UNIVERZITA KOMENSKÉHO V BRATISLAVE

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

MOBILNÝ AUTONÓMNY ROBOT S POČÍTAČOVÝM VIDENÍM

Richard Jurík

2015

UNIVERZITA KOMENSKÉHO V BRATISLAVE

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

MOBILNÝ AUTONÓMNY ROBOT S POČÍTAČOVÝM VIDENÍM

Diplomová práca

Študijný program : Aplikovaná informatika

Študijný odbor: 9.2.9 Aplikovaná informatika

Školiace pracovisko: Katedra aplikovanej informatiky

Školiteľ: Mgr. Pavel Petrovič, PhD.

Bratislava 2015

Richard Jurík

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Richard Jurík
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: Mobilný autonómny robot s počítačovým videním / *Mobile Autonomous Robot with Computer Vision*

Cieľ: Úlohou je vytvoriť robotickú mobilnú platformu pre experimenty v umelej inteligencii s počítačovým videním nadväzujúc na bakalársku prácu vytvorenú v spoločnom robotickom laboratóriu, ktorej výsledkom bola učebná pomôcka - mobilný robot SBOT 2.0. Úlohou diplomanta je rozšíriť túto platformu o riadiacu jednotku s vyšším výpočtovým výkonom a kameru a navrhnúť, implementovať a zdokumentovať rámec pre experimenty v umelej inteligencii. Vytvorený rámec overiť na úlohe lokalizácie robota v prostredí pomocou počítačového videnia.

Literatúra: Stephen Se, David Lowe, Jim Little: Vision-based Mobile Robot Localization And Mapping
using Scale-Invariant Features. Proceedings of the 2001 IEEE International Conference on
Robotics & Automation, Soul, 2001.

Kľúčové slová: mobilný autonómny robot, počítačové videnie, lokalizácia

Vedúci: Mgr. Pavel Petrovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: doc. PhDr. Ján Rybár, PhD.

Dátum zadania: 28.11.2013

Dátum schválenia: 02.12.2013

prof. RNDr. Roman Ďurikovič, PhD.
garant študijného programu

.....
študent

Pavel Petrovič
vedúci práce

Čestné prehlásenie

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím uvedených zdrojov.

V Bratislave dňa .....

Richard Jurík

Poďakovanie

Ďakujem môjmu školiteľovi diplomovej práce, Mgr. Pavlovi Petrovičoci, PhD., za potrebné poznámky a rady, ktoré mi poskytol pri tvorení tejto práce.

Richard Jurík

Abstrakt

Diplomová práca nadväzuje na bakalársku prácu, ktorej výsledkom bola učebná pomôcka – mobilný robot Sbot 2.0. Túto platformu sme rozšírili o riadiacu jednotku s vyšším výpočtovým výkonom a kameru a tak vytvorili mobilnú robotickú platformu pre pokročilé experimenty v umelej inteligencii s počítačovým videním. Navrhli sme framework, implementovali ho a zdokumentovali jeho funkčnosť a spôsob použitia na rôznych experimentoch. Robot dokáže plniť v prostredí pomocou počítačového videnia a umelej inteligencie rozličné úlohy.

Framework je navrhnutý v štýle architektúry Behavior-Based Robotics, kde paralelne vykonávané správania komunikujú posielaním správ. Správania sú distribuované na dvoch hadvérových platformách prepojených sériovou linkou, pričom vytvorený framework zabezpečuje potrebnú abstrakciu a jednotné spoločné prepojené softvérovo-architektonické riešenie.

Výsledný robot a ľahko rozšíriteľný framework je použiteľný na výučbu robotiky alebo realizáciu nových cieľov umelej inteligencie.

Kľúčové slová: mobilný autonómny robot, počítačové videnie, neurónové siete

Abstract

This diploma thesis builds on a bachelor thesis, where an educational tool - a mobile robot SBot 2.0 was designed. We have extended this platform with a control unit with higher computational power and camera in order to create a mobile robotic platform for advanced experiments in Artificial Intelligence with Computer Vision. We designed and implemented a framework, and documented its functionality and usage on several experiments. The robot can solve various tasks in its environment using Computer Vision and Artificial Intelligence.

The framework follows the Behavior-Based Robotics architectural style with behaviors running in parallel and communicating using message passing. The behaviors are distributed on two hardware platforms interconnected by a serial line and the framework provides the necessary abstraction and a common unified interconnected software-architectural solution.

The resulting robot and the easily extensible framework are useful for robotics courses or the realization of new targets of Artificial Intelligence.

Key words: mobile autonomous robot, computer vision, neural network

Obsah

Úvod

V dnešnom elektronickom svete, kde už každodennou nevyhnutnou súčasťou človeka je nejaké elektronické zariadenie, sa dopopredia dostáva robotika. Jej nárast je obdivuhodný a môžeme ho porovnávať s nástupom počítačovej éry. Vo svete sa organizuje nespočetne veľa rôznych sútaží a výstav, ktoré sú dôležitými platformami pre výmenu informácií a skúseností a vývojári sa doslova predbiehajú vo vynálezoch. Práve preto sa programovanie robotov dostáva do popredia a venuje sa mu široká škála vyvojárov od odborníkov v špecializovaných organizáciach vrátane gigantov ako je Google, či donedávna Microsoft, cez univerzity až po amatérskych nadšencov robotiky. Niekoľkými zaujímavými projektami v tejto sfére sa môže pochváliť aj naša fakulta na Katedre aplikovanej informatiky.

Projekt s robotom e-puck sa aktuálne špecializuje na robotickú súťaž Myš v bludisku a využitie vizuálnej informácie, čo doposiaľ nik v tejto súťaži neurobil, avšak kvôli rozmerom a vysokej cene doplnkov je ťažko rozšíriteľný. Iným projektom je robot Smelý Zajko určený do robotickej súťaže Robotour, avšak vzhľadom na svoje rozmery vyžadujúci veľké priestory, alebo robot Ferdo-Mravec, ktorý využíva stereo-videnie, chápadlo a plánovanie na plnenie úloh vo svojom prostredí, užitočná, ale tiež rozmerovo rozsiahlejšia a špecifická platforma. Preto sme sa rozhodli vyvinúť malú, dobre rozšíriteľnú platformu a univerzálny framework, ktorý môže slúžiť na výuku robotiky, ale aj na realizáciu experimentov v umelej inteligencii.

Na prácu sme sa rozhodli využiť jednu z viacerých dostupných platforiem u nás na katedre, ktorou je robot Sbot. Robot je navrhnutý ako učebná pomôcka pre robotiku a pri dostupnosti viacerých kusov sa stal jasnou voľbou na realizáciu našej práce. Nakoľko výkon robota nie je na vysokej úrovni, budeme ho musieť rozšíriť o riadiacu jednotku s vyšším výpočtovým výkonom a pre širší rámec pôsobnosti robota doplníme kameru.

Teda cieľom našej práce je vytvoriť framework pre experimenty v umelej inteligencií s počítačovím videním pre robotickú platformu Sbot a rozšíriť ju o riadiaci systém s vyšším výpočtovým výkonom. Navrhnutý framework implementovať a zdokumentovať rámec experimentov. Ako riadiaci systém si môžeme zvoliť miniatúrnu platformu Gumstix, ktorá obsahuje dostatočnú vypočtovú silu potrebnú pre výpočty umelej inteligencie.

Rámec experimentov si môžeme rozdeliť do troch náročnostných kategórií. V prvej sa budeme zaoberať základnými vlastnosťami robota ako je testovanie pohybu, senzorov, ovládania a inými vlastnosťami zabezpečujúcimi správanie robota. V druhej kategórií budeme chcieť zdokumentovať experimenty týkajúce sa práce s počítačovým videním a so spracovaním obrazu z kamery. Nakoniec v tretej kategórií budeme implementovať príklady umelej inteligencie, kde na jej realizáciu si môžeme zvoliť napríklad použitie neurónových sietí. Na komunikáciu dvoch platforiem budeme musieť dôkladne premyslieť ich prepojenie a štruktúru pre komunikačné balíčky, pomocou ktorých budú spolu komunikovať.

Na záver zhodnotíme našu celkovú prácu, vyhodnotíme výsledky a navrhneme ďalšie nové implementácie pre robota.

1 Východisková kapitola

1.1 Teoretické východiská

Organizácia alebo architektúra softvéru u autonómneho robota má veľký vplyv na špecifické prístupy ovládania robota. V tejto kapitole si bližšie predstavíme niekoľko známych architektúr využívaných na prácu a riadenie mobilných robotov.

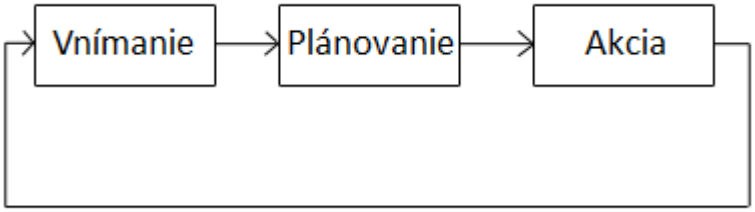
1.1.1 Čo to je architektúra robota

Architektúra robota sa využíva na označenie softvéru robotického systému. Architektúra reprezentuje štruktúru softvéru, akým spracováva senzorické vstupy, vykonáva kognitívne funkcie a poskytuje signály výstupným riadiacim jednotkám, nezávisle na tom, ako bola daná architektúra navrhnutá. Za definíciu architektúry robota považujeme definíciu uvádzanú v knihe o umelej inteligencií[1] v znení „The architecture of a robot defines how the job of generating actions from percepts is organized“.

1.1.2 Hierarchická architektúra

Hierarchická architektúra[2] je postavená na štruktúre v ktorej každá vrstva poskytne explicitné pokyny vrstve pod ňou. V niektorých prevedeniach hierarchickej architektúry môže vrstva komunikovať aj s viacerými nižšími vrstvami v hierarchií. Okrem toho, hierarchická architektúra si zachováva model sveta v ktorom sa robot pohybuje.

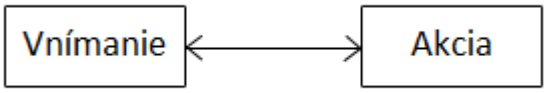
Základnou myšlienkou architektúry je, že robot najskôr vníma model sveta, potom plánuje ďalšiu akciu a následne vykonáva akciu (Obr. 1.1). Architektúra je zložitá na plánovanie, nakoľko robot v každom kroku plánuje nasledujúci pohyb.



Obr. 1.1: Schéma – Hierarchická architektúra

1.1.3 Reaktívna / subsumpčná architektúra

Subsumpčná architektúra bola navrhnutá a prvý-krát predstavená v roku 1986 Rodneym Brooksom[3]. Charakteristika architektúry spočíva medzi tesnou väzbou stavu vnímania a akcie, bez strednej kognitívnej vrstvy (Obr. 1.2). Architektúra si nezachováva model sveta, v ktorom sa robot pohybuje. Robot má viac inštancií väzieb Vnímanie-Akcia. Tieto väzby sú všeobecné procesy, takzvané správania.



Obr. 1.2: Schéma – Reaktívna / subsumpčná architektúra

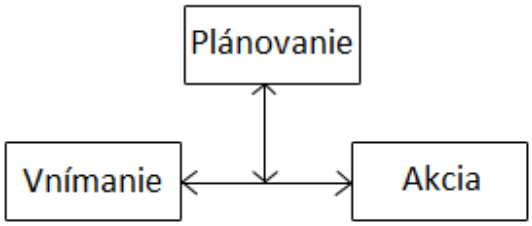
Všetky správania v subsumpčnej architektúre pracujú súčasne, paralelne a asynchrónne. Správania robota sú popísané v podobe pravidiel, ktoré uvádzajú reakciu robota pre daný vstup zo senzorov. Implementácia pravidiel je ponechaná na vývojára. Vyššie a zložitejšie správania zahŕňajú tie pod nim. Napríklad správanie preskúmania priestoru zahŕňa správanie vyhýbania sa prekážok. Je zrejmé, že nižšie vrstvy nemajú povedomie o úrovniach nad sebou v hierarchií, takže architektúra môže byť navrhnutá a postupne sa môžu pridávať nové vrstvy.

Základné vlastnosti architektúry:

- Situovanosť
  - zdôrazňuje skutočnosť, že robot je súčasťou svojho prostredia, že vníma svoje prostredie a koná na podnety výstupov zo senzorov
- Stelesnenosť
  - fyzický charakter robota, má konkrétne telo so svojími senzormi a aktuátormi a pomocou nich vníma zmeny prostredia a reaguje na ne
- Inteligencia
  - je tvorená reakciami systému vzhľadom na prostredie, nie je súčasťou vnútornej štruktúry
- Emergencia
  - správanie systému, ako celku, vzniká interakciou častí z ktorých sa systém skladá

1.1.4 Hybridná architektúra

Obidve vyššie popísané architektúry majú svoje výhody aj nevýhody. Hybridné architektúry sa vyvinuli z dôvodu prepojenia pozitívnych vlastností hierarchickej a reaktivnej architektúry (Obr. 1.3). Nižšia úroveň je reaktívna, kde senzory a aktuátory sú úzko spojené. Vyššia úroveň obsahuje hierarchický komponent, zahrňujúci kognitívne funkcie, plánovanie. Pri riešení zložitých úloh je možné tieto úlohy dekomponovať a niektoré časti riešiť reaktívne, iné pomocou plánovania. Príkladom hybridnej architektúry je napríklad INTERRAP[5]. Jeho hlavným rysom je, že v sebe spája vzory správania s explicitným plánovaním.



Obr. 1.3: Schéma – Hybridná architektúra

1.1.5 Behavior-based architektúra

Behavior-based architektúra[2] sa u autonómnych robotov v poslednom období využíva bežne. Koncepcia architektúry sa skladá z modulov, ktorým hovoríme správania. Každé toto správanie reprezentuje určitú úlohu, správanie sa robota v prostredí. Jednotlivé správania nezasahujú do koncepcie iných modulov a je na vývojárovi ako naimplementuje komunikáciu medzi danými správaniami. Na aktiváciu správaní rozlišujeme dva typy podmienok:

- podmienky prostredia
  - správanie sa aktivuje vzhľadom na aktuálny stav v prostredí
- podmienky sekvencie
  - aktiváciu spúšta iné správanie na základe jeho výstupov. Simuluje jednoduché plánovanie správania sa robota.

Z hľadiska histórie a vývinu behavior-based architektúra integruje výhody hierarchickej a reaktívnej architektúry vlastným distribuovaným spôsobom. Vzniknutá modularita je výhodou pre ďalší rozvoj systému, vyznačujúca sa veľkou dynamikou.

1.1.6 Umelá inteligencia a robotika

Čo rozumieme pod pojmom umelá inteligencia? Tento pojem zahŕňa velké množstvo teorií. Zjednodušeným zhrnutím by sme mohli povedať, že umelá inteligencia je vedná disciplína, ktorá sa prostredníctvom strojov, počítačov snaží napodobniť správanie človeka, alebo iných biologických druhov. Avšak výsledné správanie stroja nie je dopredu naprogramované, počítač by sa mal dopracovať k výslednému správaniu na základe spracovania informácií z prostredia.

Demonštrovali by sme to na príklade Turingovho testu[6]. Turingov test spočíva v tom, že pýtajúci sa človek a hodnotiaca komisia nevie rozhodnúť, či komunikácia prebieha medzi človekom a počítačom alebo medzi ľudmi. Teda počítač dopredu nevie aké otázky dostane ale mal by sa správať a odpovedať ako ľudská bytosť.

V snahe dosiahnuť, čo najlepšie inteligentné systémy využívame rôzne techniky učenia. Väčšiu pozornosť venujeme týmto témam:

- Učenie s učiteľom
- Učenie bez učiteľa
- Učenie odmenou a trestom
- Pravdepodobnostná robotika

1.1.6.1 Učenie s učiteľom

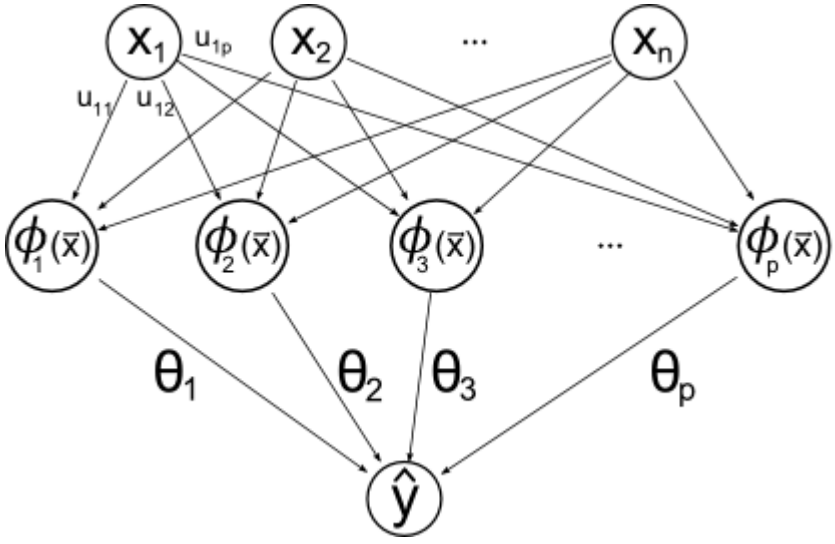
Učenie s učiteľom je métoda stojového učenia, ktorá využíva trénovacie dáta, aby bol systém schopný naučiť sa požadovaný výsledok zo vstupných údajov. Ak sa na trénovacie dáta dívame ako na funkciu výstupu vzhľadom na vstup, pričom obor hodnôt tejto funkcie je zväčša spojitý, ide o regresiu. Ak máme vopred stanovenú množinu tried a účelom zobrazenia je priradiť vstupným vektorom jednu z týchto tried, ide o klasifikáciu. V oboch prípadoch od učenia očakávame schopnosť generalizácie, čiže priradenia správnej výstupnej hodnoty aj na také vstupy, ktoré neboli súčasťou trénovacej množiny.

Generalizácia je možná vďaka skrytým pravidelnostiam učeného konceptu, tie ho odlišujú od bieleho šumu.

Bližšie sa budeme zaoberať neurónovými sieťami.

Neurónové sieťe

Sú výpočtovým modelom, ktorým sa snažíme napodobniť vlastnosti biologických neurónových sieti. Neurónová sieť sa typicky skladá z troch vrstiev, ktoré sú navzájom poprepájané (Obr. 1.4). Na vstupe je n atribútov, z ktorých pozostáva vstupná vrstva. Vstupná vrstva posiela vstupné hodnoty do druhej, skrytej vrstvy. Tá sa skladá z p neurónov a preberá ako vstup váhované výsledky zo vstupnej vrstvy. Každý neurón vykoná lineárnu kombináciu všetkých vstupov a výsledný súčet zobrazí cez prechodovú funkciu $\phi_i(\vec{x})$. Funkcia $\phi_i(\vec{x})$ je väčšinou nelineárna funkcia $R \rightarrow R$. Ako posledná je výstupná vrstva, ktorá obsahuje m neurónov. Každý neurón výstupnej vrstvy vykoná lineárnu kombináciu vstupov, ktoré sú výstupnými hodnotami skrytej vrstvy, s váhami θ. Váhy prepojení skrytej aj výstupnej vrstvy sa neurónová sieť učí, aby minimalizovala hodnotu chybovej funkcie.



Obr. 1.4: Neurónová sieť

Využívame, teda maticu U a vektor θ :

$$U = \begin{bmatrix} u_{1,1} & \dots & u_{1,n} \\ & \vdots & \\ & \vdots & \\ u_{p,1} & \dots & u_{p,n} \end{bmatrix} \theta = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_p \end{bmatrix} \tag{1.1}$$

K dispozícií máme množinu hypotéz H:

$$H = \{h_{\theta,U} : \vec{x} \to \theta^T \phi(U\vec{x})\} \tag{1.2}$$

Našou úlohou je nájsť také U a θ , že minimalizujeme chybovú funkciu J(U, θ):

$$J(U,\theta) = \sum_{i=1}^{t} (h_{u,\theta}(x_i) - y_i)^2 \tag{1.3}$$

Tu nastáva problem, pretože pre väčšinu funkcií θ nepoznáme uzavretý tvar. Na riešenie neurónových siete je popísaných viacero metód. Jednou z nich aj algoritmus – spätné šírenie chýb (backpropagation algorithm), ktorý popísali vo svojej práci Rumelhart, Hinton a Williams[7].

1.1.6.2 Učenie bez učiteľa

Pri učení bez učiteľa nemáme k diszpozícii trénovacie dáta, preto nemáme možnosť natrénovania nejakej hypotézy. Zmyslom tohto učenia je teda vyhodnocovať zhody vstupov podľa ich vlastností. Spomenieme dva prístupy vyhodnocovania. Prvému prístupu, kde dáta rozdeľujeme do zhlukov, hovoríme zhlukovanie. Pri zhlukovaní sa snažíme čo najlepšie rozdeliť vstupné dáta na základe daných kritérií. Druhý prístup nazývame redukcia dimenzionality. Jeho cieľom je vyselektovať nejakú významnú kombináciu atribútov veľarozmerného vstupného vektora. Atribúty v transformovaných dátach môžu predstavovať zastupujúcu hodnotu pre širšiu množinu dát.

1.1.6.3 Učenie odmenou a trestom

Učenie odmenou a trestom by sme mohli zaradiť niekde medzi učenie s učiteľom a učenie bez učiteľa. Hlavnou myšlienkou učenia odmenou a trestom je dobré akcie odmeňovať a zlé penalizovať. Úlohou je zistiť čo sa vykonalo správne a čo nie v sekvencií akcií. Systém by mal byť schopný učiť sa z predošlých akcií a hľadať nové v tej-ktorej situácii, resp. kontexte.

1.1.6.4 Pravdepodobnostná robotika

Za pomoci pravdepodobnostných reprezentácií vnímania, akcií a prostredia sme schopní s určitou chybou zväčšovať rozsah pôsobnosti v prostredí, ktoré nie je vopred úplne známe. Napríklad pravdepodobnosť, že zajtra bude slnečno je 0.8. Zapíšeme ako P(slnečno) = 0.8, táto pravdepodobnosť je nepodmienená.

Ak máme náhodné premenné A,B, podmienenú pravdepodobnosť zapíšeme P(A | B). Čo znamená aká je pravdepodobnosť A ak nastal B. Pravdepodobnosť za predpokladu, ak platí A aj B – vyjadríme ako P(A,B).

Na vyčíslovanie pravdepodobnosti náhodných premenných najčastejšie využívame Bayesovsku formulu. Majme náhodne premenné B,A , potom:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)} \tag{1.4}$$

Na pravdepodobnostné modelovanie systému sú často používané Bayesove siete alebo Skryté Markovove modely.

V našej práci sa s pravdepodobnosťou budeme môcť stretnúť pri experimentoch lokalizácie alebo mapovania, ktoré si predstavíme nižšie.

Lokalizácia

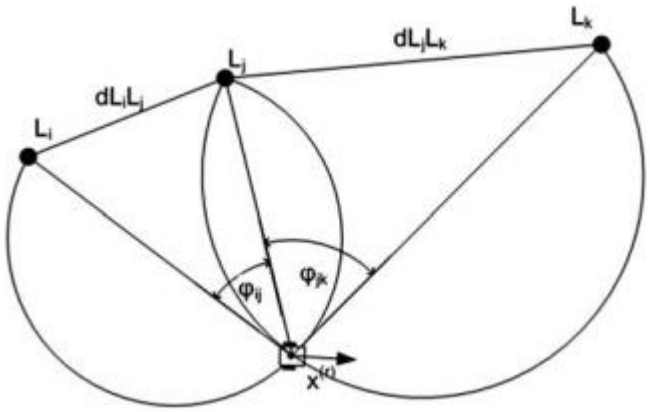
Problém lokalizácie rieši úlohu stanovenia polohy robota – buď vzhľadom na známu mapu (globálna lokalizácia), alebo vzhľadom na aktuálne objekty v prostredí robota (lokálna lokalizácia).

Pravdepodobnostné metódy sa tiež využívajú na lokalizáciu robota. Takúto lokalizáciu realizujeme pomocou nameraných hodnôt vzhľadom na prostredie, ktoré spracovávame pravdepodobnostnými metódami. Výsledná lokalizácia závisí od počtu meraní, presnosti senzorov a požadovanej úrovne lokalizácie. Jednou z metód, ktoré využívajú pravdepodobnostnú lokalizáciu, je Monte Carlo lokalizácia.

Monte Carlo lokalizácia, tak ako väčšina pravdepodobnostných metód na lokalizáciu využíva mapu prostredia. Táto metóda odhaduje výslednú polohu robota pomocou opakovaných meraní a množiny hypotéz. Pravdepodobnostné výsledky vedú k modifikácii aktuálnej množiny hypotéz a v nasledujúcom meraní porovnáva už len informáciu

o predchádzajúcej polohe, o akcií ktorú vykonal a hodnoty, ktoré má aktuálne namerané. Proces meraní sa opakuje stále počas navigácie robota v prostredí, pričom presnosť polohy robota sa zlepšuje najmä vtedy, keď sa nachádza v okolí jedinečných a teda jednoznačne rozpoznateľných priestorových charakteristík v prostredí.

Ďalšou metódou lokalizácie je lokalizácia pomocou význačných bodov[8]. Metóda vychádza z toho, že robot je schopný nájsť dané význačné body prostredia za pomoci kamery. Algoritmus na vyhľadávanie význačných bodov je popísaný nižšie. Na lokalizáciu musí robot rozpoznať aspoň dva body v obraze. Z nájdených kalibračných parametrov kamery vieme pre každý pixel obrazu v súradnicovom systéme zafixovanom v referenčnom bode robota určiť rovnicu priamky, ktorej všetky body sa na daný pixel premietajú. Pre dvojicu nájdených bodov premietnutých na dva rôzne pixle vieme určiť uhol, ktoré zvierajú zodpovedajúce priamky. Ak poznáme umiestnenie význačných bodov na mape prostredia, získame kružnicový oblúk, na ktorom sa nachádza robot. V najjednoduchšom prípade pre presné určenie lokalizácie robota sú potrebné tri body L_i, L_j, L_k, ktoré určia dva kruhové oblúky s priesečníkom, určujúcim polohu robota(Obr. 1.5). Tento proces je možné opakovať pre viacero nájdených význačných bodov a stanovenie polohy tým spresniť.



Obr. 1.5: Lokalizácia robota pomocou význačných bodov

Mapovanie

Cieľom robota pri mapovaní je zkonštruovať mapu neznámeho prostredia. Problémy pri mapovaní vznikajú z nepresných nameraných hodnôt alebo nepresnosti pohybu robota. Môže vznikať takzvaná kumulatívna chyba, ktorá môže rýchlo rásť

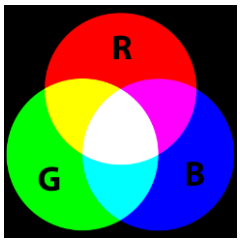
Ďalšie problémy, ktoré môžu nastať je, že dva rozdielne miesta prostredia môžu byť vnímané senzormi rovnako. Taktiež sa v prostredí môžu vykytnúť cykly.

1.1.7 Spracovanie obrazu

„Ľudské videnie je jednoduché a prirodzené, počítačové napodobňovanie je zložité“[9]. Pomocou videnia vnímame svet okolo nás najdokonalejšie. Vizuálny dátový kanál obsahuje najviac informácií a preto sa ho snažíme spracovať aj na počítači. Na spracovanie obrazu existuje veľa metód, nie je to však jednoduchý proces, nakoľko si to vyžaduje zapojenie vyšších kognitívnych procesov, ktoré doposiaľ nikto dostatočne vierohodne modelovať nedokázal.

Obrázky sú reprezentované nedeliteľnými obrazovými bodmi, ktoré nazývame pixely. Každý pixel uchováva informáciu o farbe a jase daného bodu. Pixely môžu mať rozličnú bitovú hĺbku (napr. 8, 16 alebo 24), z toho vyplýva koľko rôznych farieb je v jednom pixeli. Konkrétne pre 8 bitov to je 256, 16 -> 65536 a pre 24 -> 16777216 farieb. Farby, ktoré sú v pixeloch uchovávané reprezentujeme farebným modelom. Predstavíme si najznámejšie z nich, keďže v práci budeme implementovať konverziu medzi dvoma z nich.

RGB – výsledná farba vzniká zmiešaným troch farieb, z červenej(R), zelenej(G) a modrej(B)(Obr. 1.6). Tento model sa využíva hlavne vo farebných monitoroch, projektoroch a fotografiách. V 8-bitovej reprezentácií farebné zložky nadobúdajú hodnoty z intervalu <0,255>. Čierna farba má hodnotu 0-0-0, čo znamená že žiadna farba z modelu nie je zastúpená. Naopak biela 255-255-255, každá farba z modelu má maximálnu hodnotu. Tieto čísla vyjadrujú množstvo základnej farby, ktorá je vo výslednej farbe.



Obr. 1.6: RGB model

HSV – tento farebný model sa skladá z troch zložiek: Odtieň(Hue), Sýtosť(Saturation), Jas(Value). Odtieň určuje farbu objektu, sýtosť udáva primes iných

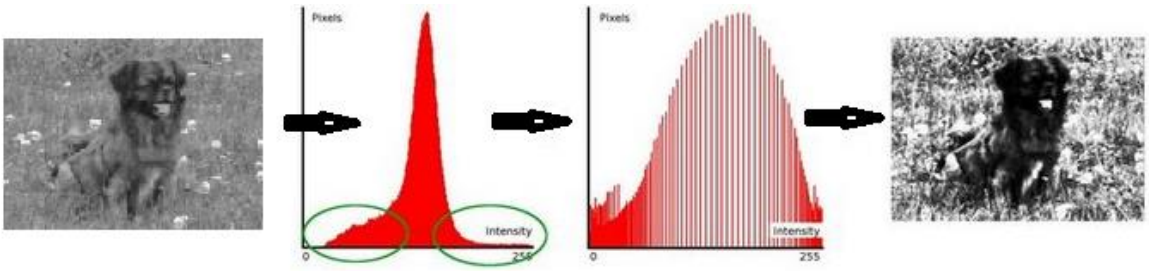
farieb a jas určuje hodnotu bielej farby. Model HSV sa najčastejšie využíva v grafických aplikáciach.

YCbCr – využíva sa pri videu a digitálnych fotografiách. Y predstavuje komponent jasu a Cb, Cr predstavujú modrý a červený komponent. YcbCr nie je úplný farebný model, slúži skôr na kódovanie RGB modelu.

YUV – rovnako ako predchádzajúci model aj tento YUV model slúži skôr na kódovanie RGB modelu. Zložka Y predstavuje jas a UV farebnosť. U je hodnota odtieňu medzi modrou a žltou a V je odtieň medzi červenou a žltou. Kódovanie pozostáva z transformácie: R = Y + 1,403V; G = Y - 0,344U - 0,714V; B = Y + 1,770U

Zlepšenie kvality obrazu

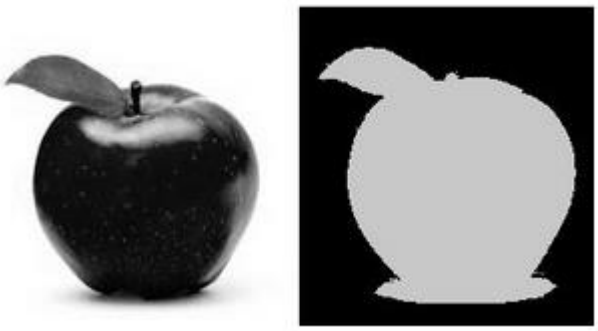
Na zlepšenie kvality obrazu, teda budeme chcieť zvýšiť kontrast v obraze, budeme používať ekvalizáciu histogramu. Čo je to histogram? Je to funkcia, ktorá pre každú hodnotu jasu v obraze určuje počet pixelov. Ekvalizácia histogramu je algoritmus, ktorý zmení rozloženie intenzít v obrázku tak, aby sa v ňom vyskytovali intenzity v širokom rozmedzí, a to približne s rovnakým počtom (Obr. 1.7).



Obr. 1.7: Ekvalizácia histogramu

Segmentácia obrazu

Cieľom segmentácie obrazu je oddeliť predmety s ktorými chceme pracovať od ostatných objektov. Najjednoduchšou metódou segmentácie je prahovanie, ktorá je veľmi rýchla a nie je náročná. Pri prahovaní sa využíva prah(threshold), ktorý zodpovedá jasovej konštane. Hodnoty výstupného obrazu po prahovaní rozdelujeme do dvoch skupín. V prvej hodnota jasu pôvodného obrazu je menšia ako prah a v druhej väčšia ako prah. Následne môžme daným skupinám priradiť farebné hodnoty pre lepšiu vizualizáciu(Obr. 1.8).



Obr. 1.8: Segmentácia obrazu

1.1.8 Vyhľadávanie význačných bodov

Pomocou význačných bodov môžeme jednoducho opísať dané objekty na obrázku, určiť ich polohu alebo nájsť rovnaký objekt vo viacerých obrázkoch. Veľmi dobre popísaná funkcia vyhľadávania význačných bodov je funkcia OpenCV knižnice, SURF[10].

Metóda SURF využíva reprezentáciu prechodových obrazov, ktoré nazývame integrálne obrazy. Tie sú vypočítané rýchlo zo vstupného obrázku a používajú sa na urýchlenie výpočtu akéhokoľvek obdĺžnikového priestoru. Majme vstupný obrázok I, bod (x,y), potom integrálny obraz vypočítame formulou:

$$I_{\sum}(x,y) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq y} I(x,y) \tag{1.5}$$

SURF detector je založený na determinante Hessian matice. Táto matica obsahuje parciálne derivacie funkcie f(x,y). Hessian maticu počítame ako funkciu pozície x = (x,y) a škáli o:

$$H(\mathbf{x},\sigma) = \begin{bmatrix} L_{xx}(\mathbf{x},\sigma) & L_{xy}(\mathbf{x},\sigma) \\ L_{xy}(\mathbf{x},\sigma) & L_{yy}(\mathbf{x},\sigma) \end{bmatrix} \tag{1.6}$$

Pričom $L_{xx}(x,o)$ odkazuje na konvolúciu druhého rádu Gaussovej derivácie $\frac{\partial^2 g(\sigma)}{\partial x^2}$ s obrazom v bode x = (x,y), podobne pre L_{xy},L_{yy}. Tie sa aproximujú pomocou filtrov D_{xx}, D_{xy}, D_{yy}. Následne sa vypočíta determinant Hessian matice pre každý pixel obrázku:

$$det(H_{approx}) = D_{xx}D_{yy} - (0.9D_{xy})^2 \tag{1.7}$$

Lokálne maximá v obraze x = (x,y) a škále o sú hľadané význačné body.

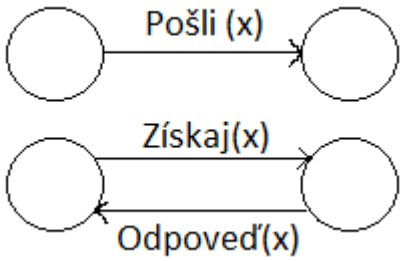
1.2 Existujúce systémy

1.2.1 Multiagentové architektúry

Tieto architektúry sú založené na decentralizácií a modularite. Jednotlivé moduly nazývame agentmi. "Agent je proces, ktorý neustále (opakovane)vníma svoje prostredie a na základe toho volí a vykonáva v ňom akcie, pričom sa jeho počínaniu dá pripísať určitý cieľ"[11]. Agenty sú autonómne, konajú nezávisle a proaktívne, k aktivite nie sú nikým vyvolané.

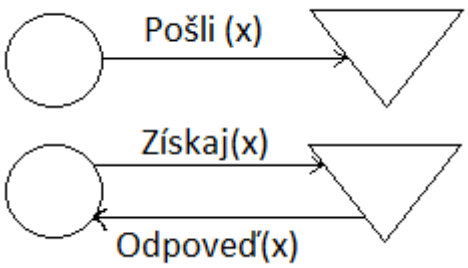
Medzi danými agentmi prebieha nejaká komunikácia, rozlišujeme dva typy komunikácie:

- Priama komunikácia (Obr. 1.9)
  - agenty si medzi sebou posielajú správy



Obr. 1.9: Priama komunikácia

- Nepriama komunikácia (Obr. 1.10)
  - agenty si vymieňajú správy cez prostredie

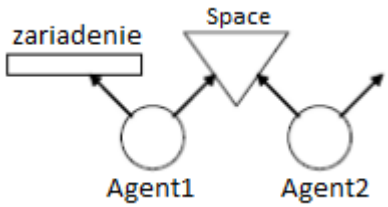


Obr. 1.10: Nepriama komunikácia

Agent-space Architektúra

Architektúra navrhnutá na FMFI UK[11]. Je postavená na multiagentovej architektúre, ktorá využíva nepriamu komunikáciu medzi agentmi. Agenty sú navrhnuté ako najnižšie jednotky systému a obsahujú práve jedno vlákno. Komunikácia medzi nimi je nepriama, teda dáta si medzi sebou vymieňajú pomocou určeného prostredia, ktoré sa nazýva space. Agenty majú možnosť zapisovať a čítať dáta zo space. Space sa skladá z dátových buffrov, ktoré sa nazývajú bloky. V blokoch si agenty zanechávajú správy a je na vývojárovi aby zosynchronizoval beh komunikácie, nakoľko bloky nemajú schopnosť nasmerovať nové dáta k príjemcovi. Implementácia systému v jazyku Java na synchornizáciu využíva štandardné javovské monitory.

Môžme predstaviť jednoduchý príklad architektúry, kde Agent1 komunikuje s vonkajším zariadením a o riadiaci systém sa stará Agent2. Na komunikáciu využívajú Space(Obr. 1.11). Dôležité dáta, Agent2 posiela prostredníctvom Space Agentovi1a ten sa stará len o komunikáciu so zariadeným na základe získaných dát zo Space. Agent1 a Agent2 pracujú nezávisle.



Obr. 1.11: Príklad architektúry Agent-Space

1.2.2 Robotický operačný systém - ROS

ROS je softvérová platforma robotického systému a v súčastnej dobe je najpoužívanejšia na svete. Je prispôsobená pre veľkú škálu vývojárov od amatérov cez univerzity až po veľké spoločnosti. ROS je postavený na objektovom systéme, kde si vývojár sám postaví svoj systém z modulov. Tieto moduly si môže naprogramovať sám alebo využije už hotové komponenty. Vďaka tejto modulárnej architektúre môžu vývojári vytvoriť veľké množstvo robotických aplikácií. K dispozícií majú ovládače hardverou, senzorov a rôzne typy knižníc pre plánovanie, lokalizáciu, mapovanie, rôzne kinematické úlohy, spracovanie obrazu, kalibráciou senzorických prvkov a mnoho ďalších.

ROS je open source platforma, čo je obrovská výhoda, pretože vývojári po celom svete sa podieľajú na vývoji systému. Aktuálne je k dispozícií viac ako 3000 balíčkov pre vývoj, ktoré si môže nový vývojár stiahnuť a aj pomocou nich vyvíjať svoj systém.

ROS je optimalizovaný pre operačné systémy postavené na Unix platforme.

1.2.3 Robot Shakey

Robot Shakey[2], ktorý tu uvádzame ako často citovaný míľnik a inšpiráciu k neskoršiemu vývoju v oblasti robotických architektúr, bol zkonštruovaný inštitúciou Stanford Research Institute v roku 1969. Bol vyvinutý na prístupoch hierarchickej architektúry. Jeho vybavenie pozostávalo z vizuálneho systému a nárazových senzorov. Avšak, výstupy senzorov neboli priamo spárované s motormi ale bola vytvorená plánovacia vrstva, ktorú využíval plánovač pomenovaný STRIPS. Plánovač využíval obrázky získané z vizuálneho systému na zkonštruovanie mapy okolia robota a cieľovej pozície zadanou užívateľom pre výpočet navigačného plánu, ktorý bol potom vykonávaný. Operácie robota sa skladali z akcií Vnímanie – Plánovanie – Akcia.

1.2.4 Open-R softvér

Daný softvér bol vyvinutý spoločnosťou Sony Corporation v roku 1997[2], avšak nie je open-source. Slúži na vývoj nových aplikácií mobilných robotov s Behavior-Based architektúrou. Cieľe Open-R sú poskytnúť:

- spoločné rozhranie pre senzory a aktuátory tak, aby boli vzájomne zameniteľné medzi robotmi
- metódu pre stanovenie funkcií a špecifikácie jednotlivých komponentov
- vrstevnú architektúru, čo uľahčuje rozvoj softvéru a hardvéru aplikácie

Softvérová platforma je objektovo orientovaná a preto všetky softvérové a aj hardvérové komponenty môžu byť indentifikované ako objekty. Na tejto platforme spoločnosť predstavila robota pomenovaného AIBO.

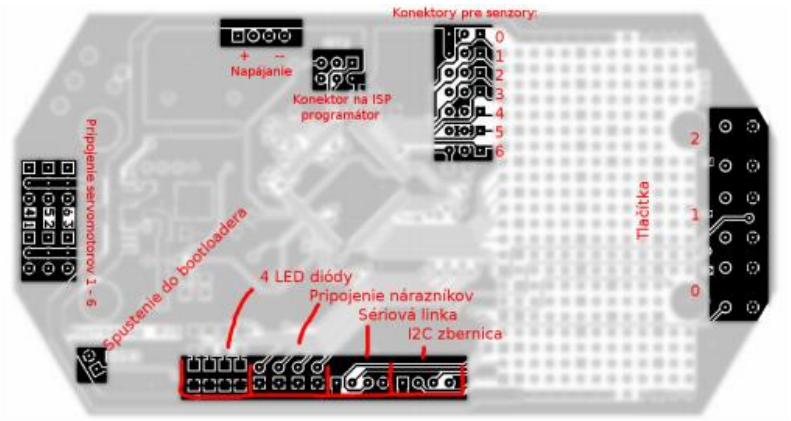
AIBO je autonómny robot, zostrojený v tvare psa. Obsahuje niekoľko senzorov a aktuátorov aby napodobňoval psie správanie. Má naprogramované simulácie emócií a štyri inštinkty: lásku, hľadanie, pohyb a hlad. Hlad znázorňuje vybytie batérie . AIBO zachytáva životný cyklus od mláďaťa po dospelého psa, pričom sa učí prostredníctvom okolia alebo jeho pána. Jeho správanie je vývojár schopný taktiež editovať, pomocou

špecifického programu. Okrem toho je k dispozícii Open-R SDK, ktoré umožňuje vývoj vlastných aplikácií pre robota využitím všetkej hw výbavy (kamera, dotykové senzory, inerciálny systém, wifi, veľmi kvalitne prepracovaná štvornohá morfológia riadená servo-motormi), čo z tohto robota urobila jednu z najpopulárnejších výskumných platforiem začiatkom tisícročia. Vyvinutých bolo viacero modelov, ale momentálne už žiaden nie je na trhu a vývoj sa zameriava najmä na humanoidné roboty (napr. Aldebaran Nao [12]).

1.3 Technológie

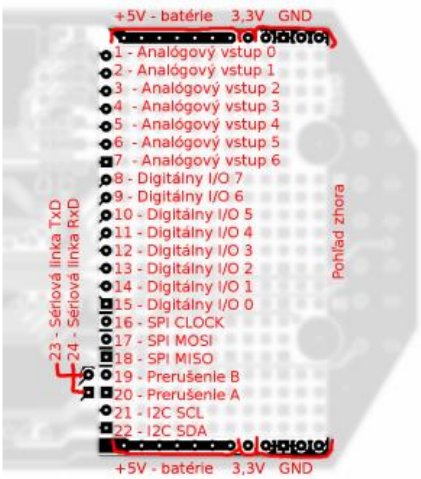
1.3.1 Mobilný robot SBot

SBot[13] je robotický systém, postavený pre úlohy v robotike a programovanie mikroprocesorov. Je stavaný ako aj pre jednoduché aplikácie, tak aj pre zložitejšie úlohy umelej inteligencie. Na komunikáciu s inými zariadeniami využíva Bluetooth rozhranie. Jeho konštrukcia je prispôsobená pre pridávanie rôznych, viacerých komponentov. Hlavnou súčiastkou robota je riadiaca doska – na ktorej je procesor ATMega128, Bluetooth modem, pripojené sensory a aktuátory a napájanie. Riadiaca doska sa skladá z dvoch častí: hlavná doska (Obr. 1.12) a univerzálna doska (Obr. 1.13).



Obr. 1.12: Sbot – schéma hlavnej dosky

Univerzálna doska slúži pre užívateľov, aby si na ňu pripojili vlastné zapojenia, rôzne senzory a konektory.



Obr. 1.13: Sbot – schéma univerzálnej dosky

Príklady použitia

- výuka a výskum mobilných robotov
- jednoduché úlohy ako sledovanie čiary, vyhýbanie sa prekážkam, hľadanie svetla a pod.
- diaľkové riadenie robota
- podvozok pre vyšší systém

1.3.2 SBot a servomotory a senzory

Na SBota je možné pripájať servomotory, rôzne senzory alebo užívateľské zariadenia. Servomotory budeme využívať na pohyb robota. Pomocou ich rotačného pohybu dosiahneme otáčanie kolies robota. Predstavíme si niekoľko senzorov, ktoré je možné prepojiť s SBotom:

- Line senzor
  - senzor určený na sledovanie čiary, vracia hodnotu v závislosti od farebnosti povrchu
- Sharp IR
  - senzor na meranie vzdialenosti
- Nárazník
  - mikrospínač, pomocou ktorého budeme určovať kolízie robota v prostredí. Ak jeho výstupná hodnota je 0, senzor nie je zatlčený, ak je 1 je zatlačený, vieme že nastal náraz

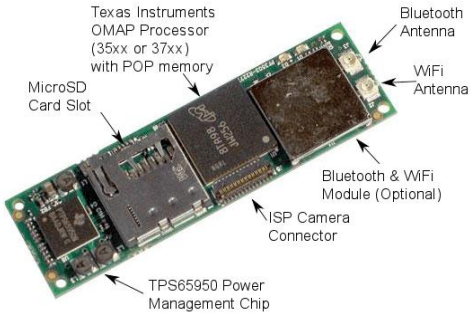
- Ultrazvuk
    - ultazvukový senzor na meranie vzdialenosti pomocou odrazu ultrazvukových vĺn
- Akcelerometer
    - senzor na meranie zrýchlenia, uhla náklonu alebo vychýlenia
- Kompas
    - senzor meria uhol natočenia robota

1.3.3 AVR Studio

Špecifické prostredie, ktoré je prispôsobené na prácu s procesormi AVR, ktoré SBot obsahuje. Vďaka tomuto programu môžme jednoducho vytvárať, ladiť a kompilovať programy určené pre SBota. AVR Studio teda poskytuje pre užívateľa príjemné editačné prostredie a jednoduchý proces údržby zdrojového kódu projektu.

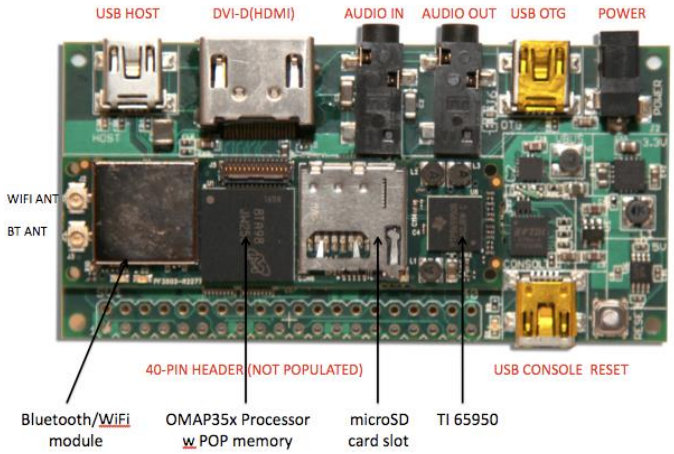
1.3.4 Gumstix Overo Air COM

Ako riadiacu jednotku s vyšším výpočtovým výkonom sme si zvolili zostavu Gumstix Overo Air COM, ktorá nám poskytuje výsoký výpočtový výkon v malom balení. Vďaka čomu sa nám s ním bude dobre pracovať na našom mobilnom robotovi. Model a jeho hlavné komponenty sú zobrazené na obrázku (Obr. 1.14).



Obr. 1.14: Gumstix Overo Air COM

Každý Gumstix Overo Air COM sa zmestí na všetky rozširujúce dosky série Overo. Napriek malej veľkosti, kombinácia Gumstixa Overo Air COM a rozširujúcej dosky funguje v plnej veľkosti. Gumstix Overo Air COM je postavený pre linuxovú platformu a tento linuxový počítač môže byť naprogramovaný tak, aby vykonával širokú škálu funkcií v ľubovoľných aplikáciach. Rozširujúca doska obsahuje viaceré konektory (Obr. 1.15).



Obr. 1.15: Rozširujúca doska Overo

1.3.5 Kamera Caspa pre Gumstix

Kamera Caspa určená na prácu s Gumstixami, využíva senzor MT9V032 , ale tento senzor nie je podporovaný priamo v jadre a je potrebné využiť dynamicky zavedený kernelovský modul.

Procesory OMAP3 v Gumstix Overo COM majú jednoúčelový hardvér pre snímanie a spracovanie dát z obrazových snímačov. Kamerový snímač Caspy vydáva 10-bitové Bayer obrázky, ktoré sú prenášané na Image Signal Processor (ISP), cez paralelné rozhranie. ISP obsahuje rôzne podmoduly, ktoré je možné exportovať ako video pre linux (V4L2) zariadenia namapované do súborového systému v /dev.

1.3.6 Softwarová platforma - Yocto Project

Na výber máme viacero softwarových platforiem, ktoré podporujú Gumstix Overo Air COM. Napríklad Yocto Project, Ubuntu, Angstrom alebo Android. Na našu prácu sme si zvolili Yocto Project. Yocto Project umožňuje vytváranie vlastných linuxových distribúcií pre vstavané systémy, platformy Gumstix. Jedná sa o súbor git úložísk, z ktorých každá poskytuje popis na zostavenie softvérových balíkov, ako aj informácie o konfigurácii.

Na zostavenie platformy Yocto Project budeme potrebovať nasledujúce súbory:

- MLO - second-stage bootloader binary image pre COM
- ROOTFS - súborový systém
- U-BOOT - bootloader binary image
- UIMAGE - kernel binary image

- MD5 - overovanie preberaných súborov

1.3.7 Video4Linux

Video4Linux (v4l) je API pre riadenie vstupných i výstupných video zariadení a framework pre ovládače týchto zariadení pre linuxové jadro. Je podporovaný mnohými webkamerami, tv tunermi a ďalšími zariadeniami.

1.3.8 OpenCV

OpenCV je knižnica a implementáciami algoritmov zameraná hlavne na spracovanie obrazu a počítačové videnie. Bola vyvinutá výskumným centrom Intel Russia a teraz podporovaná spoločnosťami Willow Garage and Itseez. OpenCV je open-source a je napisaný v C++, ale udržuje si plné rozhrania pre viacero programovacích jazykov, ako napríklad C++, C, Python, Java, MatLab. Podporuje rôzne operačné systémy – Linux, Windows, Mac OS a iné.

1.3.9 FANN knižnica

FANN (Fast Artificial Neural Network) je open source knižnica, ktorá implementuje viacvrstvové umelé neurónové siete v programovacom jazyku C. Obsahuje dobre zdokumentovaný framework pre ľahkú prácu s trénovacími dátami. Je univerzálna, rýchla a podporuje viac ako 20 programovacích jazykov.

1.3.10 Level shifter

Level shifter sa využíva, keď signály prechádzajú z jednej napäťovej domény do inej, aby logické 1 a 0 boli správne prečítané. Napríklad ak vstup má napätie 1.5V ale napäťová doména potrebuje 5V, potrebuje daný level shifter. Príkladom je obvod BSS od spoločnosti FairChild Semiconductor[14] predinštalovaný 4x na doske spoločnosti Adafruit 757 [15].

2 Platforma Cipísek

2.1 Ciele a špecifikácia

Našim cieľom je s použitím mobilného robota – Sbota 2.0 vytvoriť robotickú mobilnú platformu na experimenty umelej inteligencie a rozšíriť ho o riadiacu jednotku

s vyšším výpočtovým výkonom – t.j. Gumstix. Navrhnúť, implementovať a zdokumentovať rámec experimentov a zabezpečiť ich komunikáciu pomocou sériového portu, nakoľko Sbot má +5V napájanie a Gumstix +1.8V napájanie.

2.1.1 Senzorické vybavenie

Vzhľadom na experimenty, ktoré budeme chcieť vykonávať bude potrebné namontovanie a prepojenie nejakých senzorov.

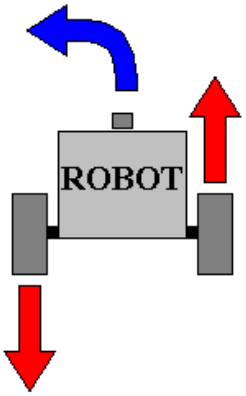
- Nárazníky (mikrospínače)
  - dané senzory budeme využívať aby sme nepoškodili robota a vedeli odhadovať prekážky
- Line senzory
  - budeme chcieť aby robot bol schopný napríklad sledovať čiaru alebo rozpoznávať na akom farebnom povrchu sa nachádza, na vykonanie týchto a podobných úloh budeme využívať tieto senzory
- Sharp IR senzory
  - tieto senzory sú pre nás veľmi dôležité a ich zapojenie je priam nutné. Vďaka ním budeme môcť odmerať vzdialenosť robota od predmetov, čo môže byť rozhodujúcim faktorom v mnohých experimentoch. Ich rozmiestnenie si bude potrebné veľmi dôkladne premyslieť.
- Kamera Caspa
  - kameru prepojíme s Gumstix na experimenty týkajúce sa spracovania obrazu

2.1.2 Pohyb a navigácia

Čo sa týka pohybu robota požadujeme aby robot bol mobilný, nie stacionárny. Jeho pohyb by mal byť v priestore efektívny, to znamená aby dosahoval ľubovoľné miesta v rovine v ľubovoľnom natočení, postavení.

Za vhodnú morfológiu preto považujeme differential drive robot, čo je spôsob ovládania robota s dvoma nezávislými motorizovanými kolesami a oporným bodom. Napríklad ak ľavé koleso budeme točiť smerom dozadu a pravé dopredu rovnakou rýchlosťou, potom sa robot bude točiť na mieste smerom doľava (Obr. 2.1). Týmto

spôsobom môžeme získať pohyb po kruhových oblúkoch len zmenou rýchlosti obidvoch kolies.



Obr. 2.1: Ukážka pohybu robota. Červené šípky predstavujú smer otáčania kolies, modrá smer rotácie robota.

Z hľadiska navigácie by mal byť robot ľahko rozšíriteľný o:

- odometriu
    - jedná sa o meranie otáčok, napríklad ružicou
- senzorické vybavenie uľahčujúce navigáciu
    - kompas, akcelerometer

Nekladieme vysoké nároky na presnosť pohybu, pretože robot je určený skôr pre experimenty v umelej inteligencií, kde robot neustále sníma prostredie pomocou senzorov a na základe aktuálnych informácií volí vhodné akcie. Preto namiesto presných posunov a otočiek o zvolenú vzdialenosť, respektíve uhol uprednostňujeme navigáciu za pomoci vnímania senzorov.

Taktiež systém by mal poskytovať dostatočnú podporu pre riadenie navigácie, pod ktorou máme namysli aktuálnu zmenu rýchlosti, smeru, zastavenia alebo prípadne pohyby po kruhových oblúkoch.

2.1.3 Riadiaca architektúra

Riadiacu architektúru chceme postaviť na základných vlastnostiach Behavior Based architekúry. Mala by sa skladať z jednotlivých modulov, správaní a mala by byť ľahko rozšíriteľná o ďalšie moduly.

Jednotlivé moduly by si mali zachovať konkrétnu šablónu správania, od ktorej bude zdedené každé nové správanie. Štruktúra, by mala obsahovať:

- meno
- prioritu
- aktivačnú premennú
- funkciu, ktorá reprezentuje beh správania
- funkciu, ktorá bude slúžiť na komunikáciu medzi správaniami

Musí byť zachovaná hierarchia architektúry, že jednotlivé správania nebudú schopné pristupovať a meniť hodnoty premenných iným správaniam. Medzi správaniami bude prebiehať priama komunikácia, správania si medzi sebou budú môcť posielať správy. Taktiež ak to správanie požaduje, bude sa môcť zapísať u iného správania, že chce od neho dostávať správy o hodnote konkrétnej premennej. Nepriamu komunikáciu v štýle Blackboard architektúry (resp. AgentSpace) možno jednoducho doplniť pridaním jediného správania/modulu reprezentujúceho blackboard.

Správanie by nemalo vôbec byť ovplyvňované tým, na ktorej platforme sa nachádza, či je to Sbot alebo Gumstix. Napríklad ak správanie nachádzajúce sa na Sbotovi bude chcieť poslať správu správaniu na Gumstix, jednoducho mu ju pošle a systém by mal zabezpečiť pomocou interných smerovacích mechanizmov, že daná správa sa pošle z Sbota na Gumstix, jeho adresátovi. Rovnako aj opačným smerom. Architektúra by mala byť otvorená ľahkému rozšíreniu o ďalšie platformy – napr. časť výpočtu by mohla prebiehať na PC prepojenom s Gumstix cez wifi.

Jednotlivé správania by mali byť volané v rovnakých frekvenciách a v rovnakých postupnostiach. Rozhodnutie o tom, ktorá výsledná hodnota správania bude vykonávať akciu robota by mala vyriešiť daná priorita správania. Na platforme SBOT, keďže sa jedná o jednočipový 8-bitový mikropočítač sa predpokladá nepreemptívny kooperatívny multitasking, t.j. každé správanie by malo v pomerne krátkom čase odovzdať riadenie naspäť plánovaču. V praxi to znamená implementáciu správaní v štýle stavových automatov. Na platforme Gumstix tento základný kooperatívny multitasking môže byť doplnený preemptívnym, ktorý poskytuje OS Linux – štartovanie výpočtových vlákien si manažujú jednotlivé správania samostatne podľa potreby, ale poskytnutá infraštruktúra architektúry je „thread-safe“.

Jednotlivé správania by mali byť úplne nezávislé od iných. Ak napríklad budeme využívať jedno konkrétne správanie v systéme, ale časom zistíme že jeho správanie nie je

optimálne a nahradíme ho novým optimálnym správaním ostatné moduly by nemali „pocítiť“, že dané správanie bolo zmenené.

2.1.4 Sada experimentov

Pomocou sady experimentov budeme chcieť ukázať funkčnosť nášho navrhnutého robota a jeho architektúry. Ukázať názorné príklady na prácu z danými zavedenými senzormi a algoritmami umelej inteligencie a počítačovým videním. Zdokumentovať výsledky a vyhodnotiť úspešnosti. Experimenty si rozdelíme podľa náročnosti do troch kategórií:

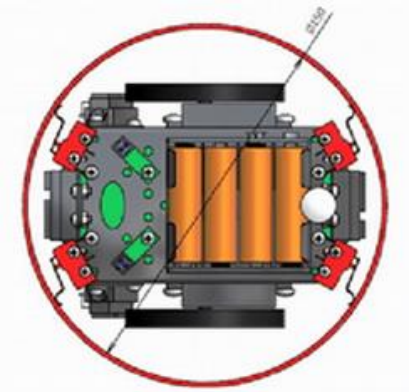
- Základné experimenty
  - Náhodný pohyb robota bez kolízie v prostredí
  - Ovládanie robota cez Bluetooth pomocou klávesnice vo vymedzenom priestore – priestor určený čiarou na podlahe
  - Udržovanie sa v strede koridoru
  - Sledovanie čiary a vyhýbanie sa prekážkam
- Experimenty s počítačovým videním
  - Sledovanie predmetu pomocou kamery
  - Zaznamenávanie význačných bodov do pamäti
- Experimenty s učením správania sa
  - Natrénovanie pohybu robota v prostredí bez kolízií pomocou neurónových sietí

2.2 Návrh

Navrhnutú robotickú mobilnú platformu postavíme v programovacom jazyku C, kvôli tomu, že daný jazyk je optimálny na prácu s Sbotom. Na Gumstix nainštalujeme vyššie spomínanú linuxovú platformu Yocto Project a budeme musieť zabezpečiť kompiláciu programu v jazyku C. Nemôžeme zabudnúť zabezpečiť komunikáciu medzi Sbotom a Gumstix, kde sa dá využiť level shifter na prevod medzi napätiami.

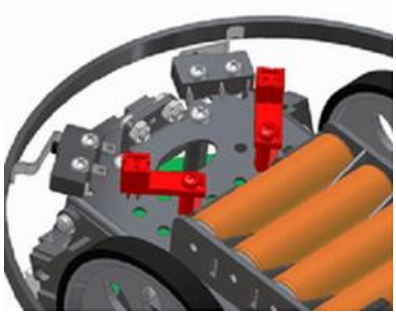
2.2.1 Senzorické vybavenie

Na konštrukciu robota budeme chcieť postupne namontovať dané cieľové senzory. Najskôr pripevníme nárazníkové senzory, tak aby sme zabezpečili rozpoznanie nárazu z každej strany robota. Môžme použiť plastový pásik ohnutý do tvaru kruhu a pripevnený na každý mikrospínač (Obr. 2.2). Tým zabezpečíme rozpoznanie obrazu po celom obvode robot.



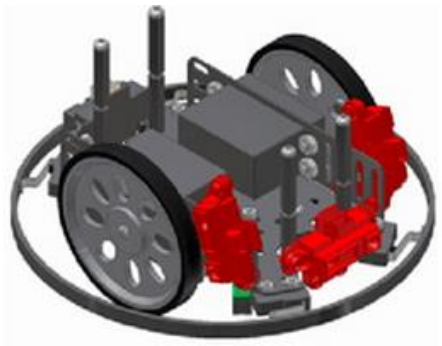
Obr. 2.2: Plastový pásik pripevnený na mikrospínače[16]

Line senzory pripevníme na spodnú stranu robota, aby boli schopné rozpoznávať povrch prostredia po ktorom sa robot pohybuje (Obr. 2.3). Na snímanie budeme chcieť využívať dva senzory tohto typu.



Obr. 2.3: Umiestnenie line senzorov[16]

Senzory Sharp IR sme sa rozhodli pripevniť na prednú stranu robota. Budeme využívať tri tieto senzory. Prvý nasmerujeme smerom dopredu a budeme pomocou neho merať vzdialenosť predmetov pred robotom. Ostatné dva pripevníme po stranách robota. aby sme boli schopný získať vzdialenosti predmetov na jednotlivých stranách od robota (Obr. 2.4).

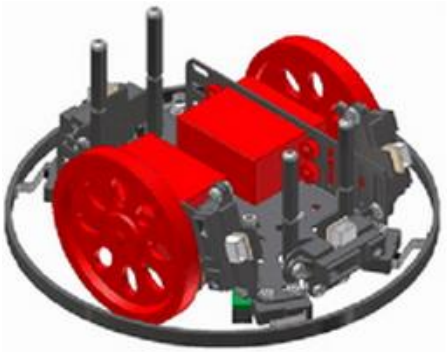


Obr. 2.4: Umiestnenie Sharp IR senzorov[16]

Na konštrukciu Sbota musíme pripojiť aj Gumstix s pripojenou kamerou a taktiež napájanie Gumstixa. Tento problém môžeme vyriešiť postavením pevnej nadstavby nad telo Sbota, kde pripevníme batérie na napájanie Gumstix, samotnú dosku Gumstix a kameru. Kameru budeme chcieť nasmerovať smerom dopredu, aby bolo možné prispôsobovať pohyb robota na základe jej obrazu.

2.2.2 Pohyb a navigácia

Na splnenie cieľov pohybu a navigácie máme k dispozícií dva servomotory využívajúce princíp differential drive robot (Obr. 2.5). Každé koleso obsahuje svoj vlastný pohon, teda sa dajú jednotlivo ovládať. Tým zabezpečíme rôznorodosť pohybov robota.



Obr. 2.5: Umiestnenie servomotorov[16]

2.2.3 Riadiaca architektúra

Základný a najnižší prvok architektúry je modul, ktorý nazývame správanie. Každé nové správanie by malo zdediť základnú štruktúru architektúry podľa podmienok stanovených v cieľoch architektúry.

Návrh základnej štruktúry môžeme v našom vybranom programovacom jazyku C reprezentovať ako štruktúru, typ v jazyku C – `struct`, ktorá bude obsahovať meno

správania, prioritu, aktivačnú premennú, funkciu zabezpečujúcu beh správania a funkciu na komunikáciu.

```
struct spravanie {
      // konkrétna implementácia
}
```

Priamu komunikáciu medzi správaniami zabezpečíme pomocou funkcie *handler*. Každé správanie môže na základe jeho mena získať referenciu na deskriptor správania a poslať mu správu pomocou danej funkcie. Naríklad správaniu strojovna, chceme poslať nejakú správu:

```
spravanie = vrat_spravanie("strojovna");
spravanie.handler((dĺžka správy), (správa));
```

Ak správanie, ktorému sa správa posiela sa nachádza na inej platforme, posielateľovi by malo byť vrátené správanie, ktoré zabezpečuje len posielanie správ medzi platformami. Toto správanie by si malo udržiavať všetky informácie o tom komu sa to posiela, jej veľkosť a danú správu. To akú binárnu štruktúru bude mať správa je na vývojárovi.

Jednotlivé správania môžu mať zaregistrovaných odoberateľov, ktorí budú získavať správy od správania, u ktorého sú ako odberatelia zaregistrovaní. Ak je správanie zaregistrované u niekoho, to znamená, že od daného správania chce dostávať informáciu o hodnote premennej pravidelne, resp. v prípade jej zmeny, ktorá môže zodpovedať nejakej udalosti, na ktorú sa zaregistrovalo.

```
struct odoberatel {
     char *kto;
   int co;
}
```

Systém si bude udržiavať pole všetkých správaní a v nekonečnom cykle ich bude zaradom spúšťať, avšak realizuje sa len to správanie, ktoré má aktivačnú premennú nastavenú na hodnotu 1. Každé spustené správanie sa vyhodnotí a môže poslať správu jednému alebo viacerým správaniam a tiež správaniu AKCIA, ktoré riadi akcie robota. Na to aby, pri vykonaní akcie sa správanie AKCIA vedelo rozhodnúť, podľa koho sa má riadiť, tak zavedieme priority. Správanie s vyššou prioritou, tak prebije riadenie požadované ostatnými správaniami.

```
while(1) {
      for (i = 0; i < pocet_spravani; i++) {
```

```
                    if(spravanie[i].active == 1){
                        spravanie[i].loop();
                    }
                }
            }
```

Všetky správania musia byť sa sebe nezávislé a nevstupovať do behu inému správaniu.

2.2.4 Sada experimentov

Predstavíme si jednotlivé experimenty a aké výsledky budeme od nich očakávať.

1. **Náhodný pohyb robota bez kolízie v prostredí**

   Jedná sa o základný experiment. Chceli by sme ním ukázať pohyblivosť robota a schopnosť vnímania prekážok a ich vyhýbaniu sa. Testovať budeme správania v rámci Sbota.

2. **Ovládanie robota cez Bluetooth pomocou klávesnice vo vymedzenom priestore**

   Pri náročnejších úlohách s robotom, budeme chcieť robota ovládať. Na to musíme mať skompletizované a otestované základné ovládanie robota a daný experiment slúži práve na to. Robot bude ovládaný klávesnicou a bude sa musieť udržať vo vymedzenom prostredí. Robot by už mal obsahovať výsledné správanie z prvého experimentu, čiže nemal by prísť do kolízie s inými predmetmi aj keď ho užívateľ bude na to viesť. Daný experiment bude prebiehať len na Sbotovi.

3. **Udržovanie sa v strede koridoru**

   Experimentom budeme chcieť ukázať funkčnosť senzorov na meranie vzdialenosti a predviesť s nimi daný experiment udržanie sa v strede koridoru. Správania budú štarované len na Sbotovi.

4. **Sledovanie čiary a vyhýbanie sa prekážkam**

   Robot bude pomocou line senzorov sledovať čiaru, pomocou predchádzajúcich experimentov bude vedieť vyhodnotiť prekážku a mal by ju vedieť obísť a vrátiť sa späť na čiaru. Experiment bude prebiehať len na Sbotovi.

5. **Sledovanie predmetu pomocou kamery**

   V experimente budeme tiež testovať komunikáciu systémov, nakoľko budeme využívať Gumstix na výpočtovo náročnejšie operácie ako je spracovanie obrazu z kamery. Gumstix bude výsledok vyhodnocovať a výslednú akciu robota zabezpečí Sbot na základe dosiahnutých výsledkov. Robot by mal byť schopný sledovať nejaký objekt.

6. **Zaznamenávanie význačných bodov do pamäti**

   Takisto ako v predchádzajúcom experimente robot na výpočtovo náročnejšie operácie bude využívať Gumstix. Týmto experimentom chceme ukázať, otestovať a zdokumentovať základnú prácu s význačnými bodmi. Užívateľ bude vedieť pomocou klávesnice ovládať robota a nastaviť ho na zaujímavé miesto v prostredí a zaznamenať význačné body z obrazu do pamäti na jeho príkaz.

7. **Natrénovanie pohybu robota v prostredí bez kolízií pomocou neurónových sietí**

   Výpočtovo náročnejšie operácie vykonáva Gumstix. V prvom kroku experimentu si robot získa trénovacie dáta, tak že užívateľ sa bude pohybovať po prostredí a robot si bude zapisovať vzdialenosti zo senzorov a vykonanú akciu užívateľa. Následne získané dáta použijeme ako trénovaciu množinu pre neurónovú sieť. Výsledkom bude, že robot na základe nameraných hodnôt pomocou trénovacej množiny a neurónovej siete vyhodnotí akciu. Robot by mal byť schopný pohybovať sa v prostredí bez kolízií.

2.3 Realizácia

2.3.1 Inštalácia – Gumstix a Caspa

Ako prvú časť práce sme sa rozhodli oboznámiť sa s platformou Gumstix a kamerou Caspa a vzájomne ich prepojiť, nakoľko ešte u nás na katedre KAI FMFI Caspa nebolo nikým úspešne spojazdnená. Prvý operačný systém, ktorý sme použili na beh Gumstix bol Angstrom. Zvolili sme si ho kvôli nie náročnej inštalácií, menšej veľkosti a dostačujúcej rýchlosti systému. Po úspešnej inštalácií softvéru sme prepojili Gumstix a kameru Caspa a pokúsili sa získať obraz z kamery. Stretli sme sa s viacerými podobnými problémami ako naši kolegovia pracujúci s touto kamerou. Napríklad rozpoznanie kamery zariadením Gumstix nebolo vždy optimálne, kamera často vracala len čisto zelený obraz alebo bol

viditeľný len šum predmetov v obraze. Jej optimálnu konfiguráciu sa nám podarilo aj pomocou skontaktovania výrobcov kamery rozbehať pomocou ovládača MT9V032. Avšak z testovania sme spozorovali, že kamera po aktivovaní vždy vzhľadom na automatické nastavenie jasu potrebuje dlhší čas na adaptáciu na svetelné podmienky v prostredí, preto prvý frame ktorý vracala neobsahoval žiadnu relevantnú obrazovú informáciu. Tento problém sme vyriešili uložením viacerých framov do pamäti a následne vystrihnutím posledného dostupného framu, čo nám zabezpečilo vyhovujúci obraz.

Nový kernel OS Linux a modul pre zariadenia mt9v032

Počas našej práce, bol vydaný nový kernel Linuxu 3.5.7, ktorý sme sa rozhodli využívať, pretože pre neho existoval modul s podporou pre zariadenia vrátane kamery Caspa. Výrobcovia systému Gumstix pripravili distribúciu na platforme Yocto Project, preto sme preinštalovali systém na Gumstix a tento OS doteraz používame. Na získanie snímaného obrazu z kamery zo zariadenia video4linux možno použiť Gstreamer, t.j. knižnicu pre konštrukcie a manipuláciu s médiami, ktorú bolo potrebné nainštalovať. Knižnica obsahuje veľa balíčkov a preto bolo potrebné pripojiť Gumstix na Internet pomocou wifi modemu, ktorý obsahuje. Konfiguračné nastavenia Gumstix na pripojenie cez wifi:

```
ctrl_interface=/var/run/wpa_supplicant
ctrl_interface_group=0
eapol_version=1
ap_scan=1
fast_reauth=1

network={
    ssid="add-your-ascii-ssid"
    proto=WPA2
    key_mgmt=WPA-PSK
    pairwise=CCMP TKIP
    group=CCMP TKIP
    scan_ssid=1
    psk="add-your-ascii-passphrase"
    priority=10
}
```

Pripojenie a odpojenie sa vykoná príkazmi `Ifup wlan0` *a* `Ifdown wlan0`. Akonáhle sme mali prístup na Internet, nainštalovanie GStreamera a potrebných balíčkov nebolo náročné, vykonali sme príkaz:

```
sudo smart install gstreamer gst-plugins-good-video4linux2 gst-plugins-bad-autoconvert gst-plugins-base-theora gst-plugins-good-rtp gst-plugins-good-udp gst-plugins-base-videorate gst-plugins-good-jpeg gst-plugins-good-multipart gst-plugins-base-tcp
```

Následne po naištalovaní príslušných balíčkov musíme kameru nastaviť. Ako prvé nastavíme spojenie:

```
media-ctl -r -l '"mt9v032 3-005c":0->"OMAP3 ISP CCDC":0[1], "OMAP3 ISP CCDC":2->"OMAP3 ISP preview":0[1], "OMAP3 ISP preview":1->"OMAP3 ISP resizer":0[1], "OMAP3 ISP resizer":1->"OMAP3 ISP resizer output":0[1]'
```

Ako ďalší nastavíme formát:

```
media-ctl -V '"mt9v032 3-005c":0[SGRBG10 752x480], "OMAP3 ISP CCDC":2[SGRBG10 752x480], "OMAP3 ISP preview":1[UYVY 752x480], "OMAP3 ISP resizer":1[UYVY 752x480]'
```

Tieto nastavenia je potrebné vykonať vždy po štarte systému ak ideme pracovať s kamerou. Nakoniec zavoláme príkaz na získanie obrazu:

```
Gst-launch –v –e v4l2src device=/dev/video6 num-buffers=5 ! autoconvert ! ffmpegcolorspace ! jpegenc quality=80 ! filesink location=img.jpg
```

Prácu s Casps sme si vďaka novému kernelu uľahčili a zrýchlili, ale niektoré problémy s ktorými sme sa stretli predtým sa prejavovali aj tu. Hlavne adaptácia kamery na prostredie. Daný problém riešime rovnakým spôsobom ako predtým.

2.3.2 Konštrukcia a prepojenie robota

Keď sme mali rozbehaný systém na Gumstix a vedeli sme získať obraz z Caspa, mohli sme sa pustiť do stavania konštrukcie nášho robota. Postupovali sme podľa návrhu, kde servomotory s kolesami sme pripojili po stranách motora na pevnú, obdĺžnikovo kovovú kostru. Na jej spodok je pripevnený držiak na batérie, pomocou ktorých budeme napájať Sbota. Nárazníkové senzory sme pripevnili po rohoch a na ne nasadili plastový pásik, ktorý nám zabezpečuje odhalenie nárazu po celom obvode okolo robota. Senzory na snímanie čiary sme namontovali z dolnej strany konštrukcie, aby mohli snímať povrch

prostredia(Obr. 2.6). Sharp IR senzory na snímanie vzdialenosti sme pripevnili v prednej časti konštrukcie, využívame tri kusy týchto senzorov(Obr. 2.7). Prvý smeruje dopredu a ostatné dva do strán robota, čiže vľavo a vpravo. Na hornú časť sme položili hardverovú dosku Sbota, na ktorú sme pripojili jednotlivé senzory, konkrétne:

- Servomotory na vstupy: ľavý 1, pravý 4
- Nárazníkové senzory na vstupy: pravý predný 0, ľavý predný 1, ľavý zadný 2, pravý zadný 3
- Line senzory na vstupy: ľavý 2, pravý 3
- Sharp IR senzory na vstupy: pravý 4, stredný 5, ľavý 6

Na dosku Sbota môžeme pripojiť až 6 servomotorov, ktorých prípojky rozdeľujeme na dve podskupiny. Sadu servomotorov 1-3 zapíname/vypíname vo frameworku makrami *servo_123_on() / servo_123_off(),* pre servomotory 4-6 sú to *servo_456_on() / servo_456_off().* Následné nastavenie smeru motora nastavujeme príkazom *set_servo(číslo, smer),* kde číslo je konkrétne číslo motora. Pohyb môže nadobúdať hodnotu vpred, vzad alebo stáť. Vo frameworku sú tieto hodnoty predstavované príkazmi *SERVO_FW, SERVO_BW, SERVO_STOP*. Robot má naprogramovaný časovač na automatické vypínanie motorov. Ak mu chceme nastaviť neustály beh, nastavíme premennú: *ticks_until_stop = TIMER_SERVO_CONTINUOUS_OPERATION* alebo ak chcem určitý čas hodnotu premennej vyrátame: (počet sekúnd, koľko sa má hýbať) x 100.

K takto zapojeným senzorom, potom pristupujeme k jednotlivým výstupným hodnotám v programe nasledovne:

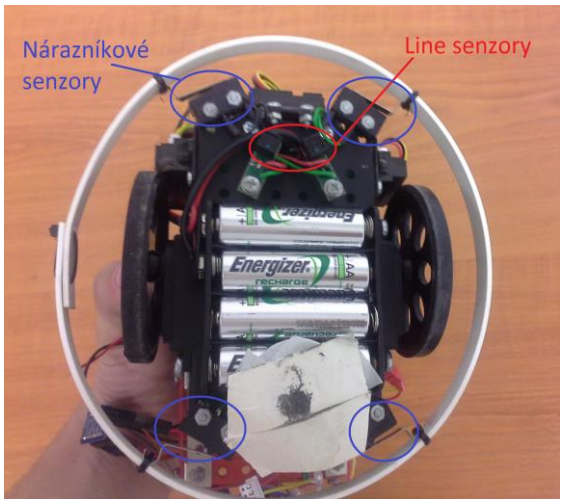
- Nárazníky: `bumper-stat(BUMPER0-4)`, číslo od 0 do 4 si zvolíme podľa toho, ktorý nárazník chceme.
- Line senzory: `adc_vals[2] a adc_vals[3],` ľavý a pravý čiarový senzor
- Sharp IR senzory: `adc_vals[4], adc_vals[5], adc_vals[6],` postupne sensor na ľavej strane, v strede a pravej strane

Keď prvá časť konštrukcie bola hotová, začali sme pracovať na pripevnení Gumstix s Caspa na robota. Nadstavili sme novú konštrukciu nad hornú časť, ale zachovali sme

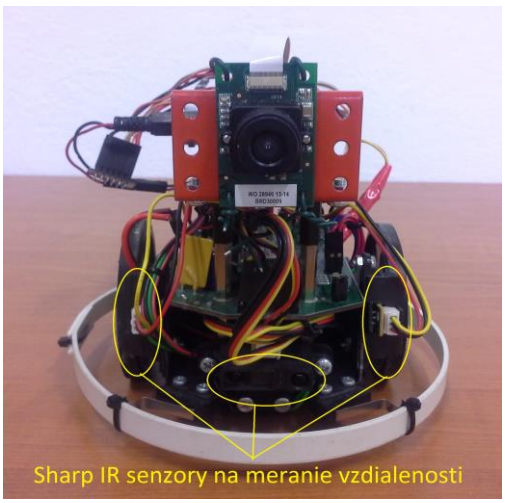
dostatočný priestor, aby sme mohli manipulovať so zapájaním alebo úpravami komponentov na doske Sbota. Na novej nadstavbe sme vyhradili priestor v prednej časti na Caspa, smerujúcu dopredu pred robota. V strede, priestor na upevnenie Gumstix a v zadnej časti na batériu, ktorá bude napájať Gumstix. Ako zdroj napájania sme použili dva články vyradených mobilných batérií. Aby sme ich mohli používať skonštruovali sme ich prepojenie s koncovkou pre napájanie Gumstix. Na kábliky sme zapojili spínač, ktorý musí byť zapnutý, aby prechádzal prúd (Obr. 2.8). Takisto sme si nechali k dispozícii prepojku, kde môžeme zapojiť buď koncovku pre Gumstix alebo nami pripravenú koncovku na nabíjanie baterií (Obr. 2.10).

Keď sme už mali hotovú konštrukciu Sbota a spojazneného Gumstix aj s vyriešeným napájaním, mohli sme tieto dve platformy spolu prepojiť pomocou sériového portu. Ako sme spomínali vyššie, Sbot má +5V napájanie a Gumstix +1.8V napájanie, preto na ich prepojenie sme využili level shifter, ktorý nám zabezpečil ich korektnú komunikáciu(Obr. 2.9).

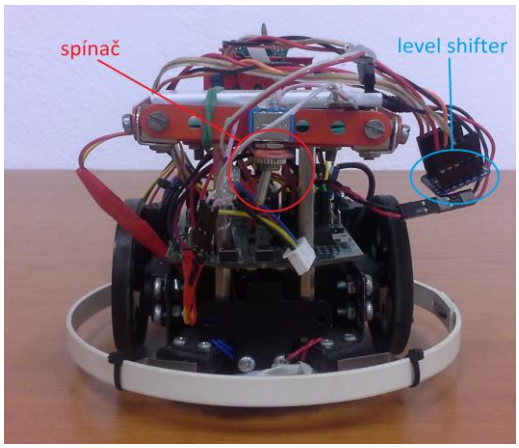
Robot Cipísek je hotový(Obr. 2.11).



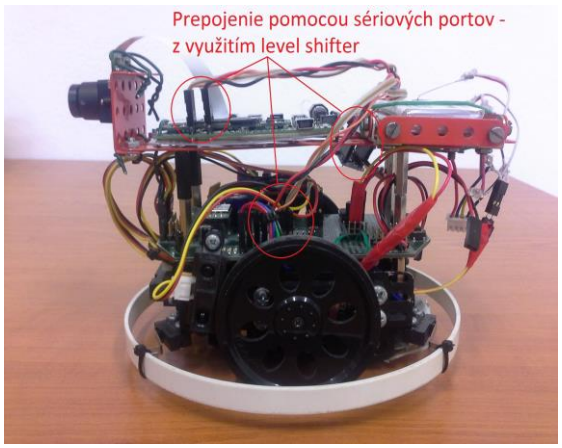
Obr. 2.6: Spodná časť robota



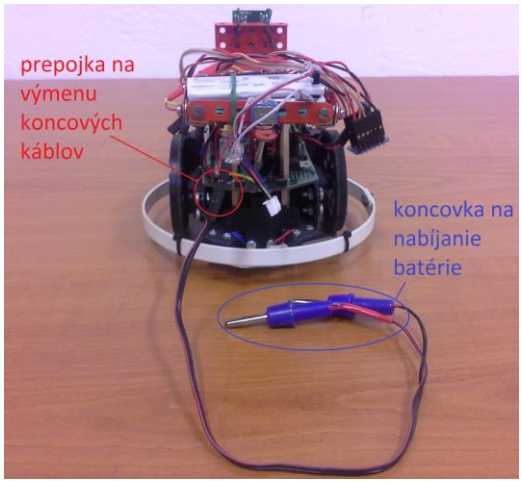
Obr. 2.7: Predná časť robota



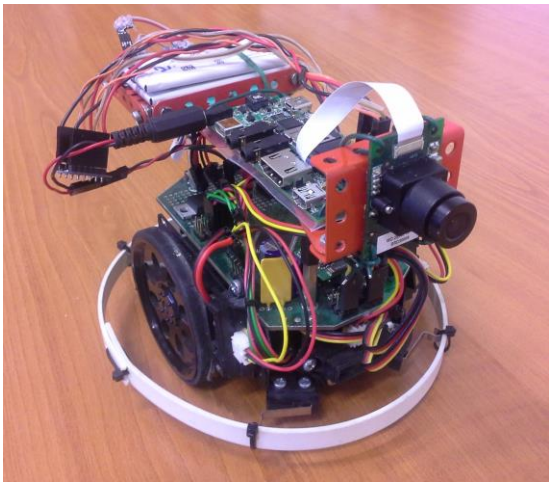
Obr. 2.8: Zadná časť robota



Obr. 2.9: Bočna strana robota



Obr. 2.10: Zapojenie nabíjania batérií



Obr. 2.11: Robot Cipísek

2.3.3 Realizácia architektúry

Základná softvérová inštalácia robota Sbot obsahuje okrem iného aj programovacie prostredie AVR Studio a základný framework na prácu s ním. Tento framework obsahuje rôzne základné nastavenia ako napríklad funkcie na získavanie hodnôt zo senzorov, ovládanie servomotorov, časovač alebo funkcie určené na komunikáciu pomocou sériovej linky. Všetky tieto základné funkcie budeme využívať v našej architektúre.

Navrhovanú dátovú štruktúru, ktorú budú dediť naše správania, sme určili nasledovne:

```
struct beh_str {
    char *name;
```

```
    int priority;
    char active;
    void (*loop)();
    void (*handler)(int weight, char *msg);
}typedef behavior;
```

Každé správanie obsahuje:

- meno, ktoré by malo byť jedinečné
- prioritu, ktorá nám udáva ktoré správanie sa má vykonávať
- aktivačnú premennú, pomocou ktorej môžeme správanie naštartovať alebo zastaviť
- funkciu loop, ktorá je hlavná časť správania a vykonáva sa pri každom zavolaní správania
- funkciu handler, ktorá zabezpečuje posielanie správ medzi správaniami

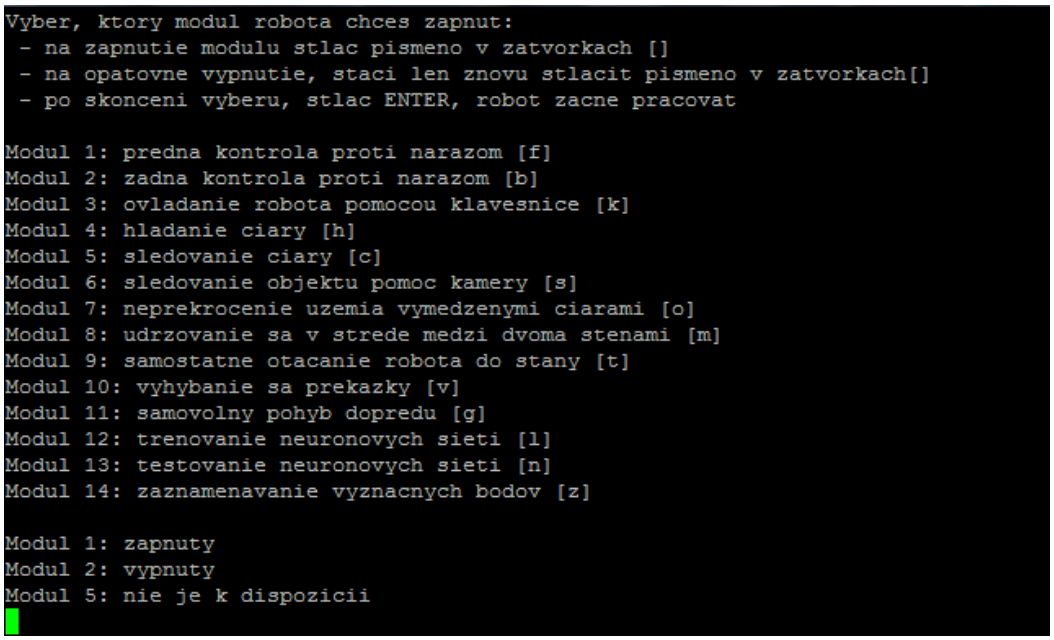
Štruktúra na zaregistrovanie odberateľov je implementovaná v základnom module, každé správanie ju dedí a vyzerá nasledovne:

```
struct message_subscriber {
      char *behavior;
      int wanted;
}typedef subscriber;
```

- `behavior` označuje, ktoré správanie sa registrovalo
- `wanted`, o ktorú premennú sa zaujíma

Každé správanie môže obsahovať aj svoje interné funkcie pre vlastnú potrebu. My sme si zaviedli u niektorých správaní funkciu *`setup()`*, pomocou ktorej si nastavíme hodnoty premenných pred jej každým naštartovaním.

Vývojárovi sme poskytli možnosť zvoliť si štartovanie robota v normálnom alebo v testovacom režime. Výber daného režimu závisí od nastavenia premennej *`RUN_ACTIVE`*. Ak táto premenná má hodnotu 1, robot štartuje automaticky všetky svoje správania a vykonáva svoju činnosť. Avšak, ak vývojár chce otestovať beh len niektorých svojich správaní nastaví si hodnotu na 0 a naštartuje sa mu testovací režim(Obr. 2.12). Dané správania si potom môže zapínať alebo vypínať. Spustenie behu robota realizuje klávesou ENTER a rovnakou klávesou preruší beh a vráti sa späť do testovacieho režimu.



Obr. 2.12: Ukážka testovacej obrazovky

2.3.4 Základné moduly správania architektúry

V tejto kapitole by sme chceli bližšie popísať základné správania robota.

Správanie na pohyb a navigáciu

Toto správanie má jednu z najdôležitejších úloh, pretože zaobstaráva pohyb robota, pomenovali sme ho *machine_room*. Jeho hlavnou úlohou je prijímať správy od ostatných a vyfiltrovať správy podľa prichádzajúcich priorít a na základe toho určiť rýchlosť a smer pohybu robota. Prioritu pre *machine_room* nastavujeme úplne najnižšiu, pretože toto správanie nikdy nerozhoduje o pohybe, iba ho vykonáva. Toto správanie je vždy automaticky spustené pri štarte robota.

Schéma správy, pomocou ktorej sa nastavuje smer pohybu vizerá nasledovne:

`(číslo priority koncového správania)#move#(priorita správania)#(číslo smeru)`

- číslo priority koncového správania si posielame, aby sa správa priradila danému správaniu, hlavne keď sa posiela z jednej platformy na druhú
- move – je klúčové slovo, ktoré vyjadruje nastavenie pohybu robota v danom smere
- priorita správania sa posiela preto, aby *machine_room* vedelo rozhodnúť, či zmení svoj súčasný pohyb

- Číslo smeru určuje smer, k dispozíciu máme 6 smerov: dopredu, dozadu, točenie na mieste vľavo, točenie na mieste vprevo, točenie po kružnici vľavo a točenie po kružnici vpravo
- Znak # slúži ako oddelovač klúčových hodnôt

Nastavovať rýchlosť motorov, môžeme na 3 stupne: odporúčaná rýchlosť pohybu od výrobcu, spomalený pohyb a zrýchlený pohyb. Schéma správy je nasledovná:

```
(číslo priority koncového správania)#speed#(priorita správania)#
(stupeň rýchlosti)
```

Ďalej sme pre vývojára pripravili možnosť nastaviť si hodnoty motorov samostatne a tým pádom si môže určiť rôznorodý pohyb robota. Správa má tvar:

```
(číslo priority koncového správania)#motors#(priorita správania)#
(hodnota otáčania ľavého motora)#(hodnota otáčania pravého motora)
```

Pomocou funkcie *handler* si správanie prečíta správu, kde si uchová informácie a na základe toho vyhodnocuje pohyb, využívame na to nasledovnú podmienku:

```
if(num <= NUM_OF_PRIORITY || CO_ROBIA_MOTORY[NUM_OF_PRIORITY]==-1)
                    {
                        NUM_OF_PRIORITY = num;
                        CO_ROBIA_MOTORY[NUM_OF_PRIORITY] = val;
                    }
```

Hodnoty num a val sú prečítané hodnoty zo správy. A podmienka hovorí, že ak nová hodnota priority je menšia alebo rovná ako aktuálna alebo hodnota pohybu pre túto prioritu je -1, čo znamená, že správanie s touto prioritou už neovláda robota, potom sa priradia príslušné prečítané hodnoty na aktuálne úkony. Na základe toho sa vykonáva určený pohyb robota.

Správania zabezpečujúce komunikáciu medzi Sbotom a Gumstixom

Na Gumstix sme si postavili rovnakú architektúru ako na Sbotovi naprogramovanú v jazyku C. O ich vzájomnú komunikáciu sa aj v jednom aj v druhom prípade starajú správania určené len na to. V obidvoch prípadoch sme ich pomenovali *dispatcher*. Obidve správania sú automaticky spustené pri štarte robota.

Ak si správania posielajú správu a nenachádzajú sa rovnakej platforme zavolá sa *handler dispatchera* a ten správu spracuje a pošle druhému *dispatcherovi* a on ju

už prepošle konkrétnemu správaniu, pre ktorého je správa určená. Rovnakým spôsobom to funguje obidvoma smermi.

Dispatcher na strane Sbota posiela správy pomcou funkcie `usart0_putchar(char c, FILE *stream)` a prijíma správy v obslužnej rutine prerušenia (ISR), ktoré nastváva vždy keď sa prijal nejaký znak zo sériovej linky. Využíva funkciu `usart0_getchar()`, ktorá vráti prijaty znak. Na strane Gumstix si musíme zabezpečiť otvorenie sériového portu sami, príkazom:

```
int fd = open("/dev/ttyO0", O_RDWR | O_NOCTTY | O_NONBLOCK)
```

Potom z daného portu čítame a zapisujem dáta:

čítanie - `read(fd, &buf, 1)`

zapísanie - `write(fd, c, 1)`

Dispatcheru na Gumstix sme priradili aj vlastnosti *machine_room* z dôsledku úrychlenia posielania dát sériovým portom. V našom prípade už *dispatcher* vyhodnotí zo svojich správaní, ktoré hodnoty s najvyššou prioritou sa majú poslať na Sbota, kde naďalej súťažia s povelmi vytvorenými správaniami na platforme Sbot. Snažili sme sa týmto spôsobom urýchliť komunikáciu medzi týmito platformami, nakoľko sériovým potom posielame už len jednu premennú o stave smeru.

Správania kontroly nárazu robota

Aby sme robota chránili pred poškodením, ktoré môžu vzniknúť hlavne nárazmi, implementovali sme dve správania na kontrolu nárazu. Prvé správanie zaobstaráva kontrolu vpredu, pri pohybe robota vpred a druhé sa stará o kontrolu vzadu pri pohybe vzad. Tieto dve správania majú najvyššie priority, pretože musia prebiť všetky iné a sú tiež automaticky zapntué pri štarte robota.

Prvé správanie sme pomenovali *front_control* a na svoju prácu využíva predný Sharp IR senzor, ktorým sníma vzdialenosť pred sebou. Ak táto vzdialenosť je príliš malá prikáže robotovi aby zastavil. Okrem toho využíva dva predné nárazníkové senzory, ktoré majú podobnú funkciu a to, že ak sú zatlačené prikazujú robotovi zastať.

Podmienka prednej kontroly:

```
if ((bumper_stat(BUMPER_0) ==0) || (bumper_stat(BUMPER_1) ==0) ||
((adc_vals[5]/ADC_REF_AVCC*1023) > (1023/ADC_REF_AVCC*1023-800)))
```

Ak nastane situácia pravdepodobného nárazu robot dostáva príkaz zastaviť a správanie si uloží do premennej hodnotu otočenia do výhodnejšej strany. Pod výhodnejšou stranou rozumieme stranu, kde pravdepodobnosť nárazu je nižšia. Túto vlastnosť zabezpečujeme pomocou Sharp IR senzorov, ktoré sú umiestnené na stranách robota.

Druhé správanie sme nazvali *back_control* a prostredníctvom nárazníkových senzorov sa stará aby robot zastavil ak tieto senzory sú zatlačené.

Podmienka zadnej kontroly:

```
if (bumper_stat(BUMPER_2) == 0 || bumper_stat(BUMPER_3) == 0)
```

2.3.5 Inštalácia knižnice OpenCV na Gumstixa

OpenCV knižnica je najvyužívanejšia a hlavne open source knižnica na spracovanie obrazu, preto sme sa rozhodli ju využiť aj v tejto práci na platforme Gumstix, ktorú budeme využívať na náročnejšie výpočty, kde spracovanie obrazu určite patrí.

Pri inštalácií sme sa stretli s problémom, že sa zaplnila RAM pamäť. Stačilo však vytvoriť dočasnú swap partíciu na microSD karte, čo však nie je možné počas bežnej prevádzky, keďže microSD karta používa pamäť typu flash s obmedzeným počtom opakovaných zápisov. Celková inštalácia knižnice trvala približne osem hodín.

Kompilácia programu C s využívaním OpenCV vyzerá nasledovne:

```
gcc –ggdb `pkg-config –cflags opencv` -o `basename test.c .c` test.c `pkg-config –libs opencv`
```

2.3.6 Získavanie obrazu z Caspa pomocou v4l a OpenCV

Akonáhle sme mali naimplementovanú a otestovanú OpenCV knižnicu pustili sme sa do práce na získanie obrazu z Caspa. Využili sme na to v4l. Celý priebeh a konkrétny postup je implementovaný v súbore *ziskanieObrazku.c*.

Priebeh procesu:

- Otvoríme si kameru: `fdevice=open(dev_name,O_RDWR|O_NONBLOCK, 0);`
- Nainicializujeme ju:

```
fmt.type = V4L2_BUF_TYPE_VIDEO_CAPTURE;
fmt.fmt.pix.width = 752;
fmt.fmt.pix.height = 480;
fmt.fmt.pix.pixelformat = V4L2_PIX_FMT_UYVY;
fmt.fmt.pix.field = V4L2_FIELD_NONE;
```

- Spustíme čítanie framov z kamery a aby sme sa vyhli problému z adaptáciou kamery v prostredí, spracovávať začíname najskôr tretí frame v poradí pomocou funkcie:

```
yuvImg = load_raw_image(buffers[buf.index].start,
buffers[buf.index].length);
```

- Funkcia nám uloží obrázok do YUV formátu, obrázok zkonvertujeme to RGB, pomocou vlastnej implementácie konverznej funkcie a uložíme na microSD kartu
- Nakoniec uvoľníme *buffers*, zaviereme kameru a vrátime obrázok v pamäti
    - `free (buffers)`
    - `close(fdevice)`
    - `return OutImage`

2.4 Experimentálna časť

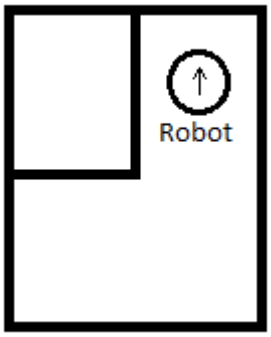
2.4.1 Experiment č.1 – Náhodný pohyb robota bez kolízie v prostredí

Konceptuálny opis

Jedná sa o základný experiment. Testovanie pohyblivosti robota a schopnosť vnímania prekážok a ich vyhýbaniu sa. Robot by sa mal plynule pohybovať a pri detekovaní prekážky by mal zmeniť smer jazdy. Nemala by nastať kolízia robota a prekážky.

Experimentálny setup

Robot sa nachádza v prostredí ohraničenom kovovými stenami, veľkosť prostredia predstavuje 1m x 1,5m. V jednom z rohov je umiestnená prekážka v tvare L, ktorá odrezáva časť prostredia a tým prostredie nie je klasické obdĺžnikové prostredie (Obr. 2.13).



Obr. 2.13: Ukážka prostredia pre experiment č.1

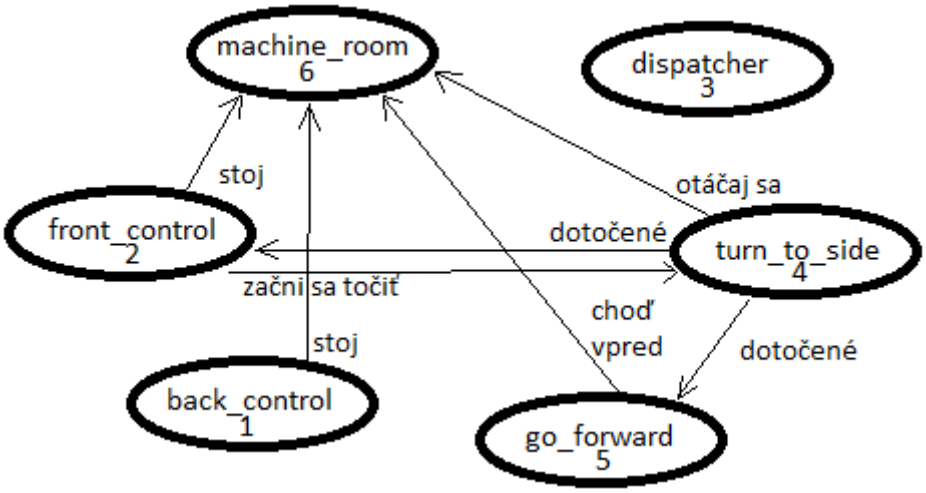
Senzorické vybavenie robota využíva tri sharp IR senzory a štyri nárazníkové senzory. Robot pri štarte bol umiestnený v užšej časti prostredia smerujúci k stene.

Realizácia

Všetky základné správania robota boli spustené a k nim sme si dodali ďalšie, prvým správaním, *go_forward* sme zabezpečili robotovi samostatný pohyb vpred. Druhým správaním, *turn_to_side* robota otáčame do strán. Robot sa otáča do svojej výhodnejšej strany, tá je určená a poslaná správaniu od správania *front_control*. Týmito modulmi sme stanovili celkové správanie robota pre prvý experiment.

Robot na začiatku má stanovený pohyb vpred na neurčitú dobu. Ak je zistená prekážka *front_control* oznámi aby robot zastal a dá povel správaniu *turn_to_side*, aby sa začal točiť a prestane detekovať prekážku, pretože už pri prvom pohybe otočenia by ju nameral a to nechceme. Preto akonáhle *turn_to_side* dokončí svoj pohyb oznámi správaniam *front_control* a *go_forward*, že je dotočené a môžu realizovať svoju činnosť.

Schéma správaní pre daný experiment(Obr. 2.14):



Obr. 2.14: Schéma správaní pre experiment č.1. Elipsy predstavujú správania a čísla v nich konkrétnu prioritu správania. Šípky predstavujú posielanie správ v stanovenom smere. Popis na šípkach pre správy je len orientačný.

Výsledky

Po mnohonásobnom testovaní a ladení systému sme sa dopracovali na vyhovujúci stav, kde sa robota správal, tak ako sme očakávali. Experiment považujeme za úspešný a jeho kratší priebeh sme zdokumentovali na video, ktoré je priložené v elektronickej prílohe.

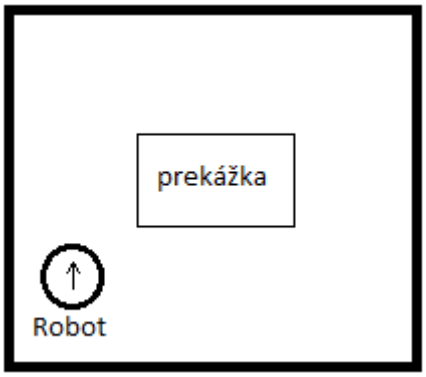
2.4.2 Experiment č.2 – Ovládanie robota cez Bluetooth pomocou klávesnice vo vymedzenom priestore

Konceptuálny opis

Experiment slúži na otestovanie komunikácie robota s iným zariadením prostredníctvom Bluetooth zariadenia, otestovaním funkčnosti line senzorov a samozrejme využívanie kontroly nárazu, ktorú už máme otestovanú a plne funkčnú. Robot bude komunikovať s našim počítačom, pomocou ktorého budeme ovládať jeho pohyb. Robot by nemal opustiť priestor vymedzený čiarou aj napriek tomu, že ho tam budeme posielať. Nakoniec by taktiež nemala nastať žiadna kolízia robota a prekážky.

Experimentálny setup

Robot sa nachádza vo voľnom prostredí, kde len na povrchu je ohraničená zóna jeho pohybu čiernou páskou. Povrch prostredia je biely a zóna má približne rozmery 1m x 1m. V strede tejto zóny je položená prekážka (Obr. 2.15).



Obr. 2.15: Ukážka prostredia pre experiment č.2

Realizácia

Spustené sú základné správania a k nim sme vytvorili nové správania a to *keyboard_control* a *no_overrun_line*. Prvé menované správanie sa stará o ovládanie robota pomocou klávesnice počítača, ovládanie je nastavené na klávesy W- vpred, S – vzad, A – vľavo, D – vpravo a MEDZERNÍK – stop. V ISR si uložíme hodnotu prijateho znaku a oznámime, že prišiel nový znak.

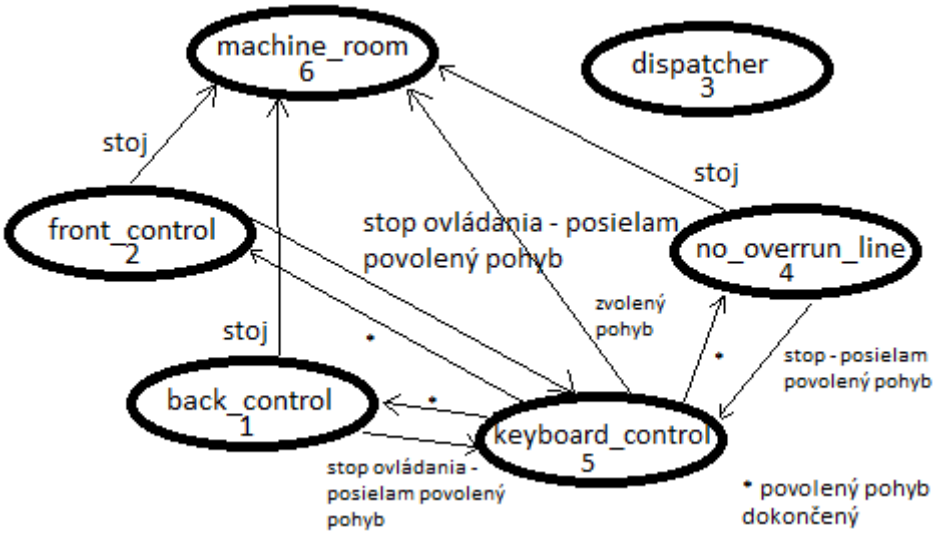
```
newchar = usart1_getchar(0);
new_packet = 1;
```

Správanie *no_overrun_line* slúži ako kontrolné správanie, ktoré kontroluje pomocou line senzorov, či sa robot dostal na čiaru. Podmienka kontroly:

```
int max_value = (1023 / ADC_REF_AVCC) * 1023 - 300;
if (((adc_vals[3] / ADC_REF_AVCC) * 1023 > max_value)||
((adc_vals[2] / ADC_REF_AVCC) * 1023 > max_value))
```

Akonáhle sa správanie ovládania robota dostane do konfliktu s kontrolami, tak nemôže pokračovať, pretože pre kontroly sme nastavili vyššie priority. Jediný možný spôsob ako sa dostaneme z tejto situácie je, že využijeme opačnú operáciu pohybu. Čiže ak je robota pred prekážkou alebo na čiare a dostal sa tam pohybom vpred, jediný možný pohyb je teraz vzad a naopak. Keď je pohyb dokončený správanie dostane o tom správu a pokračuje vo svojom kontrolovaní.

Schéma správaní pre daný experiment(Obr. 2.16):



Obr. 2.16: Schéma správaní pre experiment č.2. Elipsy predstavujú správania a čísla v nich konkrétnu prioritu správania. Šípky predstavujú posielanie správ v stanovenom smere. Popis na šípkach pre správy je len orientačný.

Výsledky

Podarilo sa nám úspešne implementovať, otestovať a spojazniť požadované správanie robota. Pri testovaní sme narazili na niekoľko problémov, ktoré sa nám podarilo vyladiť. Napríklad jeden z problémov nastal pri cúvaní od prekážky, kde nám robot podskočil a nameral hodnotu čiary a potom čakal na to, kým znovu cúvneme a často sa to opakovalo. Čiže veľa krát sme mohli len cúvať, kým priebeh cúvania nebol perfektný. Tento problém sme vyladili tým, že ak robot docúval a stál ďaleko od prekážky alebo na bielom povrchu, poslali sme správu správaniu na ovládanie robota, že môže teraz vykonávať hocijaký pohyb. Experiment hodnotíme ako úspešný a jeho videodokumentáciu prikladáme k prílohám.

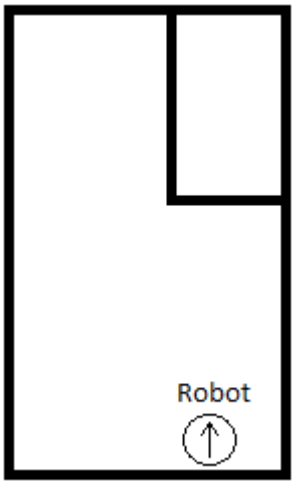
2.4.3 Experiment č.3 – Udržovanie sa v strede koridoru

Konceptuálny opis

Pomocou tohto experimentu si otestujeme funkčnosť Sharp IR senzorov, pomocou ktorých sa budeme chcieť udržať v strede medzi dvoma stenami, takzvanom koridore. Cieľ teda je prechod robota koridorom s čo najpresnejším udržaním sa v strede

Experimentálny setup

Robot má k dispozícií tri senzory na meranie vzdialenosti, umiestené vpredu a po stranách robota. Nachádza sa v prostredí ohraničenom kovovými stenami o veľkosti 1m x 3m. V pravej hornej časti sme postavili prekážku na zúženie koridoru, aby sme odpozorovali správanie robota pri zmene šírky koridora (Obr. 2.17).



Obr. 2.17: Ukážka prostredia pre experiment č.3

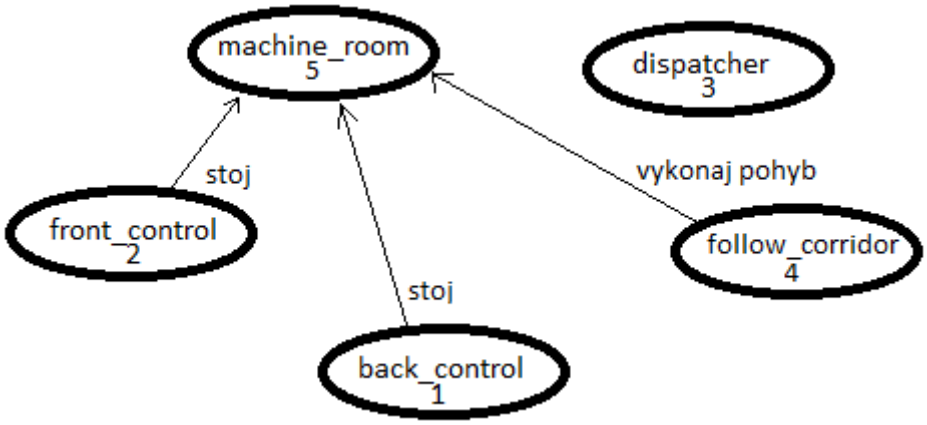
Realizácia

Všetky základné správania sú automaticky zapnuté a vykonávajú svoju činnosť. Na snímanie vzdialenosti po stranách robota a následné centrovanie sme si vytvorili nové správanie, ktoré sa nám bude o to starať. Pomenovali sme ho *follow_corridor*. Jeho úloha je získať vzdialenosti na lavej a pravej strane robota, čo sme aplikovali nasledovne:

```
left_distance = (adc_vals[6] / ADC_REF_AVCC) * 1023;
right_distance = (adc_vals[4] / ADC_REF_AVCC) * 1023;
```

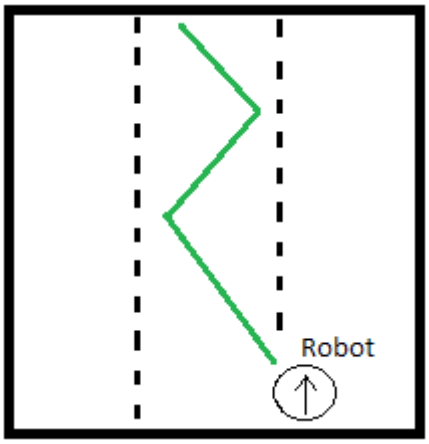
Potom tieto vzdialenosti porovnávame, otočíme robota do strany s väčsou vzdialenosťou a pustíme smerom vpred až kým sa znovu robot nedostane do situácie že smerovaná vzdialenosť sa stane kratšia ako opačná. V tomto prípade robot vykonáva tú istú činnosť len do opačného smeru. Týmto spôsobom sme sa snažili zabezpečiť pohyb robota stredom s akceptovateľnou odchylkou, cikcakovitým spôsobom(Obr. 2.19).

Schéma správaní pre daný experiment(Obr. 2.18):



Obr. 2.18: Schéma správaní pre experiment č.3. Elipsy predstavujú správania a čísla v nich konkrétnu prioritu správania. Šípky predstavujú posielanie správ v stanovenom smere. Popis na šípkach pre správy je len orientačný.

Schéma pohybu robota pre experiment, ktorý očakávame od vytvoreného správania:



Obr. 2.19: Očakávaný pohyb robota pri experimente č.3

Výsledky

Pohyb robota zodpovedal našej realizácií a robot sa udržiaval v strede koridoru. V niektorých prípadoch pohyb robota vpred nebol plynulý a jemne zatáčal do jednej strany. To mohlo spôsobiť napríklad prešmykovanie jedného z kolies na povrchu. Potom pri testovaní pridania prekážky, ktorá spôsobila zúženie koridoru, robot namiesto vzdiaľovania sa od strany sa k nej pomaly približoval. Tento problém sme vyriešili tak, že pri zmene pohybu sme si uložili kratšiu štartovaciu vzdialenosť, tú od ktorej sa chceme vzdialiť. A ak nastala situácia, že nová nameraná vzdialenosť rovnakej strany je menšia ako štartovacia tak sme povolili robotovi sa znovu otočiť do vzdialenejšieho smeru. Toto opatrenie nám

vyriešilo vyskytnutý problém a robot počas zvyšného testovania sa pohyboval podľa očakávaní. Experiment sme nakrútili na video a záznam sme priložili medzi prílohy.

2.4.4 Experiment č.4 – Sledovanie čiary a vyhýbanie sa prekážkam

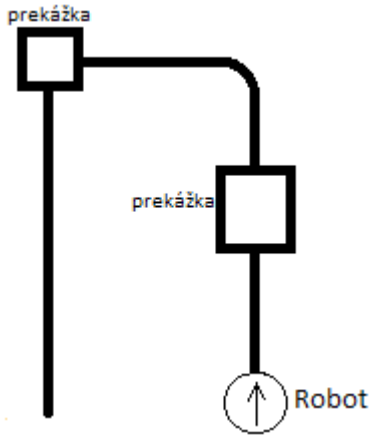
Konceptuálny opis

V prvých experimentoch sme si vyskúšali a otestovali prácu so senzormi a ich funkcionalitou. V tomto experimente budeme chcieť tieto vedomosti zúžitkovať na úlohu sledovania čiary a vyhýbaniu sa prekážok na trase vedenej čiarou. Robot by mal byť schopný sledovať čiaru a ak zaregistruje prekážku, vyhnúť sa kolízii a pokúsiť sa vrátiť späť na čiaru a pokračovať v trase.

Experimentálny setup

Pri experimente budeme využívať line senzory na detekovanie čiary, nárazníkové senzory a Sharp IR senzory na detekovanie prekážok. Robot sa nachádza v prostredí na bielom povrchu, kde je vyznačená trasa čiernou čiarou. Na čiare sa vyskytujú dve prekážky a jedna deväťdesiat-stupňová zákruta(Obr. 2.20):

- prekážka, ktorá preruší čiaru a jej pokračovanie pôjde plynule za ňou
- prekážka, ktorá preruší čiaru a pokračovanie čiary je zo strany prekážky, simulovaná pravouhlá zákruta

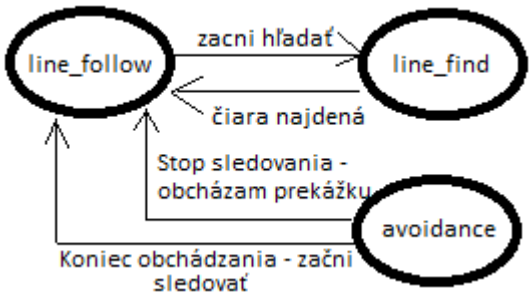


Obr. 2.20: Ukážka prostredia pre experiment č.4

Realizácia

Pri experimente sú samozrejme zapnuté základné správania a k ním sme vytvorili ďalšie správania na vyriešenie experimentu. Prvé implementované správanie, *line_find* sme vytvorili kvôli potrebe nájdenia čiary. Môže nastať

situácia, že robot stratí čiaru a nevie ju detekovať, vtedy sa spustí toto správanie. Jeho beh skončí akonáhle jeden z line senzorov detekuje čiaru. Jeho správanie spočíva v tom, že najskôr sa robot otočí jemne do jednej strany, potom do druhej a ak nie je nájdená čiara, pokračuje smerom vpred po dobu troch sekúnd. Po ich uplynutí jeho proces prebieha od začiatku. Druhé implementované správanie, *line_follow* nám zabezpečuje sledovanie čiary. Tento proces sledovania sme navrhli pomocou kruhového pohybu. Ak sníma čiaru jeden z line senzorov, tak vzdialenejšie koleso sa točí smerom vpred a bližšie stojí. Jeho pohyb sa vykonáva až kým čiaru nezačne snímať druhý line senzor. Tieto senzory behajú dnu a von z čiary a striedavý kruhový pohyb motorov nám umožní cikcakovým spôsobom sledovať čiaru. V prípade, že ani jeden z line senzorov nesníma čiaru, pošle správu správaniu *line_find*, aby ju začal hľadať(Obr. 2.21). Posledným implementovaným správaním, *avoidance* sme navrhli obchádzanie prekážok. Správanie sa začne vykonávať akonáhle je detekovaná prekážka a končí ak pri obchádzaní prekážky line senzory spozorujú čiaru.



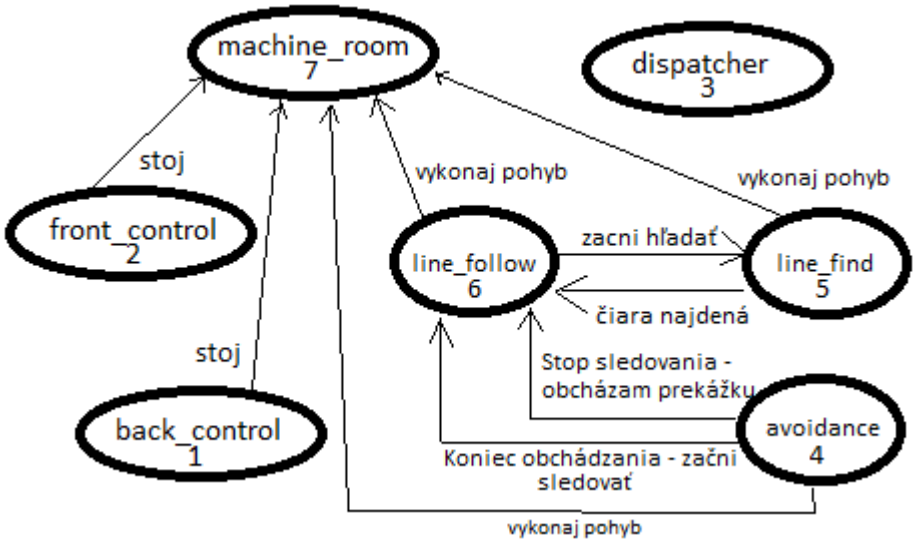
Obr. 2.21: Schéma komunikácie medzi správaniami

Obchádzanie prekážky sme realizovali nasledovne:

1. ak bola detekovaná prekážka pred robotom, otočí sa doľava
2. po dotočení si odmeria bočným Sharp IR senzorom vzdialenosť od prekážky do premennej *dist* a pokračuje rovno
3. pohyb vpred vykonáva až kým *dist* nie je väčsia ako aktuálna hodnota na tom istom senzore, čo nám indikuje, že prekážka skončila
4. posunie sa o určitú hodnotu vpred, otočí doprava a po otočení vyráža vpred o určitú hodnotu aby bol vedľa prekážky
5. odmeria si vzdialenosť od prekážky do premennej *dist* a pokračuje vpred

6. ak znovu aktuálna nameraná hodnota je väčšia ako *dist*, čo znamená koniec prekážky, process obchádzania sa vracia do bodu 4. Touto rekurziou dokážeme obehnúť komplet celú prekážku
7. ak je prekážka členitá, tak predný sensor nameria počas behu prekážku pri pohybe vpred a celý process obiehania ide od začiatku

Schéma správaní pre daný experiment(Obr. 2.22):



Obr. 2.22: Schéma správaní pre experiment č.4. Elipsy predstavujú správania a čísla v nich konkrétnu prioritu správania. Šípky predstavujú posielanie správ v stanovenom smere. Popis na šípkach pre správy je len orientačný.

Výsledky

Pri testovaní sme spozorovali pár problémov, ktoré vznikli nerovným pohybom a zadnou podopieracou časťou robota, ktorá sa niekedy zasekla a následne boli podvihnuté line senzory. Dôsledkom toho bolo, že senzory na bielom povrchu namerali slabú intenzitu, čo znamenalo čiernu farbu, teda čiaru. Aby sme sa čo najčastejšie vyhýbali takýmto situáciám, pridali sme podložky na upevňovaciu časť senzorov, aby sme ich posunuli bližšie k podlahe. Nakoniec sa nám navrhnutým prostredím podarilo prejsť, preto hodnotím tento experiment ako úspešný. Videodokumentácia zdolávania prekážok a sledovania čiary je priložená v prílohách. Zdokumentovali sme aj samostatné obchádzanie prekážky na otestovanie nášho rekurzívneho algoritmu, ktorý zabezpečuje obchádzanie. Videozáznam prikladáme k prílohám.

2.4.5 Experiment č.5 – Sledovanie predmetu pomocou kamery

Konceptuálny opis

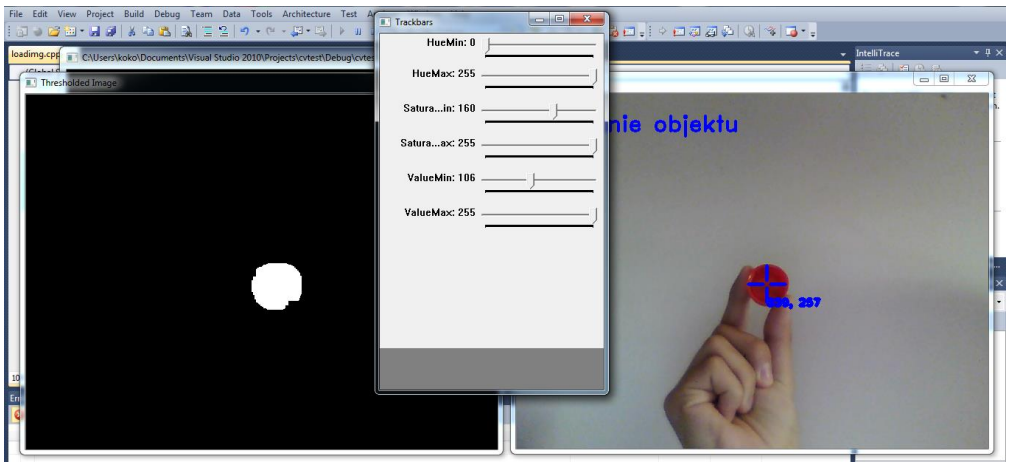
Experimentom otestujeme navrhnutú komunikáciu medzi našimi dvoma platformami. S využitím prenosu údajov z Gumstix na Sbot by mal robot na základe získaného obrazu z kamery zvládnuť sledovať navrhnutý predmet.

Experimentálny setup

Robot sa nachádza v prostredí bez obmedzení. Za sledovaný predmet sme si zvolili predmet červenej farby v tvare kruhu. V experimente sme využili dva sledované predmety:

- červený vrchnák z flaše v tvare kruhu
- pevný bod červenej farby kruhového tvaru, nalepený na stenu v prostredí

Na získanie prahového rozsahu pre červený objekt využívame externý program, spojaznený na osobnom počítači. Program otvorí webkameru a získava z nej obraz. Pomocou trackbarov si nastavujeme rozsah pre minimálnu a maximálnu hodnotu. Okrem trackbarov máme k dispozícii dva výstupné obrazy, prvý nám ukazuje v čiernobielom obraze získaný obraz určený rozsahom. V druhom môžme vidieť obraz, ktorý nám dáva kamera spolu s označením nájdeného objektu(Obr. 2.23).



Obr. 2.23: Ukážka externého programu na získanie prahového rozsahu pre farbu sledovaného objektu

Realizácia

Okrem základných správaní sme na vyriešenie experimentu implementovali nové správanie *follow_object*. Toto správanie je umiestnené na Gumstix a zabezpečuje nám hodnotu, stred nájdeného objektu, v ktorej časti obrazu sa objekt nachádza. Potom sa táto hodnota porovnáva nami nastavenými podmienkami, ktoré nám určia či predmet je v ľavej, pravej alebo strednej časti obrazu. Výsledok sa posiela na Sbota, ktorý vykoná na základe daného výsledku výsledný pohyb. Napríklad nájdená hodnota je na bode 300, jedná sa o hodnotu na x-ovej osi, pričom vieme, že obraz získaný z Caspa má veľkosť 752x480 pixelov. Podmienky sme nastavili nasledovne:

- ak je hodnota menej ako 200 chceme robota natočiť doľava
- ak je hodnota viac ako 500 chceme robota natočiť doprava
- ak je hodnota medzi 200 a 500 robot pôjde rovno

V našom ukážkovom príklade je hodnota 300, teda chceme aby robot išiel rovno. Na Sbota sa posiela príkaz určujúci jazdu, čiže 1. *Dispatcher* správu spracuje a pošle ju na *machine_room,* ktoré vykonáva pohyb robota. Ak je pohyb dokončený, *dispatcher* pošle správaniu správu, že chce ďalšie pokyny a proces prebieha od začiatku.

Follow_object si ako prvé získa obraz pomocou funkcie *grabImage(),* ktorú sme popísali vyššie(Obr. 2.24). Ďalej v experimente budeme využívať funkcie knižnice OpenCV na prácu s obrazom. V nasledujúcom kroku má vyvojár k dispozícií naprogramované vylepšenie obrazu pomocou histogramov, ktorých použitie môže zvážiť na základe vplyvov prostredia, svetla a iných vonkajších nezrovnalosti. Získaný obraz si konvertujeme do HSV farebného modelu(Obr. 2.25), funkciou:

```
cvCvtColor(image, srcHSV, CV_RGB2HSV);
```

Výstupom je *srcHSV*, ktorý je zároveň vstup pre funkciu *cvInRangeS,* ktorá určuje, ktoré prvky patria do rozsahu určenom pre minimálne a maximálne hodnoty HSV. Hodnoty sme si nastavili pomocou externého programu a výsledné hodnoty sme získali nasledovne:

```
hueMin = 0;     //nastavenie pre červený objekt
```

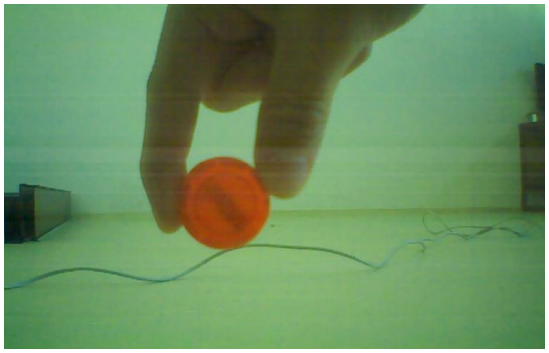
```
        hueMax = 255;
        saturationMin = 160;
        saturationMax = 255;
        valueMin = 106;
        valueMax = 255;
    CvScalar lower = CV_RGB(hueMin,saturationMin,valueMin);
    CvScalar upper = CV_RGB(hueMax,saturationMax,valueMax);
```

Výsledné hodnoty sme vložili do funkcie *cvInRangeS* a na výstup sme dostali čierno biely obraz *threshold*(Obr. 2.26), kde bielou farbou sú len prvky určené rozsahom o ktoré sa zaujímame:

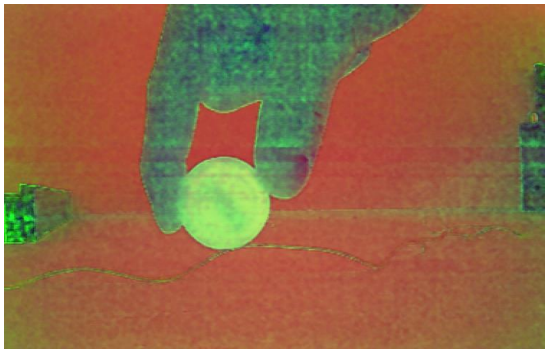
```
cvInRangeS (srcHSV, lower, upper, threshold);
```

Nakoniec *threshold* vložíme do funkcie *cvHoughCircles*, ktorá nájde kruhové objekty v čierno bielom obraze pomocou Houghovej transformácie. Výsledný, nájdený, stredový bod napokon porovnáme s našimi podmienkami a získavame výslednú hodnotu pre pohyb robota(Obr. 2.27).

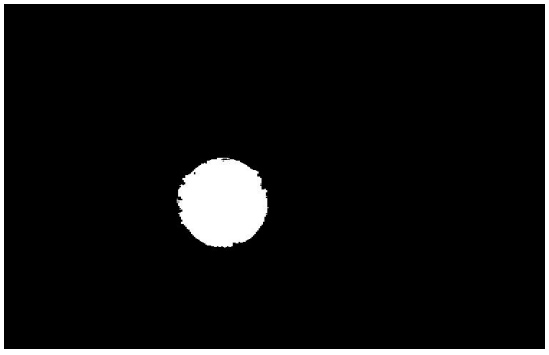
Priebeh procesu v obrázkoch:



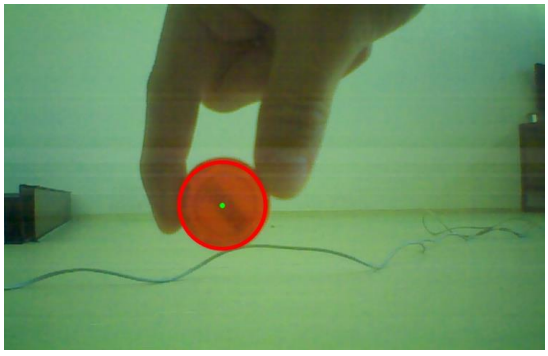
Obr. 2.24: Obraz z kamery



Obr. 2.25: Obraz zkonvertovaný do HSV

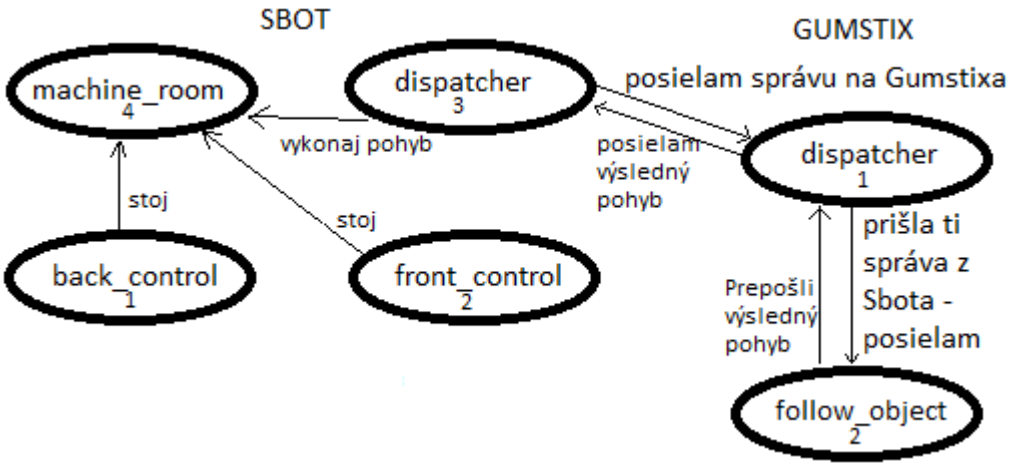


Obr. 2.27: Threshold obrazu



Obr. 2.28: Výsledný nájdený objekt v obraze

Schéma správaní pre daný experiment(Obr. 2.28):



Obr. 2.28: Schéma správaní pre experiment č.5. Elipsy predstavujú správania a čísla v nich konkrétnu prioritu správania. Šípky predstavujú posielanie správ v stanovenom smere. Popis na šípkach pre správy je len orientačný.

Výsledky

Pri viacnásobných testovaniach sme spozorovali viaceré vonkajšie vplyvy na kvalitu obrazu, ktorý nám poskytovala kamera. Pri umelom osvetlení prostredia, lampou kamera vracala obraz s jemným odtieňom žltej. Z toho dôvodu sme zapracovali vylepšenie obrazu pomocou histogramov. Najlepšie výsledky sme dosiahli pri normálnom dennom svetle, kde ukážku môžeme vidieť vyššie. Výsledkom experimentu bolo, že robot zvládol sledovať predmet, ktorý sme si pripravili. Bol schopný ísť za ním rovno a taktiež sa za ním otáčať. V druhej časti experimentu, kde úlohou robota bolo prísť pred červený kruh nalepený na stene v prostredí, robot úlohu zvládol a dokázal prísť pred daný objekt a pred ním zastať. To nám zabezpečilo základné správanie *front_control*, ktoré malo vyššiu prioritu a robot sa pri detekovaní steny riadil jeho pokynmi. Videozáznamy sú pripojené v prílohách.

2.4.6 Experiment č.6 – Zaznamenávanie význačných bodov do pamäti

Konceptuálny opis

Experiment slúži na nájdenie význačných bodov v obraze a následne ich uloženie do pamäti. Užívateľ si sám bude môcť určiť miesto, o ktoré sa zaujíma, pomocou ovládania robota klávesnicou počítača a vykonať akciu na uloženie význačných bodov do pamäti. Spracovanie obrazu, algoritmus na nájdenie

význačných bodov a úložisko údajov budeme realizovať prostredníctvom Gumstix a ovládanie robota a vyvolanie akcie sa spustenie celého procesu získania bodov budeme realizovať na Sbotovi.

Experimentálny setup

Prostredie robota bude ohraničené kovovými stenami, rovnako ako v prvom experimente (Obr 2.1). Robota môžme umiestniť do stredu prostredia, ale jeho umiestenie pre nás momentálne nie je zaujímavé nakoľko, ho budeme ovládať do pre nás zaujímavej časti. Na steny prostredia sme nalepili rozličné obrázky o ktoré sa budeme zaujímať.

Realizácia

Základné moduly ostávajú všetky zapnuté. Použijeme správanie *keyboard_control* na ovládanie robota a vytvoríme si nové správanie *key_point*, pomocou ktorého budeme posielať správu na Gumstix, nech uloží obraz s význačnými bodmi do pamäte.

Posielanie správy sme nastavili na stlačenie klávesy "n", poslanie správy je nasledovné:

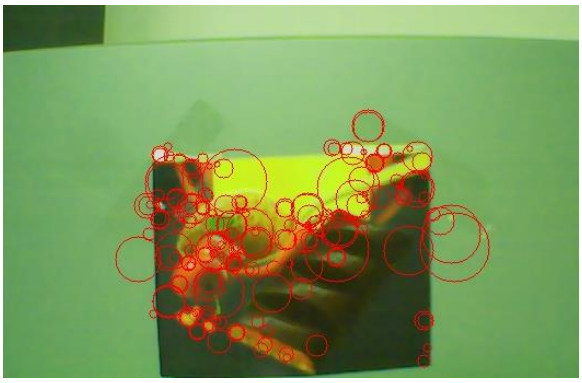
```
sprintf(kp_record,"%d#savekey", D_KEYPOINT_PRIORITY);
beh = return_handler("save_key_point");
beh.handler(sizeof(kp_record), kp_record);
```

Na strane Gumstix sme si vytvorili správanie *save_key_point,* ktoré má za úlohu uložiť obraz s význačnými bodmi do pamäti. Obraz z kamery si správanie získa pomocou funkcie *grabImage()* a ďalej s ním pracuje zo pomoci knižnice OpenCV. Ako prvé si nastavujeme parametre algoritmu, ktoré budeme neskôr používať, pomocou funkcie *CvSURFParams* a jej inštancie *pams*.

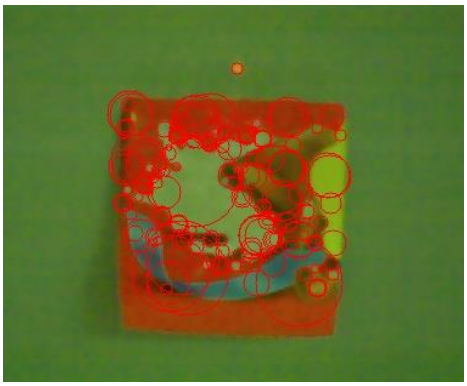
Nastavenia:

- nastavenie prahovej hodnoty: `pams.HessianThreshold = 300`
- počet vrstiev v každej oktáve: `pams.nOctaveLayers = 4`
- počet oktáv použitých na extrakciu: `pams.nOctaves = 3`
- nastavenie orientácie 0-počíta sa /1 – nepočíta sa: `pams.upright=1`

Spustíme extrakciu na získaný obraz funkciou *CvExtractSURF*, ktorá nám vráti význačné body a deskriptory bodov. Prejdeme cez všetky nájdené body, zaznačíme ich výskyt v obraze a uložíme ich do pamäte. Výsledný obraz si nakoniec tiež uloží (Obr 2.29, Obr 2.30).

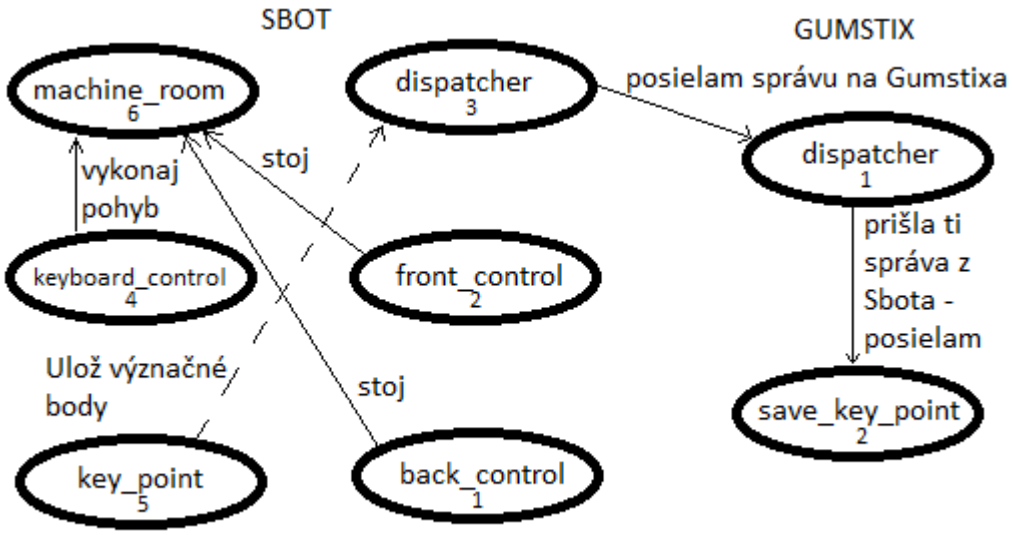


Obr. 2.29: Ukážka č.1 pre nájdené význačné body v obraze



Obr. 2.30: Ukážka č.2 pre nájdené význačné body v obraze

Schéma správaní pre daný experiment(Obr. 2.31):



Obr. 2.31: Schéma správaní pre experiment č.6. Elipsy predstavujú správania a čísla v nich konkrétnu prioritu správania. Šípky predstavujú posielanie správ v stanovenom smere. Popis na šípkach pre správy je len orientačný. Prerušovaná šípka určuje, že správanie posiela správu správaniu na inej platforme, ktorá je posielaná cez *dispatcher*.

Výsledky

V experimente sa nám podarilo implementovať ovládanie robota a pomocou príkazu od užívateľa poslať správu Gumstix, kde prebehol proces nájdenia význačných bodov a ich uloženie do pamäti. Precvičili sme si prácu s obrazom, získaným z kamery a jeho spracovanie. Tento experiment je vhodným základom pre vyššie ciele na prácu s význačnými bodmi. Pre budúcnosť, užívateľ má už k dispozícií nájdenie význačných bodov a môže sa zamerať na ich porovnávanie, lokalizáciu v prostredí alebo iné zaujímavé experimenty.

2.4.7 Experiment č.7 – Natrénovanie pohybu robota v prostredí bez kolízií pomocou neurónových sieti

Konceptuálny opis

Experiment bude pozostávať z dvoch častí. V prvej časti budeme získavať trénovacie dáta prostredníctvom ovládania robota a zapisovania nameraných hodnôt zo senzorov Sharp IR a nami vykonanej pohybovej akcie do pamäti. Trénovanie prispôsobíme pohybu v prostredí bez kolízií. Trénovacie dáta spracujeme a natrénujeme neurónovú sieť. V druhej časti využijeme natrénovanú neurónovú

sieť, kde jej vstupom budú namerané hodnoty senzorov Sharp IR a výstup bude predikovaná hodnota pohybu robota. Robot pomocou takto natrénovanej neurónovej siete by sa mal vedieť pohybovať v prostredí bez kolízií.

Experimentálny setup

Robot sa bude náchadzať v prostredí ohraničenom kovovými stenami s rozmermi 1,5x1m. Trénovacie aj testovacie dáta budeme získavať pomocou všetkých troch Sharp IR umiestnených v prednej, ľavej a pravej časti robota.

Realizácia

Kvôli realizácií a testovaniu samovoľného, bezkolízneho pohybu sme vypli základné správania pre kontrolu nárazu v prednej, *front_control* a zadnej, *back_control* časti robota. Ostatné základné správania ostali zapnuté.

Na získanie trénovacích dát sme si vytvorili samostatné správanie *train_neuron_net*, ktorým ovládame robota smerom vpred, doľava a doprava. Spolu s nameranými hodnotami zo senzorov Sharp IR posielame vykonanú akciu na Gumstix, kde sa spracuje daná správa a ďalšie správanie *train_neuron* ukladá hodnoty do pamäti. Štruktúra dátového súboru pozostáva v prvom riadku z troch čísel:

- celkový počet dát
- počet vstupných premenných
- počet výstupných premenných

Ďalej nasledujú len konkrétne dáta. Namerané hodnoty sme normalizovali do intervalu < 0,1 >. Výstupné hodnoty nadobúdajú hodnoty v rozsahu < -1,1 > , preto sme prispôsobili akcie pohybu na tento rozsah.

- pohyb vpred zaznamenávame hodnotou 1
- pohyb vľavo hodnotou 0
- pohyb vpravo hodnotou -1

Ukážka trénovacích dát:

```
500 3 1
0.680352 0.301075 0.226784
0
0.378299 0.247312 0.187683
1
0.379277 0.278592 0.269795
1
0.387097 0.398827 0.650049
1
0.392962 0.347996 0.586510
-1
```

Trénovanie neurónovej siete sme realizovali pomocou knižnice FANN, ktorá nám vytvorila natrénovanú sieť, ktorú budeme neskôr využívať pri testovaní. Proces trénovania je nasledovný:

- vytvorenie štruktúry pre knižnicu so vstupnými údajmi – počet vrstiev, počet vstupov, počet skrytých neurónov, počet výstupov

```
struct fann *ann = fann_create_standard(3, 3, 6, 1);
```

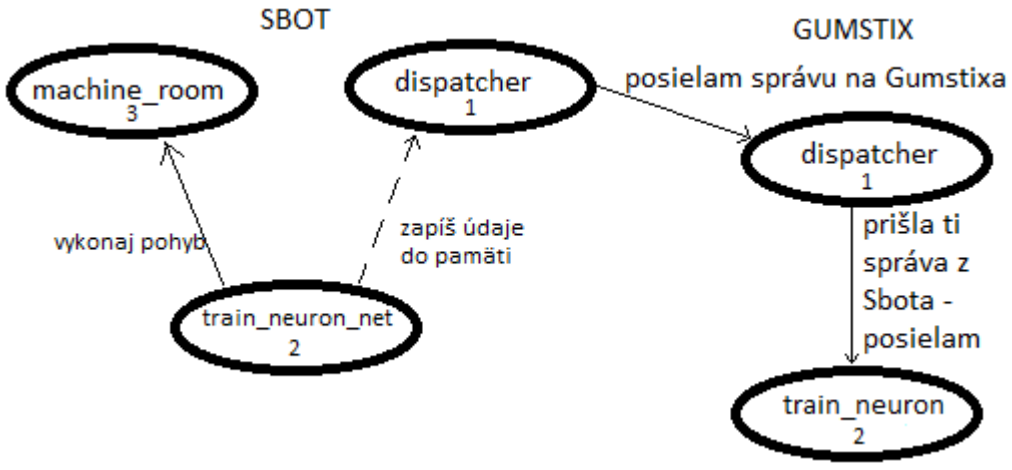
- natrénovanie siete

```
fann_train_on_file(ann, "data.txt", 5000, 10, 0.0001);
```

- uloženie natrénovanej siete, vstup – natrénovaná štruktúra, výstup – názov uloženej siete v pamäti

```
fann_save(ann, "gum_neuron.net");
```

Schéma správaní pre trénovanie experimentu (Obr. 2.32):



Obr. 2.32: Schéma správaní pre trénovanie experimentu č.7. Elipsy predstavujú správania a čísla v nich konkrétnu prioritu správania. Šípky predstavujú posielanie správ v stanovenom smere.

Popis na šípkach pre správy je len orientačný. Prerušovaná šípka určuje, že správanie posiela správu správaniu na inej platforme, ktorá je posielaná cez *dispatcher*.

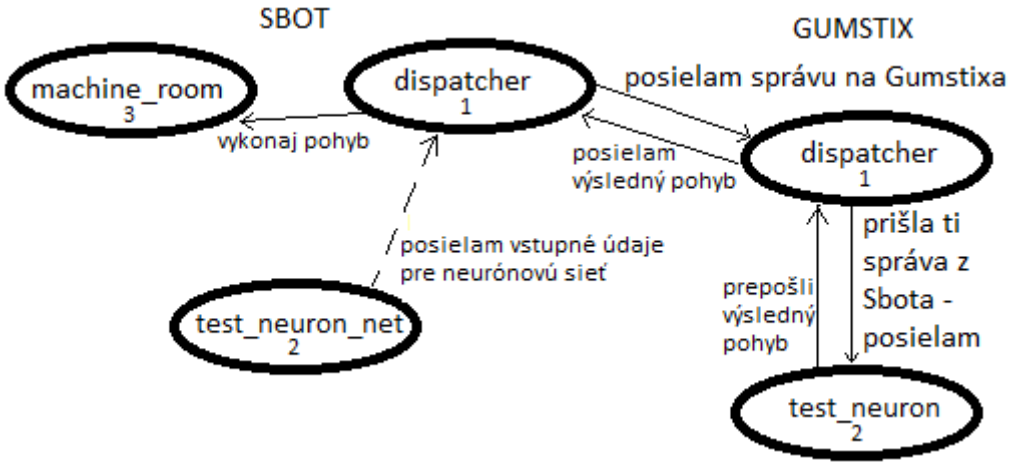
Na testovanie sme si vytvorili nové správanie *test_neuron_net*, umiestené na Sbotovi, ktoré len preposiela namerané hodnoty zo senzorov na Gumstix, akonáhle je dokončený predchádzajúci pohyb. Gumstix spracuje správu a nové správanie *test_neuron* si načíta natrénovanú neurónovú sieť:

```
struct fann *ann = fann_create_from_file("gum_neuron.net");
```

Potom na základe prečítaných dát získame výstupnú hodnotu za pomoci natrénovanej neurónovej siete, ktorá je spätne poslaná Sbotovi, ktorý vykonáva prislúchajúci pohyb.

```
float *calc_out = fann_run(ann, input);
```

Schéma správaní pre testovanie experimentu(Obr. 2.33):



Obr. 2.33: Schéma správaní pre testovanie experimentu č.7. Elipsy predstavujú správania a čísla v nich konkrétnu prioritu správania. Šípky predstavujú posielanie správ v stanovenom smere. Popis na šípkach pre správy je len orientačný. Prerušovaná šípka určuje, že správanie posiela správu správaniu na inej platforme, ktorá je posielaná cez *dispatcher*.

Výsledky

Výsledný pohyb robota bol veľmi dobrý a dokázal sa pohybovať po prostredí bez kolízie. Avšak v jeho trénovacích dátach bolo veľké množstvo dát, ktoré mali výstupný pohyb vpred, preto sa robot dostával pri testovaní blízko k stene. No k nárazu nedošlo. Vyskúšali sme preto vylepšiť trénovacie dáta, tak že sme počet dát pre každý pohyb vyrovnali, čiže mali sme rovnaký počet dát pre pohyb vpred, vľavo a vpravo. Po následnom natrénovaní novej množiny dát sme spozorovali, že

robot sa otáča skôr a nepribližuje sa k stene ako v prvom prípade. Ale z dôvodu rovnakých počtov dát pre iný druh pohybu sa robot niekedy viac krát zatočil namiesto možno lepšieho pohybu vpred. Je už na užívateľovi, ktoré dáta sú pre neho lepšie alebo si može natrénovať vlastné dáta. Naše dáta obsahovali:

- v prvom prípade – 500 záznamov, z toho 324 vpred, 84 vpravo a 95 vľavo
- v druhom prípade – 250 záznamov, 84 vpred, 83 vpravo a 83 vľavo

Videozáznamy z obidvoch prípadov testovania prikladáme k prílohám.

2.5 Funkcia logger

Danú funkciu sme implementovali, aby sme poskytli používateľovi spätnú kontrolu výsledných pohybov robota. Jej úlohou bude zaznamenávať do pamäti dôležité údaje pri rôznych experimentoch. Napríklad pri sledovaní objektu pomocou kamery si ukladáme do pamäti obraz podľa ktorého sa robot pohybuje a *Logger* uloží do súboru názov obrazu a výsledný pohyb (Obr 2.15). Používateľ je potom schopný si otestovať správnosť pohybu na základe získaného obrazu a prípadne rýchlejšie porozumieť a vyriešiť vzniknuté problémy.

```
cas: 28.04.2015 11:45:26, obraz: ./Imgs/20150428_114526.jpg, smer pohybu: 1
cas: 28.04.2015 11:45:27, obraz: ./Imgs/20150428_114527.jpg, smer pohybu: 1
cas: 28.04.2015 11:45:27, obraz: ./Imgs/20150428_114527.jpg, smer pohybu: 1
cas: 28.04.2015 11:45:28, obraz: ./Imgs/20150428_114528.jpg, smer pohybu: 4
cas: 28.04.2015 11:45:29, obraz: ./Imgs/20150428_114529.jpg, smer pohybu: 4
cas: 28.04.2015 11:45:30, obraz: ./Imgs/20150428_114530.jpg, smer pohybu: 3
```

Funkcia *Logger* je implementovaná na Gumstix, keďže ten nám poskytuje dostatočný priestor pre ukladanie dát do pamäti.

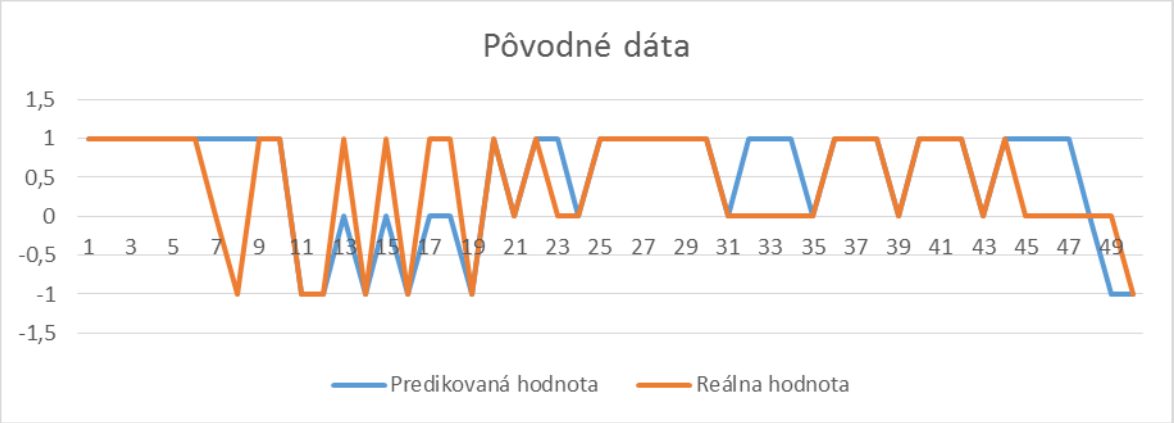
Využívame ju pri experimente sledovania objektu pomocou kamery, ako sme popísali vyššie. Ďalej pri experimente č.6, kde si uchovávame názov uloženého obrazu a názov uložených dát význačných bodov. Taktiež pri experimente č.7, kde si ukladáme reálne namerané hodnoty zo senzorov a výstupnú hodnotu pohybu neurónovej siete.

2.6 Výsledky pre implementované neurónové siete

Na zobrazenie výsledkov správnosti predikovaných výstupných hodnôt z neurónovej siete sme využili dáta zapísané v pamäti pomocou funkcie *Logger*. Zo súboru *20150428-11hod_NeuronNet.txt*, ktorý je priložený aj v prílohách, sme zobrali hodnoty nameraných vzdialeností senzorov a výstupnú správnu hodnotu sme určili my. Určovali sme to spôsobom, že ktorá vzdialenosť je najväčsia tam chceme ísť a porovnávali sme to

s predikovanou hodnotou algoritmu. Test sme realizovali na obidvoch množinách dát aj na pôvodných, aj na vyrovnaných dátach.

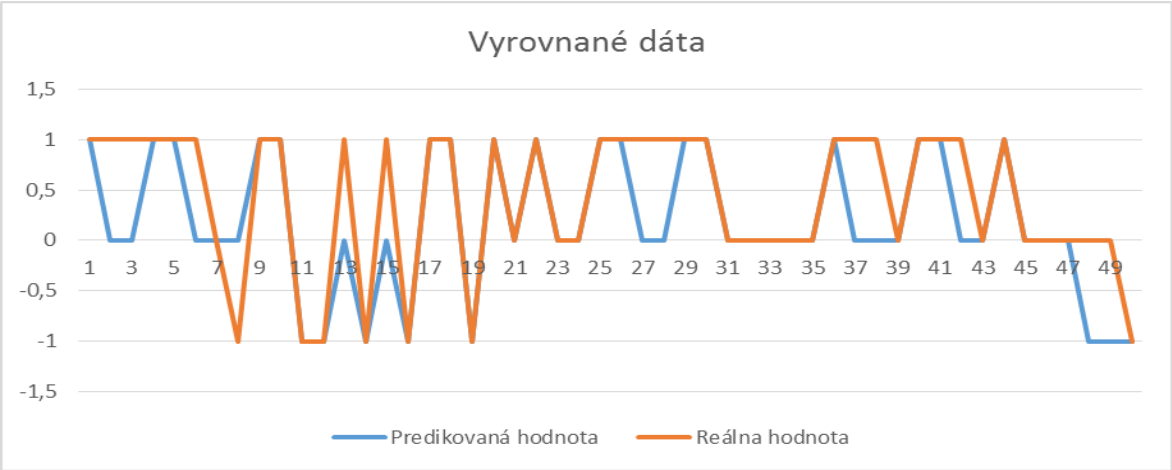
Výsledok z pôvodných dát môžeme vidieť na tomto grafu(Obr. 2. 34):



Obr. 2.34: Graf testovania neurónovej siete na pôvodných dátach. Na x-ovej osi sú jednotlivé dáta, na y-ovej osi je výsledok predikovanej a reálnej hodnoty

Počet nezhodujúcich sa hodnôt je 14 z 50. Avšak až na jeden prípad č.8, ktorý by sme mohli považovať za nesprávny, sa jednalo len o rozdiel v miere reakcie, ktorá bola principiálne správna a ďaleké vzdialenosti, čiže rozhodnutie ktoré učila neurónová sieť a my môžme považovať za akceptovatelné. Takisto ako v reálnom experimente, aj tu nám vyšlo, že neurónová sieť je schopná zabezpečiť nekolízny pohyb v prostredí za daných podmienok.

Rovnako aj pri vyrovnaných dátach(Obr. 2.35), môžme vidieť pár nezhôd, ktoré ale vznikli rovnakým spôsobom ako aj pri pôvodných dátach. A teda aj keď nám vzniklo 13 rozlišných výsledkov, môžeme tvrdiť a aj preukázať videozáznamom, že aj tu je natrénovaná neurónová sieť schopná zabezpečiť nekolízny pohyb v prostredí.



Obr. 2.35: Graf testovania neurónovej siete na vyrovnaných dátach. Na x-ovej osi sú jednotlivé dáta, na y-ovej osi je výsledok predikovanej a reálnej hodnoty

2.7 Ukážka vytvorenia nového správania

Vytvoríme si dva nové súbory *example.c, example.h*.

example.c obsahuje:

```
void example_init(){
      example.name = "example"; //meno správania
      example.priority = EXAMPLE_PRIORITY; // priorita správania
      example.active = RUN_ACTIVE; // defaultná aktivačná premenná
      example.loop = example_loop; // funckia pre beh správania
      example.handler = example_handler; // funkcia na spracovanie
}                                        // správy
void example_setup(){
      E_SET_MOTORS = -1; // premenná uchovávajúci hodnotu pohyb
}               //a posielaná správaniu, ktoré zabezpečuje pohyb
void example_loop(){
           //telo správania
     wait(10);
}
void example_handler(int weight, char* msg){
// spracovanie správy, základ je možné zobrať z už vytvorených
// správaní
}
void e_connect_send_msg(){// funkcia na odoslanie správy
      beh = return_handler("machine_room"); // správa sa posiela
                                      //správaniu machine_room
```

```
    beh.handler(sizeof(example_msg), example_msg);
}
```

example.h obsahuje:

```
#ifndef EXAMPLE_H
#define EXAMPLE_H
#define EXAMPLE_PRIORITY 7

    int E_SET_MOTORS;
    behavior example;

    void example_init();
    void example_setup();
    void example_loop();
    void example_handler(int weight, char* msg);
    void e_connect_send_msg();
#endif
```

Ďalej musíme nastaviť v systéme:

- v súbore *behavior.h*, premennú *NUMBER_OF_BEHAVIORS* zvýšiť o 1
- v súbore *behavior.c*, pridať správanie do poľa správaní:

  ```
  behaviors[7]= example;
  ```

- v *main.h*, pridať *#include "example.h"* a deklaráciu

  void example_init();,

  zvýšiť veľkosť pola *set_robot* o 1, pre testovací režim
- v *main.c*, zavolať inicializačnú funckiu *example_init();*
- pre testovací režim ponúknuť možnosť zapnutia správania na úvodnej obrazovke, pri štarte nastaviť zapnutie, pri prerušení vypnutie

3 Záver

V tejto diplomovej práci sme sa zamerali na vytvorenie základného frameworku mobilného autonómneho robota s počítačovým videním. Cieľom bolo navrhnúť, implementovať a otestovať požadovaný framework na rôznych experimentoch pomocou počítačového videnia a umelej inteligencie.

Na implementáciu robota sme využili dve robotické platformy Sbot a Gumstix. Zabezpečili sme ich komunikáciu pomocou sériového portu. Platforma Gumstix slúži ako riadiaci systém s vyšším výpočtovým výkonom, čo nám umožnilo vykonávať náročné operácie ako je spracovanie obrazu alebo realizácia neurónových sieti. Napriek mnohým náročným hardvérovým a softvérovým výzvam je výsledkom plne funkčný, nezávislý a dobre zdokumentovaný systém.

V predvedených experimentoch sme ukázali rôzne pôsobnosti robota. Predviedli sme jeho funkcionalitu na základných experimentoch ako nekolízny pohyb v prostredí, riadenie robota pomocou klávesnice, udržanie robota v strede koridoru alebo sledovanie čiary a obchádzanie prekážok. Ďalej sme implementovali experimenty zaoberajúce sa spracovaním obrazu získaného z kamery, kde sme dokázali sledovať pomocou kamery navrhnutý objekt alebo zaznamenať si nájdené význačné body z obrazu do pamäti. V poslednom experimente sme ukázali využitie neurónových sieti, kde sme natrénovali robota na nekolízny pohyb v prostredí a pomocou natrénovaných dát realizovali jeho výsledný nekolízny pohyb v prostredí.

Prínos práce

Hlavným prínosom práce je navrhnutý a zdokumentovaný framework v štýle Behavior-Based Robotics, ktorý je vysoko modulárny a preto ľahko rozšíriteľný a tým prispôsobený pre vytvorenie nových sofistikovaných riešení pre robota. Implementáciou rôznych riešení sme ukázali rôzne pôsobnosti robota, ktorý môže slúžiť aj ako učebná pomôcka pri výučbe robotiky alebo môže byť využívaný pri realizácií nových experimentov.

Námety nových experimentov

- prepojenie Gumstix s PC cez wifi a zobrazenie obrazu z Caspa i kompletného stavu senzorov, motorov a jednotlivých správaní pomocou aplikácie na PC

- rozšírenie pohybu a navigácie o odometriu a tým zabezpečiť presné pohyby otáčania kolies
- rozšíriť prácu s význačnými bodmi na zmapovanie prostredia a následnú lokalizáciu v prostredí
- doplniť robota novými senzormi, napr. kompasom a riadnie robota rozšíriť o jeho funkcionalitu

Zoznam použitej literatúry

[1] RUSSELL S., NORVIG P.: *Artificial Intelligence: A Modern Approach, 2nd Edition.* Prentice Hall 2002

[2] BEKEY A.: *Autonomous Robots: From Biological Inspiration to Implementation and Control.* A Bradford Book, 2005

[3] BROOKS R.: *A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation.* 1986

[4] ARKIN R.: *AuRA: Principles ans Practice in Review. Journal of Experimental and Theoretical Artificial Intelligence.* 1997

[5] MULLER J., PISCHEL M.: *The Agent Architecture INTERRAP: Concept and Application.* 1993

[6] TURING A.: *Computing machinery and intelligence.* 1950

[7] RUMELHART D., HINTON G., WILLIAMS R.: *Learning Representations by Back-Propagating Errors.* 1986

[8] SE S., LOWE D., LITTLE J.: *Vision-based Mobile Robot Localization And Mapping using Scale-Invariant Features. Proceedings of the 2001 IEEE International Conference on Robotics & Automation.* Soul, 2001

[9] FTÁČNIK M.: *Základy spracovania obrazu.* Katedra Aplikovanej Informatiky, Fakulta Matematiky, Fyziky a Informatiky, Univerzita Komenského, Bratislava

[10] BAY H., ESS A., TUYTELAARS T., VAN GOOL L.: *Speeded-Up Robust Features (SURF).* Zurich 2006

[11] LÚČNY A.: *Multiagentové systémy*. Katedra Aplikovanej Informatiky, Fakulta Matematiky, Fyziky a Informatiky, Univerzita Komenského, Bratislava

[12] MAISONNIER B.: *Aldebaran Robotics – SAS.* Paris. Online: <https://www.aldebaran.com/>

[13] GUSTAFÍK D.: *Aplikácie mobilného robota Sbot2.* Bakalárska práca, Fakulta Elektrotechniky a Informatiky, Slovenská Technická Univerzita, Bratislava 2011

[14] BSS138: N-Channel Logic Level Enhancement Mode Field Effect Transistor. Online: <http://www.adafruit.com/datasheets/BSS138.pdf>

[15] 4-Channel I2C-Safe Bi-Directional Logic Level Converter. Online: <https://www.adafruit.com/products/757>
[16] GUSTAFÍK D., KRASŇANSKÝ P.: *Robot SBot v2.0.* Bratislava 2008

Prílohy

Príloha č.1: CD so zdrojovými kódmi a videami z experimentov

Príloha č.2: Link na dostupné zdrojové kódy, ktoré sú open-source:

http://dai.fmph.uniba.sk/projects/sbot/