ , potom\\ &\Delta_{ij}(t)=min\left(\Delta_{ij}(t-1)\cdot\eta^{+},\Delta_{max}\right)\\ &\Delta w_{ij}(t)=-sign\left(\frac{\partial E}{\partial w_{ij}}(t)\right)\cdot\Delta_{ij}(t)\\ &w_{ij}(t+1)=w_{ij}(t)+\Delta w_{ij}(t)\end{aligned}$$

Ak sa znamienko zmenilo, vrátime predchádzajúcu hodnotu váhy:

$$\begin{aligned}&Ak \ \frac{\partial E}{\partial w_{ij}}(t-1)\cdot\frac{\partial E}{\partial w_{ij}}(t)<0 \ , potom\\ &\Delta_{ij}(t)=max\left(\Delta_{ij}(t-1)\cdot\eta^{-},\Delta_{min}\right)\\ &\Delta w_{ij}(t)=-\Delta w_{ij}(t-1)\\ &\frac{\partial E}{\partial w_{ij}}(t)=0\end{aligned}$$

Nastavenie uloženej hodnoty derivácie na nulu je implementačný trik na zamedzenie úpravy rýchlosti učenia Δ_{ij} v ďalšej iterácii.

Ak je súčin derivácii nula, nemeníme rýchlosť učenia, len vypočítame zmenu váhy a upravíme ňou váhu.

$$Ak \frac{\partial E}{\partial w_{ij}}(t-1) \cdot \frac{\partial E}{\partial w_{ij}}(t)=0 \quad , potom$$
$$\Delta w_{ij}(t)=-sign\left(\frac{\partial E}{\partial w_{ij}}(t)\right)\cdot\Delta_{ij}(t)$$
$$w_{ij}(t+1)=w_{ij}(t)+\Delta w_{ij}(t)$$

Rprop-

Verzia bez „weight-backtracking“, v ktorej nie je nutné si ukladať predchádzajúce zmeny váh.

$$Ak \frac{\partial E}{\partial w_{ij}}(t-1) \cdot \frac{\partial E}{\partial w_{ij}}(t)>0 \; potom$$
$$\Delta_{ij}(t)=min\left(\Delta_{ij}(t-1)\cdot\eta^{+}, \Delta_{max}\right)$$

$$Ak \frac{\partial E}{\partial w_{ij}}(t-1) \cdot \frac{\partial E}{\partial w_{ij}}(t)<0 \; potom$$
$$\Delta_{ij}(t)=max\left(\Delta_{ij}(t-1)\cdot\eta^{-}, \Delta_{min}\right)$$

$$Ak \frac{\partial E}{\partial w_{ij}}(t-1) \cdot \frac{\partial E}{\partial w_{ij}}(t)=0 \; potom$$
$$\Delta_{ij}(t)=\Delta_{ij}(t-1)$$

Nakoniec vo všetkých prípadoch upravíme váhu:

$$w_{ij}(t+1)=w_{ij}(t+1)-sign\left(\frac{\partial E}{\partial w_{ij}}(t)\right)\cdot\Delta_{ij}(t)$$

iRprop+

Toto je vylepšenie Rprop+ algoritmu. Ak sa zmenilo znamienko parciálnej derivácie, znamená to že algoritmus preskočil lokálne minimum, ale neznamená to že sa znížila ani zvýšila chyba. Algoritmus zostáva rovnaký ako Rprop+ zmeníme len časť kde sa menilo znamienko parciálnej derivácie.

$$Ak \frac{\partial E}{\partial w_{ij}}(t-1) \cdot \frac{\partial E}{\partial w_{ij}}(t) < 0 \quad , potom$$
$$\Delta_{ij}(t) = max\left(\Delta_{ij}(t-1) \cdot \eta^{-}, \Delta_{min}\right)$$
$$Ak\ E(t) > E(t-1)\ potom \quad w_{ij}(t+1) = w_{ij}(t) - \Delta w_{ij}(t-1)$$
$$\frac{\partial E}{\partial w_{ij}}(t) = 0$$

iRprop-

Algoritmus iRprop- je iRprop+ bez „weight-backtracking“. Ak sa zmení znamienko parciálnej derivácie zmení len veľkosť kroku, ale nie váhu. Od Rprop- sa líši tým že uloženú parciálnu deriváciu nastaví na nulu a tým nezmení váhu.

$$Ak \frac{\partial E}{\partial w_{ij}}(t-1) \cdot \frac{\partial E}{\partial w_{ij}}(t) < 0\ potom$$
$$\Delta_{ij}(t) = max\left(\Delta_{ij}(t-1) \cdot \eta^{-}, \Delta_{min}\right)$$
$$\frac{\partial E}{\partial w_{ij}}(t) = 0$$

1.4.3 Techniky na zlepšenie trénovania a generalizácie siete

Spomeniem len zopár techník ktoré sme používali a ktoré sú popísané v [Haykin, 2000].

Krížová validácia (Cross validation)

Potrebujeme zistiť ako sa model siete správa na dátach vo všeobecnosti, ako generalizuje teda nie len na tých ktorými sme trénovali. To preto aby sme mohli vybrať najlepší model.

Dáta rozdelíme na trénovaciu a testovaciu množinu. Trénovaciu ďalej na estimačnú a validačnú. Trénovacia bude slúžiť na zistenie parametrov modelu siete – trénovanie, validačná na vybranie najlepšieho modelu. Aby sa nám však náhodou nestalo aby model bol preučený práve na validačnú množinu dát, generalizáciu odhadneme pomocou testovacej chyby.

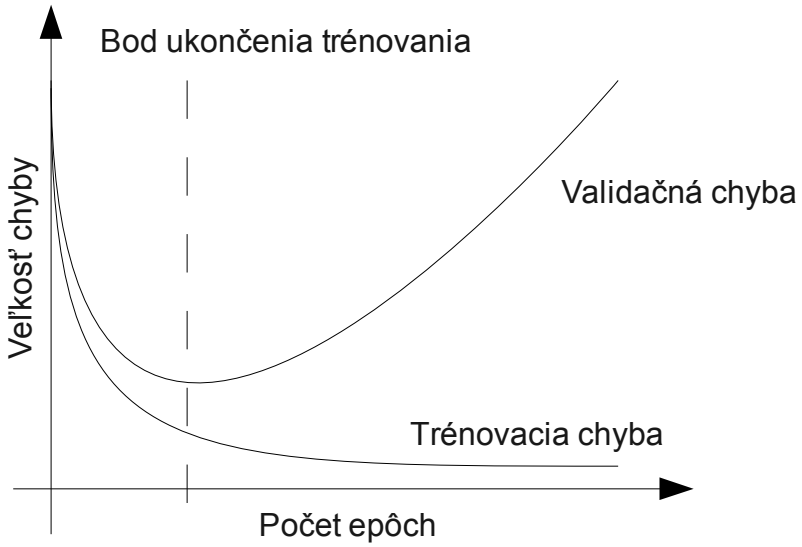
Selekcia modelu

Potrebujeme zistiť ktorý model z hľadiska zložitosti siete je najlepší, pretože príliš zložitá sieť má tendenciu sa preučiť a zase príliš jednoduchá podučiť. Zložitosť siete môžeme vyjadriť pomocou voľných parametrov siete (váh a prahov), počtu skrytých neurónov, počtu skrytých vrstiev.

Trénujeme viacero sietí s rôznou zložitosťou z ktorých vyberáme najlepší.

Skoré zastavenie (Early stopping)

Na obrázku 7 je zobrazený priebeh trénovania. V určitom okamžiku začne validačná chyba stúpať. Je to pretože sieť je preučená na trénovacie dáta. Aby sme zamedzili tomuto preučeniu, zastavíme trénovanie keď začne validačná chyba stúpať.



Obrázok 7: Priebeh trénovania siete

1. sieť trénujeme určitý počet iterácií
2. zistíme validačnú chybu, ak sa zvýšila skončíme, inak znova trénujeme

Normalizácia dát

Je to štatistické predspracovanie vstupných dát. Pomáha k lepšej konvergencii trénovacích algoritmov neurónových sietí. Každý vstupný neurón môže mať vstupné dáta merané v rôznych jednotkách a tie môžu mať rôznu variabilitu.

Preto pre každú dimenziu dát(vstupný neurón) robíme samostatne. Čiže máme dáta $x_1, x_2, \ldots, x_n \in \mathbb{R}$.

Normalizácia sa skladá sa z dvoch častí:

1. Nulová stredná hodnota. Od každej hodnoty odpočítame strednú hodnotu.
2. Jednotková štandardná odchýlka. Vydelíme hodnoty štandardnou odchýlkou.

$$\mu = \frac{1}{n}\sum_i x_i$$
$$\sigma^2 = \frac{1}{n}\sum_i x_i^2$$
$$x_i' = \frac{(x_i - \mu)}{\sigma}$$

Výsledné hodnoty po normalizácii sú x_i' .

1.5 Aplikácie umelej inteligencie v robotike

V tejto práci sa budeme zameriavať na rozpoznávanie cesty z obrazu takže spomenieme predchádzajúce práce v tejto oblasti.

1.5.1 Videnie pomocou neurónových sietí na ovládanie robota

V tejto práci „Neural Network Vision for Robot Driving“ [Pomerleau , 1996] je použitá neurónová sieť na autonómne ovládanie auta. Celý systém sa volal ALVINN (Autonomous Land Vehicle In a Neural Network). Vstup siete je „sietnica“ s 30x32 pixlov, čo je buď z kamery alebo laserového senzoru na vzdialenosť. Skrytá vrstva obsahuje len 4 neuróny a 30 výstupných, kde každý z výstupných neurónov reprezentuje jeden smer zatáčania. Výsledný smer zatáčania sa určoval pomocou porovnania distribúcie aktivačných hodnôt výstupných neurónov s normálnym rozdelením (krivkou Gaussovej funkcie).

Trénovanie prebiehalo za chodu vozidla. Z dát nazbieraných počas jazdy boli generované aj nové pomocou transformácie existujúcich, aby bolo zachytených viacero prípadov, pretože zozbierané dáta neboli postačujúce. Transformácia prebiehala tak, že sa pôvodný obrázok posunul a otočil tak, aby reprezentoval nový prípad. Po transformácii však ostávajú miesta kde chýbajú pixly, a tie sú následne určené extrapoláciou. Taktiež bolo treba transformovať smer zatáčania, aby zodpovedal transformovanému obrázku.

Následne bol pridaný buffer na trénovacie príklady, tak aby sa nezabúdali, pretože by sa pretrénovali inými. Udržiavala sa množina príkladov tak, aby priemer ich smerov bol priamy smer. Čiže ak bolo veľa príkladov so zákrutou doprava, tak sa vyhodil príklad so zákrutou doprava.

Bolo zistené, že na správne trénovanie treba použiť aj transformované trénovacie príklady aj bufferovanie.

1.5.2 Detekcia cesty segmentáciou

V článku „A simple and efficient Road Detection Algorithm for Real Time Autonomous Navigation based on Monocular Vision“ [Neto, 2006] je opísaný spôsob detekcie cesty z obrazu použitím segmentácie. Algoritmus bol použitý v súťaži Grand Challenge, ktorú poriada DARPA.

Jeden spôsob ako robiť segmentáciu je prahovanie. Je veľmi rýchla, avšak je problém nájsť správne hodnoty prahu. Najlepšie v tejto oblasti je hľadať prahy automaticky.

Prahovnie rozlišujeme na globálne a lokálne. V globálnom sa používa jeden prah pre celý obrázok. Pri lokálnom sa obrázok rozdelí na menšie pod-obrázky, ktoré majú každý určený vlastný prah.

Podstatou bolo určiť čo najpresnejšie región kde sa nachádza cesta a následne sa použilo globálne prahovanie. Z pôvodného obrázka sme vytvárali menšie obrázky tak, že smerom z dola zoberieme 10% výšky pôvodného obrázku, ďalší 20%, atď. Čiže každý ďalší menší obrázok obsahoval predchádzajúci. Každý z nich bol potom segmentovaný pomocou metódy ktorú predstavil Otsu v roku 1978 [Otsu, 1978].

Následne sa analyzovalo koľko percent bielych bodov (reprezuntujúce cestu) z prechádzjúceho menšieho obrázka sa nachádza aj v ďalšom. Následne sa pomocou štandardnej odchýlky našiel index obrázka, ktorý určil kde bude zlom horizontu.

Teraz sa použilo globálne prahovanie na oblasť pod horozontom.

Ďalej sa určil výsledný vektor, a to tak že z prahovaného obrázka sa spočítala suma bielych pixelov v stĺpci. Tento vektor bol ďalej posunutý do navigačnej vrstvy systému.

1.5.3 Detekcia cesty z obrazu pomocou modelov cesty

Prvou vecou čo ma zaujalo v článku „Vision–based Road Detection using Road Models “ [Álvarez, 2009], je prístup hľadania cesty v obraze pomocou klasifikácie modelu-geometrie cesty a jej kombinácia s klasifikáciou pomocou pixlov.

Teda bola zvolená množina modelov cesty. Boli to binárne obrázky kde bol zachytený tvar cesty, zákruta doľava, doprava, cesta rovno, s prekážkou na ceste, atď. Potom bol natrénovaný klasifikátor ktorý vstupný obraz klasifikoval na jeden model z množiny. Išlo o kombináciu algoritmov SIFT na hľadanie čŕt, k-means na redukciu dimenzionality a SVM na klasifikáciu.

Ďalej bola táto metóda spojená s klasifikáciou pomocou pixlov, konkrétne šlo o klasifikátor pomocou najmenšej vzdialenosti (minimum–distance classifier) v priestore invariantnom na osvetlenie.

Najskôr sa teda určil model. Podľa toho ktorý bol zvolený sa použili parametre pre klasifikáciu pomocou pixlov. Teda každý jeden model cesty mal tie parametre samostatne určené.

1.5.4 Klasifikácia pomocou neurónových sietí

V tomto článku „Path Recognition for Outdoor Navigation Using Artificial Neural Networks: Case Study “ [Shinzato, 2010] je rozobrané klasifikovanie cesty pomocou neurónových sietí. Išlo o obraz z kamery mobilného robota kde boli chodníky, cesty, tráva, stromy. A bolo treba zistiť oblasť, kde sa robot môže bezpečne pohybovať, teda najmä cesty a chodníky.

Vstupný obraz bol rozdelený na menšie obrázky, ktoré boli následne spracované do vstupu pre

neurónovú sieť. Išlo o váhované priemery a entropie RGB, HSV farebných modelov, ich jednotlivých komponentov a rôznych kombinácií medzi sebou.

Boli uskutočnené tri experimenty. V prvom experimente boli siete trénované zo segmentov z celého vstupného obrázka. V ďalšom len z oblasti pod horizontom. A v poslednom len z oblastí, ktoré určite boli cestou a tých ktoré určite neboli. Na vstupnom obrázku to vyzeralo tak že oblasť priamo pred robotom bola určite cesta a dva štvorce nad horizontom (jeden napravo, jeden naľavo na okraji) boli oblasti kde určite cesta nebola.

Najlepšie výsledky vychádzali keď vstupom bol váhovaný priemer RGB zložiek pixlov, plus ďalší vstup ako entropia RGB. Tiež bolo zistené že klasifikáciou menšej oblasti sa zlepšili výsledky (druhý experiment) a aj pri menšom počte, no kvalitnejších dát sa zlepšujú výsledky (tretí experiment).

Neurónové siete boli MLP, 4 rôzne modely: jednovrstvová a 5 skrytých neurónov, jednovrstvová a 10 skrytých neurónov, dvojvrstvová a 5 skrytých neurónov v oboch vrstvách, dvojvrstvová a 10 skrytých neurónov v oboch skrytých vrstvách.

1.6 Robotour

Je sútaž outdoorových robotov v parku poriadaná českým združením robotika.cz. S vedúcim tejto diplomovej práce sme sa ako tím „Smelý zajko“ zúčastnili na ročníku 2010.

Zadanie súťaže:

- Doviesť náklad do určeného cieľa
- Robot sa môže pohybovať len po vyznačených cestách v parku (pomocou mapy)
- Robot nesmie prísť do kontaktu s prekážkou na ceste, prekážku môže buď obísť, alebo zastaviť sa a ak prekážka zmizne, musí sa pohnúť ďalej
- Robot musí uviesť 5l súdok piva, buď prázdny alebo plný
- Ak robot vyjde z cesty, pokus sa končí a zaznamená sa vzdušná vzdialenosť od cieľa
- Robot musí mať veľké červené tlačidlo na okamžité zastavenie robota
- Tím môže používať len spoločnú mapu zo servera Open Street Map, iné mapy sú zakázané

Bodovanie:

Jeden bod za každý ušetrený meter vzdušnej vzdialenosti smerom k cieľu. Ak robot neopustí štartovaciu pozíciu získava 0 bodov. Ak robot vozí plný súdok piva, získava dvojnásobný počet bodov.

Súťaž má 4 kolá a vyhráva tím s najväčším počtom bodov. Rýchlosť nehrá rolu (je obmedzená na 2,5m/s).

Homologizácia:

Prebiehala deň pred súťažou, išlo o testovanie základných funkcií robotov. Každý robot musel prejsť aspoň 10 metrov po testovacej ceste bez toho aby vyšiel na trávu. Ďalej musel zastaviť pred prekážkou, človekom, a po zmiznutí prekážky sa robot musel dať opäť do pohybu. Nakoniec bola overená funkčnosť núdzového tlačidla na okamžité zastavenie.

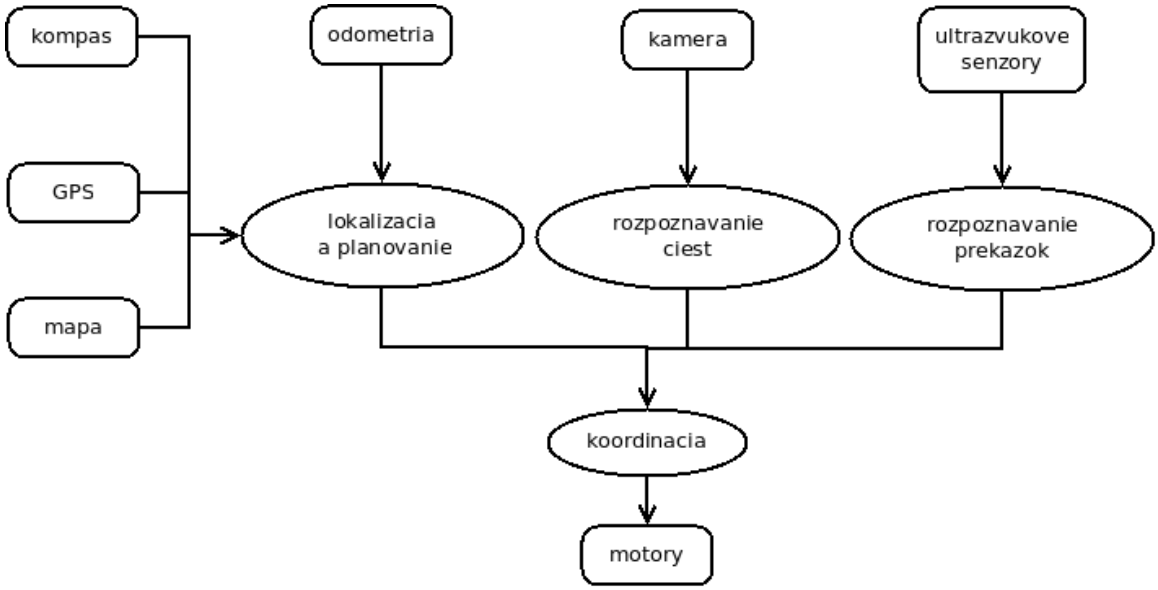
Miesto:

Súťaž sa konala v Botanickej záhrade v Bratislave. V čase konania súťaže bol môj projekt len v polčase, napriek tomu sa nám podarilo pripraviť robota, ktorý sa úspešne homologizoval a pohyboval po chodníkoch v parku. V nasledujúcej časti opíšeme finálne riešenie, ktoré pripravujeme na súťaž Robotour 2011, ktorá sa koná na jeseň vo Viedni.

2 Časť

2.1 Návrh riešenia

Celý systém robota sme rozdelili do nasledujúcich logických modulov.



Obrázok 8: Modulárna dekompozícia systému

Aktuátor: motory

Senzory: kompas, GPS, odometria, kamera, ultrazvukové senzory

Softvérové moduly: lokalizácia a plánovanie, rozpoznávanie ciest, rozpoznávanie prekážok

Softvérové moduly nám systém rozdelili na pod-problémy z ktorých kľúčovým bolo rozpoznávanie ciest z obrazu, bližšie v kapitole 2.2 , ostatné moduly sú popísané v kapitole 2.3 .

Mechanika robota

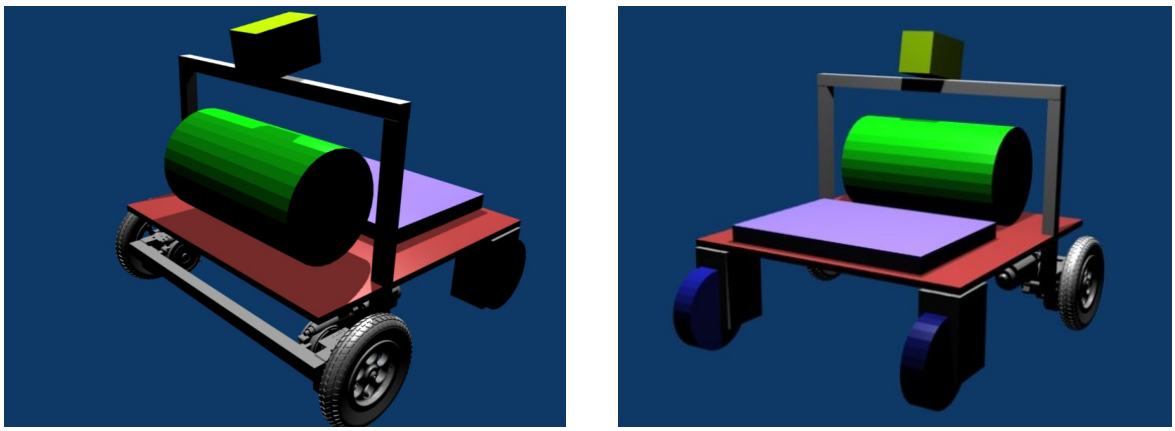
Najskôr som si preštudoval ako prebieha Robotour súťaž, taktiež minulé ročníky a výsledky. Čiastočne som sa inšpiroval robotmi z minulých ročníkov. Robota sme navrhovali tak, aby bol schopný odviesť plný 5l súdok piva, pretože je za to dvojnásobný počet bodov. To ovplyvňovalo veľkosť robota, silu konštrukcie a tiež aj motorov.

Chcel som taktiež robota, ktorý by bol schopný sa otočiť na mieste, teda nulový polomer otáčania. To pre prípad keď by sa robot potreboval otočiť na úzkej ceste.

Najjednoduchší systém je takzvaný „differential drive“, kde poháňané kolesá pomocou rozdielnej

rýchlosti slúžia aj na zatáčanie. Teda nie je potrebné riešiť natáčanie kolies.

Najskôr som uvažoval o štvorkolesovom variante s dvoma poháňanými kolesami. No nakoniec boli kolesá namontované tri, a to preto, že na hrboľatom teréne sa mu mohlo dostať jedno z poháňaných kolies do vzduchu. To by spôsobovalo problémy, lebo robot by na nerovnom povrchu zatáčal inak ako by chcel.



Obrázok 9: Model návrhu

Súdok (zelený valec) je v prednej časti kvôli posunutiu ťažiska ku poháňaným kolesám. Za ním bude umiestnený laptop (fialovou), pod ktorým bude umiestnená zvyšná elektronika. Kamera sa bude nachádzať na ráme, pre vhodný pozorovací uhol.

2.2 Problém rozpoznávania ciest z obrazu

Robot má rozpoznávať cestu z obrazu, takže vždy zachytí z kamery jeden obrázok a vyhodnotí ho a následne vypočíta výstup. Systém teda nebude rozhodovať na základe sekvencie obrázkov v čase.

Na tento problém som sa rozhodol použiť MLP neurónové siete, čo je vhodný prístup na rozpoznávanie vzoriek z obrazu. Jedným z dôvodov bolo aj to, že v doterajších riešeniach tímov v súťaži Robotour táto metóda použitá nebola a chceli sme vidieť nakoľko môže iným metódam konkurovať.

Nechcel som vkladať celý obraz naraz do jednej siete, pretože by to bolo veľmi pomalé alebo príliš malé rozlíšenie vstupu, a tak isto by bol problém aký by mala sieť výstup. Tento postup bol síce úspešne použitý v projekte ALVINN [Pomerleau, 1996], ale v našom prípade išlo o chodníky s rôznymi druhmi povrchu i okolitého prostredia, s križovatkami a nepravideľnosťami. Preto som uprednostnil prístup klasifikácie oblastí v obraze kamery neurónovými sieťami v kombinácii s následným vyhodnotením a spracovaním.

Celkový výstup z modulu rozpoznávania ciest, bude vektor ohodnotení jednotlivých smerov. Teda z obrázku sa zistí ktorý smer a nakoľko je z hľadiska videnia robota výhodný. Toto predurčuje neskoršie vhodné spájanie s ďalšími modulmi v module koordinácia, pretože bude možné vybrať si z viacero „dobrých“ smerov na základe informácií z ďalších modulov.

Z predchádzajúcich vlastností som postupne prišiel ku experimentálnemu riešeniu a to:

1. vstupný obraz rozdeliť na menšie časti / segmenty
2. každý z tých segmentov bude vstupom neurónovej siete
3. výsledok z behov siete bude matica výstupov siete
4. na tejto matici sa ohodnotia smery pohybu

Čiže neurónová sieť bude slúžiť len na zistenie či oblasť na obraze je cesta alebo nie je (výstup zo siete však bude mať spojitý obor hodnôt).

Potrebujeme riešiť 3 problematiky:

1. trénovanie siete vrátane vytvárania trénovacích príkladov
2. zistiť ako vplýva použitie prídavných vstupov na kvalitu predikcie sieťou
3. ohodnotiť smery pohybu na základe výstupu siete

2.2.1 Trénovanie siete

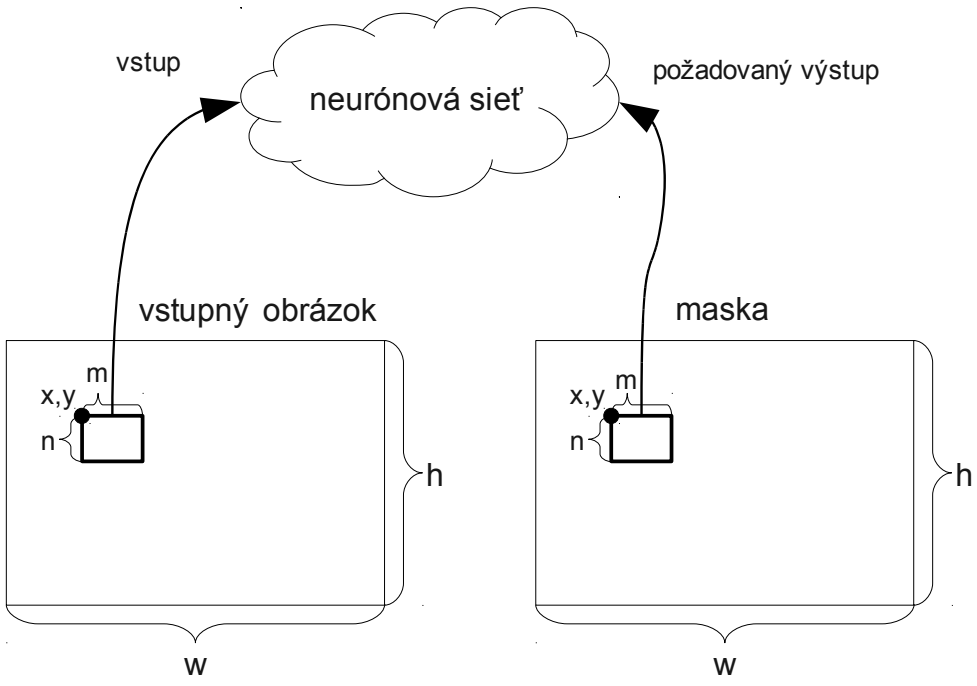
Získanie trénovacích príkladov

Videozáznam nasekáme na jednotlivé obrázky a následne ich budeme ručne ohodnocovať. Treba teda vyznačiť na obrázku oblasť kde sa nachádza cesta vhodná pre pohyb robota. Vytvoríme masku, kde bude miesto vhodné pre pohyb robota ohodnotené hodnotou 1 a zvyšná oblasť hodnotou 0. Toto nám zabezpečí zároveň aj pozitívne aj negatívne príklady.

Potom budú takto dvojica obrázok, maska nasekané na segmenty, kde konkrétny trénovací príklad bude pozostávať zo vstupu – segment zo vstupného obrázku a žiadaného výstupu – segment z masky.

Obrázok 10: Príklad obrázku a k nemu vytvorenej masky

Postup trénovania



Obrázok 11: Základný prehľad trénovacieho procesu

Najskôr vygenerujeme trénovacie príklady.

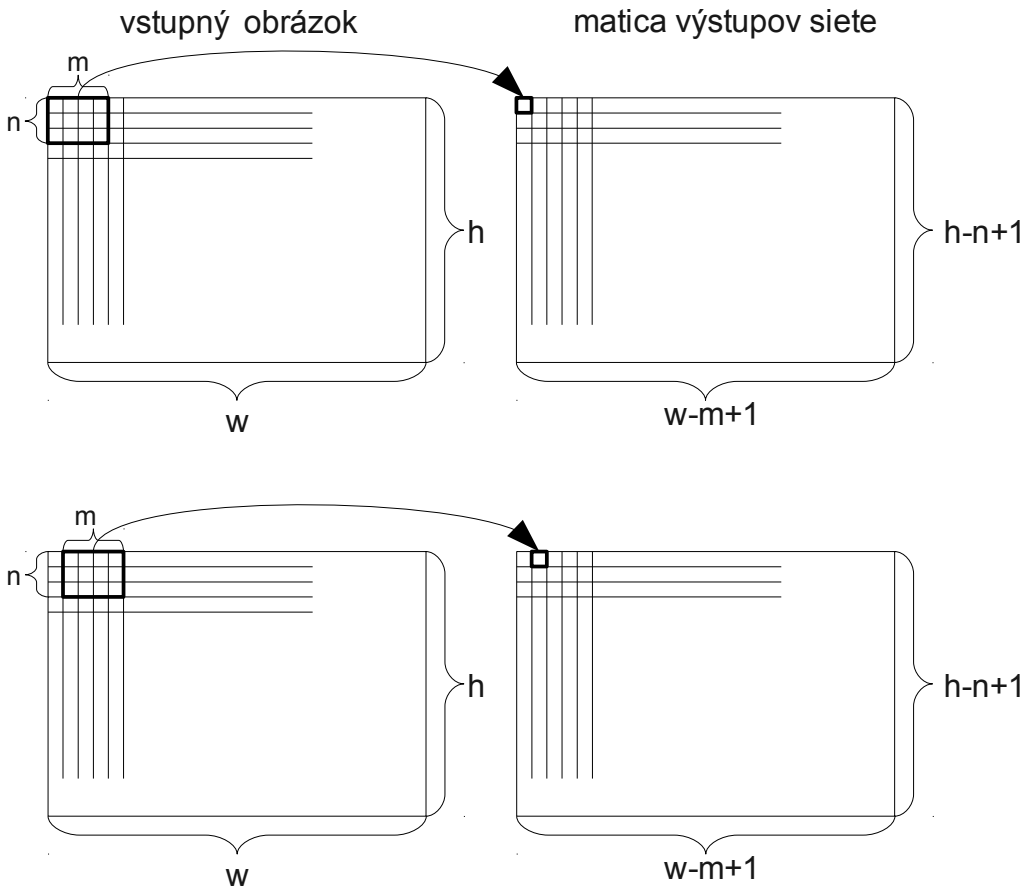
Náhodne vygenerujeme dvojicu *(x,y)*, tak aby $0 \leq x \leq w-m+1 ; 0 \leq y \leq h-n+1$. Všetky pixely zo vstupného obrázka ktoré patria do intervalu <*x, x+m*> na x-ovej osi a <*y, y+n*> na y-ovej osi zapíšeme do vektora dĺžky *m*n*3*. Každý takýto pixel bude zapísaný ako trojica hodnôt r,g,b farebných zložiek. Tento vektor bude vstupom siete pri trénovaní.

Podobne z masky zoberieme pixely z intervalu <*x, x+m*> na x-ovej osi a <*y, y+n*> na y-ovej osi. Vypočítame priemernú hodnotu z týchto pixelov (pixely majú hodnoty len {0,1}), a to bude slúžiť ako požadovaný výstup siete.

Z každého vstupného obrázku vygenerujeme 1000 takýchto trénovacích príkladov. Mohli by sme zobrať aj všetky ale bolo by ich príliš veľa.

Keď máme vygenerované trénovacie príklady / dáta, normalizujeme ich čo prispeje k lepšej konvergencii siete (popísané v kapitole 1.4.3). Normalizované dáta rozdelíme na estimačnú, validačnú a testovaciu množinu. Trénovanie ukončíme ak validačná chyba začne stúpať.

Vyhodnotenie vstupného obrázku sieťou



Obrázok 12: Prehľad priebehu vyhodnotenia obrázku neurónovou sieťou a reprezentácia výstupu

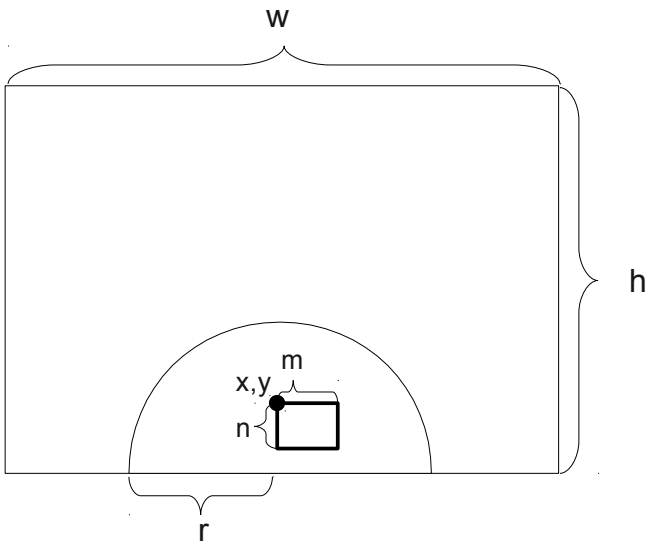
2.2.2 Prídavné vstupy

Sieť dávala dobré výsledky, avšak mala problémy najmä zo „zlými“ vstupnými obrázkami, ako rôzna svetelnosť a farebnosť prostredia, rôzne tiene na ceste a rôzne objekty v obraze. Preto sme sa rozhodli pridať ďalšie vstupy, ktoré by pomohli ku lepším výsledkom.

Potrebovali by sme informáciu o obrázku ako celku. Pretože zatiaľ sieť fungovala len s lokálnou informáciou.

Vstup oblasti s veľkou pravdepodobnosťou výskytu cesty

Inšpirovali sme myšlienkou, ktorú využívajú aj niektoré algoritmy iných tímov: robot bude mať s veľkou pravdepodobnosťou priamo pred sebou cestu. Tento fakt môžeme využiť ako prídavný vstup siete. Malo by to zaručiť lepšie výsledky.



Obrázok 13: Vstupný obrázok a oblasť s veľkou pravdepodobnosťou výskytu cesty, vyznačená ako polkruh s polomerom r.

Definitoricky sme určili s polomerom *r* oblasť od stredu dolnej časti obrázka. Potom bol náhodne vybraný segment, potom bolo možné pokračovať dvoma spôsobmi:

1. Pridať sieti priamo farebné zložky pixlov prídavného segmentu (veľa vstupov).
2. Pridať sieti vypočítanú hodnotu vzdialenosti aktuálneho segmentu od segmentov z celého obrazu (strata informácií). Použili sme funkciu vzdialenosti $y=(x_1-x_2)^2$ (SQDIFF).

V druhom prípade budeme postupovať nasledovne:

1. vyberieme náhodne segment zo vstupného obrázka (tak isto ako bez prídavného vstupu)
2. vyberieme náhodne segment prídavného vstupu z oblasti s veľkou pravdepodobnosťou cesty.
3. porovnáme prídavný segment so vstupným obrázkom a sieť dostane ako vstup, hodnotu, ktorá sa nachádza na mieste vstupného segmentu

Histogram farieb

Pri pokusoch sme videli, že sieť musí nielen vedieť nájsť ideálnu vzorku cesty, a tiež treba odfiltrovať okolie cesty. V trénovacích dátach sa nachádzali rôzne typy ciest.

Ďalšou globálnou informáciou by mohli byť histogramy farieb. Ak veľká časť zorného poľa bude určitá cesta, jej farebné vlastnosti sa premietnu do histogramu, takisto aj svetelné podmienky.

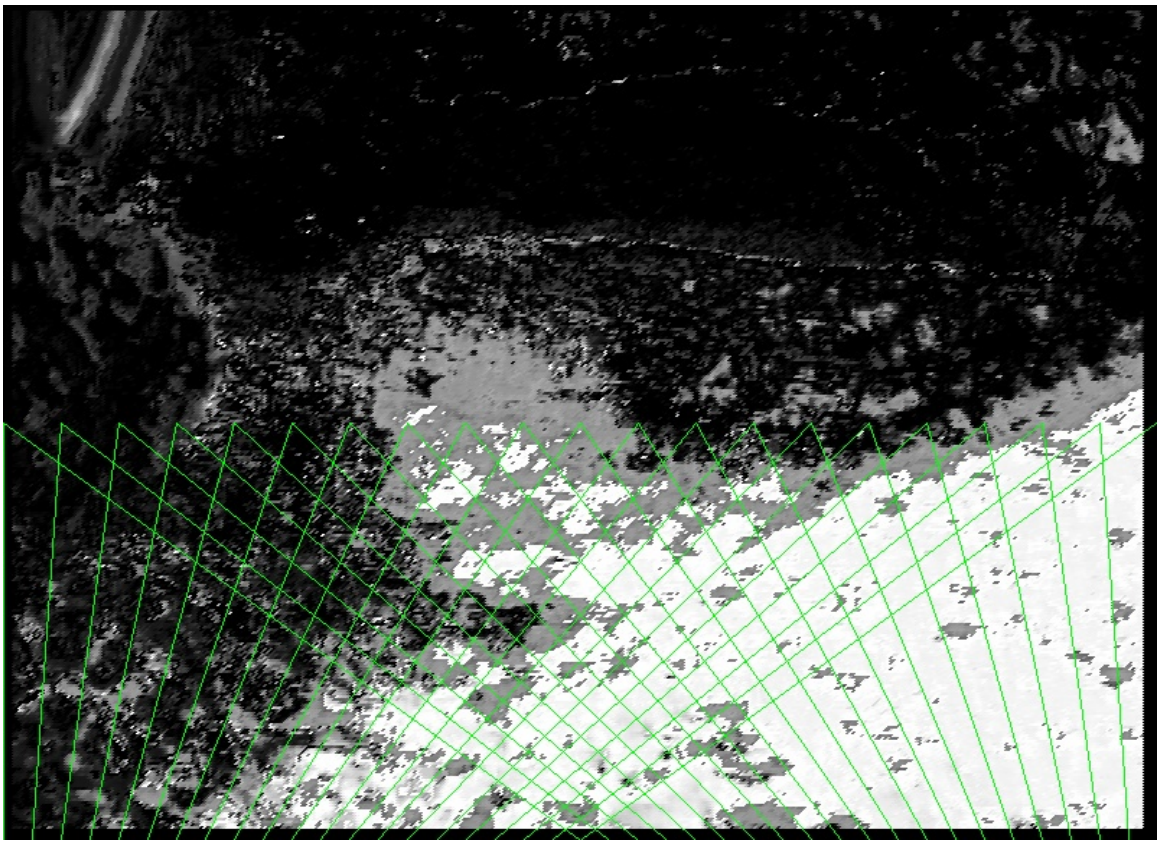
Veľkosťou vstupu v tomto prípade rozumieme rozlíšenie histogramu. Teda koľko rozličných farieb histogram dokáže rozlíšiť.

V oboch prípadoch(vstupu oblasti aj histogramu) budeme teda porovnávať chybu siete s pridanými vstupmi a bez nich.

2.2.3 Ohodnotenie smerov pohybu

Z matice výstupov siete je treba získať „najlepší“ smer kam sa pohnúť podľa kamery. Každý smer bol reprezentovaný jedným trojuholníkom a na základe priemernej hodnoty v tejto ploche sa určilo, nakoľko je daný smer dobrý, aby robot nevyšiel z cesty.

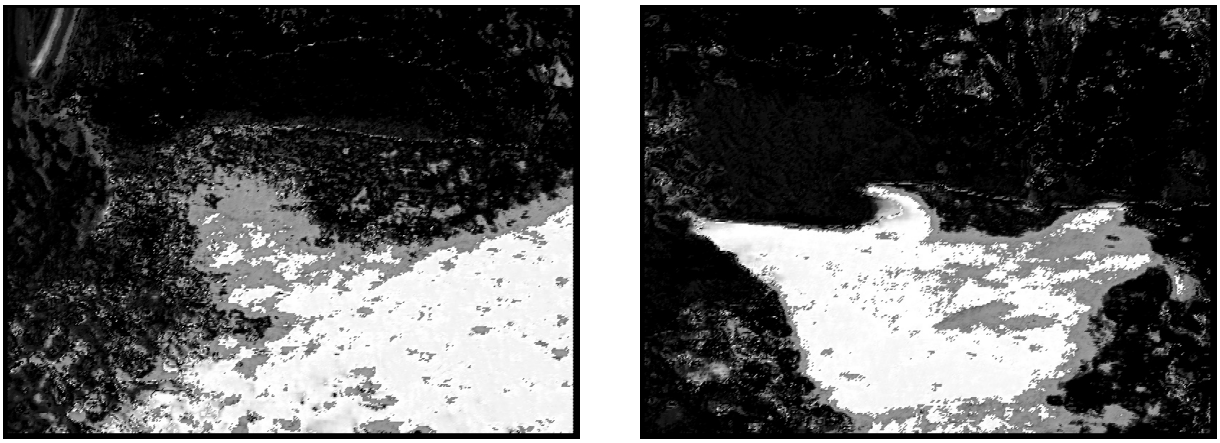
Na obrázku 14 je zobrazený výstup z neurónovej siete po aplikovaní na vstupný obrázok. Zelenou farbou sú naznačené trojuholníky ktoré reprezentujú smer. Pre každý trojuholník je vypočítaná priemerná intenzita pixla. A tá je ohodnotením smeru.



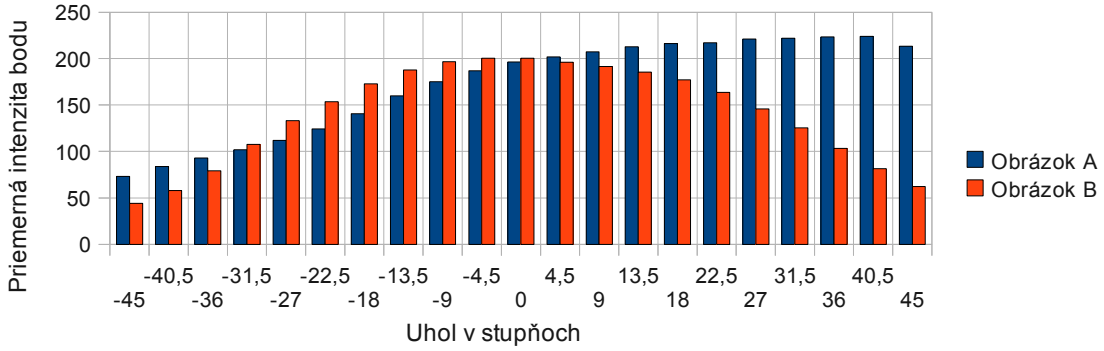
Obrázok 14: Trojuholníky reprezentujúce možné smery pohybu

Príklad tejto metódy. Najskôr sú vstupné obrázky(obrázok 15 vpravo a vľavo) spracované sieťou. K nim prislúchajúce matice výstupov sú na obrázku 16. Na obrázku 17 je výsledok analýzy týchto matíc výstupov.

Obrázok 15: Vstupné obrázky



Obrázok 16: Príklady matíc výstupov zo siete



Obrázok 17: Príklad ohodnotenia dvoch matíc výstupu zo siete (A – obrázok 16 vľavo, B – obrázok 16 vpravo).

Tieto hodnoty, vektor ohodnotení smerov, bude výstupom modulu rozpoznávania ciest.

2.3 Ostatné moduly

2.3.1 Lokalizácia a plánovanie

Do tohto modulu bude patriť všetko, čo sa týka mapy, lokalizácie a plánovania trasy. Výstupom celého modulu bude uhol, kam je z hľadiska pozície a orientácie robota najlepšie sa pohnúť.

Podľa zadania Robotouru, mapu majú všetky tímy spoločnú a tým odpadá potreba mapovať prostredie. Poskytnutú mapu si prevedieme do grafovej reprezentácie, teda budeme pracovať len s vrcholmi a hranami, kde hrany sú úsečky reprezentujúce cestu a vrcholy sú buď križovatky alebo body na ceste, ktorými je vyjadrené zakrivenie cesty. Vstupom bude tiež poloha cieľu.

Na prácu s mapou potrebujeme zistiť aktuálnu pozíciu a orientáciu v prostredí. Na to sme použili GPS a elektronický kompas. Oba tieto senzory majú relatívne veľkú chybu merania. Vrcholy v grafe majú rovnaký formát ako výstup z GPS prijímača. A za aktuálnu polohu budeme pre jednoduchosť považovať tento výstup.

Plánovanie trasy

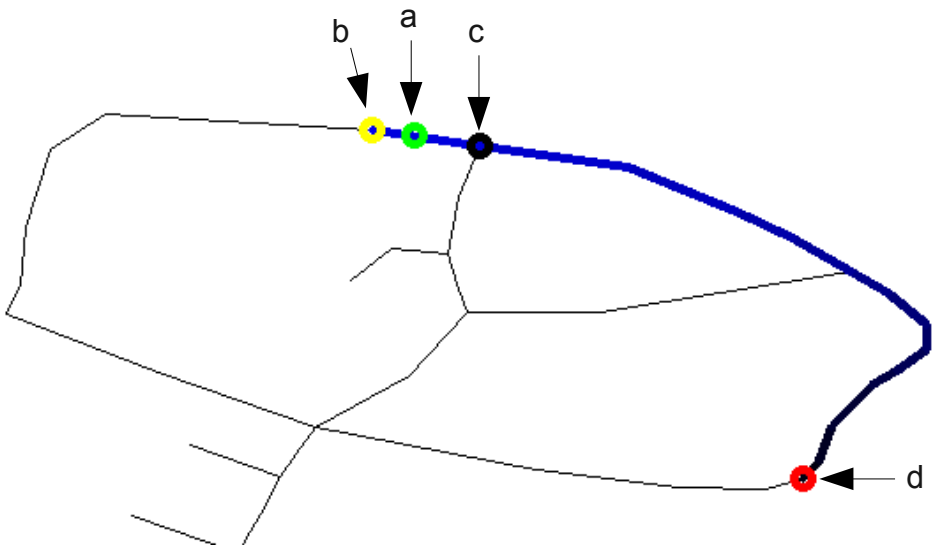
Podľa aktuálnej pozície, vieme naplánovať trasu, tak aby bola čo najkratšou cestou ku cieľu. Toto budeme robiť pomocou prehľadávania do šírky.

Okrem zadaného cieľu budeme rozlišovať dlhodobý a krátkodobý cieľ. Dlhodobý cieľ je vrchol na mape ktorý je najbližšie k zadanému cieľu. Krátkodobý cieľ je vrchol na mape ku ktorému priamo(rovno) smerujeme. Po dosiahnutí krátkodobého cieľa sa určí ďalší, až kým neprídeme do cieľa. Pracujeme len s vrcholmi, pretože sa môžeme pohybovať len po mape.

Najskôr zistíme ku ktorému vrcholu na mape sme najbližšie. Od tohto vrcholu plánujeme trasu k dlhodobému cieľu.

Keď je trasa naplánovaná (nájdená prehľadávaním) a pohneme sa (ďalšia iterácia), zistíme ku ktorému vrcholu na mape sme najbližšie. Ak je tento vrchol mimo trasy, vytvoríme novú trasu, ktorá bude začínať v tomto vrchole. Ak dosiahneme krátkodobý cieľ, zistíme z naplánovanej trasy ďalší vrchol a ten bude ďalší krátkodobý cieľ.

Dosiahnutie dlhodobého aj krátkodobého cieľa považujeme, ak vzdialenosť aktuálnej polohy a cieľa je menšia ako definovaná konštanta.



Obrázok 18: Príklad mapy a plánovania trasy. ***a*** *– aktuálna poloha robota,* ***b*** *– najbližší vrchol ku polohe robota,* ***c*** *– krátkodobý cieľ,* ***d*** *– dlhodobý cieľ*

Výpočet výstupu

Z aktuálnej polohy a naplánovanej trasy vieme zistiť ideálnu orientáciu. Z kompasu vieme zistiť aktuálnu orientáciu. Z tohto vieme vypočítať uhol o ktorý sa máme otočiť, tak aby sme dosiahli optimálnu orientáciu, čo je výstupom tohto modulu.

2.3.2 Rozpoznávanie prekážok

Na rozpoznávanie prekážok používame ultrazvukové senzory.

Tento modul nebol priamo implementovaný. Pravidlá súťaže požadujú iba, aby robot zastavil, ak sa pred ním nachádzala prekážka a túto požiadavku sme splnili.

Čiastočne vplýval na rozpoznávanie prekážok aj modul rozpoznávania ciest, ak objekt sa vizuálne nepodobal na možnú cestu, robot prekážku plynule obišiel.

2.3.3 Koordinácia

V tomto module budeme kombinovať výsledky z predchádzajúcich modulov a na základe výstupu tohto modulu sa budú vykonávať akcie (pohyb robota).

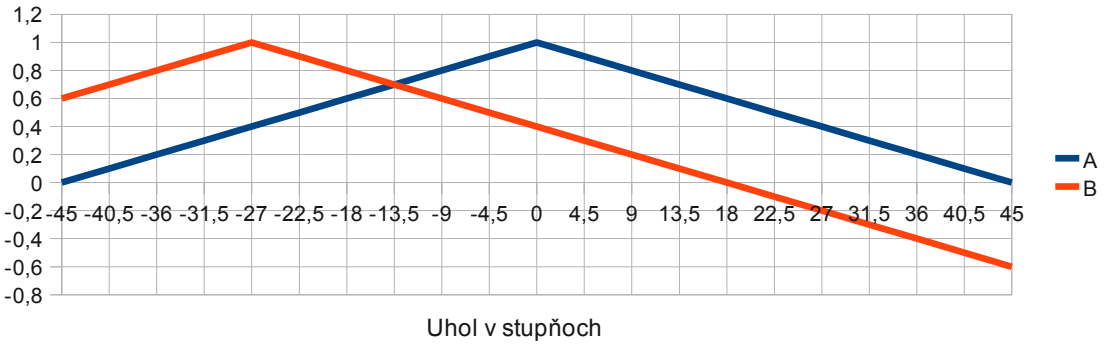
Z dôvodu absencie modulu rozpoznávania prekážok, sa účel tohto modulu zúžil na spájanie výsledkov len z modulu rozpoznávania ciest a modulu lokalizácie na mape.

Z modulu rozpoznávania ciest dostaneme vektor ohodnotenia smerov v rozsahu <-45°, 45°>, a z modulu lokalizácie optimálny smer v rozsahu <-180°, 180°>. Keď sa bude treba otočiť podľa lokalizácie mimo rozsahu distribúcie smerov, potom bude najlepšie zastaviť sa na mieste a otočiť

sa, a potom pokračovať ďalej.

Z vektoru ohodnotenia smerov, tiež chceme vylúčiť možnosti, ktoré sú v každom prípade neprijateľné. Napríklad robot má pred sebou len trávu kde nemôže ísť, aj keby optimálny smer z lokalizácie bol ísť rovno, treba nájsť inú trasu. Na riešenie tohto problému budeme prahovať distribúciu ohodnotenia smerov. Určíme si prah, pod ktorým už všetky hodnoty nastavíme na 0.

Ideálny smer určený modulom lokalizácie a plánovania prevedieme na funkciu, kde každému smeru priradíme hodnotu. Optimálny smer bude mať hodnotu 1 a na obe strany bude funkcia lineárne klesať.



Obrázok 19: Príklad grafu funkcie optimálneho smeru. A – pre smer 0°, B – pre smer -27°

Teraz vynásobíme distribúciu ohodnotenia smerov hodnotou z funkcie optimálneho smeru. Z tejto výslednej distribúcie vyberieme smer, kde je maximum a upravíme aktuálne rýchlosti motorov robota, tak aby sa približoval k tomuto smeru.

2.4 Realizácia

2.4.1 Stavba Robota

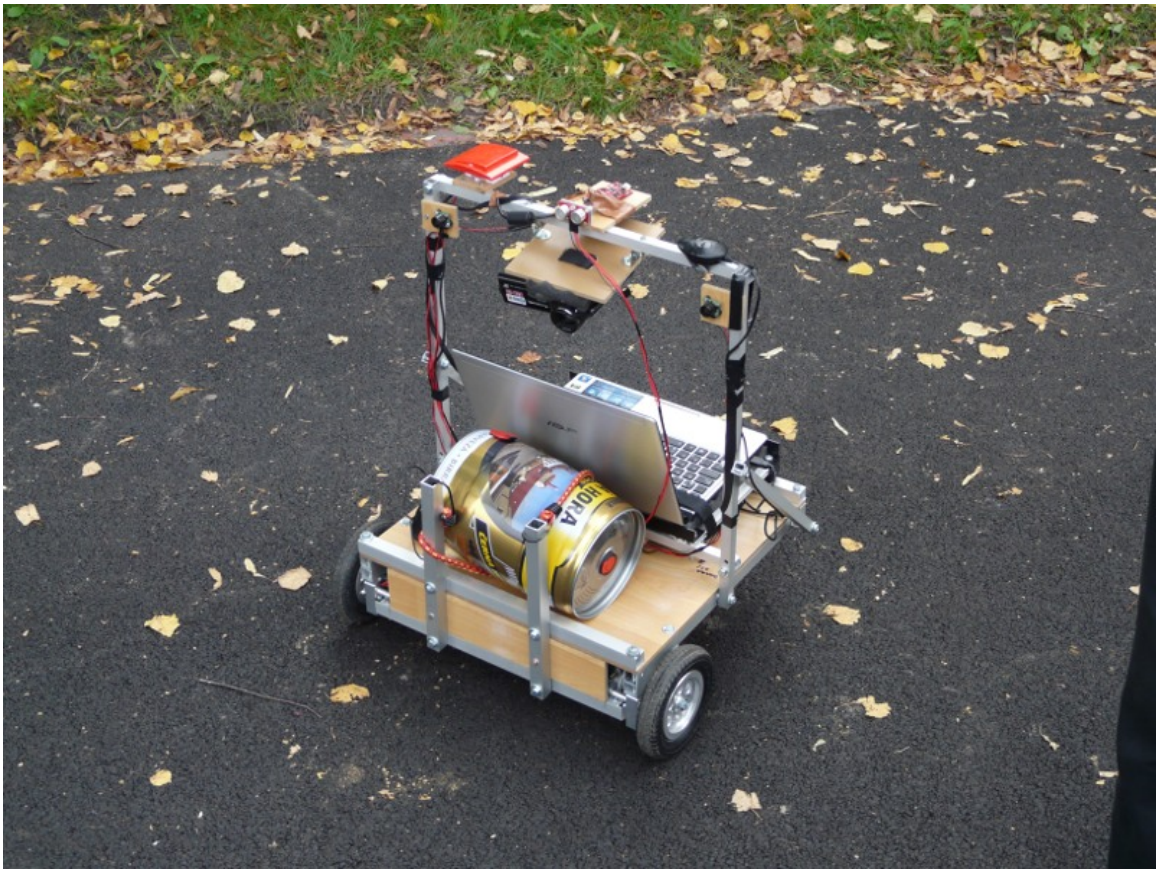
Celá konštrukcia je poskladaná z hliníkových profilov štvorcovej podstavy o šírke 45cm, na ktorú je pripevnená drevovláknitá doska. Kamera, kompas, ultrazvukové senzory, tlačidlo okamžitého zastavenia a GPS sú namontované na ráme vo výške asi 50cm od základne konštrukcie.

Obrázok 20: Ukážky zo stavby robota

Ako vidieť na obrázku 20 vľavo dole je ešte kamera nad rámom, a na obrázku 20 vpravo dole je už pod. Zmenili sme to kvôli pozorovaciemu uhlu kamery, Chceli sme zachytiť aj miesto pred robotom, aby mohol presne vedieť obchádzať malé prekážky a presne vidieť okraj cesty v zákrute. Taktiež sme chceli aby na kamere bol vidieť horizont, a to je výhodné v prípade dlhej rovnej cesty. Kamera bola aj inak upevnená pretože sa príliš triasla, keď robot jazdil po nerovnostiach. Pridali sme pod ňu penu proti otrasom.

Batéria sa nachádza pod drevenou doskou vpredu medzi kolesami, aby ťažisko bolo čo najnižšie. Pod laptopom je miesto na elektroniku a káble. Po bokoch sú kontroléry motorov, ktoré sme namontovali smerov von, tak aby ich aktívne chladenie bolo účinné.

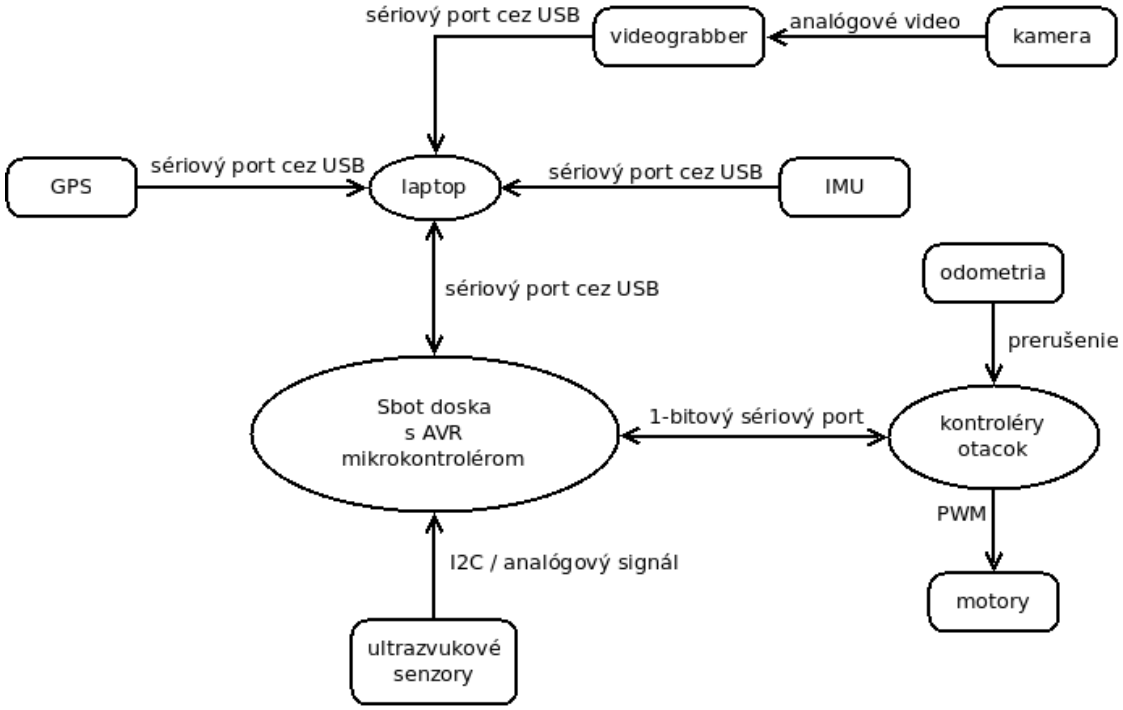
Laptop má vytvorené lôžko pomocou hliníkových a plastových profilov, aby sa neuvoľnil pri prejazde nerovnosťami. Je tiež uchytený gumou a suchým zipsom.



Obrázok 21: Robot po dokončení, foto zo súťaže Robotour

Súdok s pivom bol uchytený gumovými popruhmi o pripravenú hliníkovú konštrukciu v prednej časti robota.

2.4.2 Hardvérová architektúra robota



Obrázok 22: Zapojenie a komunikácia elektronických súčiastok

Použité súčiastky

Pohon:

Kolesá s motormi:

Rozhodol som sa pre *Parallax Motor Mount and Wheel Kit with Position Controller*, pretože veľkosť kolies, sila motorov zodpovedala našim požiadavkám. Taktiež odpadli problémy so skladaním vlastného kompletu, čo by bolo časovo náročnejšie.

Kontroléry k motorom:

Priamo doporučené výrobcom pre motory - *Parallax HB-25 Motor Controller*.

Zadné otočné kolesá:

Ku kolesám s motormi od Parallax-u existuje originálne zadné otočné koliesko. Na naše potreby však malo malý priemer, síce bolo nafukovacie, ale bolo dostupné iba za vysokú cenu. Hľadal som nafukovacie kolesá s otočným uchytením, ale potrebnej veľkosti som nafukovacie nenašiel. Preto sme použili gumové koleso od slovenského výrobcu, ktoré dostačuje na to, aby robot nezapadol do nerovností.

Senzory:

Ultrazvukové senzory na meranie vzdialenosti:

Nemali sme dostatočný rozpočet na laserový skener, takže sme na detekciu prekážok zvolili ultrazvukové senzory. Dva *MaxBotix MaxSonar EZ1* a jeden *Devantech SRF08*.

Kamera:

Hľadali sme model, ktorý by mal optický stabilizátor, kvôli nerovnostiam na cestách a CCD optický senzor, pretože pri použití senzorov CMOS dochádza k postupnému snímaniu obrazu po riadkoch a následne k jeho skresleniu, ak sa kamera pohybuje. Kamery ktoré boli určené na robotiku boli však príliš drahé, a tak sme siahli po *Panasonic SDR-T50* s kvalitnou optikou za dostupnú cenu. Dnešné camcordery bohužiaľ spravidla nedisponujú on-line digitálnym video výstupom a preto sme museli použiť analógový signál a videograbber, konkrétne cenovo dostupný *AVerMedia DVD EZMaker USB Gold*, ku ktorému sme našli ovládač aj pre Linux.

Kompas:

Elektronický kompas má tú nevýhodu, že pri naklonení dáva zlé hodnoty. Existujú aj kompasy s kompenzáciou náklonu. Cena celého IMU, nebola oveľa vyššia ako samostatného kompasu s kompenzáciou a tak sme použili *SparkFun 9DOF Razor IMU*, ktorý okrem 3-osého kompasu obsahoval aj 3-osý gyroskop a 3-osý akcelerometer, čo by sa neskôr mohlo hodiť.

Neskôr sme však použili kompas s kompenzáciou náklonu, pretože s výsledkami IMU sme neboli spokojní.

GPS:

Navilock NL-302U, štandardný model s USB a čipom SIRF3.

Enkodéry:

Zabudované v kolesách s motorom.

Ostatné komponenty:

Mikrokontrolér:

Pôvodne som plánoval použiť dosku *Arduino Duemilanove*, no nakoniec nám ju dodávateľ nedokázal ani po opakovaných urgenciách dodať takže sme použili dosku pre robot Sbot, ktorá na naše účely postačovala [Gustafík, 2008]. Doska obsahovala mikrokontrolér ATMega128.

Núdzové vypínacie tlačidlo:

Klasický veľký vypínač (používaný na elektroinštaláciu), nafarbený na červenou farbou.

Laptop:

Bolo potrebné aby bol čo najľahší s dlhou výdržou batérie ale aj dosť výkonný. Preto sme zvolili model *Asus UL30Vt*. Laptop má 13,3 palcový displej, dvojjadrový procesor Intel SU7300 Core 2 Duo, pamäť 4GB, výdrž 12 hodín, váhu 1,3 Kg a grafickú kartu NVidia G210M. Tú je možné použiť na akceleráciu rôznych výpočtov. Vzhľadom na fyzické otrasy počas jazdy robota po nerovnom povrchu sme pôvodne osadený pevný disk nahradili diskom SSD, ktorý na ne nie je citlivý.

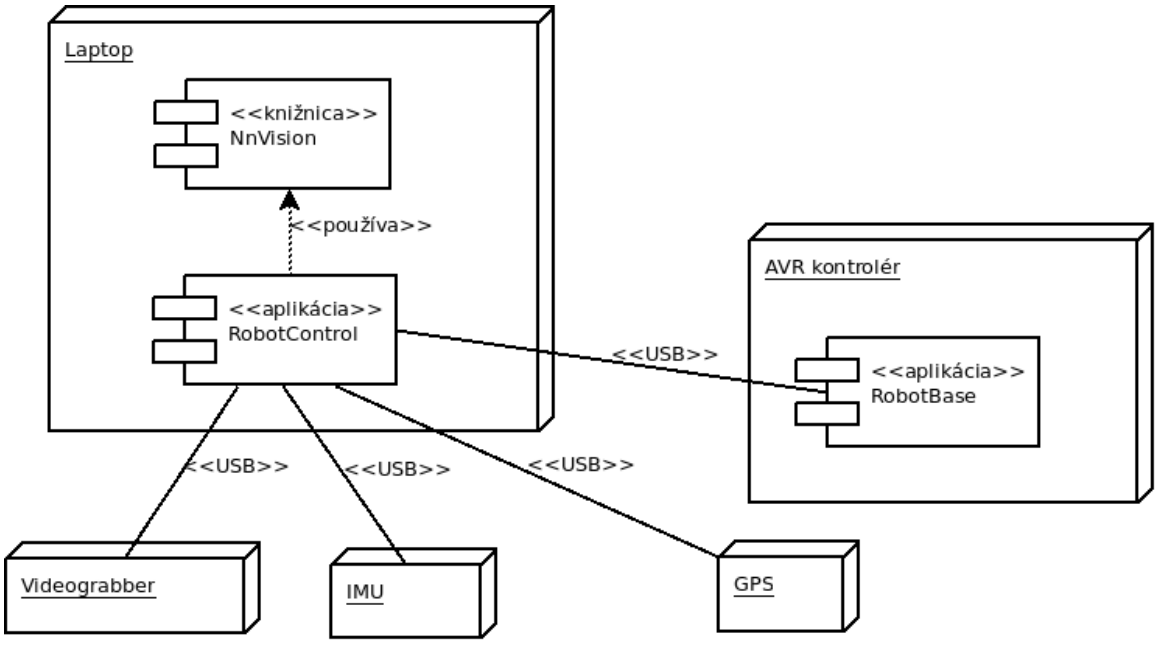
Gamepad:

Genius MaxFire G-12X, na pohodlné ovládanie robota na zbieranie dát. Nakoniec však mal dosť krátky dosah, asi meter, čo nedostačovalo našim potrebám.

Batéria:

HAZE HZS 12-9, AGM akumulátor pre elektrický pohon, 12V, 9Ah, s hmotnosťou 1,6 Kg.

2.4.3 Softvérová architektúra



Obrázok 23: Diagram zariadení, komponentov a komunikácie medzi nimi.

Softvér beží na dvoch zariadeniach:

Laptop

Ako základ poslúžil operačný systém Linux, distribúcia Ubuntu 10.04. Všetky komponenty boli

napísané v jazyku C++.

Softvérové komponenty môžeme rozdeliť do troch kategórií:

- aplikácia bežiaca v autonómnom móde robota: RobotControl
- pomocné aplikácie: aplikácia na trénovanie siete, aplikácia na ohodnocovanie obrázkov
- zdieľaná knižnica: NnVision

Vo všetkých softvérových komponentoch som používal na reprezentáciu a spracovanie obrázkov / matíc OpenCV knižnicu (verzia 2.1) [Bradski, 2008]. Na prácu s neurónovými sieťami knižnicu FANN (verzia 2.1) [Nissen, 2003].

AVR mikrokontrolér

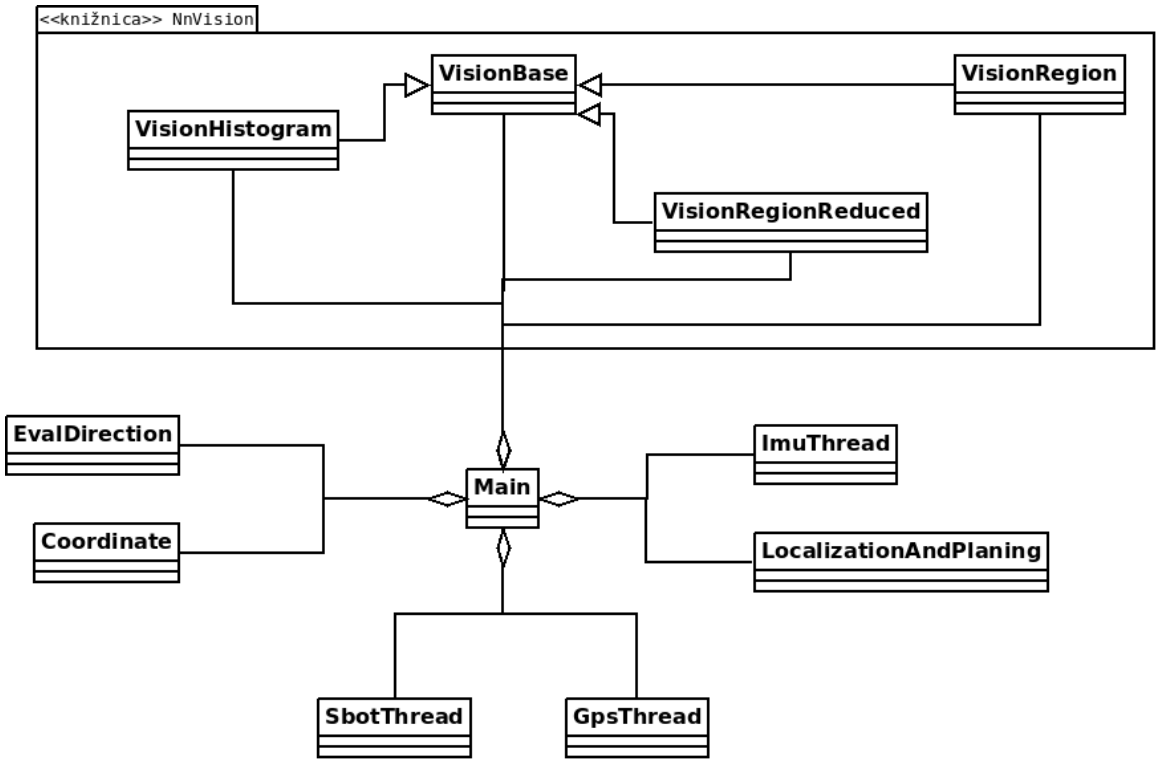
Hlavný program pre mikrokontrolér bol napísaný v jazyku C.

2.4.4 Popis softvérových komponentov

Všetky naprogramované komponenty sa nachádzajú na elektronickej CD prílohe práce.

RobotControl

Aplikácia na autonómne ovládanie robota, ktorá zahŕňa implementácie softvérových modulov robota popísaných v kapitole 2.1 . Pre plnohodnotné využitie dvojjadrového procesora a neblokovanie vstupov/výstupov je aplikácia viac-vláknová.



Obrázok 24: Diagram závislostí tried v aplikácii RobotControl

SbotThread

Trieda ktorej hlavná časť beží v samostatnom vlákne. Slúži na komunikáciu s Sbot AVR mikrokontrolérom podľa protokolu komunikácie popísaného nižšie v tejto kapitole.

ImuThread

Trieda, ktorá v samostatnom vlákne komunikuje s elektronickým kompasom.

GpsThread

Trieda, ktorá v samostatnom vlákne komunikuje s GPS prijímačom. Použili sme a

GPS zariadenie komunikuje s Laptopom pomocou USB. Priamo GPS čip má výstup RS-232 port, ktorý je následne konvertovaný na USB pomocou FTDI čipu. Použité jadro Linuxu obsahuje ovládač na zariadenie FTDI.

Skúšali sme použiť *gpds*, systémová služba pre Linux na komunikovanie s GPS. Toto nám robilo problémy pretože vždy pri pripojení nejakého iného USB FTDI zariadenia, služba sa vyskúšala pripojiť na toto zariadenie. To nám zničilo komunikáciu s ostatnými zariadeniami.

Ďalší problém je že máme viacero takýchto zariadení (Kompas, Sbot doska) a nevedeli sme ich rozlíšiť(ktorý identifikátor zariadenia v operačnom systéme zodpovedá ktorému zariadeniu).

BindSerialPorts

Trieda na zisťovanie identifikátorov zariadení. Použili sme program *gpsctl* z balíka *gpsd* vie rozpoznať GPS zariadenie v operačnom systéme. Ďalšie zariadenia mali vždy iný baudrate a po vyskúšaní komunikácie sme overovali, či dané zariadenie vrátilo validné dáta.

EvalDirection

Trieda s implementáciou ohodnotenia smeru pohybu z matice výstupov siete. (kapitola 2.2.3)

LocalizationAndPlanning

Základom lokalizácie je mapa. Tá bola spoločná pre všetky tímy a nachádzala sa na openstreetmap.org (teraz je už pravdepodobne zmenená). Bola reprezentovaná jednoduchým XML formátom. Na parsovanie tohto XML súboru sme použili knižnicu RapidXML.

Na výpočet uhlu o ktorý sa má robot otočiť aby šiel presne podľa naplánovanej ceste, bol treba údaj z kompasu. Celá mapa ako aj naplánovaná trasa a pozícia robota boli zobrazené na obrazovke.

Coordinate

Trieda s implementáciou modulu koordinácia.

Main

Hlavná trieda, ktorá len spája funkčnosť predošlých tried. Na začiatku sa naštartujú vlákna na zberanie dát. Potom sa v cykle vykonáva:

1. Vyhodnotenie obrázka neurónovou sieťou pomocou knižnice NnVision
2. Ohodnotenie smerov pohybu použitím triedy EvalDirection
3. Pozbieranie dát z SbotThread, GpsThread, ImuThread
4. Plánovanie akcie pomocou triedy Coordinate
5. Poslanie príkazu akcie pre RobotBase cez SbotThread

NnVision

Knižnica ktorú som vytvoril na rozpoznávanie vzoriek z obrazu pomocou neurónových sietí. Obsahuje 4 triedy, každá s iným variantom (kapitola 2.2.1 a 2.2.2).

Obsahuje implementáciu funkcionality:

- trénovania sietí
- predikcie sieťou
- vytvárania trénovacích príkladov z obrázku jeho masky

- konvertovanie dát v poli na obrázky/matice a naopak.

V OpenCV knižnici sa tiež nachádza implementácia MLP neurónových sietí a algoritmov učenia. Nanešťastie vo verzii ktorú som používal ešte obsahovala veľké množstvo chýb. Nakoniec som použil knižnicu FANN, ktorá fungovala výborne a má dobrú dokumentáciu.

Aplikácia na ohodnocovanie obrázkov

Na ohodnocovanie obrázkov som napísal jednoduchú aplikáciu. Užívateľ v nej má možnosť označiť oblasť na obrázku ktorá je cestou. Zvyšok nie je cestou. Výsledok, binárna matica, sa ukladá do XML súboru a má rovnaký rozmer ako pôvodný obrázok.

Ďalej som do nej pridal funkcionalitu aby mohol užívateľ vyhodnotiť obrázok neurónovou sieťou pomocou knižnice NnVision a tiež ukladanie výstupov do súboru.

Aplikácia na trénovanie siete

Vstupom pre túto konzolovú aplikáciu je zložka s obrázkami z videa, k nim prislúchajúce masky a parametre siete. Najskôr sa vygenerujú trénovacie príklady, potom sa na nich trénuje sieť. Počas trénovania sa zaznamenávajú údaje o trénovaní. Na konci trénovania sa natrénovaná sieť uloží do súboru.

Aplikácia využíva knižnicu NnVision.

RobotBase

Obsahuje funkčnosť na ovládanie pohybu kolies, čítanie údajov zo senzorov a tiež detekciu prekážok a tlačidlo na okamžité zastavenie.

Ovládanie pohybu nebolo také jednoduché ako sme predpokladali, originálny firmvér na kontroléroch otáčok nebol stavaný na dynamické menenie rýchlosti, čo sme nutne potrebovali. Po dostatočnom množstve experimentov sme našli vyhovujúce riešenie.

Ďalej čítame údaje z ultrazvukových senzorov z I^2C zbernice a analógových pinov (rôzne senzory majú rôzne výstupy), a aj pozície enkodérov, a aktuálnu rýchlosť.

Detekcia prekážok bola jednoduchá, ak sa pred robotom nachádzala prekážka vo vzdialenosti menšej ako 50cm, tak robot zastavil. Bolo výhodné túto funkcionalitu implementovať na nízkej úrovni v tomto komponente, aby v prípade zlyhania komunikácie robot aj tak zastavil pred prekážkou.

Protokol komunikácie aplikácie RobotControl a RobotBase

Príkazy smerujúce z RobotControl aplikácie do RobotBase aplikácie:

Popis	**Formát**	**Rozsah hodnôt**
Nastavenie rýchlosti pohybu	„s %d;“	%d je číslo z intervalu <-10;10>
Nastavenie smeru pohybu	„d %d;“	%d je číslo z intervalu <-100;100>
Odblokovanie po stlačení veľkého červeného tlačidla	„u;“	
Zapnutie ignorovania prekážok	„i;“	
Vypnutie ignorovania prekážok	„o;“	

Pravidelne odosielaná správa smerujúca z RobotBase do RobotControll:

„@LKROK PKROK LRYCHLOST PRYCHLOST ZABLOKOVANY PREKAZKA LVZDIAL SVZDIAL PVZDIAL“

Frekvenciu odosielania správ sme nastavili tak, aby nedochádzalo k zahlteniu komunikácie na sériovom porte, ktorý komunikoval na 115200 bps.

Časť správy	**Popis**
LKROK	Hodnota počítadla krokov z ľavého enkodéru
PKROK	Hodnota počítadla krokov z pravého enkodéru
LRYCHLOST	Rýchlosť ľavého motoru
PRYCHLOST	Rýchlosť pravého motoru
ZABLOKOVANY	1 – zablokovaný červeným tlačidlom, 0 - nezablokovaný
PREKAZKA	1 – prekážka pred aspoň jedným senzorom, 0 – nie je prekážka
LVZDIAL	Vzdialenosť prekážky od ľavého ultrazvukového senzoru
SVZDIAL	Vzdialenosť prekážky od stredného ultrazvukového senzoru
PVZDIAL	Vzdialenosť prekážky od pravého ultrazvukového senzoru

2.5 Výsledky

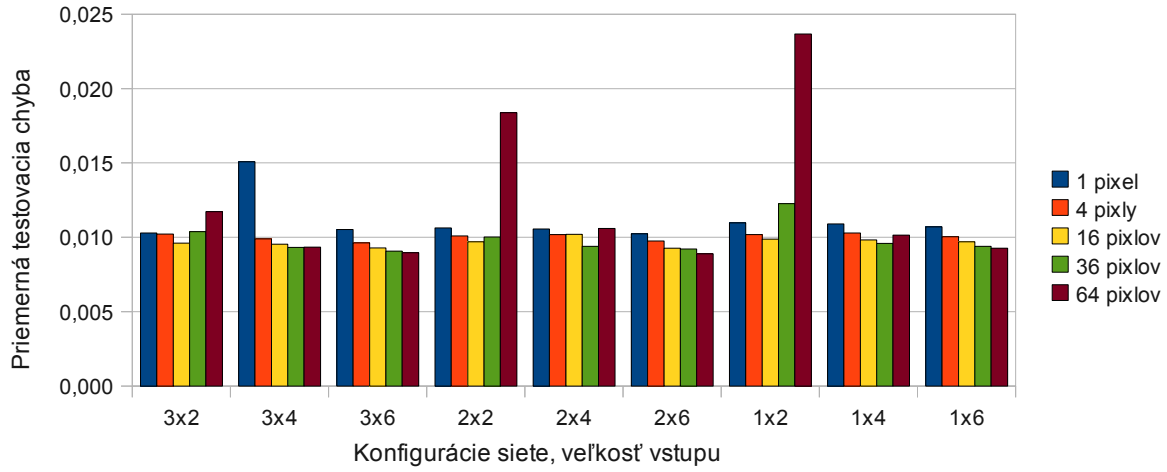
V tejto časti uvedieme výsledky získané z trénovania neurónovej siete na rozpoznávanie cesty z obrazu. Detailný opis tohto procesu sa nachádza v kapitole 2.2.1 .

Natrénujeme viacero konfigurácií siete, čiže rôzny počet skrytých neurónov a skrytých vrstiev. To preto aby sme zistili či treba zvýšiť komplexitu siete, aby sa mohla naučiť zložitejšie vlastnosti. Bola použitá krížová validácia, a skoré zastavenie trénovania.

2.5.1 Veľkosť vstupu

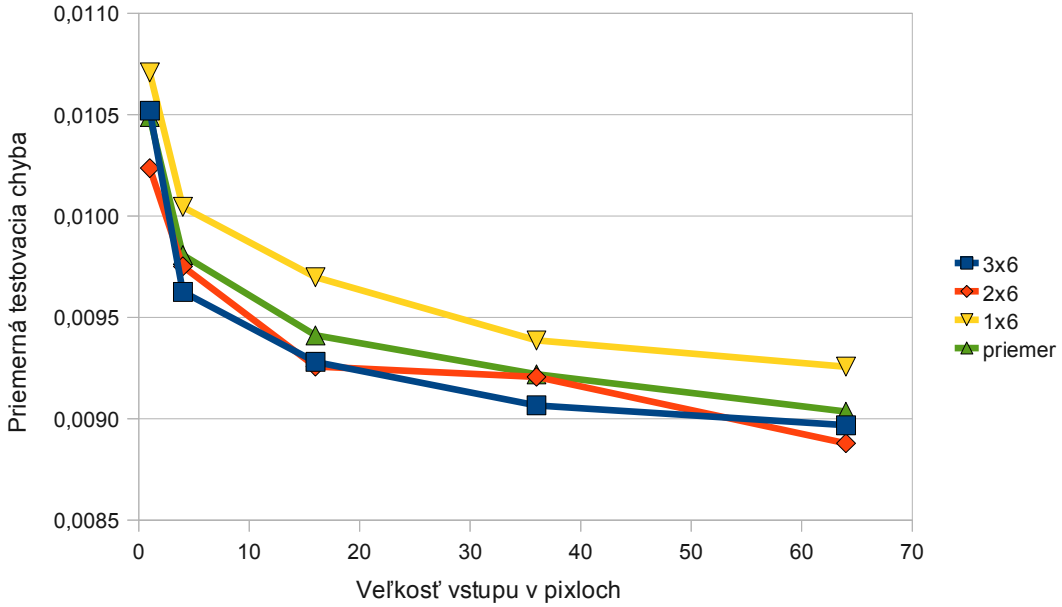
Začneme najjednoduchšou variantou, ktorá má na vstupe len RGB hodnoty pixlov v segmente. Každý jednotlivý príklad sme opakovali 10 krát a v grafe sú vždy priemerné údaje z týchto 10-tich pokusov.

Pokúsime sa zistiť ako vplýva veľkosť vstupu(segmentu) v závislosti od chyby a zároveň aj ako si počínali jednotlivé konfigurácie siete. Tie sú označené ako (počet skrytých vrstiev)x(počet neurónov v každej skrytej vrstve), čiže 2x6 je sieť s dvoma vrstvami skrytých neurónov, spolu teda 12 skrytých neurónov.



Obrázok 25: Graf závislosti testovacej chyby od konfigurácie siete a veľkosti vstupu

Z obrázku 25 je vidieť, že jednoduché siete ako 1x2 a 2x2, mali problém s veľkým vstupom. Je tiež vidieť že okrem predchádzajúcich prípadov, so stúpajúcou veľkosťou vstupu klesá testovacia chyba, avšak rastie výpočtová náročnosť.



Obrázok 26: Detail priebehu vývinu testovacej chyby vzhľadom na veľkosť vstupu. Tri najlepšie konfigurácie siete a ich priemer. (Pozor, posunutá y-ová os)

Typ / veľkosť vstupu	1	4	16	36
3x6	0,010520	0,009626	0,009281	0,009066
2x6	0,010237	0,009751	0,009257	0,009207
1x6	0,010708	0,010045	0,009698	0,009386
priemer	0,010488	0,009807	0,009412	0,009220

Tabuľka 1: Hodnoty testovacej chyby pre 3 najlepšie konfigurácie a priemeru.

Z obrázku 26 je vidieť že pri zdvojnásobení vstupu z 36 na 64, nedosahujeme významné zlepšenie, preto som sa rozhodol ďalej pracovať so vstupom veľkosti 36 ako postačujúcim.

2.5.2 Vstup oblasti s veľkou pravdepodobnosťou výskytu cesty ako prídavný vstup

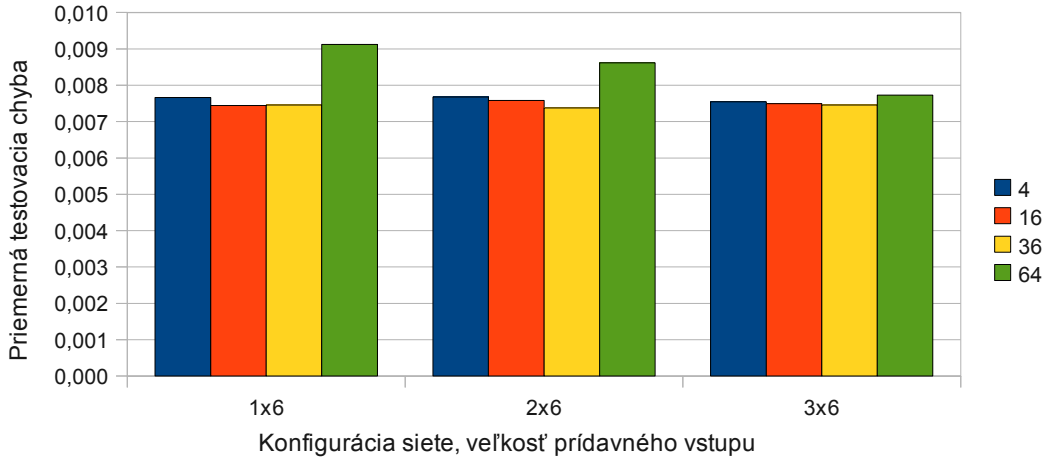
Ďalej sme používali na trénovanie len 3 najlepšie konfigurácie 1x6, 2x6, 3x6.

Tu boli dve možnosti pridania ďalšieho vstupu:

1. vstup priamo RGB hodnoty segmentu s veľkou pravdepodobnosťou výskytu cesty
2. vstup ako rozdiel (SQDIFF) segmentu s veľkou pravdepodobnosťou výskytu cesty so vstupným segmentom

V prvej možnosti som použil len obmedzený vstup do veľkosti 8x8 pixlov, pretože potom už trénovanie bolo príliš pomalé lebo sieť mala veľa váh. Hodnoty testovacích chýb tiež dosť

oscilovali, tak sme zvýšili počet trénovaní, z ktorých sme robili priemer, z 10 na 20. A tak isto aj vo všetkých nasledujúcich meraniach.



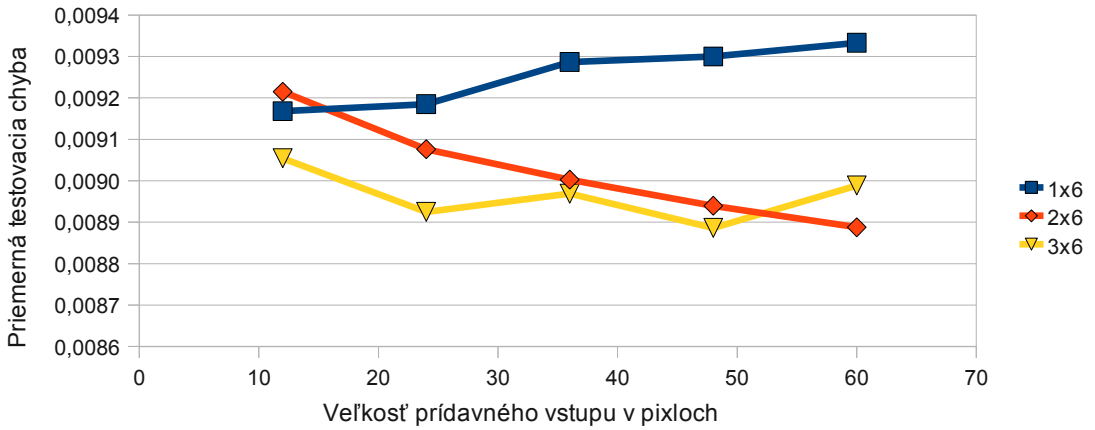
Obrázok 27: Graf závislosti testovacej chyby od konfigurácie siete a veľkosti prídavného vstupu, pre priamy vstup RGB hodnôt prídavného segmentu

Na obrázku 27 vidíme výsledky, z ktorých vieme povedať že pridanie ďalších vstupov pomohlo. Dokonca, s väčším prídavným vstupom chyba klesá okrem prídavného vstupu veľkosti 64. Konkrétne hodnoty sú v tabuľke 2.

Typ / veľkosť prídavného vstupu	4	16	36	64
1x6	0,0076653	0,00744075	0,0074559	0,0091182
2x6	0,00768035	0,0075821	0,0073773	0,00862
3x6	0,00754525	0,0074915	0,00746005	0,0077309

Tabuľka 2: Hodnoty testovacej chyby pre prídavný vstup pomocou RGB hodnôt

V druhom prípadne som použil oveľa väčšie veľkosti segmentu s veľkou pravdepodobnosťou výskytu cesty, pretože vždy mala sieť len jeden prídavný vstup, takže trénovanie bolo rýchle.



Obrázok 28: Graf závislosti testovacej chyby od veľkosti prídavného vstupu, pre vstup ako SQDIFF od aktuálneho vstupného segmentu

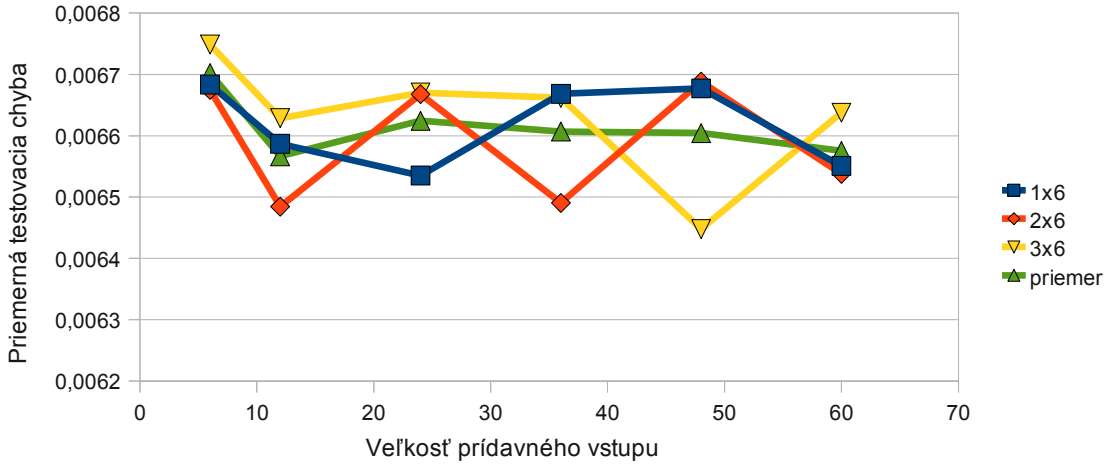
Z grafu na obrázku 28 je vidieť že najlepšie si počínali 2x6 a 3x6 konfigurácie siete. Tieto prídavné informácie zjavne nepomohli k lepšiemu natrénovaniu, v porovnaní s chybami sietí bez prídavného vstupu. Možno by sme dosiahli menšiu chybu, avšak pri väčšom prídavnom vstupe.

Typ / veľkosť prídavného vstupu	12	24	36	48	60
1x6	0,00916805	0,009185	0,0092867	0,00930015	0,00933305
2x6	0,00921495	0,0090762	0,00900285	0,00893985	0,00888785
3x6	0,0090544	0,00892495	0,008969	0,00888635	0,00898875

Tabuľka 3: Hodnoty testovacej chyby pre prídavný vstup pomocou SQDIFF

2.5.3 Histogram farieb ako prídavný vstup

Veľkosť vstupu pri histograme znamená, koľko farieb dokáže rozlíšiť. Pre každú farebnú zložku (RGB) bol počítaný samostatne, čiže počet vstupov ktoré pribudli sieti, je 3*(veľkosť vstupu)



Obrázok 29:Graf závislosti veľkosti vstupu od testovacej chyby, pre prídavný vstup pomocou histogramov farieb

Ako vidieť na obrázku 29, výsledky značne oscilovali. Ak sa však pozrieme na hodnoty zistíme že tento typ prídavného vstupu dosahuje najmenšie chyby.

Typ / veľkosť Prídavného vstupu	6	12	24	36	48	60
1x6	0,00668345	0,00658685	0,006535	0,00666825	0,0066771	0,0065507
2x6	0,00667465	0,0064843	0,0066677	0,00649045	0,00668825	0,00653785
3x6	0,00674855	0,00662855	0,0066704	0,00666175	0,0064478	0,00663755

Tabuľka 4:Hodnoty testovacej chyby pre prídavný vstup pomocou histogramov farieb

2.6 Záver

V tejto práci sme sa zamerali na vytvorenie funkčného systému autonómneho robota pre súťaž Robotour podľa zadania. Navrhli sme riešenie, ktoré bolo následne realizované.

S robotom sme sa zúčastnili na súťaži Robotour v septembri 2010. V tomto čase ešte softvér robota nebol úplne dokončený, napriek tomu sme úspešne dokázali prejsť homologizáciou. Veľa problémov nám robila hardvérová stránka robota, no teraz už máme plne funkčného robota, pripraveného na ďalší ročník súťaže.

V čase konania súťaže tiež prebiehala konferencia Robotics in Education 2010 (http://rie2010.stuba.sk), na ktorej sme prezentovali článok o tomto projekte [Nadhajský, 2010].

Navrhli sme spôsob rozpoznávania ciest z obrazu pomocou neurónových sietí. Následne sme z tohto výstupu siete ohodnotili jednotlivé smery pohybu. Tento systém fungoval, no ešte sme ho vylepšili prídavným vstupom a to konkrétne histogramami farieb.

Robot má naimplementovanú navigáciu pomocou mapy, GPS a elektronického kompasu. Vie naplánovať trasu a v prípade vybočenia z trasy prepočíta novú.

Kombinácia rozpoznávania ciest a navigácie robota, pomocou modulu koordinácie následne ovláda pohyb robota.

Námety na ďalšiu prácu:

1. Rozpoznávanie ciest pomocou prídavného vstupu sieti ako priemerná farba pixla, použité v [Shinzato, 2010].
2. Filtrovanie pozície pomocou kombinácie údajov z GPS a odometrie, čo môže pomôcť lokalizácií, aj keď GPS nemá signál, prípadne príliš slabý.
3. Zapracovaním pravdepodobnostnej lokalizácie by sme sa mohli dopracovať k lepším výsledkom.
4. Implementácia obchádzania prekážok, na ktorú už máme pripravených 5 ultrazvukových senzorov (namiesto pôvodných troch).

Prínos práce:

Vyskúšali sme nový spôsob rozpoznávania neurónových sietí, kombináciou rozdelenia vstupného obrázku na segmenty a priamym vstupom RGB hodnôt do siete. Tiež sme navrhli viacero typov prídavných vstupov siete, ktoré prinášajú globálnu informáciu o vstupnom obrázku.

Robot ktorého sme postavili pre účely tejto práce, je naďalej použiteľný pre budúce experimenty.

2.7 Zoznam bibliografických odkazov

1. Murphy R.R. Introduction to AI Robotics. The MIT Press. 2000. ISBN: 0-262-13383-0
2. Russel S., Norvig P. Artificial Inteligence A Modern Approach. Prentice Hall. 1995. ISBN: 0-13-103805-2
3. Haykin S. Neural Networks – A Comprehensive Foundation. Second Edition. Prentice-Hall. 1999. ISBN: 81-7808-300-0
4. Kvasnička V., Beňušková., Pospíchal J., Farkaš I., Tiňo P., Kráľ A. Úvod do teórie neurónových sietí. Iris. 1997.
5. Igel C., Husken M. Improving the Rprop Learning Algorithm. 2000.
6. Pomerleau D.A. Neural Network Vision for Robot Driving. 1996.
7. Neto A.M., Rittner L. A simple and efficient Road Detection Algorithm for Real Time Autonomous Navigation based on Monocular Vision. 2006.
8. Álvarez J.M., Gevers T., López A.M. Vision–based Road Detection using Road Models. 2009.
9. Shinzato P.Y., Fernandes L.C., Osorio F.S., Wolf D.F. Path Recognition for Outdoor Navigation Using Artificial Neural Networks: Case Study. 2010.
10. Riedmiller M., Braun H. A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm. 1993.
11. Otsu N. A Threshold Selection Method from Gray-Level Histograms. 1978.
12. Gustafík D. SBot v2.0 – Educational Robot for Clubs and Classrooms, User manual to Sbot robot. 2008. online: http://webcvs.robotika.sk/cgi-bin/cvsweb/~checkout~/robotika/sbot/2.0/doc/dokumentacia_en.pdf
13. Bradski G., Kaehler A. Learnning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media. 2008.
14. Nissen S. Implementation of a Fast Artificial Neural Network Library (fann). 2003. online: http://freefr.dl.sourceforge.net/project/fann/fann_doc/1.0/fann_doc_complete_1.0.pdf
15. Nadhajský M., Petrovič P. Robotour Solution as a Learned Behavior Based on Artificial Neural Networks . 2010.