ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

F3

Fakulta elektrotechnická
Katedra kybernetiky

Bakalářská práce

# Využití robota LEGO Mindstorms - návrh soutěžních úloh pro ROBOSOUTĚŽ

**Dušan Stěhule**

Únor 2019

Vedoucí práce: Ing. Martin Hlinovský, Ph.D.

# ZADÁNÍ BAKALÁŘSKÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Stěhule** Jméno: **Dušan** Osobní číslo: **457225**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra kybernetiky**

Studijní program: **Kybernetika a robotika**

Studijní obor: **Robotika**

## II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

**Využití robota LEGO Mindstorms - návrh soutěžních úloh pro ROBOSOUTĚŽ**

Název bakalářské práce anglicky:

**Usage of the LEGO Mindstorms Robots - Design of the Competition Tasks for ROBORACE**

Pokyny pro vypracování:

1. Seznamte se s možnostmi robota LEGO Mindstorms (současný stav, HW a SW vybavení).
2. Navrhněte a realizujte dvě nové soutěžní úlohy pro LEGO Mindstorms EV3 s návrhem řízení ve Vámi vybraném programovacím prostředí.
3. První soutěžní úloha, s názvem „PAC-MAN“ – úkolem robota je ve stanoveném časovém limitu projet co největší část bludiště, přičemž velikost projeté plochy bude odpovídat počtu získaných bodů. Dále připravte čtyři typy „Duchů“, kteří se budou pohybovat po předem definovaných drahách bludiště včetně návodu na stavbu robota (ducha) a podrobné fotodokumentace.
4. Druhá soutěžní úloha „Sledování černé čáry“ – úkolem robota je sledovat černou čáru na bílém podkladu s připravenými překážkami.
5. Vytvořte webové stránky k navrženým soutěžním úlohám a vypracujte podrobné návody ve formě pravidel k oběma soutěžním úlohám.

Seznam doporučené literatury:

[1] James Floyd Kelly - LEGO MINDSTORMS NXT-G programming Guide, Second Edition
[2] Daniele Benedettelli - Programming LEGO NXT Robots using NXC
[3] https://www.youtube.com/watch?v=STSDa4K7ixI

Jméno a pracoviště vedoucí(ho) bakalářské práce:

**Ing. Martin Hlinovský, Ph.D., katedra řídicí techniky FEL**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **02.08.2018** Termín odevzdání bakalářské práce: **08.01.2019**

Platnost zadání bakalářské práce: **30.09.2019**

| Ing. Martin Hlinovský, Ph.D. | doc. Ing. Tomáš Svoboda, Ph.D. | prof. Ing. Pavel Ripka, CSc. |
|---|---|---|
| podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | podpis děkana(ky) |

## III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

# Poděkování / Prohlášení

Rád bych tímto poděkoval vedoucímu práce Ing. Martinovi Hlinovskému, Ph.D. za cenné rady a vstřícný přístup při tvorbě této práce. Dále bych rád poděkoval své přítelkyni, rodině a přátelům za zázemí a podporu.

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 7.1.2019

.....

# Abstrakt / Abstract

Teoretická část práce seznamuje čtenáře s historií a aktuálním stavem Lego Mindstorms. Porovnává parametry jednotlivých verzí a detailně představuje nejnovější verzi EV3, včetně všech senzorů a softwarových možností.

V praktické části jsou podrobně rozebrány obě navrhované úlohy. V první úloze, s názvem pac-man, jsou nejdříve popsány všechny varianty duchů, konkrétně jejich konstrukce, vzhled, software a možnosti pohybu. Následuje detailní popis konstrukce a řídícího programu robota projíždějícího bludiště. Jsou zde do detailu rozebrány principy funkce programu a hlavní procedury a funkce. V úloze sledování černé čáry je popsána konstrukce robota, jeho software a možnosti, jak nastavit PID regulátor.

V poslední části je popis vytvořených webových stránek s pravidly a závěr.

**Klíčová slova:** LEGO Mindstorms EV3, leJOS, bludiště, pac-man, duchové, BFS, černá čára, PID regulátor, Ziegler-Necholsova metoda

The theoretical part of the work introduce reader with history and actual state of Lego Mindstorms. It compares parameters of each version and in detail introduce the latest version EV3, including all of sensors and software options.

In the practical part two designed tasks are in detail analyzed. In the first task, called pac-man, all variants of ghosts are described, specifically their construction, design, software and possibilities of movement. Follows detailed description of construction and control program of robot, which passes through labyrinth. Principles of program function and main procedures and function are in detail analyzed here. The construction of robot, its software and possibilities and how to set up the PID controller is describes in the second task of tracking black line.

In the last section is description of created web pages with rules and conclusion.

**Keywords:** LEGO Mindstorms EV3, leJOS, labyrinth, pac-man, ghosts, BFS, black line, PID controller, Ziegler-Nichols method

**Title translation:** Bachelor thesis (Usage of the LEGO Mindstorms Robots - Design of the competition tasks for ROBORACE)

# Obsah /

# Tabulky /

# Kapitola 1
# Úvod

Cílem této bakalářské práce je vytvořit stručný přehled hardwarových a softwarových možností Lego Mindstorms a připravit dvě soutěžní úlohy pro robosoutěž. Příprava soutěžních úloh spočívá v přesném stanovení jejich pravidel, vytvoření a naprogramování podpůrných robotů (například duchů v úloze Pac-Man) a vytvoření a naprogramování robota představujícího referenční řešení obou z úloh.

V první úloze, s názvem Pac-Man, je úkolem robota za stanovený časový limit projít co největší část bludiště, a přitom magnetem zhasínat barevná světla umístěná v podložce. Bludiště je možné si představit jako matici čtverců, každý o rozměrech 28 x 28 cm, kde jsou v náhodných čtvercích umístěné překážky. Ty vyplňují celou plochu čtverce a jsou 7.5 cm vysoké. Plocha čtverců, ve kterých nejsou umístěné překážky je souvislá, uprostřed každého čtverce je umístěno světlo o průměru 6.5 cm. V elektronice světla je umístěna Hallova sonda, díky které může světlo detekovat přítomnost magnetického pole. Robot se může pohybovat pouze po čtvercích bez překážek a pomocí přiloženého magnetu zhasíná světla.

V bludišti se současně s robotem pohybují po předem stanovených drahách duchové. Jsou to roboti postavení z Lego Mindstorms, kteří jsou vyšší než překážky (tedy než 7.5 cm) tak, aby je mohli soutěžní roboti detekovat. Pokud se soutěžní robot dotkne ducha, ubere se mu jeden život a vrátí se zpět na start. Aktuální dosažené skóre (tedy počet zhasnutých světel) tímto není ovlivněno.

V druhé soutěžní úloze, s názvem Sledování černé čáry, je úkolem robota co nejrychleji projet zadanou oblast po černé čáře nakreslené na podložce. Čára není souvislá, vyskytuje se na ní několik druhů překážek (například rozdvojení čáry), se kterými si robot musí umět poradit.

Dále je cílem této práce vytvořit webové stránky k oběma úlohám, kde budou podrobné návody ve formě pravidel a způsob hodnocení.

# Kapitola 2
# Lego Mindstorms

## 2.1 Historie Lego Mindstorms

Historie Lego Mindstorms sahá až do roku 1998, kdy byla představena první kostka Lego Mindstorms RCX. Od svého uvedení na trh se Lego Mindstorms stalo nejlépe prodávaným produktem společnosti Lego. Robotics Invention System™ rozdmýchal představy robotických nadšenců, což vedlo k rozvoji komunity studentů a dalších uživatelů, kteří vytvářejí a ovládají roboty Lego. [1]

Ještě téhož roku vlastník společnosti Lego, Kjeld Kirk Kristianse, a známý vynálezce, Dean Kamen, zakládají FIRST Lego League, robotickou soutěž pro studenty středních škol využívající Lego Mindstorms, při které studenti v týmech řeší problémy reálného světa, staví a programují robota Lego Mindstorms a následně svá řešení prezentují odborné porotě. Cílem této soutěže je hravou formou rozvíjet kritické myšlení u mladých lidí a zároveň rozvíjet jejich zájem o vědu a techniku. [1][2]

V roce 2006 byla představena nová verze Lego Mindstorms NXT 1.0. O tři roky později společnost Lego uvedla na trh vylepšenou verzi Lego Mindstorms NXT 2.0. Aktuálně nejnovější verze (k roku 2018) pochází z roku 2013 a je označována jako Lego Mindstorms EV3. Tato práce je postavena na Lego Mindstorms EV3, dále tedy bude zmiňována hlavně tato nejnovější verze Lego Mindstorms. [1]

### 2.1.1 Parametry jednotlivých verzí

| | RCX | NXT 2.0 | EV3 |
|---|---|---|---|
| procesor | H8/300 16MHz | 32-bit ARM7 48 MHz | ARM9 300 MHz |
| takt procesoru | 16MHz | 48 MHz | 300 MHz |
| paměť | 16 KB | 256 KB FLASH | 16 MB FLASH |
| paměť RAM | 32 KB | 64 KB | 64 MB |
| vstupní porty | 3 | 4 | 4 |
| výstupní porty | 3 | 4 | 4 |
| displej | 43 segmentů | maticový 100x60 px | maticový 178x128 px |
| komunikace | RS 232 | USB 2.0 | USB 2.0 (EV3-PC) |
| | infraport | Bluetooth 2.0 | USB 1.1 (EV3-EV3) |
| | | | Bluetooth |
| | | | Micro SD card |
| | | | WiFi (přes USB adaptér) |

**Tabulka 2.1.** Porovnání parametrů jednotlivých verzí [3] [4] [5]

## 2.2 Hardware EV3

### 2.2.1 Kostka

Kostka je základní stavební kámen celé stavebnice. Slouží jako řídící centrum a napájecí stanice pro všechny periferie. Na kostce se nachází 4 vstupní porty typu RJ12 pro připojení senzorů a 4 výstupní porty (taktéž typu RJ12) pro připojení motorů. Pro připojení k počítači je zde, na straně výstupních portů, umístěn mini-USB port. Kostka dále obsahuje hostitelský port USB, například pro připojení Wi-Fi adaptéru, slot na SD kartu, pomocí které lze rozšířit paměť kostky, a reproduktor. Na obrázku 2.1.jsou zobrazeny tři různé generace kostek Lego Mindstorms. [5]


**Obrázek 2.1.** Lego Mindstorms: RCX, NXT 2.0 a EV3 [6]

### 2.2.2 Velký a střední EV3 motor

O pohon robota se starají dva druhy motorů. Velký EV3 motor se se svými otáčkami 160-170 otáčkami/min a točivým momentem 20 Ncm řadí k pomalejšímu a silnějšímu z obou motorů. Je optimalizován jako hnací jednotka robota. Naproti tomu střední EV3 motor dosahuje rychlosti 240-250 otáček/min a točivého momentu 12 Ncm. Díky své menší velikosti a hmotnosti je schopen reagovat rychleji než velký EV3 motor. Oba motory mají integrovaný senzor otáčení s rozlišením 1 stupně pro přesné ovládání. Velký i střední EV3 motor je zobrazen na obrázku 2.2. [5]

**Obrázek 2.2.** Velký a střední EV3 motor [5]

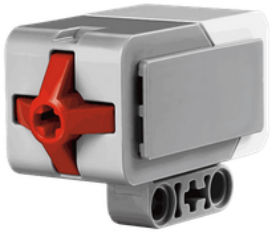
### 2.2.3 Dotykový senzor

Dotykový senzor je jediný analogový senzor. Dokáže rozeznávat tři stavy: stisknutí, uvolnění a náraz (= stisknutí a následné uvolnění). Je zobrazen na obrázku 2.3. [5]

### 2.2.4 Barevný senzor

Barevný senzor je digitální senzor rozeznávající barvu a intenzitu světla do něho vstupující. Pracuje ve třech různých režimech. V barevném režimu je senzor schopen rozpoznat až sedm barev: černou, modrou, zelenou, žlutou, červenou, bílou a žádnou barvu.

V režimu měření intenzity odraženého světla senzor vyzařuje červené světlo a následně měří intenzitu světla, která se odrazila od zkoumaného povrchu. Senzor požívá škálu 0 (velmi tmavá) - 100 (velmi světlá). V posledním režimu, měření intenzity okolního světla, senzor opět měří intenzitu světla vstupujícího do senzoru a škáluje ji na stupnici 0 - 100. Vzorkovací frekvence barevného senzoru je 1 kHz. Barevný senzor je zobrazený na obrázku 2.4. [5]



**Obrázek 2.3.** Dotykový senzor [5]

**Obrázek 2.4.** Barevný senzor [5]

### 2.2.5 Ultrazvukový senzor

Ultrazvukový senzor je digitální senzor, který měří vzdálenost pomocí ultrazvukových vln. Může pracovat ve dvou různých režimech. V prvním režimu vysílá a následně detekuje odražené ultrazvukové vlny, čímž měří vzdálenost objektu v rozsahu 0 - 255 cm. Ve druhém režimu senzor pouze detekuje ultrazvukové vlny, čímž může například zjistit přítomnost jiného robota vybaveného ultrazvukovým senzorem. Je zobrazen na obrázku 2.5. [5]

**Obrázek 2.5.** Ultrazvukový senzor [5]

### 2.2.6 Gyroskopický senzor

Gyroskopický senzor je digitální senzor, který detekuje otáčivý pohyb v rovině vyznačené šipkami na horní straně senzoru. Měří úhlovou rychlost otáčení (ve stupních za sekundu) a také celkové natočení (ve stupních). Gyroskopický senzor je zobrazen na obrázku 2.6. [5]

**Obrázek 2.6.** Gyroskopický senzor [5]

### 2.2.7 Infračervený senzor a infračervený maják

Infračervený senzor je digitální senzor, který vysílá a následně detekuje infračervené záření. Pracuje ve třech různých režimech. V prvním režimu, nazývaném režim přiblížení, používá infračervené záření odražené od překážky k určení vzdálenosti této překážky od senzoru. Maximální detekovatelná vzdálenost je 70 cm a je škálována na stupnici 0 (velmi blízko) - 100 (velmi daleko). Ve druhém módu (režim majáku) senzor pomocí infračerveného záření komunikuje s infračerveným majákem. Po detekování signálu je senzor schopen určit směr a vzdálenost (na stupnici 0 - 100) majáku, a to maximálně do 200 cm. V posledním režimu (režim vzdáleného ovládání) funguje infračervený maják jako dálkový ovladač. Infračervený senzor při tomto režimu detekuje, které tlačítko (případně kombinace tlačítek) je na infračerveném majáku stisknuto. Infračervený senzor i infračervený maják jsou zobrazeny na obrázku 2.7. [5]

Infračervený senzor ani infračervený maják nejsou součástí Lego Mindstorms Education EV3 Setu, se kterým pracuje tato práce.

**Obrázek 2.7.** Infračervený senzor a infračervený maják [5]

## 2.3 Software EV3

Po zakoupení stavebnice Lego Mindstorms je na kostce nahrán standardní Lego software. Pomocí počítačového programovacího grafického softwaru EV3-G, který komunikuje s kostkou, se vytvářejí programy z grafických bloků, které se umisťují za sebe. Existuje pět druhů bloků: akční, funkční, senzorové, datové operační a pokročilé. Akční bloky ovládají výstupní zařízení. Mohou zapínat a vypínat motory, případně regulovat jejich otáčky. Dále mají na starost výpis na displeji, zvukové efekty a podsvícení kostky. Funkční bloky mají na starosti strukturu a běh programu. Mimo jiné se mezi ně řadí startovní blok, kterým začínají všechny programy vytvořené v EV3-G. Funkční bloky dále obsahují blok na zdržení, větvení či opakování části programu. Senzorové bloky, jak už název napovídá, mají na starosti čtení informací přicházejících z použitých senzorů. Datové operační bloky jsou uzpůsobené pro práci s proměnnými. Umožňují nejen proměnné číst a psát, ale také provádět mezi nimi různé matematické operace. A ko-

nečně pokročilé bloky, které umožňují spravovat připojení, pracovat se soubory a další činnosti potřebné k ovládání EV3. [7]

Aplikace EV3 Programmer je podobná EV3-G, je však určena pro mobilní telefony, či tablety. Stačí připojit kostku k mobilnímu telefonu/tabletu pomocí bluetooth. Tato mobilní aplikace však obsahuje méně bloků než desktopová verze. [8]

### 2.3.1 NXC

Další možností jak programovat robota bez nutnosti přehrát firmware kostky je jazyk NXC. Syntaxe jazyka NXC je velmi podobná jazyku C a je překládána do bytecodu NXT, který lze nahrát přímo do kostky. Interpreter bytecodu však přináší různá omezení (oproti jazyku C) vyplývající z hardwarových možností Lego Mindstorms. NXC API, definováno v hlavičkovém souboru, popisuje systémové funkce, konstanty a různá makra, které jsou použitelná pro programování robota. [9]

NXC zatím bohužel nepodporuje Lego Mindstorms EV3. Je tedy nutné použít program, který převede kód napsaný v NXC na kód pro EV3. K tomuto účelu je možné použít program NXC4EV3, který nejdříve přeloží jazyk NCX na C, a ten následně překompiluje do nativního kódu. [10]

### 2.3.2 RobotC

RobotC je výkonný programovací jazyk, založený na jazyce C, pro psaní a debugování programů, a také vyšší programovací jazyk, který umožňuje komplexní debugování v reálném čase. RobotC používá syntaxi jazyka C doplněnou o další jazykové rozšíření nutné pro ovládání specifického hardwaru Lego Mindstorms. Velkou výhodou této možnosti je plně integrované ladění softwaru, které umožňuje provádět dílčí příkazy řádek po řádku a tím přesně analyzovat konkrétní příkazy a proměnné. Pomocí dalších ladících nástrojů je možné zobrazit aktuální stav všech motorů a senzorů v reálném čase. [11]

### 2.3.3 Matlab a Simulink

Existují dva různé balíčky podpory- pro Matlab a pro Simulink. Balíček pro Matlab obsahuje funkce na ovládání motorů a senzorů. Pomocí Matlabu se tedy lze připojit k EV3 kostce (pomocí Bluetooth, WiFi nebo USB), inicializovat motory a senzory, ovládat motory a číst výstupní hodnoty ze všech senzorů. Přímo na kostce je možné pomocí Matlabu vypisovat text, vydávat zvuky, či detekovat stisk tlačítek. [12]

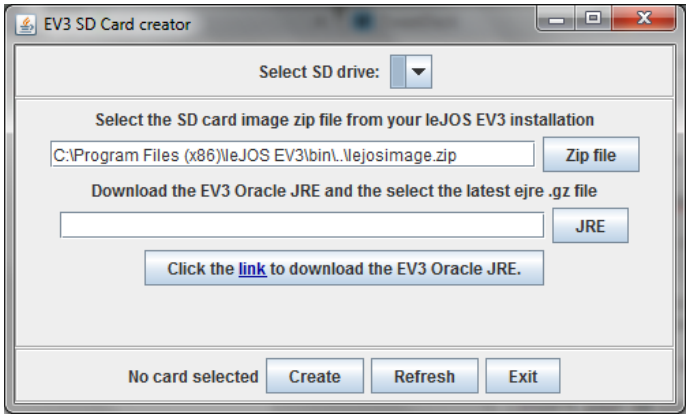
Balíček určený pro Simulink dále rozšiřuje výše uvedené možnosti o knihovnu bloků, pomocí kterých lze přistupovat k senzorům, akčním členům a komunikačním rozhraním. Vytvořené algoritmy lze pomocí těchto bloků simulovat přímo v prostředí Simulinku, případně je nahrát do EV3 kostky. Pomocí Simulinku lze také ladit (měnit) parametry robota v reálném čase. [13]

### 2.3.4 LejOS

Dalším softwarem použitelným k programování EV3 kostky je leJOS. Jak už název napovídá, umožňuje tento software programovat robota v jazyce Java. Existují různé verze pro RCX, NXT i EV3 kostky, pro nejnovější EV3 však ještě není tento software plně dokončen (k roku 2018 je nejnovější verze 0.9.1-beta). LeJOS používá standardní Java syntaxi, a navíc zahrnuje speciální třídy pro ovládání specifického hardwaru Lego Mindstorms, které jsou přehledně popsány v dokumentaci k leJOS EV3 API. [14]

Nevýhodou softwaru leJOS oproti ostatním softwarovým možnostem je jeho velmi složitá instalace a první spuštění. Návod na instalaci a první spuštění je následovný:

1 Nejdříve je nutné ověřit na počítači přítomnost Java SE Development Kitu. Pokud není na počítači nainstalován, je možné si ho stáhnout ze stránek: `http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html`.
2 Poté je možné přistoupit ke stažení a nainstalování lejOSu a to konkrétně souboru 0.9.1-beta ze stránky: `https://sourceforge.net/projects/ev3.lejos.p/files/`.
3 Po nainstalování lejOSu se spustí program EV3 SD Card Creator. Jeho náhled je na obrázku 2.8. V tomto miniprogramu je nutné nastavit umístění SD karty v počítači a cestu k instalačnímu souboru EV3 (cesta k tomuto souboru je již vyplněna automaticky). Poté je nutné stáhnout EV3 Oracle JRE, a to nejpohodlněji kliknutím na tlačítko níže. Na otevřené stránce se nachází dva soubory ke stažení: JDK a JRE a je snadné se přehlédnout a stáhnout špatný soubor. Po doplnění všech polí se po stisknutí tlačítka Create vytvoří instalační soubor na kartě SD.
4 Připravená karta se vloží do vypnutého robota. Po jeho zapnutí se lejOS automaticky nainstaluje.
5 V programovacím prostředí `Eclipse` se v záložce `Help` vybere možnost `Install new software`. Do řádku `Work with` se vloží řetězec: `LeJOS Update Site - http://lejos.sourceforge.net/tools/eclipse/plugin/ev3` a dokončí se instalace.
6 V záložce `Window` se vybere možnost `Preferences`. V zobrazeném okně se vybere záložka `lejOS EV3`, kde se do pole `Name` nastaví: `10.0.1.1`.
7 Nyní už jen zbývá připojit robota pomocí přiloženého kabelu a začít programovat.




**Obrázek 2.8.** EV3 SD Card Creator

### 2.3.5 Základní ukázky kódu

V níže uvedených ukázkách jsou zobrazeny základní bloky kódu, specifické pro práci s Lego Mindstorms EV3 v programovacím jazyce lejOS. Všechny tyto konstrukce jsou podrobně popsány v lejOS API, nicméně pro programátory seznamující se s jazykem lejOS by mohlo být výhodné, mít přehled základních funkcí na jednom místě. Všechny tyto konstrukce jsou také použité v řídících programech robotů v praktické části této práce.

Motor:

```
RegulatedMotor motor = new EV3LargeRegulatedMotor(MotorPort.C);
motor.rotate(360); //motor se otočí o 360 stupňů
motor.close();
```

Synchronizace dvou motorů:

```
RegulatedMotor motorL = new EV3LargeRegulatedMotor(MotorPort.B);
RegulatedMotor motorR = new EV3LargeRegulatedMotor(MotorPort.C);
motorL.synchronizeWith(new RegulatedMotor[] { motorR });

motorL.startSynchronization();
motorL.setSpeed(700); //nastaví rychlost otáčení 700 stupňů / sec
motorR.setSpeed(700);
motorL.forward(); //oba motory se rozjedou současně dopředu
motorR.forward();
motorL.endSynchronization();
Delay.msDelay(1000); //spoždění 1000ms; motory se točí

motorL.startSynchronization();
motorL.stop(); //oba motory se současně zastaví
motorR.stop();
motorL.endSynchronization();
```

Dotykový senzor:

```
EV3TouchSensor touch = new EV3TouchSensor(SensorPort.S1);
SensorMode touchM = touch.getTouchMode();
float[] sampleTouch = new float[touchM.sampleSize()];
touchM.fetchSample(sampleTouch, 0); //příkaz ke zjištění stavu
if(sampleTouch[0] == 1) .. //pokud je dotykový senzor stisknutý..
touch.close();
```

Ultrazvukový senzor:

```
EV3UltrasonicSensor distSensor = new EV3UltrasonicSensor(SensorPort.S2);
SampleProvider distanceSampler = distSensor.getDistanceMode();
float[] sampleDistance = new float[distanceSampler.sampleSize()];
distanceSampler.fetchSample(sampleDistance, 0); //příkaz ke změření vzd.
if(sampleDistance[0] < 1) .. //pokud je vzdálenost menší než 1m..
distanceSensor.close();
```

Gyroskopický senzor:

```
EV3GyroSensor gyro = new EV3GyroSensor(SensorPort.S3);
SampleProvider gyroSampler = gyro.getAngleAndRateMode();
float[] sampleGyro = new float[gyroSampler.sampleSize()];
gyroSampler.fetchSample(sampleGyro, 0);
if(sampleGyro[0] < 90) .. //pokud je úhel otočení menší než 90 stupňů..
gyro.close();
```

Barevný senzor:

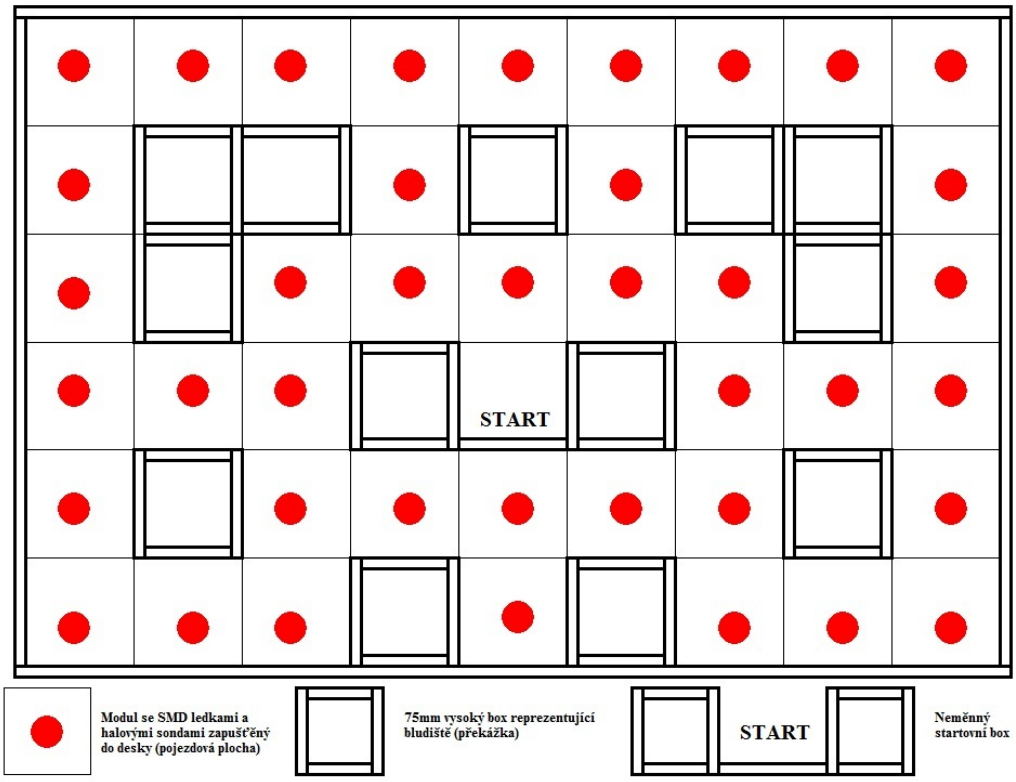
```
EV3ColorSensor color = new EV3ColorSensor(SensorPort.S4);
SampleProvider colorSampler = color.getRedMode();
float[] sampleColor = new float[colorSampler.sampleSize()];
colorSampler.fetchSample(sampleColor, 0); //příkaz ke změření intenzity
if(sampleColor[0] < 0.1) .. //pokud je intenzita barvy menší než 0.1..
color.close();
```

# Kapitola 3
# Pac-Man

V této úloze je úkolem robota za stanovený časový limit (90 s) projet co největší část bludiště, a přitom magnetem zhasínat barevná světla (vybavená Hallovou sondou pro detekci magnetického pole) umístěná v podložce. Bodové ohodnocení odpovídá počtu zhasnutých světel. Bludiště je možné si představit jako ohraničenou plochu z jednotlivých čtverců, přičemž některé z nich jsou 75 mm vysoké překážky. Příklad bludiště je na obrázku 3.1.

Při této úloze se bludištěm nebude pohybovat pouze soutěžní robot (dále jen robot), ale i duchové. V případě, že se robot dotkne ducha, vrací se zpět na start. Jeho dosažené bodové skóre se přitom nezmění a časový limit dále beze změny běží. Duchové mohou vykonávat 4 druhy pohybu:

1 po úsečce mezi dvěma stěnami vertikálně
2 po úsečce mezi dvěma stěnami horizontálně
3 okolo překážky (skládající se z jednoho čtverce) ve směru hodinových ručiček
4 stání v prostoru jakožto překážka




**Obrázek 3.1.** Příklad bludiště [15]

## 3.1 Duchové

Duchové jsou stejně jako soutěžní roboti postavení ze stavebnice Lego Mindstorms. Jsou jednoduché konstrukce a jednotlivé typy duchů pro různé pohyby jsou si konstrukčně podobné. Je tedy možné pomocí jednoduchých úprav (například přidáním senzorů) ducha přestavět z jednoho typu na jiný. Při tvorbě duchů byl kladen velký důraz na robustnost řešení, tedy aby byl duch co nejméně ovlivnitelný náhodnými událostmi.

### 3.1.1 Duchové po úsečce

Duchové tohoto typu se pohybují po úsečce ohraničené z obou stran stěnou. Na obou stranách robota (přední a zadní straně ve směru pohybu) je umístěn dotykový senzor a na boku robota je ultrazvukový senzor, který měří vzdálenost robota od stěny. V první fázi programu jede robot dopředu, a přitom si pomocí PD regulátoru udržuje žádaný odstup od stěny. Když je aktivován dotykový senzor na přední straně robota, začne robot couvat. V momentě kdy je stisknut dotykový senzor na zadní straně robota, celý cyklus se opakuje.

PD regulátor byl nastavován experimentálně s pomocí znalostí o jednotlivých složkách PD regulátoru. P složka má na starosti velikost akčního zásahu, tedy to, jak moc se robot vychýlí z přímého směru. Větší P složka znamená větší vychýlení. D složka PD regulátoru naopak představuje tlumení závislé na aktuální změně výchylky robota, neboli ovlivňuje, jak rychle se robot ustálí v přímém směru. Vyšší D složka znamená, že se robot rychleji ustálí v přímém směru.

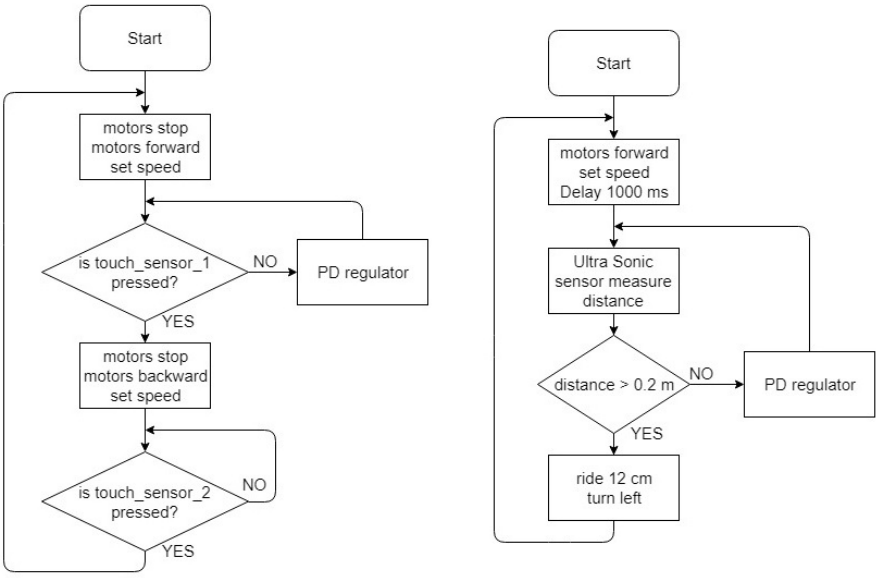
### 3.1.2 Duchové kolem překážky

V tomto případě duchové jezdí okolo (jedné) čtvercové překážky. Na boku robota je stejně jako v předchozím případě umístěn ultrazvukový senzor, který měří vzdálenost robota od překážky. Robot jede dopředu podél překážky, přičemž si od ní pomocí ultrazvukového senzoru udržuje stálou vzdálenost. Pokud ultrazvukový senzor změří vzdálenost větší než 20 cm, tedy že robot právě minul vrchol čtverce, ujede robot ještě krátký úsek a otočí se o 90 stupňů doleva. Pokud by robot po detekování konce překážky neujel ještě onu vzdálenost, narazil by do překážky, jelikož je ultrazvukový senzor umístěn v přední části robota. Nyní robot pokračuje opět v jízdě dopředu. V této fázi je implementována prodleva 1 s, než opět začne ultrazvukový senzor měřit vzdálenost. Je tomu tak z důvodu, aby nedetekoval prázdný prostor a tím pádem nedal povel k zatočení robota doleva. Za onu 1 s robot dojede k další hraně a celý cyklus se opakuje. PD regulátor byl nastavován experimentálně, stejným způsobem jako v předchozím odstavci. Vývojové diagramy, znázorňující princip funkce obou typů duchů, jsou na obrázku 3.2.

### 3.1.3 Univerzální duchové

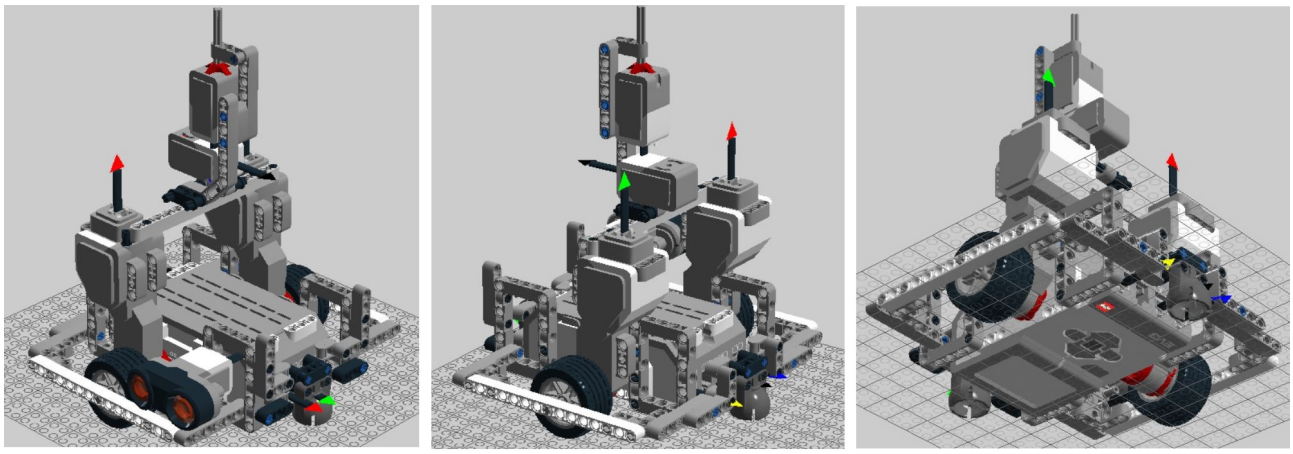
Ve vlastní soutěži byli použiti univerzální duchové, kteří se stejnou konstrukcí zvládají dva různé pohyby (po úsečce od stěny ke stěně a okolo překážky), přičemž se liší pouze použitým softwarem. Velkou výhodou univerzálních duchů tedy je, že všechny typy duchů vypadají stejně.

Každý duch se fyzicky skládá ze dvou částí- základní části vyrobené ze stavebnice Lego Mindstorms a z nasazovací části, vyrobené z barevných Lego kostiček. Základní část je zobrazena, v programu Lego Digital Designer, na obrázku 3.3. Fotografie celkového pohledu na ducha, s nasazenou nasazovací částí, jsou na obrázku 3.4.




**Obrázek 3.2.** Vývojový diagram ducha jezdícího po úsečce (vlevo) a okolo překážky (vpravo)

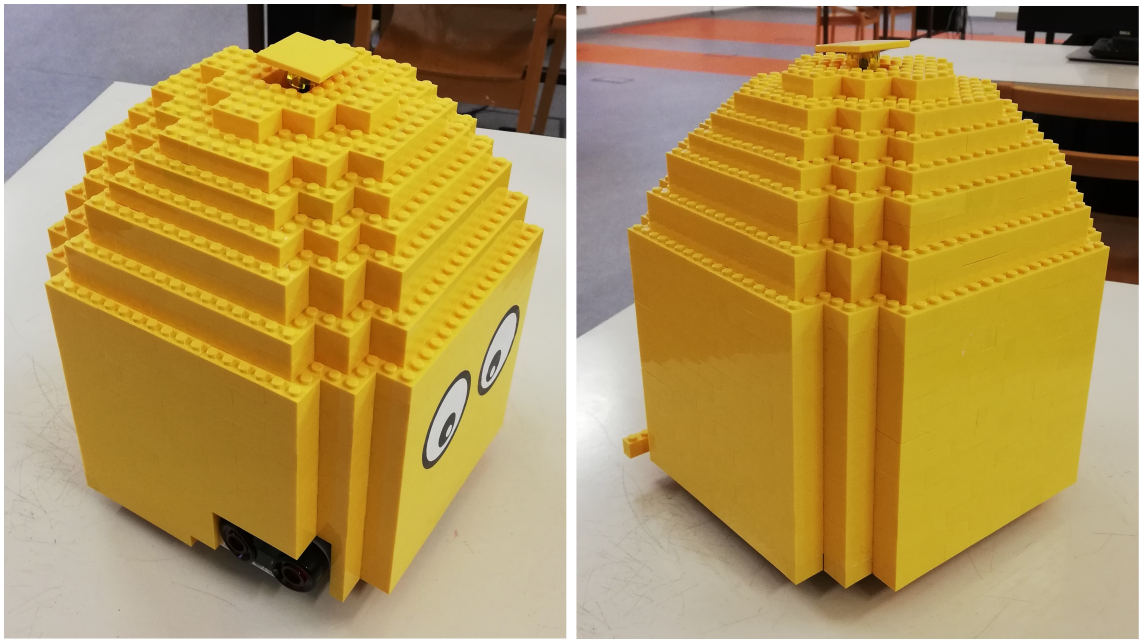
Řídící program univerzálních duchů je velmi podobný programům duchů jezdících po úsečce od stěny ke stěně a kolem překážky. Jedinou velkou změnou je přidání gyroskopického senzoru a odebrání dvou dotykových senzorů v případě ducha jezdícího po úsečce. U tohoto ducha jeho nárazy tedy místo dotykových senzorů detekuje gyroskopický senzor. Ten je dále využit při zatáčení robota jezdícího okolo překážky, kdy je pomocí něho měřen úhel otočení.



**Obrázek 3.3.** Pohled na základní část ducha v programu Lego Digital Designer

## 3.2 Robot projíždějící bludiště

Úkolem robota je za stanovený časový limit 90 sekund projet celé bludiště, případně co největší jeho část. V bludiště se společně s robotem mohou pohybovat duchové, kterých se robot nesmí dotknout. Pokud se tak stane, je vrácen zpět na startovní pole a odečte se mu jeden z celkových tří životů. Při vrácení robota na startovní pole může



**Obrázek 3.4.** Fotografie ducha

soutěžící spustit jiný program, případně nechat stejný a stisknout například tlačítko, aby byl robot o této změně informován.

Jelikož soutěžící znají ještě před startem soutěžní jízdy, zda v této jízdě budou v bludišti přítomní duchové, či nikoliv, existují dvě verze programu- jedna pro bludiště s duchy a druhá pro bludiště bez duchů. Verze bez duchů neobsahuje procedury a funkce pro detekci a mapování duchů a je zde robotovi dovoleno narážet do překážek. Jízda je tedy rychlejší, než v případě, že se v bludišti společně s robotem nacházejí i duchové.

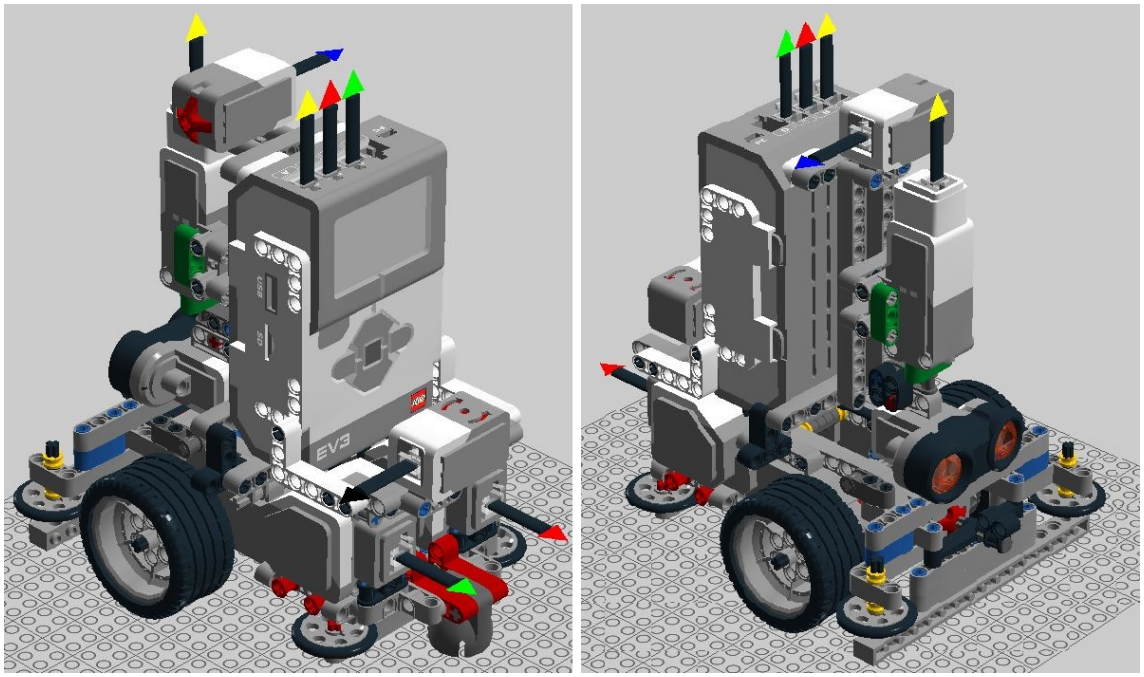
## 3.3 Konstrukce robota

Při konstrukci robota byl kladen velký důraz na minimální rozměry (hlavně šířku a délku) při zachování potřebné funkčnosti. Podvozek robota je tedy co nejužší a v rámci možností i co nejkratší, pro pohodlnější průjezd bludištěm a snížení pravděpodobnosti nechtěného nárazu do překážky. V každém ze čtyř rohů robota se nachází pomocné kolečko, které jednak dokáže srovnat robota při šikmé jízdě podél zdi, a také ulehčuje otáčení díky tomu, že robot nedrhne o zeď, ale jede po ní otočným kolečkem.

Na podvozku je v přední části přidělán dotykový senzor (pro detekci nárazu robota) a v zadní části gyroskop (pro přesné zatáčení). Směrem nahoru je v svislém směru přidělaná kostka, na které se dále nachází malý motor a druhý dotykový senzor (pro spuštění programu a řešení kolize s duchy). Malý motor volně otáčí s ultrazvukovým senzorem, který je díky tomu schopný měřit vzdálenost od robota ve třech různých směrech: dopředu, doleva a doprava. Model robota vytvořený v programu Lego Digital Designer je na obrázku 3.5.

## 3.4 Souřadnicový systém

Pro snadnější orientaci v bludišti vznikl souřadnicový systém, který každému políčku (čtverci bludiště) přiřazuje souřadnice x a y. Souřadnicový systém je matice o velikosti 6 x 9, přičemž levý horní roh bludiště má souřadnice (0,0), levý dolní roh (5,0) a pravý



**Obrázek 3.5.** Pohled na robota v programu LEGO Digital Designer

dolní roh (5,8). Robot si při svém pohybu neustále uchovává a aktualizuje obě souřadnice a směr, je tedy možné v každém okamžiku jednoduše zjistit, na jakém políčku se právě nachází.

Tato matice souřadnic je v řídícím programu definována jako dvourozměrné pole typu `Integer`. Hodnota každého políčka reflektuje jeho aktuální stav, respektive zda se na něm nachází překážka či duch, a zda je zde rozsvícena LED dioda. Různé hodnoty políčka vyjadřují:

0 LED dioda je již zhasnutá (robot již přes toto políčko přejel)
1 LED dioda stále svítí
2 na tomto políčku se nachází překážka nebo duch
3 o tomto políčku robot zatím nic neví
4 políčko podezřelé z výskytu překážky nebo ducha

## 3.5 Detekce duchů

Robot při dané konstrukci (ultrazvukový senzor v dolní části robota) nerozlišuje mezi duchem a překážkou. Je tedy naprogramován, aby nenarazil do žádné překážky. Při své jízdě si robot mapuje překážky primárně na své pravé straně. Pokud jede těsně při okraji bludiště a senzor na pravé straně by snímal pouze zeď bludiště, otočí se ultrazvukový senzor doleva a mapuje překážky i na levé straně. Při otočkách a dalších vynucených zastaveních, si robot může mapovat překážky na své levé i pravé straně, případně před sebou. Pokud robot zaznamená, že na políčku vedle sebe nemá překážku, změří vzdálenost od další překážky (nebo konce bludiště) na dané straně, a za pomocí znalosti rozměrů jednotlivých políček určí, na kterém políčku se nachází duch (překážka). Když poté jede daným směrem, zastaví se těsně před duchem (překážkou) a ultrazvukovým senzorem se přesvědčí, zda tam duch (překážka) opravdu je a označí ji v souřadnicovém systému.

Z časových důvodů (na průjezd bludiště je maximálně 90 sekund) robot nerozlišuje mezi jezdícími a stojícími duchy. Pokud by robot čekal u každé podezřelé překážky, zda se náhodou nepohne, ztratil by tím spoustu času, který by mu poté chyběl na samotný průjezd bludištěm. V případě jezdících duchů se může stát, že se duch přesune a políčko, kde byl původně a kde je v souřadnicovém systému zaznamenán, je volné a se svítící LED diodou. Pokud by taková situace nastala, robot při dalším průjezdu okolo daného políčka nebude detekovat překážku (která už tam reálně není), vymaže ji tedy v souřadnicovém systému a na dané políčko pojede.

## 3.6 Srážka s duchem

Ve výjimečných případech se může stát, že se duch a robot srazí (například pokud duch narazí do robota z levé strany, když robot detekuje situaci před sebou a napravo). Dle pravidel robot ztrácí jeden život a přesouvá se na startovní pole. Bezprostředně po nárazu je nutné na robotovi stisknout dotykový senzor, umístěný v jeho horní části. Díky tomu se zastaví motory a robot se přesune v souřadnicovém systému na výchozí pole, přičemž si pamatuje všechna projetá pole a překážky, které již zmapoval. Po opětovném stisknutí dotykového senzoru robot najde nejbližší pole, kde ještě nebyl nebo zde nemá zmapované území, spustí se motory a robot opět pokračuje v prozkoumávání bludiště.

## 3.7 Problémy s přesností

Při testování robota se objevovaly velké problémy s přesností. Prvotní nepřesnost vzniká již při měření senzory. Například gyroskopický senzor má udávanou přesnost +- 3 stupně. Tato přesnost je však závislá na rychlosti pohybu (otáčení) a dalších parametrech. Je tedy možné, že za specifických podmínek, může být nepřesnost ještě vyšší. Ultrazvukový senzor má udávanou přesnost +- 1 cm. Větším problémem u tohoto senzoru však je, že se měřící paprsek s rostoucí vzdáleností rozšiřuje a senzor má poté problémy s měřením malého předmětu na velké vzdálenosti (například stěna bludiště na velkou vzdálenost). Záleží také na povrchu měřeného předmětu a úhlu jeho natočení vůči senzoru.

Další nepřesnosti vznikají při pohybu motorů. Čidlo otáčení zabudované v motorech má samo o sobě nepřesnost 1 stupeň. Po zastavení motorů (například po ujetí dané vzdálenosti) však motorům trvá malý okamžik, než se dotočí a skutečně zastaví, čímž vzniká další nepřesnost. Lépe na tom není ani rozjezd motorů, které, ač by se měly rozeběhnout synchronizovaně ve stejný okamžik, se rozeběhnou o malý časový úsek po sobě, čímž robot při rozjezdu mírně zatočí. Při vyšších rychlostech (a tedy i vyšším zrychlení při startu) se kola v závislosti na povrchu mohou protáčet, obvykle každé kolo jinou dobu.

Ani bludiště není vyrobeno úplně přesně. Některé překážky mají odlišné rozměry, než jiné (jedná se o rozdíly v řádech milimetrů). Některá světla v podlaze bludiště také nejsou zasazena úplně zároveň s okolím a na jejich okrajích vzniká drobná hrana, která může robota při průjezdu rozhodit.

Všechny výše uvedené nepřesnosti mohou ale také nemusí nastat, proto je každá jízda robota na základě náhodných vlivů mírně odlišná. Některé nepřesnosti se dají hardwarově vyřešit, jako například přidělat přídavná kolečka na strany robota, aby se v případě šikmého nárazu do zdi srovnal do přímého směru, nebo před samotnou jízdou pečlivě umýt celou dráhu, včetně koleček robota, pro lepší přilnavost a tím eliminaci protáčení

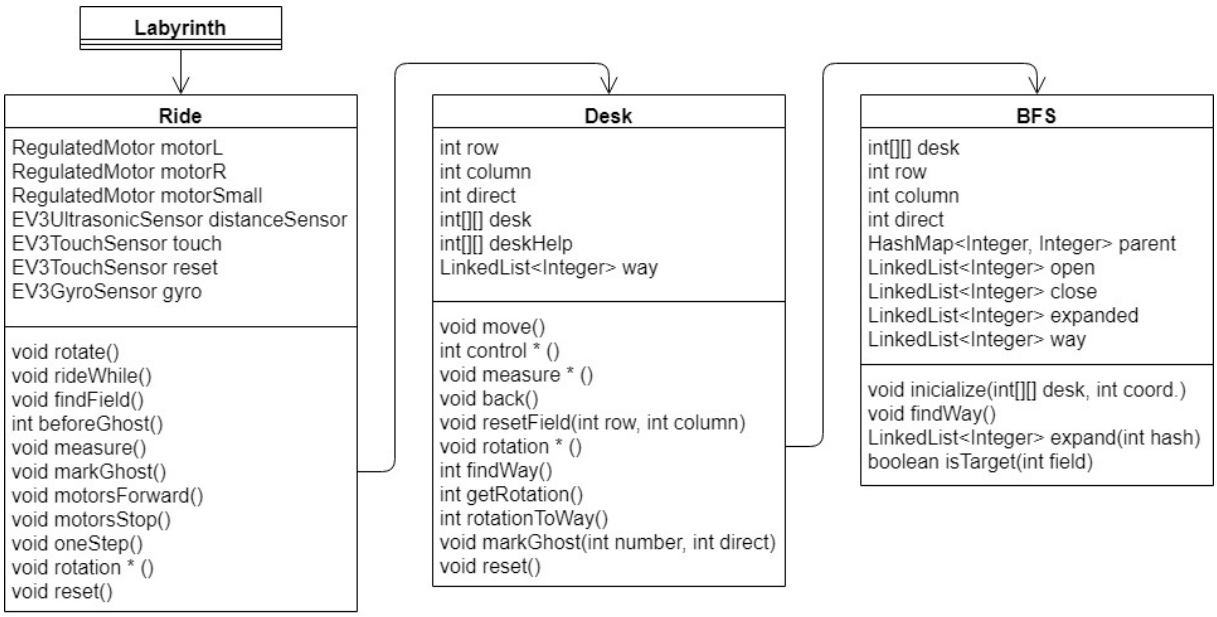
koleček. Mírné zatočení při rozjezdu robota z důvodu postupného spouštění motorů se dá částečně vylepšit omezením počtu rozjezdů. Některé problémy však eliminovat nejde a je proto nutné s nimi počítat při návrhu řídícího algoritmu.

## 3.8 Objektový návrh

Celý program je rozdělený do několika objektů, jejichž vzorové třídy jsou znázorněny na obrázku 3.4. Ve třídě `Labyrinth` je metoda `Main`, ve které je vytvořen objekt `labyrinth`, dle třídy `Ride`. Objekt `labyrinth` ovládá všechny motory a senzory a je v něm implementovaná logika rozhodování. Sbírá data ze senzorů, zpracuje je a dá příkaz motorům.

Objekt `labyrinth` si dále vytváří objekt `desk`, dle třídy `Desk`. Tento objekt má na starosti veškeré činnosti spojené se souřadnicovým systémem. Uchovává aktuální polohu a směr robota, jsou v něm procedury na pohyb a zatáčení robota v souřadnicovém systému, na mapování okolí robota, či kontrolní funkce, které vrací informace o okolí robota.

Pokud se stane, že je robot na políčku, kde jsou všechna okolní již projetá nebo překážky, vytvoří si objekt `desk` nový objekt `bfs`, dle třídy `BFS`, ve kterém je implementované prohledávání do šířky, včetně všech pomocných procedur a funkcí. Tento objekt má za úkol najít nejbližší nezmapované či neprojeté políčko.




**Obrázek 3.6.** Objektový návrh programu

## 3.9 Třída Ride

Třída `Ride` definuje všechny procedury pro ovládání senzorů a motorů včetně rozhodovací logiky, která vyhodnocuje údaje ze senzorů a na jejich základě dává povely motorům. Z konstruktoru této třídy je zavolána procedura `goToLabyrinth()`, která nejdříve čeká na stisknutí dotykového senzoru, a poté se zde v nekonečném cyklu opakují procedury `rotate()` a `rideWhile()`.

### 3.9.1 rotate()

V této proceduře je implementovaná většina rozhodovací logiky celého programu. Na začátku procedury se otočí ultrazvukový senzor doprava, pokud se robot právě nenachází těsně u okraje bludiště (pokud nemá na pravé straně stěnu bludiště). Pokud by tomu tak bylo, ultrazvukový senzor se otočí doleva. Dále se změří vzdálenost a zavolá se procedura `measure()`, která mapuje překážky v okolí robota. Následně se zavolá několik funkcí objektu `desk`, které mají za úkol podat informace o okolí robota. Konkrétně se jedná o:

- zda je nalevo od robota překážka nebo duch
- zda je napravo od robota překážka nebo duch
- zda je napravo od robota okraj bludiště
- zda je před robotem okraj bludiště
- zda je před robotem překážka nebo duch
- zda robot už byl na políčku těsně před ním
- zda robot už byl na políčku napravo
- zda je před robotem, za robotem, nalevo a napravo od robota překážka (nebo duch), nebo zda už na těchto polích byl

Na základě těchto informací program rozhodne o tom, jestli a případně jakým směrem se robot otočí. Při jednom průchodu procedurou se robot může otočit pouze jednou. K tomu dopomáhá pomocná proměnná `rot`, jejíž hodnota je na začátku procedury nastavena na 0 a po každé otočce je nastavena na 1. Robot smí zatáčet pouze pokud je hodnota `rot` 0.

Pokud napravo od robota (nebo nalevo od robota, nachází-li se u okraje bludiště) není překážka, a zároveň robot na tomto poli ještě nebyl, zavolá se procedura `markGhost(double length)`, která označí překážky (duchy), které budou stát robotovi po zatočení v cestě, a poté robota otočí doprava (případně doleva, nachází-li se robot u okraje bludiště).

Pokud robot na políčku před ním už byl nebo je těsně před okrajem bludiště, zároveň před ním není duch a neví nic o políčku nalevo, otočí se ultrazvukový senzor doleva, a zmapuje překážky na levé straně robota zavoláním procedury `measure()`. V případě, že není nalevo překážka, robot zatočí doleva. V opačném případě, kdy robot nemůže jet dopředu, doprava ani doleva, otočí se o 180 stupňů. Pokud by však robot předem věděl, že je nalevo od něho překážka, nebo už zde byl, a tím pádem věděl, že nemůže jet dopředu, doleva, doprava ani dozadu, spustí se procedura `findField()`, která pomocí BFS najde nejbližší neprozkoumané políčko.

### 3.9.2 rideWhile()

Základní procedura, která se stará o pohyb robota. V době jejího spuštění se robot nachází, většinou po zatočení, přibližně v polovině políčka. Nejdříve se resetuje úhel otočení levého motoru, ultrazvukový senzor se otočí směrem doprava (pokud se robot pravým bokem nenachází u okraje bludiště), a pokud se před robotem nenachází políčko, kde už robot byl, spustí se oba velké motory směrem dopředu.

V první fázi této procedury se motory otočí o 200 stupňů. Pokud by při tomto pohybu robot narazil do zdi (aktivoval se dotykový senzor na přední straně robota), oba motory se synchronizovaně otočí o 30 stupňů dozadu, aby robot poodjel od zdi a mohl se bez drhnutí otočit. Poté se motory vypnou a políčko před robotem se označí jako překážka. Pokud však do ničeho nenarazí, mezi 160. a 200. stupněm otočení levého motoru je zavolána procedura `measure()`, která mapuje překážky okolo robota. Po ujetí

výše zmíněné vzdálenosti se opět resetuje úhel otočení levého motoru. Jelikož na začátku nemusí být robot uprostřed políčka, ale může být na jeho okraji, je zde ještě zdržení 200ms, kdy neprobíhá žádné mapování okolí, robot pouze jede dopředu. Tato ujetá vzdálenost se už počítá do další fáze procedury, a je zde pro překlenutí momentu, kdy robot už může ale nemusí být na dalším políčku (a případně mít okolo sebe překážky, které mapuje).

Na začátku druhé fáze procedury se robot nachází na začátku nového políčka. Tato část se stále opakuje, dokud robot nemůže zatočit doprava (nebo doleva v případě, že je pravým bokem u okraje bludiště), není před ním překážka (případně duch) nebo by jel na políčko, kde už byl. Podobně jako v první fázi, i zde se mezi 450. a 490. stupněm otočení levého motoru spouští procedura `measure()`, která mapuje okolí robota. Když levý motor dosáhne otočení o 540 stupňů, zavolá se metoda `move()` objektu `desk`, která posune robota v souřadnicovém systému o jedno políčku dopředu a resetuje se otočení levého motoru. Dále se zavolají příslušně funkce objektu `desk`, který ověří, zda se na dalším políčku nenachází překážka (duch), případně zda zde robot již byl.

Pokud by se na dalším políčku nacházela překážka (duch), spustí se funkce `beforeGhost()`. V případě, že robot může zatočit doprava nebo doleva (ultrazvukový senzor detekuje, že na pravé/levé straně robota není překážka), robot jede ještě 370ms rovně, a poté zatočí doprava/doleva. Je to z důvodu, aby neodbočil ihned, co detekuje prázdný prostor, a tím pádem pravděpodobně narazil do rohu překážky, ale aby dojel doprostřed strany políčka a až poté zatočil. Jelikož má duch menší rozměry než překážka, je zde ještě integrovaná podmínka, že v případě, když se objeví volno vpravo/vlevo při otočení motoru o 450 až 540 stupňů (540 stupňů je přibližně rozměr překážky), zavolá se i tak metoda `move()` objektu `desk` a robot se pohne v souřadnicovém systému. Pokud by zde tato podmínka nebyla, při projetí okolo volně stojícího ducha by se rozhodila robotova poloha v souřadnicovém systému. Pokud by při pohybu robot narazil, bude následovat stejný postup jako v první fázi.

### 3.9.3 findField()

Tato procedura se spustí v případě, pokud má robot vepředu, vzadu, vlevo a vpravo překážku (nebo ducha), či už na těchto políčkách byl (nebo kombinace těchto dvou faktorů). Nejprve se zavolá funkce `findWay()` objektu `desk`, která najde cestu k nejbližšímu neprozkoumanému políčku (tedy políčku, kde robot ještě nebyl, nebo zde není překážka) a vrátí délku této cesty (počet políček, které robot cestou projede).

Nyní se spustí for cyklus o počtu opakování rovnému délce cesty a zavolá se funkce `rotationToWay()` objektu `desk`, jejíž návratová hodnota je směr, kterým se má robot vydat z aktuálního políčka na příští. Po případném zatočení se zavolá funkce `getRotation()` objektu `desk`, která vrátí směr, kterým robot pojede z příštího políčka na přespříští. Tímto směrem se otočí ultrazvukový senzor (v případě přímého směru zůstává ultrazvukový senzor vpravo).

### 3.9.4 beforeGhost()

Tato funkce se spustí, když je podezření, že je na políčku před robotem duch. Její návratová hodnota je informace (0 nebo 1), zda tomu tak opravdu je, či se jedná o falešně pozitivní detekci.

Nejprve se oba velké motory synchronizovaně otočí o 350 stupňů dopředu. Pokud přitom robot narazí (aktivuje se dotykový senzor na čelní straně robota), případně ultrazvukový senzor detekuje volné pole na pravé straně, pohyb je přerušen. Poté se ultrazvukový senzor natočí, aby zabíral oblast před robotem, změří vzdálenost, a ta

se jako parametr pošle proceduře `markGhost(double length)`. Pomocné proměnné `helpGhost` je přiřazena návratová hodnota (1, pokud ano, v opačném případě 0) funkce `controlGhost()` objektu `desk`, která říká, zda se před robotem nachází duch.

Pokud se před robotem opravdu nachází duch (`helpGhost` má hodnotu 1), spustí se motory a robot dojede přibližně 10 cm před ducha (překážku). Nakonec se ultrazvukový senzor otočí doprava. Pokud se před robotem duch nenachází, otočí se ultrazvukový senzor doprava a oba velké motory se dotočí o celkem 540 stupňů, čímž se robot přesune na další políčko. Na závěr je zavolána procedura `move()` objektu `desk`, která robota přesune v souřadnicovém systému v závislosti na směru o jedno políčku dopředu.

### 3.9.5 measure(), markGhost(double length)

Tyto dvě jednoduché procedury mají za úkol mapování okolí robota. Než však bude vysvětlen jejich princip, je nutné popsat otáčení ultrazvukového senzoru. Ten je připevněn na malý motor, jehož úhel natočení je na začátku programu, kdy míří přímo dopředu, resetován. Poté si program udržuje hodnotu úhlu otočení malého motoru, díky čemuž zná směr měření ultrazvukového senzoru.

Procedura `measure()` dle úhlu natočení ultrazvukového senzoru zavolá příslušnou proceduru objektu `desk`, tedy buď `measureLeft()` nebo `measureRight()`. Při volání `markGhost(double length)` je této proceduře jako parametr předána předem naměřená vzdálenost. Z této vzdálenosti je vypočítáno o kolik políček dále se nachází duch (nebo překážka). Zároveň je opět z úhlu natočení malého motoru zjištěn směr ultrazvukového senzoru a oba parametry jsou předány proceduře `markGhost(int number, int directHelp)` objektu `desk`.

### 3.9.6 motorsForward(), motorsStop(), oneStep()

Tyto tři procedury řídí motory. První procedura nejdříve nastaví rychlost obou motorů na 600 stupňů/sec, a poté spustí oba motory směrem dopředu. Obě tyto činnosti jsou v synchronizovaném bloku, což znamená, že se provedou najednou. Pokud by nebyly synchronizovány, jeden motor by se rozjel dříve než druhý a robot by zatočil. Procedura `motorsStop()` naopak oba motory synchronizovaně zastaví. Poslední procedura oba motory synchronizovaně otočí o 600 stupňů (což je přibližně velikost jednoho políčka), a to rychlostí 600 stupňů/sec, a poté se motory zastaví.

### 3.9.7 rotation*()

Tyto tři procedury (`rotationLeft()`, `rotationRight()` a `rotation180()`) mají shodný cíl, a to změnit směr pohybu robota. Jak už jejich název napovídá, procedura `rotationLeft()` zatočí o 90 stupňů doleva, `rotationRight()` o 90 stupňů doprava a `rotation180()` robota otočí o 180 stupňů. Na počátku každé procedury je restartován gyroskopický senzor. Poté jsou spuštěny motory proti sobě (jeden dopředu a druhý dozadu) rychlostí 400 stupňů/sec. Po otočení o úhel x, který měří gyroskopický senzor, jsou motory vypnuty. Jelikož se motory nezastaví ihned, ale trvá malý okamžik než se dotočí, a tím pádem robot ještě o něco zatočí, je úhel x menší než 90 stupňů.

Pokud by se stalo, že se robot nemůže volně otáčet, ale například při otáčení drhne o zeď (a tím pádem se motory vlivem odporové síly zastaví dříve a robot se neotočí o celých 90 stupňů), změří gyroskopický senzor, po pauze 120ms, úhel otočení znovu. Pokud je menší než požadovaný, opět se spustí motory proti sobě, dokud není dosaženo požadovaného úhlu otočení.

### 3.9.8 reset()

Pokud by robot narazil do ducha, je možné stisknutím dotykového senzoru aktivovat právě tuto proceduru. Ta má za úkol vypnout motory a zavolat proceduru `reset()` objektu `desk`, která vrátí robota v souřadnicovém systému na výchozí pozici. Nakonec tato procedura zavolá `goToLabyrinth()`, kde robot opět čeká na stisknutí dotykového senzoru, než se rozjede do bludiště.

## 3.10 Třída Desk

Třída `desk` má na starosti vše okolo souřadnicového systému, včetně uchovávání aktuální polohy a směru robota, která je určena pomocí tří proměnných: `row`, `column` a `direct`. Udržuje dvourozměrné pole `desk[6][9]`, které je virtuálním ekvivalentem fyzického bludiště. Pomocí hodnoty každého z políček udržuje aktuální informace o stavu LED diod a rozmístění překážek, či duchů. Obsahuje také funkce, které předávají informace o stavu bludiště okolním objektům.

### 3.10.1 move(), back()

`Move()` je základní procedura, která se stará o pohyb robota v souřadnicovém systému. Kdykoliv je zavolána a robot se právě nenachází před okrajem bludiště, změní polohu robota dle aktuálního směru o jedno políčko dopředu.

Procedura `back()` je volána v případě, kdy robot narazí do políčka před sebou. Označí tedy dané políčko jako překážku a vrátí robota v souřadnicovém systému dle daného směru o jedno pole zpět. Funguje však pouze pokud robot není na okraji bludiště, a to z důvodu, že procedura `move()` na okraji neposouvá robota a mohlo by tedy dojít k rozhození robota v souřadnicovém systému.

### 3.10.2 control*(),

Všechny funkce, které začínají slovem `control`, mají podobný účel, a to vrátit 1, pokud je splněna kontrolovaná vlastnost, jinak 0. Následuje tabulka, kde je přehled všech `control*()` funkcí, včetně kontrolovaných vlastností:

### 3.10.3 measure*()

Všechny procedury začínající slovem `measure` mají podobný úkol, a to mapovat překážky v okolí jednoho políčka od robota. Procedura `measureForward()` je jediná z trojice, která před zakreslením překážky do mapy neověřuje její přítomnost. Je volána v případě, kdy robot narazí do překážky (a je tedy zřejmé, že zde překážka opravdu je) nebo při označování ducha. Pokud se tak stane, procedura přidělí danému políčku hodnotu 2 (překážka).

Procedury `measureLeft()` a `measureRight()` pracují na stejném principu, pouze na opačnou stranu. Pokud je naměřená vzdálenost menší než 25 cm a příslušné políčko nebylo již navštíveno, je tomuto políčku přiřazena taktéž hodnota 2 (překážka). Pokud je však naměřená vzdálenost větší než 25 cm a dané políčko je označeno jako překážka, přidělí mu procedura hodnotu 3 (zatím nenavštíveno). Toto opatření reaguje na jezdící duchy, které robot při prvním kontaktu vyhodnotí jako překážku. Při druhém průjezdu okolo daného políčka však robot ducha již nedetekuje a překážku v podobě ducha z mapy vymaže.

| funkce | políčko | kontrolovaně vlastnost |
|---|---|---|
| controlForward() | před robotem | zhasnutá LED dioda<br>překážka |
| controlGhost() | před robotem | přítomnost ducha |
| controlGhostRight() | vpravo | přítomnost ducha |
| controlEdge() | před robotem | okraj bludiště |
| controlPresenceRight() | vpravo | zhasnutá LED dioda<br>překážka |
| controlPresenceRightPresence() | vpravo | zhasnutá LED dioda |
| controlPresenceLeftPresence() | vlevo | zhasnutá LED dioda |
| controlRightEdge() | vpravo | okraj bludiště |
| controlPresenceAround() | před robotem | zhasnutá LED dioda<br>překážka |
| | za robotem | zhasnutá LED dioda<br>překážka |
| | vlevo | zhasnutá LED dioda<br>překážka |
| | vpravo | zhasnutá LED dioda<br>překážka |

**Tabulka 3.1.** Přehled control*() funkcí

### 3.10.4 rotation*()

Tyto tři procedury (`rotationL()`, `rotationR()` a `rotationB()`) mají všechny stejnou funkci, a to změnu směru robota v souřadnicovém systému. V případě `rotationL()` (zajišťuje otočení doleva v souřadnicovém systému) se od proměnné `direct` odečítá 1. U `rotationR()` se 1 přičítá a u `rotationB()` (rotace o 180 stupňů) se proměnné `direct` přiřadí hodnota opačného směru, než aktuálního.

### 3.10.5 findWay()

Tato funkce má na starosti nalezení nejkratší cesty k ještě nenavštívenému nebo neprozkoumanému políčku pomocí objektu `bfs`. Nejprve zavolá proceduru `inicialize (int[][] desk, int row, int column, int direct)` objektu `bfs`, které předá jako parametr aktuální stav bludiště a aktuální polohu robota včetně jeho směru. Poté zavolá funkci `getWay()` objektu `bfs`, která vrátí nalezenou cestu. Návratová hodnota této funkce je velikost cesty (počet políček, která robot cestou navštíví).

### 3.10.6 rotationToWay(), getRotation()

První funkce má jako návratovou hodnotu změnu směru, kterou musí robot na aktuálním políčku vykonat, aby dorazil k dalšímu políčku cesty. Nejdříve se určí pomocný směr, a to porovnáváním aktuálních souřadnic robota a souřadnic dalšího políčka cesty. Poté se tento pomocný směr odečte od aktuálního. Nakonec se odstraní první políčko cesty.

Funkce `getRotation()` pracuje na podobném principu. Jelikož je však první políčko cesty již odstraněno a robot se nachází stále na výchozím políčku, je nutné z návratové hodnoty předchozí funkce dopočítat souřadnice prvního políčka cesty. Poté se hledá směr jízdy na další políčko, podobně jako v předchozí funkci. Tato funkce neodstraňuje žádné políčko cesty.

### 3.10.7 markGhost(int number, int directHelp)

Parametry této procedury jsou číslo vyjadřující kolikáté políčko od robota se nachází duch (případně překážka) a směr natočení ultrazvukového senzoru. Z aktuálního směru robota a směru natočení ultrazvukového senzoru je spočítán směr, jakým se má mapovat duch (překážka).

Pokud na daném políčku (podezřelému z výskytu ducha/překážky) nebyla zhasnuta LED dioda, ani se zde nenachází překážka, je jeho aktuální hodnota zkopírována do proměnné `deskHelp` a poté je mu přiřazena hodnota 4 (políčko podezřelé, že se na něm vyskytuje duch/překážka). V případě, že se na nějakém políčku mezi robotem a nově označeným políčkem s duchem/překážkou nachází políčko s hodnotou 2 nebo 4, tedy tu byl chybně detekován duch/překážka, je hodnota tohoto políčka vrácena na hodnotu původní, a to pomocí zavolání procedury `resetField(row, column)`.

### 3.10.8 resetField(int row, int column)

Tato jednoduchá procedura přiřadí políčku o souřadnicích, daných jako parametr při zavolání procedury, hodnotu, jakou mělo před označením tohoto políčka jako políčka s podezřením na výskyt ducha (překážky). Přiřadí políčku v souřadnicovém systému `desk[][]` hodnotu políčka v pomocném souřadnicovém systému `deskHelp[][]`, který se aktualizoval před označením daného políčka indexem 4.

## 3.11 Třída BFS

Třída BFS má na starosti vyhledávání nejbližšího neprozkoumaného políčka. Využívá k tomu prohledávání do šířky (BFS), díky čemuž nalezne vždy tu nejkratší cestu (pokud existuje). Jednotlivá políčka jsou v této třídě reprezentována pomocí jednoho čísla, které vznikne zakódováním z proměnných `row` a `column` dle vztahu: `(row + 1) * 10 + column`.

### 3.11.1 expand(int hash), isTarget(int field)

Funkce `expand(int hash)` vrací `LinkedList` všech sousedních políček daného políčka, které nemají hodnotu 2 nebo 4 (nejsou překážka nebo duch). Pro každý směr ověří, zda příslušné políčko splňuje dané požadavky a pokud ano, přidá jeho zakódovanou hodnotu do `LinkedListu exp`, který je návratovou hodnotou. Políčka jsou seřazena v následujícím pořadí:

1 před robotem
2 za robotem
3 nalevo od robota
4 napravo od robota

Funkce `isTarget(int field)` typu `boolean` vrací `true`, pokud má políčko předané v parametru funkce hodnotu 3, tedy pokud ještě nebylo navštíveno ani zmapováno.

### 3.11.2 findWay()

Tato procedura obsahuje algoritmus prohledávání do šířky (BFS). Nejdříve se zavolá funkce `expand(int hash)`, s parametrem aktuálního políčka, která vrátí `LinkedList` všech sousedních polí, po kterých by se mohl robot pohybovat. Tento `LinkedList` se přidá do seznamu `open`. Následně se pomocí hash mapy přiřadí všem políčkům z `open` jejich rodič, a to políčko, které se původně expandovalo.

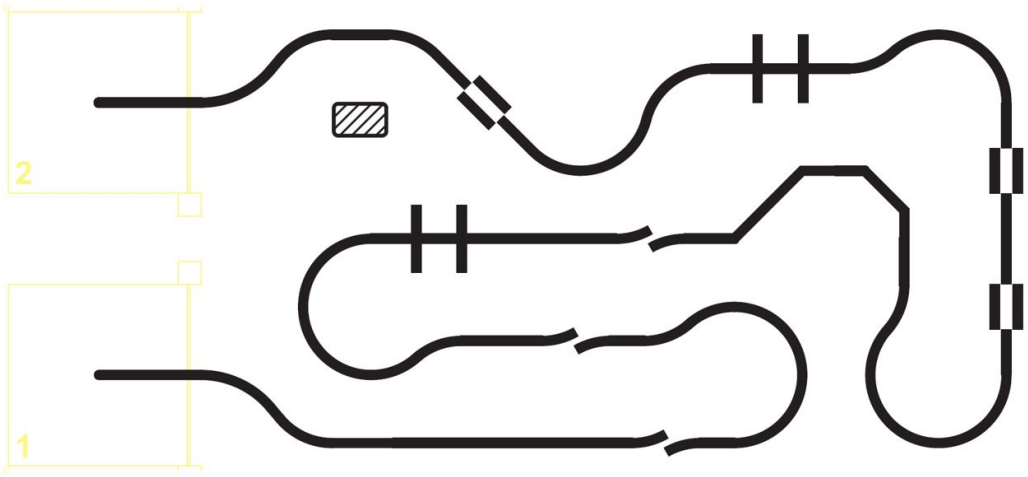
Následuje while cyklus, který probíhá, dokud není seznam open prázdný. Na začátku cyklu se vezme první políčko ze seznamu `open`, odstraní se z `open` a přidá

se do seznamu `close`. Pokud je toto políčko cíl cesty, což se ověří pomocí funkce `isTarget(int field)`, označí se jako cíl a cyklus předčasně končí. V případě, že není toto políčko cíl cesty, expanduje se pomocí funkce `expand(int hash)` a všechna expandovaná políčka, co nejsou v seznamu `open` ani `close`, se přidají do seznamu `open` a je jim pomocí hash mapy přiřazen rodič.

Nakonec se přidá cesta do `LinkedListu way`. Jako první se do ní přidá políčko cíle, načež se pomocí hash mapy nejde jeho rodič. Stejným způsobem se pokračuje dále, dokud se nedojde k aktuálnímu políčku, na kterém stojí robot.

# Kapitola 4
# Sledování černé čáry

V této části práce je úkolem sestavit a naprogramovat robota, který v nejkratším možném čase projede principem sledování černé čáry vytyčenou trasu. Robot musí jet po černé čáře samostatně, bez jakékoliv další pomoci. Černá čára může mít proměnou šířku, může se různě rozdvojovat, a opět spojovat, případně se na ni mohou nacházet další překážky, které budou v této práci dále specifikovány. Příklad černé čáry je uveden na obrázku 4.1.




**Obrázek 4.1.** Příklad provedení černé čáry [16]

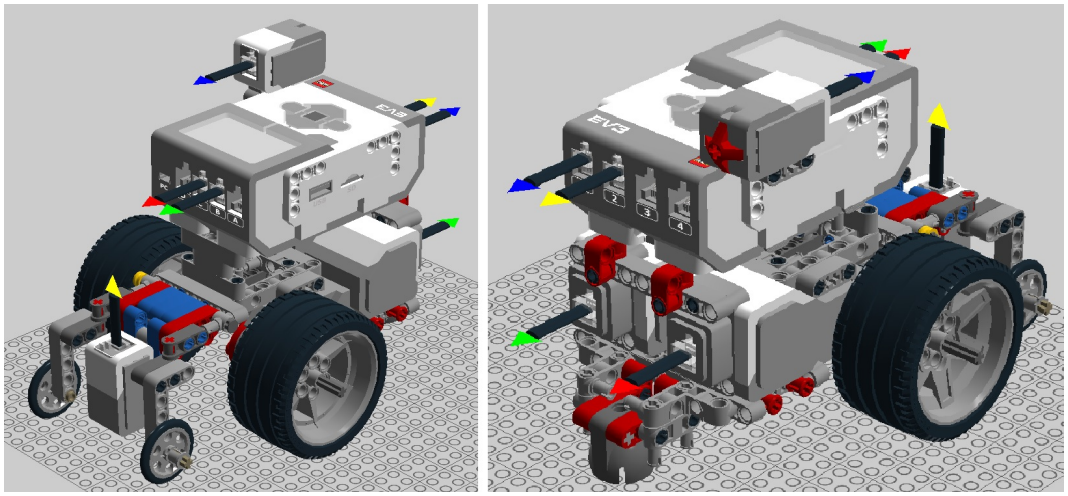
## 4.1 Konstrukce robota

Na konstrukci robota nejsou v tomto případě kladeny takové nároky, jako v předchozí úloze. V principu robot potřebuje pouze dva motory, přičemž každý pohání kolo (kola) na jedné straně, a barevný senzor, který detekuje černou čárou. Na obrázku 4.2. je zobrazený model robota v programu Lego Digital Designer.

Barevný senzor by měl být pro snadnější ovládání robota umístěn před osou obou motorů. Pokud by byl umístěn přímo na jejich ose, trvalo by déle, než se projeví aktuální úpravy chování robota a ten by byl hůře řiditelný. Pokud by však byl senzor umístěn příliš velkou vzdálenost před robotem, při nájezdu na překlápěcí houpačku by se senzor přiblížil k podložce, čímž by se změnila naměřená intenzita odraženého světla a robot by se rozkmital.

Aby se zabránilo rozkmitání robota při nájezdu na překlápěcí houpačku, a zároveň mohl být barevný senzor před robotem, je tento senzor umístěn na plošině, která je k robotovi připevněna tak, aby se mohla vůči němu volně otáčet nahoru a dolu. V dolní části plošiny jsou připevněna kolečka, která jsou umístěna těsně nad zemí. Ta se v momentě, kdy robot najíždí nebo sjíždí z překlápěcí houpačky, začnou odvalovat po podložce,

díky čemuž se barevný senzor nepřiblíží k podložce více než při jízdě po rovině. V dolní poloze plošiny jsou nastaveny dorazy, aby nemohla klesnout více, než je žádoucí. Při jízdě po rovné podložce je plošina vlivem gravitace a tlaku přívodního kablíku k senzoru tlačena na dorazy v dolní poloze.



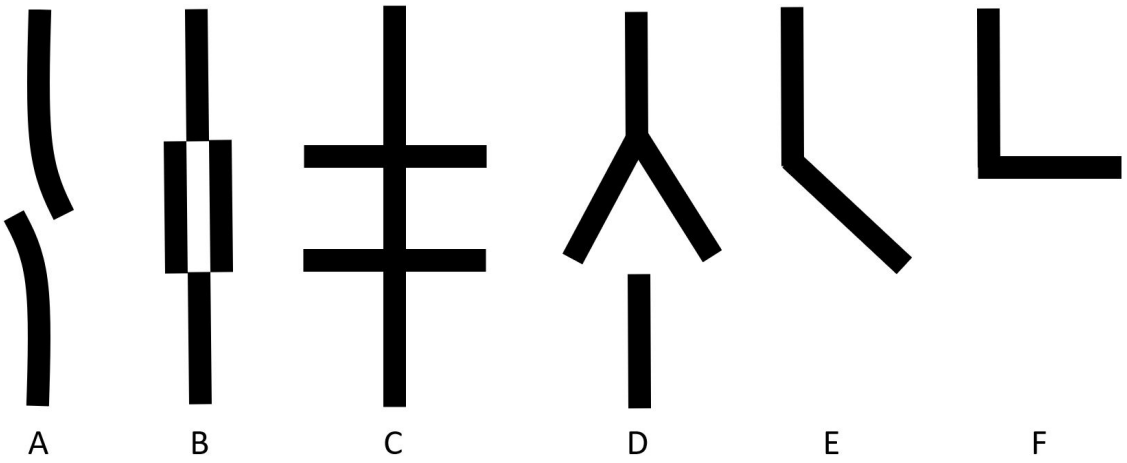
**Obrázek 4.2.** Pohled na robota v programu Lego Digital Designer

## 4.2 Překážky na trati

Na černé čáře, se může vyskytovat několik různých překážek. Jejich přehled je znázorněn na obrázku 4.3. Zde je uveden jejich stručný popis:

A černá čára je zde rozpojena a oba konce jsou vychýleny do strany
B barva čáry se na krátkém úseku změní z černé na bílou
C přes černou čáru jsou přidány dvě příčné černé čáry
D černá čára je zde přerušena, přičemž robot může pokračovat podél jedné ze dvou přilehlých černých čar
E lomená černá čára s úhlem 45 stupňů
F lomená černá čára s úhlem 90 stupňů

Dále se na trati mohou nacházet dvě speciální překážky, a to překlápěcí houpačka a tunel. Překlápěcí houpačka je v podstatě plošina o rozměrech 50 x 25 centimetrů, která je uprostřed připevněna k otáčející se ose přibližně 4.5 centimetrů nad zemí. Když robot najede na plošinu a přejede přes osu otáčení (přes půlku), plošina se vlastní vahou a vahou robota překlopí na druhou stranu a robot může pokračovat v jízdě. Na plošině bude stejně jako na zbytku trati nakreslena černá čára, aby se robot mohl orientovat i při jejím přejezdu. Tunel bude mít tvar a vzhled klasického tunelu, který bude postaven nad černou čárou. Na obou překážkách bude umístěna rovná černá čára bez zatáček. Na obrázku 4.4 jsou dva různé pohledy na tunel. Na třetím obrázku 4.4. se pak nachází detail mechanismu otáčení překlápění houpačky.




**Obrázek 4.3.** Překážky na trati

## 4.3 Software robota

Po spuštění programu se nejprve motorům nastaví výchozí rychlost a poté se oba spustí směrem dopředu. Následně se spustí nekonečný cyklus, ve kterém se na základě aktuálně naměřené hodnoty barevného senzoru mění rychlost motorů. Pomocí PID regulátoru je z naměřené hodnoty barevným senzorem a dalších parametrů vypočten akční zásah. Ten se přičte k rychlosti jednoho motoru a od rychlosti druhého se naopak odečte. Díky tomu robot zatočí požadovaným směrem. Pro malé odchylky od černé čáry (například když se robot ustálí při jízdě po rovné čáře) jsou nastaveny menší konstanty P, I a D než pro odchylky větší (například při jízdě v zatáčkách).

Nejprve je nutné změřit si návratovou hodnotu barevného senzoru černé čáry a jejího okolí. Černá barva bude mít obecně nižší hodnotu, než barva bílá. Požadovaná hodnota, tedy návratová hodnota senzoru na rozhraní černé čáry a okolí se spočítá jako aritmetický průměr těchto dvou hodnot.

Z principu je možné s daným programem sledovat čáru pouze z jedné strany (robot se bude pohybovat po rozhraní levého okraje čáry a okolí, nebo po rozhraní pravého okraje a okolí). Například pokud by se pohyboval po jejím levém okraji a mírně se vychýlil z optimální trasy směrem doleva, akční zásah bude větší než 0, zvýší se rychlost levého motoru (pravého o stejnou hodnotu sníží) a robot zatočí mírně doprava, čímž se vrátí zpět na rozhraní černé čáry a okolí. Tím se změní návratová hodnota barevného senzoru a robot se opět ustálí.

**Obrázek 4.4.** Speciální překážky na trati: tunel a detail překlápěcí houpačky

## 4.4 PID regulátor

PID regulátor použitý pro tento systém je ve tvaru: $akčnízásah = K_p * odchylka + K_i * integrace + K_d * derivace$, přičemž $K_p, K_i$ a $K_d$ jsou konstanty, jejichž hodnota bude určena později. Odchylka, se kterou pracuje proporční (P) složka regulátoru, je rozdíl aktuální a požadované hodnoty. Tato odchylka, přenásobená konstantou Kp, přímo ovlivňuje velikost akčního zásahu a tím pádem i rychlost obou motorů. Robot řízený samotným P regulátorem se velmi dlouho ustaluje. Pokud je konstanta Kp dostatečně velká na to, aby byl robot schopný projet zatáčkou (což musí být), bude poté robot na přímém úseku kmitat ze strany na stranu.

Aby se zabránilo zdlouhavému kmitání, obsahuje regulátor derivační (D) složku. Ta se spočítá jako rozdíl aktuální a minulé odchylky přenásobená konstantou $K_d$. To znamená, že pokud se robot přibližuje k rozhraní černé čáry a okolí, zmenšuje se tím i odchylka, a derivační složka bude záporná (aktuální odchylka je menší, než minulá), čímž se sníží i akční zásah a robot nedosáhne takového překmitu, jako kdyby zde D složka nebyla.

Integrační (I) složka regulátoru má za úkol odstranit trvalé regulační odchylky. Malé odchylky znamenají i malý akční zásah a může být velmi zdlouhavé, než je regulátor odstraní. Integrační složka pracuje na principu sčítání odchylek (integrační - obsah plochy pod křivkou, pouze převedený do diskrétního světa), takže i malá odchylka se brzy projeví.

V této konkrétní úloze nebyla integrační složka regulátoru využita. Na rozhraní černé čáry a bílého okolí se mění intenzita odraženého světla velmi rychle a malá odchylka regulátoru je tedy téměř nepostřehnutelná, neboť robot stále sleduje rozhraní černé čáry a okolí. Navíc u některých překážek použitých v této úloze musí robot nutně jet nějakou dobu jen po bílé nebo černé (nikoliv rozhraní těchto dvou barev), což by mělo za následek rychlý nárůst integrační složky a následné rozkmitání robota.

V následující tabulce je uvedeno, jaký mají vliv hodnoty konstant na dobu náběhu, dobu ustálení a překmit. Vyšší konstanta se projeví na sledovaných hodnotách následovně:

| konstanta | doba náběhu | doba ustálení | překmit |
|---|---|---|---|
| $K_p$ | snížení | žádný | zvýšení |
| $K_i$ | snížení | zvýšení | zvýšení |
| $K_d$ | žádný | snížení | snížení |

**Tabulka 4.1.** Vliv konstant PID regulátoru

## 4.5 Nastavení konstant PID regulátoru

Přesné nastavení konstant PID regulátoru je složitá a zdlouhavá činnost. Existují různé analytické metody, které dané konstanty přesně vypočítají. Při jejich použití je však potřeba mít přesný popis systému, jehož vytvoření by v tomto případě bylo zhruba stejně náročné nebo i náročnější, než využít heuristické metody, které vrátí přibližné hodnoty konstant, a poté PID regulátor doladit experimentálně.

Jednou z heuristických metod je Ziegler-Nicholsova metoda. Její princip spočívá v nalezení tří parametrů systému (kritické zesílení $K_u$, perioda kmitů $T_u$ a doba běhu jednoho cyklu $dT$), ze kterých se následně dle daných vztahů vypočítají přibližné konstanty PID regulátoru. [17]

Kritické zesílení $K_u$ je definováno jako zesílení, při kterém se systém nachází na mezi stability. To znamená, že systém donekonečna osciluje a překmity se nezvětšují ani nezmenšují. V tomto konkrétním případě bude robot na mezi stability kmitat kolem černé čáry, přičemž kmity budou stále stejně velké. Pokud by se hodnota $K_u$ ještě zvýšila, robot by kmital stále více, až opustí černou čáru.

Perioda kmitů $T_u$ označuje dobu, za kterou proběhne jeden periodický děj. Jinými slovy se jedná o časový úsek, ve kterém se robot vychýlí na jednu i na druhou stranu a vrátí se zpět do výchozí polohy. Jelikož v tomto případě bude perioda kmitů řádově desetiny, maximálně jednotky sekund, je možné změřit trvání několika dějů a poté naměřený čas vydělit jejich počtem.

Doba běhu jednoho cyklu $dT$ vyjadřuje časový úsek, za který proběhne jeden cyklus řídícího programu, tedy barevný senzor změří aktuální barvu, vypočítá se akční zásah a upraví se rychlost obou motorů. Výsledná hodnota bude velmi malá, je proto možné pustit například 100 000 cyklů po sobě (pomocí for cyklu), změřit dobu jejich trvání a vydělit počtem cyklů.

Vztahy, pomocí kterých se dají ze změřených parametrů vypočítat konstanty PID regulátoru jsou uvedeny v tabulce 4.2.

| regulátor | $K_p$ | $K_i$ | $K_d$ |
|---|---|---|---|
| P | $0.5 * K_u$ | - | - |
| PI | $0.45 * K_u$ | $1.2 * K_p * dT/T_u$ | - |
| PD | $0.8 * K_u$ | - | $K_p * T_u/(8 * dT)$ |
| PID | $0.6 * K_u$ | $2 * K_p * dT/T_u$ | $K_p * T_u/(8 * dT)$ |
| Pessen Integral | $0.7 * K_u$ | $2.5 * K_p * dT/T_u$ | $3 * K_p * T_u/(20 * dT)$ |
| PID s překmity | $0.33 * K_u$ | $2 * K_p * dT/T_u$ | $K_p * T_u/(3 * dT)$ |
| PID bez překmitů | $0.2 * K_u$ | $2 * K_p * dT/T_u$ | $K_p * T_u/(3 * dT)$ |

**Tabulka 4.2.** Ziegler-Nicholsova metoda [17] [18]

## 4.6 Použití Ziegler-Nicholsovi metody

Následuje popis Ziegler-Nicholsovi metody:

1. Nejprve je potřeba dočasně vyřadit I a D složku PID regulátoru. Toho lze docílit například nastavením hodnoty konstant $K_i$ a $K_d$ na 0.
2. Nyní se postupně zvyšuje konstanta $K_p$, dokud se robot nedostane na mez stability, tedy dokud nebude kmitat a další zvýšení konstanty $K_p$ by robota již rozhodilo. Po nalezení takové konstanty $K_p$ se zadefinuje konstanta kritického zesílení $K_u = K_p$.
3. Pokud by robot i s kritickým zesílením nebyl schopný projet zatáčky (nedokázal by zatočit tak hodně, aby projel zatáčkou), je třeba snížit rychlost robota a opakovat bod 2.
4. Pro zjištění periody kmitů $T_u$ se nechá robot jet po rovném úseku černé čáry. V případě velmi malé periody je možné změřit celkový čas více dějů, přičemž se poté výsledný čas vydělí počtem dějů.
5. Doba běhu jednoho cyklu $dT$ se zjistí tak, že se nechá běžet několik cyklů (například 100 000), změří se jejich doba a poté se vydělí počtem cyklů.
6. Ze zjištěných parametrů $K_u$, $T_u$ a $dT$ se za pomocí vztahů z tabulky 4.2 vypočítají přibližné konstanty $K_p, K_i$ a $K_d$.

# Kapitola 5
# Tvorba internetových stránek s pravidly

Úkolem poslední části této práce je vytvořit webové stránky k navrhnutým úlohám. Každou úlohu je třeba představit, podrobně vysvětlit její pravidla, popsat použité pomůcky (jako například překážky v bludišti, houpačku u černé čáry,..) a nastínit způsob hodnocení, včetně určování pořadí jednotlivých týmů.

Internetové stránky `robosoutez.fel.cvut.cz` jsou vytvořené pomocí open sorce redakčního systému Drupal, což velmi zjednodušuje práci při jejich tvorbě. Díky tomuto systému odpadá práce s programovacími jazyky určenými pro tvorbu webových stránek. Stačilo tedy připravit prostý text, včetně obrázků, který byl poté vložen na internetové stránky robosoutěže.

## 5.1 Struktura stránky

Internetová stránka o dané úloze by měla mít přibližně tuto strukturu:

- Zadání úlohy - Stručný přehled o úloze
- Vybavení pro řešení úlohy - Přehled Lego setů, které lze použít
- Konstrukce robota - Přehled, z čeho všeho lze robota postavit
- Programování robota - Informace o softwarových možnostech
- Soutěžní plocha - Informace o fyzickém vybavení úlohy
- Pravidla úlohy - Podrobná pravidla dané úlohy
- Informace o překážkách - Informace o překážkách na trati
- Organizace soutěže - Způsob hodnocení, určování pořadí týmů
- Obecná ustanovení - Informace o možnostech odvolání, případně vyloučení

## 5.2 Pravidla soutěžních úloh

Obě aktuální soutěžní úlohy se v mírně odlišné (a jednodušší) podobě na robosoutěži již objevily (Pac-Man v roce 2014 v kategorii SŠ a v roce 2015 v kategorii ZŠ; sledování černé čáry v roce 2016 v kategorii ZŠ). Bylo tedy možné částečně čerpat z pravidel již proběhlých soutěží. Nicméně v aktuálním ročníku obsahují úlohy zcela nové aspekty, které bylo nutné popsat.

## 5.3 Hodnocení, určování pořadí

Soutěž probíhá ve třech kolech. První dvě kola mají za cíl vybrat 16 nejlepších týmů, které se utkají ve finále. U úlohy Pac-Man postupuje z prvního kola maximálně 24 týmů, které získali více než 20 bodů (v součtu, za dvě jízdy; počet bodů odpovídá počtu projetých polí = počtu zhasnutých světel). Z těchto 24 týmů prvních 8 automaticky postupuje do finále a dalších 16 se utká v druhém kole soutěže, při kterém se vybere zbývajících 8 týmů do finále.

U úlohy sledování černé čáry postupuje do druhého kola maximálně 48 týmů, které měly nejnižší součet (případně součin) časů dvou rozjížděk. V druhém kole, které je nezávislé na prvním, se dle stejného principu vybere 16 nejlepších týmů, které následně postoupí do finále.

Finále bude u obou úloh probíhat formou vyřazovacího systému na dvě porážky. Tento systém má oproti klasickému pavoukovi tu velikou výhodu, že má tým po jedné porážce možnost dále soutěžit v pavoukovi poražených. Vítěz v pavoukovi poražených se nakonec utká s vítězem klasického pavouka. Je tedy možné, aby celkově vyhrál tým i s jednou porážkou.

# Kapitola 6
# Závěr

Tato práce se zabývá přípravou dvou soutěžních úloh na robosoutěž. Teoretická část nejprve obsahuje stručný přehled historie Lego Mindstorms, včetně porovnání parametrů hardwaru tří nejvýznamnějších verzí Lego Mindstorms (RCX, NXT 2.0 a EV3). Tato část pokračuje podrobným představením nejnovější verze Lego Mindstorms EV3. Je zde popsána samotná kostka, jednotlivé motory a senzory. Teoretickou část uzavírá přehled programovacích jazyků i se základními ukázkami kódu, pomocí kterých se dá ovládat Lego Mindstorms EV3.

Druhá část práce je věnována úloze Pac-Man. Je zde stručný popis duchů s vysvětlením algoritmu jejich pohybu. Fotodokumentace duchů a jejich model v programu Lego Digital Designer je umístěn na přiloženém DVD. Následuje popis robota, vyjadřujícího samotného Pac-Mana. Po jeho představení po hardwarové stránce jsou zde vysvětleny interakce robota s duchy a jeho pohyb po bludišti. Kapitolu uzavírá detailní popis programu robota. Videa, jak robot projíždí bludiště (s duchy i bez), jsou rovněž umístěna na přiloženém DVD.

V předposlední kapitole, která se zabývá robotem sledujícím černou čáru s překážkami, je nejprve představena dráha s podrobným popisem překážek. Následuje popis robota po hardwarové i softwarové stránce a několik možností, jak doladit konstanty PID regulátoru.

V poslední kapitole, věnované tvorbě internetových stránek je v krátkosti popsaná technologie tvorby stránek a základní struktura stránky. Následuje popis vytváření pravidel a hodnocení soutěže.

Při programování soutěžních robotů se objevovaly, hlavně v oblasti přesnosti, konstrukční nedostatky stavebnice Lego Mindstorms EV3. Každá ze součástek má danou určitou přesnost, přičemž celková přesnost při použití více součástek narůstá. Tato nepřesnost dále narůstá s vyšší rychlostí. Pro splnění soutěžních úloh bylo tedy nutné najít kompromis mezi co nejvyšší rychlostí a přijatelnou přesností robota, aby byl schopný dojet do cíle.

# Literatura


[1] History of LEGO Robotics. LEGO [online]. [cit. 2018-7-20]. Dostupné z: https://www.lego.com/cs-cz/mindstorms/history.

[2] What is FIRST LEGO League?. FIRST LEGO League [online]. [cit. 2018-7-20]. Dostupné z: http://www.firstlegoleague.org/about-fll.

[3] CAPRANI, Ole. RCX Manual. LEGO Lab, University of Aarhus [online]. Nordre Ringgade 1, 8000 Aarhus C, Dánsko [cit. 2018-7-21]. Dostupné z: http://www.legolab.daimi.au.dk/CSaEA/RCX/Manual.dir/RCXManual.html.

[4] Lego Mindstorms NXT User Guide. LEGO Group, 2006 [cit. 2018-07-20]. Dostupné z: https://www.lego.com/en-us/service/buildinginstructions/search?initialsearch=8547 #?text=8547.

[5] Lego Mindstorms EV3 User Guide. LEGO GROUP., 2013 [cit. 2018-07-20]. Dostupné z: https://education.lego.com/en-us/support/mindstorms-ev3/user-guides.

[6] What are NXT BRICKS?. In: NXTcbgerard [online]. [cit. 2018-7-20]. Dostupné z: http://nxtcbgerard.blogspot.com/p/what-are.html.

[7] NAUČ SE PROGRAMOVAT - JE TO SNADNÉ. Lego [online]. [cit. 2018-7-21]. Dostupné z: https://www.lego.com/cs-cz/mindstorms/learn-to-program.

[8] Podpora aplikace EV3 Programmer. Lego [online]. [cit. 2018-7-21]. Dostupné z: https://www.lego.com/cs-cz/mindstorms/apps/ev3-programmer-app/support.

[9] NXC: Introduction. Bricx Command Center 3.3 [online]. [cit. 2018-7-31]. Dostupné z: http://bricxcc.sourceforge.net/nbc/nxcdoc/nxcapi/intro.html.

[10] VANĚK, Jakub. NXC4EV3. Robosoutěž ČVUT [online]. [cit. 2018-7-31]. Dostupné z: https://robosoutez.fel.cvut.cz/nxc4ev3.

[11] Overview. ROBOTC: a C Programming Language for Robotics [online]. [cit. 2018-7-24]. Dostupné z: http://www.robotc.net/download/.

[12] LEGO MINDSTORMS EV3 Support from MATLAB: Acquire sensor data and control motors on your LEGO MINDSTORMS EV3 robot. MathWorks [online]. [cit. 2018-7-24]. Dostupné z: https://www.mathworks.com/hardware-support/lego-mindstorms-ev3-matlab.html.

[13] LEGO MINDSTORMS EV3 Support from Simulink: Run models on LEGO MINDSTORMS EV3. MathWorks [online]. [cit. 2018-7-24]. Dostupné z: https://www.mathworks.com/hardware-support/lego-mindstorms-ev3-simulink.html.

[14] LeJOS EV3: What is leJOS?. LeJOS [online]. [cit. 2018-7-23]. Dostupné z: http://www.lejos.org/ev3.php.

[15] Soutěžní plocha. In: A3B99RO ROBOTI [online]. [cit. 2018-8-13]. Dostupné z: http://rbs.felk.cvut.cz/index.php?sekce=robosoutezprozs&id=robosoutezprozs_1415_rules.

[16] URVÁLEK, Michal. Využití robota LEGO MINDSTORMS ve výuce - návrh soutěžních úloh. 2017. Bakalářská práce. České vysoké učení technické v Praze. Vedoucí práce Ing. Martin Hlinovský Ph.D..

[17] Ziegler-Nichols Tuning Rules for PID. Microstar Laboratories [online]. [cit. 2018-11-14]. Dostupné z: http://www.mstarlabs.com/control/znrule.html.

[18] ZIEGLER, J. G. a N. B. NICHOLS. Optimum Settings for Automatic Controllers. Journal of Dynamic Systems, Measurement, and Control. 1993, 115(2B). DOI: 10.1115/1.2899060. ISSN 00220434. Dostupné z: http://DynamicSystems.asmedigitalcollection.asme.org/article.aspx?articleid=1406231.

# Příloha A
# Použité zkratky, obsah přiloženého CD

## A.1 Zkratky

| | |
|---|---|
| FIRST | For Inspiration and Recognition of Science and Technology |
| ot. | otáčky (například motoru) |
| NXC | Not eXactly C |
| API | Application Programming Interface |
| leJOS | Lego Java OS |
| JDK | Java Development Kit |
| JRE | Java Runtime Environment |
| LED | Light-Emitting Diode |
| BFS | Breadth First Search |

## A.2 Obsah přiloženého DVD

- Zdrojové soubory robota projíždějícího bludiště
- Zdrojové soubory robota sledujícího černou čáru
- 3D model robota projíždějícího bludiště v Lego Digital Designeru
- 3D model robota sledujícího černou čáru v Lego Digital Designeru
- 3D model univerzálního ducha v Lego Digital Designeru
- 3D model ducha jezdícího po úsečce v Lego Digital Designeru
- 3D model ducha jezdícího kolem překážky v Lego Digital Designeru
- 4 videa robota projíždějícího bludiště bez duchů
- 5 videí robota projíždějícího bludiště s duchy
- 2 videa robota sledujícího černou čáru
- Fotografie univerzálního ducha
- Fotografie překlápěcí houpačky
- Fotografie tunelu
- Webová stránka se zadáním úlohy Pac-Man
- Webová stránka se zadáním úlohy Sledování černé čáry
- Zadání bakalářské práce