UNIVERZITA KOMENSKÉHO V BRATISLAVE

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

# Lokalizácia robota pomocou senzorov na meranie vzdialenosti

Rok predloženia: 2011

Tomáš Štibraný

# UNIVERZITA KOMENSKÉHO V BRATISLAVE

# FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY


## Lokalizácia robota pomocou senzorov na meranie vzdialenosti

Bakalárska práca

| | |
|---|---|
| Študijný program: | **Aplikovaná informatika** |
| Študijný odbor: | **9.2.9 Aplikovaná informatika** |
| Školiace pracovisko: | **Katedra Aplikovanej Informatiky** |
| Školiteľ: | **Mgr. Pavel Petrovič, PhD.** |
| Evidenčné číslo: | **949b5e3b-742d-425f-b1c8-6b48378791e8** |


Bratislava 2011 Tomáš Štibraný


Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Tomáš Štibraný
**Študijný program:** aplikovaná informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
**Študijný odbor:** 9.2.9. aplikovaná informatika
**Typ záverečnej práce:** bakalárska
**Jazyk záverečnej práce:** slovenský

**Názov:** Lokalizácia robota pomocou senzorov na meranie vzdialenosti

**Cieľ:** Robot s jedným senzorom na meranie vzdialenosti sa nachádza v obdĺžnikovej miestnosti. Nepozná svoju polohu a orientáciu. Úlohou robota je presunúť sa do stredu oblasti. Ako sa riešenie zmení, ak má robot dva, tri, alebo viac senzorov? Následne sa robot pohybuje a potrebuje poznať svoju polohu v každom okamihu. A ak sa v prostredí pohybujú i ďalšie roboty? Úlohou študenta je preskúmať tieto situácie, navrhnúť riešenia a overiť ich na reálnom alebo simulovanom robotovi.

**Vedúci:** Mgr. Pavel Petrovič, PhD.

**Spôsob sprístupnenia elektronickej verzie práce:**
bez obmedzenia

**Dátum zadania:** 01.10.2010

**Dátum schválenia:** 25.10.2010

Štibraný

(*) doc. RNDr. Mária Markošová, PhD.
garant študijného programu

(*) študent

Pavel Petrovič

vedúci

Ďakujem môjmu školiteľovi, Mgr. Pavel Petrovič, PhD., za odborné vedenie, cenné rady, nápady a pripomienky k mojej práci a čas strávený pri konzultáciách.

Čestne prehlasujem, že som túto bakalársku prácu vypracoval samostatne s použitím uvedených zdrojov a literatúry.

Bratislava 3.6.2011 ….....

Tomáš Štibraný

# Abstrakt

Táto práca sa zaoberá problémom lokalizácie robota v prostredí obdĺžnikovej miestnosti. V prvej časti zhromažďuje teoretické poznatky potrebné k hlbšiemu pochopeniu problému. V druhej časti sa snaží navrhnúť, opísať implementáciu a zdokumentovať aplikáciu, ktorá tieto poznatky využíva k úspešnej lokalizácii robota. Výsledná aplikácia je schopná lokalizovať robota v obdĺžnikovej a štvorcovej miestnosti za pomoci odometrie a senzorových vnemov.

**Kľúčové slová:**

robotnačka, robot, lokalizácia, metóda Monte Carlo, odometria

# Abstract

This thesis aim is to investigate the problem of robot surrounded by rectangular room localization. In first part, it groups theoretical knowledge base necessary for deeper understanding of this problem. In second part it tries to design, describe implementation and document aplication, which uses gathered knowledge to successfully locate the robot. Actual aplication is able to locate the robot in rectangular room using odometry and senzor-measured values.

# Obsah

# 1 Úvod

## 1.1 Motivácia práce

V dnešnej dobe ide vývoj veľmi rýchlo dopredu a robotika nie je výnimkou. Čím ďalej, tým častejšie sa stretávame aj v našom každodennom živote s robotmi rôznych druhov. Medzi prvými možno boli kuchynskí roboti, teraz prichádzajú do módy autonómne robotické vysávače. Niektoré sú jednoduchšie a iba chodia po miestnosti, iným treba zadať tvar miestnosti a oni ju potom celú povysávajú. V Japonsku už je dokonca bar v ktorom používajú robotov namiesto čašníkov.

Ako ide vývoj robotov dopredu, pokladá sa na robotov stále viac požiadaviek a jednou z hlavných všeobecných požiadavok na akéhokoľvek mobilného robota sa stáva, aby sa robot vedel orientovať vo svojom okolí. Toto však nejde bez toho, aby sa robot vedel v tomto prostredí lokalizovať. Z tejto potreby vychádza aj moja záverečná práca. Moja práca je však zameraná na špeciálny prípad lokalizácie a to na lokalizáciu robotov, hrajúcich robotický futbal.

## 1.2 Cieľ práce

Cieľom práce je preskúmať možnosti orientácie robota v obdĺžnikovej miestnosti a to za pomoci informácie o jeho pohybe po miestnosti a senzorických vnemoch. Robot sa nazačiatku nachádza v miestnosti, ale nevie kde. Mojou úlohou je navrhnúť aplikáciu, ktorá dostane robota do stredu miestnosti, v ktorej sa nachádza a aplikáciu realizujúcu hľadanie polohy robota.

Nájsť polohu robota znamená nájsť jeho lokálne súradnice vzhľadom na miestnosť a jeho natočenie vzhľadom na miestnosť.

Moja práca musí brať v úvahu, že robot sa v reálnej situácii (pri robotickom futbale)

bude hýbať autonómne a lokalizovať sa potrebuje „za chodu“ a tiež možnú (v robotickom futbale istú) prítomnosť iných robotov.

# 1 Východiská

## *1.1 Robotnačka – kresliaci robot*

### 1.1.1 Všeobecný popis

Robotnačka je autonómny mobilný kresliaci robot. Je určený hlavne na spestrenie programovacích jazykov typu Logo, ale dá sa ovládať takmer akýmkoľvek programovacím jazykom(už spomenuté Logo, C, Java, Pascal, …) a preto slúži aj ako experimentálna platforma pre prácu s robotom.

Robot má dve kolesá spojené diferenciálom a ovládateľné kresliace pero v strede svojho tela, ktoré sa dá meniť a teda robot môže kresliť rôznymi farbami a na rôzne povrchy podľa toho, aké pero má práve prichytené. Srdcom Robotnačky je 8 bitový mikrokontroler, reagujúci na požiadavky v reálnom čase. Dá sa ovládať bezdrôtovo cez BlueTooth za pomoci počítača umiestneného pri robotovi.

Obrázok 1. Robotnačka – kresliaci robot.

Robotnačka je navrhnutá tak, aby pomohla pri výučbe základov programovania v prostredí Imagine vytvorenom na Univerzite Komenského v Bratislave. Je to vlastne fyzická reprezentácia korytnačky v Imagine a tým pridáva tomuto prostrediu novú dimenziu reálneho sveta a robí programovanie s korytnačkou zaujímavejším.

Robot bol navrhnutý a vyvinutý spoločnosťou MicroStep-MIS.

### 1.1.2 Technická špecifikácia

Robotnačka má kruhový tvar a jej priemer je presne 210 mm. Kresliace pero sa nachádza presne v strede medzi dvoma kolesami, ktorých priemer je 50 mm a rozchod 180 mm. Na jej hlavnej doske sa nachádzajú dva krokové motory, mechanizmus na ovládanie pera, elektronika, štyri farebné LED diódy, pomocou ktorých dáva robot svetelné informácie(napríklad, že batéria je slabá), malý reproduktor, hlavný spínač a konektory.

Otočiť sa o 360 stupňov znamená pre robota 2880 krokov oboch motorov v opačných smeroch.

Robot je poháňaný dvoma krokovými motormi MICROCON SX17. Vďaka nim a presne tvarovaným hliníkovým kolieskám je robot schopný prejsť necelých 0,2 mm na jeden krok. Robot je vybavený prvkami, ktoré umožňujú plynulú akcelerácia a deceleráciu, čo zaručuje, že počas počiatočných a konečných stavov vykonávania príkazu sa nestratia žiadne kroky. Ovládanie robota je založené na open loop systéme(viď. nižšie).

Na svojej spodnej strane je robot vybavený šiestimi IR senzormi, ktoré robotovi umožňujú sledovať povrch plochy, po ktorej sa pohybuje, alebo zistiť, že sa nachádza napríklad na konci stola a už nesmie ísť ďalej. Robot je vybavený aj 5 ultrazvukovými senzormi typu SRF08, ktoré merajú vzdialenosti robota od najbližšej pevnej prekážky v centimetroch. Jeden senzor je umiestnený na prednej strane robota a je natočený v smere pohybu robota. Zvyšné štyri sú umiestnené vzhľadom na smer pohybu robota v uhloch 45°, 90°, 270° a 315°.

Centrálna výpočtová jednotka je založená na Atmel AT89S8252 8-bit microkontroleri s

taktovacou frekvenciou 11MHz.

Robot je hnaný 6V/3,2 Ah akumulátorom s výdržou približne troch hodín práce. Pri odstavenom robotovi je výdrž batérie približne jeden a pol dňa.

Na veľmi slabú baterku reaguje robot vyslaním dlhého zvukového signálu a následne sa sám vypne. V takomto prípade je potrebné zapojiť ho do siete. Nabíjanie je indikované svietiacou červenou LED diódou počas celého procesu nabíjania. Po dokončení nabíjania svieti zelená dióda. Robot sa dá používať, aj keď batéria nie je nabitá úplne. Úplné nabitie batérie trvá približne 6 hodín.

Robot váži približne 2 kg.

### 1.1.3 Komunikácia

Komunikácia robota sa vykonáva cez BlueTooth interface za pomoci modulu vyvinutého spoločnosťou MicroStep-MIS založeného na Infineon BT čipoch. Čip má dosah približne 10m.

Interfejs na komunikáciu s robotom kontroluje tok dát do počítača a aj posúvanie príkazov z počítača robotovi. Taktiež umožňuje aj komunikáciu s viacerými robotmi naraz. Dáta sú posielané po packetoch. Prijatie packetu poslaného počítačom je potvrdené packetom, ktorý robot pošle ako odpoveď na požiadavku. Robot navyše posiela stavové packety, keď sa nastane predprogramovaná situácia alebo keď dokončí zadanú úlohu.

Robot je ovládaný za pomoci jednoduchých príkazov typu "fd x", ktorý znamená, že robot sa má posunúť o x krokov dopredu. Okrem pozičných príkazov pozná aj iné, napríklad príkazy na ovládanie pera, príkaz na zmeranie vzdialeností za pomoci senzorov, príkaz na zistenie stavu batérie, príkazy na vydávanie zvuku a ovládanie rýchlosti motorov, príkazy na kreslenie textu a kalibráciu robota a mnohé iné. Všetky príkazy sú vysielané virtuálnym sériovým portom počítača cez BlueTooth robotovi.

Užívateľ môže kontrolovať robota priamo cez Imagine Logo, Java aplikácie alebo akýkoľvek iný programovací jazyk, ktorý vie použiť DLL alebo ActiveX knižnicu.

Všetok software pre Robotnačku je open-source.

### 1.1.4 Príslušenstvo

K robotovi sa dá pripojiť robotická ruka umožňujúca prenášanie malých predmetov, alebo bezdrôtová kamera napájaná z baterky robota.

## *1.2 Odometria*

Odometria je proces transformácie dát o pohybe koliesok robota na dáta o relatívnej polohe robota, vzhľadom na jeho pôvodnú pozíciu. Je to najjednoduchšia implementácia lokalizácie pomocou dead reckoningu. Samotné slovo odometria je zložené z dvoch gréckych slov hodos (cestovať, cesta) a metron (merať).

Základom odometrie je poznanie geometrického modelu robota. Podľa tohto sa roboty delia na druhy:

**Auto**

Do kategórie auto by sme zaradili všetkých robotov, ktorí sa nevedia otočiť namieste. Tento druh robotov sa musí otáčať pohybom dopredu alebo dozadu, ktorému pretchádza otočenie nápravy alebo kolesa do príslušného smeru. Táto kategória zahrňuje väčšinu dnešných automobilov a nazýva sa tiež aj Ackermanovo riadenie.

**Tank**

Roboti kategórie tank sa vyznačujú schopnosťou otáčať sa na mieste okolo svojej osy a možnosťou pohybu dopredu a dozadu. Ich základným rysom sú dve nezávislé kolieska alebo nápravy. Zmena orientácie pohybu závisí na rozdiely rýchlostí koliesok, preto sa zvyknú označovať ako diferenciálne riadené.

Robot sa pohne priamo dopredu, ak sa točia obe kolieska rovnakou rýchlosťou a rovnakým smerom. Ak sa kolieska točia rovnakou rýchlosťou, ale opačným smerom, robot sa točí na mieste, okolo svojej osy.

Ak sa točí každé koliesko inou rýchlosťou, ale rovnakým smerom, robot sa bude pohybovať po kružnici. Vzdialenosť stredu robota od stredu kružnice je daná vzťahom 1 uvedenom nižšie.

Zmenu uhlu v radiánoch po prejdení istého počtu krokov vypočítame vzťahom 2, ktorý je tiež uvedený nižšie. Z tohto vzťahu tiež vyplýva, že natočenie robota nezávisí na priebehu pohybu robota, závisí iba na celkovej delte vzdialeností prejdených pravým a ľavým kolieskom.

Pokial je rozdiel rýchlostí medzi nápravami nenulový, ale obe nápravy sa točia na rovnakú stranu, robot sa pohybuje po kružnici. Vzdialenosť stredu tejto kružnice je daná pomerom oboch rýchlostí: $R = ½ \cdot b \cdot (vL + vR) / (vL – vR)$, kde b je vzdialenosť kolies od seba (rozchod) a vL a vR sú rýchlosti.

Vzťah tri vyjadruje celkovú prejdenú trajektóriu stredom poháňanej osy.

Robotnačka patrí dokategórie tank.

Vzťah 1: $R = ½ \cdot b \cdot (vL + vR) / (vL – vR)$, b je rozkol koliesok robota, vL je rýchlosť ľavého kolieska a vR je rýchlosť pravého kolieska

Vzťah 2: $\theta = (dL – dR)/b$, b je roykol koliesok, dL je vzdialenosť prejdená ľavým kolieskom a dR je vzdialenosť prejdená pravým kolieskom

Vzťah 3: $d = (dL + dR)/2$, dL je vzdialenosť prejdená ľavým kolieskom a dR je vzdialenosť prejdená pravým kolieskom

**Pohyb do všetkých strán**

Táto kategória robotov je najviac zriedkavá, aj keď v poslednej dobe sa stáva populárnejšou v robotickom futbale. Roboti patriaci do tejto kategórie sa vyznačujú tým, že sa môžu pohnúť dopredu v akomkoľvek smere, bez ohľadu na to, do akého smeru sú otočený. Táto kategória sa označuje aj ako omnidrive alebo omnidirectional drive.

## 1.3 Open loop

Open loop je spôsob ovládania systému, pri ktorom systém vyhodnocuje reakciu pre každý vstup iba na základe momentálneho stavu systému a modelu systému.

Hlavnou charakteristikou open loop ovládača je, že nepoužíva žiadnu spätnú väzbu aby zistil, či systém splnil cieľ ktorý mu bol zadaný alebo nie. To znamená, že vôbec nekontroluje výstup procesu, ktorý ovláda. Takže pravý open loop ovládač nevie opraviť žiadnu chybu, ktorú spôsobí.

Open loop ovládanie je užitočné pre systémy, kde je vzťah medzi vstupom a výstupom ľahko vyjadriteľný, napríklad matematickým vzorcom. Napríklad pre určovanie voltáže elektrického motora vo vozidle, ktoré ťahá náklad známej hmotnosti tak, aby sa vozidlo pohybovalo zadanou rýchlosťou je pekný príklad, kedy by stačil open loop ovládač voltáže. Avšak ak by sme nevedeli hmotnosť nákladu, open loop ovládač by už nestačil.

Open loop ovládače sa používajú pri jednoduchých procesoch pre ich nízku cenu a jednoduchosť a hlavne pri systémoch, pri ktorých nie je odozva až tak dôležitá.

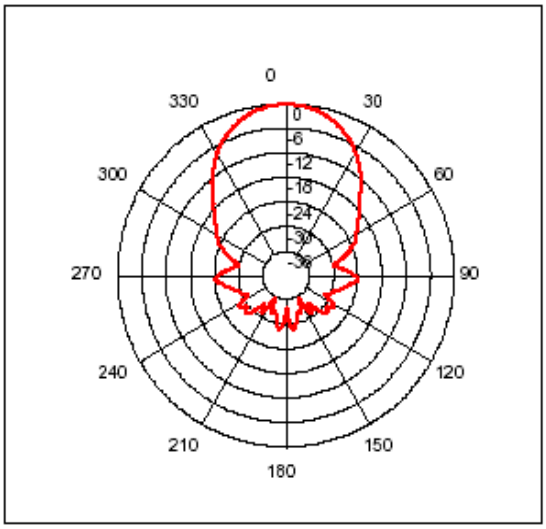
Ovládač, ktorý sa rozhoduje aj na základe spätnej väzby ovládaného procesu sa označuje ako close loop.

## 1.4 Ultrazvukový senzor SRF08

Ultrazvukové senzory sú založené na meraní doby medzi vyslaním akustického impulzu a prijatím odrazeného signálu od prekážky(ďalej echo). Pomocou rýchlosti zvuku(konštanty) a časovej delty týchto dvoch momentov vieme ľahko vypočítať vzdialenosť senzora od prekážky.

Obrázok 2. Ultrazvukový senzor SRF08.

Senzor SRF08 obsahuje dva meniče pracujúce na frekvencii 40 kHz a umožňuje meranie vzdialenosti od 3 cm do 6 m. Uhol jeho vyžarovania je 55°. Prednastavená doba merania je 65 ms, čo prevyšuje maximálnu merateľnú vzdialenosť takmer dvojnásobne. Senzor má dva módy merania a tými sú meranie vzdialeností a ANN(Artificial Neural Network).




Obrázok 3. Šírenie zvukových vĺn.

Odchýlka pri meraní môže byť až 3-4 cm. Väčšinou býva lepšia, ale nedá sa to zaručiť. Senzor meria echo tak, že sa snaží zachytiť odrazené vlnenie. Vracajúce sa vlnenie postupne silnie, až dosiahne vrchol a potom začne slabnúť. Odchýlka záleží od toho, ktorá vlna bude

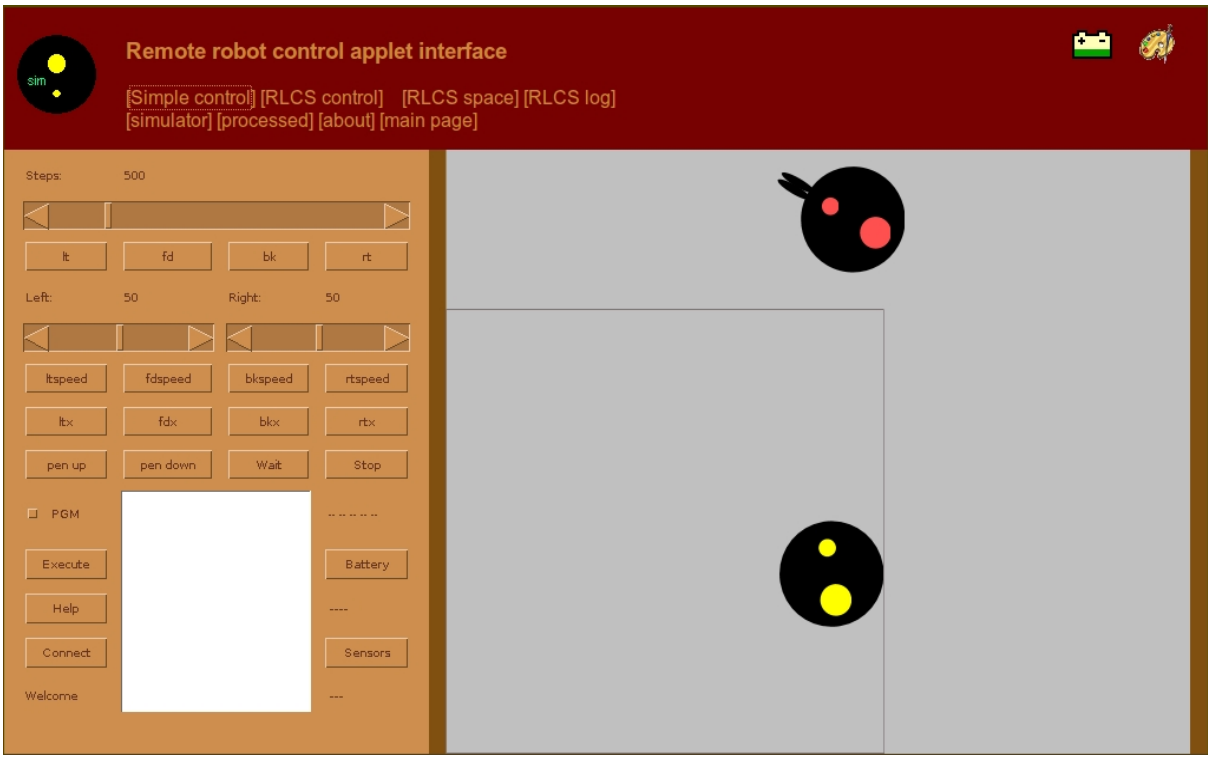
ako prvá dosť silná na to, aby ju senzor zachytil.

Ďalší faktor ovplyvňujúci presnosť je, ak sa signál neodráža od bodu ale napríklad od steny. Stena je väčšia ako bod, takže aj plocha, od ktorej sa bude signál odrážať, je väčšia, preto senzor zachytí oveľa viacej šumu, ako pri bode. Takže aj odchýlka môže byť väčšia, ak sa signál odráža od väčších objektov.

## 1.5 Robotické laboratórium so vzdialeným prístupom

„Robotické laboratórium sa nachádza na Fakulte elektrotechniky a informatiky Slovenskej technickej univerzity v Mlynskej doline v budove číslo D na siedmom poschodí. Vzniklo v spolupráci tejto fakulty spolu s Fakultou Matematiky, fyziky a informatiky Univerzity Komenského a v spolupráci s firmou MicroStep-MIS v roku 2005. Slúži na výskumno-vývojovú činnosť v oblasti robotiky. Účelom založenia laboratóriá je takisto aj zvýšiť záujem študentov venovať sa tejto problematike. V rámci tejto skupiny bol vytvorený robot Robotnačka, ktorý je určený na výučbu ako edukačná pomôcka programovateľná v programovacom prostredí Comenius Logo. Študenti navštevujúce toto laboratórium sa zúčastňujú rôznych robotických súťaží ako je napríklad Istrobot, najväčšia slovenská robotická súťaž aj so zahraničnou účasťou. Ďalej sú to súťaže First LEGO League, RoboCup, Robotchallenge, a ďalšie. “(Mišanič, 2010, strana 17)

Robotnačky sú prístupné širokej verejnosti cez internet na stránke laboratória - http://virtuallab.kar.elf.stuba.sk/. Na tejto stránke má verejnosť prístup aj do simulovaného laboratória s dvoma simulovanými robotmi s rovnakými funkciami ako majú reálne roboty.




Obrázok 4. Ukážka apletu na ovládanie robotnačky cez internet a simulovaného laboratória.

### 1.5.1 Ovládanie robotnačky cez Java aplikáciu

Pre ovládanie robota pomocou Java aplikácie cez internet je potrebné najprv si stiahnuť dva súbory: Robot.java a RemoteRobot.java a importnúť RemoteRobot do aplikácie. Oba sú voľne dostupné a dajú sa stiahnuť zo stránky www.robotika.sk.

```
import sk.robotics.robot.RemoteRobot;
```

Ukážka 1. Import RemoteRobot

V aplikácii treba vytvoriť stavovú premennú typu RemoteRobot a vytvoriť spojenie so serverom. Port sa udáva pri vytvorení novej inštancie RemoteRobot triedy. Virtuallab server počúva na portoch 3333, 3334 a 7123, pričom prvé dva porty patria robotom v simulovanom laboratóriu. IP adresa servera sa zadáva pri pripájaní k serveru za pomoci funkcie connect.

Virtuallab server má ip adresu  147.175.125.30. Funkcii connect sa zadáva aj meno a heslo užívateľa, ktorý sa chce k robotovi pripojiť. Ako ukážka dva naznačuje, connect vracia true ak sa robotovi podarí pripojiť, false inak, takže treba testovať, či sa to podarilo a iba vtedy možno pokračovať.

```
r = new RemoteRobot(null, port);
System.out.print("Connecting to server... ");
if(r.connect(virtuallab_ip, nick, password)) {
      System.out.println("connected.");
      r.close();
}else {
      System.out.println("connection refused.");
}
```

Ukážka 2. Vytvorenie RemoteRobot premennej a jej pripojenie k serveru

Príkazy na pohyb robota dopredu sa začínajú s fd, dozadu s bk, doľava s lt a doprava s rt. Príkaz na pohyb želaným smerom bez prípony znamená posunutie robota o zadaný počet krokov. Príkaz s príponou -speed sa zadáva s dvoma argumentami a tými sú rýchlosť pravého kolesa a rýchlosť ľavého kolesa. Prípona -x znamená navyše od -speed príkazu aj počet krokov, ktoré sa majú vykonať.

Pohybové príkazy robota navyše hádžu IOException, takže túto treba odchytiť a vysporiadať sa s ňou.

```
try {
      r.fd((short) 20);
} catch (IOException e) {
      spracujError(e);
}
```

Ukážka 3. Pohybovanie s robotom

Príkaz fd z ukážky 3 takisto ako všetky ostatné príkazy nie sú synchronizované. To znamená, že hneď po odoslaní príkazu robotovi program ide ďalej a nečaká kým robot zadaný

príkaz splní. Toto sa dá ošetriť pomocou funkcie completed.

```
while(!r.completed()) {
      Thread.sleep(30);
}
```

Ukážka 4. Synchronizovanie aplikácie s robotom

Po dokončení práce s robotom sa treba odhlásiť od servera, aby aplikácia zbytočne nezaberala port (a teda robota, na ktorého bola pripojená), lebo kým je k port obsadený, nik iný sa na neho nemôže pripojiť. Odhlasuje sa pomocou procedúry close.

```
r.close();
```

Ukážka 5. Odpojenie

## 1.6 Monte Carlo lokalizácia

Metóda Monte Carlo patrí do triedy výpočtových algoritmov, ktoré počítajú svoje výsledky na základe opakovaného náhodného vzorkovania. Takéto algoritmy sa často používajú vo fyzikálnych a matematických simuláciách, hlavne vtedy, keď je nerealizovateľné alebo nemožné spočítať výsledok deterministickým algoritmom.

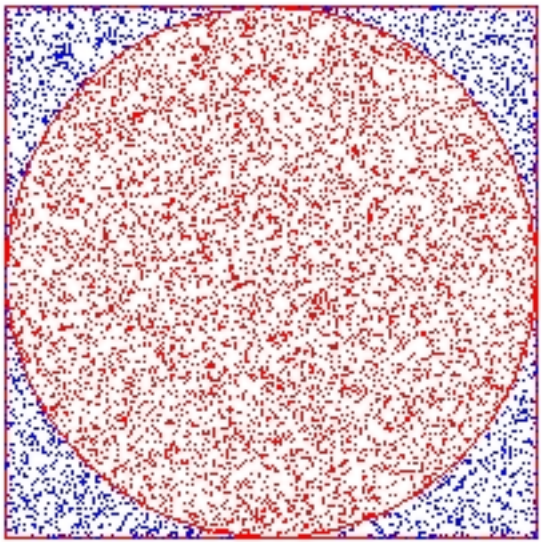
Každá metóda Monte Carlo je o trochu iná, používa iný prístup, ale všetky tieto prístupy majú tendenciu sledovať rovnaký vzor:

1. Definovanie domény možných vstupov.
2. Generovanie náhodných vstupov rovnomerne rozložených po celej doméne.
3. Vykonanie deterministického výpočtu na týchto náhodných vstupoch.
4. Zhrnutie jednotlivých výsledkov do požadovaného výstupu.

Jednoduchý príklad Monte Carlo metódy je odhad hodnoty π na základe štvorca a

doňho vpísaného kruhu a poznatku, že pomer obsahu kruhu k obsahu štvorcu je π/4. Postup je nasledovný:

1. Nakresli štvorec a do neho vpíš kružnicu.
2. Náhodne vyplň štvorec predmetmi(bodkami, ak je nakreslený) rovnakej veľkosti.
3. Spočítaj počet predmetov(bodiek) v kruhu a počet predmetov celkovo.
4. Pomer počtu predmetov v kruhu k počtu predmetov celkovo je približne rovnaký ako pomer obsahu kruhu k obsahu štvorca. A teda vynásobením spomínaného pomeru počtu predmetov číslom 4 dostaneš aproximáciu čísla π.

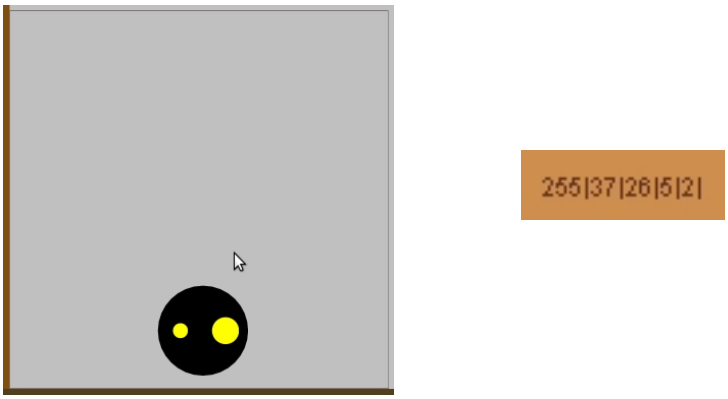


Obrázok 5. Náhodné rozloženie bodov pri odhade hodnoty π za pomoci Monte Carlo metódy.

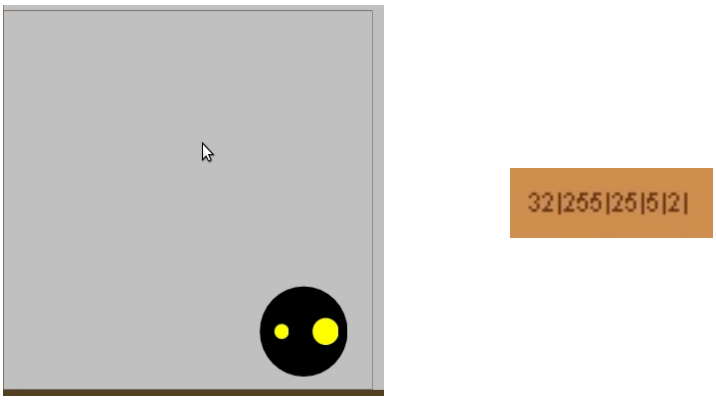
Monte Carlo Lokalizácia robota využíva predpoklad, že na určenie momentálnej polohy robota nepotrebujem vedieť všetky jeho niekdajšie polohy (x, y, uhol), akcie (pohyb robota) a vnemy (výstup zo senzorov), ale stačí mi poznať polohu robota pred poslednou akciou, túto akciu a vnem.

# 2 Špecifikácia problému

Najväčším problémom bude asi vysporiadať sa s veľkou mierou šumu senzorov spôsobenou hlavne vyžarovacím uhlom 55°. Nasledujúce štyri obrázky demonštrujú zašumenie vstupov. Robot na obrázku 7 je podľa senzorov 26 cm od prekážky pred ním. Po tom, ako som odmeral vzdialenosti som poslal robota dozadu približne o 27,5 cm a stav simulovaného laboratória je zachytený na obrázku 9, kedy sa robot podľa senzorov nachádza 25 cm od prekážky pred ním.




Obrázok 6. Poloha robota v simulovanom(vľavo) laboratóriu a senzorický vnem(vpravo).




Obrázok 7. Poloha robota v simulovanom laboratóriu(vľavo) a senzorický vnem(vpravo).

Ďalším problémom bude navrhnúť samotný algoritmus na hľadanie robota, ktorý musí

brať do úvahy už spomínaný šum, nepresnosť motorčekov, ktoré robota poháňajú a možnú prítomnosť iných robotov v miestnosti, tak aby vedel čo najpresnejšie určiť kde sa v miestnosti robot momentálne nachádza.

# 3 Návrh riešenia – robot hľadajúci stred miestnosti

## 3.1 Návrh aplikácie

Aplikácia hľadajúca robota bude konzolová a užívateľ ju iba spustí. Aplikácia bude po spustení sama ovládať robota s cieľom dostať ho do stredu obdĺžnikovej miestnosti v ktorej sa robot nachádza.

Robot pre potreby tejto aplikácie nepotrebuje vedieť rozmery miestnosti v ktorej sa nachádza a preto táto miestnosť nebude v aplikácii vôbec zahrnutá.

Aplikácia bude naprogramovaná v programovacom jazyku Java.

Aplikácia bude obsahovať jedinú triedu "DoStredu".

## 3.2 Trieda DoStredu

Trieda DoStredu zrealizuje spojenie aplikácie s robotom a následne ho začne posúvať a otáčať tak, aby po skončení aplikácie ostal robot v strede miestnosti v ktorej sa nachádza.

Idea algoritmu je zobrazená na ukážke zdrojového kódu číslo 6. Pokiaľ robot nie je v strede, tak sa otočí tak, aby sa hýbal po takej priamke, o ktorej si myslí, že od jej stredu je vzdialený najviacej. Po natočení sa pohne o toľko, aby sa nachádzal v strede tejto priamky. Potom skontroluje za pomoci senzorov vzdialenosti od pevných prekážok na 4 priamkach, z ktorých každá zviera so susednou priamkou uhol 45°. Ak zistí, že na ktorejkoľvek z priamok sa nenachádza v jej strede, postup sa opakuje. Ak zistí, že na všetkých priamkach sa nachádza v strede, aplikácia úspešne skončí.

```
private static void dostanRobotaDoStredu() throws IOException,
                                NumberFormatException, InterruptedException {
      while(nieJeVStrede()) {
            otoc();
            posun();
      }
}
```

Ukážka 6. Idea hľadania stredu

Očakávaný rozdiel pri robotovi s jedným senzorom oproti robotovi s viacerými senzormi je, že robot s jediným senzorom sa bude oveľa viacej točiť, aby zistil, či sa v strede nachádza, alebo nie. Vo všeobecnosti by malo platiť, že čím viacej senzorov, tým menej sa musí robot točiť pri zisťovaní, či je v strede (ak má robot 8 rovnomerne rozložených senzorov, t.j. orientácie senzorov zvierajú uhol 45° so svojimi susedmi, tak má pokrytých všetkých 8 strán a pri meraní by sa nemusel vôbec točiť).

Pre nepresné merania senzorov bude algoritmus, ktorý určuje, či je robot v strede alebo nie implementovaný s istou toleranciou odchýlky.

# 4 Implementácia – robot hľadajúci stred miestnosti

Konzolovú aplikáciu som implementoval podľa návrhu. Nenaskytli sa žiadne väčšie problémy. Aplikáciu som naprogramoval tak, aby robot používal iba jeden zo svojich senzorov. Vyskúšal som funkčnosť aplikácie a zistil som, že funguje. Potom som aplikáciu prepísal tak, aby robot používal všetkých päť senzorov. Aj v tomto prípade aplikácia fungovala tak, ako mala.

Po porovnaní správania robota pri prvej a druhej verzii aplikácie som usúdil, že môj predpoklad bol správny a teda keď mal robot k dispozícii viacej senzorov, nepotreboval sa tak často otáčať ako sa musel keď mal iba jeden senzor a tým skrátil čas potrebný na dokončenie úlohy.

# 5 Návrh riešenia – robot hľadajúci sa v obdĺžnikovej miestnosti

## 5.1 Návrh ovládania aplikácie

Aplikácia bude obsahovať menu, ovládací panel, plochu určenú na vykreslenie obdĺžnikovej miestnosti podľa zadaných údajov spolu s predpokladanou polohou robota a konzolu.

Novo pustená aplikácia bude mať prednastavené údaje servera na IP adresu robotického laboratória a port na simulovaného robota. Aj IP adresa aj port sa budú dať meniť pomocou dialógu v nastaveniach aplikácie. Rozmery miestnosti budú 0x0 cm a teda tieto treba pred pripojením nastaviť tiež za pomoci dialógu prístupného v nastaveniach aplikácie.

Po nastavení IP adresy(nepovinný krok) a rozmerov miestnosti v ktorej sa robot nachádza sa bude dať pripojiť na server a vytvoriť spojenie s robotom.

Po vytvorení spojenia s robotom bude užívateľ môcť ovládať robota pomocou tlačidiel dopredu, dozadu, doľava a doprava v spolupráci s textovým poľom, kde bude potrebné zadať požadovaný počet centimetrov ak sa jedná o príkaz pohybu, alebo stupňov ak sa jedná o príkaz otočenia. Ďalej bude možné zmerať vzdialenosti pomocou senzorov a užívateľ bude mať možnosť prikázať robotovi náhodný pohyb, t.j. aplikácia bude robota ovládať sama a to náhodne. Nebude však prikazovať robotovi aby šiel do kolízii, to znamená, ak senzor zachytí pred robotom pevnú prekážku, aplikácia robota otočí tak, aby do tejto prekážky robot nenarazil. Aplikácia bude tiež obsahovať tlačidlo, ktorým sa náhodný pohyb robota zastaví.

Aplikácia bude obsahovať aj možnosť prerušenia spojenia s robotom.

Súčasne za chodu aplikácie bude prebiehať kalkulácia polohy robota, ktorej priebežné výsledky sa budú zobrazovať na ploche a to tak, že aplikácia bude vykreslovať možné polohy robota aj s jeho orientáciou ako šípku s úsečkou a ukotvením. Je možné, že pri skorých fázach

hľadania sa bude vykreslovať iba bodka a pri neskorších iba jedna ukotvená šípka s úsečkou pre jednu možnú polohu robota a to pre zvýšenie prehľadnosti aplikácie.

Konzola bude užívateľa informovať o momentálnej činnosti aplikácie, to znamená o momentálne prebiehajúcom príkaze pre robota a o fáze MCL algoritmu.

## 5.2 Náhodný pohyb robota

Pre účely testovania a prezentovania aplikácie bude táto obsahovať funkciu náhodného pohybu robota. Aplikácia teda bude náhodne ovládať robota, avšak bude pri tom dodržiavať isté pravidlá.

1. Robot nemôže naraziť do žiadnej prekážky.
2. Po každom splnenom príkaze, ktorý bol buď pohybový (dopredu), alebo príkaz otočenia (doľava, doprava) si aplikácia vyžiada senzorový vnem robota.

A teda pri náhodnom pohybe robota sa nestane, že by šiel dozadu, lebo vzadu nemá senzory a mohlo by sa stať, že narazí.

## 5.3 Návrh algoritmu lokalizácie robota

Lokalizácia robota bude implementovaná pomocou štatistickej metódy Monte Carlo. Túto metódu som si vybral hlavne pre jej štatistickú povahu, vďaka ktorej sa dokáže lepšie vysporiadať s prítomným šumom či už pohybu, alebo senzorových vnemov. Daňou za túto schopnosť štatistických metód, lepšie sa vysporiadať so šumom, sú výpočtovo náročnejšie algoritmy, ale usúdil som, že aj napriek tomu bude lepšie použiť štatistickú metódu, ako deterministický algoritmus.

Samotná Monte Carlo metóda bude implementovaná v 4 fázach.

Prvá fáza spočíva vo vytvorení náhodných vzoriek, čo znamená vytvoriť dátovú štruktúru možných polôh a natočení robota. Pri vytvorení spojenia s robotom bude treba

zaplniť celú plochu možnými pozíciami robota (každý robot je aj náhodne natočený), nazvime ich vzorkami. Z povahy metódy Monte Carlo vyplýva, že čím viacej vzoriek vytvorím, tým presnejší môžem dostať výsledok, avšak pri prehnanom počte vzoriek by bola aplikácia spomalená. Povedzme, že vzoriek bude n a každej vzorke bude priradená pravdepodobnosť s akou táto vzorka reprezentuje reálneho robota v miestnosti. To je 1/n (pri inicializácii).

Druhá fáza sa realizuje vždy keď sa robot pohne alebo otočí. Pomocou odometrie vypočítam pre každú vzorku z dátovej štruktúry, kde by sa robot nachádzal a ako by bol natočený ak by bol reprezentovaný konkrétnou vzorkou. Všetky vzorky, ktoré sa budú nachádzať mimo plochy vylúčim ako potenciálnu reprezentáciu robota tak, že ich jednoducho vymažem z dátovej štruktúry. Pre vzorky, ktoré sa nachádzajú na ploche vygenerujem ďalšie vzorky, také, že ich vzdialenosť od pôvodnej vzorky bude malá a natočenie podobné natočeniu pôvodnej vzorky. Tento proces má simulovať nepresnosť robota a odráža open-loop konfiguráciu motorčekov, to znamená, že neviem zistiť, či robot prešiel naozaj toľko krokov, koľko mal prikázané prejsť.

Tretia fáza prebieha keď aplikácia dostane výsledky merania senzorov robota. Keďže poznám tvar a rozmery miestnosti, v ktorej sa robot nachádza viem ľahko pre každú vzorku určiť jej ideálny senzorový vnem. Reprezentácia robota si zapamätá tieto svoje senzorové hodnoty a potom ich budem porovnávať za pomoci heuristiky s reálnymi hodnotami, ktoré nameral môj robot. Heuristika mi vráti pravdepodobnosť s akou vzorka reprezentuje reálneho robota. Vzorke však priradím iba pseudo pravdepodobnosť s ktorou reprezentuje robota. Táto bude určená skutočnou pravdepodobnosťou, ale bude môcť nadobúdať zhora neobmedzené hodnoty. Takto bude číslo vyjadrovať nie momentálnu pravdepodobnosť, ale dlhodobú pravdepodobnosť, že vzorka reprezentuje reálneho robota. A teda ak robot aj niekedy nameria inú ako predpokladanú hodnotu, lebo senzory zachytia iného robota, vzorka, ktorá doteraz často mala vysokú pravdepodobnosť, aj keď v tomto prípade dostane menšiu, jej pseudo pravdepodobnosť bude stále vysoká.

Po niekoľkých opakovaniach fázy dva a tri (po inicializácii prvou fázou), by sa mal počet možných miest, kde sa robot nachádza rapídne zmenšiť, a mali by sa vytvoriť nazvime ich zhluky možných pozícii. Tieto budem detekovať heuristikou a vykreslovať ich vážený

priemer daný pravdepodobnosťou s akou konkrétna vzorka reprezentuje robota. To znamená, predstavme si, že takýto zhluk obsahuje m možností, kde sa nachádza robot. Potom moja aplikácia ukáže robota na mieste, ktoré bude priemerom všetkých bodov v zhluku s natočením, ktoré bude tiež priemerom všetkých natočení vzoriek v zhluku.

V štvrtej fáze algoritmu prebehne generovanie nových vzoriek na základe kopírovania a mierneho pozmenenia už existujúcich vzoriek. Vzorky zdedia pravdepodobnosť reprezentácie robota po svojom otcovi. Po vygenerovaní nových vzoriek, sa bude mazať istý počet vzoriek tak, aby sa populácia vzoriek pomaly zmenšovala, až zastane na zadanej hodnote. Štvrtá fáza má zaručiť, že Monte Carlo metóda bude pracovať s dostatočným počtom vzoriek a že počet vzoriek sa nebude meniť veľmi rýchlo.

## *5.4 Návrh tried*

### 5.4.1 Reprezentácia robota

Za účelom reprezentácie pozície robota vytvorím triedu AbstractRobot, ktorá si bude pamätať x-ovú a y-ovú súradnicu robota a keďže ma zaujíma aj natočenie robota bude si pamätať aj uhol natočenia robota vzhľadom na miestnosť v stupňoch.

AbstractRobot:

- private double x, y
- private int uhol

Konštruktor bude mat tri argumenty a to počiatočnú polohu x a y a uhol natočenia.

AbstractRobot sa musí vedieť aj posunúť dopredu a dozadu a otočiť sa doľava a doprava. Preto bude mať aj tieto metódy:

- public void chodDopredu(double pocet_cm)
- public void chodDozadu(double pocet_cm)
- public void otocSaDoprava(int pocet_stupnov)

- public void otocSaDolava(int pocet_stupnov)

AbstractRobot musí tiež vedieť vypočítať svoj teoretický výstup zo senzorov. Na tento účel bude slúžiť funkcia s dvoma argumentami a to so šírkou a výškou miestnosti, v ktorej sa robot nachádza (ak by miestnosť nebola obdĺžniková, argumentom tejto funkcie by bola vhodná reprezentácia tejto zložitejšej miestnosti):

- public int[] vzdialenosti(int sirka, int vyska)

a za účelom zistenia, či sa robot nachádza v miestnosti alebo nie bude slúžiť funkcia s dvoma argumentami (také isté ako u vzdialeností):

- public Boolean somVMiestnosti(int sirka, int vyska)

Pri implementovaní triedy AbstractRobot musím brať v úvahu, že priemer reálnej robotnačky je 21 cm. Keďže robot je v tejto triede reprezentovaný ako bod, pri počítaní výstupu senzorov a či je robot v miestnosti musím brať do úvahy polomer robota 10,5 cm.

### 5.4.2 Reprezentácia vzoriek

Na reprezentáciu vzorky vytvorím triedu Vzorka, ktorá bude obsahovať AbstractRobota a reálne číslo, reprezentujúce pseudo pravdepodobnosť, že práve tento robot reprezentuje skutočného robota.

Vzorka:

- AbstractRobot r
- double p

Vzorky si budem ukladať do LinkedList<Vzorka>, teda do predprogramovanej dátovej štruktúry.

### 5.4.3 Reprezentácia plochy

Na reprezentáciu plochy vytvorím triedu RoomCanvas. Táto trieda bude vykreslovať miestnosť a možné polohy robota. Tieto údaje budem vykreslovať v mierke, ktorú si táto trieda vypočíta podľa veľkosti voľnej plochy v aplikácii a zadanej veľkosti miestnosti.

Polohu robota, resp. možné polohy robota si bude táto trieda vyžiadavať od MCL objektu.

Na sprehľadnenie vykreslenej plochy budem používať heuristiku, ktorá určí, kde môžem vykresliť namiesto veľa bodiek reprezentujúcich možné polohy robota jednu ukotvenú šípku s úsečkou. Na sprehľadnenie priebehu MCL výpočtu nebudem vykreslovať všetky možné polohy robota, ale len tie, ktoré majú väčšiu pravdepodobnosť, že reprezentujú reálnu polohu robota v miestnosti. Teda napríklad po prvej fáze nevykreslím všetky náhodne vygenerované polohy robota, ale plochu nechám prázdnu. Lepšie to bude reprezentovať skutočnosť že zatiaľ vôbec neviem, kde sa vlastne robot nachádza.

### 5.4.4 Návrh Monte Carlo Lokalizácie

Pre potreby Monte Carlo algoritmu vytvorím triedu MCL, ktorá bude obsahovať zoznam vzoriek, s ktorými pracuje a rozmery miestnosti. Pri vytvorení novej inštancie triedy MCL si táto podľa zadaných rozmerov miestnosti inicializuje zoznam vzoriek podľa pravidiel špecifikovaných vyššie. Počet vzoriek bude ovplyvnený veľkosťou plochy miestnosti, v ktorej sa robot nachádza. Konštruktor Sa bude volať s dvoma argumentami a to so šírkou a výškou miestnosti.

MCL:

- LinkedList<Vzorka> vzorky
- int sirka, vyska

- public MCL(int sirka, int vyska)

Keďže spustenie druhej a tretej fázy algoritmu je podmienené aktivitou robota (príkazmi od užívateľa, alebo aplikácie, ak je spustený mód náhodného pohybu), tieto fázy sa budú sa vykonávať len vtedy, keď bude zadaný niektorý z príkazov, ktoré aplikácia poskytuje. Nazvime ho spúšťací. Spúšťacími príkazmi pre druhú fázu sú príkazy dopredu, dozadu, doľava alebo doprava. Tieto príkazy bude spracovávať procedúra akcia.

- public void akcia(string typ-akcie, int pocet_krokov)

Táto procedúra vykoná príslušnú akciu na každej zo vzoriek, zistí, ktoré vzorky nemôžu na základe ich polohy reprezentovať robota (tieto vzorky vymaže zo svojho zoznamu vzoriek) a vygeneruje niekoľko nových vzoriek, ktoré majú reprezentovať chybovosť a open-loop konfiguráciu robotových motorov.

Spúšťacími príkazmi tretej fázy je iba meranie vzdialenosti pomocou ultrazvukových senzorov. Pri automatickom náhodnom pohybe robota sa toto meranie volá po každom pohybovom príkaze. Vzdialenosti robota od pevných prekážok bude spracovávať procedúra vnem.

- public void vnem(int[] vzdialenosti)

Procedúra vnem porovná nameraní vzdialenosti robota so simulovanými vzdialenosťami vzoriek a podľa toho určí pravdepodobnosti s akými dané vzorky reprezentujú robota.

Trieda MCL bude tiež poskytovať zoznam možných polôh robotov triede RoomCanvas. Vždy keď si trieda RoomCanvas vyžiada nové polohy robota, tak MCL vytvorí nový LinkedList a skopíruje do neho iba tie vzorky, ktoré budú mať pravdepodobnosť väčšiu ako je minimálna.

# 6 Implementácia – robot hľadajúci sa v obdĺžnikovej miestnosti

## 6.1 Priebeh implementácie

Ako prvé som implementoval grafické rozhranie aplikácie, t.j. menu, ovládací panel, konzolu a kanvas, na ktorý sa vykreslujú najpravdepodobnejšie polohy robota. Hlavné okno bolo implementované triedou Okno, ktorá je podtriedov triedy javax.swing.JFrame. Trieda Okno tiež implementuje aj java.awt.event.ActionListener, a teda spracováva interakcie užívateľa s aplikáciou.

Po skontrolovaní funkčnosti ovládacích prvkov aplikácie som pristúpil k implementácii tried potrebných k funkčnosti Monte Carlo lokalizácie. To sú triedy AbstractRobot a Vzorka. Trieda Vzorka je implementovaná podľa návrhu, ale v triede AbstractRobot som bol nútený dodatočne robiť zmeny. Funkciu vzdialenosti som upravil tak, aby nevracala iba jednorozmerné pole s ideálnymi vzdialenosťami od prekážok, ale aby vracala dvojrozmerné pole o rozmeroch 2x5 s ideálnymi a predpokladanými vzdialenosťami od prekážok.

Ďalej som implementoval triedu RoomCanvas. Trieda RoomCanvas vo výslednej aplikácii iba vykresluje zoznam AbstractRobot-tov, ktorý dostane od triedy MCL. Ak sa majú vykresliť iba dvoja alebo štyria roboti, vykreslia sa zjednodušené modely robota, t.j. kružnica, s úsečkou, ktorá určuje natočenie robota. Ak sa má vykresliť iný počet robotov, vykreslia sa iba bodky.

Ako poslednú som implementoval triedu MCL.

## 6.2 Popis zmien oproti návrhu

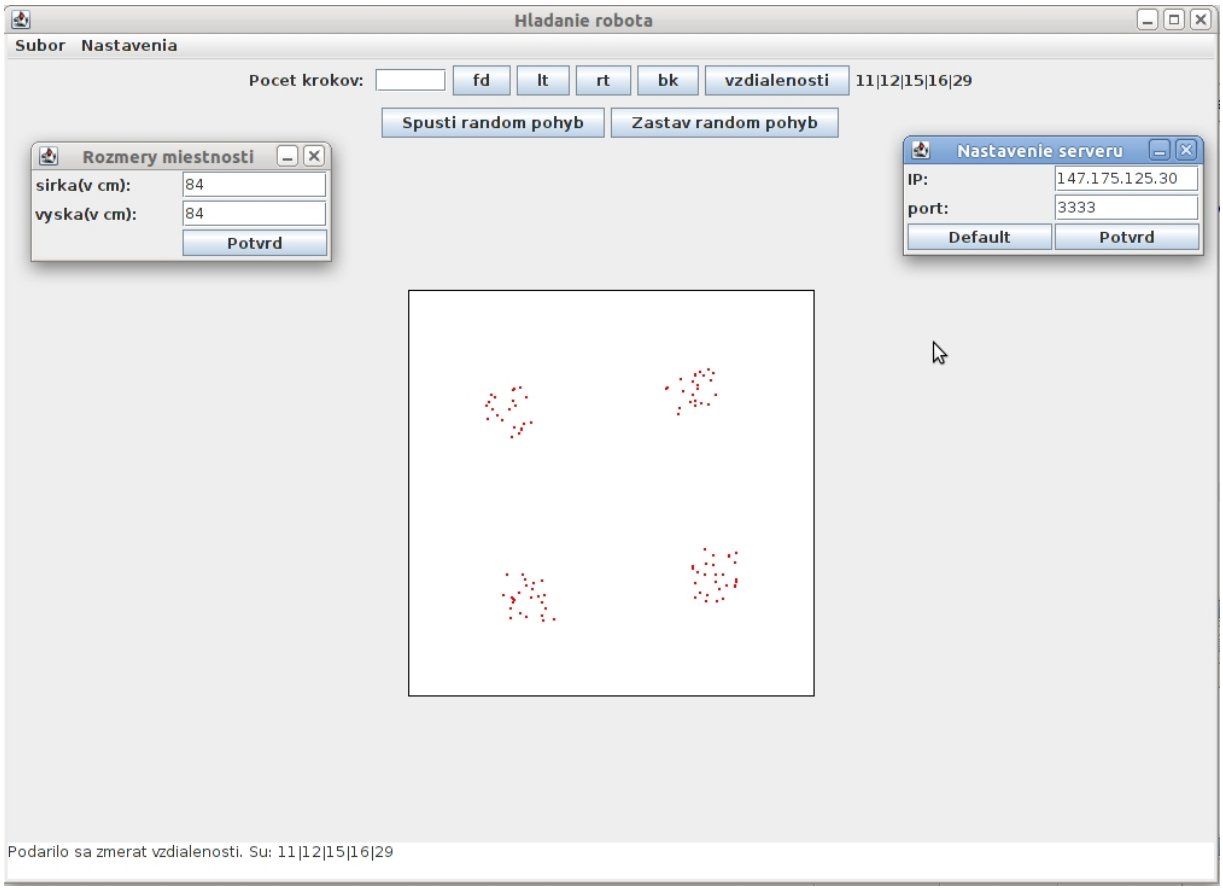
Pri implementovaní heuristiky odhadu pravdepodobnosti reprezentácie reálnej polohy

robota konkrétnou vzorkou v dátovej štruktúre som usúdil, že heuristika bude lepšia, ak v nej budem schopný zohľadniť okrem ideálnej vzdialenosti od prekážky aj odhad, ako mohol senzor reálneho robota zmerať vzdialenosť. Preto som zaviedol do aplikácie už spomínanú zmenu metódy vzdialenosti() AbstractRobota. Táto metóda teraz vracia dvojrozmerné pole 2x5. Prvý riadok obsahuje priame vzdialenosti senzorov AbstractRobota od prekážok a druhý obsahuje predpoklad, akú vzdialenosť by mohol namerať senzor reálneho robota. Tento predpoklad počítam na základe uhlu vyžarovania, ktorý je 55°. Keďže sa nachádzam v obdĺžnikovej miestnosti, tak vypočítam priamu vzdialenosť v smere okrajových priamok vyžarovania a vypočítam, odkiaľ by sa zvuk odrazil skôr a teda odkiaľ by sa aj skôr vrátil k senzoru. Inak povedané, snažím sa nájsť najbližší bod, od ktorého sa môže odraziť zvuková vlna. Vzdialenosti senzorov od týchto, pre každý senzor zvlášť vypočítaných, bodov tvoria druhý riadok.

Pri testovaní aplikácie som zistil, že MCL lokalizácia je výpočtovo veľmi náročná, hlavne pri prvých príkazoch, pre veľké množstvo inicializačných vzoriek. Keďže robot by nemal byť obmedzený pri pohybe čakaním na skončenie algoritmu, rozhodol som sa implementovať dve úplne nové triedy, ktoré neboli v návrhu. Sú to Lokalizacia, ktorá je podtriedou java.lang.Thread a Príkaz. Trieda Príkaz obsahuje záznam o príkaze pre robota. Pamätá si aký príkaz bol robotovi udelený a o koľko krokov sa mal pohnúť alebo aké vzdialenosti boli namerané. Trieda Lokalizácia obaluje prácu s MCL triedov. Obsahuje dátovú štruktúru, buffer, typu FIFO(First In First Out) príkazov udelených robotovi. Takto môže užívateľ ovládať robota bez toho, aby musel čakať kým skončia prepočty v triede MCL. Trieda Lokalizácia po každom kroku MCL nanovo prekreslí plochu, aby používateľ priebežne videl, kde sa robot pravdepodobne nachádza. Nevýhodou tejto zmeny však je, že na pomalšom počítači môže byť vykreslený obrázok s pravdepodobnými polohami robota neaktuálny.

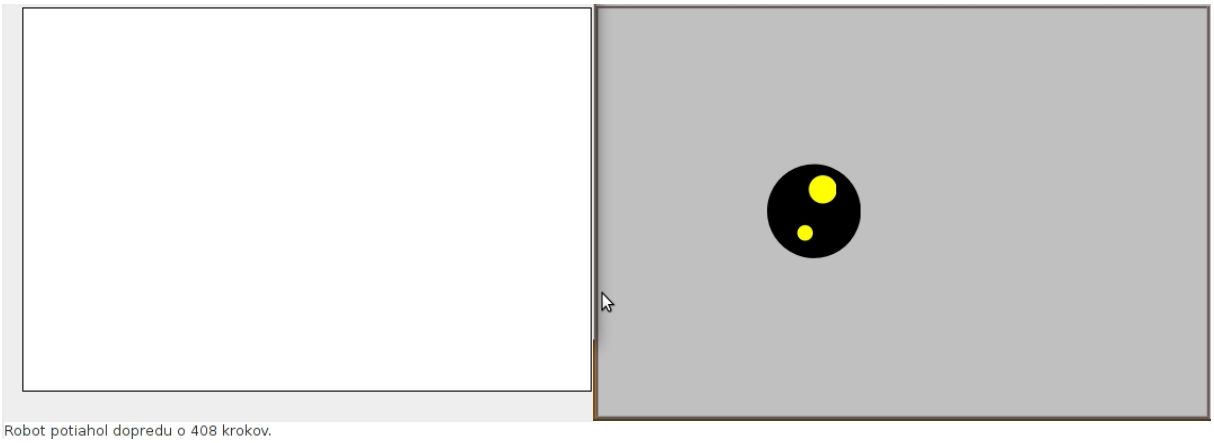
# 7 Zhodnotenie

Vo všeobecnosti som s výsledkom mojej práce spokojný, aj keď má svoje slabé miesta, ale myslím si, že cieľ sa mi splniť podarilo. V tejto kapitole sa nachádzajú obrázkové ukážky výstupu aplikácie. Ku každému výstupu mojej aplikácie prináleží obrázok virtuálneho laboratória, ktorý ukazuje reálnu polohu robota (okrem obrázku 8, ktorý iba ukazuje grafické rozhranie aplikácie). Pre symetrickosť miestností sa robot nikdy nenájde na presne jednom mieste. Vždy to budú minimálne dve, pri obdĺžnikovej miestnosti a štyri pri miestnosti štvorcovej.




Obrázok 8. Ukážka grafického rozhrania aplikácie.

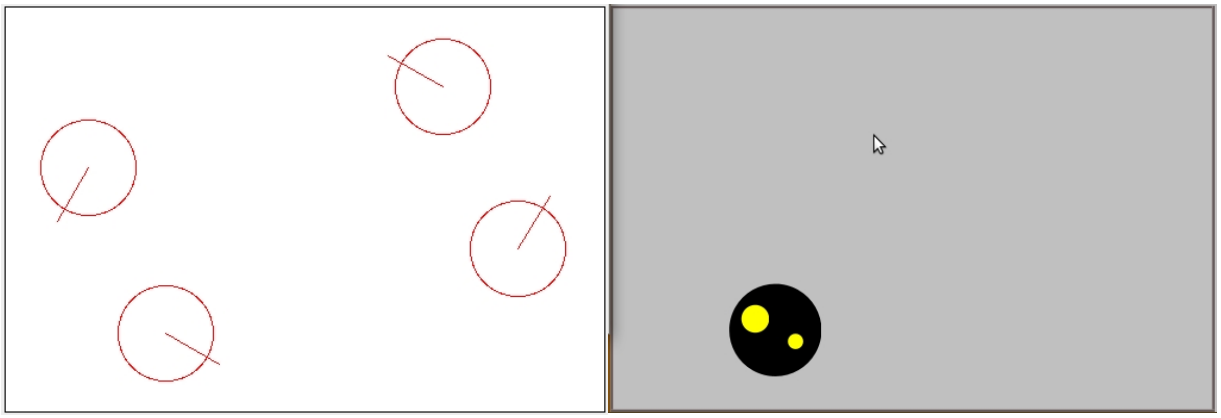
Na Obrázku 8 je ukážka grafického rozhrania pustenej aplikácie, ktorá sa snaží lokalizovať robota. Naľavo vidíme okno určené na zadávanie rozmerov miestnosti (dostupné z menu „Nastavenia“) a napravo je okno na zadávanie IP adresy servera a portu (tiež dostupné z menu „Nastavenia“). Vo vrchnej časti aplikácie vidíme dva rady ovládacích prvkov. Prvý rad slúži na ovládanie robota užívateľom a druhý na spustenie a zastavenia ovládania robota náhodne aplikáciou. V strednej časti vidíme vykreslený štvorec, ktorý symbolizuje miestnosť, v ktorej sa nachádza robot(veľkosť a pomer strán sa mení podľa zadaných rozmerov miestnosti). V štvorci vidíme červené bodky, ktoré symbolizujú najpravdepodobnejšie polohy robota. Keďže sa vykreslujú bodky, znamená to, že lokalozátor si ešte nie je úplne istý natočením robota. V spodnej časti obrázku vidíme konzolu, ktorá informuje o činnosti robota, t.j. o jeho pohybe alebo výsledkoch merania senzorov. Momentálne informuje užívateľa, že robot práve zmeral vzdialenosti od prekážok a tieto nám ukazuje.




Obrázok 9. Výstupy aplikácie pri prvých iteráciách MCL(vľavo) a simulované prostredie(vpravo)

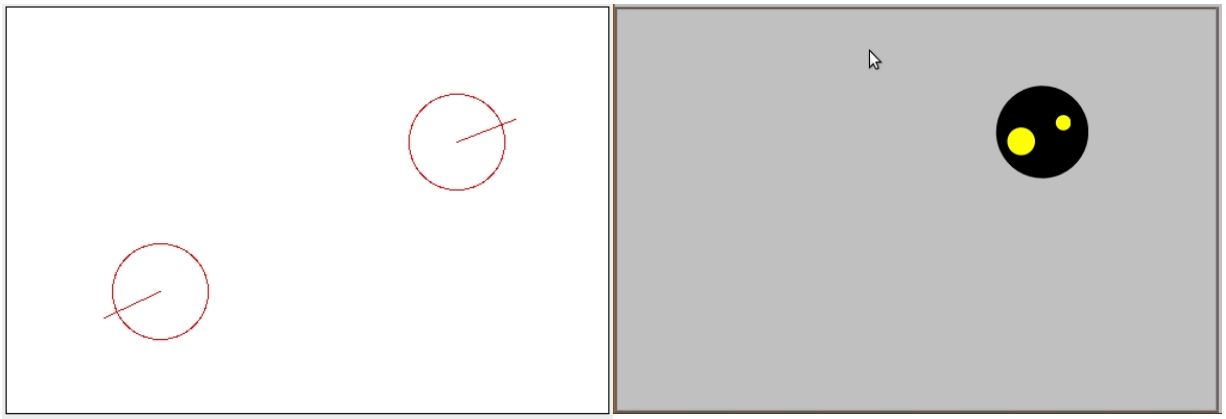
Na obrázku 9 vidíme správanie aplikácie, keď ešte nevie povedať kde sa robot nachádza. V skutočnosti majú už teraz vzorky, ktoré reprezentujú reálnu polohu robota oveľa väčšiu pravdepodobnostnú hodnotu, avšak keďže môžu byť miestnosti aj iný roboti, aplikácia si zatiaľ nemôže byť istá, či títo neskresľujú senzorické vnemy robota, preto prvé výsledky možnej polohy robota ešte neoznačí za správne, ale čaká, či sa jej v ďalšej iterácii Monte

Carlo algoritmu potvrdia.



Obrázok 10. Výstupy aplikácie po niekoľkých iteráciách MCL(vľavo) a simulované prostredie(vpravo)

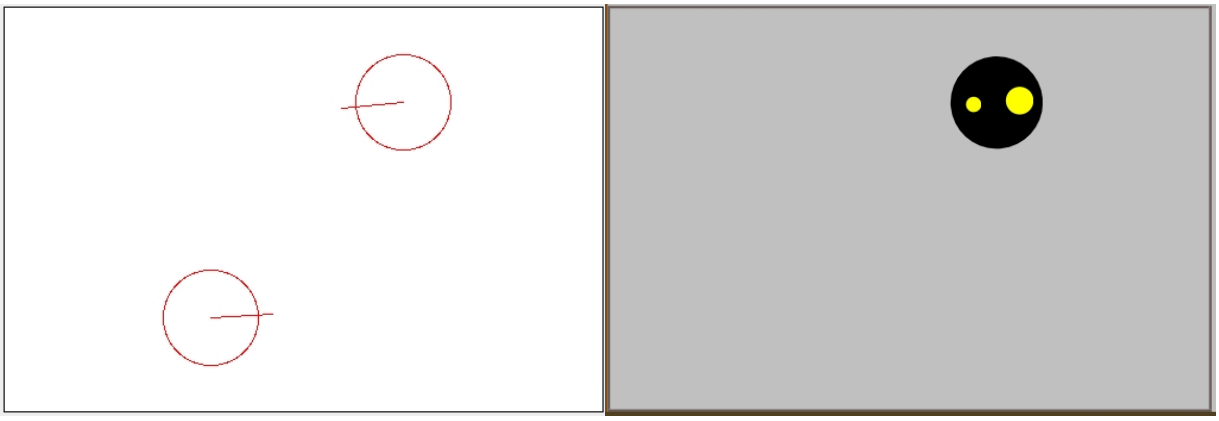
Obrázok 10 zobrazuje situáciu, kedy už prebehlo dostatočne veľa iterácii na to, aby aplikácia vedela, kde sa robot nachádza, berúc do úvahy symetrickosť miestnosti, ale ešte nemá dosť informácii na to, aby mohla povedať ktoré dva zo štyroch robotov nemôžu reprezentovať reálneho robota.



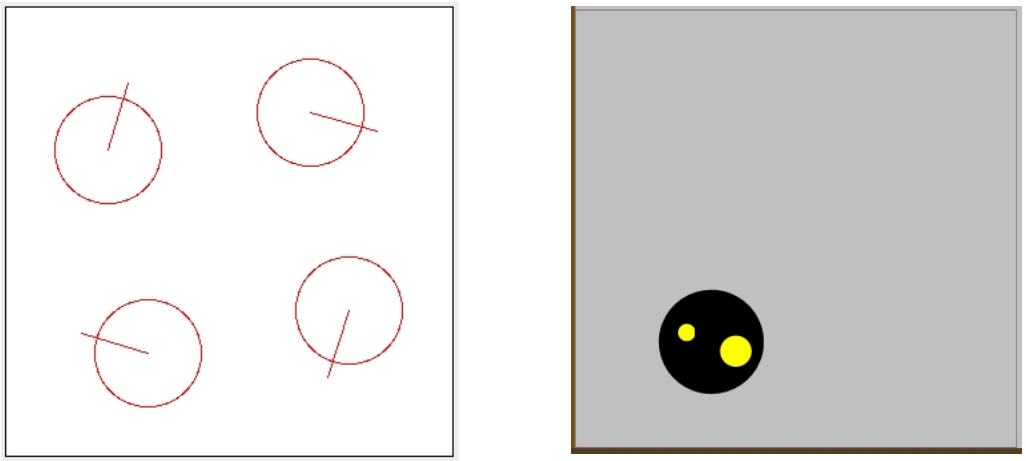
Obrázok 11. Výstupy aplikácie (vľavo) a simulované prostredie(vpravo)

Na obrázku 11 už vidíme konečnú fázu MCL algoritmu, kedy už považujeme robota

za nájdeného a teda poznáme súradnice v miestnosti a natočenie robota a k nemu symetrického robota podľa jednej z hlavných uhlopriečok miestnosti.



Obrázok 12. Ukážka výstupu aplikácie (vľavo) pre nájdeného robota v obdĺžnikovej miestnosti a pozícia robota vo virtuálnom laboratóriu (vpravo)



Obrázok 13. Ukážka výstupu aplikácie (vľavo) pre nájdeného robota v štvorcovej miestnosti a pozícia robota vo virtuálnom laboratóriu (vpravo)

# 8 Záver

Táto práca mala za úlohu preskúmať schopnosti robota lokalizovať sa v známej obdĺžnikovej miestnosti za pomoci ultrazvukových senzorov, navrhnúť algoritmus, ktorý by bol schopný robota lokalizovať, tento implementovať a zistiť, aký bol algoritmus úspešný.

Aplikácia ktorú som vytvoril je schopná náhodne ovládať robota, ale umožňuje používateľovi, aby robota ovládal on, ak chce. Po niekoľkých meraniach senzorov je aplikácia schopná lokalizovať robota a určiť jeho štyri možné polohy, ktoré sa pri pohybe v obdĺžnikovej miestnosti zredukujú na dve. Zo zadaných informácii sa nedá určiť jedna konkrétna poloha.

Myslím si, že úlohu sa mi podarilo uspokojivo splniť, to však neznamená, že sa na nej nedá už nič vylepšiť. Pokračovať by sa dalo optimalizáciou MCL pre vstupy z reálneho robota, nie simulovaného, viac spracovať možnosť výskytu iných robotov v miestnosti, nakoľko mi na túto časť práce nezostalo dostatočne veľa času, poprípade upraviť reprezentáciu miestnosti a triedu abstraktného robota tak, aby bola aplikácia schopná nájsť robota v akejkoľvek miestnosti s prekážkami, nie len v obdĺžnikovej.

# 9 Použitá literatúra


1. ĎURINA, D. - PETROVIČ, P. - BALOGH, R. 2003. *Robotnačka – The Drawing Robot* In *Acta Mechanica Slovaca.* vol. VII, no 1

2. WINKLER, Z. *Odometrie: modely kolových vozidel.* [online] Dostupné na internete: <http://robotika.cz/guide/odometry/cs> [pristupované 17.5.2011]

3. KUO, B. C. 1991. *Automatic Control Systems* (6th ed.). New Jersey: Prentice Hall.

4. NOVÁK, P. 2004. *Ultrazvukové sonary* In *Automa.* no 5

5. JUHÁS, M. 2009. *Senzorový systém pro mobilní robot. Bakalárska práca.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií

6. MIŠANIČ, D. 2010. *Lokalizácia za použitia senzorov na meranie vzdialenosti. Bakalárska práca.* Bratislava: Univerzita Komenského v Bratislave

7. KALOS, M. H. - WHITLOCK, P. A. 2008. *Monte Carlo Methods,* Berlin: Wiley-VCH