Univerzita Pardubice

Fakulta elektrotechniky a informatiky

# Lokalizace mobilního robotu ve zmapovaném prostředí

Bc. Milan Kacálek

Diplomová práce

2013

**Univerzita Pardubice**
**Fakulta elektrotechniky a informatiky**
Akademický rok: **2012/2013**

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

| | |
|---|---|
| Jméno a příjmení: | **Bc. Milan Kacálek** |
| Osobní číslo: | **I11385** |
| Studijní program: | **N2646 Informační technologie** |
| Studijní obor: | **Informační technologie** |
| Název tématu: | **Lokalizace mobilního robotu ve zmapovaném prostředí** |
| Zadávající katedra: | **Katedra softwarových technologií** |

Zásady pro vypracování:

Cílem diplomové práce je sestavit ze stavebnice Lego Mindstorms Education originální mobilní robot schopný autonomního pohybu ve členitém prostředí (prostředí je tvořeno průchody a nepřekonatelnými překážkami). Úkolem robotu je lokalizovat svou polohu na základě znalosti mapy okolí a informací z čidel (ultrazvukové čidlo vzdálenosti, dotyková čidla, kompas, ...). Komunikace mezi robotem a operátorskou stanicí bude probíhat pomocí bluetooth.
Rešeršní část práce bude obsahovat rozbor problému, definici učebnicových příkladů (globální lokalizace, unesený robot, ...), seznam a stručný popis možných přístupů k řešení (částicový filtr, Kalmanův estimátor, ...) a především detailní teoretický rozbor zvoleného způsobu řešení upravený pro konkrétní řešení zadaného problému.
Praktická část práce sestává z návrhu originálního robotu ve tvaru optimálním vzhledem k řešení problému, jeho oživení, zajištění komunikace mezi robotem a operátorem a zejména implementace vybraného upraveného algoritmu pro lokalizaci robotu.
Operační systém robotu bude Lejos, operační systém operátorské stanice bude Android, veškerý programový kód bude psán v Javě, popř. její modifikaci NXJ. Práce bude obsahovat přehlednou uživatelskou příručku pro práci se všemi vytvořenými aplikacemi.
Práce bude vypracována podle interních pokynů Fakulty elektrotechniky a informatiky v souladu s normami ČSN ISO 7144 a ČSN ISO 690.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

**[1] BORENSTEIN, J; EVERETT, H. R.; FENG, L. Where Am I? Sensors and Methods for Mobile Robot Positioning. University of Michigan, 1996.**
**[2] THUN, S. Particle Filters in Robotics. In Proceedings of Uncertainty in AI, 2002.**
**[3] Lejos. [online]. [cit. 2012-09-21]. Dostupné z: http://lejos.sourceforge.net/.**

Vedoucí diplomové práce: **Ing. Petr Doležel, Ph.D.**
Katedra řízení procesů

Datum zadání diplomové práce: **31. října 2012**
Termín odevzdání diplomové práce: **17. května 2013**

Univerzita Pardubice Česká republika

L.S.

prof. Ing. Simeon Karamazov, Dr.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2012

**Prohlášení autora**

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 15. 8. 2013

Milan Kacálek

**Poděkování**

Děkuji svému vedoucímu diplomové práce Ing. Peru Doleželovi Ph.D. za jeho trpělivost, odborné i neodborné rady, konzultace a hlavně za jeho čas. Poděkování také patří rodině a přátelům, kteří mě ve studiu podporovali.

Tato diplomová práce vznikla v rámci řešení projektu „Podpora stáží a odborných aktivit při inovaci oblasti terciárního vzdělávání na DFJP a FEI Univerzity Pardubice, reg. č.: CZ.1.07/2.4.00/17.0107“, v týmu Moderní metody řízení – vývoj a aplikace metod prediktivního řízení s využitím umělé inteligence.

**Anotace**

Diplomová práce je věnovaná demonstraci fungování lokalizace robota ve zmapovaném terénu pomocí lokalizační metody Částicový filtr. Robot je sestaven ze stavebnice LEGO Mindstorms Education, který po umístění do zmapovaného terénu určí svou polohu.

V první polovině se práce zabývá teoretickým fungováním lokalizačních technik, které se používají pro lokalizaci v mobilní robotice. Druhá polovina práce se zabývá sestavením a oživením robota. Na takto oživeném robotovi je poté popsán způsob použití Částicového filtru.

**Klíčová slova**

Java, LeJOS, NXT, LEGO Mindstorms Education, Částicový filtr, lokalizace

**Title**

Localization of a mobile robot in the mapped area

**Annotation**

The thesis is focused on the demonstration of robot's localization in mapped terrain using the method of Particle Filtering. Robot is constructed of LEGO Mindstorms Education kit and is able to determine its position when is placed into mapped field.

In the first part of the thesis, there are described the techniques of positioning which are used for the localization in mobile robotics. The second part describes the construction plan of the robot and robot control. The robot is then used for the localization using Particle Filtering.

**Keywords**

Java, LeJOS, NXT, LEGO Mindstorms Education, Particle filter, localization

# Obsah

# Seznam obrázků a tabulek

# Úvod

Robotika již delší dobu proniká z vědeckých laboratoří, průmyslových linek a sci-fi příběhů do běžného života. S tím, jak se možnosti robotických pomocníků stále rozšiřují, již není robot umístěn na jenom místě, ale stále častěji se setkáváme s mobilními roboty. Mezi takové roboty se může například řadit vysavač, který sám uklidí celý byt, nebo jeho obdoba v podobě automatické sekačky na trávu. Ale asi největším úspěchem poslední doby je robotické auto, které je schopné samostatné jízdy v běžném městském provozu. Úspěchy těchto robotů by však nebylo možné bez možnosti lokalizovat svou polohu v prostoru, ve kterém se pohybují.

Tato práce nepojednává o lokalizaci, která je běžně známá jako GPS (Global Positioning System), která pro svou nepřesnost v řádech několika metrů není vhodná. Práce popisuje metody lokalizace na základě senzorického měření prostředí, ve kterém se lokalizovaný objekt pohybuje. Právě tyto metody umožnily rozvoj autonomně se pohybujících robotů.

# 1 Cíl diplomové práce

Cílem diplomové práce je sestavit ze stavebnice Lego Mindstorms Education originální mobilní robot, schopný pohybu ve členitém prostředí. Úkolem robota je lokalizovat svou polohu na základě znalosti mapy okolí a informací ze senzorů.

V teoretické části:

a) Obecně vysvětlit a rozdělení metod lokalizace v mobilní robotice.
b) Vyjmenovat a stručně popsat možné přístupy k lokalizaci.
c) Detailně popsat vybranou metodu lokalizace, která je použitá v praktické části práce.

V praktické části:

a) Sestavit robota.
b) Připravit vývojářské prostředí.
c) Oživit robota.
d) Realizovat vybranou metodu lokalizace.

# 2 Metody lokalizace v mobilní robotice

Lokalizace robota představuje jeden ze základních problémů v mobilní robotice. Lokalizace v mobilní robotice se zabývá přesnou lokalizací mobilního robota. Požadavky na přesnost lokalizačních metod jsou dva až deset centimetrů.

Lokalizace je nutná pro autonomní fungování mobilních robotů. Je důležitá pro řešení klíčových úloh mobilního robota, jako plánování cílů nebo navigace v prostoru.

## 2.1 Rozdělení lokalizačních metod

Lokalizační metody se rozdělují do skupin podle svého určení, schopností, nutných předpokladů ke svému fungování.

### 2.1.1 Relativní a absolutní metoda lokalizace

Rozdělení lokalizace na relativní a absolutní je založené na charakteru určování polohy.

- Relativní lokalizace je založená na odhadu aktuální polohy ze změny polohy robota, typicky posunem a rotací v rovině vůči jeho předcházející poloze. Poloha je potom určena, vůči první nebo některé známé poloze v prostoru, skládáním jednotlivých změřených změn. Protože v každém měření vzniká jistá nepřesnost a tato měření se na sebe řetězí, dochází ke kumulaci těchto chyb. To znamená, že chyba nepřesnosti s ujetou vzdálenosti značně stoupá a proto je tato lokalizace vhodná pouze pro krátkodobý odhad.
- Absolutní lokalizace umožňuje odhad polohy mobilního robota v prostoru, bez nutnosti znát některou z předchozích pozic, nebo způsobu, jak bylo pozice dosaženo. Díky tomu není absolutní lokalizace zatížená akumulujícími se chybami, jako tomu je u relativní lokalizace. Ale ve srovnání s relativní lokalizací je absolutní lokalizace výpočetně náročnější a může být omezena pouze na určitou část prostoru, ve kterém se robot pohybuje.

V praxi se tyto dvě metody lokalizace kombinují. Mobilní robot používá jednu techniku relativní lokalizace a jednu techniku absolutní lokalizace.

### 2.1.2 Globální a lokální techniky lokalizace

Rozdělení na globální a lokální techniky se dělí podle typu problému, který lokalizační techniky řeší a zda je nutné znát počáteční polohu robota. Toto rozdělení je dané jak senzorickou výbavou robota, tak způsobem zpracování senzorických dat.

- Pro lokální techniky je důležitá znalost počáteční pozice robota v okamžiku zahájení lokalizace. Techniky poté za běhu udržují odhad aktuální polohy robota v závislosti na počáteční poloze a snaží se eliminovat jednotlivé chyby měření. Pokud selže lokalizace a odhad se příliš odchýlí od skutečné polohy robota, vznikne chyba, která již pomocí lokálních technik nelze opravit.
- Globální techniky lokalizace nejsou závislé na znalosti počáteční nebo předcházející pozice robota v prostoru. Globální techniky lokalizace můžeme rozdělit na dvě podkategorie, které řeší různé problémy:
    - Problém probuzeného robota (wake-up robot problem), řeší schopnost určit polohu bez znalosti polohy při jeho probuzení. Jediným rozdílem od následující metody je ten, že robot si je vědom, že byl právě probuzen na neznámém místě.
    - Problém uneseného robota (kidnapped robot problem), řeší schopnost určit polohu robota po jeho unesení během jeho provozu. Unesením je myšleno přemístění robota z jednoho místa na jiné, bez toho, aby na to byl robot nějak upozorněn.

### 2.1.3 Pasivní a aktivní lokalizace

Podle toho zda lokalizace může přímo ovlivňovat řízení robota, rozdělujeme lokalizaci na pasivní a aktivní.

- Pasivní lokalizace udržuje pouze odhadovanou polohu robota na základě senzorických dat. Nedovoluje lokalizačnímu systému zasahovat do řízení robota.

- Aktivní lokalizace dovoluje převzít částečnou, nebo i plnou kontrolu nad řízením. Lokalizační systém toho to zpravidla využívá, jen když selže pasivní lokalizace, protože při aktivní lokalizaci zpravidla robot není schopen provádět jinou činnost. Aktivní lokalizace se rozděluje na dvě podkategorie:
    - Aktivní snímání, které dovoluje lokalizačnímu systému převzít kontrolu nad senzory a například určit kterým směrem se mají senzory nasměrovat.
    - Aktivní navigace umožňuje lokalizačnímu systému převzít kontrolu nad navigací celého robota a určovat jeho pohyb.

### 2.1.4 Lokalizace ve statickém a dynamickém prostředí

Podle neměnnosti prostředí a možnosti ovlivnění měření, ve kterém se robot pohybuje, se rozděluje na statické a dynamické prostředí.

- Statické prostředí je neměnné. V ideálním statickém prostředí je vše neměnné, nejen že se v prostředí nepohybuje nic jiného než robot, ale nemění se ani okolní prostředí tak, aby ovlivnilo senzorické měření.
- Dynamické prostředí mění svoje vlastnosti v čase. Dynamické prostředí může obsahovat jiné pohyblivé předměty než robota a změny, které mohou ovlivnit měření.

# 3 Pravděpodobnostní lokalizace

V předchozí kapitole jsme rozdělili lokalizační techniky, představili jejich klady a zápory a vysvětlili si, že žádná z lokalizačních technik není dokonalá sama o sobě. Relativní techniky lokalizace trpí chybou rostoucí s ujetou vzdáleností, nebo časem a jsou proto použitelné jen pro krátkodobý odhad pozice. Kdežto absolutní techniky kumulující se chybou netrpí, ale jejich přesnost a spolehlivost je také omezená. Nejlepším řešením se proto jeví data z relativních a absolutních technik vhodným způsobem kombinovat a při vědomí silných i slabých stránek těchto metod dosáhnout spolehlivou, robustní a přesnou lokalizaci.

Pravděpodobností lokalizace se zabývá právě kombinací lokalizačních technik. Vysvětluje, jak zpracovávat nepřesné informace z více senzorů a reprezentovat na výstupu pozici robota v prostoru. Podobné zpracování dat se v robotice používá i pro jiné potřeby, například pro rozpoznávání objektu, nebo mapování prostoru.

Pravděpodobnostní přístup nám umožňuje pracovat s nepřesnými vstupními informacemi ze senzorů a poté reprezentovat ne zcela jednoznačnou pozici robota v prostoru. Podle tohoto přístupu představují veličiny měřené pomocí senzorů i skutečná pozice robota v prostoru náhodnou veličinu. Z problému lokalizace robota v prostoru se pak stává problém nalezení takové hustoty pravděpodobnosti, která co nejlépe odpovídá hustotě pravděpodobnosti skutečného výskytu robota. Pravděpodobností způsob lokalizace se pro řešení úloh zpracování dat z více senzorů osvědčil a je považován za standardní metodu řešení těchto úloh.

Kromě pravděpodobnostního přístupu existují i další metody pro zpracování a kombinaci dat z více senzorů. Například fuzzy logika nebo intervalový počet. Tyto metody se ovšem obecně pro zpracování dat z více senzorů nepoužívají.

## 3.1 Princip pravděpodobnostního lokalizace a filtrování

Princip pravděpodobnostní lokalizace spočívá v odhadu pozice robota v prostoru jak podle všech dostupných informací ze senzorů, tak i podle předcházejících odhadů jeho polohy. Poloha robota není vyjádřená jedním bodem (místo kde se robot nachází v daném prostoru), ale jde o hustotu pravděpodobnosti, tedy jde o funkci, která každé možné pozici v prostoru určuje pravděpodobnost výskytu robota právě v této pozici.

Pro lokalizaci robota, který jezdí po zemi, se pro vyjádření polohy většinou používá tří informací. A to pozice v dvourozměrné rovině $x$, $y$ a orientace robota $\theta$.

$$l = (x, y, \theta) \tag{3.1.1}$$

Pravděpodobnostní lokalizace je instancí obecnější metody, totiž pravděpodobnostního odhadu v čase se měnícího stavu na základě nepřesných, chybami a šuměním zatížených informací o tomto stavu. Takové odhadování se nazývá filtrování. V tomto případě bude odhadovaným stavem poloha robota a informace o ní data z relativních a absolutních měření. Filtrování se také používá v celé řadě dalších aplikací použité v robotice, ale nejen v ní.

### 3.1.1 Základní pojmy z teorie pravděpodobnosti

Pro pochopení pravděpodobnostní lokalizace je potřeba znalost některých základních pojmů z teorie pravděpodobnosti. Proto budou v této kapitole v krátkosti některé uvedeny.

*Podmíněná pravděpodobnost* $P(A|B)$ je definovaná jako pravděpodobnost výskytu jevu A za předpokladu, že nastal náhodný jev B. Zároveň předpokládáme, že výskyt náhodného jevu B není nulový.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{3.1.2}$$

*Nezávislé* náhodné jevy A a B nazýváme tehdy, jestliže $P(A \cap B) = P(A) \cdot P(B)$. Ekvivalentem této podmínky je rovnost $P(A|B) = P(A)$ nebo $P(B|A) = P(B)$.

*Věta o úplné pravděpodobnosti.* Jestliže náhodné jevy $A_1, A_2, \dots, A_n$ tvoří úplný systém jevů (to znamená, že jsou jevy po dvou neslučitelné a sjednocení všech jevů je jistý jev), pak pravděpodobnost libovolného jevu B lze určit pomocí následujícího vzorce.

$$P(B) = \sum_{i=1}^{n} P(A_i) \cdot P(B|A_i) \tag{3.1.3}$$

*Bayesova věta.* Mějme dva náhodné jevy A a B s pravděpodobností $P(A)$ a $P(B)$, přičemž platí, že pravděpodobnost jevu B není nulová. Potom platí, že podmíněnou pravděpodobnost $P(A|B)$ jevu A za předpokladu, že nastal jev B.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{3.1.4}$$

Pro jev podmíněný více jevy poté platí následující varianta vzorce:

$$P(A|B \cap C) = \frac{P(B|A \cap C) \cdot P(A|C)}{P(B|C)} \tag{3.1.5}$$

## 3.2 Principy odhadu polohy

Při použití pravděpodobnostní lokalizace se odhadovaná pozice robota v prostoru definuje hustotou pravděpodobnosti. Tato hustota určuje s jakou pravděpodobností je robot na dané pozici v prostoru umístěn. Součet pravděpodobností všech odhadů polohy je vždy roven

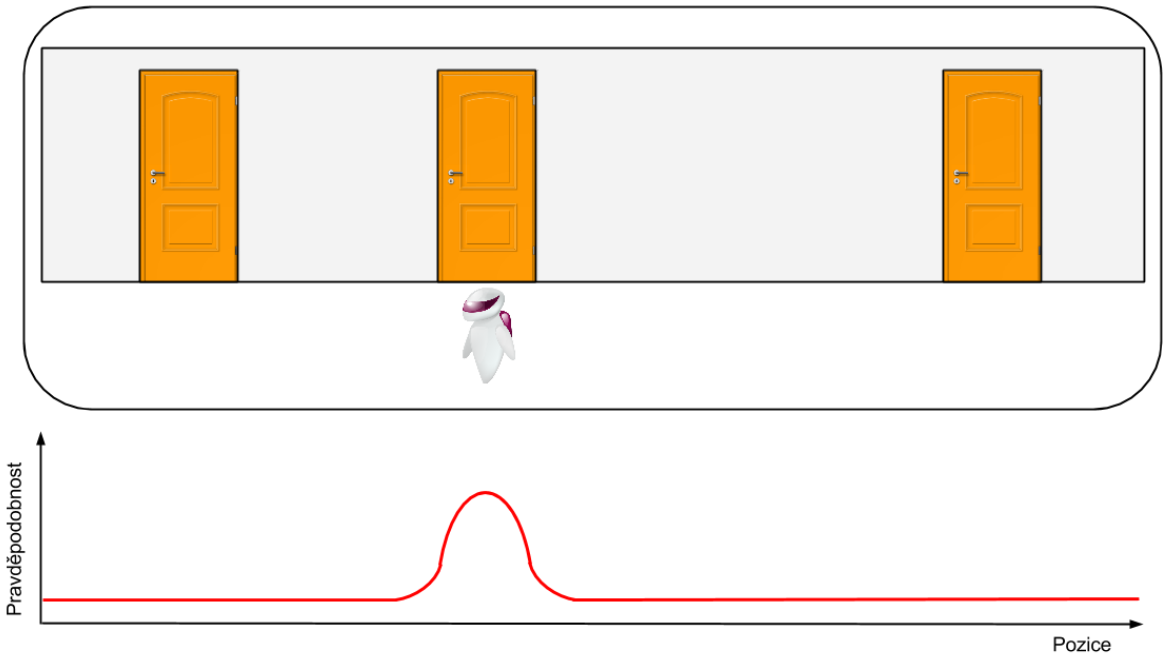
jedné. Tento odhad značíme $Bel(l)$ (z anglického slova *Belief*) a udává hustotu pravděpodobnosti výskytu na pozici $l$ [5].

Cílem pravděpodobnostní lokalizace je udržovat odhad pozice $Bel(l)$ tak, aby co nejlépe odpovídala skutečnému rozdělení pravděpodobnosti výskytu robota. V ideálním případě má funkce $Bel(l)$ jediný vrchol, který se blíží k hodnotě jedna a mimo něj se hodnota funkce blíží nule.

Pro určování pozice robota v prostoru, pravděpodobnostní lokalizace používá několik odhadů polohy. Jednotlivé odhady budou popsané na následujících řádcích.

### 3.2.1 Počáteční odhad polohy

Techniky lokalizace požadují výchozí hodnotu odhadu polohy v okamžiku zapnutí robota. Tento odhad je požadován ještě před tím, než robot získal jakékoliv informace ze svých senzorů. Pro správné fungování lokalizačních technik je počáteční odhad polohy nutné znát.

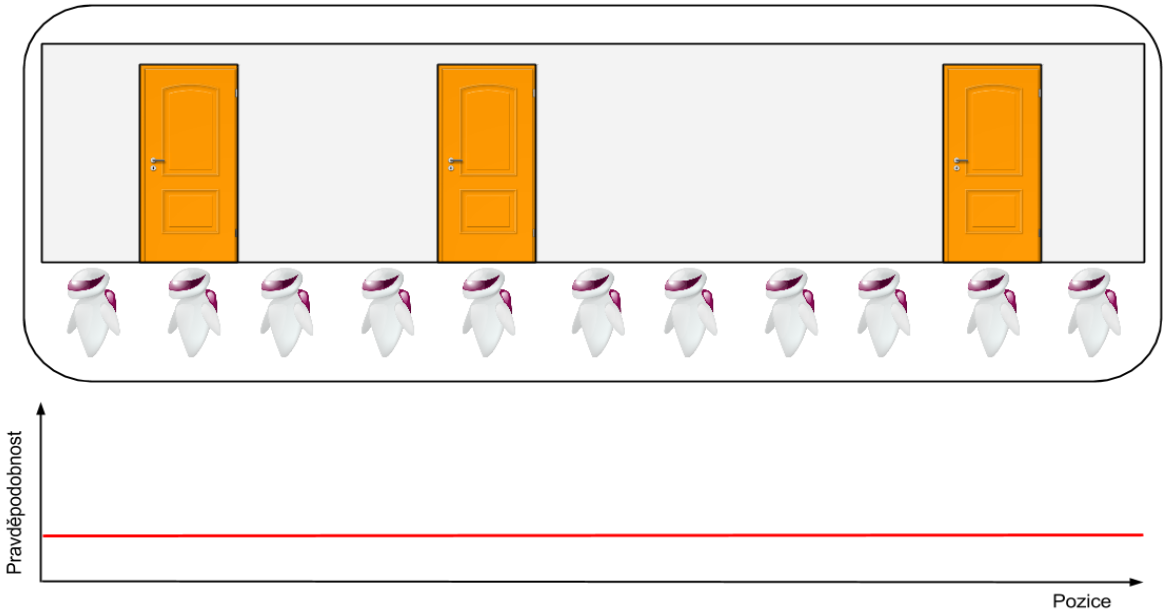



**Obr. 3.2.1 – Inicializace odhadu polohy pro jednu konkrétní polohu.**

Je-li robotu známá jedna konkrétní počáteční poloha, bude mít počáteční odhad $Bel(l_0)$ jeden vrchol v místě skutečné pozice robota při jeho zapnutí. Hodnoty mimo tento vrchol se budou blížit nule. V případě znalosti jedné výchozí polohy, se inicializace počátečního odhadu polohy provádí hodnotami úzkého normálního rozdělení. Znalost jedné konkrétní počáteční polohy je vyžadována v případě, že robot je schopen používat pouze lokální lokalizační techniky. Příklad inicializace hustoty pravděpodobnosti pro jednu výchozí

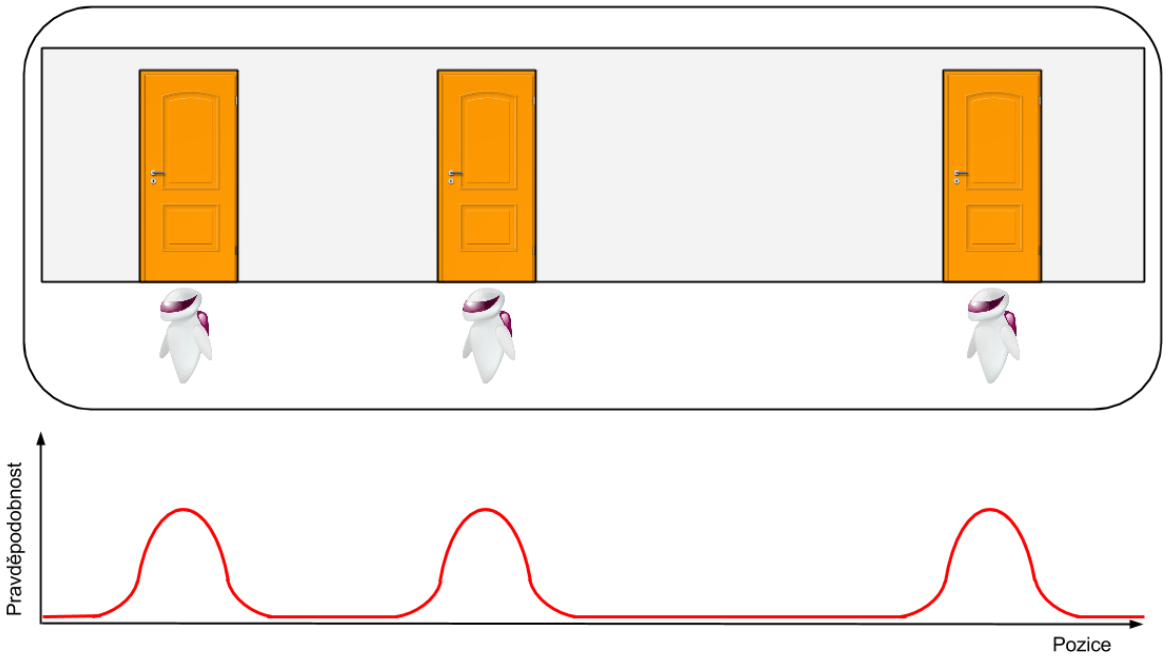
hodnotu, kdy robot stojí před prostředními dveřmi a podél stěny se může pohybovat pouze doleva, nebo doprava, je uveden na obrázku (Obr. 3.2.1) [5].

Velice často nastává situace, kdy počáteční polohu robota nelze odhadnout. Ale i přes neznalost skutečné počáteční polohy, je nutné odhad počáteční polohy inicializovat. V této situaci se počáteční hodnota odhadu inicializuje nějakou neutrální hodnotou, která reprezentuje skutečnost, že se robot v době svého spuštění nachází ve všech možných polohách. Tuto vlastnost splňuje rovnoměrné rozdělení pokrývající celý prostor, v němž se provádí lokalizace robota. Příklad inicializace rovnoměrným rozdělení (robot se nachází ve všech bodech) v jednodimenzionálním prostoru, kde se může robot pohybovat jen doprava, nebo doleva je uveden na obrázku (Obr. 3.2.2) [5].




**Obr. 3.2.2 – Inicializace počáteční polohy rovnoměrným rozdělením.**

Pomocí vhodné inicializace počátečního odhadu je možné vyjádřit i situace, kdy není přesně známa jedna počáteční poloha, ale lze o poloze získat určitou představu. Například může existovat několik možných výchozích pozic, nebo je známá určitá výchozí oblast. Tyto informace nejsou nutné zahrnout do počátečního odhadu polohy, ale při jejich použití se lokalizace robota může značně zrychlit, než při použití rovnoměrného rozložení. Na obrázku (Obr. 3.2.3) je uvedena inicializace tří výchozích poloh, kdy robot ví, že jeho výchozí poloha je před dveřmi, ale neví před jakými [5].



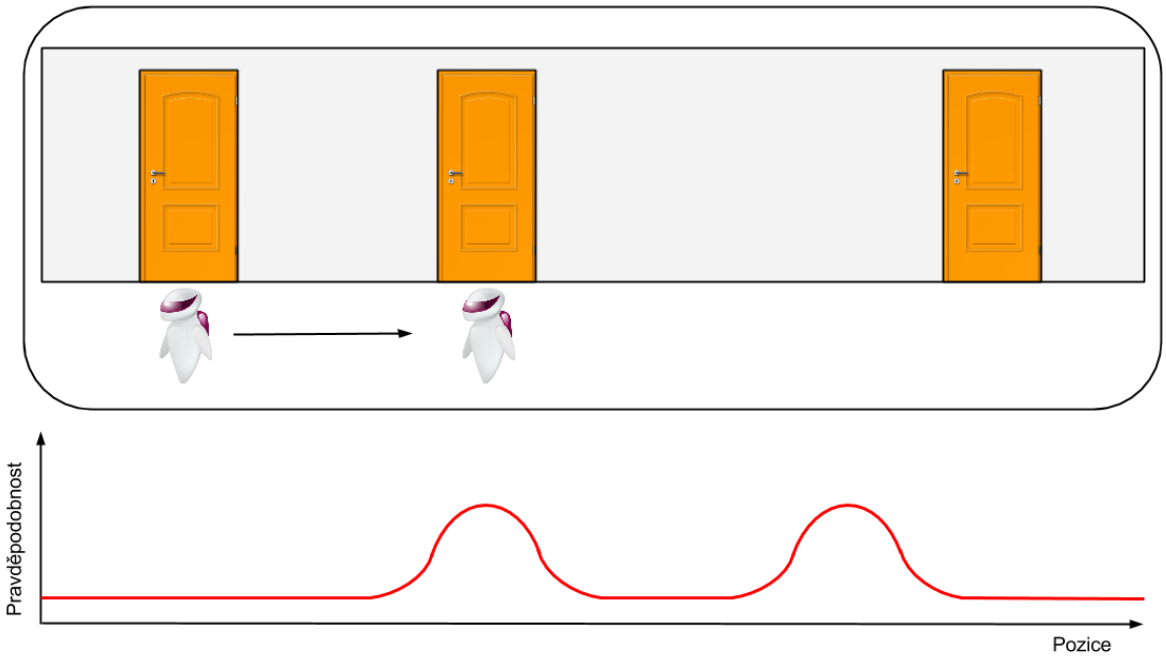

**Obr. 3.2.3 – Inicializace počáteční polohy pro více známých poloh.**

### 3.2.2 Apriorní odhad polohy

Apriorní odhad polohy (anglicky *prior belief*) se získá použitím predikčního kroku, který odhaduje polohu robota na základě předcházejícího odhadu polohy a posledních dat z relativních měření o změně polohy robota. Predikční krok se také někdy označuje jako time update a to proto, že se v pravidelných intervalech opakuje.

K výpočtu aktuálního apriorního odhadu pozice robota je potřeba znát počáteční polohu robota a data ze všech dosavadních měření, jak absolutních, tak relativních. Dále budeme potřebovat pravděpodobnostní popis toho, jak se data z posledního relativního měření promítnou do změny robotovy polohy. Proto bude v následující části vysvětlen pravděpodobnostní pohybový model.

Na obrázku (Obr. 3.2.4) je uveden příklad apriorního odhadu polohy, který vychází z počátečního odhadu polohy, kde robot ví, že stojí před dveřmi (Obr. 3.2.3) a robot se posune doprava.




**Obr. 3.2.4 – Apriorní odhad polohy.**

### 3.2.2.1 Pravděpodobnostní pohybový model

Pro apriorní odhad polohy je důležité vědět, jakým způsobem určitá akce změní polohu robota z původní polohy, na polohu novou. Akcí je v tomto případě míněn nějaký pohyb robota, který je detekován nebo změřen relativním měřením. Znalost těchto změn se nazývá pravděpodobnostní model, nebo také akční model.

Pravděpodobnostní pohybový model vyjadřuje podmíněnou pravděpodobnost nové polohy $l_k$ v aktuálním kroku $k$ za předpokladu, že v předchozím kroku $k-1$ provedl robot akci $a_k$ a nacházel se v poloze $l_{k-1}$. Vzhledem k tomu že nezáleží na kroku, ve kterém se akce provedla, není třeba indexovat [5].

Pravděpodobnostní pohybový model se nejčastěji získává odvozením z kinematického modelu robota. Méně častým způsobem získávání tohoto modelu je samoučení robota. Pravděpodobnostní pohybový model také zahrnuje očekávané systematické chyby, vznikající při použití konkrétních technik relativní lokalizace.

V případě, že robot není vybaven žádnými senzory pro relativní měření, lze místo relativních měření použít jako akci $a$ přímo ovládací povel, kterým se řídí pohyb robota. Tento způsob do odhadované polohy vnáší předpoklad, že řídící povel bude robotem bezchybně vykonán a povel se promítne do aktuální polohy robota a potencionálně další chyby, protože splnění řídícího příkazu nemusí být vždy dokonalé.

#### 3.2.2.2 Predikční krok

Za předpokladu, že pravděpodobnost přechodu z předcházející polohy $l_{k-1}$, do aktuální polohy $l_k$, závisí pouze na předcházející poloze $l_{k-1}$ a na provedené akci $a_k$, nikoli však na polohách, akcích a pozorování provedených v předcházejících krocích, můžeme určit pravděpodobnostní pohybový model $P(l_k|l_{k-1}, a_k)$.

Apriorní odhad polohy v kroku $k$ je tedy integrál přes všechny potenciální polohy $l_{k-1}$, ze součinu aposteriorní pravděpodobnosti polohy $l_{k-1}$ a pravděpodobnosti přechodu z polohy $l_{k-1}$ do polohy $l_k$ při provedení akce $a_k$.

$$Bel^{-}(l_k) = \int P(l_k|l_{k-1}, a_k) \cdot Bel(l_{k-1}) dl_{k-1} \qquad (3.2.1)$$

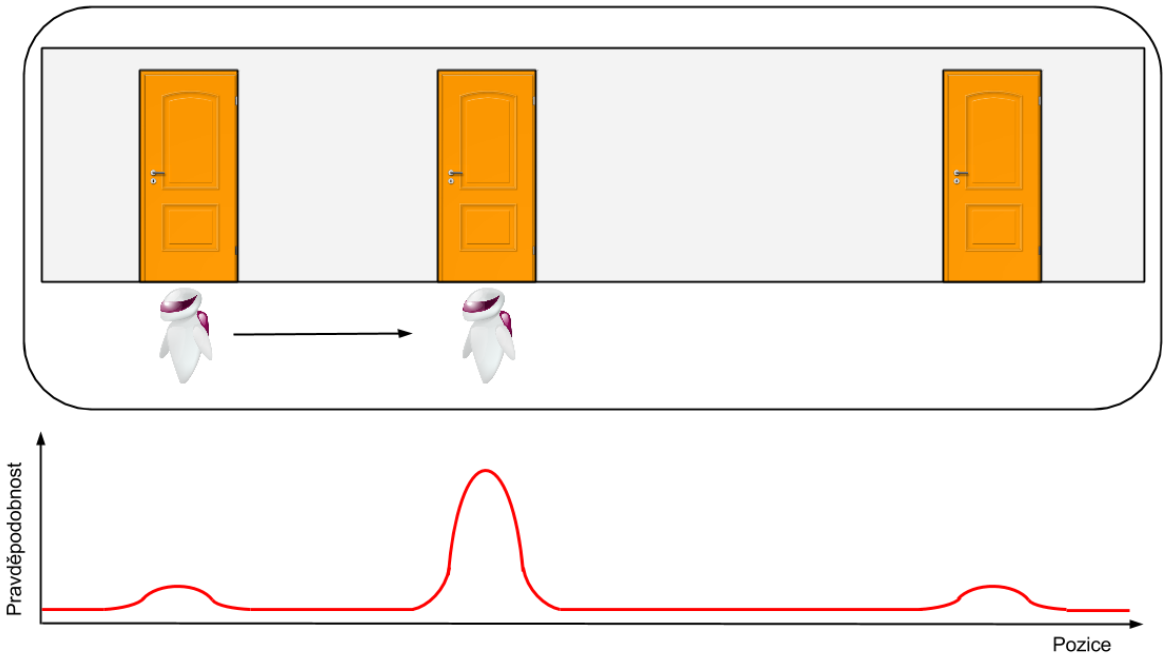
### 3.2.3 Aposteriorní odhad polohy

Pro provedení apriorního odhadu polohy, se provede korekční krok. Korekční krok se provádí na základě všech aktuálních informací o absolutní poloze robota. Odhad polohy pomocí korekčního kroku se nazývá aposteriorní (anglicky *posterior belief).*

K výpočtu aktuálního aposteriorního odhadu polohy je potřeba znát počáteční polohu robota a data ze všech dosavadních absolutních i relativních měření, přičemž poslední zahrnutou informací v aposteriorním odhadu polohy jsou data z posledního absolutního měření.

K výpočtu aposteriorního odhadu polohy budeme potřebovat vědět, jak data z absolutního měření interpretovat jako aktuální polohu robota v prostoru. K tomuto výpočtu je potřeba pravděpodobností model, který se v tom to případě nazývá senzorický model.

Na obrázku (Obr. 3.2.5) je uveden příklad aposteriorního odhadu polohy, který vychází z apriorního odhadu polohy, kde robot ví, že stojí před dveřmi (Obr. 3.2.4).




**Obr. 3.2.5 – Aposteriorní odhad polohy.**

### 3.2.3.1 Senzorický model

K výpočtu aposteriorního odhadu polohy $Bel^{+}(l_k)$ je nutné znát pravděpodobnostní model, který popisuje jakým způsobem použít naměřená data, získaná absolutním měřením (pozorováním) k určení aktuální polohy $l_k$. Tento model se nazývá pravděpodobnostní senzorický model (anglicky *sensor model*, nebo *perceptual model*).

Pravděpodobnostní senzorický model vyjadřuje podmíněnou pravděpodobnost zjištění pozorování senzory za předpokladu, že se robot nachází na poloze $l_k$. Vzhledem k tomu, že předpokládáme, že pravděpodobnost pozorování v jedné konkrétní poloze nezáleží na předchozích okolnostech, není třeba pozorování $z$ indexovat krokem $k$.

$$P(z|l) \tag{3.2.2}$$

Pravděpodobnostní senzorický model může být reprezentován více způsoby. Zpravidla to jsou:

- Předpis, který popisuje, jak lze hodnotu $P(z|l)$ vypočítat. Například ze znalosti rozmístění orientačních bodů (majáčků) v prostoru nebo pomocí ray-tracingu a známé mapy prostředí.
- Model, který se předem vypočítá pro všechny možné polohy $l$ a pozorování $z$.
- Pravděpodobnostní senzorický model se robot může také naučit. Ale robot musí mít i jiný způsob lokalizace, i když například jen dočasný.

Volba vhodné reprezentace pravděpodobnostního senzorického modelu, záleží na konkrétních požadavcích aplikace. Předpis je více paměťově úsporný, ale náročnější na výpočetní čas. Naopak uložení všech možných hodnot $P(z|l)$ je velice náročné na paměť, ale velice nenáročné na výpočetní výkon, což může být velice důležité pro lokalizaci v reálném čase.

V některých případech, například při vizuální lokalizaci pomocí kamer, mohou být pozorování natolik složitá, že by bylo vytvoření pravděpodobnostního senzorického modelu natolik složité, že by nebylo reálné ho používat. V takovém případě je nutné složitá vstupní data předzpracovat a získat z nich tzv. charakteristické rysy (anglicky *feature vector*). Předzpracování vstupních dat na charakteristické rysy se snaží zmenšit složitost a objem dat, při zachování co nejvíce informací. Takovéto předzpracování dat se nazývá extrakce charakteristických rysů (anglicky *feature extraction*).

Charakteristické rysy se poté používají ve výpočtu aposteriorního odhadu polohy místo dat absolutního měření. Způsobů, jakým se extrakce charakteristických rysů provádí, je více například: algoritmy pro zpracování obrazu, neuronové sítě. Extrakce charakteristických rusů při použití vizuální lokalizace pomocí kamery, může být například přítomnost nebo absence orientačního objektu v zorném poli kamery.

Při extrakci charakteristických rysů dochází zpravidla ke ztrátě informací ze senzorů, a proto dochází k znepřesnění lokalizace. To je ale vyváženo snížením výpočetní náročnosti, která by byla potřeba při zpracování dat, bez popsané extrakce na charakteristické rysy. A díky snížení výpočetního výkonu je možné lokalizaci využívat v reálném čase.

#### 3.2.3.2 Korekční krok

Jak již bylo naznačeno, k určení aposteriorního odhadu polohy je potřeba provést korekční krok. Protože se korekční krok provádí až po získání nějakého pozorování, označuje se také jako observation update. Korekční krok se vypočte následujícím vzorcem.

$$Bel^{+}(l_k) = \frac{P(z_k|l_k) \cdot Bel^{-}(l_k)}{C_k} \tag{3.2. 3}$$

Smyslem jmenovatele v tomto zlomku je zajistit, aby integrál pravděpodobností výskytu robota přes všechny polohy $l_k$ byl roven jedné. Tato hodnota se nazývá normalizační konstanta $C_k$. A aby byla splněna podmínka $\int Bel^{+}(l_k)\, dl_k = 1$ musí pro normalizační konstantu $C_k$ platit následující vzorec.

$$C_k = \int P(z_k|l_k) \cdot Bel^{-}(l_k) dl_k \qquad (3.2.4)$$

Aposteriorní odhad polohy v kroku $k$, je tedy normovaným součinem apriorního odhadu polohy a podmíněné pravděpodobnosti detekce absolutního pozorování.

## 3.3 Lokalizační vzorec

Robot pro svou lokalizaci potřebuje znát lokalizační vzorec (3.3.1) a také dva pravděpodobnostní modely, které jsou ve vzorci používány. Pravděpodobnostní pohybový model popsán v 3.2.2.1, který vyjadřuje vliv relativních měření na změnu polohy robota. Druhý model je pravděpodobnostní senzorický model popsán v 3.2.3.1, který popisuje souvislost polohy robota a výsledků absolutního měření provedené v dané poloze.

Kompletní rekurzivní vzorec pro aktualizaci odhadu polohy robota po získání relativních a absolutních měření se vyjádří spojením vzorců pro apriorní odhad polohy (3.2.1) a aposteriorní odhad polohy (3.2.3).

$$Bel(l_k) = \frac{1}{C_k} \cdot P(z_k|l_k) \cdot \int P(l_k|l_{k-1}, a_{k-1}) \cdot Bel(l_{k-1}) dl_{k-1} \qquad (3.3.1)$$

Kvůli rekurentnímu charakteru lokalizačního vzorce (3.3.1) potřebuje robot navíc znát počáteční odhad své polohy, získání počátečného odhadu polohy je popsané v 3.2.1.

## 3.4 Reprezentace odhadu polohy

Vzhledem k velkému počtu možných poloh robota, který je za předpokladu spojitosti souřadnic $x, y, \theta$ nekonečný, není reprezentace hustoty pravděpodobnosti nad tímto prostorem triviální. Protože je odhad polohy obecnou hustotou pravděpodobnosti a tuto hustotu lze obtížné přesně vyjádřit, vzniklo několik způsobů implementace, které hustotu pravděpodobnosti vyjadřují různými způsoby.

Různé reprezentace odhadu polohy se v zásadě liší ve třech aspektech:

- Přesnost, se kterou dokáže aproximovat skutečnou polohu odhadu.
- Omezení a potřebné předpoklady, které plynou z takového omezení.
- Výpočetní náročnost při jednotlivých operacích odhadu polohy.

Reprezentace odhadu polohy se může rozdělit do dvou základních kategorií, na diskrétní reprezentaci odhadu polohy a na spojitou reprezentaci odhadu polohy. Podrobněji lze dělit podle toho, jak je diskrétní nebo spojitá reprezentace implementována.

### 3.4.1 Diskrétní reprezentace odhadu polohy

Diskrétní reprezentace odhadu polohy definuje na spojitém prostoru pouze konečné množství hodnot, které odpovídají jednotlivým částem původního spojitého prostoru. Díky této vlastnosti lze integrály použité v lokalizačním vzorci (3.3.1) považovat za konečné sumy a explicitně je vypočítat.

Existuje celá řada implementací diskrétní reprezentace odhadu polohy. Jednotlivé implementace se v zásadě liší složitostí implementace, náročností výpočtů a úrovní detailu pohledu. V dalším textu budou stručně popsané některé implementace. Podrobný popis těchto implementací je nad rámec této práce.

#### 3.4.1.1 Topologický graf

Topologický graf je stejně jako každý graf tvořen uzly a hranami. Uzel představuje možnou polohu robota v prostoru, ve kterém se provádí lokalizace. Hrana grafu reprezentuje možný přechod z jedné polohy do druhé polohy. V každém uzlu grafu je uložena hodnota $Bel(l)$, která vyjadřuje pravděpodobnost, že se v tomto bodě nachází robot. Díky tomu umožňuje topologický graf vyjádřit umístění robota ve více místech současně.

Při použití topologických grafů mohou být jednotlivým uzlům grafu přiřazeny části prostoru, ve kterém se má robot lokalizovat. Přiřazení celých částí k uzlům, zajistí menší počet uzlů, a díky tomu je výpočetní náročnost samotného topologického grafu mnohem menší než u ostatních metod.

Protože topologický graf z principu neobsahuje žádné geometrické, respektive mapové informace, nelze na základě naměřených dat ze senzorů lokalizovat robota vůči geometrické mapě prostředí. Proto je nutné informace získané pomocí senzorů transformovat na charakteristické rysy přiřazené jednotlivým uzlům. Přepracování senzorických dat na charakteristické rysy může být natolik náročné, že i přes menší náročnost samotného lokalizačního výpočtu, může být topologický graf výpočetně velice náročný.

#### 3.4.1.2 Pravděpodobnostní mřížka

Pravděpodobnostní mřížka je jedna z jednodušších vyjádření odhadu polohy. Jde o rozdělení prostoru, v němž se robot lokalizuje pravidelnou mřížkou na jednotlivé buňky stejné velikosti. Každá buňka je indexována souřadnicemi a uchovává hodnotu $Bel(l)$, která reprezentuje pravděpodobnost, s jakou se v této buňce nachází robot.

Nevýhodou této implementace je výpočetní náročnost, která roste s velikostí prostoru a rozlišení mřížky (velikosti jedné buňky). Tento problém lze například řešit selektivní aktualizací, kde se buňkám s větší pravděpodobností věnuje větší pozornost než buňkám s malou pravděpodobností.

Pro výpočet, odhadu polohy při použití pravděpodobnostní mřížky, se použije lokalizační vzorec (3.3.1) ve svém diskrétním tvaru. Jako senzorický model se používá předem připravená tabulka o stejné velikosti, jako jsou rozměry mřížky. Takový senzorický model se také někdy označuje jako observation grid.

$$Bel(l_k) = \frac{1}{C_k} \cdot P(z_k|l_k) \cdot \sum P(l_k|l_{k-1}, a_{k-1}) \cdot Bel(l_{k-1}) \tag{3.4.1}$$

$$C_k = \sum P(z_k|l_k) \cdot Bel^{-}(l_k) \tag{3.4.2}$$

### 3.4.2 Spojitá reprezentace odhadu polohy

Spojitá reprezentace odhadu polohy reprezentuje vhodně zvolené spojité rozdělení pravděpodobnosti, které lze vyjádřit pomocí přesně daného počtu parametrů zvolené spojité funkce. Aktualizace odhadu apriorního a aposteriorního odhadu polohy potom spočívá v přepočítání parametrů zvolené funkce.

#### 3.4.2.1 Kalmanův filtr

Kalmanův filtr reprezentuje odhad polohy $Bel(l_k)$ prostřednictvím Gaussova, nebo-li normálního rozdělení. Gaussovo rozdělení je definováno pouze dvěma parametry, střední hodnotou ($l_k$), která určuje, kde se nachází vrchol Gaussovy křivky, a rozptylem ($\sigma_k^2$), který určuje šířku křivky. Kalmanův filtr není schopen vyjádřit tvrzení, že se robot nachází na více místech současně, a to proto, že vrchol Gaussovy křivky je pouze jeden [13].

$$Bel(l_k) = N(l_k, \sigma_k^2) \tag{3.4.3}$$

Kalmanův filtr má díky reprezentaci odhadu pouze dvěma hodnotami, na rozdíl od ostatních reprezentací polohy, velmi nízkou prostorovou a výpočetní náročnost. Tato vlastnost je kompenzována poněkud složitějšími vzorci pro aktualizaci odhadu polohy, která roste s počtem dimenzí reprezentujících prostor, ve kterém se lokalizuje.

Použití Gaussova rozdělení přináší na tuto metodu silné předpoklady, které se u ostatních metod nemusejí řešit. Jedním z hlavních předpokladů je, že po provedení predikčního i korekčního kroku a zahrnutí relativních i absolutních měření, zůstává odhad polohy ve tvaru popsatelném normálním rozdělením. Dále Kalmanův filtr předpokládá, že

systém, ve kterém se lokalizuje, je proměnný v čase. Tedy lineární dynamický systém, jehož model lze popsat pomocí diferenciálních rovnic [13]. Pro řešení složitějších modelů, které jsou nelineární, slouží složitější varianta této metody, která se nazývá Rozšířený Kalmanův filtr.

Požadavky jsou také kladeny na pravděpodobnostní senzorický model a pravděpodobností pohybový model. Senzorický i pravděpodobnostní pohybový model musí mít, stejně jako odhad polohy, normální rozdělení. Tento požadavek, zejména u senzorického modelu, zcela vylučuje použití některých potencionálních možností absolutního měření. Mezi metody absolutního měření, které při použití Kalmanova filtru nelze použít, patří například rozpoznávání barvy, detekce neunikátních orientačních bodů jako jsou dveře, okna a rohy.

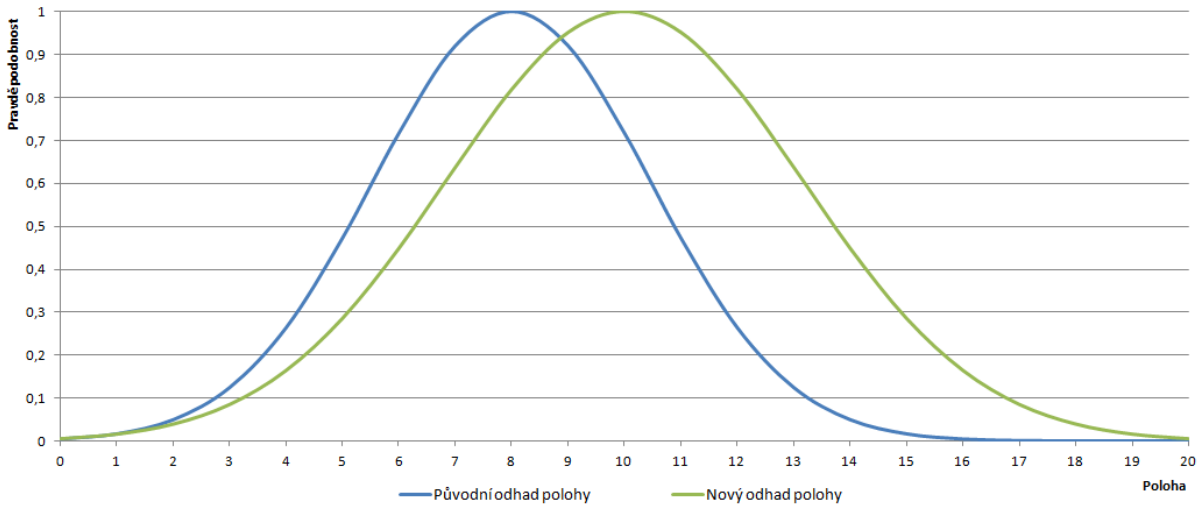
Požadavek normálního rozdělení pro pravděpodobnostní senzorický a pohybový model se realizuje tak, že se k naměřeným hodnotám absolutního nebo relativního měření přidá šum s normálním rozdělením [13].

Průběh lokalizace touto metodou se principiálně neliší od kroků popsaných v pravděpodobnostní lokalizaci. Nejprve se provede počáteční odhad polohy robota a to tak, že se hodnota odhadu polohy inicializuje úzkým nebo širokým normálním rozdělením, podle toho jak jistě robot zná svou počáteční polohu. Poté se střídá apriorní a aposteriorní odhad polohy. Dále zde bude použit princip fungování Kalmanova filtru na jednodimenzionálním prostoru.

Při apriorním odhadu polohy se podle relativního měření upraví odhadovaná pozice. Rozptyl odhadu polohy se zvětšuje o rozptyl, neboli nejistotu, kterou do odhadu vnáší provedené měření. Vzorce pro aktualizace parametrů Gaussovy křivky jsou pro jednodimenzionální prostor velice jednoduché. Pro výpočet nové polohy (střední hodnoty) $l_k$ se přičte posun $a$ k přecházející poloze $l_{k-1}$. Pro aktualizace rozptylu $\sigma_k^2$ se k přecházejícímu rozptylu $\sigma_{k-1}^2$ přičte nepřesnost relativního měření $r^2$.

$$l_k = l_{k-1} + a \tag{3.4.4}$$

$$\sigma_k^2 = \sigma_{k-1}^2 + r^2 \tag{3.4.5}$$



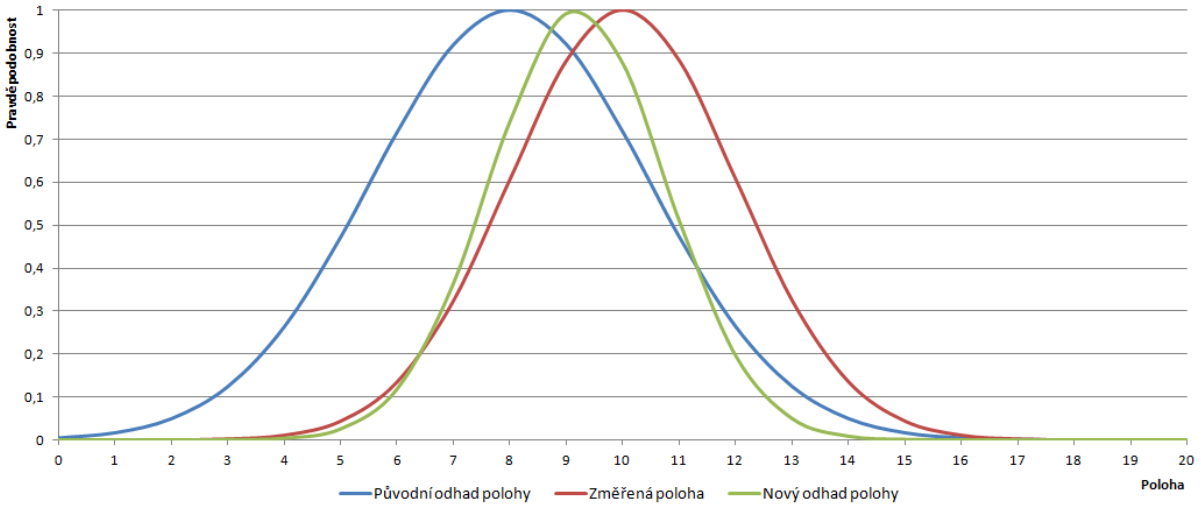

**Obr. 3.4.1 – Apriorní odhad polohy při použití Kalmanova filtru.**

Po apriorním odhadu následuje aposteriorní odhad polohy. Při aposteriorním odhadu polohy se nová poloha v zásadě vypočítá jako vážený průměr původní polohy a polohy získané absolutním měřením. Parametr určující váhu původní polohy a absolutního měření se nazývá Kalmanův zisk a je závislý na rozptylu jak z původního odhadu, tak z absolutního měření. Do aposteriorního odhadu se větší mírou promítá menší rozptyl, proto se v tomto kroku rozptyl odhadu v zásadě zmenšuje [5].

$$l_k = \frac{1}{\sigma_{k-1}^2 \cdot r^2} \cdot (r^2 \cdot l_{k-1} + \sigma_{k-1}^2 \cdot z) \tag{3.4.6}$$

$$\sigma_k^2 = \frac{1}{\frac{1}{\sigma_{k-1}^2} + \frac{1}{r^2}} \tag{3.4.7}$$

Kde $z$ je střední hodnota a $r^2$ je rozptyl získaný absolutním měřením.




**Obr. 3.4.2 – Aposteriorní odhad polohy při použití Kalmanova filtru.**

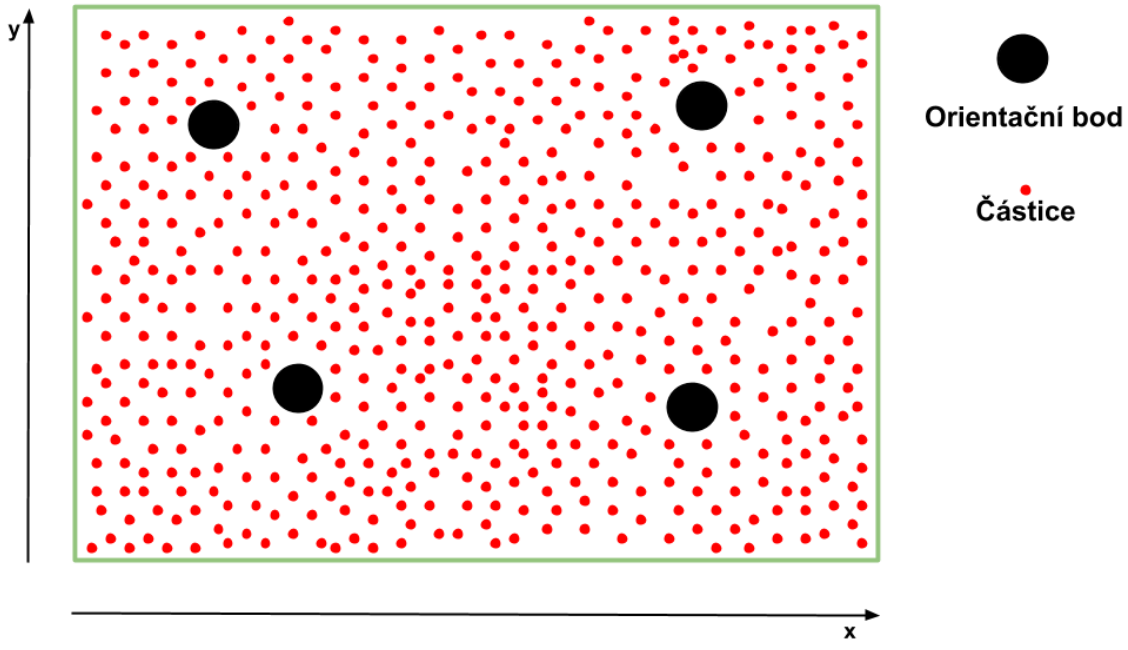
Kalmanův filtr se používá v celé řadě dalších oblastí, jako je například navigace, počítačové vidění, předzpracování senzorických dat, sledování stavů pomocí nepřímého měření. Kalmanův filtr je použit v samo-řídicím autě od Google ke sledování okolních automobilů a odhadu jejich rychlosti a polohy. Kalmanův filtr byl také použit v Stanley, robotickém automobilu, který jako první v roce 2005 dokončil robotický závod DARPA Grand Challenge [4].

#### 3.4.2.2 Částicový filtr

Částicový filtr reprezentuje odhad polohy $Bel(l_k)$ prostřednictvím hustoty částic, které jsou rozprostřeny v prostoru, ve kterém se provádí lokalizace. Každá částice obsahuje informace, které určují její polohu v prostoru. Díky tomu, že částice v prostoru mohou tvořit více shluků na více místech, lze pomocí částicového filtru vyjádřit více hypotéz o umístění robota.

Částicový filtr je velice jednoduchý na implementaci, a proto je v posledních letech stále více používán. Na rozdíl od Kalmanova filtru lze pomocí částicového filtru snadněji vyjádřit více dimenzí s mnohem menší výpočetní i prostorovou náročností.

Všechny příklady v následujícím textu, budou uvedeny pro tří dimenzionální prostor, který je definován dvěma rozměry $x, y$ a hodnotou $\theta$, která určuje natočení robota. Každý částicový filtr uchovává informaci o počtu použitých částic $N$ a seznam všech částic $P$.



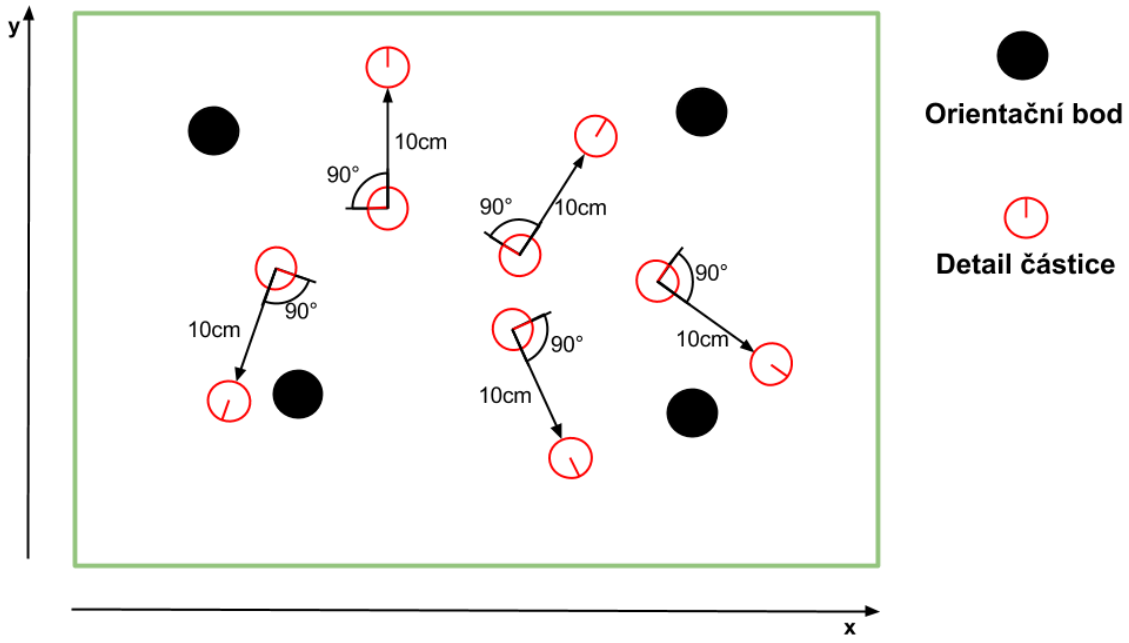

**Obr. 3.4.3 – Inicializace částicového filtru.**

Lokalizace probíhá ve stejných krocích jako u ostatních metod. Nejprve se provede počáteční odhad polohy. Počáteční odhad polohy se v případě, že robot neví, kde se při zapnutí nachází, provede vygenerování $N$ částic s náhodným umístěním v prostoru. Náhodné umístění částic zajistí téměř rovnoměrné pokrytí celého prostoru.

Příklad počátečního odhadu polohy je zobrazen na Obr. 3.4.3. Na obrázku jsou vidět náhodně rozmístěné částice. Každá částice ($P_i$) obsahuje informace o své poloze. V prostoru jsou dále umístěny orientační body, jejich význam pro lokalizaci bude popsán v následujícím textu.

Po počátečním odhadu polohy následuje apriorní odhad polohy. Po provedení relativního měření se na základě naměřených dat provede na všech částicích stejný pohyb, který provedl robot. Pro zjednodušení implementace se předpokládá, že je svět pro částice cyklický. To znamená, že pokud by se některá z částic měla dostat za hranice vymezeného prostoru, pokračuje od protější strany omezující prostor z druhé strany.

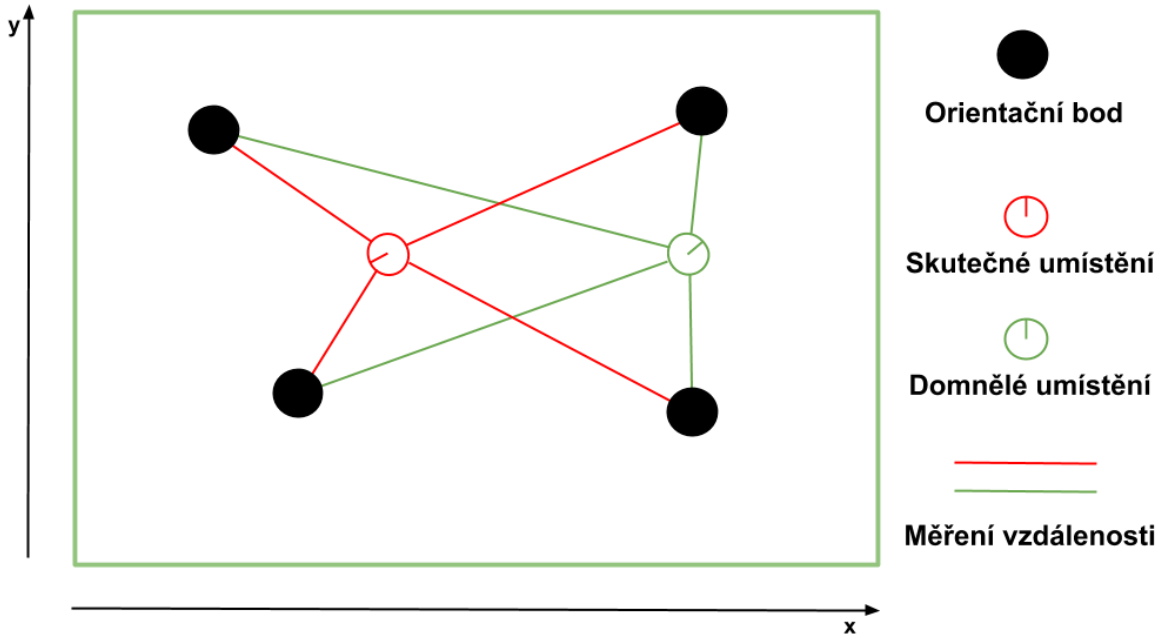
Za předpokladu že relativní měření je nahrazeno příkazem, který je poslán robotovi k vykonání pohybové akce, se tento příkaz může aplikovat na všechny částice v prostoru. To znamená, že když robot obdrží příkaz, který říká, aby se otočil o 90° a ujel 10cm, všechny částice se otočí o 90° a posunou se o 10cm ve směru svého natočení. Příklad takovéhoto pohybu je na Obr. 3.4.4.




**Obr. 3.4.4 – Ukázka změny polohy částic, při apriorním odhadu polohy.**

Aposteriorní odhad polohy se může rozdělit do několika kroků. Nejprve se pro každou částici určí její váha významnosti (anglicky *importance weights*), která, říká jak významná je daná částice. Čím větší hodnota, tím větší význam. Poté následuje krok, který se nazývá převzorkování (anglicky *Resampling*), ten na základě vah významnosti vytvoří novou sadu částic, který nahradí předchozí sadu částic [5].

Výpočet váhy významnosti ($w_i$) se provádí na základě absolutního měření, provedeného robotem a předpokládaného měření, které provádí částice v modelu prostoru. Nesoulad mezi těmito dvěma měření vede k jištění váhy významnosti. Za předpokladu, že robot provádí absolutní měření k orientačním bodům, které se nacházejí v prostoru lokalizace, může takové měření vypadat stejně jako na Obr. 3.4.5.




**Obr. 3.4.5 – Skutečné a předpokládané absolutní měření.**

Před zahájením samotného převzorkování, se musejí váhy jednotlivých částic normalizovat. Normalizace spočívá v tom, že součet všech hodnot normalizovaných vah se rovná jedné. Normalizovaná váha se označuje $\propto_i$, kde $i$ je index částice.

$$\propto_i = \frac{w_i}{W} \tag{3.4.8}$$

kde

$$W = \sum w_i \tag{3.4.9}$$

Když jsou váhy ($w_i$) jednotlivých částic normalizované ($\propto_i$), provede se převzorkování. Převzorkování znamená, že se vytvoří nová sada částic, která nahradí stávající

sadu vzorků. Do nové sady částic se může dostat kterákoliv částice, ale částice s větší váhou významnosti mají větší pravděpodobnost, že se tam dostanou, než částice s nízkou váhou.

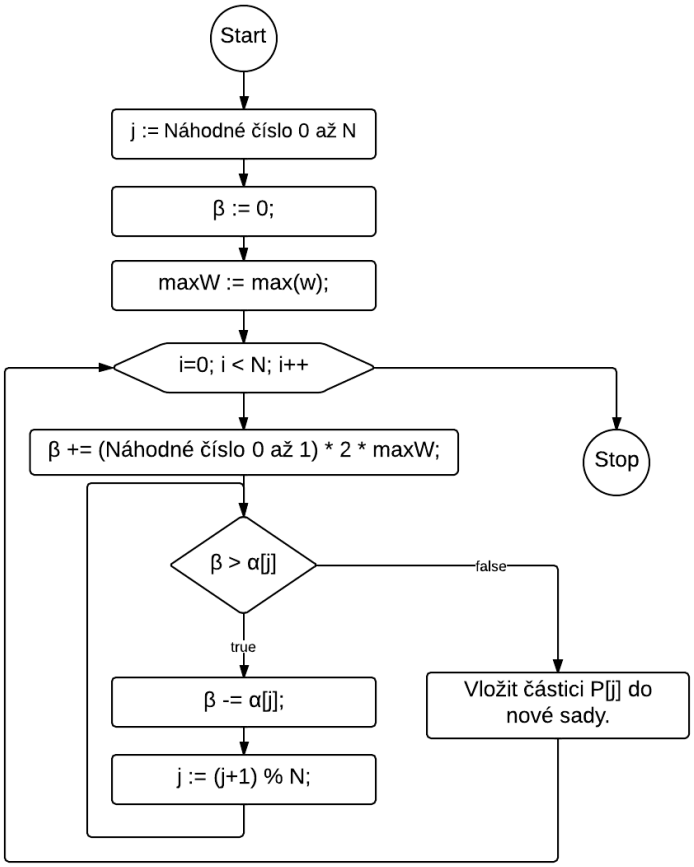
Protože normalizovaná váha ($\propto_i$) udává, s jakou pravděpodobností se částice dostane do nové sady částic, tak by převzorkování mohlo postupně procházet všechny částice a na základě hodnoty $\propto_i$ rozhodnout zda bude, či nebude v nové sadě. Ovšem pro převzorkování se doporučuje algoritmus, který je někdy označován *resampling wheel*. Tento algoritmus je jednoduchý na implementaci a experimentálně bylo dokázáno že vytváří lepší vzorky než ostatní algoritmy [5].

Algoritmus funguje tak že normalizované váhy všech částic se rozprostřou do kruhu a každá částice, tak v tomto kruhu zabírá jinak velkou výseč. Velikost výseče je přímo úměrná váze částice. To znamená že, částice s větší váhou bude zabírat větší část kruhu, než částice s malou váhou. Poté náhodně vygeneruje index (označovaný jako $j$), kterékoliv částice z dosavadní sady částic. Dále se nastaví hodnota $\beta = 0$. Následující kroky se budou $N$-krát opakovat, tedy tolikrát jako je počet používaných částic.

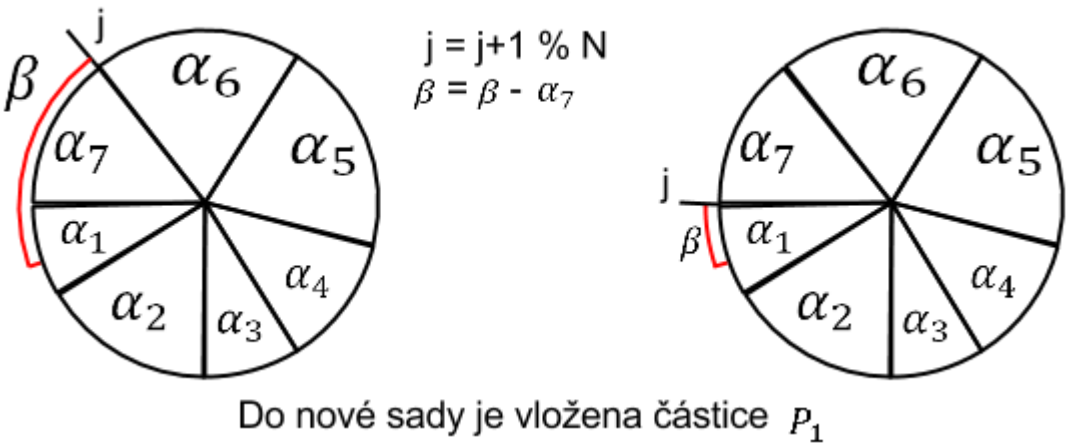
1. K hodnotě $\beta$ se přičte náhodné číslo z rozsahu nula až dvojnásobek maximální hodnoty normalizované váhy $w_i$, tedy $\max(w_i)$.
2. Pokud je normalizovaná váha částice s indexem $j$ menší než hodnota $\beta$ provedou se následující kroky.
    - Od hodnoty $\beta$ se odečte normalizovaná váha částice s indexem $j$, tedy $\alpha_j$.
    - Index $j$ se posune na následující částici v aktuální sadě částic.
    - Vrací se ke kroku 2.
3. Vloží se částice s indexem $j$ do nové sady částic.
4. Pokud již nebylo opakováno $N$-krát, vrací se ke kroku 1.

Po provedení tohoto postupu vznikne nová sada částic, která nahradí aktuálně používanou sadu částic. Vývojový diagram algoritmu pro převzorkování je na Obr. 3.4.6 a grafický příklad fungování tohoto algoritmu je na Obr. 3.4.7.

Částicový filtr se kromě lokalizace používá v celé řadě oblastí navigace, rozpoznávání objektů, počítačové vidění a ekonometrii. Často se používá jako alternativa k rozšířenému Kalmanovu filtru, kde při použití dostatečného počtu částic, dosahuje částicový filtr výrazně lepších výsledků. Částicový filtr je použit v samo-řídícím autě Google ke zpřesnění své pozice na základě senzorických dat a silných mapových podkladů [5].




**Obr. 3.4.6 – Vývojový diagram algoritmu resampling wheel.**




**Obr. 3.4.7 – Grafický příklad algoritmu resampling wheel.**

# 4 Praktické řešení

Cílem praktické části práce je sestavit robota ze stavebnice LEGO Mindstorms Education a na něm aplikovat lokalizaci pomocí částicového filtru. V této kapitole je popsána stavba robota, instalace potřebného softwarového vybavení, způsob komunikace mezi robotem a řídicí stanicí a složitější části implementace částicového filtru.

## 4.1 Konstrukce LEGO robota

Robot je sestaven ze stavebnice LEGO Mindstorms Education, která si klade za cíl zpřístupnit výuku robotiky zájemcům z širší veřejnosti. Použitá stavebnice LEGO Mindstorms Education je v pořadí již druhou generací robotické stavebnice od skupiny LEGO, která začala s vývojem robotických stavebnic již v roce 1998. Použitá verze stavebnice NXT 2.0, kterou předcházela verze NXT 1.0, byla produkována od roku 2006 a v době psaní této práce (2013) byla oznámená nová verze stavebnice NXT 3.0. Stavebnice se skládá z konstrukčních dílu LEGO Technic, snímacích a řídicích prvků [7].

Jedna základní sada obsahuje následující bloky:

- NXT řídící jednotku.
- Dva dotekové senzory.
- Jeden senzor snímající barvu.
- Jeden ultrazvukový senzor.
- Tři servomotory se zabudovaným čidlem snímající rotaci.
- Sedm různě dlouhých kabelů spojující motory a čidla s NXT kostkou.
- Příručka obsahující základní pokyny pro sestavení robota a obsluhu programovacího prostředí.
- CD s grafickým programovacím jazykem NXT-G.
- Konstrukční díly LEGO.

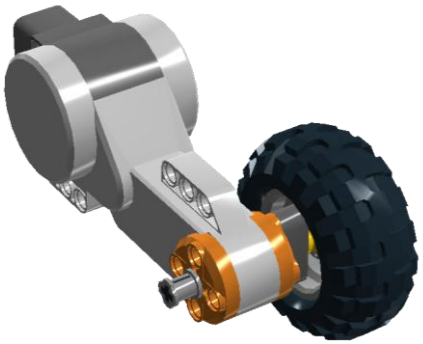
Základním řídicím prvkem stavebnice LEGO Mindstorms Education je tzv. NXT kostka. Ta zajišťuje veškeré řídící a výpočetní operace. Disponuje také bezdrátovým rozhraním bluetooth pro komunikaci s ostatními zařízeními, díky tomu že rozhraní disponuje více kanály, může komunikovat s více zařízeními ve stejný okamžik [5].

Od výrobce je v NXT kostce nahrán software zpracovávající instrukce grafického programovacího jazyka NXT-G, který byl společností LEGO za tímto účelem vyvinut. Ale protože programovací jazyk NXT-G je značně omezen svou jednoduchostí a složitější programy by v něm nešlo vytvořit vůbec nebo s velkou námahou a za cenu značné

nepřehlednosti. Proto byly nezávislými vývojáři vyvinuty další firmwary, které jsou schopné zpracovávat programy napsané v jiných programovacích jazycích, jako je Java, C++ a další. Firmware se nahraje do interní paměti NXT kostky a přehraje předchozí používanou verzi firmwaru.

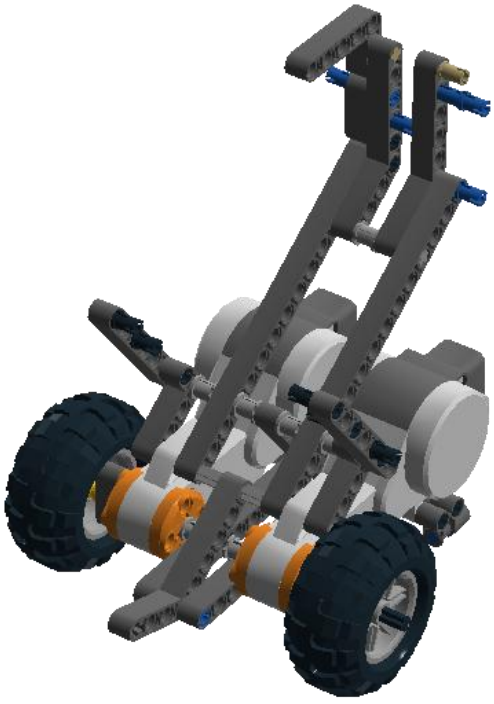
Podvozek robota sestaveného za účely této práce, tvoří tři kola, přední dvě kola mají každé svůj samostatný pohon, slouží k natáčení a pohybu robota. Malé zadní kolo slouží jako stabilizační. Na vrcholu je umístěna otočná věž s ultrazvukovým senzorem. Na spodní straně robota je ještě umístěn kompas, který ale není nutný pro fungování lokalizace. Kompas je zde umístěn kvůli snadnějšímu ovládání pohybu robota. Stručná konstrukce robota je zachycena na několika následujících obrázcích, podrobný návod na sestavení robota je v přiloženém soboru Konstrukce robota.

**Obr. 4.1.1 – Robot pohled zepředu.**



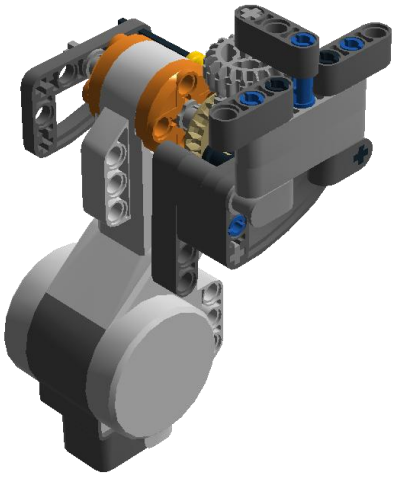
**Obr. 4.1.2 – Kolo s motorem.**

**Obr. 4.1.3 – Pomocné kolo.**



**Obr. 4.1.4 – Podvozek robota.**

**Obr. 4.1.5 – Senzorická věž.**



**Obr. 4.1.6 – Pohon senzorické věže.**

**Obr. 4.1.7 – Robot zezadu.**

## 4.2 LeJOS

LeJOS[1] je kompletní open source software, pod licencí Mozilla Public Licence, který je určen pro ovládání robota řízeného LEGO NXT kostkou. Jeho součástí je miniaturní Java Virtual Machine (JVM), která byla navržena pro běh na omezených parametrech NXT kostky. LeJOS JVM byla vytvářena na základě konceptu TynyVM, který byl určen pro embedded systémy, a protože byl vytvořen v jazyce C s důrazem na multiplatformost, je ho možné použít i na jiné zařízení než LEGO NXT [3].

LeJOS dále obsahuje soubor tříd založených na jazyce Java SE, které slouží k programování robota. LeJOS využívá silných vlastností jazyka Java SE jako je například použití primitivních datových typů, objektového programování, preemptivní vlákna, datové streamy, výjimky a další. Také podporuje většinu tříd z balíčku java.lang, java.util a java.io. Třídy LeJOS lze rozdělit do dvou základních skupin, na třídy pro implementaci programu určeného pro běžnou JVM a na třídy pro implementaci programu určeného pro LeJOS JVM.

LeJOS také obsahuje doplňkové programy, které slouží k ovládání zařízení. Patří mezi ně například program na přehrání firmware do NXT kostky, kompilace a odeslání programu do kostky a další. Protože LeJOS nemá své vlastní vývojové prostředí, obsahuje zásuvné moduly do dvou Java IDE a to do Eclipse a ve starších verzích také do NetBeans.

### 4.2.1 Instalace LeJOS

LeJOS lze nainstalovat na celou řadu platforem, oficiální cestou na Windows, Linux a MacOS. V této práci bude popsán pouze postup pro instalaci na platformu Windows, kde je předpoklad, že toto řešení je nejčastější a také byla na této platformě tato práce vypracována.

Nejprve se musí stáhnout a nainstalovat nejnovější verze Java Developer Kit (JDK) ze stránek výrobce: http://www.oracle.com/technetwork/java/javase/downloads/index.html. V tomto kroku je nutné dát pozor na verzi JDK. LeJOS je kompatibilní jen s 32bitovou verzí Javy SE. Po instalaci JDK je nutné zkontrolovat nastavení proměnných prostření počítače [5]. Ve Windows se dá k nastavené proměnných prostředí dostat přes *Ovládací panely -> Systém -> Změnit nastavení -> Upřesnit -> Proměnné prostředí -> Systémové proměnné.* Nastavení by mělo být stejné, jako je uvedeno v Tab. 4.2.1.

[1] Vyslovuje se jako španělské slovo lejos, které v překladu znamená vzdálený či daleko a vyslovuje se léhos. Jde o zkratku výrazu LEGO Java Operating System, kde první dvě písmena (LE) zkracujíc slovo LEGO a zbylé tři Java Operating System. Zkratku lze také kompletně překládat jako The Java operating system, to proto že v některých jazycích je „le“ překládáno jako „the“.

| Proměnná | Popis | Příklad |
|---|---|---|
| **JAVA_HOME** | Cesta k adresáři, kde je nainstalované JDK. | C:\Program Files\Java\jdk1.7.0 |
| **PATH** | Cesta k adresáři bin, nainstalovaného JDK. | C:\Program Files\Java\jdk1.7.0\bin |
| **NXJ_HOME** | Cesta k adresáři, kde je nainstalován LeJOS | C:\Program Files\leJOS NXT |

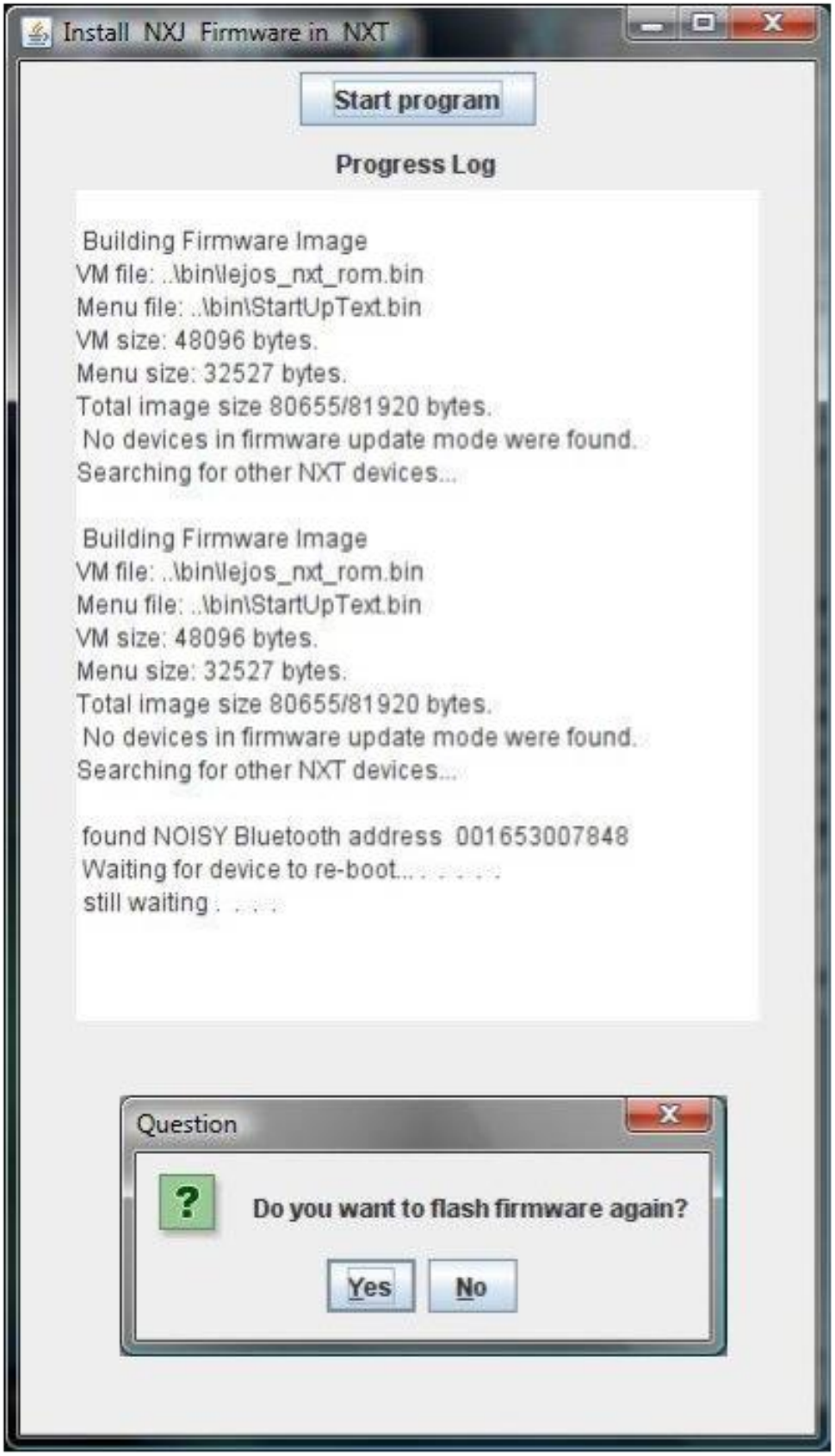
**Tab. 4.2.1 – Nastavení proměnných prostředí Windows.**

Dalším krokem před instalací samotného LeJOS je instalace oficiálních USB ovladačů (USB Fantom Driver) pro NXT kostku od společnosti LEGO. Ten se většinou dodává společně s robotem, ale doporučuje se stáhnout nejnovější verzi z oficiálních stránek výrobce: http://mindstorms.lego.com/en-us/support/files/Driver.aspx.

Když jsou nainstalované všechny programy z předchozích kroků, může se přejít k instalaci samotného LeJOS. Instalační balíček je ke stažení na stránkách vývojářů: http://sourceforge.net/projects/lejos/files/. Doporučuje se stáhnout nejnovější verzi, ale pro případy používání programu, který již není v nové verzi podporován, je možné na stránkách stáhnout i předcházející verze. Po stažení instalačního balíčku se spustí instalace, během instalace se postupuje podle pokynů instalátoru.

Při posledním kroku instalace se průvodce dotáže, zda chceme provést flash NXT kostky, to znamená, zda chceme nahrát LeJOS NXT do kostky. Program pro instalaci firmware do kostky lze také spustit v příkazovém řádku příkazem *nxjflashg*, nebo spuštěním stejnojmenného programu ve složce /bin v adresáři LeJOS.

Instalaci lze provést i bez grafického průvodce to je doporučeno, jen když grafická instalace selže nebo pokud se požadují nějaká nestandardní nastavení, která při grafické instalaci nelze nastavit. Oba dva způsoby instalace jsou popsány v tutoriálu na stránkách: http://lejos.sourceforge.net/nxt/nxj/tutorial/Preliminaries/GettingStartedWindows.htm.




**Obr. 4.2.1 – Průběh instalace firmwaru do NXT kostky.**

### 4.2.2 Příprava vývojového prostředí NetBeans

LeJOS neobsahuje žádné vlastní vývojové prostředí, jako je tomu například u jazyka NXT-G vyvíjeného firmou LEGO, který má vlastní IDE. K vývoji LeJOS aplikace se dají použít dvě cesty.

První cestou je psaní kódu v libovolném textovém editoru a následném přeložení Java kompilátorem s využitím LeJOS NXT tříd přes příkazový řádek. Přeložená aplikace ve formátu nxj se poté musí ručně přenést do NXT kostky pomocí Bluetooth, nebo USB rozhraní. Zároveň dojde vytvoření nxd souboru, který slouží k ladícím účelům. Přesto, že tento způsob je krátký, vývoj tímto způsobem je značně nepohodlný a pro složitější aplikace se nedoporučuje.

Druhou a určitě lepší cestou je použití některého z podporovaných Java IDE, které celý proces kompilace, přenesení a případné ladění vykonávají automaticky za nás. V této práci bude popsáno nastavení vývojového prostředí NetBeans a to dvěma způsoby. Jedním z nich je využití zásuvného modulu, dalším řešením je použití nástroje Ant.

#### 4.2.2.1 Konfigurace s použitím nástroje Ant

První cestou jak nastavit vývojové prostředí NetBeans pro programování LeJOS aplikací je použitím softwarového nástroje Ant, který NetBeans nabízí. Jedná se o nástroj, který na základě pokynů uložených v XML dokumentu, celou aplikaci sestaví.

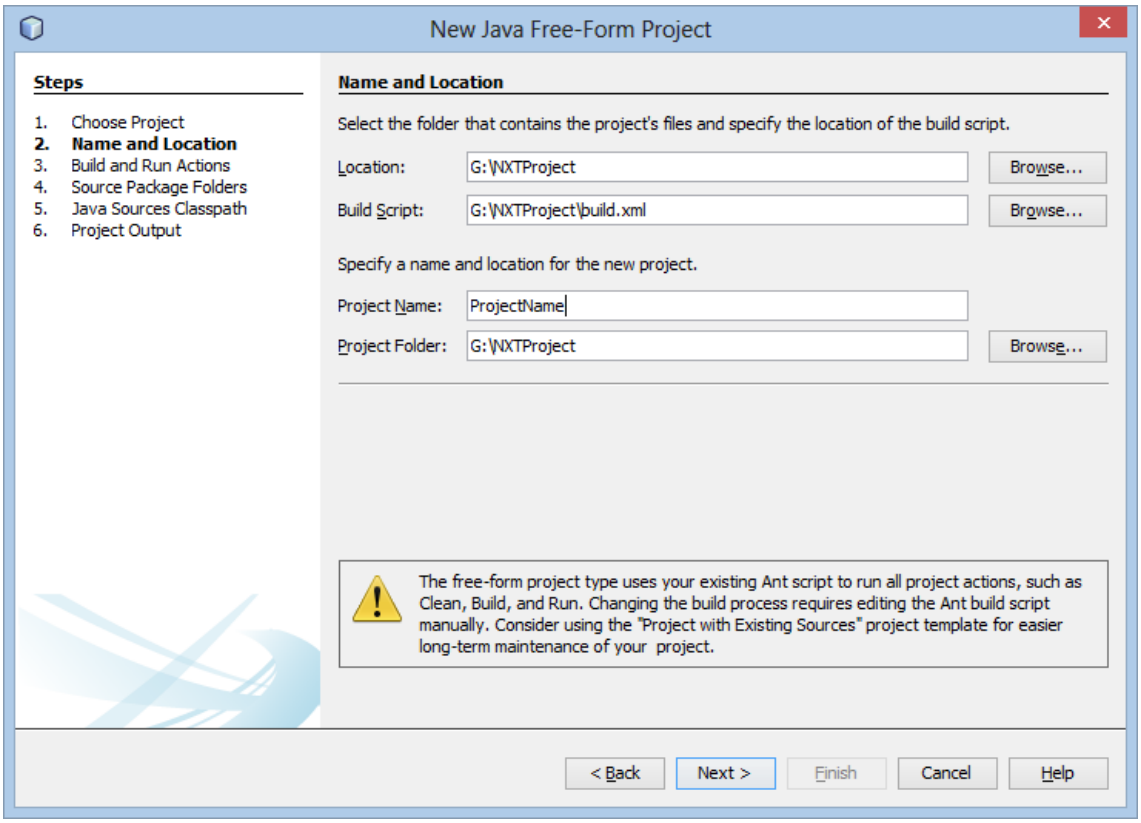
Nejprve je nutné vytvořit složku s názvem projektu. V takto vytvořené složce se vytvoří adresář *src* a nakopírují se soubory *build.xml* a *build.PROPERTIES* z adresáře [3]:

*\LeJOS NXJ Development Kit Sources\org.lejos.example*

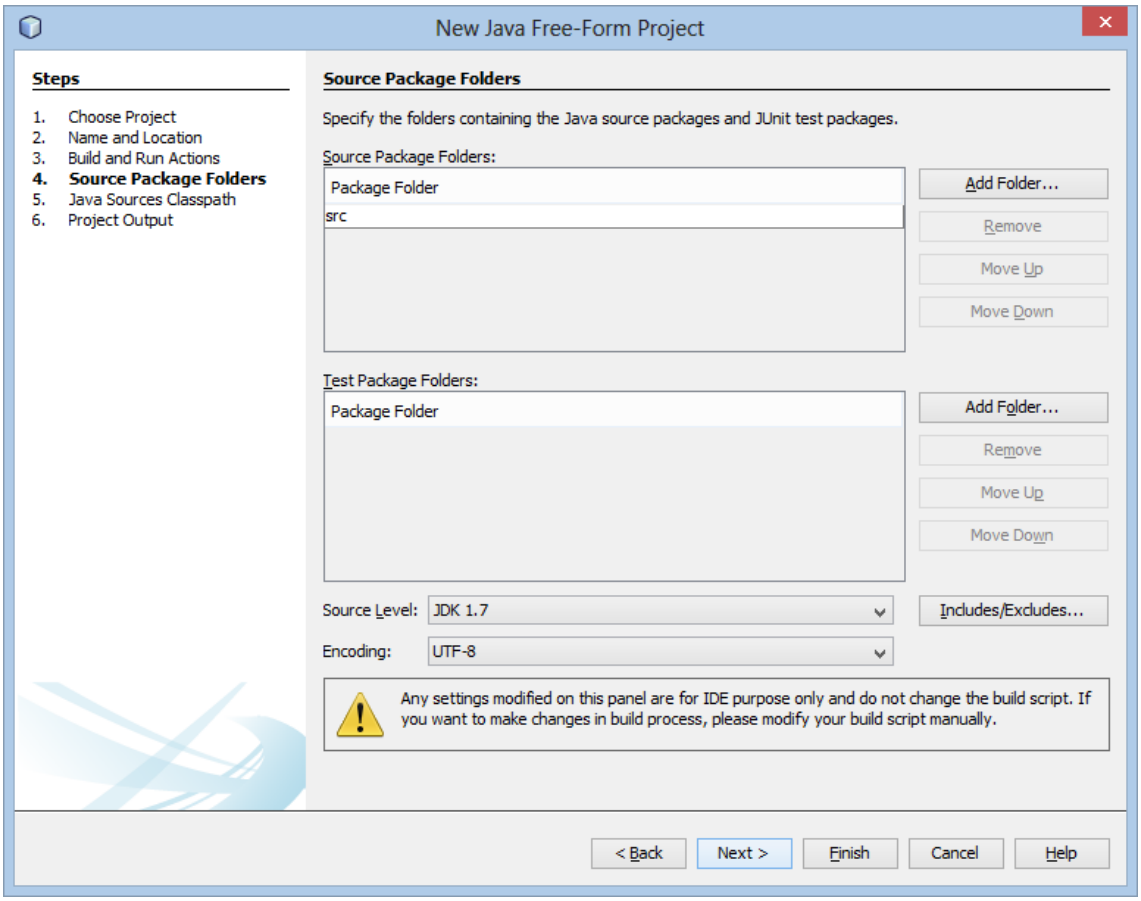
Nyní se v NetBeans vytvoří nový Java Free-Form Projekt, který pro kompilaci využívá Ant skript. Tento projekt se vytvoří v: *File -> New Project… -> Categories: Java -> Projects: Java Free-Form Project*. Následující dialogové se vyplní podle obrázku Obr. 4.2.2, kde stačí vyplnit umístění složky vytvořené v předešlém kroku a název projektu.

Po kliknutí na tlačítko *Next* je vyžadované přiřazení příslušných funkcí (sestavení projektu, spuštění projektu) ke skriptům obsaženým v *build.xml*. Zde není nutné nic nastavovat, hodnoty by se měly nastavit automaticky [3].

Na další kartě se nastavuje umístění adresářů se zdrojovými kódy, adresář pro uchování testovacích kódů, verze JDK a kódování. Nastavení tohoto kroku je zobrazeno na obrázku Obr. 4.2.3. Kódování se doporučuje nastavit na UTF-8.



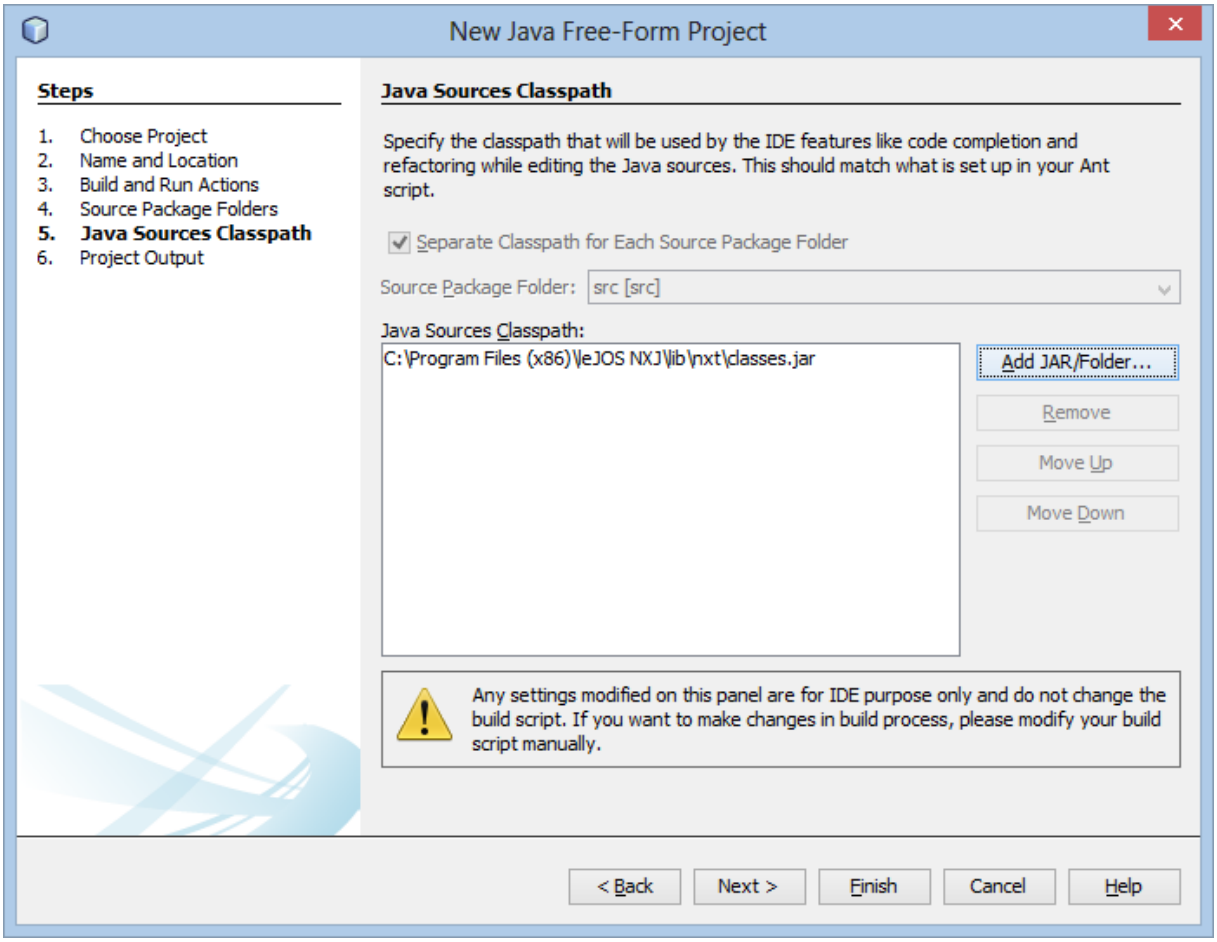

**Obr. 4.2.2 – Vytvoření Java Free-Form Projektu.**




**Obr. 4.2.3 – Nastavení adresáře se zdrojovými kódy.**

Posledním krokem před vytvořením projektu je připojení knihovny s NXT třídami pro ovládání kostky. Knihovna se nachází v složce kde je nainstalovaný LeJOS v adresáři:

*\lib\nxt\classes.jar*




**Obr. 4.2.4 – Připojení NXT knihovny k projektu.**

Po potvrzení posledního kroku se vytvoří prázdný projekt. Dále je nutné vytvořit vhodně pojmenovaný balíček a do něj vytvořit spouštěcí třídu *Demo* (na názvu nezáleží) s metodou *main*.

Posledním krokem je upravit skript *build.xml*. V souboru se vyhledá

```
<description>
        org.lejos.example.HelloWorld build file
</description>
```

a edituje org.lejos.example.HelloWorld na balíček a název spouštěcí třídy vytvořené v přecházejícím kroku.

```
<description>
        nazev_balicku.SpousteciTrida build file
</description>
```

Poslední úprava se provede v soboru *build.PROPERTIES* kde je nutné vyhledat a upravit cestu ke spouštěcí třídě.

```
main.class= nazev_balicku.SpousteciTrida
output.basename=SpousteciTrida
```

Po všech těchto úpravách je projekt připraven ke spuštění. Spuštění se provádí přes kontextovou nabídku vyvolanou nad souborem *buil.xml*: *Run target -> uploadandrun*. Tento příkaz provede přeložení projektu a jeho nahrání do NXT kostky právě dostupným rozhraním USB, nebo Bluetooth.

#### 4.2.2.2 Konfigurace s použitím zásuvného modulu

Použití zásuvného modulu, který byl součástí balíčků LeJOS bylo podporováno do verze 0.9.0. Od verze 0.9.1 již není podporován a je doporučeno použít nástroj Ank popsaný v 4.2.2.1 [3]. Přesto může být pro někoho tento přístup vhodnější, proto je v této práci uveden.

Zásuvný modul je umístěn ve složce, která se vytvořila během instalace LeJOS

*leJOSNXJProjects\NXJPlugin\build\nxjplugin.nbm*

Jelikož od verze 0.9.1 není zásuvný modul podporován, instalační balíčky od této verze zásuvný modul neobsahují. Z archívu lze ovšem stáhnout starší verzi a modul tak získat i pro použití na novější verzi.

Zásuvný modul do NetBeans přidáme přes menu *Tools -> Plugins -> Záložka Download -> Add Plugins.* Po přidání zásuvného modulu by měla být přístupná volba pro vytvoření NXT projektu. *File -> New Project… -> Categories: Sample -> Projects: NXT Project.*

Posledním krokem před používáním této varianty je připojení balíčku LeJOS tříd pro práci s NXT kostkou. Připojení knihovny se provádí v nastavení projektu přes kontextové menu *Properties -> Záložka Java Source Classpath -> Add JAR/Folder.* Knihovna, která se má připojit je umístěna ve složce kam byl nainstalován LeJOS: *\lib\nxt\classes.jar.*

## 4.3 Oživení robota

Tato kapitola popisuje jak přivést robota k životu. V první části se zabývá tím, jaké třídy a jakým způsobem jsou zodpovědné za to, že se robot dokáže pohybovat a provádět případné měření svými senzory. A ve druhé části je popsaný princip komunikace robota s řídicí stanicí. Jak robot navazuje spojení s řídící stanicí, přijímá příkazy, které vykonává a odesílá zpětnou vazbu zpátky stanici.

### 4.3.1 Pohybové a měřící funkce robota

Pohyb robota v prostoru zajišťují dvě samostatně poháněná kola. Proto, aby motory reagovaly správně na zadané příkazy je potřeba před samotným spuštěním robota, znát některé jeho fyzické parametry, především jeho velikost a rozvor kol. Dále je potřeba nastavit do jakých portů, které jsou umístěné na NXT kostce jsou zapojeny jaké senzory, nebo motory. Pro potřeby tohoto nastavení, těchto konstant je vytvořena třída *Setting* v balíčku *nxt.control*.

Ukázka nastavení konstant:

```
public final class Setting {
        //...Senzory
        public static UltrasonicSensor TOP_LENGTH_ULTRA_SONIC
                = new UltrasonicSensor(SensorPort.S1);
        //...Motory
        public static NXTRegulatedMotor TOWER_MOTOR = Motor.B;
        public static NXTRegulatedMotor LEFT_MOTOR = Motor.C;
        public static NXTRegulatedMotor RIGHT_MOTOR = Motor.A;
        //...Nastaveni konstant pro řízení
        public static float WHEEL_DIAMETER = new Float(5.6);
        public static float TRACK_WIDTH = new Float(11.0);
        public static int SPEED_ROTATE_TOWER = 1200;
        public static int ACCELERATION = 50;
        public static int ROTATE_SPEED = 100;
        public static int TRAVEL_SPEED = 150;
        ….
        ….
```

Pro ovládání dvou motorů, kde každý motor ovládá jedno kolo, slouží třída *MotorsControl* z balíčku *nxt.control*. Třída obsahuje nastavení pilota, které proběhne při inicializaci (konstrukci) třídy a metody pro rovnou jízdu a otáčení se. Pilot je třída z knihovny LeJOS určená k ovládání robota, který pro svůj pohon a řízení používá dvě nezávisle poháněná kola.

Příklad inicializace pilota ve třídě ovládající pohon *MotorsControl*:

```
public MotorsControl() {
        this.pilot = new CompassPilot(Setting.COMPASS, Setting.WHEEL_DIAMETER,
                Setting.TRACK_WIDTH, Setting.LEFT_MOTOR,
                Setting.RIGHT_MOTOR);
        this.pilot.setAcceleration(Setting.ACCELERATION);
        this.pilot.setRotateSpeed(Setting.ROTATE_SPEED);
        this.pilot.setTravelSpeed(Setting.TRAVEL_SPEED);
}
```

Poté, co je vytvořena instance kompas pilot, s parametry získanými z nastavení, které je popsané v předcházejícím textu, jsou nastaveny ještě další konstanty určující zrychlení, rychlost otáčení a cestovní rychlost. *CompassPilot* je jednou z implementací třídy *DifferentialPilot*, která využívá kompas k přesnějšímu vykonávání příkazů.

Třída *MotorsControl* ještě obsahuje metody pro jízdu vpřed a otáčení. Tyto metody pouze volají metody obsažené ve třídě *DifferentialPilot*.

Metoda pro jízdu vpřed:

```
public void travel(double distance) {
        this.pilot.travel(distance);
}
```

Atribut *distance* udává jakou vzdálenost má robot ujet. Vzdálenost se zadává v centimetrech, a pokud se zadá záporná hodnota robot jede pozadu.

Metoda pro otáčení robota:

```
public void rotate(double value) {
        this.pilot.rotate(value);
}
```

Atribut *value* udává o kolik stupňů se robot otočit.

Robot disponuje senzorickou věží, která se otáčí a na vrcholu je ultrazvukový senzor. Pohyb věže zajišťuje jeden motor, který přes převody pohybuje otočnou věží. Řízení tohoto motoru má na starost třída *TowerControl* z balíčku *nxt.control.* Tato třída přistupuje přímo k motoru, který otáčí věží. Při vytvoření se nastaví rychlost otáčení a poté se již jen určuje o kolik se má věž otočit. U otáčení věže je potřeba dát pozor na to, že se kvůli kabelu k senzoru může otočit jen o určitý úhel.

Nastavení rychlosti otáčení:

```
public TowerControl() {
        this.towerMotor.setSpeed(Setting.SPEED_ROTATE_TOWER);
}
```

Rychlost se nastaví podle hodnoty, která je zadaná v nastavení.

Otočení věže o určitý počet stupňů:

```
public void rotate(int angle) {
        this.towerMotor.rotate(angle);
}
```

Po každém otočení senzorové věže se provádí měření ultrazvukovým senzorem. Měření probíhá tak, že se nejprve získá přístup k senzoru, umístěném na senzorické věži a pak se z tohoto senzoru získá vzdálenost k překážce. Hodnoty získané měřením jsou v centimetrech a maximální hodnota je 255.

Ukázka měření ultrazvukovým senzorem:

```
UltrasonicSensor ultraSonic = Setting.TOP_LENGTH_ULTRA_SONIC;
distance = ultraSonic.getDistance();
```

### 4.3.2 Komunikace robota s řídící stanicí

Protože výpočty, které se provádí při lokalizaci robota jsou příliš náročné na to, aby se prováděly přímo v NXT kostce, je nutná komunikace s řídicí stanicí. Řídicí stanice posílá povely, které má robot vykonat a robot zpět posílá informace o tom, zda požadovaný povel vykonal a data naměřené senzory.

Ke komunikaci mezi robotem a řídicí stanicí je použito rozhraní bluetooth. Řídicí stanice běží na platformě Android a je použit programovací jazyk Java. Díky použití programovacího jazyku Java se dají použít stejné postupy pro komunikaci i v programech, které nejsou určené pro platformu Android.

V následujícím textu je popsán princip komunikace mezi robotem a řídicí stanicí. Řídicí stanice může být jakékoliv zařízení, které podporuje soketovou komunikaci a rozhraní bluetooth.

#### 4.3.2.1 Navázání a ukončení spojení

Při navázání spojení mezi řídicí stanicí a robotem se vytvoří vstupní a výstupní datové proudy a prostřednictvím nich se provádí výměna informací mezi zařízeními. Práce s datovými proudy v Androidu je podobná jako v klasické Java SE, ale je třeba si dát pozor na některé drobnosti. Například v Androidu nelze pracovat se síťovými proudy v hlavním vlákně.

Spojení se vytváří podle modelu klient-server, to znamená, že jedno zařízená připraví spojení a čeká na připojení klientského zařízení. V této práci stranu, která čeká na spojení, zaujímá robot. Robot po spuštění čeká na připojení řídicí stanice.

Pro správu spojení a datových proudů pro NXT kostku je určená třída *Connection* v balíčku *nxt.connection*. Tato třída zodpovídá za vytvoření a ukončení spojení, také zodpovídá za příjímání příkazů a odesílání informačních dat.

Navázání bluetooth spojení na NXT kostce:

```
private void connection() {
        Display.getSingleton().printLCD("Cekam na spojeni.");
        while (this.btConnect == null) {
                this.btConnect = Bluetooth.waitForConnection();
        }
        this.openStream();
        Display.getSingleton().printLCD("Pripojeni navazano.");
}
```

Metoda *connection* čeká na vytvoření spojení od řídicí stanice. Po vytvoření spojení volá metodu *openStream*, která otevře vstupní a výstupní proudy pro následnou komunikaci.

Ukončení bluetooth spojení na NXT kostce:

```
private void disconnect() {
        this.closeStream();
        this.btConnect.close();
        this.btConnect = null;
        Display.getSingleton().printLCD("Odpojeno");
}
```

Metoda *disconnect* provede uzavření vstupních a výstupních proudů a ukončí bluetooth spojení.

Pro správu spojení a datových proudů pro řídící stanici slouží třída *Connection* v balíčku *pc.connection*. Tato třída zodpovídá za vytvoření a ukončení bluetooth spojení, také zodpovídá za odesílání příkazů a příjem senzorických dat.

Navázání bluetooth spojení pro řídící stanici:

```
public boolean connect() {
        if (!this.connect.connectTo("btspp://")) {
                System.err.println("Spojení s NXT se nepodařilo.");
                this.connected = false;
        } else {
                System.out.println("Spojení s NXT bylo navázáno.");
                this.connected = true;
                this.openStream();
                new Thread(this).start();
        }
        return this.connected;
}
```

Metoda *connect* se pokusí vytvořit spojení s NXT, pokud se spojení povede, zavolá metodu *openStream*, která otevře vstupní a výstupní proudy pro komunikaci. Dále se spustí v novém vlákně rutina, která přijímá data z NXT kostky.

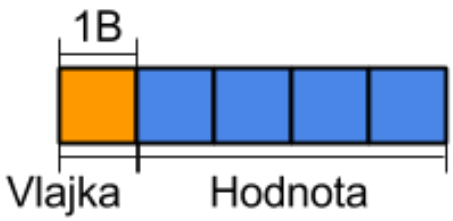
Ukončení bluetooth spojení pro řídicí stanici:

```
public void disconnect() {
        try {
                this.closeStream();
                this.connect.close();
                this.connected = false;
        } catch (IOException ex) {
                System.err.println(ex.getMessage());
        }
}
```

Metoda *disconnect* uzavře vstupní a výstupní proudy a ukončí bluetooth spojení.

#### 4.3.2.2 Odesílání a příjem povelů

Pro odesílání a příjem dat mezi zařízeními se používá otevřených datových proudů. Otevření datových proudů je ukázáno v předcházející kapitole. Pro účely komunikace mezi robotem a řídicí stanicí je vytvořen komunikační balíček jménem *Packet* z balíčku *nxt.control.packet*, který je poté posílán datovými proudy jako sekvence bajtů. Protože je třída *Packet* posílána a přijímána v podobě sekvence bajtů, je nutné, aby bylo možné třídu převést na tuto sekvenci nebo jí ze sekvence znovu sestavit. Komunikační balíček je tvořen informační vlajkou a nějakou užitnou hodnotou. Informační vlajka je číslo od 0 do 255, které určuje význam komunikačního balíčku, přehled všech vlajek a jejich význam je v Tab. 4.3.1. Informační vlajka zabírá první bajt celého komunikačního balíčku. Užitná hodnota je reálné číslo vyjadřující, nějakou hodnotu vztahující se k významu informační vlajky. Užitná hodnota zabírá poslední čtyři bajty. Rozložení informací v balíčku je znázorněno na Obr. 4.3.1.




**Obr. 4.3.1 – Struktura informačního balíčku.**

| Hodnota informační vlajky | Význam | Význam užitné hodnoty |
|---|---|---|
| **0** | Příkaz:<br>Vypnutí robota. | Žádný. |
| **1** | Příkaz:<br>Otoční robota. | Počet stupňů. |
| **2** | Příkaz:<br>Jízda vpřed. | Vzdálenost. |
| **10** | Příkaz:<br>Otočení senzorické věže. | Počet stupňů. |
| **9** | Senzorické měření:<br>Ultrazvukový senzor. | Vzdálenost od překážky. |
| **11** | Potvrzení příkazu:<br>Jízda vpřed. | Žádný. |
| **12** | Potvrzení příkazu:<br>Otočení senzorické věže. | Žádný. |
| **13** | Potvrzení příkazu:<br>Otočení robota. | Žádný. |

**Tab. 4.3.1 – Význam hodnot informačních vlajek.**

Třída *Packet* má dva konstruktory. První konstruktor vytvoří třídu ze zadané hodnoty informační vlajky a užitné hodnoty. Druhý konstruktor ze zadané sekvence pěti bajtů zrekonstruuje třídu.

Ukázka konstruktorů třídy *Packet*, která reprezentuje komunikační balíček:

```
public Packet(byte flag, double value) {
        this.flag = flag;
        this.value = value;
}
public Packet(byte[] bytes) {
        byte[] data = new byte[4];
        this.flag = bytes[0];
        System.arraycopy(bytes, 1, data, 0, 4);
        this.value = Convert.fourBytesToReal(data);
}
```

Pro převedení informací obsažených ve třídě *Packet* slouží metoda *toByte*:

```
public byte[] toBytes() {
        byte[] result = new byte[5];
        result[0] = this.flag;
        byte[] data = new byte[4];
        data = Convert.realToFourBytes(this.value);
        System.arraycopy(data, 0, result, 1, 4);
        return result;
}
```

## 4.4 Implementace částicového filtru

Jak bylo popsáno v kapitole 3.2 a kapitole 3.4.2.2, lokalizace částicovým filtrem se skládá ze tří kroků. Prvním krokem je počáteční odhad polohy. Další dva kroky jsou apriorní odhad polohy a aposteriorní odhad polohy, tyto dva kroky se interaktivně střídají. V následující kapitole je popsáno, jak jsou tyto kroky implementovány pro účely této práce. Také je zde popsána částice a model prostoru, ve kterém se provádí lokalizace.

### 4.4.1 Částice

Částice je základní prvek částicového filtru. Obsahuje informace o své poloze v prostoru, natočení a o váze významnosti. Třída zodpovědná za uchování těchto informací se jmenuje *Particle* a je umístěna v balíčku *particlefilter.particle*.

Třída není zodpovědná pouze za uchování informací, ale obsahuje i některé metody pro manipulaci s částicí. Nejdůležitější jsou metody pro pohyb částice, zjištění dosahu měření, provedení očekávaného měření a výpočet váhy významnosti.

Metody pro pohyb částice v prostoru:

```
public Particle move(double angle, int distance) {
        double newRotation = this.rotation + angle;
        newRotation %= 2.0 * Math.PI;
        Point newPosition = this.newPosition(this.position, newRotation, distance);
        return new Particle(newPosition, newRotation);
}

private Point newPosition(Point position, double angle, int distance) {
        int x = position.x + (int) Math.round(Math.cos(angle) * distance);
        int y = position.y + (int) Math.round(Math.sin(angle) * distance);
        return new Point(x, y);
}
```

Metoda *move* vytvoří novou částici, který se vypočítá na základě zadaného úhlu a vzdálenosti.

Další důležitou metodou je určení dosahu měření. Tato metoda vypočítá obdélník, který reprezentuje oblast možného měření. Na základě této oblasti se rozhodne, jaké

orientační body budou součástí měření a jaké ne. Tato oblast závisí na umístění částice v prostoru a dosahu senzorů.

Metody pro určení oblasti možného měření:

```
public Viewport createParticleRectangle(int range) {
        return this.createParticleRectangleParticleRobot(range);
}

private Viewport createParticleRectangleParticleRobot(int range) {
        Point point3 = this.newPosition(this.position, this.rotation - Math.PI / 2.0, range);
        Point point1 = this.newPosition(point3, this.rotation, range);
        Point point4 = this.newPosition(this.position, this.rotation + Math.PI / 2.0, range);
        Point point2 = this.newPosition(point4, this.rotation, range);
        Point center = this.newPosition(this.position, this.rotation, range / 2);
        return new Viewport(center, point1, point2, point3, point4);
}
```

Svět je implementován ve třídě *World* v balíčku *particlefilter.world*. Tato třída uchovává informace o modelu prostředí, ve kterém se lokalizuje, a také obsahuje metodu *getLandmarksInRectangle*. Tato metoda na základě oblasti měření vrací seznam orientačních bodů.

Ukázka metody *getLandmarksInRectangle*:

```
public List<Landmark> getLandmarksInRectangle(Viewport viewport, Particle particle) {
        List<Landmark> landmarksInRectangle = new ArrayList<>();
        Map<Double, Landmark> map = new TreeMap<>();
        for (Landmark landmark : this.landmarks) {
        if (landmark.contains(viewport)) {
                landmarksInRectangle.add(landmark);
                }
        }
        …
        return landmarksInRectangle;
}
```

Další důležitou metodou ve třídě *Particle* je metoda *sense*, která provádí předpokládané měření. Toto měření je důležité pro vypočet váhy významnosti. Předpokládané měření je vlastně změření vzdáleností ke všem orientačním bodům.

Ukázka předpokládaného měření:

```
public List<Double> sense(List<Landmark> landmarks) {
        List<Double> senseValues = new ArrayList<>();
        for (Landmark landmark : landmarks) {
                senseValues.add(landmark.distance(this.position));
        }
        return senseValues;
}
```

Poslední důležitou funkcí částice je výpočet váhy významnosti a její normalizace. Váha významnosti se používá při aposteriorním odhadu.

Metody pro výpočet a normalizaci váhy významnosti:

```
public void updateWeight(List<Double> sense, List<Double> measurement) {
        double prob = 1.0;
        if (sense.size() != measurement.size()) {
                prob = Math.exp(-100);
        }
        for (int i = 0; i < Math.min(sense.size(), measurement.size()); i++) {
                prob *= calculeProb(sense.get(i), measurement.get(i));
        }
        this.weight = prob;
}
private double calculeProb(double x, double y) {
        return Math.exp(-Math.pow(x - y, 2));
}
public void normalizeWeights(double sumWeights) {
        this.weight /= sumWeights;
}
```

### 4.4.2 Inicializace – Počáteční odhad polohy

Při vytváření částicového filtru je nutné definovat, několik parametrů. Patří mezi ně především počet částic, používaných pro lokalizaci a model prostoru, neboli svět, ve kterém se lokalizace provádí.

Po získání potřebných parametrů, se provede počáteční odhad polohy a tím se částicový filtr inicializuje. Počáteční odhad polohy se nastavuje tak, že robot je se stejnou pravděpodobností kdekoliv v prostoru.

Inicializace částicového filtru (obsahuje třída *particlefilter.ParticleFilter*):

```
private void initParticles() {
        for (int i = 0; i < this.countParicles; i++) {
                this.particles.add(RandomParticle.createRandomParticle(this.world));
        }
}
```

Při inicializaci se vygeneruje sada náhodných částic. Množství částic bylo definováno při vytváření částicového filtru.

Vytvoření náhodné částice (obsahuje třída *particlefilter.particle.RandomParticle*):

```
public static Particle createRandomParticle(World world) {
        double rotation = random.nextInt(4) * Math.PI/2.0;
        Point location = createRandomLocation(world);
        return new Particle(location, rotation);
}
```

Náhodně se vygeneruje natočení částice do jednoho ze čtyř směrů a náhodné umístění v prostoru, který představuje svět (*world*).

### 4.4.3 Apriorní odhad polohy

Apriorní odhad polohy se provádí na základě relativních měření. Protože robot nemá žádné senzory pro relativní měření, používá se jako relativní měření příkaz k pohybu robota. Jinak řečeno to znamená, že pohyb, který má robot provést, provedou všechny částice. Pro částice se předpokládá cyklický svět, to znamená, že když částice opustí souřadnice vymezené prostorem, ocitne se na začátku druhé strany prostoru.

Pohyb všech částic, které jsou součástí filtru (obsahuje třída *particlefilter.ParticleFilter*):

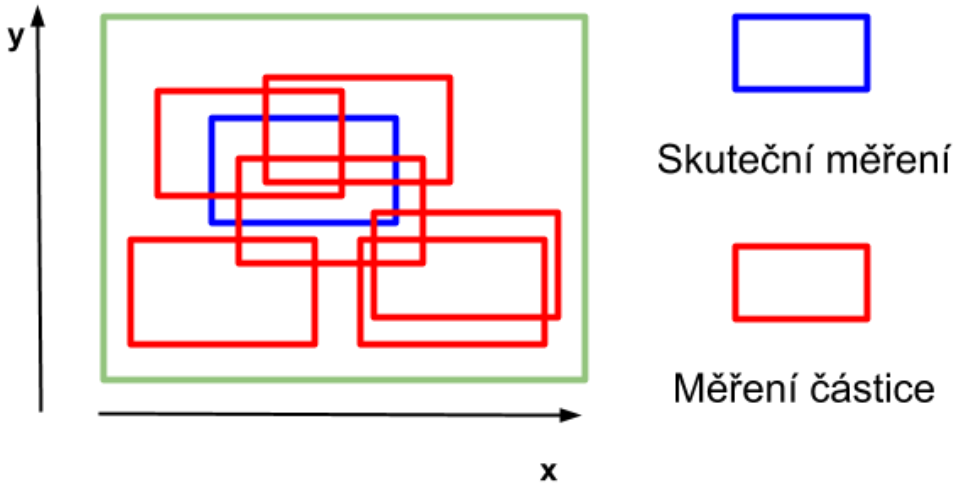
```
public void move(double angle, int distance) {
        for (int i = 0; i < this.countParicles; i++) {
                Particle oldParticle = this.particles.get(i);
                Particle newParticle = oldParticle.move(angle, distance);
                int x = Math.abs(newParticle.getLocation().x);
                int y = Math.abs(newParticle.getLocation().y);
                x %= world.getWorldSize().width;
                y %= world.getWorldSize().height;

                newParticle = new Particle(new Point(x, y), newParticle.getRotation());
                if (this.world.isValidLocation(newParticle.getLocation())) {
                        this.particles.set(i, newParticle);
                }
        }
}
```

Nejprve se na základě staré částice, směru pohybu a vzdálenosti vytvoří nová částice. U nové částice se přepočítají souřadnice tak, aby částice nebyla mimo vymezený prostor. Poté se částice vloží do nové sady částic.

### 4.4.4 Aposteriorní odhad polohy

Aposteriorní odhad polohy se dá rozdělit do několika kroků. Nejprve se vypočítá váha významnosti pro všechny částice, poté se provede normalizace těchto vah a jako poslední krok se provede převzorkování.

Nejprve se provádí výpočet váhy významnosti. Jak bylo popsáno v kapitole 3.4.2.2 velikost váhy významnosti závisí na rozdílu mezi skutečným měřením a očekávaným měřením. V principu se dá říct, že se porovnávají změřené oblasti prostoru s mapovými podklady. Na Obr. 4.4.1 je znázorněné skutečné umístění provedeného měření a měření simulované jednotlivými částicemi. Čím menší rozdíl mezi jednotlivými měřeními je, tím větší je hodnota váhy významnosti.




**Obr. 4.4.1 – Porovnání senzorického měření.**

Výpočet váhy významnosti (obsahuje třída *particlefilter.ParticleFilter*):

```
public void updateWeight(List<Double> measurement) {
        for (Particle particle : this.particles) {
                Viewport particleRectangle = particle.createParticleRectangle(this.range);
                List<Landmark> landmarks =
                        this.world.getLandmarksInRectangle(particleRectangle, particle);
                List<Double> senseValues = particle.sense(landmarks);
                particle.updateWeight(senseValues, measurement);
        }
}
```

Postupně se prochází přes všechny částice, zjistí se oblast a orientační body, které částice může změřit. K těmto orientačním bodům se provede měření, to se porovná se skutečným měřením a vypočítá se nová váha významnosti.

Dalším krokem aposteriorního odhadu polohy je normalizovat všechny váhy významnosti. To se provede tak, že se jednotlivé váhy vydělí sumou věch vah významnosti. Normalizovaná váha významnosti určuje, s jakou pravděpodobností se částice dostane do další sady částic a její vypočet je nezbytnou podmínkou pro provedení operace převzorkování.

Normalizace váhy významnosti (obsahuje třída *particlefilter.ParticleFiletr*):

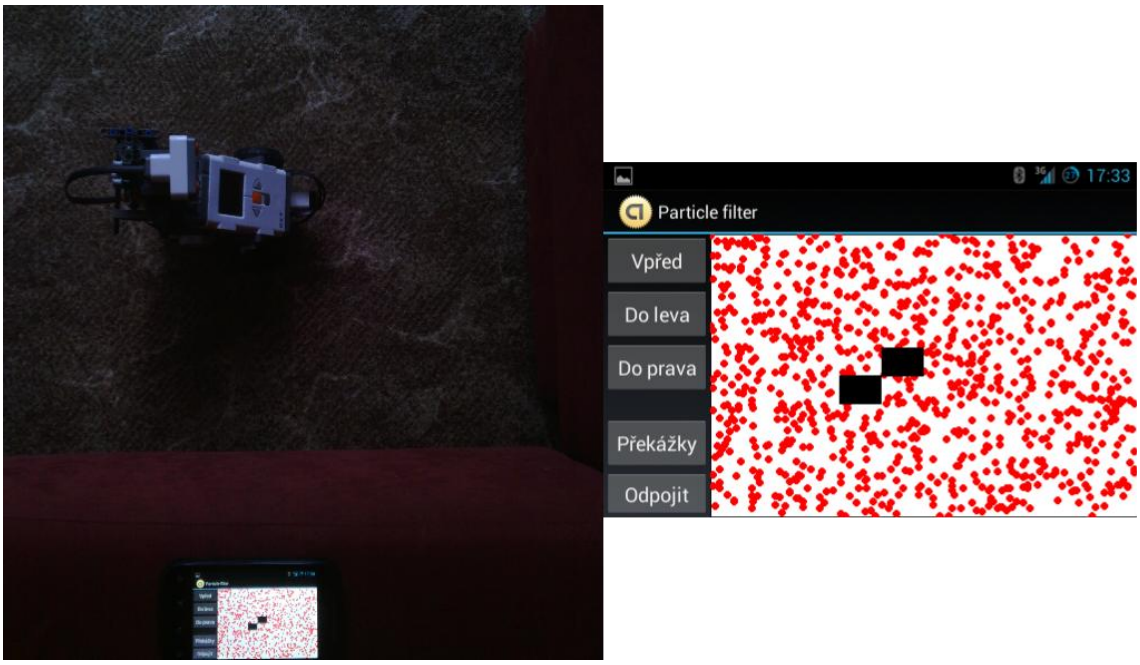
```
public void normalizeWeights() {
	double sumWeights = sumWeigths();
	for (Particle particle : this.particles) {
		particle.normalizeWeights(sumWeights);
	}
}
private double sumWeigths() {
	double sumWeights = 0;
	for (Particle particle : this.particles) {
		sumWeights += particle.getWeight();
	}
	return sumWeights;
}
```

Posledním krokem aposteriorního odhadu polohy je převzorkování. Převzorkování vytvoří novou sadu částic, která nahradí doposud používanou sadu. Do nové sady částic se může dostat kterákoliv částice z předchozí sady a to dokonce i ve více kopiích.

```
public static List<Particle> roulette(List<Particle> oldItems) {
	List<Particle> newItems = new ArrayList<>();
	int index = new Random().nextInt(oldItems.size());
	double beta = 0.0;
	double max = max(oldItems);
	for (Particle oldItem : oldItems) {
		beta += Math.random() * 2.0 * max;
		while (beta > oldItems.get(index).getWeight()) {
			beta -= oldItems.get(index).getWeight();
			index = (index + 1) % oldItems.size();
		}
	newItems.add(oldItems.get(index));
	}
	return newItems;
}
```

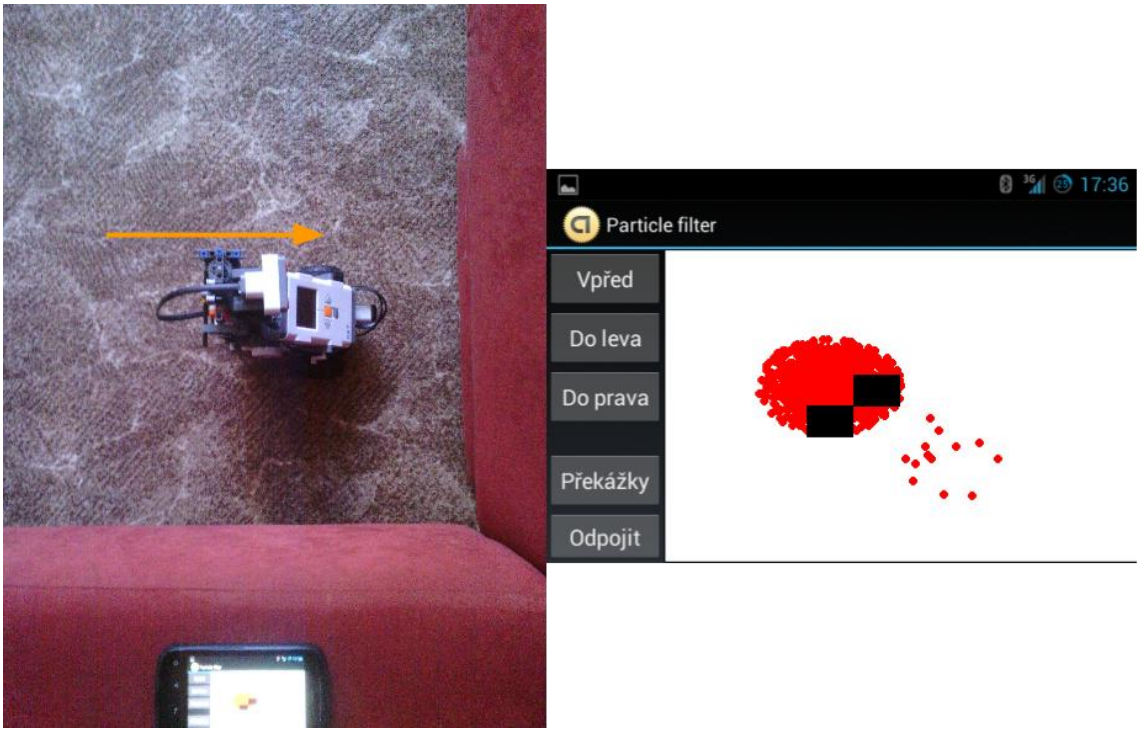
### 4.4.5 Ukázka fungování programu

V počátečním stavu není umístění robota známé, proto jsou částice roztroušené po celém prostoru.



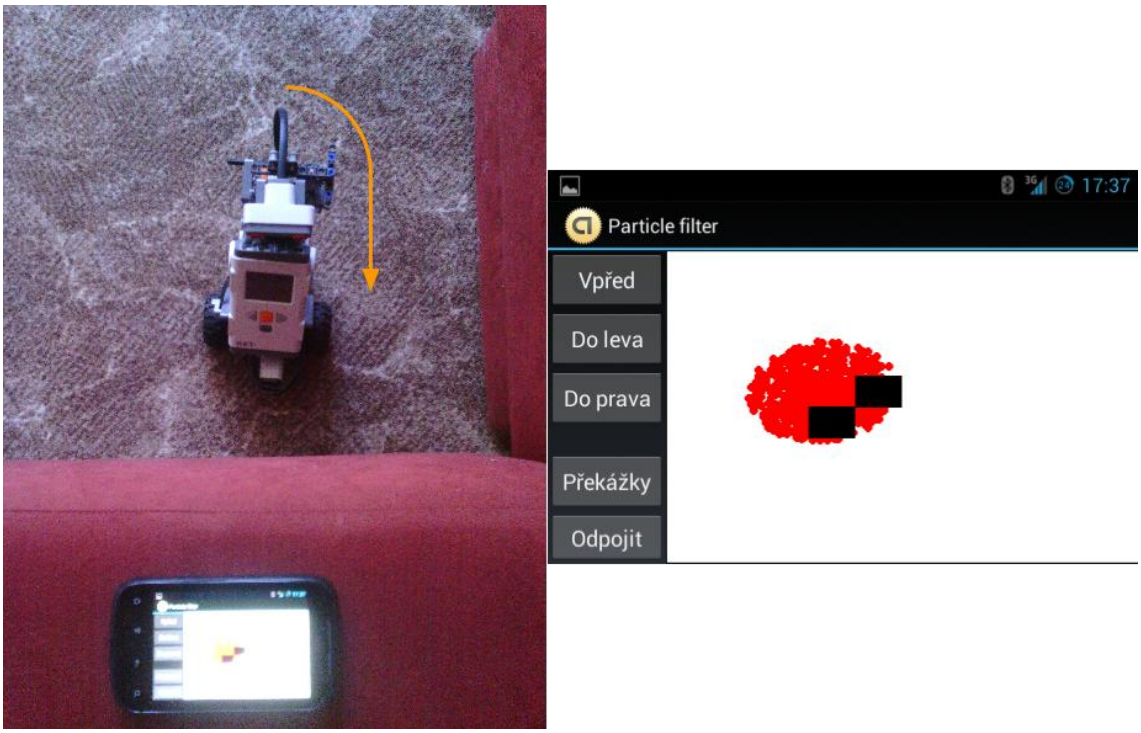

**Obr. 4.4.2 – Ukázka: počáteční stav.**

Ihned po prvním kroku je určen téměř přesný odhad polohy.



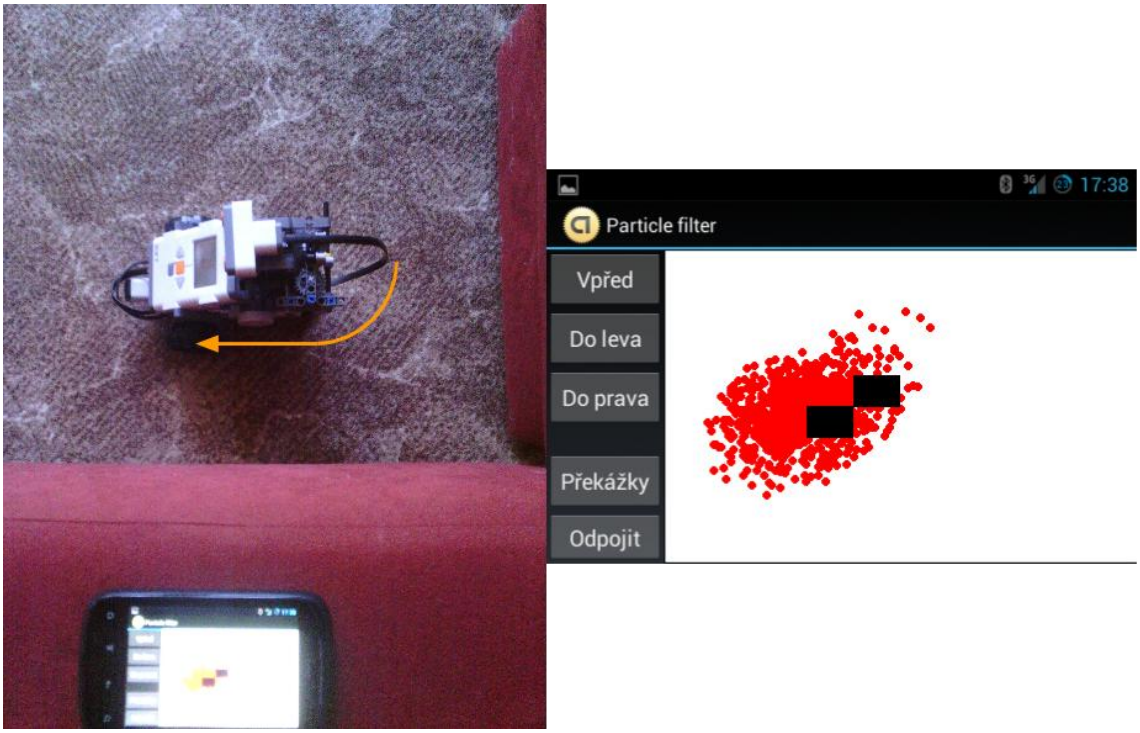

**Obr. 4.4.3 – Ukázka: první krok.**

Další měření polohu ještě více upřesní.



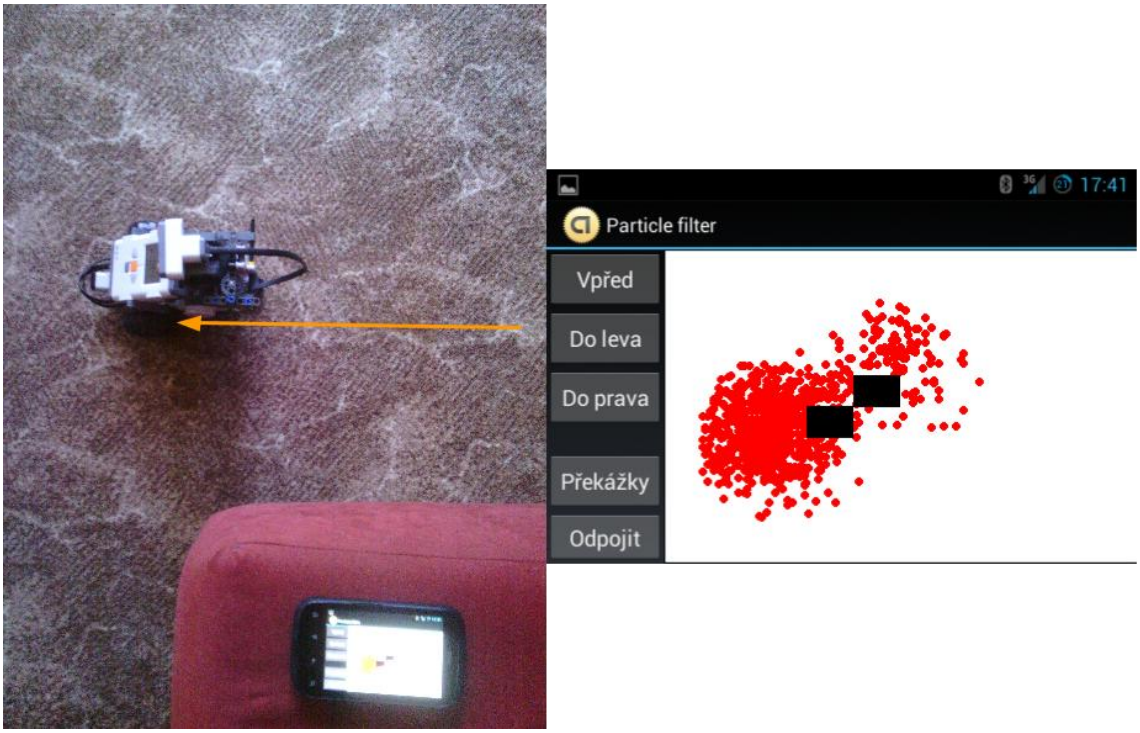

**Obr. 4.4.4 – Ukázka: druhý krok.**

V dalším kroku se robot otočí zády k jednomu z orientačních bodů, což způsobí, že ho není schopen zaměřit a tím se odhad polohy zhoršuje.



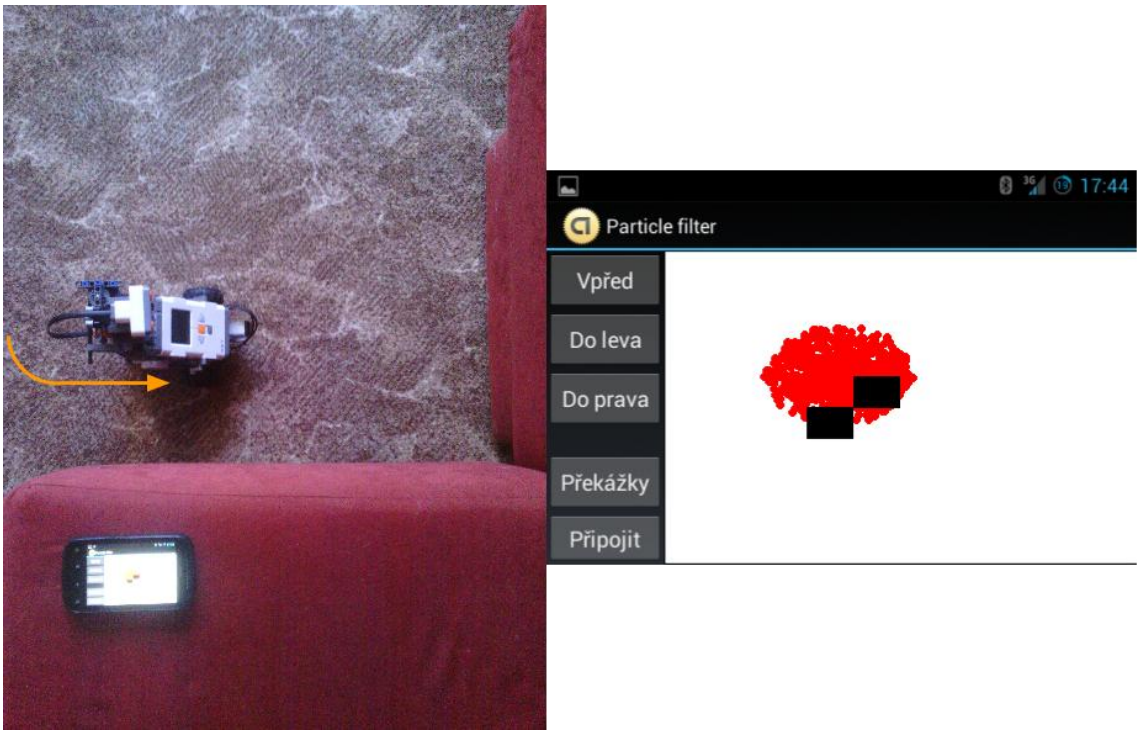

**Obr. 4.4.5 – Ukázka: třetí krok.**

Při dalším kroku již není v dosahu senzorů žádný orientační bod a odhad polohy se stále zhoršuje.




**Obr. 4.4.6 – Ukázka: čtvrtý krok.**

Při otočení je robot schopen opět zaměřit oba orientační body a proto se odhad polohy opět upřesní.




**Obr. 4.4.7 – Ukázka: patý krok.**

# Závěr

Téma této diplomové práce jsem si vybral po absolvování online kurzu *Introduction to Artificial Intelligence* pod vedením Petra Norviga a Sebastiana Thruna. Tento kurz mě velice nadchl, a proto jsem využil možnost realizace některé lokalizační techniky na reálném robotovi. Tématem práce bylo sestavit robota a demonstrovat na něm funkčnost některé lokalizační metody. Za demonstrovanou lokalizační metodu byl vybrán částicový filtr.

Robot byl sestaven ze stavebnice LEGO Mindstorms Education. Základním prvkem stavebnice je programovatelná kostka NXT, která je srdcem, nebo spíše mozkem robota. Ke kostce lze připojit a následně ovládat celou řadu periferií. Tato stavebnice je vhodným a především dobře dostupným způsobem jak simulovat reálné situace, které by se za jiných situací musely modelovat pouze teoreticky.

Lokalizace robota pomocí částicového filtru ve zmapovaném prostoru byla úspěšná, ovšem přesnost odhadu polohy po dlouhodobějším používání značně klesá. Tyto nepřesnosti jsou zapříčiněny nedostatečným množstvím senzorických dat. Robot sestavený v rámci této práce získá při jednom měření senzorická data ze tří směrů, oproti tomu zařízení, které tuto metodu lokalizace běžně používají, disponují senzory provádějící měření v okruhu 360°. Takovýto senzor pro NXT kostku bohužel nebyl dostupný.

Princip částicového filtru je pro mě osobně velice zajímavý, díky jeho využití i v jiných oblastech než lokalizace robota předpokládám, že ho i nadále budu používat ve svých budoucích pracovních i volnočasových projektech.

Součástí diplomové práce je uživatelská příručka popisující práci s vytvořenými programy. Kompletní zdrojové kódy a kompletní návod na sestavení robota je umístěno na přiloženém CD.

# Literatura


[1] BORENSTEIN, J; EVERETT, H. R.; FENGL, L. *Where Am I? Sensors and Methods for Mobile Robot Positioning.* University of Michigan, 1996.

[2] THRUN, S. *Particle Filters in Robotics. In proceedings of Uncertainty in AI.* 2002.

[3] LeJOS, *LeJOS, Java for LEGO Mindstorms.* [online]. 1997-2009 [cit. 2013-07-15]. Dostupné z: http://lejos.sourceforge.net/

[4] NORVIG, P.; THRUN S. *Introduction to Artificial Intelligence.* Udacity, [online]. [cit. 2013-07-05]. Dostupné z: https://www.udacity.com/course/cs271

[5] THRUN S. *Artificial Intelligence for Robotics.* Udacity, [online]. [cit. 2013-07-05]. Dostupné z: https://www.udacity.com/course/cs373

[6] THE LEGO GROUP. *Lego.com,* [online]. [cit. 2013-07-10]. Dostupné z: http://www.lego.com/cs-cz/

[7] THE LEGO GROUP. *Lego Mindstorms,* [online]. [cit. 2013-07-20]. Dostupné z: http://mindstorms.lego.com/en-us/default.aspx

[8] LEGOENGINEERING. *Legoengineering.com,* [online]. [cit. 2013-08-3]. Dostupné z: http://www.legoengineering.com/

[9] ORACLE, *Java SE,* [online] [cit. 2013-07-28]. Dostupné z: http://www.oracle.com/technetwork/java/javase/downloads/index.html

[10] ORACLE, *NetBeans,* [online] [cit. 2013-08-4]. Dostupné z: https://netbeans.org/

[11] ANDROID, *Android developers,* [online] [cit. 2013-06-28]. Dostupné z: https://developer.android.com/training/index.html

[12] GOOGLE, *Translator google,* [online] [cit. 2013-07-13]. Dostupné z: http://translate.google.cz/

[13] NEGENBORN, R., *Robot Localization and Kalman Filters: On finding your position in a noisy word.* Utrecht, 2002. Utrecht University.

# Příloha A – Uživatelská příručka – Robot

Ovládání NXT kostky s firmwarem LeJOS a programem vytvořeným v rámci této práce je velice jednoduché. Kostka obsahuje čtyři tlačítka. Oranžové tlačítko uprostřed kostky slouží pro potvrzování příkazů a označuje se jako ENTER. Šedivé tlačítko umístěné pod tlačítkem ENTER slouží k návratu z vnořených nabídek a k vypnutí kostky je označováno jako ESCAPE. Dvě šedá tlačítka ve tvaru šipek směřující doleva a doprava, slouží k listování v nabídkách.

Pro spuštění programu se musí nejprve aktivovat kostka. To se provede stisknutím tlačítka ENTER. V zobrazeném seznamu položek je nutné nalistovat položku „Files“, která obsahuje všechny nahrané programy, položka se potvrdí stisknutím tlačítka ENTER. V zobrazeném seznamu programů je třeba nalistovat program jménem „Demo.nxj“ a vybrat stisknutím ENTER. Otevře se seznam možností, které se mohou provést se zvoleným programem. Program se může nastavit jako výchozí program pro spuštění (Set as Default), může se také vymazat (Delete file) a nakonec se může spustit (Run).

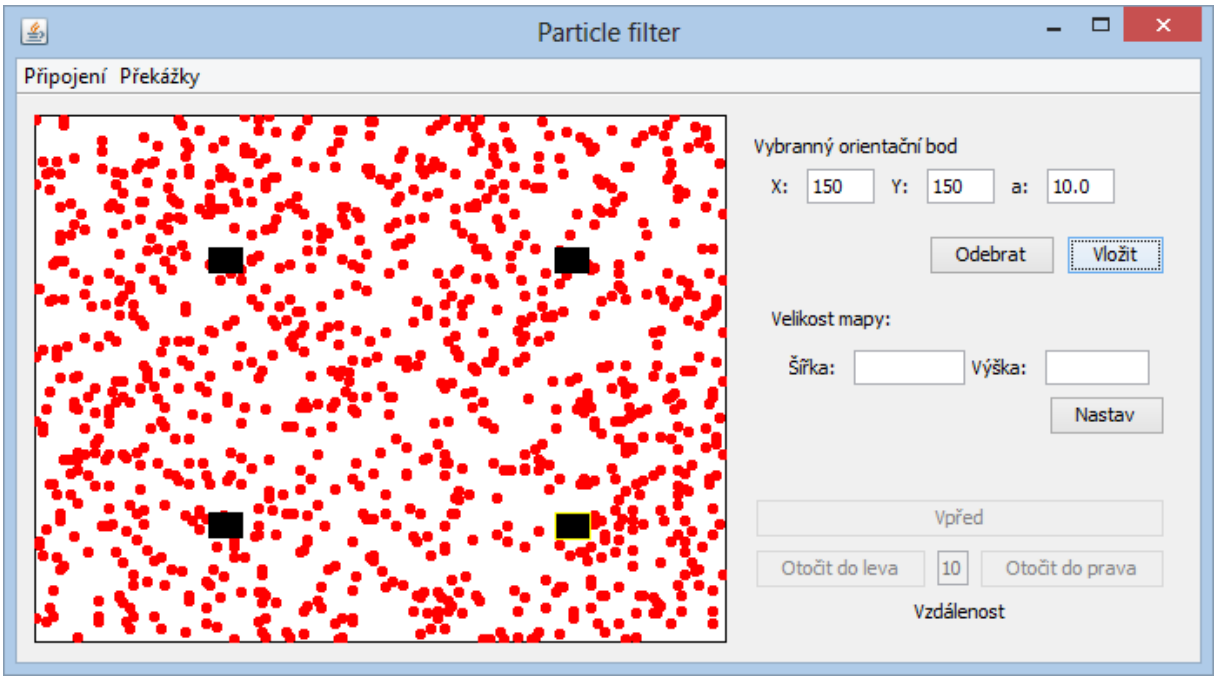
Po spuštění programu se na displeji NXT kostky objeví nápis „Cekam na spojeni.“. Tím je robot připraven na spojení s řídicí stanicí. Po navázání spojení se ozve zvukový signál a na displeji se zobrazí nápis „Pripojeni navázáno“. Robot při přijetí jakéhokoliv příkazu tuto skutečnost oznámí zvukovým znamením.

Pro vypnutí programu stačí stisknout tlačítko ESCAPE. Pokud program z jakéhokoliv důvodu „zamrzne“ je nutné vyndat a zandat baterku a tak robota restartovat.

# Příloha B – Uživatelská příručka – řídicí stanice – PC

Jak již bylo řečeno v předchozím textu, robot se připojí k řídicí stanici. Řídicí stanice odesílá příkazy, které má robot splnit a přijímá senzorická data, která robot naměřil. Program řídicí stanice je implementován pro dvě platformy. První platformou je PC.

Před prvním spuštěním aplikace je nutné spárovat počítač s NXT kostkou. Po spuštění řídicí stanice se otevře okno aplikace, které můžete vidět na následujícím obrázku.




**Obr. B.1. – Vzhled programu řídicí stanice PC.**

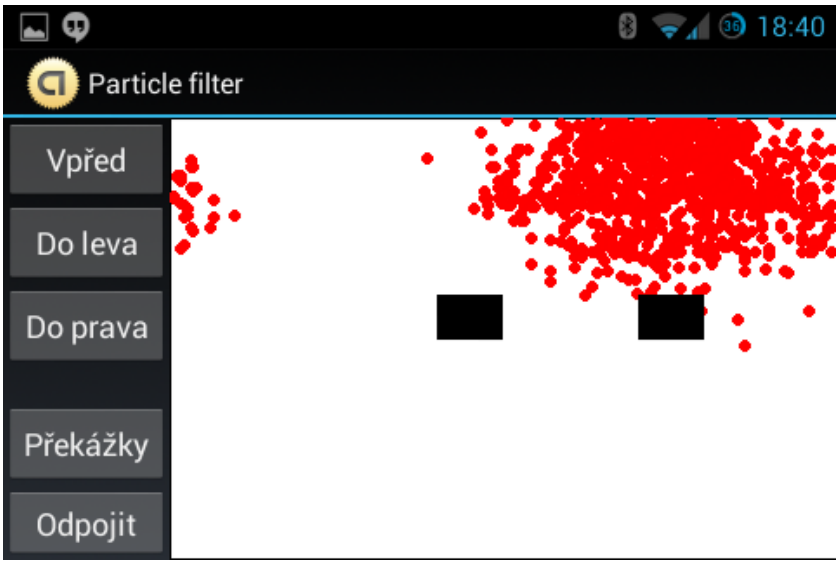
Okno programu lze rozdělit do čtyř částí. První a největší část vykresluje mapu prostoru a jednotlivé částice. Druhá část je vpravo v horní části okna, obsahuje možnost pro vložení nebo odebrání orientačních bodů a možnost nastavení velikostí prostoru. Pro vložení nového orientačního bodu stačí vyplnit souřadnice středu čtverce a délku jeho strany. Pro odebrání orientačního bodu stačí označit kliknutím na jeho umístění na mapě a následném stisknutí tlačítka odebrat.

Třetí částí je ovládání robota, to je umístěno vpravo dole. Obsahuje tlačítka otáčející robota doleva nebo doprava a tlačítko VPŘED, které posílá příkaz o rovné jízdě. Vzdálenost, kterou má robot ujet se zadává do políčka pod tlačítkem vpřed. Všechny hodnoty jsou zadávané v centimetrech.

Poslední sadou ovládacích prvků jsou nabídky *Připojení* a *Překážky*. V nabídce *připojení* lze provést připojení k robotu či jeho odpojení. V nabídce *překážky* lze aktuální sadu překážek uložit do souboru nebo ze souboru načíst dříve uložené.

# Příloha C – Uživatelská příručka – řídicí stanice – Android

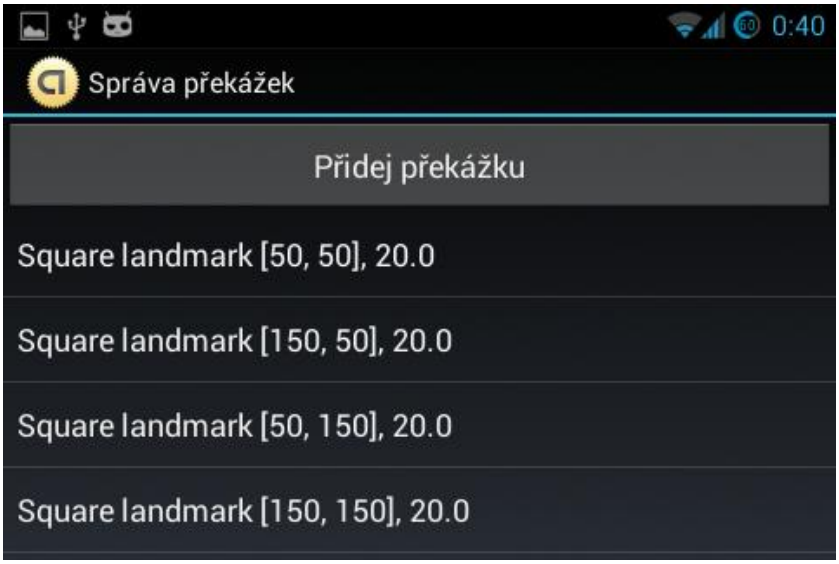
Další možnou řídicí stanicí je stanice fungující na platformě Android. Před prvním spuštěním aplikace je nutné spárovat Android zařízení s NXT kostkou. Po spuštění této řídicí stanice se objeví následující obrazovka.




**Obr. C.1. – Vzhled programu řídicí stanic Android.**

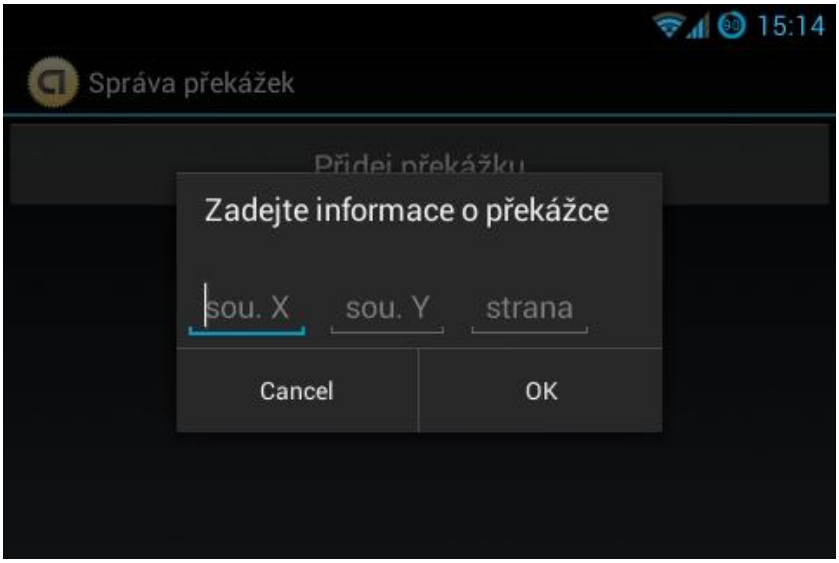
Obrazovku lze rozdělit do dvou částí. Část vpravo zobrazuje aktuální mapu a rozložení částic. Část vlevo obsahuje řídicí tlačítka. Tlačítka jdou opět rozdělit do dvou skupin. První skupinou jsou tlačítka, řídicí pohyb robota a druhá skupina ovládá nastavení překážek (orientačních bodů) a vytváří nebo ruší spojení s robotem.

Po stisknutí tlačítka pro nastavení orientačních bodů se zobrazí další obrazovka, která je vidět na následujícím obrázku.




**Obr. C.2. – Nastavení orientačních bodů.**

Po stisknutí tlačítka přidat překážku se zobrazí dialog, který chce upřesnit informace o novém orientačním bodě. Po zadání požadovaných informací se orientační bod vloží do seznamu. Pro smazání existujícího orientačního bodu stačí dlouze podržet požadovaný orientační bod v seznamu. Zpět na hlavní obrazovku se přesune stisknutím tlačítka zpět.




**Obr. C.3. – Vložení nového orientačního bodu.**

Pro ukončení programu stačí na hlavní obrazovce kliknout na tlačítko zpět.