UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Pokročilé experimenty s robotickou stavebnicou EV3

Bratislava 2015 **Ondrej Slovák**

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Pokročilé experimenty s robotickou stavebnicou EV3

Diplomová práca

Študijný program: Aplikovaná informatika
Študijný odbor: 2511 Aplikovaná informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Školiteľ: Mgr. Pavel Petrovič, PhD.

Bratislava 2015 **Ondrej Slovák**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Ondrej Slovák
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
Sekundárny jazyk: anglický

Názov: Pokročilé experimenty s robotickou stavebnicou EV3
Advanced Experiments with Robotic Construction Set EV3

Cieľ: Robotická stavebnica LEGO Mindstorms Education EV3, uvedená v roku zadania tejto diplomovej práce, je učebná pomôcka určená pre deti od 10 rokov. Jej potenciál ďaleko presahuje úroveň základných i stredných škôl a umožňuje štúdium i výskum pokročilých programovacích jazykov, navigačných, lokalizačných, riadiacich a plánovacích algoritmov a stratégií, komunikačných protokolov, senzorov, online spracovania a analýzy dát i algoritmov strojového učenia. Cieľom práce je prehľadne spracovať pokročilé možnosti EV3 a demonštrovať ich použitie na sade experimentov z uvedených oblastí.

Literatúra: G.A.Bakey: Autnomous Robots, MIT Press, 2005.

Kľúčové slová: mobilné autonómne roboty, LEGO Mindstorms EV3, strojové učenie

Vedúci: Mgr. Pavel Petrovič, PhD.
Katedra: FMFI.KAI - Katedra aplikovanej informatiky
Vedúci katedry: prof. Ing. Igor Farkaš, PhD.

Dátum zadania: 04.12.2013

Dátum schválenia: 10.12.2013

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

student

vedúci práce

Čestné vyhlásenie

Čestne vyhlasujem, že som diplomovú prácu "Pokročilé experimenty s robotickou stavebnicou EV3" vypracoval samostatne s použitím uvedenej literatúry a zdrojov dostupných na internete.

V Bratislave dňa 3.5.2015

_____

Ondrej Slovák

Poďakovanie

Chcel by som sa poďakovať vedúcemu diplomovej práce Mgr. Pavlovi Petrovičovi, PhD. za cenné rady a usmernenie pri vypracovaní tejto práce.

Abstrakt

Cieľom diplomovej práce je vytvoriť sériu experimentov na robotickej stavebnici LEGO MINDSTORMS EV3 využitím systému Lejos založenom na programovacom jazyku Java. Účelom práce bolo dôkladnejšie sa zoznámiť s touto novou platformou, sprístupniť ju pre potreby výuky a výskumu a všetky naše zistenia dôkladne zdokumentovať. Podarilo sa nám navrhnúť, spracovať, naimplementovať a otestovať šesť experimentov, ktoré pokývajú rôzne oblasti robotiky a umelej inteligencie. Tieto experimenty sme naviac následne upravili na cvičenia pre výuku robotiky. Výsledkom práce sú zdrojové súbory pre cvičenia ako aj experimenty a prehľadne sformulované zadania cvičení. V rámci cvičení predmetu Algoritmy pre AI robotiku v letnom semestri 2014/2015 sme ich odskúšali na skupine študentov.

Kľúčové slová: mobilné autonómne roboty, LEGO Mindstorms EV3, strojové učenie

Abstract

Goal of this thesis is to create series of experiments for robotic kit LEGO MINDSTORMS EV3 using the Lejos system based on the programming language Java. The purpose of this work is to explore this new platform in more depth, make it available to the needs of teaching and research and thoroughly document all our findings. We did design, create, implement and test six experiments, which cover wide variety of fields from Robotics and Artificial Intelligence. Furthermore, these experiments were adjusted for exercises for teaching robotics. Outcome of our work is source files for experiment and exercises and formulated exercises assignments. We have evaluated the exercises in a course Algorithms for AI Robotics on a group of students in the summer semester of 2014/2015.

Keywords: autonomous mobile robots, LEGO Mindstorms EV3, machine learning

Úvod

Robotická stavebnica LEGO MIDSTORMS EV3 je učebná pomôcka vydaná v auguste roku zadania diplomovej práce. Bola to jedna z horúcich noviniek a preto sme sa rozhodli ju použiť v našej práci. Spoločnosť LEGO v tej dobe už mala dve úspešné robotické stavebnice, takže potenciál mala určite veľký. Robotické stavebnice LEGO MINDSTORMS sú využívané na všetkých stupňoch škôl – od základných škôl až po univerzity, keďže na jednej strane komunita nadšencov a na druhej strane samotná firma LEGO formou SDK zverejnili podrobnú technickú dokumentáciu hardvérových aj softvérových komponentov. To viedlo k vzniku množstva vývojových softvérových platforiem a jazykov, medzi najznámejšie patria BricX/NXC [14] alebo RobotC, okolo ktorého je vybudovaný celý vzdelávací program CMU Robotics Academy [16]. V priebehu 17 rokov od uvedenia prvej verzie vzniklo množstvo súťaží, medzi nimi aj robotická súťaž pre mládež s najväčším počtom účastníkov (na celom svete okolo 200-tisíc) FIRST LEGO League [17]. Stavebnice sa úspešne využívajú v juniorskej kategórii celosvetovej súťaže RoboCup, ktorú iniciovali delegáti na Joint Conference on Artificial Intelligence koncom 90-tych rokov a ktorá sa zameriava na výskum algoritmov a metód umelej inteligencie. Medzi silné stránky stavebníc patrí univerzálnosť a vysoká modularita, je možné z nich poskladať takmer ľubovoľný model. Študenti technických škôl využívajú platformu ako pomerne výkonnú riadiacu jednotku, ku ktorej si navrhujú a vyrábajú vlastné senzory a iné elektronické zariadenia. Fakt, že najnovšia generácia stavebnice ešte len vyšla na trh ale skrýval aj výzvu. Pre stavebnicu neexistovali návody na vytváranie experimentov a komunita ochotná pomôcť nebola taká rozsiahla.

Cieľom práce bolo vytvoriť experimenty na zoznámenie sa so stavebnicou a so základnými algoritmami. Ďalej by sme chceli vytvoriť rôzne experimenty, na ktorých budeme ukazovať potenciál stavebnice EV3. Jej možnosti komunikácie, podporovanie rôznych pokročilých algoritmov, pripojenie z rôznych zariadení, atď. Nakoniec by sme chceli vytvoriť experiment demonštrujúci pravdepodobnostný algoritmus Monte Carlo lokalizácie. Pri všetkých týchto úlohách využijeme programovací jazyk Java a systém Lejos, ktorý najlepšie vyhovuje potrebám pokročilých experimentov, keďže (na rozdiel od predchádzajúcich verzií robotických stavebníc) na tejto platforme beží plnohodnotná verzia jazyka Java distribuovaná priamo spoločnosťou Oracle, systém obsahuje asi

najbohatšiu knižnicu hotových algoritmov (Lejos API) a umožňuje hladkú spoluprácu javovských aplikácií bežiacich na PC a programov napísaných v Lejose bežiacich na EV3.

Z realizovaných experimentov by sme chceli následne vytvoriť cvičenia pre výuku robotiky. Ako nepovinný cieľ sme si zvolili otestovanie týchto cvičení so študentami.

V prvej kapitole si rozoberieme povahu úloh robotiky a dôsledky z nej vyplývajúce. Porovnáme si rôzne softvérové ako aj hardvérové pomôcky na výuku robotiky.

V druhej kapitole si rozoberieme každý experiment zvlášť. Uvedieme si motiváciu ku každému experimentu a návrh cvičenia.

V tretej kapitole si ukážeme vzorové riešenia cvičení a v dvoch prípadoch si ukážeme aj riešenie študentov.

1. Východiská

1.1. Robotika a umelá inteligencia

Umelá intilegencia je všeobecný termín, ktorý znamená použitie počítača na modelovanie a/alebo replikovanie inteligentného správania. To zahŕňa schopnosť učiť sa, schopnosť uvažovať, schopnosť dorozumievať sa a schopnosť tvoriť originálne nápady.

Žiadna oblasť a ani robotika sa zatiaľ nepriblížila takejto úrovni umelej inteligencie. Dnešná robotika dokáže replikovať niektoré špecifické prvky inletigentného správania ako schopnosť riešiť problémy v obmedzenom prostredí, učiť sa v limitovanej kapacite, atď. Neexistuje komplexný robotický inteligentný systém, ktorý by zvládal všetky vyššie spomínané schopnosti. Z tohoto dôvodu sa vyvýjajú robotické systémy so špeciálnym zameraním a povahou úloh.

1.1.1. Povaha úloh robotiky

Povaha úloh sa mení v závistlosti od zamerania daného problému, prostredia, v ktorom sa bude vykonávať. Poďme sa pozrieť na niektoré z týchto faktorov.

1.1.1.1. Prostredie

Množstvo prestredí, v ktorých sa odohrávajú úlohy pre robotiku je očividne obrovské. Dokážeme však identifikovať malé množstvo dimenzií, do ktorých vieme tieto prostredia kategorizovať:

- Plne alebo čiastočne pozorovateľné – ak má robot pomocou senzorou prístup ku celému prostrediu, toto prostredie je plne pozorovateľné. Prostredie môže byť čiastočne pozorovateľné z dôvodu šumu alebo nepresnosti senzorov, alebo z dôvodu, že časť prostredia sa nedá senzormi namerať.
- Deterministické alebo stochastické – ak ďalší stav prostredia je daný súčasným stavom a akciou tak hovoríme, že prostredie je deterministické. V opačnom prípade je stochastické.
- Statické alebo dynamické – Ak sa prostredie mení pokiaľ robot spracováva ostatné údaje hovoríme, že prostredie je dynamické. V opačnom prípade je statické.
- Či sa v prostredí nachádza jeden alebo viac robotov

1.1.1.2. Komunikácia

V budúcnosti budú všetky systémy prepojené a budú spolu komunikovať. Preto sú kladené nároky aj na robotiku, aby roboty boli schopné komunikovať. Roboty musia byť schopné komunikovať s prostredím, komunikovať s ostatnými robotmi a robiť rozhodnutia na základe získaných informácií. Bez komunikácie by dva roboty mohli vykonávať konfliktné akcie.

Uveďme si príklad z robotického futbalu. Dva roboty vidia loptu, bez komunikácie by sa mohli obaja rozhodnúť ísť ku lopte, čo by bolo nežiadúce správanie. S pomocou komunikácie sa roboty dohodnú, ktorý z nich sa bude správať ako útočník a pôjde za loptou, a ktorý z nich sa bude správať ako obranca a zaujme defenzívne postavenie.

1.1.1.3. Potreba adaptácie, učenia

Vstupy z prostredia by nemali byť použité len na vykonanie aktuálnej akcie, ale aj na zdokonalenie správania do budúcna. Učenie sa môže pohybovať od jednoduchého zapamätania si akcií až po vytváranie celých vedeckých teórií.

Typ spätnej väzby je zvyčajne najpodstatnejší faktor pri určovaní povahy problému, ktorému robot čelí. Odbor strojového učenia rozlišuje tri typy:

- Učenie s učiteľom – zahrňuje naučenie funkcie z trénovacích príkladov vstupov a výstupov Po skončení učenia si robot zo vstupov pomocou natrénovanej funkcie vypočíta akú akciu má vykonať.
- Učenie bez učiteľa – je problém hľadania skrytej štruktúry v neoznačených dátach.
- Učenie odmeňovaním – je učenie pomocou interakcie s prostredím. Robot sa učí z následkov akcií. V podstate je to metóda pokus-omyl, avšak zohľadňuje sa aj história vykonaných akcií, nielen posledná akcia, ktorá viedla k odmene alebo trestu.

1.1.2. Dôsledky

Z veľmi rôznorodej povahy úloh pre robotiku vyplývajú dôsledky a nároky na hardverovú ako aj softverovú časť robotov. Roboty musia byť schopné operovať v rôznych prostrediach, komunikovať s prostredím a aktormi v ňom, musia byť schopné si udržiavať znalosti o prostredí a svojich úlohách. Tieto podnety kladú nároky na spôsob uchovávania poznatkov ako aj samotnú architektúru robotov.

1.1.2.1. Reprezentácia poznatkov

Základnou funciou reprezentácie poznatkov je zachytiť podstatné prvky problému a poskytnúť tieto informácie procedúre, ktorá rieši daný problém. Robot si musí uchovávať svoje poznatky o prostredí, v ktorom sa nachádza, o jeho prekážkach, o jeho stave, prípadne ďalších robotoch.

1.1.2.1.1. Distribuovaná

Udržiavanie si všetkých poznatkov centralizovane na jednom mieste je neefektívne a redundantné. Ďaleko efektívnejšie je si rozdistribuovať si poznatky na miesta, kde sa budú reálne používať. Napríklad ak máme časť programu, ktorá zabezpečuje aby robot nenarazil do prekážky, tak v tejto časti si budeme udržiavať poznatky o najbližšej prekážke. Ostatné časti, ktoré tieto poznatky nepotrebujú, k nim nebudú mať ani prístup. Takéto distrubovanie poznatkov má vplyv aj na riadiacu architektúru robotického systému.

1.1.2.1.2. Musí počítať s neurčitosťou

Robot sa nie vždy môže spoľahnúť na presnosť a aktuálnosť svojich poznatkov. Jedným z dôvodov je dynamicky sa meniace prostredie. Aktuálny stav sa môže líšiť od posledného nameraného stavu, ktorý robot má. Ďalším dôvodom je nepresnosť senzorov. Každý senzor má odchylku, v ktorej sa pohybujú namerané hodnoty. Robotický systém musí s týmito neurčitosťami počítač pri svojich výpočtoch.

1.1.2.2. Riadiaca architektúra

Pod pojmom riadiaca architektúra máme väčšinou na mysli softvér alebo hardvér framework, ktorý zabezbečuje ovládanie robota. Nízkoúrovňový softvér, ktorý otáča motormi, nepredstavuje architektúru sám o sebe. Až vývoj modulov a komunikácie medzi nimi definuje riadiacu architektúru. V praxi to znamená, ako robot funguje. Riadiaca architektúra môže byť popísaná ako väzby medzi troma základnými primitívami robotiky: Vnímaj, Plánuj, Konaj.

Robotické systémy sú veľmi zložité a majú tendenciu byť ťažko naprogramovateľné. Musia spájať viacero senzorov ako aj systémov dokopy. Riadiace architektúry pre takéto systémy majú preto tendenciu byť špecifické na jeden typ a nie sú vhodne aplikovateľné na širšie spektrum problémov.

Z historického hľadiska rozlišujeme tieto typy riadiacej architektúry:

- Hierarchická architektúra – je to najstaršia architektúra. Robot vníma prostredie, naplánuje akciu a koná. V každom kroku robot explicitne plánuje ďalšiu akciu. Ďalšou odlišujúcou vlastnosťou hierarchickej architektúry je že všetky vstupné dáta sú zhromaždované v jednom globálnom modeli, ktorý sa využíva pri plánovaní. Vytváranie takýchto globálnych modelov je ale náročné, vypočtovo neefektívne a ťažko odladiteľné.
- Reaktívna architektúra – Táto architektúra predpokladá, že vstupom pre akciu je priamo výstup zo senzorov. Robot obsahuje viacero inštancií Vnem-Akcia dvojíc. Tieto dvojice sú paralelné procesy a nazývajú sa správania. Zoberú lokálne vstupné dáta zo senzorov a vypočítajú najlepšiu akciu nezávisle od ostatných procesov.
- Hybridná architektúra – Pod hybridnou architektúrou robot najskôr plánuje ako najlepšie rozložiť problém na podproblémy a potom vyberá najvhodnejšie správanie na vykonanie daného podproblému. Zároveň si zachováva schopnosť okamžite reagovať na podnety z prostredia, ktoré majú vysokú prioritu.

V dnešnej dobe sú nároky na architektúru väčšie. Musí byť robusná, modulárna, musí byť aplikovateľná na veľké spektrum problémov, atď. Veľmi rozšírenou je architektúra v štýle Behavior-Based Robotics, ktorá je zovšeobecnením reaktívnej architektúry, keďže pripúšťa aj deliberatívne rozhodovacie procesy.

1.1.2.2.1. Behaviour-Based Robotics

Existuje veľké množstvo architektúr, ktoré sú vytvorené na základe Behaviou-Based paradigmy. Tieto architektúry majú spoločný dôraz na tesné spojenie medzi snímaním a akciami a dekompozíciu poblému na jednotky správaní. Architektúry sa líšia rozkladom na správanie, použitými metódami na koordináciu a ďalšími faktormi.[5]

Napriek tomu, že sa využíva vo viacerých odvetviach a v rôznych podobách, niektoré špecifické vlastnosti majú spoločné.

- Základom architektúry sú jednoduché správania ako vyhýbanie sa prekážkam alebo nabíjanie batérie. Tieto správania musia fungovať aj v neštrukturovanom prostredí. Komplexnejšie správania sa pridávajú v neskoršej fáze.

- Viaceré správania sa vykonávajú simultánne a výsledná akcia sa buď vyberie ako výsledok jedného zo správaní alebo sa zo všetkých navrhovaných akcií spraví jedna zložená
- Behavior-Based Robotics architektúra sa väčšinou používa na autonómnych robotoch, teda na robotoch, ktoré sa vedia pohybovať samé bez toho, aby ich človek na diaľku riadil.

1.2. Platformy pre výuku robotiky

Experimenty, ktoré sme vrámci práce vytvorili sa budú používať na výuku robotiky. V tejto kapitole si porovnáme rôzne platformy, ktoré sa využívajú na výuku robotiky a porovnáme ich s platformou, ktorú sme si zvolili.

Rozdeľme si ich na dve hlavné kategórie: hardvér a softvér.

1.2.1. Hardware

1.2.1.1. Parallax Inc – Boe-Bot

Boe-Bot je skratka pre Board of Education robot[6]. Je to robotická stavebnica od spoločnosti Parallax Inc pre základné, stredné aj vysoké školy. Skladá z hlavnej dosky (ktorá sa volá Board of Education) a dosky pre integrované obvody, jednočipového procesora BASIC Stamp, dvoch servo motorov a hliníkového podvozku. Študenti si môžu používať diely z iných stavebníc (Lego, Erector) na vybudovanie vlastných projektov.

Robot môže byť naprogramovaný na sledovanie čiare, nájdenie cesty z bľudiska, sledovanie svetla alebo na komunikáciu s iným robotom. Boe-Bot sa dá použiť s Microsoft Robotics Developer Studio softvérom. Programuje sa pomocou jazyka PBASIC.

1.2.1.2. Khepera

Khepera je malý pohybujúci sa robot s dvoma separátne ovládanými motormi[7]. Bol vyvinutý v 1990-tych rokoch a odvtedy sa používa na veľkom množstve univerzít. Tento robot časom prešiel značnými úpravami a vylepšeniami a boli vydané už štyri generácie.

Najvšia generácia tohto robota už obsahuje ARM processor, na ktorom beží Linux, osem infračervených senzorov na detekciu objektov, ďaľšie štyri infračervené sensory používa na prevenciu pádu, päť ultrazvukových senzorov na detekciu objektov na veľkú

vzdialenosť. Ďalej obsahuje farebnú kameru, Wifi, Bluetooth, accelerometer, gyroskop, mikrofón, reproduktor, atď.

1.2.1.3. LEGO MINDSTORMS

LEGO MINDSTORMS je séria robotických stavebníc, ktoré obsahujú hardvér ako aj softvér na vytváranie upravovateľných robotov[8]. Obsahujú inteligentnú kocku, čo je počítač, ktorý kontroluje systém, senzory a motory. Ku dnešnému dňu boli vydané tri série tejto stavebnice: RCX, NXT a EV3.

RCX

Je to prvá generácia LEGO MINDSTORMS série vydaná v roku 1998. Originálna sada obsahovala 8-bitový Hitachi (od 2003 Renesas) H8/300 jednočipový mikropočítač, 32 KB operačnej pamäte RAM, dva motory, dva dotykové senzory, jeden svetelný senzor a viac ako 700 častíc na stavbu robota. Na RCX kocke sa nachádzajú tri vstupné porty pre senzory, tri výstupné porty pre motory a LCD displej, na ktorom sa zobrazuje stav batérie, vstupné a výstupné data z portov, práve bežiaci program, atď.

NXT

Druhá generácia tejto série bola vydaná v júli roku 2006. Kocka obsahovala vylepšený 32 bitový ARM7 processor a 64 KB operačnej pamäte. Takisto ako predchádzajúca verzia obsahovala LCD displej, ktorý bol ale podstatne väčší a bol dodávaný spolu so šturmi tlačidlami na ovládanie navigácie cez používateľské rozhranie. Kocka takisto mala podporu pre Bluetooth a obsahovala aj reproduktor. V základnej verzií by sme ďalej našli tri motory, dotykový senzor, sveteľný senzor a ultrazvukový senzor. Do stavebnice sa dal dokúpiť kompas, gyroskop, accelerometer, atď.

EV3

Najnovšia generácia série LEGO MINDSTORMS, ktorá bola vydaná v auguste 2013. Túto stavebnicu sme používali aj my v našej práci. Obsahuje ARM9 procesor, na ktorom beží operačný systém Linux, a 64 MB operačnej pamäte. Oproti staršej verzií kocky ešte obsahuje USB a Micro SD port. Stavebnica, určená na výuku obsahuje dva veľké motory, jeden stredný motor, dva dotykové senzory, farebný senzor, gyroskopický senzor a ultrazvukový senzor. Všetky motory a senzory zo staršej verzie NXT môžu byť použité aj so stavebnicou EV3.

1.2.2. Software

1.2.2.1. ROS (Robot Operating System)

ROS je flexibilný framework na písanie programov pre robotov[9]. Je to súbor nástrojov, knižníc a konvencií, ktoré slúžia na zjednodušenie vytvárania komplexných a robustných robotických správaní pre široké spektrum robotických platforiem.

ROS bol vytvorený ako open source, do ktorého prispievajú ľudia po celom svete. Dnes obsahuje desaťtisíce používateľou po celom svete, ktorý pracujú na širokej škále projektov od domácich až po veľké priemyselné automatické systémy.

1.2.2.2. Microsoft Robotics Developer Studio

Microsoft Robotics Developer Studio(MRDS)[10] je prostredie pre operačný systém Windows na ovládanie a simulácie robotov. Je určené pre akademické ako aj komerčné využitie a podporuje veľké množstvo robotického hardvéru.

Primárny programovací jazyk je C#. Obsahuje rôzne funkcie ako nástroj na vývoj pomocou „obrázkového“ programovacieho jazyka, Microsoft Visual Programming Language, čo je programovací jazyk na vytváranie a debugovanie robotických aplikácií, 3D simulácie, jednoduchý prístup ku senzorom a aktuátorom robota.

MRDS obsahuje podporu na pridávanie balíčkov s ďaľšími funkciami. Napríklad balíček s podporou na simuláciu robotických futbalových zápasov alebo sumo zápasov, simulátor na riešenie bludiska a ďaľšie. Vývoj MRDS bol ukončený v septembri 2014[11].

1.2.2.3. Webots

Webots[12] je vývojarské prostredie, ktoré sa používa na modelovanie, programovanie a simulovanie robotov. Používateľ si môže navrhnúť komplexné robotické štruktúry s jedným alebo viacerými robotmi. Na výber má širokú škálu senzorov a aktuátorov. Správanie robotov sa dá otestovať v realistickom prostredí. Webots sa používa na viac ako 1188 univerzitách a výskumných zariadeniech po celom svete.

1.2.2.4. LEGO MINDSTORMS

Táto stavebnica je jedna z najrozšírenejších robotických výukových pomôcok. K základnej verzii stavebnice existuje softvér na vytváranie programov, ktorý beží na

operačných systémoch Windows a Mac OS a je voľne stiahnuteľný z webstránky spoločnosti LEGO. Softvér je založený na LabView (Laboratory Virtual Instrument Engineering Workbench) od spoločnosti National Instruments. LabView je platforma a vývojárske prostredie pre "obrázkový" programovací jazyk.

Vďaka rozšírenosti tejto stavebnice existuje veľké množstvo iných knižníc a vývojarských prostredí na vytváranie programov. Spomeňme len niektoré z nich.

1.2.2.4.1. NXT-G

NXT-G, resp. EV3-G je grafické programovacie prostredie, ktoré je obsiahnuté v základnej verzií stavebnice. Pri správnom využití sa z neho stáva plnohodnotné programovacie prostredie schopné aj zložitých programov. Dajú sa v ňom naprogramovať paralelné vlákna, autonómne ovládanie ako aj ďiaľkové ovládanie. Toto prostredie obsahuje takisto tutoriály, vlastný grafický a zvukový dizajn ako aj širokú komunitu ľudí, ktorí radi pomôžu.

1.2.2.4.2. RobotC

Je vývojarské prostredie založené na jazyku C[13]. Narozdiel od NXT-G používa textové príkazy. Má možnosť debugovania, je určené na jednoduché ako aj zložitejšie problémy. Nato aby bežal vyžaduje vlastný firmware.

1.2.2.4.3. NXC

NXC (Not eXactly C)/BricX je open-source programovací jazyk podobný programovaciemu jazyku C[14]. Je postavený na NBC kompilátore. Je to jeden z najrozšírenejších programovacích jazykov pre NXT. Využíva štandardný firmware a teda poskytuje rovnakú množinu funkcií ako ikonografický programovací jazyk NXT-G.

1.2.2.4.4. Ostatné

Existuje veľké množstvo ďaľších programovacích jazykov a vývojárskych prostredí ako napríklad Bricx, RoboLab, RoboMind, ruby-nxt, PyNXC, NXT-Python, a ďaľšie.

1.2.2.4.5. leJOS

leJOS je open-source náhrada oficiálneho softvéru pre LEGO MINDSTORMS programovateľné kocky[15]. Obsahuje Java Virtual Machine čo dovoľuje programovanie

robotov v programovacom jazyku Java. leJOS podporuje všetky generácie stavebnice LEGO MINDSTORMS.

Tento softvér sme použili aj my v našej práci. Vybrali sme si ho práve z dôvodu možnosti programovania v jazyku Java, výborne spracovanej dokumentácií, veľkej podpore rôznych hardvérových ako aj softvérových funkcií.

leJOS sa na platforme EV3 používa zo špeciálne nakonfigurovanej SD karty, z ktorej sa pri zapínaní robota načíta softvér. Programy sa kompilujú do súborov s .jar príponou. Tieto súbory sa prenesú na robota a pomocou Java Virtual Machine sa vykonávajú.

leJOS Java API obsahuje podporu na prácu so senzormi, motormi, LCD displejom, podporu na komunikáciu s rôznymi zariadeniami pomocou Wifi alebo Bluetooth, podporu na pokročilé lokalizačné a mapovacie algoritmi, a ešte oveľa viac.

2. Návrhy Experimentov

V tejto kapitole sa venujeme konkrétne experimentom. Ku každému experimentu uvádzame motiváciu pre experiment, teda bližšie špecifikujeme dôvody, prečo je experiment vhodný vo výuke algoritmov využívajúcich umelú inteligenciu v robotike. Nasledujúci hárok experimentu obsahuje popis experimentu a zadanie cvičenií pre študentov. Na záver uvádzame naše vzorové riešenie. V experimentoch LineFollower a BumperCar uvádzame aj príklad študentovho výstupu.

Tieto experimenty pokrývajú základné zoznámenie sa so stavebnicou a knižnicou leJOS-u, vyťahovanie namerených údajov zo senzorov, komunikáciu robota s počítačom a mobilným telefónom, architektúru Beahaviour-Based Robotics, pravdepodobnostné algoritmi, konkrétne algoritmus Monte Carlo lokalizácie.

2.1. Experiment LineFollower

2.1.1. Motivácia experimentu

Tento experiment je jednou zo fundamentálnych robotických úloh. Veľmi často sa používa ako základ pri výučbe. My sme si ho vybrali aj z dôvodu ukážky práce s farebným senzorom a aby sa študenti zoznámili so stavenicou a softvérom, v ktorom budeme vytvárať aj ostatné experimenty.

2.1.2. Hárok experimentu

Názov: LineFollower

Téma: Základný experiment

Cieľ': Zoznámiť sa so spôsobmi vyťahovania údajov zo senzorov

Úloha: Naprogramovať robota, ktorý sleduje čiernu čiaru vyznačenú na svetlej podlahe

Konceptuálny popis experimentu:

Robot by sa mal pohybovať po čiernej čiare vyznačenej na zemi. Idea pohybu robota po čiare bude nasledovná. Robot začína svoj pohyb na čiare. V danom okamihu má robot spustený iba jeden motor. Pokiaľ je robot na čiare, má stále spustený iba jeden

motor. Keď robot vyjde z čiary čo zistí pomocou farebného senzora, zastaví motor, ktorý bežal doteraz a spustí druhý motor. Takýmto striedaním motorov sa robot pohybuje po čiare.

Usporiadanie experimentu:

Aréna: miestnosť s bielou podlahou a čiernou čiarou vyznačujúcou dráhu robota

Robot: pohybujúci sa robot s dvoma motormi a farebným senzorom, ktorý sníma farbu podlahy pred robotom. Ľavý motor má robot zapojený do portu B, pravý motor je zapojený do portu C a farebný senzor je zapojený do portu 3.

Algoritmus:

Na implementáciu tohto experimentu sme použili dve vnorené triedy. Jedna hlavná trieda zabezpečuje pohyb robota. Druhá trieda neustále zozbierava vzorky dát z farebného senzora.

Trieda, ktorá zbiera vzorky zo senzora takisto zabezpečuje prepočítavanie dát zo senzora a ich škálovanie na interval od 0 po 1.

Trieda zabezpečujúca pohyb robota najskôr inicializuje motory robota a potom podľa správania popísaného vyššie udržuje robota na čiare.

Implementácia:

Hlavná trieda LineFollower obsahuje dve vnorené triedy ColorSensor a autoAdjustFilter.

Trieda ColorSensor obsahuje globálnu premennú colorSensor, ktorá zabezpečuje prístup ku farebnému senzoru a verejnú globálnu premennú value, v ktorej budeme uchovávať hodnotu nameranú pomocou senzora. ColorSensor extenduje triedu Thread a obsahuje metódu run(), v ktorej sa vo while cykle stále updatuje hodnota nameraná pomocou farebného senzora. Na zosnímané vzorky dát zo senzora sa používa filter autoAdjustFilter.

Trieda autoAdjustFilter vytiahne dáta zo senzora a naškáluje ich na interval od 0 po 1. Takisto sa zapamätáva maximálna a minimálna hodnota nameraná senzorom. Tieto údaje slúžia na lepšie škálovanie.

Hlavná trieda LineFollower potom obsahuje dve globálne premenné typu RegulatedMotor zabezpečujúce prístup ku motorom robota a premennú typu ColorSensor, ktorá nám bude dodávať prefiltrované vzorky z farebného senzora. Hlavná metóda main() obsahuje inicializáciu nastavení motora a zabezpečovanie pohybu robota po čiare. Doplnenie tejto funkcionality bude vašou úlohou.

Postup:

1. Doplniť do triedy LineFollower do metódy main() pohyb robota po čiare.
2. BONUS: Pozmeniť inicializáciu nastavení motorov na získanie plynulejšieho pohybu.

Poznámka: ak budete využívať cyklus, je dobré do neho doplniť nasledujúci riadok, aby vedel robot zastaviť vykonávanie daného cvičenia.
if(Button.ESCAPE.isDown()) System.exit(0);

2.2. Experiment BumperCar

2.2.1. Motivácia experimentu

Takisto ako experiment LineFollower je tento experiment je jedna zo základných robotických úloh. My sme si ho vybrali hlavne ako jednoduchý príklad na ukážku Behaviour-Based Robotics architektúry. Veľmi ľahko a pekne sa na tomto experimente dajú vysvetliť princípy tejto architektúry.

2.2.2. Hárok experimentu

Názov: BumperCar

Téma: Základný experiment, Behaviour Based Robotics

Cieľ: Zoznámiť sa s architektúrami riadenia robotov založených na správaniach (Behavior-Based Robots)

Teoretické pozadie:

Beahviour-Based Robotics je architektúra, ktorá zabezpečuje riadenie robota pomocou množiny pomerne nezávislých paralelne vykonávaných správaní. Každé správanie obsahuje tri základne metódy:

1. metódu na prevzatie kontroly - táto metóda testuje stav prostredia a skúma či nenastala situácia, v ktorej sa toto správanie má aktivovať
2. metóda na potlačenie vykonávania správania - používa sa v situácií kedy si správanie s vyššou prioritou vyžiadalo kontrolu nad robotom
3. metóda vykonávajúca akcie správania robota - aké akcie má robot vykonávať v danom správaní v prípade jeho aktivovania

Jedno zo správaní je základné a toto správanie sa používa na ovládanie robota v situáciách, ktoré nezodpovedajú žiadnej z podmienok aktivácie ostatných správaní.

Úloha: Naprogramovať robota prieskumníka, ktorý náhodne prechádza prostredím a vyhýba sa stenám a prekážkam

Konceptuálny popis experimentu:

Robot sa pohybuje smerom dopredu. Ak je prekážka alebo stena príliš blízko, robot zastaví svoj pohyb a náhodne sa otočí doprava alebo doľava. Následne pokračuje vo svojom pohybe dopredu.

Usporiadanie experimentu:

Aréna: miestnosť s prekážkami

Robot: Pohybujúci sa robot s dvoma motormi a ultrazvukovým senzorom. Ľavý motor má robot zapojený do portu B, pravý motor je zapojený do portu C a ultrazvukový senzor je zapojený do portu 4.

Algoritmus:

Na implementáciu tohoto experimentu sa využíva architektúra v štýle Behavior Based Robotics. Robot obsahuje dve zadefinované správania: DriveForward a DetectWall.

Prvé základné správanie zabezpečuje pohyb robota vpred je DriveForward. Toto správanie sa spúšťa automaticky a obsahuje spúštanie oboch motorov.

Druhé správanie DetectWall zabezpečuje vyhýbanie sa prekážkam. Ultrazvukový senzor neustále meria vzdialenosť od najbližšej prekážky pred robotom. Ak je táto vzdialenosť menšia ako konštanta, toto správanie preberie kontrolu nad robotom. Robot sa zastaví a náhodne si vyberie smer kam sa natočí.

Po natočení preberie kontrolu prvé správanie zabezpečujúce pohyb vpred.

Implementácia:

Hlavná trieda EV3BumperCar obsahuje vnorenú triedu DriveForward.

Trieda DriveForward extenduje Behaviour a popisuje správanie sa robota v určitej situácií. Toto správanie je základné a chceme aby sa spúšťalo vždy keď nenastane špeciálna situácia. Metóda takeControl() preto vždy vracia true. Metóda suppress() obsahuje nastavenie globálnej premennej _suppressed na true. Táto premenná indikuje, že iné správanie si vyžiadalo kontrolu nad robotom. Metóda action() obsahuje kontrolu na to či premenná _suppressed nie je nastavená na true, ak nie robot ide vpred.

Hlavná trieda EV3BumperCar obsahuje inicializáciu parametrov robota, načítavanie všetkých podporovaných správaní robota a spúšťanie funkcionality zabezpečujúcej Behaviour-Based architektúru.

Postup:

1. Vytvorte triedu na zabezpečovanie vzoriek z ultrazvukového senzora a poskytovanie týchto vzoriek ostatným triedam.
    a. Vytvorte triedu, ktorá bude extendovať Thread, bude obsahovať globálnu premennú, ktorá bude uchovávať hodnotu nameranú senzorom, a bude obsahovať metódu run(), v ktorej bude inicializovať túto premennú.
    b. Pridajte globálnu premennú do triedy EV3BumperCar typu novo vytvorenej triedy

c. Pridajte inicializáciu premennej senzora do initializeRobotParameters() metódy v triede EV3BumperCar.

2. Vytvorte nové správanie, ktoré bude zabezpečovať vyhýbanie sa prekážkam
   a. Vytvorte triedu, ktorá bude extendovať Behaviour.
   b. V metóde takeControl() skontrolujte vzdialenosť robota od prekážky a ak bude príliš malá vráťte true
   c. V metóde action() posunťe robota smerom vzad a náhodne ho natočte do jednej strany.

2.3. Experiment BumperCarWithHandling

2.3.1. Motivácia experimentu

Tento experiment sme si zvolili na prehĺbenie znalostí o Behaviour-Based Robotics architektúre a ako ukážku komunikácie medzi robotom a počítačom. Študenti si v ňom môžu vyskúšať ako sa posielajú informácie na robota v reálnom čase.

2.3.2. Hárok experimentu

Názov: BumperCarWithHandling

Téma: Behaviour Based Robotics, Komunikácia s robotom

Cieľ: Zoznámiť sa so spôbmi komunikácie medzi robotom a počítačom, prehĺbenie si znalostí o Behaviour-Based Robotics

Teoretické pozadie:

Prvá časť teoretického pozadia je rovnaká ako v experimente BumperCar.

Posielanie príkazov z počítača na robota budeme realizovať pomocou socketov. Robot sa bude správať ako server a počítač ako client. Jedno zo správaní robota bude mať za úlohu vytvoriť server na danom porte a následne v nekonečnom cykle prijímať príkazy z počítača. Počítač sa potom pri inicializácií bude snažiť pripojiť na daný server ako client a na základe vstupu od používateľa bude posielať dané príkazy na robota.

Úloha: Naprogramovať robota ovládaného z počítača, ktorý sa vyhýba prekážkam a stenám v miestnosti a zároveň nevychádza z vyznačeného priestoru označeného čiernou čiarou na bielej podlahe

Konceptuálny popis experimentu:

Robot nadviaže spojenie s počítačom, z ktorého bude dostávať príkazy na pohyb. Robot sa bude vyhýbať prekážkam. Ak zistí, že sa nachádza príliš blízko prekážke, zastaví, otočí sa náhodným smerom a bude pokračovať v pohybe podľa príkazov z počítača. Takisto ak sa robot bude snažiť dostať z vyznačeného priestoru tak sa zastaví, vráti sa a bude pokračovať podľa príkazov z počítača.

Usporiadanie experimentu:

Aréna: miestnosť s bielou podlahou a vyznačeným priestorom čiernou čiarou, v ktorom sa môže robot pohybovať. V miestnosti by takisto mali byť prekážky, ktorým sa robot bude vyhýbať.

Robot: pohybujúci sa robot s dvoma motormi, ultrazvukovým a farebným senzorom. Ľavý motor má robot zapojený do portu B, pravý motor je zapojený do portu C, ultrazvukový senzor je zapojený do portu 4 a farebný senzor je zapojený do portu 3.

Spojenie s robotom: tento experiment vyžaduje mať vytvorenú sieť medzi počítačom a robotom. V našom konkrétnom experimente sme využívali sieť vytvorenú pomocou Bluetooth zariadenia.

Algoritmus:

Implementácia tohto experimentu pozostáva z dvoch častí: program bežiaci na robotovi a na počítači.

Program bežiaci na počítači obsahuje nadviazanie spojenia s robotom a jednoduché prostredie na ovládanie robota. Aplikácia nadväzuje spojenie s robotom pomocou socketov. Posielanie dát prebieha cez DataOutputStream získaného z pripojenia. Grafické rozhranie je vytvorené ako jednoduchý JFrame, ktorý v sebe obsahuje KeyListener. Stlačením niektorej zo šípiek sa robotovi pošle signál, aby sa začal pohybovať

požadovaným smerom a na v grafickom okne sa vypíše text o odoslaní signálu. Na písmeno 's' robot zastaví, iné klávesy ingoruje.

Program bežiaci na robotovi sme implementovali pomocou Behaviour Based Robotics štruktúry. Robot obsahuje zadefinované tri správania.

Komunikácia robota s PC je riešená pomocou socketov a je v samostatnom vlákne. Robot vytvorí ServerSocket a čaká na pripojenie klienta, čo je v našom prípade počítač. Po nadviazaní spojenia sa toto vlákno dostane do nekonečného cyklu, v ktorom číta príkazy prijaté z klienta a zapisuje ich do globálnej premennej.

Prvé základné správanie zabezpečuje riadenie robota podľa prijatých príkazov. Toto správanie sa spúšťa automaticky. Podľa príkazu poslaného z klienta, ktorý je zapísaný v globálnej premennej, môžu nastať nasledujúce udalosti:

- ak premenná obsahuje “prava” - pravý motor robota sa zastaví
  - ľavý motor robota sa spustí smerom dopredu
- ak premenná obsahuje “lava” - pravý motor robota sa spustí smerom dopredu
  - ľavý motor robota sa zastaví
- ak premenná obsahuje “hore” - pravý motor robota sa spustí smerom dopredu
  - ľavý motor robota sa spustí smerom dopredu
- ak premenná obsahuje “dole” - pravý motor robota sa spustí smerom dozadu
  - ľavý motor robota sa spustí smerom dozadu
- ak premenná obsahuje “s” - pravý motor robota sa zastaví
  - ľavý motor robota sa zastaví

Druhé správanie DetectWall zabezpečuje vyhýbanie sa prekážkam. Ultrazvukový senzor neustále meria vzdialenosť od najbližšej prekážky pred robotom. Ak je táto vzdialenosť menšia ako konštanta, toto správanie preberie kontrolu nad robotom. Robot sa zastaví a náhodne si vyberie smer kam sa natočí. Po natočení sa toto správanie vzdá kontroly nad robotom.

Tretie správanie zabezpečuje, aby sa robot nedostal z vyznačeného priestoru. Farebný senzor meria farbu na podlahe pred robotom. Ak robot narazí na hranicu vyznačeného priestoru, ktorý je označený čiernou čiarou, toto správanie preberie kontrolou

nad robotom. Robot sa vráti naspäť do vyznačeného priestoru a vzdá sa kontroly nad robotom.

Implementácia:

Program na počítači:

Grafické rozhranie bežiace na počítači, ktoré zabezpečuje ovládanie robota, je riešené ako jednoduchý JFrame. Naša trieda Ovladanie extenduje triedu JFrame.

V main metóde triedy nadviažeme spojenie s robotom. Po úspešnom nadviazaní spojenia sa nainicializuje globálna premenná out typu DataOutputStream, ktorú budeme využívať na posielanie príkazov na robota. V main metóde sa ďalej nastavuje vzhľad grafického rozhrania.

Trieda obsahuje metódu createAndShowGUI, ktorá inicializuje grafické rozhranie, pridáva do neho prvky na vkladanie údajov ako aj na ich zobrazovanie. Grafické rozhranie obsahuje dva prvky: JTextArea na zobrazovanie zadaných údajov a JTextField na vkladanie príkazov pre robota. Do komponentu JTextField sme pridali KeyListener, ktorý budeme preťažovať v našej triede.

Prekrývame metódu keyPressed(KeyEvent arg0) z KeyListener-a. V tejto metóde switchujeme cez funkciu arg0.getKeyCode(), ktorá nám vráti aktuálne stlačenú klávesu. Rozpoznávame päť podporovaných kláves a to: VK_UP, VK_DOWN, VK_RIGHT, VK_LEFT a VK_S. Na každé z týchto tlačidiel sa pošle daný signál robotovi. Ak používateľ stlačení inú klávesu vykoná sa defaultný prípad, ktorý vypíše správu o zadaní nesprávnej klávesy.

Program na robotovi:

Hlavná trieda BumperCarWithHandling obsahuje tri definície správaní robota.

DriveForwardWithHandling trieda extenduje Behavior a zabezpečuje ovládanie robota na základe príkazov posielaných z počítača. V hlavnej triede máme samostatnú triedu ReadFromClient, ktorá extenduje Thread a slúži na prijímanie príkazov z počítača. V metóde run() tejto triedy sa vytvára ServerSocket, ktorý následne čaká na pripojenie z počítača. Po pripojení metóda v nekonečnom cykle načítava príkazy a zapisuje ich hodnotu

do globálnej premennej hlavnej triedy. Dané správanie potom v metóde action() riadi robota zadaným smerom podľa aktuálneho nastavenia tejto premennej. Toto správanie je základne a preto metóda takeControl() vracia vždy true. V metóde suppress() sa nastavuje globálna premenná triedy, ktorá ukončuje vykonávanie metódy action(). Táto metóda sa používa v prípade, že správanie s vyššou prioritou si vyžiadalo kontrolu nad robotom.

Druhé správanie bude zabezpečovať vyhýbanie sa prekážkam. Trieda DetectWallWithHandling reprezentuje toto správanie. V metóde takeControl() kontroluje nameranú hodnotu z ultrazvukového senzora. Ak je hodnota príliš malá, správanie si vyžiada kontrolu nad robotom. V metóde action() potom vráti robota smerom nazad a natočí nahodným smerom vpravo alebo vľavo. Nechceme aby používateľ mohol ovládať robota počas vyhýbaniu sa prekážky, preto metóda suppress() nerobí v praxi nic.

Trieda OffTheLineWithHandling reprezentuje správanie, ktoré zabezpečuje aby robot nevyšiel z vyznačeného priestoru. V metóde takeControl() sa kontroluje nameraná hodnota z farebného senzora. Ak senzor namerá hodnotu menšiu 0.30, čo v praxi znamená čiernu farbu, tak toto správanie si vyžiada kontrolu nad robotom. V metóde action() sa potom robot vracia späť do vyznačeného priestoru. Podobne ako v správaní predtým, nechceme aby používateľ mohol ovládať robota počas tejto akcie, preto metóda suppress() nerobí nič.

Postup:

1. Vytvorte triedu na načítavanie príkazov, ktorá bude extendovať triedu Thread. Doplňte do metódy run() tejto triedy nasledovné :
    a. Vytvorenie novej ServerSocket premennej.
    b. Čakajte kým sa klient (počítač) pripojí.
    c. Nainicializujte si premennú typu DataInputStream z novovytvoreného spojenia a v nekonečnom cykle pomocou metódy readUTF() načítavajte reťazce predstavujúce príkazy.
    d. Doplňte spracovanie týchto príkazov a ich zapisovanie do premennej s menom direction v triede BumperCarWithHandling.
2. V triede BumperCarWithHandling pridajte premennú novovytvorenej triedy a do metódy initializeRobotParameters() pridajte jej inicializáciu a štartovanie.

3. Do vnorenej triedy DriveForwardWithHandling v triede BumperCarWithHandling doplňte spracovanie príkazov na pohyb.
4. Do triedy Ovladanie doplňte nasledovné:
    a. V metóde main() nadviažte spojenie s robotom a nainicializujte si premennú typu DataOutputStream
    b. Do metódy keyPressed(KeyEvent arg0) doplňte spracovanie zvyšných príkazov

2.4. Experiment AndroidLineFollower

2.4.1. Motivácia experimentu

Tento experiment sme zvolili na ukážku komunikácie robota s mobilným telefónom pomocou Wifi. Študenti si môžu vyskúšať vzdialené pripojenie na robota z mobilnej aplikácie ako aj vyskúšať si výhody a nástrahy komunikácie pomocou Wifi dongle.

2.4.2. Hárok experimentu

Názov: AndroidLineFollower

Téma: Komunikácia robota s Androidom

Cieľ': Zoznámiť sa so spôsobmi komunikácie medzi robotom a Android zariadením

Úloha: Naprogramovať Android aplikáciu, ktorá bude sledovať čiernu čiaru na zemi a ovládať robota.

Konceptuálny popis experimentu:

Aplikácia bude pomocou kamery na telefóne snímať obraz pred robotom. Na základe vyhodnotenie obrázka aplikácia zistí či sa má robot hýbať smerom dopredu, doľava, doprava alebo má zastaviť.

Usporiadanie experimentu:

Aréna: miestnosť s bielou podlahou a čiernou čiarou vyznačujúcou dráhu robota

Robot: pohybujúci sa robot s dvoma motormi. Ľavý motor má robot zapojený do portu B a pravý motor je zapojený do portu C.

Algoritmus:

Na implementáciu tohto experimentu sme použili Android aplikáciu, ktorá sa remote pripája na robota a ovláda ho z aplikácie.

Aplikácia pomocou knižnice OpenCV analyzuje každý frame získaný z kamery mobilného telefónu. Na základe tejto analýzy aplikácia ovláda pohyb robota tak, aby zostal na čiare.

Implementácia:

Aplikácia obsahuje vnorenú triedu Control, ktorá zabezpečuje nadviazanie remote spojenia s robotom a následné vykonávanie príkazov. Táto trieda extenduje AsyncTask<String, Integer, Long>, aby sa mohla vykonávať nezávisle na behu aplikácie. Ak bol zadaný príkaz "connect" nadviaže sa remote spojenie s robotom a nainicializujú sa premenné pre motory a ich atribúty. V prípade že bol zadaný príkaz "disconnect" zruší sa spojenie s robotom. V prípade zadania príkazu na riadenie pohybu robota, sa kontroluje či je úspešné nadviazané spojenie. V prípade, že aplikácia nie je pripojená alebo nastal problém pri pripájaní v onPostExecute metóde sa používateľovi zobrazuje Toast s danou informáciou.

Samotná aplikácia obsahuje tlačidlo na pripojenie/odpojenie robota a NativeCameraView, ktorý zobrazuje obrázky z kamery. Obrázky z kamery zozbieravame pomocou implementácie interface-u CameraBridgeViewBase.CvCameraViewListener2. V metóde onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) dostávame obrázok, ktorý môžeme ďalej analyzovať. V tomto obrázku si hľadáme pozíciu stredu čiernej čiary a následne podľa jej pozície vzhľadom na obrázok určujem či má robot ísť rovno, vpravo alebo vľavo.

Postup:

1. Na tento experiment je potrebné, aby bol mobilný telefón a robot prepojený pomocou wifi siete. Postupujte nasledovne:
    a. Naštartujte robota s pripojeným wifi dongle.
    b. Vytvorte si z telefónu prenosný prístupový bod. Ako úroveň zabezpečenia použite WPA2 PSK. Meno a heslo si zvolte ľubovoľné.
    c. Pripojte sa na danú sieť z robota. V menu si nájdite Wifi, vyberte Vašu sieť, zadajte heslo a stlačte znak „D“ (Done).
    d. Po úspešnom pripojení by ste mali vidieť ip adresu na úvodnej stránke robota (192.168.xxx.xxx).
2. V triede MainActivity v metóde buttonClick(View view) doplňte ip adresu robota.
3. Do vnorenej triedy Control v triede MainActivity doplňte spracovanie príkazov na pohyb robota v metóde doInBackground(String... cmd). Na ovládanie robota používajte premenné left a right, ktoré predstavujú motory. Používajte príkazy stop() a forward().

2.5. Experiment MonteCarloLocalization

2.5.1. Motivácia experimentu

Tento experiment sme zvolili ako ukážku možností stavebnice LEGO MINDSTORMS, ktorá zvláda aj pokročilé algoritmy. Jedná sa o pravdepodobnostný algoritmus Monte Carlo lokalizácie, v ktorom sa robot bude lokalizovať v miestnosti pomocou ultrazvukového senzora.

2.5.2. Hárok experimentu

Názov: MonteCarloLocalization

Téma: Pravdepodobnostné algoritmi, lokalizácia v priestore

Cieľ: Zoznámiť sa s ideami pravdepodobnostných algoritmov, konkrétne Monte Carlo lokalizáciou

Teoretické pozadie:

Monte Carlo Localization (MCL) je jeden z najrozšírenejších lokalizačných algoritmov. Je ľahko naimplementovateľný a spoľahlivo funguje naprieč širokej škále lokalizačných problémov.

Základný MCL algorimtus reprezentuje vierohodnosť pomocou množiny pozícií. Táto množina sa inicializuje náhodnými pozíciami. Pri každej akcií robota sa pomocou modelu pohybu aktualizujú všetky pozície v množine. Následne sa za pomocí meraní vzdialeností prepočítajú váhy a náhodne sa prevzorkuje celá množina. Pozície s väčšou váhou majú väčšiu pravdepodobnosť "prežiť" prevzorkovanie.

Úloha: Naprogramovať robota, ktorý sa bude lokalizovať v priestore pomocou ultrazvukového senzora.

Konceptuálny popis experimentu:

Robot by sa mal lokalizovať v miestnosti pomocou ultrazvukového senzora. Robot bude poznať mapu miestnosti, v ktorej sa nachádza. Pomocou meraní z ultrazvukového senzora a pravdepodobnostného algoritmu Monte Carlo lokalizácie zistí kde sa v miestnosti nachádza sériou náhodných pohybov.

Usporiadanie experimentu:

Aréna: miestnosť so stenami vyznačujúca priestor v ktorom sa robot má pohybovať

Robot: pohybujúci sa robot s dvoma motormi a ultrazvukovým senzorom, ktorý sníma vzdialenosť robota od steny. Ľavý motor má robot zapojený do portu B, pravý motor je zapojený do portu C a ultrazvukový senzor je zapojený do portu 2.

Algoritmus:

Tento experiment pozostáva z dvoch častí: program bežiaci na robotovi a grafické rozhranie zobrazujúce výsledky na počítači.

Program bežiaci na robotovi obsahuje inicializáciu premených potrebných pre Monte Carlo lokalizáciu a komunikáciu s počítačom.

Vrámci inicializácie sa nastavuje DifferentialPilot, ktorý slúži na ovládanie robota a RangeScanner, čo je v našom prípade ultrazukový senzor a slúži na meranie vzdialenosti robota od stien. Takisto sa pri RangeScanner-i nastavujú aj uhly, v ktorých sa merajú hodnoty. V našom prípade to sú uhly -45, 0 a 45 stupňov. Na použitie Monte Carlo lokalizácie ešte potrebujeme mapu miestnosti a množinu častíc reprezentujúcu možné pozície robota v miestnosti. Tieto dva údaje sú posielané z počítača.

Komunikácia s počítačom je použitá za účelom zobrazovania výsledkov v grafickom rozhraní. Plní tri základné úlohy a to inicializáciu premenných, kontrolu pohybu robota a posielanie údajov na zobrazovanie z robota. Inicializuje mapu miestnosti a množinu častíc potrebných pre Monte Carlo lokalizáciu. Pohyb robota je vykonávaný náhodnými pohybmi a natočením.

Grafické rozhranie pozostáva z troch základných častí: časť zobrazujúca mapu s množinou častíc, časť s tlačidlami na ovládanie experimentu a časť s obrázkami z webkamery na porovnanie polohy robota.

Grafické rozhranie dostáva na vstupe svg súbor s mapou miestnosti, ktorú zobrazuje v príslušnej časti. V tejto mape sa takisto vytvárajú a zobrazujú častice reprezentujúce polohu a natočenie robota. Pri inicializácií sú tieto častice náhodne rozdelené po miestnosti. Postupnými náhodnými pohybmi robota sa počet pozícií častíc zmenšuje ako sa robot snaží lokalizovať.

Časť s tlačidlami na ovládanie experimentu je rozdelená do 4 sekcií. Sekcia na načítanie mapy, na pripojenie sa s robotom, na ovládanie pohybu robota a na zoom-ovanie načítanej mapy.

V grafickom rozhraní je takisto zakomponovaná časť, kde sa pomocou webkamery zisťuje pozícia robota a porovnáva sa jeho skutočná poloha s polohou vypočítanou pomocou lokalizácie.

Implementácia:

Program na robotovi:

Tento program predstavuje jednoduchú triedu MclTest. Táto trieda má niekoľko premenných, ktoré slúžia na výpočty a v ktorých si uchováva všetky dáta. Pri incializácií dát si vytvárame premennú type DifferentialPilot. Táto premenná slúži na ovládanie pohybu robota a nastavuje sa pomocou motorov, priemeru koliesok a rozchodu koliesok. Pri inicializácií sa takisto vytvára RangeScanner, ktorý zabezpečuje meranie vzdialeností od stien. Využívame nato ultrazvukový senzor a jeho mód na meranie vzdialeností. Tento senzor na našom robotovi je statický, preto používame FixedRangeScanner. Ako vstupné parametre dostáva pilota na ovládanie robota a už spomínaný mód ultrazvukového senzora. Skeneru takisto nastavujeme vyššie uvedené uhly, v ktorých má merať vzdialenosť. Najdôležitejšia premenná, ktorá zabezpečuje výpočty pozícií je premenná typu MCLPoseProvider. Táto premenná potrebuje 4 základné vstupné parametre: pilota a skener, ktoré sme práve nastavovali, mapu miestnosti typu LineMap a premennú typu MCLParticleSet, ktorá predstavuje množinu pozícií robota. Posledné dve premenné sú nastavené z počítača.

Program bežiaci na robotovi obsahuje vnorenú triedu Receiver implementujúcu interface Runnable. Táto trieda slúži na komunikáciu s počítačom. V metóde run() táto trieda nadviaže spojenie s počítačom pomocou NXTCommConnector. Po nadviazaní spojenia metóda v nekonečnom cykle prijíma príkazy z počítača. Rozpoznávame tieto príkazy:

- LOAD_MAP: z počítača sa posiela mapa typy LineMap, uloží sa do globálnej premennej a nastaví sa ako parameter pre MCLPoseProvider
- PARTICLE_SET: z počítača sa posiela množina pozícií ako premenná typu MCLParticleSet, tieto pozície sa uložia do globálnej premennej a nastavia sa ako paramter pre MCLPoseProvider
- RANDOM_MOVE: príkaz na vykonanie náhodného pohybu. Náhodne si vyberieme uhol natočenia od -90 po 90 stupňov a 1-30 centimetrov dlhú trasu. Overí sa pomocou nameraných vzdialeností či robot má dostatok priestoru pred sebou na pohyb vpred. Ak áno spraví pohyb vpred. Nakoniec sa robot natočí daným smerom.

- GET_POSE: z MCLPoseProvider-a sa zavolá metóda getPose(), ktorá vracia vypočítanú pozíciu robota ako ováhovaný priemer všetkých pozícií. Túto pozíciu robot následne posiela do počítača.
- GET_PARTICLES: údaje o množine pozícií si udržiavame v globálnej premennej. Obsah tejto premennej sa posiela do počítača
- GET_READINGS: z MCLPoseProvider sa vytiahnu údaje o meraniach a posielajú sa na počítač

Grafické rozhranie na počítači:

Ako základ sme použili grafické rozhranie pre staršiu verziu, ktoré sme si upravili pre naše potreby. Toto rozhranie je vytvorené ako trieda, ktorá extenduje JFrame. Komponenty rozhrania sú rozdelené do troch častí.

Časť, ktorá zobrazuje mapu a množinu pozícií. Táto časť sa zobrazuje v komponente typu MapPanel, čo je špeciálne vytvorená trieda pre účely zobrazovania výsledkov Monte Carlo lokalizácie. Zobrazuje načítanú mapu ako čierny pôdorys miestnosti a množinu pozícií ako červené body s tenkou červenou čiarou naznačujúcou natočenie.

Časť, ktorá obsahuje tlačídlá na ovládanie experimentu by sa dala rozdeliť na 4 sekcie:

- načítanie mapy: v tejto sekcií sa nachádza textové pole, do ktorého treba zadať meno svg súboru obsahujúceho mapu, a tlačidlo na načítanie mapy. Mapa musí byť tvorená pomocou Line komponentov, ktoré sú spolu zoskupené. Na vytváranie mapy je potrebné použiť externý program na úpravu svg súborov.
- nadviazanie spojenia: v tejto sekcií sa nachádza textové pole, do ktorého je potrebné zadať adresu robota. Je potrebné aby bola vytvorená sieť, v ktorej je pripojený ako robot tak aj počítač. Táto sekcia potom ešte obsahuje tlačidlo na pripojenie sa na robota.
- sekcia na ovládanie pohybu robota: táto sekcia obsahuje dve tlačidlá. Jedno na vykonanie náhodného pohybu. Toto tlačidlo pošle signál na vykonanie náhodného pohybu a aktualizovanie množiny pozícií, ktoré sa hneď aj prekreslia. Druhé tlačidlo slúži na získavanie pozície. Robot spraví merania pomocou skenera, prepočíta váhy a prevzorkuje množinu pozícií. Na počítač sa posiela aktualizovaná

množina ako aj ohodnotená pozícia kde sa robot nachádza. Ak je táto pozícia dostatočne presná, tak sa zakreslí do mapy.

- sekcia na zoom-ovanie mapy: táto sekcia obsahuje slider, ktorým sa škáluje veľkosť zobrazenej mapy.

Posledná časť grafického rozhrania je časť, v ktorej sa zobrazujú real-time obrázky z webkamery. Tieto obrázky sa analyzujú pomocou knižnice OpenCV a zisťuje sa aktuálna poloha robota. Miestnosť má v rohoch nalepené červené body a na robotovi je nalepený modrý bod. Knižnica dané body nájde a pomocou ich pozícií a znalosti o veľkosti miestnosti vypočíta polohu robota vzhľadom na miestnosť. Táto poloha sa potom porovnáva s pozíciou, na ktorej si robot vyhodnotil, že sa nachádza.

Postup:

1. Do triedy MclTest do metódy main(String[] args) je potrebné pridať inicializáciu všetkých premenných potrebných na výpočty. Postupujte nasledovne:
    a. Vytvorte si lokálnu premennú typu DifferentialPilot. Táto premenná ako vstupné parametre potrebuje priemer a rozchod koliesok v centimetroch, ľavý a pravý motor. Nastavte rotate speed, travel speed a acceleration.
    b. Vytvorte si lokálnu premennú typu RangeFinderAdapter. Ako vstupný parameter tejto premennej dajte distance mód ultrazvukového senzora.
    c. Nainicializujte globálnu premennú scanner. Vytvorte ju ako typ FixedRangeScannerWrapper a ako vstupné parametre pošlite lokálne premenné typu DifferentialPilot a RangeFinderAdapter. Nastavte scanneru uhly, v ktorých má merať vzdialenosti. Použite globálnu premennú angles.
    d. Nainicializujte globálnu premennú mcl. Vytvorte ju ako typ MCLPoseProvider. Ako MoveProvider použite premennú typu DifferentialPilot, mapu a počet častíc nenastavujte, tieto informácie sa posielajú z počítača. Border nastavte na 0.
    e. Do globálnej premennej pilot nastavte lokálnu premennú typu DifferentialPilot.
2. Do triedy Receiver do metódy run() doplňte spracovanie signálu RANDOM_MOVE nasledovne:
    a. Z MCLPoseProvider si získajte range readings.

b. Vytvorte si náhodný pohyb dĺžky 1 až 30 centimetrov a náhodný uhol od -90 po 90 stupňov
c. Z range readings zistite vzdialenosť od najbližšej prekážky pred robotom. Ak je táto vzdialenosť menšia ako náhodne vygenerovaná pohne sa dopredu. Pouvažujte o pridaní hranice kam sa až robot má hýbať. Aby robot zastavil až úplne pri stene je nežiadúci efekt.
d. Dočasne si pretypujte premennú pilot na RotateMoveController a pomocou funkcie rotate(double angle) otočte robota vygenerovaným smerom.

3. Spustite test na robotovi a pomocou grafického rozhrani načítajte mapu, pripojte sa na robota a sériou použití tlačidiel Get Pose a Random move sa snažte robota lokalizovať.

2.6. Experiment MonteCarloLocalization – modified

2.6.1. Motivácia experimentu

Algoritmus Monte Carlo lokalizácie je veľmi komplexný a v predchádzajúcom experimente sme si vyskúšali iba základnú verziu. Tento algoritmus má veľké množstvo modifikácií a v ďaľšom experimente si vyskúšame dve z nich. Prepočítavanie váhy za pomoci starej váhy z poslednej iterácie. Táto modifikácia napomáha prežívaniu dobrých pozícií. Druhá modifikácia, ktorú si vyskúšame je vytváranie malého počtu náhodných pozícií pri každom prevzorkovaní. Napomáha to zotaveniu robota pri násilnom premiestnení počas lokalizácie.

2.6.2. Hárok experimentu

Názov: MonteCarloLocalization - modified

Téma: Pravdepodobnostné algoritmi, lokalizácia v priestore

Cieľ: Prehĺbiť si znalosti o Monte Carlo lokalizácií, vyskúšať si iné typu preváhovania a prevzorkovania

Teoretické pozadie:

Teoretické pozadie je rovnaké ako v experimente MonteCarloLocalization.

Úloha: Upraviť preváhovaciu a prevzorkovaciu metódu z experimentu MonteCarloLocalization

Konceptuálny popis experimentu:

Robot by sa mal lokalizovať v miestnosti pomocou ultrazvukového senzora ako v predcádzajúcom experimente. Pravdepodobnostnému algoritmu, ktorý využívame na poćítanie pozície, upravíme preváhovaciu a prevzorkovaciu metódu a porovnáme výsledky s predchádzajúcim experimentom.

Usporiadanie experimentu:

Aréna: miestnosť so stenami vyznačujúca priestor v ktorom sa robot má pohybovať

Robot: pohybujúci sa robot s dvoma motormi a ultrazvukovým senzorom, ktorý sníma vzdialenosť robota od steny. Ľavý motor má robot zapojený do portu B, pravý motor je zapojený do portu C a ultrazvukový senzor je zapojený do portu 2.

Algoritmus:

Pre tento experiment budeme používať separované triedy pre výpočty pravdepodobnostného algoritmu Monte Carlo lokalizácie. Zmeny budeme robit do dvoch hlavných metód. Do metódy na preváhovanie častice, ktorá sa nachádza v triede MclParticleWrapped a do metódy na prevzorkovanie množiny častíc, ktorá sa nachádza v triede MclParticleSetWrapped.

Aktuálna funkcia na preváhovanie častice nepoužíva starú váhu. Pri každom prepočítaní sa váha najskôr nastaví na jedna a zabudne sa tak jej prechádzajúca hodnota. V novej preťaženej metóde si s určitým koeficientom alfa budeme pamätať aj starú hodnotu. Pri výpočte novej váhy využijeme starú váhu ako aj novo vypočítanú váhu. Môžeme použiť napríklad nasledjúci vzorec.

$$\text{calculatedWeight} = \text{newWeight} * \beta + oldWeight * \alpha$$

Metóda na prevzorkovanie množiny nevytvára náhodné prvky v miestnosti. Upravená verzia Monte Carlo lokalizácie ale takúto vlastnosť vyžaduje. V triede MclParticleSetWrapped upravíme metódu resample(), kde sa vykonáva prevzorkovanie. Po

prevzorkovaní tak ako ho vykoná aktuálna metóda, ešte prepíšeme posledných 5% častíc a nahradíme ich náhodne vytvorenými časticami.

Implementácia:

Používame separátne triedy z predchádzajúceho experimentu Monte Carlo lokalizácie, do ktorých je potrebné doplniť úpravy metód popísaných v sekcií Algoritmus

Postup:

1. Do triedy MclParticleWrapped do metódy calculateWeight() pridajte vypočítavanie novej váhy pomocou starej váhy a novovypočítanej. Vyskúšajte si rôzne pomery starej a novej váhy.
2. Do triedy MclParticleSetWrapped do metódy resample() doplňte generovanie náhodných pozícií. Po while cykle prepíšte posledné tri percentá prvkov poľa particles náhodne vygenerovanými pozíciami.

3. Realizácie experimentov

3.1. Experiment LineFollower

3.1.1. Príklad realizácie

Na implementovanie tohto experimentu sme použili nekonečný while cyklus, v ktorom sa najskôr kontrolu či nebolo stlačené tlačidlo na ukončenie experimentu.

```
if(Button.ESCAPE.isDown()) System.exit(0);
```

Získa sa hodnota zo senzora a skontroluje sa či je väčšia ako konštanta, v našom prípade 0.30, čo znamená, že robot vyšiel z čiary. Pomocou príkazu

```
LineFollower.leftMotor.isMoving()
```

sa zistí, ktorý motor práve beží. Tento motor sa zastaví a spustí sa druhý. Nakoniec je pridaný ďalší cyklus, ktorý beží pokiaľ hodnota zo senzora nie je menšia ako vyššie spomínaná konštanta. Tento cyklus zabezpečuje, aby sa robot vrátil späť na čiaru.

Keďže zo zadania očakávame, že robot začína svoj pohyb na čiare, môžeme pred celú implementáciu pridať spustenie jedného motora. Bez tohto spustenia by sa robot vôbec nehýbal.

```
leftMotor.forward();
```

3.1.2. Príklad študentovho výstupu

Tento experiment sme mali možnosť vyskúšať so študentami predmetu Algoritmy pre AI robotiku. Tu je príklad riešenia jedného zo študentov.

Študent si takisto zvolil nekonečný while cyklus, v ktorom na začiatku kontroluje stlačenie tlačidla na ukončenie experiment. Študent sa následne najskôr pýtal, ktorý motor sa hýbe a až následne kontroloval hodnotu nameranú senzorom. Na kontrolu nameranej hodnoty použil vlastnú metódu.

```
private static boolean inRange(float x, float a, float b) {
	return x >= a && x < b;
}
```

```
!inRange(colorSensor.value, 0f, 0.4f)
```

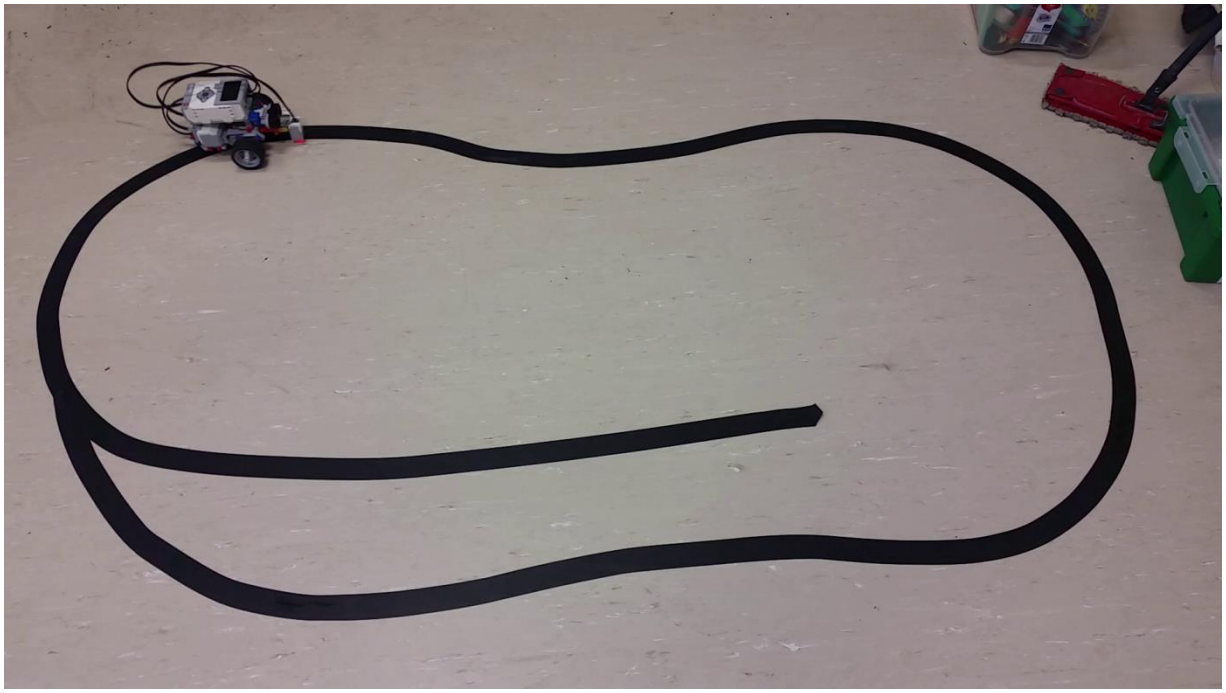
Ak táto kontrola vrátila true, motory boli vymenené. Kontrola na vrátenie sa robota na čiaru bola vynechaná. Nemalo to vplyv na správne fungovanie robota. Len v niektorých špeciálnych prípadoch je táto kontrola potrebná a ani jeden z nich nenastal v čase experimentu.

Pred nekonečným while cyklom student spúšťal pohyb robota nasledovným príkazom.

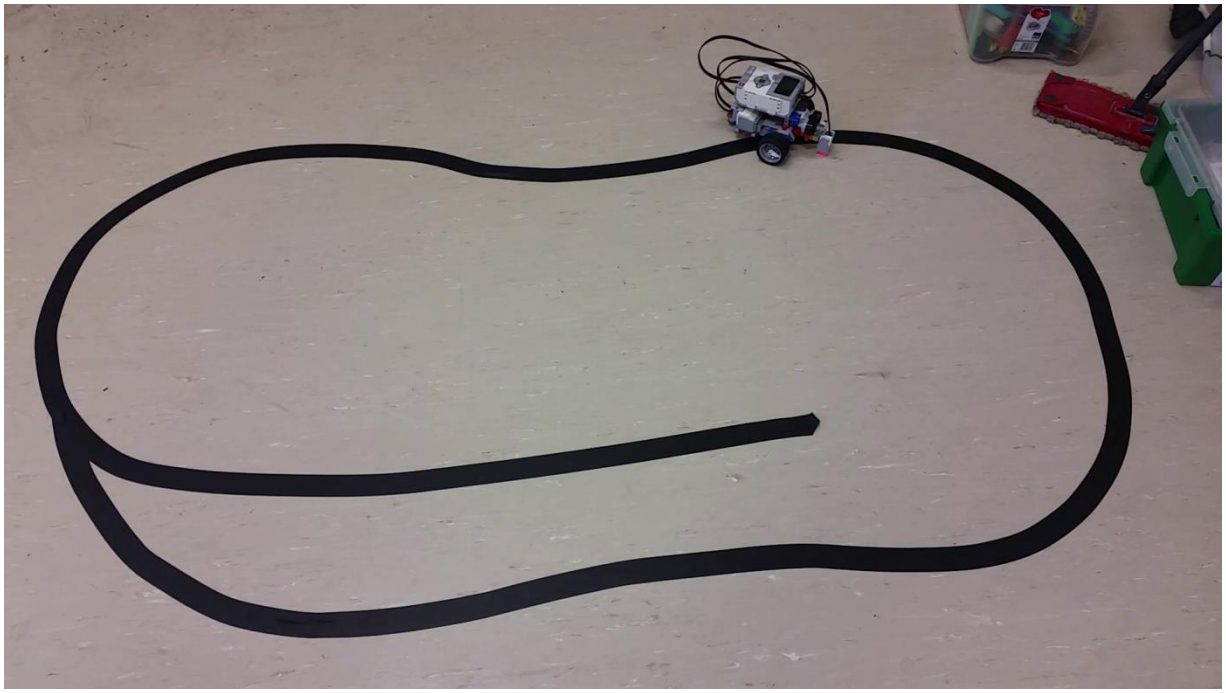
```
rightMotor.forward();
```

3.1.3. Fotodokumentácia

Na nasledujúcich obrázkoch (Obr. 1, Obr. 2) môžeme vidieť robot pomocou realizácie experimentu LineFollower sleduje čiernu čiaru.



Obrázok 1 Fotodokumentácia Experiment LineFollower



Obrázok 2 Fotodokumentácia Experiment LineFollower

3.2. Experiment BumperCar

3.2.1. Príklad realizácie

Vytvorili sme si vnorenú triedu USSensor s globálnou premennou, ktorá zabezpečuje prístup k ultrazvukovému senzoru.

```
EV3UltrasonicSensor us = new EV3UltrasonicSensor(SensorPort.S2);
```

V danej triede sme mali ešte jednu globálnu premennú, v ktorej sme uchovávali nameranú hodnotu. Táto premenná sa defaultne nastavuje na maximálnu hodnotu.

```
public float distance = 255;
```

V metóde run() tejto metódy sme si inicializovali premennú, z ktorej sme získavali vzorky.

```
SampleProvider sampleProvider = us.getDistanceMode();
```

V nekonečnom cykle sme si potom vyťahovali aktuálne vzorky a ich hodnotu sme ukladali do globálnej premennej.

```
sampleProvider.fetchSample(sample, 0);
distance = sample[0];
```

Do triedy celého experiment bola pridaná globálna premenná novovytvorenej triedy a pridaná jej inicializácia.

```
sensor = new USSensor();
sensor.setDaemon(true);
sensor.start();
```

Vytvorili sme si nové správanie DetectWall. V metóde takeControl() tejto triedy sa získa hodnota nameraná senzorom.

```
float dist = EV3BumperCar.sensor.distance;
```

Táto hodnota sa porovnáva s konštantou. V našom prípade sme si zvolili 0.15, čo predstavuje 15 centimetrov. Ak bola nameraná hodnota menšia ako táto kontštanta metóda vráti true, inak false.

V metóde action() sa najskôr robot vráti smerom vzad. Na tento pohyb používame rotáciu koliesok v zápornom uhle. Prvý motor pri rotácií takisto nastavuje premennú immediateReturn na true, aby sa rotácia koliesok vykonávala synchrónne.

```
EV3BumperCar.leftMotor.rotate(-180, true);
EV3BumperCar.rightMotor.rotate(-180);
```

Následne sa náhodne vyberie smer natočenia a robot sa natočí. Na natočenie sme použili takisto rotáciu koliesok. Jeden motor sa otáča v zápornom smere a druhý v kladnom.

```
EV3BumperCar.leftMotor.rotate(-180, true);
EV3BumperCar.rightMotor.rotate(180);
```

3.2.2. Príklad študentovho výstupu

Daný experiment sme takisto ako experiment LineFollower mali možnosť vyskúšať so študentami.

Študent si vytvoril samostanú triedu na zabezpečenie prístupu ku senzoru. V tejto triede veľmi podobne ako my si vytvoril dve globálne premenné, jednu na prístup ku senzoru a jednu na ukladanie hodnoty. V metóde run() si takisto nainicializoval mód senzora a v nekonečnom cykle si vyťahoval hodnoty zo senzora a ukladal si ich do globálnej premennej. V hlavnej triede experimentu si vytvoril a inicializoval premennú daného typu.

Študent si následne vytvoril nové správanie. V metóde takeControl() kontroluje nameranú vzdialenosť nasledovným príkazom.

```
return EV3BumperCar.sensor.value  < 0.2f;
```

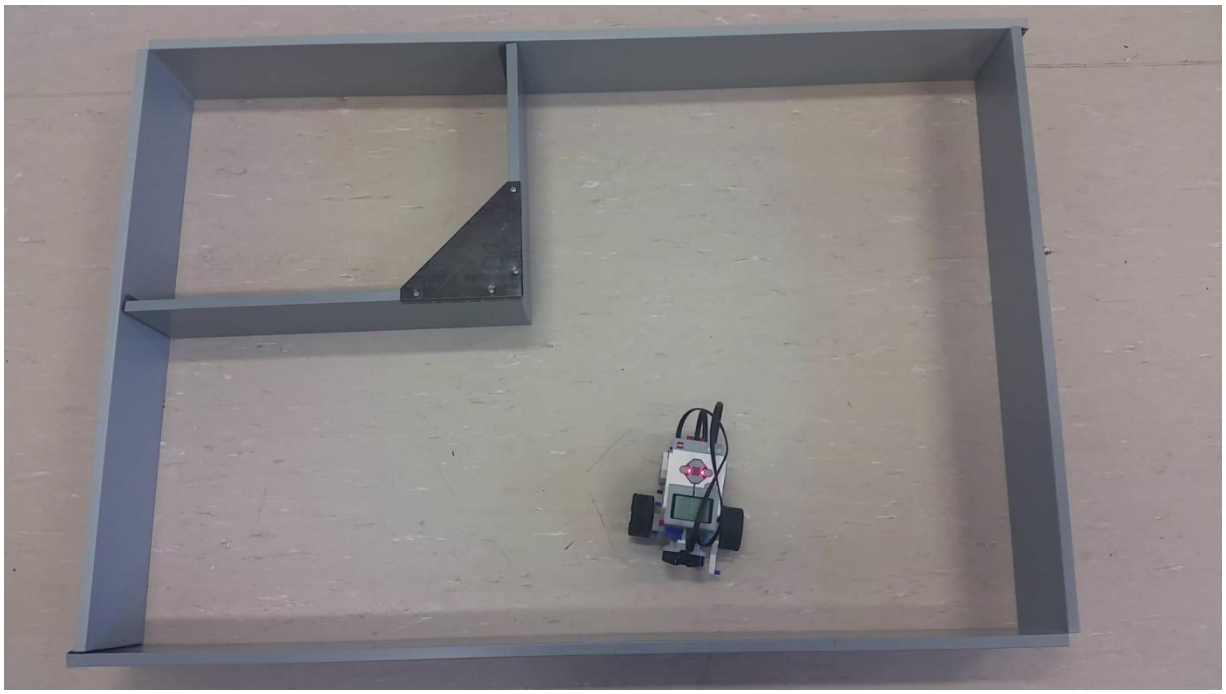
Môžeme si všimnúť, že student si zvolil inú konštantu vzdialenosti od prekážky.

V metóde action() zabezpečuje pohyb rovnako ako my v našej implementácií. Rozdiel bol vy výbere náhodného smeru. Študent použil metódu na generovanie náhodnej boolovskej premennej.

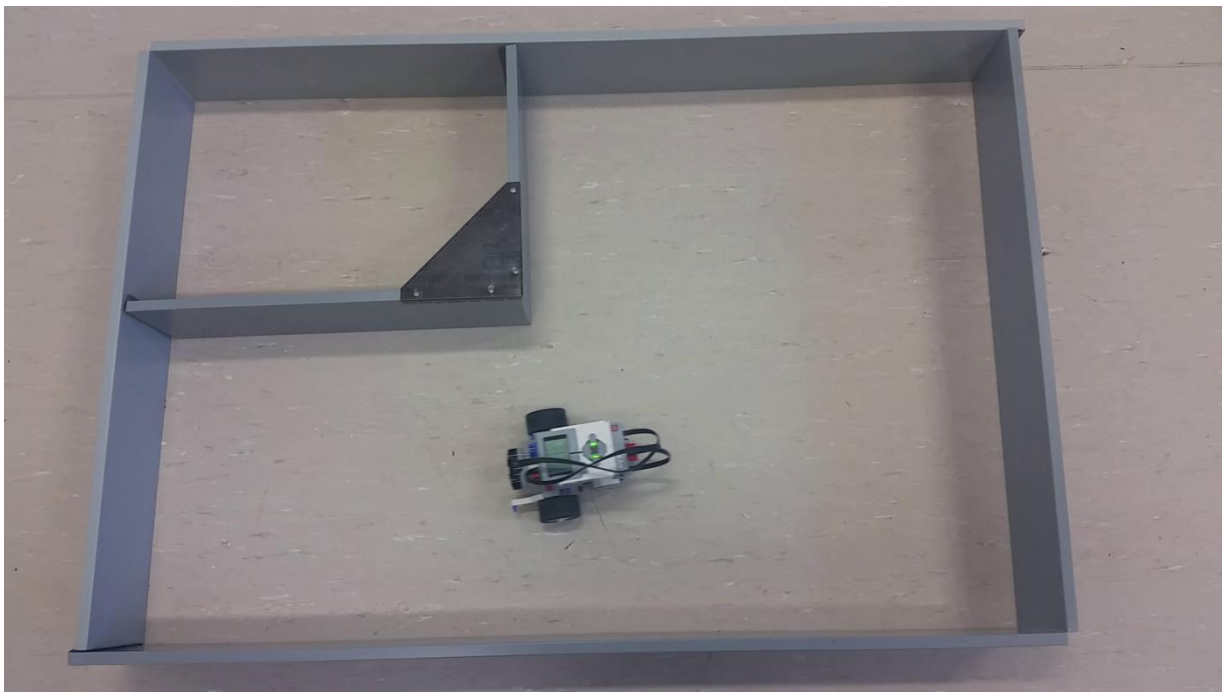
```
Random rand = new Random();
boolean temp = rand.nextBoolean();
```

3.2.3. Fotodokumentácia

Na nasledujúcich obrázkoch môžeme vidieť ako sa robot pomocou realizácie experimentu BumperCar vyhýba stenám. Ak je robot príliš blízko steny (Obr. 3) zastaví svoj pohyb vpred, náhodne sa natočí a pokračuje vpred (Obr. 4).



Obrázok 3 Fotodokumentácia Experiment BumperCar



Obrázok 4 Fotodokumentácia Experiment BumperCar

3.3. Experiment BumperCarWithHandling

3.3.1. Príklad realizácie

Vytvorili sme si novú vnorenú triedu ReadFromClient. Podľa postupu experiment sme si vytvorili nový server, čakali sme na pripojenie klienta, nainicializovali sme si premennú typu DataInputStream a v nekonečnom cykle sme načítavali príkazy z klienta.

```
serv = new ServerSocket(1111);
Socket s = serv.accept();
DataInputStream in = new DataInputStream(s.getInputStream());

...

String read = in.readUTF();
```

Signály spracovávame pomocou funkcie switch kvôli podporným výpisom a neinicializovaní neznámeho príkazu.

```
switch (read)
...
case "prava":
      BumperCarWithHandling.direction = "prava";
      System.out.println("prava");
      break;
```

Do hlavnej triedy sme potom pridali a inicializovali túto premennú, ktorá zabezpečuje prístup ku príkazom z počítača.

Do základného správania sme potom pridali spracovanie príkazov. V metóde action() tohoto správania sme do funkcie switch pridali prípady pre každý príkaz. Na základe daného príkazu sa zastavovali a spúšťali dané motory. Napríklad ak sme chceli aby robot zatáčal smerom doprava, zastavili sme pravý motor a spustili sme ľavý motor.

```
case "prava":
      BumperCarWithHandling.rightMotor.stop();
      BumperCarWithHandling.leftMotor.forward();
```

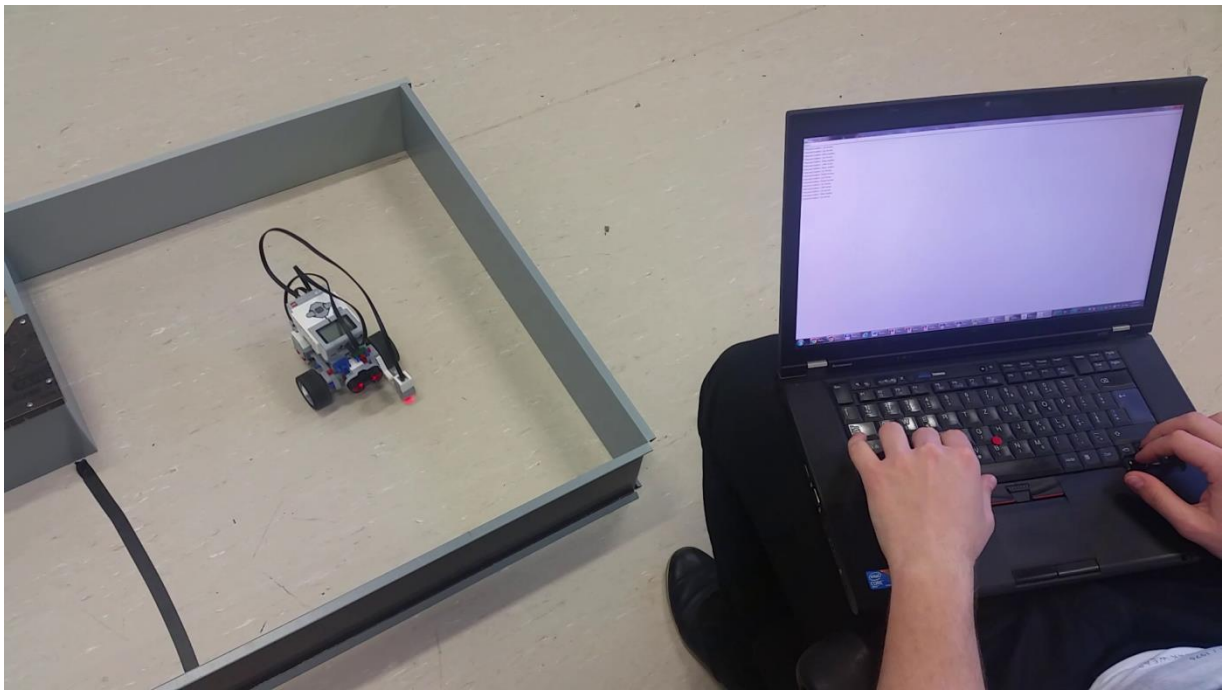
Do programu reprezentujúceho grafické rozhranie na prijímanie príkazov sme do metódy main(String[] args) pridali pripojenie sa na server (robota) a inicializovanie premennej na posielanie signálov.

```
Socket s = new Socket("10.0.1.1", 1111);
out = new DataOutputStream(s.getOutputStream());
```

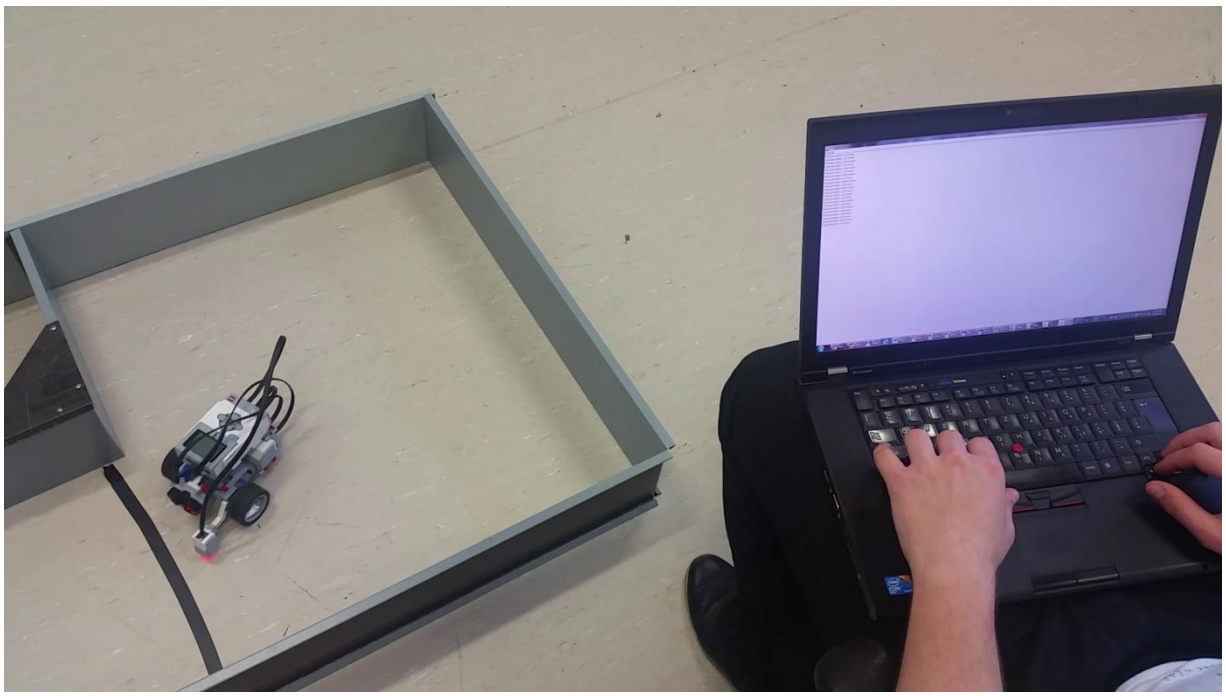
Do metódy keyPressed(KeyEvent arg0) sme podľa vzoru pridali spracovanie aj ostatných signálov.

3.3.2. Fotodokumentácia

Na nasledujúcich obrázkoch môžeme vidieť príklad realizácie experimentu BumperCarWitHandling v praxi. Robot je ovládaní pomocou počítača (Obr. 5), vyhýba sa stenám miestnosti a nevychádza z vyznačeného priestoru (Obr. 6).



Obrázok 5 Fotodokumentácia Experiment BumperCarWithHandling



Obrázok 6 Fotodokumentácia Experiment BumperCarWithHandling

3.4. Experiment AndroidLineFollower

3.4.1. Príklad realizácie

Podľa návodu v prvom bode postupu sme si vytvorili sieť medzi telefónom a robotom pomocou wifi. Získali sme ip adresu, ktorú sme pridali na požadované miesto.

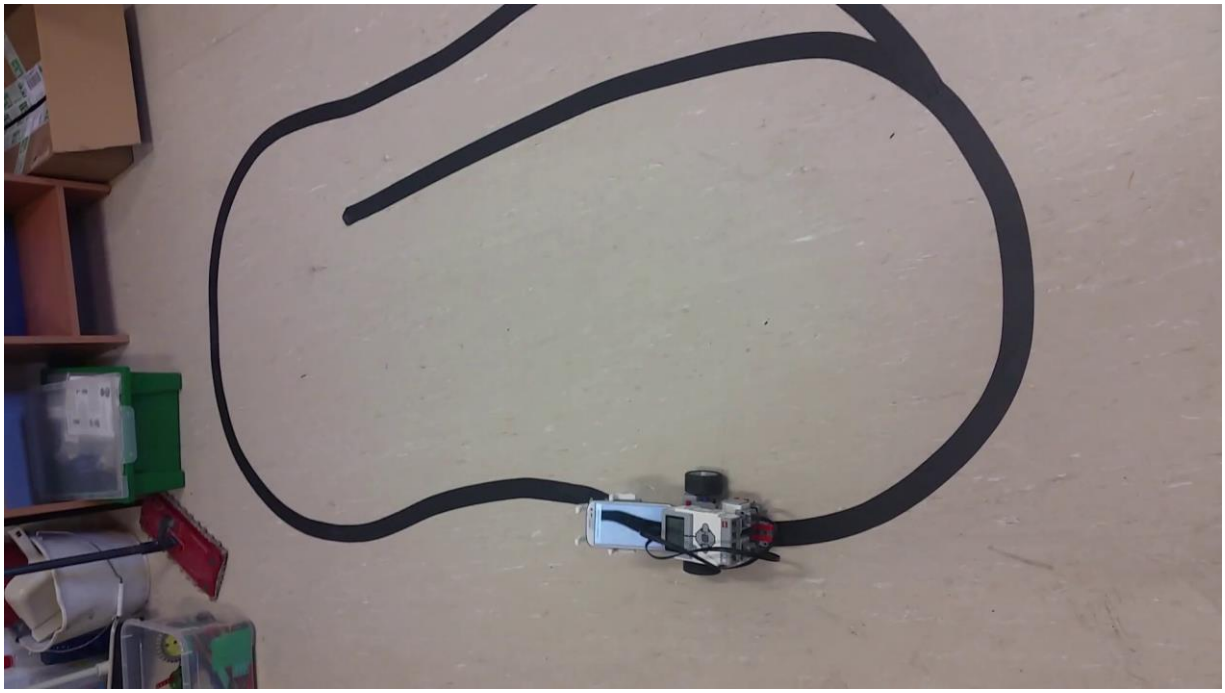
Do triedy Control sme doplnili spracovanie zadaných príkazov. Využívali sme nato remote premenné na kontrolu motorov. Motory sme vypinali a zapínali v závistlosti od zadanéo príkazu. Napríklad ak sme chceli aby sa robot otočil smerom vľavo, zastavili sme ľavý motor a spustili sme pravý motor.

```
if (cmd[0].equals("rotate left")) {
    left.stop();
    right.forward();
}
```

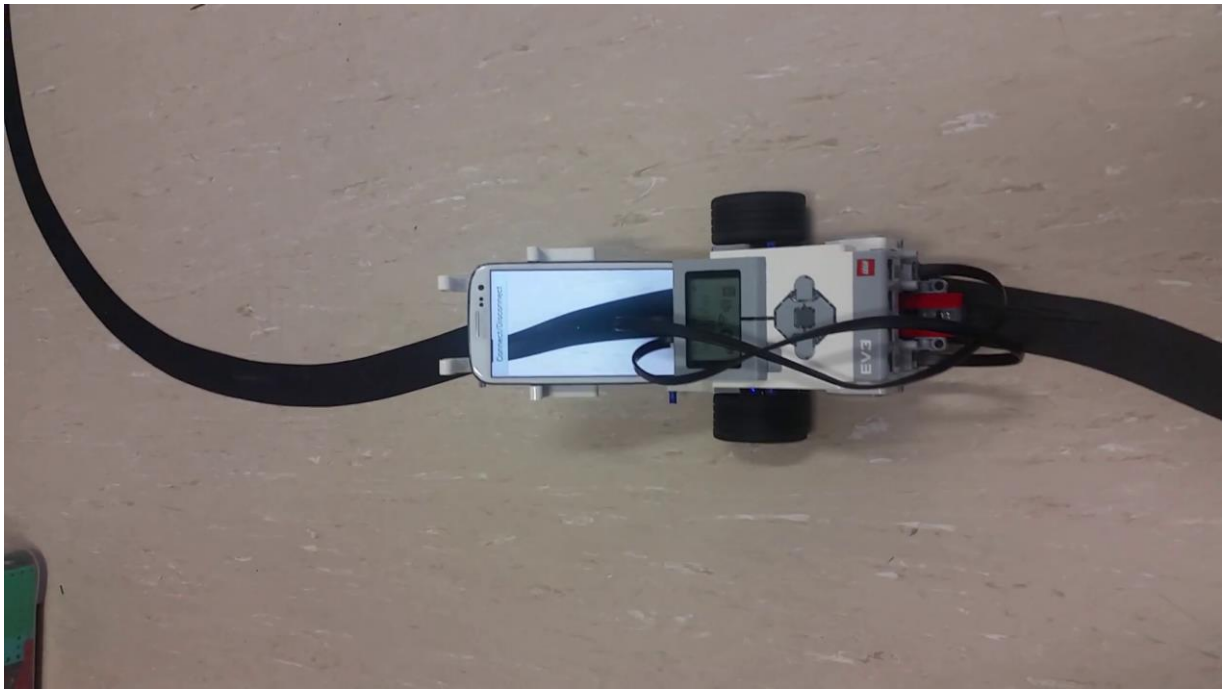
Obdobne sme spracovali aj ostatné príkazy

3.4.2. Fotodokumentácia

Na nasledujúcich obrázkoch (Obr. 7, Obr. 8) môžeme vidieť robota, ktorý sleduje čiaru pomocou kamery na mobilnom telefóne.



Obrázok 7 Fotodokumentácia Experiment AndroidLineFollower



Obrázok 8 Fotodokumentácia Experiment AndroidLineFollower

3.5. Experiment MonteCarloLocalization

3.5.1. Príklad realizácie

Podľa prvého bodu postupu sme si inicializovali všetky potrebné premenné.

Vytvorili sme si premennú typu DifferentialPilot. Zmerali sme si priemer a rozchod koliesok, nainicializovali sme si premenné motorov a poslali sme ich ako vstupné parametre. Ďalej sme si nastavili potrebné rýchlosti a akceleráciu. Danú hodnotu sme si zvolili pre plynulý, nie sekaný pohyb robota.

```
private static RegulatedMotor leftMotor = Motor.B;
private static RegulatedMotor rightMotor = Motor.C;

...

DifferentialPilot robot= new DifferentialPilot(5.6, 12.45,
leftMotor, rightMotor, false);
robot.setRotateSpeed(80);
robot.setTravelSpeed(80);
robot.setAcceleration(80);
```

Vytvorili sme si lokálnu premennú na získavanie vzdialenosti od prekážok. Ako vstupný parameter sme tejto premennej dali distance mód ultrazvukového senzora.

```
EV3UltrasonicSensor ur = new EV3UltrasonicSensor(SensorPort.S2);
RangeFinder rf = new RangeFinderAdapter(ur.getDistanceMode());
```

Nastavili sme globálne premenné podľa postupu v bodoch c,d a e.

```
scanner = new FixedRangeScannerWrapper(robot, rf);
scanner.setAngles(angles);

mcl = new MCLPoseProvider(robot, scanner, null, 0, 0);

pilot = robot;
```

Podľa druhého bodu sme pridali spracovanie signálu RANDOM_MOVE. Vygenerovali sme si náhodný posun a uhol rotácie.

```
Random random = new Random();
int angle = random.nextInt(180) + 1;
angle = angle - 90;
int distance = random.nextInt(30) + 1;
```

Z MCLPoseProvider-a sme si získali namerané vzdialenosti. Ak sú tieto vzdialenosti null robot sa nehýbe vpred iba rotuje.

```
readings = mcl.getRangeReadings();
```

Z meraní si získame vzdialenosť robota od steny v danom uhle.

```
float forwardRange = readings.getRange(0f);
```

V našej implementácií kontrolujeme či ani v jednom smere nie je robot bližšie ku stene ako náhodne vygenerovaná hodnota posunu vpred. Všetky smery kontrolujeme z praktického dôvodu, že ultrazvukový senzor pod určitým uhlom nevracal správne hodnoty a narazil do steny. Ak táto kontrola prešla robot sa posunie vpred.

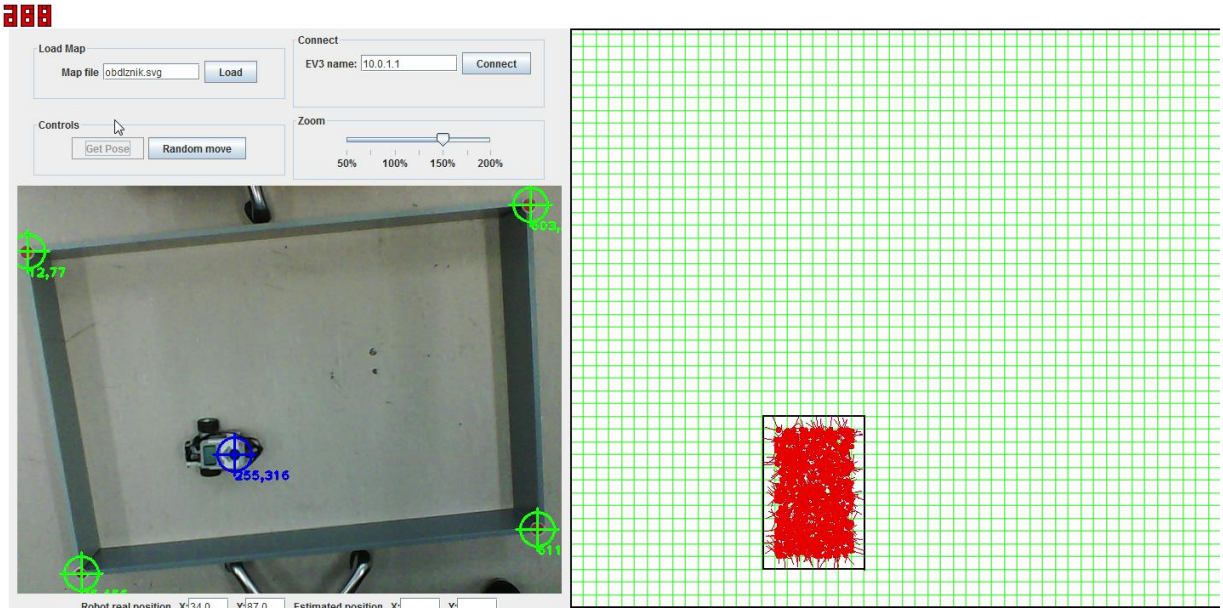
```
pilot.travel(distance,false);
```

Nakoniec sme lokálne pretypovali pilota a zrotovali sme robota o daný uhol.

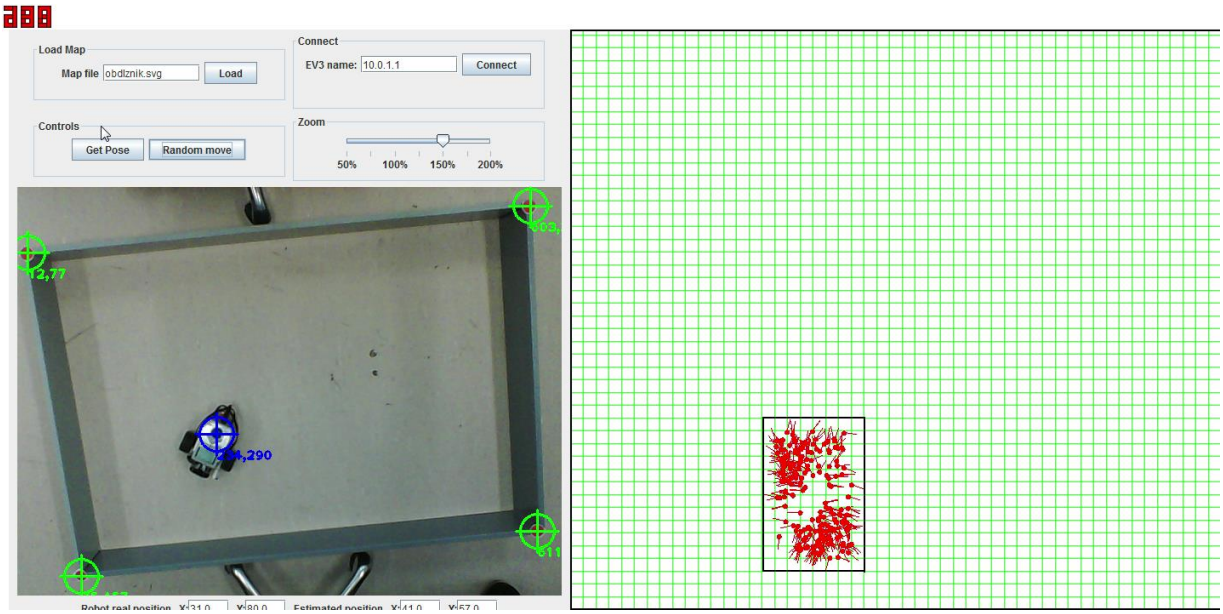
```
((RotateMoveController) pilot).rotate(angle,false);
```

3.5.2. Fotodokumentácia

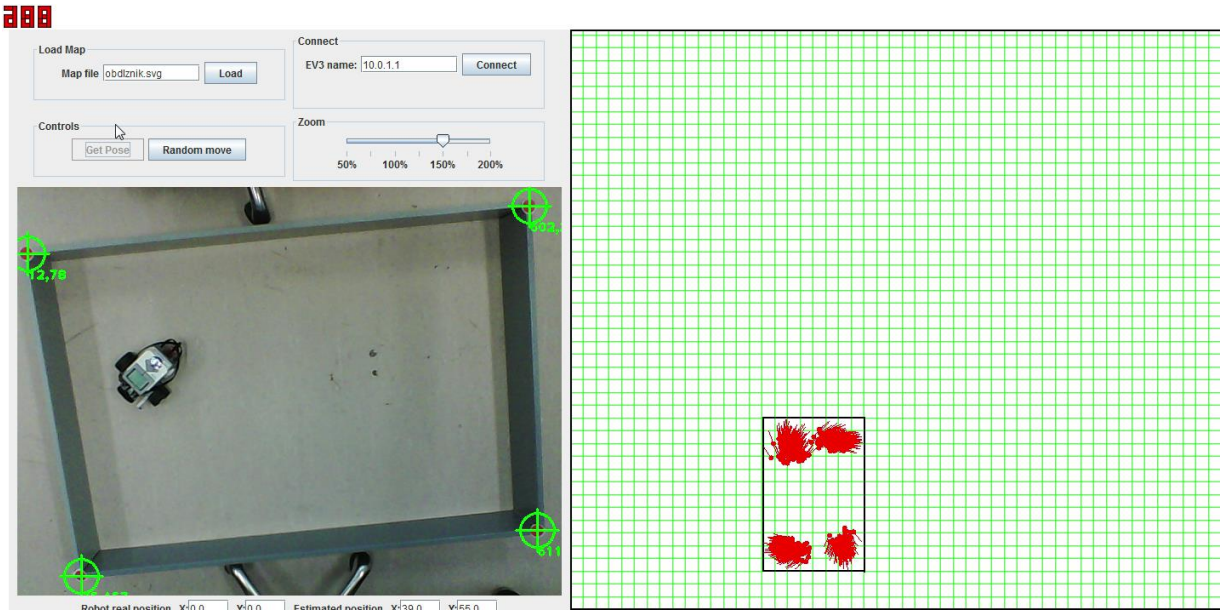
Na nasledujúcich obrázkoch vidíme príklad realizácie experimentu MonteCarloLocalization v praxi. Najskôr sa načíta mapa, vytvoria sa náhodné pozície a nadviaže sa spojenie s robotom (Obr. 9). Pomocou náhodného pohybu a meraní sa preváhujú a prevzorkujú pozície (Obr. 10). Sériou takýchto náhodných pohybou a meraní sa pozície zhlukujú najskôr do štyroch pozícií (Obr. 11), potom do dvoch (Obr. 12) a nakoniec sa robot úspešne lokalizuje (Obr. 13).



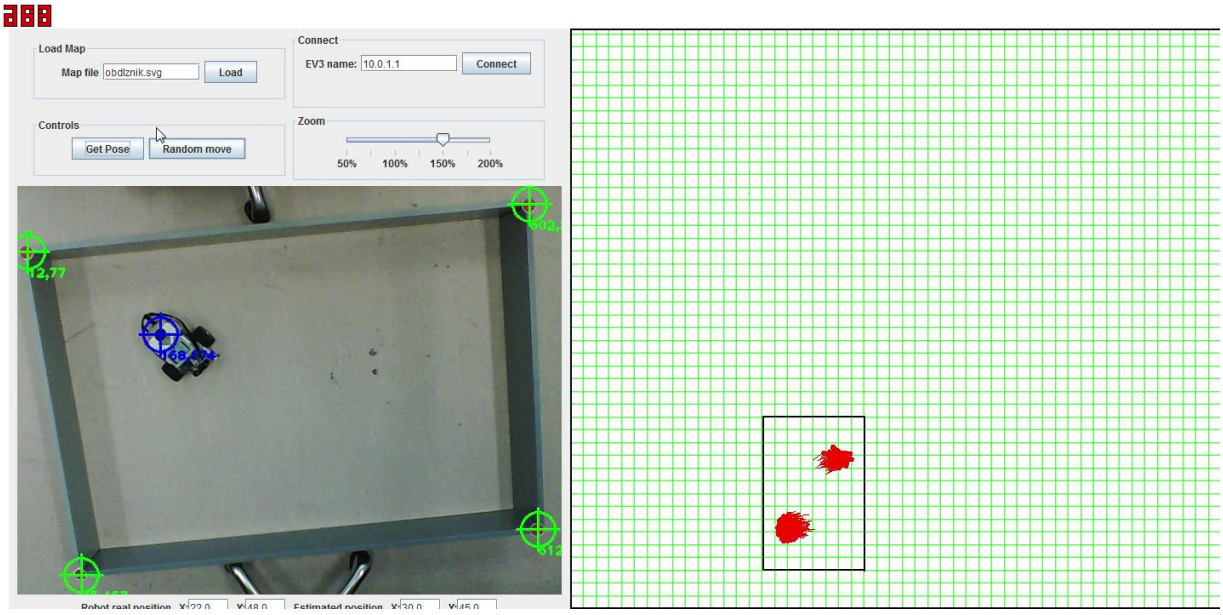
Obrázok 9 Fotodokumentácia Experiment MonteCarloLocalization



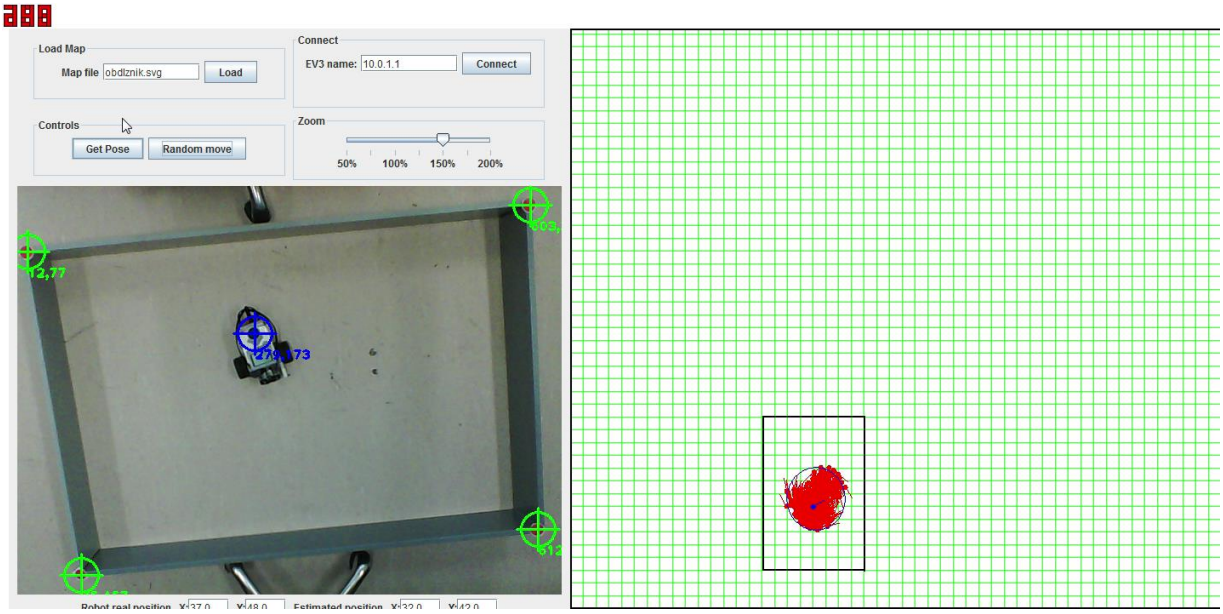
Obrázok 10 Fotodokumentácia Experiment MonteCarloLocalization



Obrázok 11 Fotodokumentácia Experiment MonteCarloLocalization



Obrázok 12 Fotodokumentácia Experiment MonteCarloLocalization



Obrázok 13 Fotodokumentácia Experiment MonteCarloLocalization

3.6. Experiment MonteCarloLocalization – modified

3.6.1. Príklad realizácie

Do metódy na prepičítavanie váh pozícií sme pridali zapamätanie si starej hodnoty.

```
float oldWeight = getWeight();
```

Celý process na vypočítanie váhy zostal nezmenený z čoho sme dostali novú váhu v danom kroku. Aktualizovanie váhy sme potom riešili nasledujúcim vzorcom.

```
setWeight(getWeight() * beta + oldWeight * alpha);
```

Experimentami sme zistili najlepší pomer starej a novej váhy a to 80% novej váhy a 20% starej váhy.

Do prevzorkovacej metódy sme pridali na koniec prepísanie posledných približne troch percent pozícií náhodne vygenerovanými.

```
for(int i = numParticles - Math.round(numParticles / 100 * 3); i < numParticles; i++){particles[i] = generateParticle();}
```

3.6.2. Zhodnotenie výsledkov

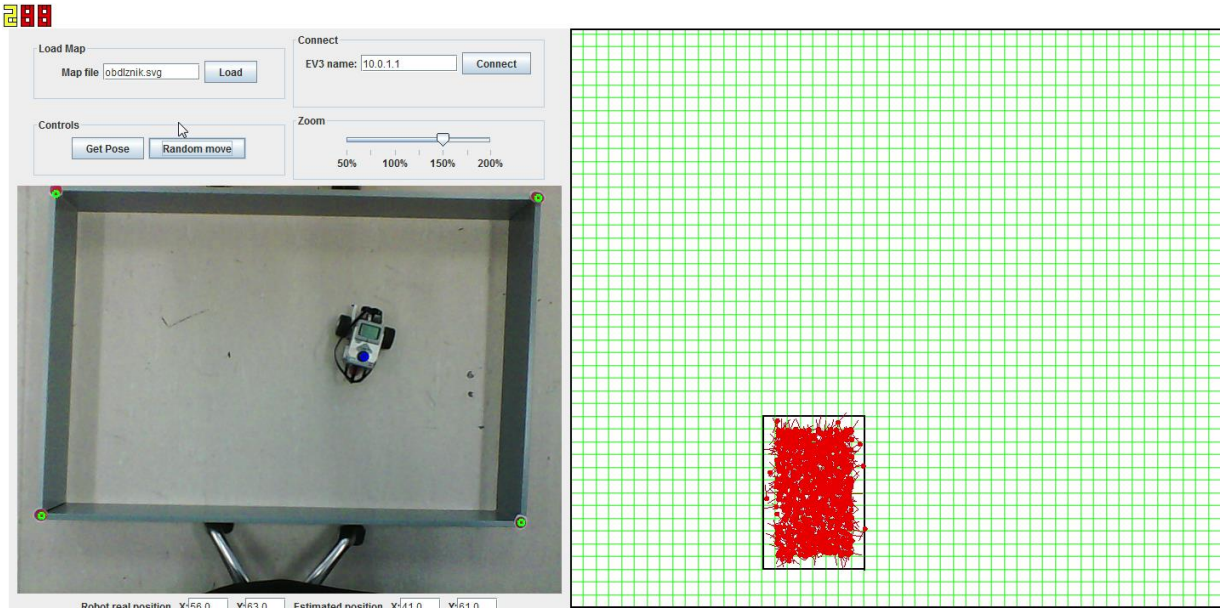
Snažili sme sa vylepšiť experiment MonteCarloLocalization. Reálne výsledky však nepriniesli veľké vylepšenie.

Prerobenie preváhovacej funkcie malo pozitívny vplyv na výslednú množinu pozícií. Tento pozitívny vplyv bol avšak zanedbateľný a v praxi nepozorovateľný.

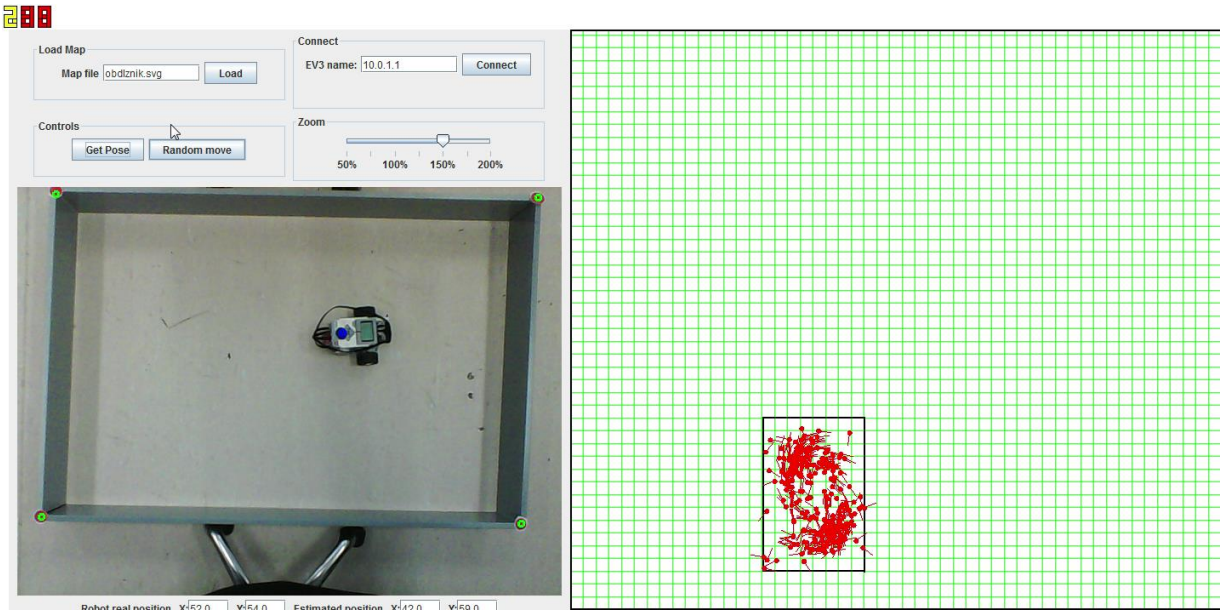
Prerobenie prevzorkovacej funkcie malo skôr negatívny vplyv na náš experiment. Generovanie náhodných pozícií pri prevzorkovaní sa do algoritmu Monte Carlo lokalizácie prirába za účelom zotavenia sa z násilného premiestnenia robota. Je potrebné ale aj prerobenie funkcie na určovanie presnej pozície robota. V našom prípade sme túto metódu neprerábali, takže náhodné pozície vygenerované pri prevzorkovaní zhoršovali schopnosť robota určiť svoju pozíciu.

3.6.3. Fotodokumentácia

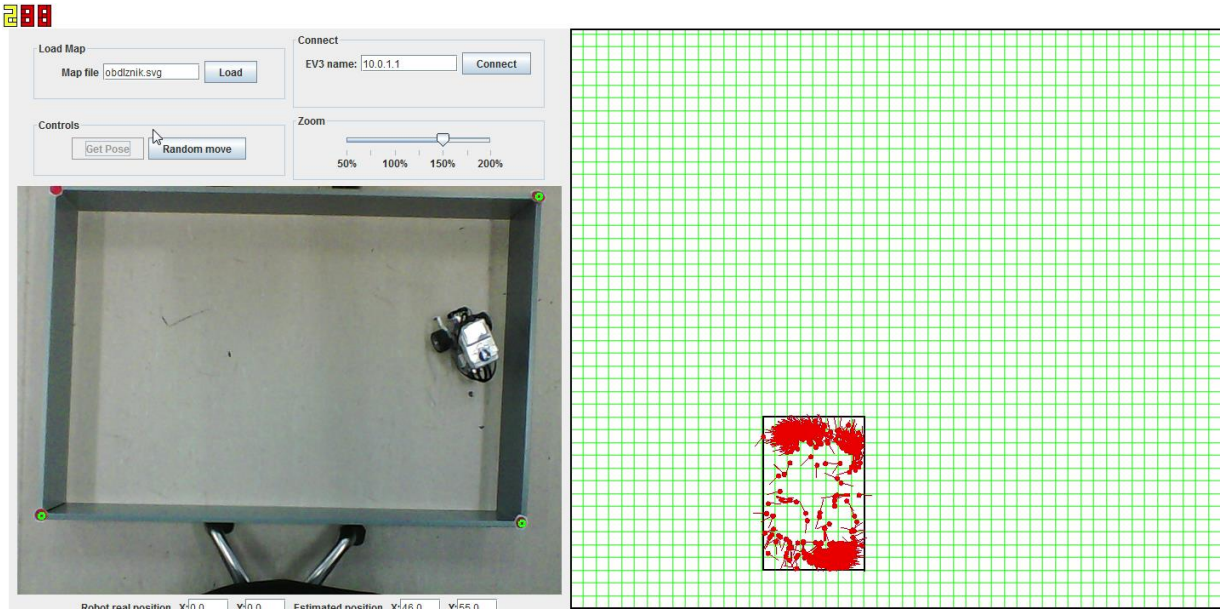
Na nasledujúcich obrázkoch môžeme vidieť príklad realizácie experimentu MonteCarloLocalization – modified v praxi. Inicializácie je rovnaká ako v predchádzajúcom experimente (Obr. 14). Pomocou náhodného pohybu a meraní sa preváhuje a prevzorkuje množina pozícií (Obr. 15). Výsledné správanie robota je veľmi podobné správaniu z predchádzajúceho experiment. Môžeme si však všimnúť nahodne vygenerované prvky pri prevzorkovaní (Obr. 16).



Obrázok 14 Fotodokumentácia Experiment MonteCarloLocalization - modified



Obrázok 15 Fotodokumentácia Experiment MonteCarloLocalization - modified



Obrázok 16 Fotodokumentácia Experiment MonteCarloLocalization – modified

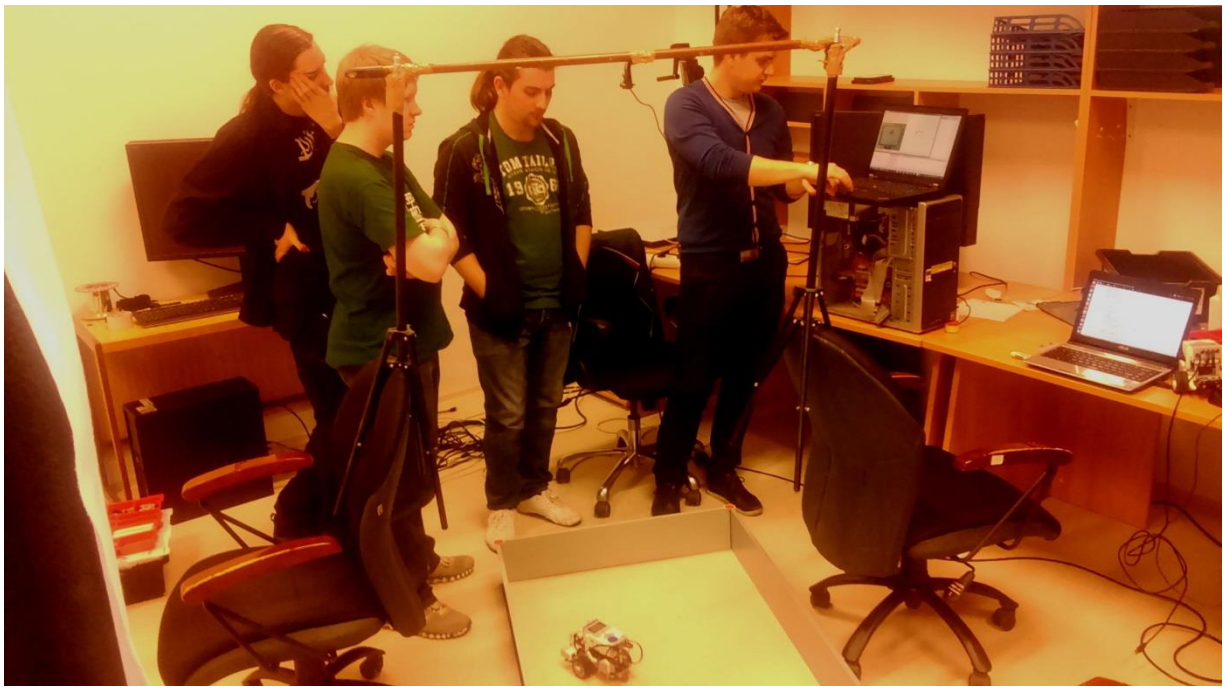
4. Overenie experimentov so študentami

Počas práce na experimentoch sme mali možnosť otestovania týchto experimentov aj so študentami.

4.1. Predmet Algoritmy pre AI robotiku

Robili sme jedno cvičenie na predmet Algoritmy pre AI robotiku pre študentov fakulty Matematiky, fyziky a informatiky Univerzity Komenského. Študenti si poskladali základnú verziu robota (Obr. 17), na vlastných počítačoch si nakonfigurovali prostredia na vývoj experimentov a podľa zadaní si vyskúšali experimenty LineFollower a BumperCar. Študentom potom bola vysvtelený a predvedený experiment MonteCarloLocalization (Obr. 18).

Obrázok 17 Cvičenie so študentami predmetu Algoritmy pre AI robotiku



Obrázok 18 Cvičenie so študentami predmetu Algoritmy pre AI robotiku. Aktuálna poloha robota sa spracováva z kamery pri pohľade na scénu zhora a porovnáva sa s lokalizáciou, ktorú zistí algoritmus a vizualizuje v grafickom okne riadiaceho softvéru.

4.2. Konferencia POPinfo 2015

Na informatickej konferencii POPinfo 2015, ktorá sa konala na Gymnáziu Viliama Paulinyho-Tótha v Martine, sme robili workshop s robotickou stavebnicou LEGO MINDSTORMS EV3.

Študenti gymnázia si postavili základnú verziu robota, bola im vysvetlená základná idea pravdepodobnostných algoritmov a neurčitosti, bol im predvedený názorný príklad pravdepodonostného algoritmu pomocou experimentu MonteCarloLocalization (Obr. 19, Obr. 20) a spolu sme vytvorili experiment, kde robot sledoval čiernou farbou vyznačenú cestu za pomoci dvoch farebných senzorov.

Obrázok 19 Workshop so študentami Gymnázia Viliama Paulinyho-Tótha v Martine

Obrázok 20 Workshop so študentami Gymnázia Viliama Paulinyho-Tótha v Martine

Záver

Hlavné ciele, ktoré sme si stanovili na začiatku našej práce sa nám podarilo splniť. Vytvorili sme sériu šiestich experimentov pre robotickú stavebnicu LEGO MINDSTORMS EV3. Experimenty pokrývajú základné zoznámenie sa so stavebnicou, získavanie údajov zo senzorov, implementovanie Behaviour-Based Robotics architektúry, komunkáciu s počítačom a mobilným telefónom, implementáciu pravdepodobnostného algoritmu, konkrétne Monte Carlo lokalizácie. Výsledkom práce je sada cvičení založených na týchto experimentoch.

Pri vytváraní experimentov sme sa stretli s radou problémov spôsobenou hlavne faktom, že stavebnica bola vydaná v roku zadania práce. Z tohoto dôvodu existovalo len málo riešení na problémy, ktoré nastali a veľkú časť z nich sme museli vyriešiť samostatne. Napríklad nastali problémy so softvérom, ktorý sme používali. Tento softvér sa nahráva z nakofigurovanej SD karty. Mali sme problémy s formátovaním karty, s poškodenými zdrojovými súbormi na karte, načítavaním softvéru na robota a rôzne chyby v implementácií zdrojových súborov.

Stavebnica má obrovský potenciál a v našej práci sme ukázali iba zlomok z toho, čo dokáže. Priestor na ďalšie experimenty je obrovský. Uveďme len niektoré z nápadov na ne: mapovanie miestnosti, trénovanie neurónovej siete, treénovanie rekurentnej siete pomocou evolučného algoritmu a iné.

Hlavné prínosy práce

V krátkosti si uveďme hlavné prínosy práce

- Vytvorenie vzdelávacieho materiálu pre študentov robotiky a východiskový prieskum pre využitie vo výskume
- Spracovanie niektorých z vlastností stavebnice LEGO MINDSTORMS EV3
- Nové spôsoby práce so stavebnicou pomocou softvéru od spoločnosti leJOS, integrovanie webkamery na spracovanie a porovnanie výsledkov
- Vlastná modifikácia a jej úspešné overenie algoritmu pravdepodobnostnej lokalizácie

Literatúra

[1] TRUN, Sebastian, BURGARD, Wolfram, FOX, Dieter. Probabilistic Robotics, Hardcover, 2005. ISBN 978-0-2622-0162-9

[2] RUSSELL, S. J. - NORVIG, P.: Artificial Intelligence. Second edition. Upper Saddle River, New Jersey: Pearson Education, Inc., 2003. ISBN 0-13-790395-2

[3] LUGER, G. F.: Artificial Intelligence. Fifth Edition. Essex: Pearson Education, 2005. ISBN 0-321-26318-9

[4] MURPHY, R.: Introduction to AI Robotics. MIT Press, 2000. ISBN 0-262-13383-0

[5] ARKIN, R. C.: Behavior-based Robotics. Third printing. MIT Press, 1998. ISBN 0262011654

[6] Boe-Bot Robot. In Parallax.com. [online]. 2015. [Citované 02.05.2015]. Dostupné na internete: https://www.parallax.com/product/boe-bot-robot

[7] Khepera IV. In K-team.com. [online]. 2015. [Citované 02.05.2015]. Dostupné na internete: http://www.k-team.com/mobile-robotics-products/khepera-iv

[8] LEGO MINDSTORMS. In Lego.com. [online]. 2015. [Citované 02.05.2015]. Dostupné na internete:http://www.lego.com/cs-cz/mindstorms/?domainredir=mindstorms.lego.com&ignorereferer=true

[9] About ROS. In Ros.org. [online]. 2015. [Citované 02.05.2015]. Dostupné na internete: http://www.ros.org/about-ros/

[10] Welcome to Robotics Developer Studio. In Msdn.microsoft.com. [online]. 2015. [Citované 02.05.2015]. Dostupné na internete: https://msdn.microsoft.com/en-us/library/bb648760.aspx

[11] Microsoft Shuts Down Its Robotics Group. In Spectrum.ieee.org. [online]. 2015. [Citované 02.05.2015]. Dostupné na internete: http://spectrum.ieee.org/automaton/robotics/robotics-software/microsoft-shuts-down-its-robotics-group

[12] Webots overview. In Cyberbotics.com. [online]. 2015. [Citované 02.05.2015]. Dostupné na internete: https://www.cyberbotics.com/overview

[13] RobotC. In Robotc.net. [online]. 2015. [Citované 02.05.2015]. Dostupné na internete: http://www.robotc.net/

[14] NXC. In Bricxcc.sourceforge.net. [online]. 2015. [Citované 02.05.2015]. Dostupné na internete: http://bricxcc.sourceforge.net/nbc/

[15] LEGO MINDSTORMS EV3. In Dai.fmph.uniba.sk. [online]. 2015. [Citované 02.05.2015]. Dostupné na internete: http://dai.fmph.uniba.sk/projects/lego/

[16] CMU Robotics Academy. In Robotics Academy. [online]. 2015. [Citované 04.05.2015]. Dostupné na internete: http://education.rec.ri.cmu.edu/

[17] FIRST LEGO League. In Firstlegoleague.org. [online]. 2015. [Citované 04.05.2015]. Dostupné na internete: http://www.firstlegoleague.org/

1. Prílohy

1. CD so zdrojovými kódmi aplikácie, diplomovou prácou vo formáte PDF a videodokumentáciou experimentov.